Analysing Neuro-Dynamic Programming Through Non-Convex Optimisation

Martin Gottwald

Technische Universität München
TUM School of Computation, Information and Technology

Analysing Neuro-Dynamic Programming Through Non-Convex Optimisation

Martin Gottwald

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitz: Prof. Dr. Sebastian Steinhorst

Prüfer der Dissertation:

1. Prof. Dr.-Ing. Klaus Diepold
2. Priv.-Doz. Dr. Hao Shen

Die Dissertation wurde am 25. September 2023 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 9. Februar 2024 angenommen.

Martin Gottwald. *Analysing Neuro-Dynamic Programming Through Non-Convex Optimisation.* v5. Technische Universität München, Munich, Germany, 2024.

Keywords: Actor-Critic, Critical Point Analysis, Gauss Newton Residual Gradient, Local Quadratic Convergence, Mean Squared Bellman Error, Neuro-Dynamic Programming.

Lehrstuhl für Datenverarbeitung, Technische Universität München, Arcisstraße 21, 80333 München, Germany, https://www.ce.cit.tum.de/ldv/.

Supported by Deutsche Forschungsgemeinschaft (DFG) through TUM International Graduate School of Science and Engineering (IGSSE), GSC 81.

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die diese Arbeit ermöglicht haben. Ohne die stetige Ermutigung und Unterstützung, die ich in den letzten Jahren von vielen Seiten erfahren habe, wäre dieses Dokument nicht entstanden.

An erster Stelle danke ich meinem Doktorvater, Prof. Klaus Diepold, für die Möglichkeit und den Freiraum am Lehrstuhl für Datenverarbeitung an selbst gewählten Themen zu forschen, und für die wertvollen Ratschläge, die er mir in dieser Zeit gegeben hat. Von besonderer Bedeutung war für mich auch, dass ich die verschiedenen Lehrveranstaltungen nach eigenem Ermessen gestalten und mit Inhalten füllen durfte.

Darüber hinaus möchte ich mich bei Dr. Hao Shen als meinem Mentor bedanken. Er gab mir immer wieder neue Impulse für Forschungsrichtungen und stand jederzeit für Diskussionen zur Verfügung. Zudem gab er mir die Möglichkeit, meine Promotion nahtlos in seiner Forschungsgruppe bei fortiss fortzusetzen.

Ein weiterer Dank gilt all meinen Kolleginnen und Kollegen am Lehrstuhl für die gemeinsame Zeit, sei es bei einem gemeinsamen Kaffee oder bei einer Runde Tischfußball. Zudem wäre ohne deren Einsatz für den Erhalt der Recheninfrastruktur am Lehrstuhl die Promotion deutlich schwieriger ausgefallen. Außerdem danke ich meinen Kollegen bei fortiss für ihre hilfreichen Kommentare in der Endphase dieser Arbeit.

Ein großer Dank geht an meine Familie. Ohne die von ihnen geschaffene Grundlage und ihre jahrelange Unterstützung wäre mein Studium und auch meine Promotion nicht zu schaffen gewesen.

Mein besonderer Dank gilt Susanne. Ohne ihre bedingungslose Unterstützung in allen Höhen und Tiefen der Dissertation wäre diese nicht möglich gewesen.

Abstract

Recent development of Dynamic Programming with Value Function Approximation has demonstrated superior performance of Neural Networks in solving challenging problems with large or even continuous state spaces. One specific approach is to deploy Neural Networks to approximate value functions by minimising the Mean Squared Bellman Error function. Despite great successes of Deep Reinforcement Learning as a subfield of Dynamic Programming, development of reliable and efficient numerical algorithms to minimise the Bellman Error is still of great scientific interest and practical demand for applications in sequential decision making. Challenges arise due to the underlying optimisation problem being highly non-convex and not fully understood.

The first group of major contributions in this work resides in Policy Evaluation or the training of a Critic. Namely, I analyse the Mean Squared Bellman Error from a smooth optimisation perspective and develop an efficient Gauss Newton algorithm. First, I conduct a critical point analysis of the error function and provide technical insights on optimisation and design choices for Neural Networks. When the existence of global minima is assumed and the objective fulfils certain conditions, suboptimal local minima can be avoided when using over-parametrised Neural Networks. Second, I construct a Gauss Newton Residual Gradient algorithm based on the analysis in two variations. The first variation applies to discrete state spaces and exact learning. I numerically confirm theoretical properties of this algorithm such as local quadratic convergence to a global minimum. The second variation employs a sampling based approximation of integrals in the Mean Squared Bellman Error and can be used in the continuous setting. New issues for the objective arise from the required sampling step and I adapt the critical point analysis accordingly. I demonstrate feasibility and generalisation capabilities of the proposed algorithm empirically using continuous control problems and provide a numerical verification of my analysis. I outline the difficulties of Semi-Gradient approaches and how a Residual Gradient formulation can help. As part of the experiments, I find that multistep Bellman Operators cannot improve the training once second-order optimisation is employed. Lastly, with a Policy Iteration experiment, I test the general applicability of the algorithm and describe a conflict between the desired over-parametrisation setting and a necessary under-parametrisation.

The second group of main contributions in my work addresses the Actor-Critic algorithm class. I extend my critical point analysis from the Critic towards the objective used to train an Actor. As the first result, I require reward signals to avoid constant regions to prevent a frequent loss of information during Actor training. Second, by relying as before on over-parametrised networks and several design choices during the training of an Actor, I can establish pleasing properties for the optimisation problem. Yet, due to the non-convex nature of the Critic, it is no longer possible to realise a Gauss Newton algorithm. A third contribution addresses an open issue in the expected input domain of the Critic and the employed image space for the Actor. Since they do not coincide, it is possible that the Actor cannot maximise the Critic and undefined behaviour is possible. I propose a solution, which is based on formulating the Actor as Supervised Regression task and restricting its image set to the unit ball. As part of the experiments with Actor-Critic algorithms, I find both positive and negative results regarding their performance.

***Keywords*:** Actor-Critic · Critical Point Analysis · Gauss Newton Residual Gradient · Local Quadratic Convergence · Mean Squared Bellman Error · Neuro-Dynamic Programming

The machine is only a tool after all, which can help humanity progress faster by taking some of the burdens of calculations and interpretations off its back. The task of the human brain remains what it has always been; that of discovering new data to be analyzed, and of devising new concepts to be tested.

– Isaac Asimov, *I, Robot* (1950)

Contents

Chapter 1

Introduction

1.1 Status Quo in Deep Reinforcement Learning

Dynamic Programming (DP) is a general purpose framework for solving discrete sequential decision making tasks and infinite horizon continuous optimal control problems [Bellman, 1957]. It contains as a subfield Reinforcement Learning (RL), which has gained large popularity over the last decades. Furthermore, Dynamic Programming may even serve as an alternative approach to automatic control. Exemplary applications include the control of magnetic confinements for fusion reactors [Degrave et al., 2022], microgrids [Adibi and Van der Woude, 2022], quadrotors [Gronauer et al., 2022] or robots in general, e.g., the work in [Levine et al., 2016]. The control of multi-agent systems through DP is explored in [Ramaswamy et al., 2021].

To handle systems with large or even continuous state spaces, DP methods cannot be applied directly and approximations become necessary [Granzotto et al., 2021, Li et al., 2021, Rawlik et al., 2012]. The related field is called Approximate Dynamic Programming (ADP) and studies Value Function Approximation (VFA) as an important and effective approximation instrument [Bertsekas, 2012, Sutton and Barto, 2020].

These VFA methods come in two groups, namely *linear* and *non-linear*. Various efficient Linear Value Function Approximation (LVFA) algorithms have been developed and analysed in the field, e.g., [Nedić and Bertsekas, 2003, Bertsekas et al., 2004, Parr et al., 2008, Geist and Pietquin, 2013, Tsitsiklis and van Roy, 1997]. Despite their significant simplicity and convergence stability, the performance of LVFA methods depends heavily on construction of features and their linear combination, which is a time consuming and hardly scalable process in general [Parr et al., 2007, Böhmer et al., 2013]. Therefore, recent research efforts have focused more on Non-Linear Value Function Approximation (NL-VFA) methods.

As a popular non-linear mechanism, kernel tricks have been successfully adopted to VFA. They have demonstrated their convincing performance in various applications [Xu et al., 2007, Taylor and Parr, 2009, Bhat et al., 2012]. Unfortunately, due to the nature of kernel learning, these algorithms can easily suffer from a high computational burden due to the required number of samples. Furthermore, kernel-based VFA algorithms can also have serious problems with over-fitting. As an alternative, Neural Networks (NN) have been another common and powerful approach to approximate value functions [Lin, 1993, Bertsekas and Tsitsiklis, 1996]. The subfield of DP dealing with NN as approximation architectures is called Neuro-Dynamic Programming (NDP) and is also known as Deep Reinforcement Learning (DRL). Impressive successes of NNs in solving challenging problems in pattern recognition, computer vision, speech recognition and game playing [LeCun et al., 2015, Yu and Deng, 2015, Mnih et al., 2015, Silver et al., 2017] have further triggered increasing efforts in applying NNs to VFA [van Hasselt et al., 2016]. More specifically,

NN-based Value Function Approximation (NN-VFA) approaches have demonstrated their superior performance in many challenging domains. For example, *Deep Q-Networks* (DQN) allow for successful playing of *Atari* games [Mnih et al., 2015] or the game *Go* [Silver et al., 2016, 2017]. *Deep Deterministic Policy Gradient* (DDPG) [Lillicrap et al., 2015] extends these capabilities towards the continuous control regime. Despite these advances, development of more efficient NN-VFA based algorithms is still of great demand for tackling even more challenging applications. So far, these impressive successes are generally only possible, if a plethora of training samples and computational resources are available.

Next to estimating or approximating the value function, the representation and training of policies is also of great interest. A common approach is the family of Actor-Critic algorithms [Barto et al., 1983], because they realise a procedure similar to Policy Iteration and allow for continuous state and action space problems. A Critic, which is constructed around obtaining a NN-VFA, is responsible for evaluating a current behaviour or policy and, thus, corresponds to Policy Evaluation. An Actor executes actions and improves its performance by querying the Critic for the quality of its output. It resembles Policy Improvement. In the remaining document, the terms Critic and Policy Evaluation are used interchangeably. Similarly, an Actor and Policy Improvement can be exchanged.

Aside from some early work in [Baird III, 1995], called Residual Gradient (RG) algorithms, omitting gradients of the TD-target has been a common practice, see, for example, the work in [Riedmiller, 2005], and is more recently referred to as Semi-Gradient (SG) algorithms [Sutton and Barto, 2020]. This means, one tries to reduce the Mean Squared Bellman Error (MSBE) by using gradient information, but the gradient of the function approximation architecture at successor states is ignored. Semi-Gradient algorithms posses a high convergence speed in contrast to their Residual Gradient counterparts, but unfortunately they may exhibit undefined behaviour and their divergence is well-known. For example, Henderson et al. [2018] or Islam et al. [2017] have outlined strong changes in the outcome of an algorithm with seemingly identical implementations or a severe sensitivity on hyper parameters. Reasons for choosing Semi-Gradients over Residual Gradients include inferior learning speed of RG methods [Baird III, 1995], limitation with non-Markovian feature space [Sutton et al., 2008] and non-differentiable operators such as the max-operator involved in Q-learning. Residual Gradients are favoured for their convergence guarantees and applicability of "classic" gradient based optimisation techniques.

Existing realisations of Actor-Critic algorithms mostly rely on a Semi-Gradient formulation to minimise the loss in the Critic part. To circumvent the aforementioned issues, different heuristics and stabilisations have been proposed. *Target Networks* [Mnih et al., 2013] decouple the evaluation of a network at current and successor states. This helps with preventing divergence by providing more stable target values for training NNs. *Double Q-Learning* [van Hasselt et al., 2016] further improves training by removing harmful overestimations of Q-values.

Also for the Actor part, many stabilising techniques have been introduced. Among the most effective ones are *Proximal Policy Optimisation* (PPO) [Schulman et al., 2017] or *Trust Region Policy Optimisation* (TRPO) methods [Schulman et al., 2015]. Restrictions on the update distance for policies are employed in various forms with the goal to keep policies on track. After improving or updating, the Actor performance is still ensured by preventing drastic changes. Also, approaches like *Knowledge Regularisation* [Gottwald et al., 2017] fall in this category. To remove challenges in Actor training altogether, authors also omit training of an explicit Actor and recover optimal actions for all states directly

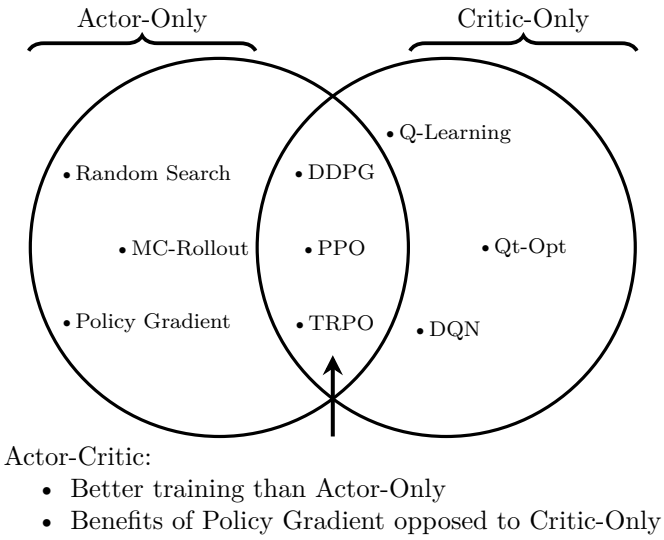


Figure 1.1: The intersection of Critic-Only and Actor-Only methods forms the class of Actor-Critic algorithms. For all three areas, a selection of representative algorithms is mentioned.

from a Q-function. They work with Critic-Only algorithms. For example, Kalashnikov et al. [2018] introduces *QT-Opt*, where the *Cross Entropy Method* searches in the input of a Q-function for optimal actions. The opposite approach to Critic-Only approaches is the set of Actor-Only methods. By employing Actor-Only algorithms, one works directly in policy space and tries to discover a well performing function without relying on a Critic. Figure 1.1 provides an overview over the full Actor-Critic landscape.

A point is reached, where one should question, why classic optimisation is not used as the common approach to Deep Reinforcement Learning. A most prominent example is the existence of the *Deadly Triad* [Sutton and Barto, 2020], which denotes divergence and stability issues for Semi-Gradient algorithms. At the same time, this triad seems to affect only Semi-Gradient algorithms but not Residual Gradient approaches, which would suggest to use that formulation instead. However, despite the historical reasons for preferring Semi-Gradient algorithms, there are also reported cases, where a Residual Gradient method reliably converges to a poor solution, thus discouraging its use. Therefore, a natural hypothesis is whether or not this behaviour of Residual Gradient algorithms only arises, because required assumptions for guarantees from the Dynamic Programming methodology are no longer satisfied and, as a result, whether Residual Gradient algorithms are merely not applied correctly any more.

A directly related challenge is the selection of a proper reward signal, which allows for actual progress during the training phase. This is still an art on its own, where various authors have investigated different strategies. For example, in [Schaul et al., 2016], *Prioritized Experience Replay* (PER) is designed to improve training by including transition data with different importance values into the objective. Events, which would have naturally a low chance for occurring, appear more often and boost the learning progress. *Hindsight Experience Replay* (HER) [Andrychowicz et al., 2017] takes this approach even further. State space transitions, which would be considered as a failure for not achieving the correct

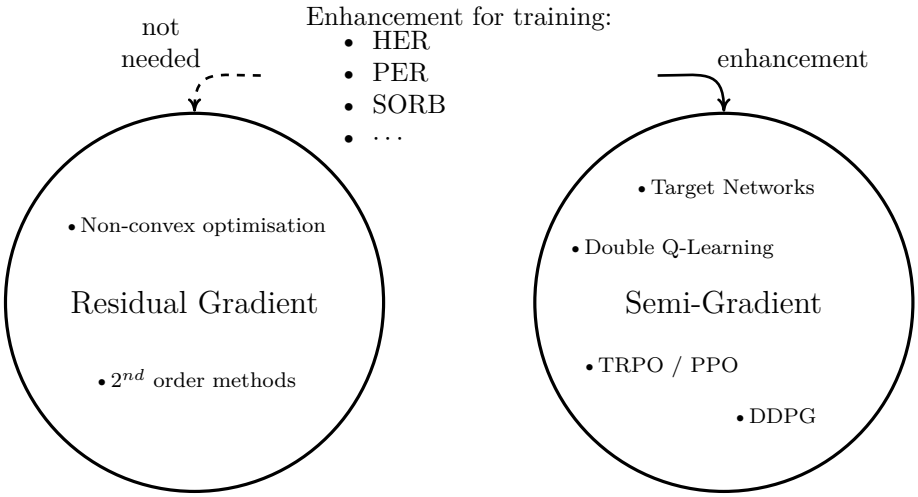


Figure 1.2: The influence of methods for improved training on Residual Gradient and Semi-Gradient formulations as algorithm classes. For both, a selection of affected algorithms and methods are mentioned. From the Dynamic Programming perspective, proposed enhancements for training should not provide a benefit for RG algorithms due to theoretical guarantees for convergence.

task, are reinterpreted to serve as a successful realisation of an alternative task. Thereby, a proper improvement of some policy is maintained. In [Eysenbach et al., 2019], the combination of conventional graph search and planning with RL methods is proposed and called *Search on the Replay Buffer* (SORB). The aim is to overcome problems for RL methods when information from sparse reward signals does not propagate sufficiently well over long horizons. Alternatively, employing massively parallel data collection and parameter updates as explored in [Mnih et al., 2016] can overcome challenges with the reward signal. Figure 1.2 provides a visual summary of the relations between enhancements for training, when Residual Gradient or Semi-Gradient formulations are employed as the main algorithmic classes for a Critic.

Opposed to these methods when employed together with Semi-Gradient formulations, one typically does not face challenges for training, which arise due to a particular choice of reward signal, if one is working with Dynamic Programming methods. The corresponding algorithms achieve their task and improve iteratively a given policy. The only uncertainty in choosing a reward signal is whether the chosen function will lead to the desired policy. But learning itself is working, which indicates again that one should ground Deep Reinforcement Learning algorithms in the foundation of Dynamic Programming. Hence, when assuming that a Residual Gradient formulation and its combination with non-convex optimisation results in a proper realisation of Dynamic Programming, then these enhancements should not provide any beneficial effect.

Instead of proposing various techniques for repairing Semi-Gradient algorithms, which actually only shift the underlying problem, it is necessary to tackle the actual design issue in Deep Reinforcement Learning algorithms. Consequently, to construct working algorithms, one should go back to the mathematical foundation of Reinforcement Learning, namely the (Approximate) Dynamic Programming theory, and rely on non-convex optimisation as framework for training. This is the goal for my thesis, which involves on the one hand

setting up Bellman Operators correctly and seeking for their unique fixed point. On the other hand, one has to formulate with the help of Residual Gradient algorithms a sound objective, which can be used together with non-convex optimisation. This may even involve relying on analytic models of the environment. Furthermore, the role of Non-Linear Value Function Approximation in continuous state spaces should be made explicit.

Once the Dynamic Programming framework is working as it is known from the past, one can use its theoretical results to make proper statements about Policy Improvement and the convergence of algorithms, even if Non-Linear Value Function Approximation is involved. Hence, a foundation based on non-convex optimisation will then be a starting point for future engineering efforts, where its benefits become visible during the design of algorithms. The construction of an algorithm will be easier, because the expected outcome is known in advance such that one can tell whether a correct result is produced or not. Also, their verification becomes more feasible because a training phase with reliable convergence reduces the need for trial-and-error.

The rest of this chapter is built around two main topics, which cover the aforementioned subjects in greater detail. First, I give in Section 1.2 a complete motivation for my work and outline, why a sophisticated analysis of the Mean Squared Bellman Error with concepts from non-convex optimisation is required once Non-Linear Value Function Approximation is present in Dynamic Programming methods. Second, in Section 1.3, I summarise the main contributions of my dissertation.

1.2 The Demand for Realising Dynamic Programming with Function Approximation through Non-Convex Optimisation

It is natural to wonder why one should approach Dynamic Programming when using Non-Linear Value Function Approximation via non-convex optimisation, despite working DRL algorithms such as PPO being already available, and why an in-depth analysis is even required. Due to a plethora of literature about Deep Reinforcement Learning and sophisticated textbooks about Approximate Dynamic Programming, it is already known that both DRL and ADP provide a well-performing methodology with impressive results. However, over the last two decades, the struggling of Deep Reinforcement Learning methods also has become well-known and the major effort in Approximate Dynamic Programming regarding a theoretical understanding of algorithms has been spent in linear function approximation architectures. But once non-linear architectures such as Neural Networks are introduced, there exist an implicit complexity and subtle algorithmic design choices, which are easy to overlook. In the following, I will address several existing issues, in no particular order, and provide a justification why one should tackle these issues via non-convex optimisation.

Reproducibility Many existing DRL algorithms have significant problems with their reproducibility. There are even papers addressing this issue as their core contribution, for example, the work in [Henderson et al., 2018, Islam et al., 2017, Gronauer and Gottwald, 2021]. There is a certain common agreement that algorithms such as *Proximal Policy Optimisation* [Schulman et al., 2017] or *Deep Deterministic Policy Gradients* [Lillicrap et al., 2015] are notoriously hard to reproduce. This may partially arise from an imprecise or even incomplete description of methods in papers, which leads to unwanted degrees

of freedom in implementations. Also, vague formulations of tasks for agents allow for misunderstanding. On top of that, most algorithms possess a severe dependency on hyper parameters. Without concise descriptions of tuning efforts it is almost impossible to tell whether a DRL algorithm shows its correct performance on a certain task, or whether more training is required.

I propose to focus on Residual Gradient algorithms, which can benefit from a proper analysis of critical points and non-convex optimisation in general. Semi-Gradient algorithms, as they are described in [Sutton and Barto, 2020], omit parts of the gradient information and would make a different analysis necessary. For RG approaches, it is possible to create design principles for the function approximation architecture, which help with formulating a sound algorithm without requiring additional tricks. Thereby, one avoids a hard to describe search space for hyper parameters.

An analysis of Residual Gradient algorithms also allows to handle explicitly continuous state space problems. Unfortunately, it happens that papers with a theoretical approach, for example, the work in [Brandfonbrener and Bruna, 2020, Holzleitner et al., 2021, Bojun, 2020, Sutton et al., 1999], only employ the Bellman Equation in matrix form or with discrete Markov Chains. Hence, one has to rely on some form of discretisation for continuous spaces to arrive in the realm of those papers, but any regular grid of a space quickly explodes in storage requirements once the dimensionality increases. Also, any irregular subdivision produces for higher dimensions extremely large cell sizes in the representation, which render the approach unusable in practice. Consequently, these theoretical results are out of reach from the engineering perspective and, thus, prevent any verification within a computer system and real world examples. Making use of the reverse direction of those findings, namely transferring statements about a transition probability matrix to continuous dynamical systems, is far from being trivial [Lind and Marcus, 1995]. From the algorithmic side and from the implementation's viewpoint, one has to provide an analysis, which takes continuous spaces directly into account such that an efficient Deep Reinforcement Learning system can be developed. In this work, I make the role of continuous spaces and the required sampling explicit and provide a Residual Gradient approach and its analysis in both continuous state and action space problems.

Strong Requirements Even if the underlying algorithm is not the source of problems, because, for example, an implementation is publicly available, it happens that DRL algorithms are used without ensuring the theoretical requirements. This may be due to the fact that such requirements are either ignored or simply not reported in sufficient detail and are therefore unknown in general. A well-suited example is the *Deterministic Policy Gradient* algorithm [Silver et al., 2014], which provides the foundation for others when using Neural Networks to approximate value functions. In particular, the reward function is subject to regularity requirements, which sometimes are not present in applications, see, for example, the work in [Matheron et al., 2020]. Furthermore, the reward signal must provide proper gradient information to be used with Actor-Critic approaches.

To arrive at a concise formulation of an optimisation task, a complete investigation of all moving components is mandatory and the goal of my thesis. Next to the question, whether a sparse or dense reward is going to be used, one also has to incorporate the effect of its derivatives. Beside the proper analysis of critical points and the construction of design principles for the Critic and Actor, the reward signal and thus the entire objective,

which appears in non-convex optimisation problems, needs to satisfy certain conditions. These conditions stem not only from a theoretical perspective, for example, decaying learning rates according to the Robbins-Monro condition, but also from an engineering and implementation perspective.

Data Consumption & Quality Real world applications do not provide data in the amount that existing Deep Reinforcement Learning algorithms demand [Moerland et al., 2020]. To provide a feeling for the numbers, consider the amount of data consumed by the Reinforcement Learning algorithm *AlphaGo* [Silver et al., 2017] for learning to play the game of Go. To produce a policy with super human performance, *AlphaGo* has used 4.9 million games of self plays, where each play consists of a plethora of state space transitions.

Furthermore, current state of the art RL algorithms operate model-free and rely on exploration mechanisms to collect the required data. Whereas it is trivial to formulate exploration based on noisy actions or random walks on paper, in practice it might be impossible to reach parts of the state space, where the reward signal is present. Depending on the environment at hand, collecting all transitions can become easily a challenge on its own. As an example, consider an autonomous car with the goal to reach a certain pose. If the current pose is close enough to the desired one, a positive reward is given. When starting from an arbitrary initial state, executing random steering and acceleration commands certainly will not bring the car to its target pose. Since no reward is received, learning is impossible and an initial policy is not changed.

Even worse, in real world applications, exploration by executing arbitrary actions is also too dangerous. See, for example, the concerns described by García and Fernández [2015]. There exists a form of agreement that RL should work with (available) models, e.g., in [Berkenkamp et al., 2017], or exploit learned models, e.g., in [Cowen-Rivers et al., 2022], such that an arbitrary number of any possible state space transition can be queried without restriction.

The need for working model-based also arises from the choice of non-convex optimisation as tool for minimising the Mean Squared Bellman Error. It is covered in more detail later in the document. When writing down the MSBE as objective for minimisation, one sees that either ergodic processes are needed or that states must be sampled freely from the entire state space. However, loosing the ergodicity happens easily once policies improve. For example, in the well-studied pole balancing control problem, an optimal policy prevents the pole from falling over. Thus, states with high angular velocities of the pole or even large angles themselves no longer show up and the ergodicity is lost. In such environments, invoking a model is the only option to create by hand transitions in the state space, which can occur in arbitrary regions.

Since one cannot guarantee that the dynamical system under control is ergodic in general and because relying on exploration is also too risky for engineering applications, I conclude that working model-based is a more or less mandatory requirement to enable the application of any RL or DP algorithm. A positive consequence is that once the necessity for a model is present, the distinction between On- and Off-Policy algorithms is no longer required. The typical definition as given, for example, in [Sutton and Barto, 2020] is:

> "On-policy methods attempt to evaluate or improve the policy that is used to make decisions, whereas off-policy methods evaluate or improve a policy different from that used to generate the data."

However, this definition does not apply to a model-based algorithm, where state transitions are created arbitrarily. States are set freely in the state space and not according some current policy (Off-Policy learning), but one still tries to evaluate the current policy and its action produced for the successor state (On-Policy setting). This means, my analysis and the approach based on Residual Gradient algorithms includes implicitly both settings.

A last aspect related to the availability of data is the *Double Sampling* issue. This issue arises from the necessity to collect per state several realisations of its successors to handle expectations correctly [Baird III, 1995]. In real world implementations of RL or DP algorithms, it is impossible to collect transition data that satisfies this requirement. However, if one already has to use models for other reasons, then solving the *Double Sampling* issue is no longer required. The information about successor states, including their entire probability distribution, is either part of the model or must be easy to obtain from an accurate simulator. Hence, Residual Gradient algorithms can be implemented without relying on additional constructions and benefit directly from my analysis.

Reliability, Runtime and Computational Concerns The bad environmental impact of artificial intelligence, or more precisely Deep Learning and the related computer programs, is no longer a secret [Strubell et al., 2019, Dodge et al., 2022]. Even dedicated effort has been spent on capturing the exact impact [Henderson et al., 2020]. Here, not only the aforementioned amount of hyper parameter tuning, which is required until one knows the values that result in a working algorithm, is a problem, but also the training itself for a certain (possibly optimal) set of hyper parameters is inefficient. One has to spend ever larger models with increasing training times for the learning algorithm with diminishing returns [Thompson et al., 2021].

My analysis might provide a beneficial effect on these concerns. Due to insights from the critical points and the derived design principles, it could become possible to employ second-order optimisation with approximated Hessian information. Thereby, one would remove the need for excessive tuning of learning rates and tweaking the parameters such as the momentum of a descend method drops away. Also, frequent restarts of the entire training process are redundant, because bad local minima or saddles would have to be suppressed such that second-order methods can be applied. On top of that, less computation time is necessary due to a better descent behaviour of such methods. Even if individual steps are more expensive, the overall time is smaller, since far less steps are enough. Additionally, the outcomes of optimisation typically have a better quality.

Policy Gradient Methods and Random Policy Search Pure Policy Gradient methods, which are an example of Actor-Only algorithms, appear as promising alternative with impressive results on robotic manipulators, e.g., [Levine et al., 2016]. The same applies to pure Random Search methods in policy space, e.g., [Mania et al., 2018] and references therein. Despite Mania et al. [2018] only considering linear functions of the state as policy class, they are still able to demonstrate competitive results on typical RL benchmarks.

However, disadvantages of Policy Gradient formulations or Random Search methods also exist. The biggest ones are twofold. First, these algorithms are tailored towards finite horizon problems. Second, no statements about the type and nature of a policy are possible. The first disadvantage results from the lack of general results regarding convergence of such methods in the more general infinite horizon discounted reward setting [Zhang et al., 2019].

Relying on finite horizon tasks or shortest path formulations is the best one can do. The second disadvantage arises from the missing access to a value- or Q-function. In particular, for the infinite horizon setting, the only way to draw conclusions about the nature of a given policy is to make use of the necessary and sufficient condition for optimality (cf. Equation (2.10)).

Hence, I propose to use still the Dynamic Programming methodology within an infinite horizon setting and to focus on Actor-Critic architectures or, more precisely, on the Approximate Policy Iteration framework. This allows for statements about optimality of a policy and provides a foundation with known behaviour of algorithms. To do so, a sophisticated analysis is required such that the Dynamic Programming framework can be applied, especially if non-linear approximation architectures for both the value function and policy are employed.

1.3 Contribution

In this work, I study the Dynamic Programming methodology when working with Non-Linear Value Function Approximation from a pure non-convex optimisation viewpoint. More precisely, I work in the framework of geometric optimisation [Absil et al., 2008], although I stay for the major part of my work in Euclidean vector spaces. It is important to understand and overcome the challenges involved in both Policy Evaluation and Policy Improvement. Naturally, these results carry over to Deep Reinforcement Learning in the form of Actor-Critic algorithms by characterising the learning behaviour of Policy Evaluation (a.k.a Critic) and Policy Improvement (a.k.a Actor).

I seek not only for a better understanding of Deep Reinforcement Learning, but also try to clarify mysteries or unanswered questions. For example, the application of non-convex optimisation is not that straightforward and will require precise conditions such that a descent algorithm is guaranteed to work. Also, the effectivenesses of DRL algorithms on continuous problems needs a careful investigation such that limitations of Neural Networks in an Actor-Critic approach become visible. Even if no universal remedy would be available, a concise characterisation of all conditions and requirements is still valuable. If one has a guarantee of the form “if some conditions are violated, then training of Neural Networks must fail”, then it would be possible to avoid unnecessary computation beforehand. My main contribution consists of two building blocks, which reflect the underlying structure of an (Approximate) Policy Iteration algorithm, and therefore covers also the Actor-Critic algorithm class. Firstly, I address the problem of minimising the Mean Squared Bellman Error, i.e., Policy Evaluation or the training of a Critic. Secondly, I focus on the maximisation of the Critic’s assessment of a policy, which corresponds the Policy Improvement step or the training of an Actor.

For the Policy Evaluation part, I conduct a critical point analysis of the MSBE when combined with Neural Networks. Additionally, I also address its Hessian and derive a proper approximation for it. I work with both discrete and continuous state spaces. With discrete states, exact learning is possible and leads to strong theoretical statements. In continuous spaces, one needs to use sampling to make the MSBE accessible. This weakens theoretical findings for discrete spaces and also gives raise to new complications. For both types of state spaces, I obtain insights in the learning process and can prevent the existence of saddle points or undesired local minima by requiring over-parametrisation of

the NN-VFA architecture and by ensuring certain properties of the optimisation objective. Furthermore, my analysis leads to an efficient and effective Approximated Newton (AN), or, more precisely, to a Gauss Newton (GN) algorithm. As an extension, I study the effect of multistep Bellman Operators and their impact on critical point conditions. Afterwards, I investigate the continuous state space setting from a numerical perspective. I test, whether or not ignoring the dependency of derivatives on the network parameters in the TD-target has a significant impact, especially in the context of second-order optimisation. Next, I outline how to overcome the convergence speed issues of Residual Gradient methods and show that gradients and higher-order derivatives of the TD-target provide critical information about the optimisation problem. They are essential for implementing efficient optimisation algorithms and result in solutions with higher quality. In particular, for a Critic-Only Policy Iteration algorithm, this property is a core requirement. As part of the experiments, I find that multistep Bellman Operators do not improve significantly the training if second-order optimisation is already employed. Finally, I conduct several experiments to confirm the results of my critical point analysis, namely the roll of over-parametrisation for MLPs, numerically and also investigate generalisation capabilities of NN-VFA methods empirically.

To investigate the optimisation problem related to Policy Improvement, I start with the construction of a sound situation, in which optimisation and training of NNs in both components of a Policy Iteration procedure work reliably. I demonstrate subtleties in formulating such a proper optimisation task and characterise sources of challenges, which affect negatively the training of an Actor. A first source is the Critic and its MLP used to approximate a Q-function. The gradients of the MLP with respect to action inputs are required to provide a clear improvement direction for the policy parameters. Additionally, the location and type of critical points need to be consistent with the training goal for an Actor. I describe how to design MLPs for achieving proper gradients. A second source of challenges is also the reward signal. Its shape translates directly into the shape of the Q-function, thus the reward function needs to be designed with its usage for non-convex optimisation in mind. I investigate the requirements and report the necessary properties one has to ensure. A third source of challenges is residing in the MLP of the Actor, which is used as parametrised policy. Similarly to the Critic, over-parametrisation affects positively the optimisation task. Yet, a new problem resides in the required MLP structure to map into the available action space. First, I capture this problem empirically and report its extent. Second, I propose a possible solution by reformulating training of Actors and exploiting geometry of the unit ball. Throughout my work regarding Actors, I construct various experiments with the goal to demonstrate severity of these subtleties and to grasp certain aspects also quantitatively.

My thesis takes the following structure. Chapter 2 describes Neuro-Dynamic Programming in a nutshell and introduces all notations or concepts around Non-Linear Value Function Approximation as they are required for later chapters. In Chapter 3, I present my critical point analysis of the Mean Squared Bellman Error and report my findings for training Non-Linear Value Function Approximation architectures to represent value functions. Afterwards, in Chapter 4, I transfer those findings from the approximation of value functions into the policy domain. I make the interplay between the two function approximation architectures explicit and also account for the impact of the reward signal on optimisation. Lastly, I conclude my work in Chapter 5.

Chapter 2

Neuro-Dynamic Programming in a Nutshell

2.1 Introduction

The purpose of this chapter is the introduction of my notation and the clarification of technical preliminaries such that a concise foundation for Chapters 3 and 4 is available. This includes, for example, a summary of Dynamic Programming itself and the notation behind Neural Networks, but also the specification of benchmarks. First, I outline the Dynamic Programming setting in general. I introduce the well-known Markov Decision Process as modelling language and define its components. Furthermore, I state the overall goal of DP, namely the iterative improvement of policies such that they approach an optimal one, and describe tools for achieving this goal. Second, I provide the definition of Multi-Layer Perceptrons, which will be the NL-VFA method I am going to analyse. A precise notation, which covers all the inner mechanisms of this function class, is mandatory to allow for a critical point analysis in later chapters. Third, I give a description of non-convex optimisation in general and address how existing approaches in the Reinforcement Learning domain interact with non-convex optimisation. Eliminating all types of ambiguities regarding optimisation strategies is indispensable for my work. Fourth, I describe how non-linear function approximation architectures can be used with Dynamic Programming and formulate the optimisation problem I want to investigate in a later chapter. A concise basis for this topic is imperative, as it affects all the rest of my work. Fifth, I introduce several benchmarks, which I will use to investigate and verify my theoretical findings also quantitatively. Each benchmark consists of the definition of a dynamical system and the corresponding reward signal to specify a certain task. The benchmarks will appear in dedicated sections with sophisticated experiment and as small intuitive illustrations throughout my theoretical analysis.

The rest of this chapter is arranged as follows. The introduction of Dynamic Programming itself is contained in Section 2.2. Section 2.3 then establishes the necessary notational conventions around Multi-Layer Perceptrons. Afterwards, in Section 2.4, I elaborate on the connection between existing RL algorithms and the field of non-convex optimisation. Next, Section 2.5 defines the objective I use for approximating value functions. Lastly, in Section 2.6, I describe the benchmarks used for numerical statements during the entire document.

2.2 Dynamic Programming

As the common approach, I model the sequential decision making task or the continuous control problem as a Markov Decision Process (MDP) by defining the tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$. For the state space $\mathcal{S}$, I consider both finite countable sets of $K_{\mathcal{S}}$ discrete elements

$\mathcal{S} = \{1, 2, \ldots, K_{\mathcal{S}}\}$ as well as compact subsets of finite dimensional Euclidean vector spaces $\mathcal{S} \subset \mathbb{R}^{K_{\mathcal{S}}}$. Depending on the state space, I denote with slight abuse of notation by $K_{\mathcal{S}} \coloneqq |\mathcal{S}|$ either the cardinality of a finite set or by $K_{\mathcal{S}} \coloneqq \dim(\mathcal{S})$ the dimension of a vector space. The meaning of $\mathcal{S}$ and $K_{\mathcal{S}}$ will be clear from the context. The action space $\mathcal{A}$ follows the same pattern. I consider environments with $K_{\mathcal{A}}$ discrete actions one can choose from, i.e., $\mathcal{A} = \{1, 2, \ldots, K_{\mathcal{A}}\}$. And there are environments, which provide also a compact subset of a vector space such that the action space takes the form $\mathcal{A} \subset \mathbb{R}^{K_{\mathcal{A}}}$. Again, by slight abuse of notation, I set $K_{\mathcal{A}} \coloneqq |\mathcal{A}|$ or $K_{\mathcal{A}} \coloneqq \dim(\mathcal{A})$ depending on the nature of the action space. The conditional transition probabilities $P\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ are either available directly in the form of analytic models or can be collected by using simulators and performing rollouts. For discrete spaces, they define a discrete probability distribution $P(s'|s, a)$ for transiting from state s to s' when executing action a. In continuous state action spaces, the term P takes the role of a probability density function and must be used with integrals. A scalar reward function $r\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [-M, M]$ with $M \in \mathbb{R}$ assigns an immediate and finite one-step reward to the transition triplet (s, a, s'). Finally, $\gamma \in (0, 1)$ represents a discount factor, which is required to ensure convergence of the overall expected discounted reward.

The goal in DP is to learn a stationary policy π, which maximises the expected discounted reward. It is sufficient to consider deterministic policies of the form $\pi\colon \mathcal{S} \to \mathcal{A}$, as the space of history independent and deterministic policies can be proven to contain an optimal policy. Derivations and proofs for this statement and others in the current section are available in chapter one and two of [Bertsekas, 2012]. Furthermore, stochastic policies of the form $\pi\colon \mathcal{S} \times \mathcal{A} \to [0, 1]$, which would be used to sample an action, do not fit into the geometric optimisation approach of my work.

The expected discounted reward starting in some state $s \in \mathcal{S}$ and following the policy π afterwards is called value function and is defined as

$$V_\pi\colon \mathcal{S} \to \mathbb{R}, \quad s \mapsto \lim_{T\to\infty} \mathop{\mathbb{E}}_{s_1, s_2, \ldots, s_T} \left[\sum_{t=0}^{T} \gamma^t r(s_t, \pi(s_t), s_{t+1}) \middle| s_0 = s \right]. \tag{2.1}$$

Taking the maximum over all allowed policies results in the optimal value function

$$V^*\colon \mathcal{S} \to \mathbb{R}, \quad s \mapsto \max_{\pi \in \Pi} V_\pi(s), \tag{2.2}$$

where Π denotes the space of all deterministic Markov policies. For a given policy π, the value function V_π satisfies a recursive relationship, which is known as the Bellman Equation. One can write for all states $s \in \mathcal{S}$

$$V_\pi(s) = \mathop{\mathbb{E}}_{w \sim \mathcal{W}(s, \pi(s))} \Big[r\big(s, \pi(s), f(s, \pi(s), w)\big) + \gamma V_\pi\big(f(s, \pi(s), w)\big) \Big] \quad \forall s \in \mathcal{S}, \tag{2.3}$$

where the entire stochasticity of the transitions from some state s with action a is encoded in the additional disturbance variable[1] $w \in \mathcal{W}(s, a)$. The disturbances w are required to be finite and countable to be used as weighted summation, or must provide a suitable

[1] Unfortunately, if the bold print is ignored, the symbol $\mathcal{W}$ is used for both the spaces of all disturbances and the space of parameters in a Multi-Layer Perceptron. Additionally, the system dynamics and an approximated function will share the same symbol f. To stick to the notation in existing literature and because there is typically no confusion arising for the reader, I keep the clash of notation. The meaning of a term is defined uniquely by the context and its arguments.

probability density function to be used in integrals. In both cases, the system equation $f\colon \mathcal{S} \times \mathcal{A} \times \mathcal{W} \to \mathcal{S}$ in the Bellman Equation is fully deterministic. Similarly to V_π, the optimal value function V^* belongs for all states $s \in \mathcal{S}$ to the Optimal Bellman Equation

$$V^*(s) = \max_{a \in \mathcal{A}} \mathop{\mathbb{E}}_{w \sim \mathcal{W}(s,a)} \Big[r\big(s, a, f(s,a,w)\big) + \gamma V^*\big(f(s,a,w)\big) \Big] \quad \forall s \in \mathcal{S}. \tag{2.4}$$

Although the disturbance w can be used equally well for discrete and continuous state spaces, there exists for discrete states the convenience to treat the successor state s' directly as the random variable. Hence, one is able to combine the operations for all states in one equation. First, one replaces the expectation over w with a weighted sum over all s'

$$V_\pi(s) = \sum_{s'} P(s, \pi(s), s') \Big(r(s, \pi(s), s') + \gamma V_\pi(s') \Big) \quad \forall s \in \mathcal{S}. \tag{2.5}$$

Second, one collects all operations for every $s \in \mathcal{S}$ as vectors or matrices to yield the Bellman Equation in matrix form

$$V_\pi = R_\pi + \gamma P_\pi V_\pi. \tag{2.6}$$

The matrix P_π contains all possible values of $P(s, \pi(s), s')$. The vector R_π is the result of collecting all $r(s, \pi(s), s')$ terms after they have been combined via the expectation. The related computation takes the form $R_\pi = (P_\pi \odot \tilde{R}_\pi) \cdot \mathbb{1}$, where $\tilde{R}_\pi$ stores all $r(s, \pi(s), s')$ as matrix similar to P_π. By multiplying their elementwise product with a vector of ones from the right, one obtains for all states the expected values. Of course, the matrix form of the Bellman Equation is only available if transition probabilities are known and can exist due to a discrete and finite state action space.

When treating $V_\pi \in \mathcal{V}$ as a variable, where $\mathcal{V}$ is the space of all possible value functions, the right hand side of Equation (2.3) induces the Bellman Operator under the policy π and is denoted by $\mathrm{T}_\pi\colon \mathcal{V} \to \mathcal{V}$. Analogously, the Optimal Bellman Operator $\mathrm{T_g}\colon \mathcal{V} \to \mathcal{V}$ is implied by Equation (2.4). It can be shown that both T_π and $\mathrm{T_g}$ are contraction mappings with modulus γ and, consequently, that the value function V_π and optimal value function V^* are their unique fixed points, respectively. Using these operators, one can introduce a shorthand notation and write compactly

$$V_\pi(s) = \left(\mathrm{T}_\pi\, V_\pi\right)(s) \quad \text{and} \quad V^*(s) = \left(\mathrm{T_g}\, V^*\right)(s) \tag{2.7}$$

for all states s in the state space. Due to their contractive nature, both operators allow for an iterative procedure to compute their respective fixed points. In Value Iteration (VI), one starts with an arbitrary initial value function $V \in \mathcal{V}$ and applies $\mathrm{T_g}$ infinite many times to arrive at the optimal value function

$$V^*(s) = \lim_{T \to \infty} \Big(\underbrace{(\mathrm{T_g} \circ \cdots \circ \mathrm{T_g})}_{T \text{ times}} V \Big)(s) \quad \forall s \in \mathcal{S}. \tag{2.8}$$

The Bellman Operator T_π appears in Policy Iteration (PI), which consists of two components. Policy Evaluation takes a form similar to Equation (2.8), but uses T_π to produce V_π when starting from some $V \in \mathcal{V}$

$$V_\pi(s) = \lim_{T \to \infty} \Big(\underbrace{(\mathrm{T}_\pi \circ \cdots \circ \mathrm{T}_\pi)}_{T \text{ times}} V \Big)(s) \quad \forall s \in \mathcal{S}. \tag{2.9}$$

Policy Improvement takes a current policy π together with its evaluation V_π and extracts a greedily induced policy (GIP) $\pi' \colon \mathcal{S} \to \mathcal{A}$ from both. This means one needs to solve

$$(\mathrm{T}_{\pi'} V_\pi)(s) = (\mathrm{T}_{\mathrm{g}} V_\pi)(s) \quad \forall s \in \mathcal{S} \tag{2.10}$$

for π'. The new GIP π' then satisfies $\pi' \geq \pi$ in the sense that $V_{\pi'}(s) \geq V_\pi(s)$ for all states in the state space with the equality being true for optimal policies. By alternating between Equations (2.9) and (2.10), one arrives again at V^* and also obtains a corresponding optimal policy π^*.

In Equation (2.10), there is a certain computational overhead involved in the derivation of an improved policy from a value function. Namely, to determine for a given state the optimal action, it is required to evaluate the system dynamics for all noisy transitions whenever a certain candidate for an optimal action is selected. This overhead can be avoided, if one introduces similar to Equation (2.1) a so-called Q-function. It is defined as

$$Q_\pi \colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}, \quad (s,a) \mapsto \lim_{T \to \infty} \mathop{\mathbb{E}}_{s_1, s_2, \ldots, s_T} \left[\sum_{t=0}^{T} \gamma^t r(s_t, \pi(s_t), s_{t+1}) \Big| s_0 = s, a_0 = a \right] \tag{2.11}$$

and has its corresponding Bellman Equation and Operator in Q

$$\begin{aligned} Q_\pi(s,a) &= \mathop{\mathbb{E}}_{w \sim \mathcal{W}(s,a)} \Big[r\big(s, a, f(s,a,w)\big) + \gamma Q_\pi\big(f(s,a,w), \pi(f(s,a,w))\big) \Big] \\ &= (\mathrm{T}_\pi Q_\pi)(s,a) \end{aligned} \tag{2.12}$$

for all $s, a \in \mathcal{S} \times \mathcal{A}$. The definition of the optimal Q-function and its corresponding Optimal Bellman Equation and Operator work as for state-only value functions. One obtains

$$Q^* \colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}, \quad (s,a) \mapsto \max_{\pi \in \Pi} Q_\pi(s,a) \tag{2.13}$$

and

$$\begin{aligned} Q^*(s,a) &= \mathop{\mathbb{E}}_{w \sim \mathcal{W}(s,a)} \Big[r\big(s, a, f(s,a,w)\big) + \gamma \max_{a' \in \mathcal{A}} Q^*\big(f(s,a,w), a'\big) \Big] \\ &= (\mathrm{T}_{\mathrm{g}} Q^*)(s,a) \quad \forall (s,a) \in \mathcal{S} \times \mathcal{A} \end{aligned} \tag{2.14}$$

The Bellman Operators T_π and T_{g} in Q possess identical contraction properties as when working with value functions alone. Hence, all algorithms work with Q-factors as they do for state-only value functions. This means it is possible to use the same symbol for the operators and accept again a slight abuse of notation. With Q-functions, a greedily induced policy can now be defined compactly as

$$\pi'(s) \in \mathop{\mathrm{argmax}}_{a \in \mathcal{A}} Q_\pi(s,a) \tag{2.15}$$

for all $s \in \mathcal{S}$. Most importantly, Equation (2.15) brings the advantage that one only needs to optimise a scalar function with any suitable method to determine an optimal action for a given state. Neither the evaluation of the dynamical system f nor the stochasticity are involved in this process and the computational burden is reduced.

With Q-functions and the idea behind Policy Iteration, it is finally possible to introduce an algorithm with practical relevance, namely Approximate Policy Iteration (API). This

algorithm makes the role of inexact realisations, which must occur in any computer system, explicit and allows for a feasible implementation. For API, the construction of the limit value in Equation (2.9) and the exact improvement for actions in all states in Equation (2.10) are replaced by two approximate formulations

$$\|Q - Q_\pi\|_\infty \leq \epsilon_1 \quad \text{and} \quad \|\mathrm{T}_{\pi'} Q_\pi - \mathrm{T}_\mathrm{g} Q_\pi\|_\infty \leq \epsilon_2, \tag{2.16}$$

where $\|x\|_\infty = \max_{ij} |x_{ij}|$ is the maximum norm. The constants ϵ_1 and ϵ_2 are non-negative scalars. As with exact Policy Iteration, each round of Approximate Policy Evaluation and Approximate Policy Improvement refines the Q-function and policy π until the approximation errors ϵ_1 and ϵ_2 prevent any further progress. Of course, Equation (2.16) also exist in terms of V and would share the same behaviour.

In practice, the approximate formulations for Policy Evaluation and Improvement will be realised as optimisation problems, whose solutions are V_π or Q_π and the policy π. For the value- or Q-function, the optimisation problem can be constructed by defining the so-called Temporal Difference error. Given an arbitrary $V \in \mathcal{V}$, the difference of both sides in Equation (2.7) is denoted by

$$\delta(s) \coloneqq V(s) - (\mathrm{T}_\pi V)(s) \quad \forall s \in \mathcal{S}. \tag{2.17}$$

The term Temporal Difference (TD) arises from the occurrence of a current state in $V(s)$ and its successor state s' inside $(\mathrm{T}_\pi V)(s)$. In this context, one calls the application of a Bellman Operator also the TD target. Similarly, the Bellman Equation in Q allows as well to define the corresponding TD error in Q

$$\delta_Q(s,a) \coloneqq Q(s,a) - (\mathrm{T}_\pi Q)(s,a) \quad \forall (s,a) \in \mathcal{S} \times \mathcal{A}. \tag{2.18}$$

Both types of TD errors are non-zero for all but the correct value- or Q-function. Thus, one can use δ or δ_Q to convert the theoretical fixed point iteration from Equation (2.9) into an implementable root finding problem. To do so, I will use a non-linear function approximation architecture, namely a Neural Network, to represent the function space $\mathcal{V}$. Then, I will combine the squared TD-error for all states with the approximation architecture to obtain a performance objective that, when minimised, leads to an accurate approximation of the true value- or Q-function under policy π .

A concise introduction of all components, which are required for approximating the space $\mathcal{V}$, is the goal for Section 2.3. The objective used for training, i.e., the Mean Squared Bellman Error, is constructed in Section 2.5. In Chapter 3, I then conduct a critical point analysis of the MSBE to provide a complete characterisation of this objective and its set of solutions.

For improving a policy π, an optimisation problem is given directly by Equation (2.15). Whereas discrete action spaces allow easily for exact Policy Improvement, continuous ones render approximation architectures necessary here as well. Possible solution approaches to this optimisation task and their analysis are subject to Chapter 4.

2.3 Multi-Layer Perceptrons

To approximate value functions, I deploy in my work a classic feed-forward and fully connected Neural Network, a.k.a. Multi-Layer Perceptron (MLP). In the following, I

summarise the well-known definition of an MLP such that for the remaining document a concise notation exists with the goal to avoid any possible source of confusion.

Let me denote by L the number of layers in the MLP structure, and by n_l the number of processing units in the l-th layer with $l = 1, \ldots, L$. By $l = 0$, I refer to the input layer with n_0 units. The value of n_0 depends on the state space and its type. For value functions and discrete state spaces, I have $n_0 = 1$ such that a single state can be processed directly as natural number by the MLP. In a continuous setting, I use $n_0 = K_{\mathcal{S}}$ units matching the $K_{\mathcal{S}}$-dimensional state vectors. To represent a Q-function, the input layer is increased by one for discrete actions, i.e., $n_0 = K_{\mathcal{S}} + 1$, or by the dimension of the action space such that one has $n_0 = K_{\mathcal{S}} + K_{\mathcal{A}}$ units in the input layer. I always restrict the number of nodes in the output layer $l = L$ to $n_L = 1$.

Let $\sigma \colon \mathbb{R} \to \mathbb{R}$ be a unit activation function and denote by $\dot{\sigma} \colon \mathbb{R} \to \mathbb{R}$ its first derivative with respect to the input. The unit activation function σ and its derivatives act element-wise on non-scalar values. Depending on the concrete choice for σ, the domain and image might have to be changed. Traditionally, the activation function σ is chosen to be non-constant, bounded, continuous and monotonically increasing (e.g., the Sigmoid function). More recent popular choices consider unbounded functions like (Leaky-) ReLU, SoftPlus or the Bent-Identity[2]. In this work, I further restrict the choice for the activation function to smooth, unbounded and strictly monotonically increasing functions such as SoftPlus, Bent-Identity or also the Identity function itself, which is used frequently in the last layer. The latter two functions are used in this work. The reason for this additional restriction will become clear in Chapter 3.

For the (l, k)-th unit in an MLP architecture, i.e., the k-th unit in the l-th layer, I define the corresponding unit mapping $\Lambda_{l,k} \colon \mathbb{R}^{n_{l-1}} \times \mathbb{R} \times \mathbb{R}^{n_{l-1}} \to \mathbb{R}$ as

$$\Lambda_{l,k}(w_{l,k}, b_{l,k}, \phi_{l-1}) \coloneqq \sigma\left(w_{l,k}^{\mathsf{T}} \phi_{l-1} - b_{l,k}\right), \tag{2.19}$$

where $\phi_{l-1} \in \mathbb{R}^{n_{l-1}}$ denotes the output from layer $(l-1)$. The terms $w_{l,k} \in \mathbb{R}^{n_{l-1}}$ and $b_{l,k} \in \mathbb{R}$ are a parameter vector and a scalar bias associated with the (l, k)-th unit, respectively. Next, I can define the l-th layer mapping by stacking all unit mappings of layer l as

$$\begin{aligned} \Lambda_l(W_l, b_l, \phi_{l-1}) &\coloneqq \left[\begin{array}{ccc} \Lambda_{l,1}(w_{l,1}, b_{l,1}, \phi_{l-1}) & \ldots & \Lambda_{l,n_l}(w_{l,n_l}, b_{l,n_l}, \phi_{l-1}) \end{array}\right]^{\mathsf{T}} \\ &= \sigma\left(W_l^{\mathsf{T}} \phi_{l-1} + b_l\right), \end{aligned} \tag{2.20}$$

with $W_l \coloneqq [w_{l,1}, \ldots, w_{l,n_l}] \in \mathbb{R}^{n_{l-1} \times n_l}$ and $b_l \coloneqq [b_{l,1}, \cdots, b_{l,n_l}] \in \mathbb{R}^{n_l}$ being the l-th parameter matrix and bias vector, respectively. It is convenient to store the bias vector as an additional row of the matrix and extend the layer input with a constant value of 1. Thus, one can write Equation (2.20) equivalently as

$$\Lambda_l(W_l, \phi_{l-1}) \coloneqq \sigma\left(W_l^{\mathsf{T}} \cdot \left[\begin{array}{c} \phi_{l-1} \\ 1 \end{array}\right]\right) \tag{2.21}$$

by using a larger parameter matrix $W_l \in \mathbb{R}^{(n_{l-1}+1) \times n_l}$. Next, let me define the overall function represented by the MLP. First, I denote by $\phi_0 \in \mathbb{R}^K$ the network input. Then,

[2] also abbreviated as Bent-Id

the output at the l-th layer is defined recursively as $\phi_l := \Lambda_l(W_l, \phi_{l-1})$. Note that the last layer in an MLP can employ the identity map as activation function and thus may only be an affine mapping. Finally, by denoting the set of all parameter matrices in the MLP as $\boldsymbol{\mathcal{W}} := \mathbb{R}^{(n_0+1)\times n_1} \times \cdots \times \mathbb{R}^{(n_{L-1}+1)\times 1}$, I can compose all layer-wise mappings to define for a set of parameters $\mathbf{W} \in \boldsymbol{\mathcal{W}}$ the overall network mapping as

$$f\colon \boldsymbol{\mathcal{W}} \times \mathbb{R}^{n_0} \to \mathbb{R}, \quad (\mathbf{W}, \phi_0) \mapsto \Big(\Lambda_L(W_L, \cdot) \circ \cdots \circ \Lambda_1(W_1, \cdot)\Big)(\phi_0), \tag{2.22}$$

which contains in total N_{net} parameters with N_{net} being computed by

$$N_{net} = \sum_{l=1}^{L} (n_{l-1} + 1) n_l. \tag{2.23}$$

With this construction, I can define the set of parametrised functions, which belong to a given MLP architecture, by writing

$$\mathcal{F} := \big\{ f(\mathbf{W}, \cdot)\colon \mathbb{R}^{n_0} \to \mathbb{R} \;\big|\; \mathbf{W} \in \boldsymbol{\mathcal{W}} \big\}. \tag{2.24}$$

With slight abuse of notation, I write $\mathcal{F}(n_0, n_1, \ldots, n_{L-1}, 1)$ to specify a concrete function class by describing the architecture of the MLP, i.e., the number of processing units in each layer as well as input and output dimensions. Sometimes it is more convenient to describe an MLP by its depth d and identical width w of all hidden layers. In this case, I use the notation $\mathcal{F}(n_0, w \times d, 1)$. The type of non-linearity is typically fixed and mentioned separately.

2.4 Non-Convex Optimisation in the Context of Dynamic Programming

Before being able to set up the optimisation task residing in the root finding problem from Equations (2.17) and (2.18), it is necessary to introduce the existing approaches in the Dynamic Programming or Reinforcement Learning domain on an abstract level first. This is, because the nature of existing approaches dictates, which framework has to be employed for the construction of algorithms. Subsequently, I will introduce the required concepts and notation for non-convex optimisation in general, without already having to define the actual minimisation problem related to DP with NL-VFA. Finally, I can then outline the connection between RG or SG approaches and the non-convex optimisation methods. Later in Section 2.5, these preliminary steps allow me to define concisely the Mean Squared Bellman Error as the objective and also the minimisation task (cf. Equation (2.30)), which will produce an approximated value function with the smallest MSBE.

In Reinforcement Learning, there exist two fundamental approaches, which make use of gradient information and derivates for computing approximated value functions:

- **Residual Gradient** – A descent algorithm, which employs a complete gradient. It has been introduced in [Baird III, 1995].
- **Semi-Gradient** – A descent algorithm, which omits parts of a gradient. The book [Sutton and Barto, 2020] and references therein describe its construction. This type of algorithm is also called *Direct Algorithm* according to [Baird III, 1995].

Both approaches are descent algorithms in the sense that they seek to decrease stepwise the MSBE. The difference between Residual Gradient and Semi-Gradient formulations resides in the choice of a descent direction. For Residual Gradient algorithms, the complete gradient is calculated such that for sufficiently small step sizes an objective is reduced. In Semi-Gradient algorithms, one ignores the dependence on parameters through the value function inside the Bellman Operator. Consequently, the guarantee to have a descent is lost and Semi-Gradient algorithms may also increase the objective depending on the current parameters. During my analysis in Chapter 3, I will only investigate Residual Gradient algorithms in detail and from a theoretical perspective. When working with non-convex optimisation as toolset, there is no insight to be gained from a critical point analysis, if one would use directions for descending, which are not always guaranteed to point in the same half space as the gradient. A comparison of both approaches with empirical studies is available in [Zhang et al., 2020b] and references therein.

Non-convex optimisation is typically encountered in a Supervised Regression setting, i.e., approximating a desired target function with a given Multi-Layer Perceptron by using samples. In the following, I summarise the most important concepts regarding non-convex optimisation. Even if Supervised Regression is a different problem than Dynamic Programming or Reinforcement Learning, many ideas and parts of the notation will be shared with my later application of non-convex optimisation to approximate value functions with Multi-Layer Perceptrons.

Let $\mathcal{X}$ be a Hilbert space, i.e., a finite dimensional vector space over some field $\mathbb{K}$. For many engineering applications, the field $\mathbb{K}$ is simply the real numbers $\mathbb{R}$. The space is endowed with an inner product $\langle \cdot, \cdot \rangle_A : \mathcal{X} \to \mathbb{R}$. An inner product is a bilinear, symmetric positive-definite form and also induces a norm $\|x\|_A = \sqrt{\langle x, x \rangle_A}$ for all $x \in \mathcal{X}$. Analogously, $\mathcal{Y}$ and $\boldsymbol{\mathcal{W}}$ define a second and third Hilbert space with identical properties. The bold print for $\boldsymbol{\mathcal{W}}$ signalises that it will represent the collection of parameter matrices in an MLP, whereas the normal calligraphic symbols $\mathcal{X}$ and $\mathcal{Y}$ represent a simple Euclidean vector space. An unknown target function is denoted by $f^* \colon \mathcal{X} \to \mathcal{Y}$ and is only accessible in the form of sampled tuples $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ with $i = 1, \dots, N$. For each tuple it holds $y_i = f^*(x_i)$. All tuples together form the dataset $\mathcal{D} \coloneqq \{(x_1, y_1), \dots, (x_N, y_N)\}$, which is used for training an MLP.

The task for Supervised Regression is to adjust the parameters $\mathbf{W} \in \boldsymbol{\mathcal{W}}$ of a parametrised function $f \colon \boldsymbol{\mathcal{W}} \times \mathcal{X} \to \mathcal{Y}$ in such a way that it approximates f^* with minimal error. Therefore, one introduces the error function $E \colon \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_0^+$, which maps two elements in $\mathcal{Y}$ to a non-negative real number. A common choice is the squared distance $E(z_1, z_2) = \|z_1 - z_2\|_A^2$ with $z_1, z_2 \in \mathcal{Y}$. When combined with the parametrised function f and the dataset $\mathcal{D}$, it is possible to define an objective $\mathcal{J} \colon \boldsymbol{\mathcal{W}} \to \mathbb{R}_0^+$ for minimising as

$$\mathcal{J}(\mathbf{W}) = \frac{1}{N} \sum_{(x_i, y_i) \in \mathcal{D}} E(f(\mathbf{W}, x_i), y_i).$$

This objective maps the current choice for parameters $\mathbf{W}$ to their quality of approximation. The best possible approximation can be stated as

$$\mathbf{W}^* \in \operatorname*{argmin}_{\mathbf{W} \in \boldsymbol{\mathcal{W}}} \mathcal{J}(\mathbf{W}).$$

An effective tool for solving this minimisation problem is a Gradient Descent algorithm. First, one needs the differential map of $\mathcal{J}$ for the current parameters, i.e., $\mathrm{D}\,\mathcal{J}(\mathbf{W})$, and

applies it to some direction $\mathbf{H} \in \boldsymbol{\mathcal{W}}$. Second, one has to compute with the help of Riesz' Representation Theorem the unique value $\mathbf{G} \in \boldsymbol{\mathcal{W}}$ such that $\mathrm{D}\,\mathcal{J}(\mathbf{W})[\mathbf{H}] = \langle \mathbf{G}, \mathbf{H} \rangle_A$ applies. The gradient of $\mathcal{J}(\mathbf{W})$ is now given by $\nabla \mathcal{J}(\mathbf{W}) := \mathbf{G}$ and allows for the iterative procedure $\mathbf{W} \leftarrow \mathbf{W} - \alpha \nabla \mathcal{J}(\mathbf{W})$ for minimising $\mathcal{J}$. The term $0 < \alpha < 1$ denotes a step size and exists to compensate numerical errors in a computer system.

An even more effective tool is provided by Newton's Method (NM). One makes use of second-order differentials and calculates for two directions $\mathbf{H}_1, \mathbf{H}_2 \in \boldsymbol{\mathcal{W}}$ the unique value $\mathsf{H} \in \mathbb{R}^{|\boldsymbol{\mathcal{W}}| \times |\boldsymbol{\mathcal{W}}|}$ such that $\mathrm{D}^2\,\mathcal{J}(\mathbf{W})[\mathbf{H}_1, \mathbf{H}_2] = \langle \mathbf{H}_1, \mathsf{H} \cdot \mathbf{H}_2 \rangle_A$ holds. Once $\mathbf{H}_1$ and $\mathbf{H}_2$ are converted to a flat vector, then H denotes the Hessian matrix $\nabla^2 \mathcal{J}(\mathbf{W})$. A Newton's Method results in a similar iterative descent procedure of the form $\mathbf{W} \leftarrow \mathbf{W} - \alpha\eta$ to minimise $\mathcal{J}$. But due to the usage of Newton's direction $\eta = (\nabla^2 \mathcal{J}(\mathbf{W}))^{-1} \nabla \mathcal{J}(\mathbf{W})$, one achieves a super linear convergence rate as opposed to direct Gradient Descent. In practice, one typically works with approximations of the Hessian to ease the computation burden. Such an approach is then called an Approximated Newton algorithm.

The connection between Residual Gradient algorithms and non-convex optimisation in general arises now in the root finding problems from Equations (2.17) and (2.18). Since Residual Gradient methods compute a full gradient as it is done for non-convex optimisation, it is possible to investigate their behaviour by studying the underlying objective and its critical points. Furthermore, the realisation of Newton-like algorithms for the application in Dynamic Programming with Non-Linear Value Function Approximation becomes a promising topic. Thus, the construction and study of the Hessian matrix, or at least an estimation of it, is a mandatory task for my analysis.

2.5 Approximate Dynamic Programming with Function Approximation

If a state space is finite but too large for value functions to be stored in memory, or even has to be treated as infinite due to its size, an exact representation of the value function $V_\pi(s)$ for all states $s \in \mathcal{S}$ is practically impossible. This phenomenon has been coined as the *Curse of Dimensionality* [Bellman, 1957]. Also, for continuous state spaces, where a proper iteration over all states is not available due to the lack of a natural discrete formulation, exact representations are out of reach. An accurate value function approximation is thus useful and necessary for representing the actual value function V_π of a current policy π in any computer system. For training such a function approximation architecture, one has to rely on the Temporal Difference error from Equation (2.17) and use an optimisation approach. For the sake of simplicity, I omit in the following Q-functions, because they require the exact same construction steps.

2.5.1 The Mean Squared Bellman Error as Objective for Optimisation

Let $\mathcal{V} \subseteq \hat{\mathcal{V}}$ be the set of all approximations considered, which is typically a smaller set than that of all possible value functions $\hat{\mathcal{V}}$. Let further $F \colon \mathcal{S} \to \mathbb{R}$ with $F \in \mathcal{V}$ be a concrete but arbitrary value function approximation. To create a quality assessment for such an approximation at hand, one can combine the aforementioned Temporal Difference error from Equation (2.17) for a single state s together with a weighted norm $\|\cdot\|_\nu$, where the weighting is given by some function $\nu \colon \mathcal{S} \to (0, 1)$. This yields the Mean Squared

Bellman Error

$$\mathrm{MSBE}(F) = \frac{1}{2}\Big\|F - (\mathrm{T}_\pi F)\Big\|_\nu^2, \tag{2.25}$$

which accepts a value function approximation $F \in \mathcal{V}$ and returns a non-negative scalar value. The pre-factor $1/2$ is included for convenience. Due to the contraction property of T_π, the only function, which can render the MSBE zero, is the correct value function V_π.

As the MSBE contains a norm over a function space, the realisation of $\|\cdot\|_\nu$ comes in two different ways depending on the type of state space. In discrete spaces, one can enumerate all states and express the value function as a table. Hence, one can write down directly

$$\mathrm{MSBE}(F) = \frac{1}{2}\Big\|F - (\mathrm{T}_\pi F)\Big\|_\nu^2 = \frac{1}{2}\sum_{s\in\mathcal{S}} \nu(s)\Big(F(s) - (\mathrm{T}_\pi F)(s)\Big)^2, \tag{2.26}$$

where the weighting needs to satisfy $\sum_{s\in\mathcal{S}} \nu(s) = 1$. Since the weighting also shares the properties of a probability distribution, one can treat the summation as expectation and approximate it by Monte Carlo sampling. One arrives at

$$\mathrm{MSBE}(F) = \frac{1}{2}\mathop{\mathbb{E}}_{s\in\mathcal{S}}\left[\Big(F(s) - (\mathrm{T}_\pi F)(s)\Big)^2\right] \approx \frac{1}{2N}\sum_{i=1}^{N}\Big(F(s_i) - (\mathrm{T}_\pi F)(s_i)\Big)^2, \tag{2.27}$$

where N samples s_i are drawn according to the distribution ν. In continuous spaces, the value function remains, as the name suggests, a continuous function. Thus, one needs to work directly in the Hilbert space L^2 and realise the norm via integrals

$$\mathrm{MSBE}(F) = \frac{1}{2}\Big\|F - (\mathrm{T}_\pi F)\Big\|_\nu^2 = \frac{1}{2}\int_{\mathcal{S}} \nu(s)\Big(F(s) - (\mathrm{T}_\pi F)(s)\Big)^2 ds. \tag{2.28}$$

Now, ν takes the role of a normalised probability density function, which has to satisfy $\int_{\mathcal{S}} \nu(s)ds = 1$. Therefore, one can make use of Monte Carlo Integration again, which allows to write integrals as summations over (many) samples

$$\mathrm{MSBE}(F) = \frac{1}{2}\int_{\mathcal{S}} \nu(s)\Big(F(s) - (\mathrm{T}_\pi F)(s)\Big)^2 ds \approx \frac{1}{2N}\sum_{i=1}^{N}\Big(F(s_i) - (\mathrm{T}_\pi F)(s_i)\Big)^2. \tag{2.29}$$

As before in the discrete case, the samples s_i need to be distributed according to ν.

For the choice of ν, ergodic decision making problems play a special role. These problems possess a so-called steady state distribution $\xi(s) \in (0, 1)$ for each state $s \in \mathcal{S}$, which, when used in place of ν, allows for pleasing properties. Firstly, by choosing ξ as weighting in discrete state spaces, one maintains for Linear Value Function Approximation architectures the contraction properties of T_π when combining it with a linear projection onto that function space. Secondly, relying on ξ weighted states opens up the possibility to work model-free and online. Although this is of great interest for practical applications, this also brings up additional problems, which I cover on their own in Section 2.5.2.

In general, the choice of ν does not give rise to strong limitations. Most importantly, changing the weighting inside a norm to an equivalent one can only change the steepest descent direction. The types and locations of critical points remain the same, which is crucial for my critical point analysis in Chapter 3.

From Equations (2.27) and (2.29) it becomes clear that a realisation of an optimisation procedure for both types of state spaces results in the same instructions for a computer. As the only requirement, one has to ensure that the integrand $(F(s) - (\mathrm{T}_\pi F)(s))$ in Equation (2.28) is square integrable, i.e., an element in L^2. This is not a problem for the definition of V_π as the infinite discounted sum in Equation (2.1). Already necessary assumptions such as finite rewards $|r(\cdot)| < M$ for some $M > 0$ and bounded state spaces guarantee square integrability. The system dynamics must be chosen to fit implicitly into these conditions. Still, care must be taken for the selected function space $\mathcal{V}$. If using for example Neural Networks with arbitrary non-linearities as function class, square integrability might not be ensured. Furthermore, Neural Networks are defined for the whole Euclidean space, but the integral covers only a bounded subspace. Hence, to avoid issues at the boundary of the state space, the integral should be extended to the whole Euclidean space instead of just $\mathcal{S}$. However, this stands in conflict with the assumptions required for DP and also can affect in turn the square integrability of V_π. I leave these mathematical considerations as a topic for future research.

The extension of the MSBE to Q-factors is in both settings straightforward. One has to add another sum or integral over all actions to the definition of the MSBE and include the action as second argument for F.

Dynamic Programming with Non-Linear Value Function Approximation now manifests itself in the optimisation problem

$$F_\pi \in \operatorname*{argmin}_{F \in \mathcal{V}} \mathrm{MSBE}(F), \tag{2.30}$$

where F_π is an optimal approximation to V_π in terms of minimising the MSBE. In general, one has $F_\pi \neq V_\pi$ for at least some states $s \in \mathcal{S}$ and only aims to be close enough to V_π. The accuracy of the solution F_π as defined in Equation (2.30) is known to be bounded by

$$\left\|F_\pi - V_\pi\right\|_\xi \leq \frac{1+\gamma}{1-\gamma} \inf_{F \in \mathcal{V}} \left\|F - V_\pi\right\|_\xi. \tag{2.31}$$

Obviously, the challenge is to find a function space $\mathcal{V}$, which is convenient to use for optimisation and which still contains a good enough approximation for the desired value function V_π.

2.5.2 Model-Based vs. Model-Free

A desired property of any Reinforcement Learning algorithm is that it typically needs to work without requiring a model of the system under control. This property is called model-free and applies for example to the famous Q-Learning algorithm [Watkins and Dayan, 1992]. Mere transition data in the form of consecutive (s, a, r, s', a') tuples is enough to allow for a successful learning progress. These transition tuples can be created by executing a policy directly in the environment and collecting the states on the fly. A well performing simulator or a benevolent physical system are enough to gather all states, which are then distributed natively according to the steady state distribution ξ. Therefore, the empirical mean in Equations (2.27) and (2.29) approximates automatically the correct quantity. Aside from contraction properties for projected Bellman Equations, this is a main reason, why the assumption of an ergodic MDP is a core ingredient in many RL algorithms.

A second reason is related to the MSBE itself and arises from the approach via non-convex optimisation. It becomes visible once I set $\nu = \xi$ and write the norm in the MSBE from Equation (2.26) for discrete state spaces as inner product

$$\begin{aligned}\mathrm{MSBE}(\vec{F}) &= \frac{1}{2}\left\|\vec{F} - (\mathrm{T}_\pi\,\vec{F})\right\|_\xi^2 \\ &= \frac{1}{2}\left\langle \vec{F} - (\mathrm{T}_\pi\,\vec{F}), \vec{F} - (\mathrm{T}_\pi\,\vec{F})\right\rangle_\Xi \\ &= \frac{1}{2}\Big(\vec{F} - (\mathrm{T}_\pi\,\vec{F})\Big)^\mathsf{T}\Xi\Big(\vec{F} - (\mathrm{T}_\pi\,\vec{F})\Big). \end{aligned} \tag{2.32}$$

Here, the vector $\vec{F} = [F(1), \ldots, F(K_\mathcal{S})]^\mathsf{T}$ represents a tabular VFA. The diagonal matrix $\Xi = \mathrm{diag}(\xi_1, \ldots, \xi_{K_\mathcal{S}})$, which consists of all steady state distributions ξ_i, is responsible for defining a proper inner product. In general, to define an inner product $\langle x, y\rangle_A$, the matrix A must have full rank, be positive definite and symmetric. Since the steady state distribution Ξ is a diagonal matrix and thus always symmetric, the main requirement is that every state occurs under the current policy and, additionally, this needs to happen infinite many times. Having a proper inner product in Equation (2.32) will be essential for my analysis in Chapter 3, because only in a Hilbert space one has a closed-form expression for the gradient direction. However, here arises a fundamental problem of model-free RL algorithms. Many dynamical systems and environments, which are used for RL or DP applications, are easily not ergodic. Additionally, if changing policies are present, ergodicity can appear or vanish every time a policy is updated.

As an intuitive example, consider the task of balancing a pole, which is a famous and well-studied continuous control benchmark. Once a policy becomes optimal, the only visited states are those around the balancing point. All other states do not show up any more such that the inner product is no longer valid and the direction information, which is used to define a unique gradient, is destroyed. Consequently, training of function approximation architectures based on derivatives becomes impossible.

This issue also shows up in a different context of a more recent work [Bojun, 2020]. Furthermore, even a practical relevance of this problem has been described by the community. For example, de Bruin et al. [2015] observe this problem for the DDPG algorithm. A replay memory must be representative for the entire state space to enable stable training. Gu et al. [2016] elaborate on a necessary presence of good and bad transition examples in the training data. For a correct learning of the value function under some policy, bad events such as the execution of an action that would destroy a robot, must happen and be part of the transition tuples.

Whereas authors propose to involve rollouts based on hypothetical transitions or learned models, one could also arrive at the conclusion that the original model of the environment under control is needed to obtain the required transition data without executing actions in the real world. If there is no access to an analytic model, then one will face unavoidable obstacles regarding the distribution ξ. Namely, it is important to formulate at any time during the learning process a proper optimisation task.

Since samples from the entire state space are required to define the objective uniquely, one has to inject manually the transition data from parts of the state space, which would not be visited by the policy on its own. Therefore, I move completely away from the concept of ξ-distributed states and rely on other weightings, which cover the complete state space. Fortunately, critical points of an objective are in general not affected by a

change of weighting. The only result is that whenever states have to be sampled from the state space, they are drawn according to the weights ν.

A simple solution is to incorporate uniformly distributed values. Hence, whenever sampling is involved to obtain the MSBE, I use for $\nu(s)$ in Equations (2.26) to (2.29) the uniform distribution

$$\tilde{\nu}(s) = \frac{1}{\int_{\mathcal{S}} z dz}, \tag{2.33}$$

which is simply removed in the remaining document, because a positive constant factor does change the objective.

If it is mandatory to collect transitions in the state space independently of the currently active policy to implement Bellman Operators as needed by Equation (2.25), then the main consequence is that working model-based is the sole option. Only if one is using an analytical model, or alternatively a powerful simulator, where states and arbitrary transitions can be collected without restrictions, it is possible to distribute states freely in the entire state space and retrieve their successors. This implies that I move with my work entirely out of the Reinforcement Learning field and that I am now completely rooted in the Dynamic Programming methodology. There is no exploration happening any more and algorithms from later chapters cannot be applied for online learning. A beneficial side effect is that handling stochastic environments is no longer a severe challenge. This is because once the model is needed and available with high enough quality, incorporating noisy transitions only adds another expectation, which does not conflict with the remaining algorithm and is straightforward to compute. Where exactly this challenge would show up and cause problems is explained in Section 3.3.2.

2.6 Benchmarks

To test Dynamic Programming algorithms, one or more environments are required. Their description is provided here at the beginning of the document such that the remaining text can refer to these environments and show some numerical results on the fly for explaining theoretical concepts. Of course, these benchmarks appear in my experiments as well.

For stochastic dynamical systems, only discrete state space problems are considered. Their description takes the form of a graph or table specifying explicitly the probability for transiting from state s to s' when executing action a. Deterministic discrete problems are not covered in my work on their own, because they are a special case of the stochastic formulation and do not bring additional insights.

Deterministic continuous dynamical systems possess an analytic description of the behaviour as their core component. A transition function $f\colon \mathcal{S} \times \mathcal{A} \times \mathcal{W} \to \mathcal{S}$ maps a current state s, an action a and the disturbance w to a unique successor. Due to certain considerations, which are addressed explicitly in Section 3.3.2, stochastic transitions in continuous dynamical systems are not part of my work. Hence, I simplify the transition function to $f\colon \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ for the remaining work.

2.6.1 Adapted Seven State Star Problem

Baird's *Seven State Star Problem* [Baird III, 1995] is a well-known example to demonstrate convergence issues of algorithms. It is a discrete state space problem with only a few number of states. Furthermore, the *Star Problem* enables closed form and exact solutions

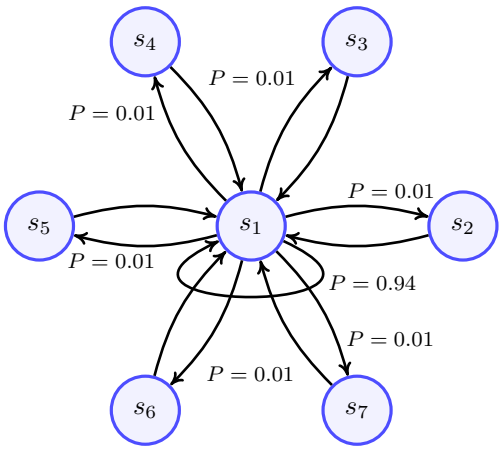


Figure 2.1: An adapted version of Baird's *Seven State Star Problem* [Baird III, 1995]. Additional transitions with low probabilities from the central node back to the six outer states are added. This restores an ergodic infinite horizon problem such that the MSBE is well-defined when relying on the steady state distribution.

because all transition parameters are known. Thus, it allows for extensive testing and comparison against a ground truth.

The original problem consists of six starting states with single transitions to a central absorbing terminal state. This implies that the *Star Problem* is non-ergodic. Since I want to maintain the setting, in which algorithms with linear function approximation architectures and projects on the subspace remain convergent, a direct usage of the *Star Problem* is not possible for my work. Its original construction stands in conflict with the ξ-weighted definition of the MSBE from Equation (2.32).

To obtain a dynamical system, where all states occur infinite many times under the current policy, I extend the *Star Problem* with transitions from the central node to all others. In Figure 2.1, a graph with all transitions and their probabilities is shown.

The graph suggests why it is called *Star Problem*. To complete the definition of a benchmark for Dynamic Programming applications, a reward is added to the central node with value one. Next, I use $\gamma = 0.99$ as discount factor whenever this benchmark is used. Finally, a policy for evaluation is already defined implicitly by setting the transition probabilities to fixed values. Since the Bellman Operator will be used in matrix form, there is no need to introduce a dedicated mapping from states to actions.

2.6.2 Simple Linear Dynamics

A mostly linear dynamical system in one dimension serves as a continuous benchmark, whose ground truth is known and easily available. Furthermore, the construction of controllers by hand is straightforward. Although this benchmark must be treated as a toy problem, it still occurs in the literature, e.g., in [Matheron et al., 2020], and is used for more fundamental statements about the behaviour of algorithms.

To meet the requirements of the Dynamical Programming methodology, the benchmark cannot be a linear dynamical system for the entire state-action space. At boundaries of $\mathcal{S}$, successor states need to be kept inside of the compact subset. Thus, the dynamics of the

environment is given by

$$f(s, a) = \text{clip}(s + a/2), \tag{2.34}$$

where $\text{clip}(\cdot)$ projects $s + a/2$ back into $\mathcal{S}$. The clipping operation is mandatory to restrict states to a finite range such that the state space is indeed a compact subset of $\mathbb{R}$. Thereby, the linearity is destroyed at the boundaries. The environment has a one dimensional state space $\mathcal{S} = [-3/2, 5/2]$ and, accordingly, a one dimensional action space $\mathcal{A} = [-1, 1]$. The borders for the state space and the scaling for actions are selected with no particular reason in mind. The action space matches the image of $\tanh(\cdot)$, such that it is straightforward to use parametrised policies.

I consider for this environment several types of rewards. Which type of reward will be used depends on the context and intended message of later chapters and experiments. The first reward function consists of piecewise constant functions with discontinuous transitions between them. It takes the form

$$r_1(s) = \begin{cases} 1 & \text{if } |s - r_c| \leq r_e \\ 0 & \text{else} \end{cases}, \tag{2.35}$$

where the constants $r_c = 1$ and $r_e = 1/5$ denote a centre and extent of the non-zero reward region, respectively. This type of reward would be classified as sparse and one would expect this reward to be harder for a learning algorithm.

The second reward function is the dense counterpart to the previous one and is constructed around a triangular shape. Intuitively, it should turn out favourably for the learning process. The reward signal is given by

$$r_2(s) = \begin{cases} \dfrac{s - r_c}{r_e} + 1 & \text{if } s \leq r_c \\ \dfrac{r_c - s}{r_e} + 1 & \text{else} \end{cases} \tag{2.36}$$

and makes use of the same constants as before. The main property of r_2 is that it is everywhere strictly monotonic, which will become necessary for Chapter 4.

A third reward function resembles r_1, but uses explicitly continuous transitions between different reward levels (left $r_l = 0.0$, mid $r_m = 1.0$ and right $r_r = 0.0$). These transitions are necessary to meet certain regularity requirements, which will be relevant for Chapter 4. Still, since it is mostly a constant function, it remains a challenging reward signal. The definition of the third reward is

$$r_3(s) = \begin{cases} r_l & \text{if } s \leq l_1 \\ \dfrac{r_m - r_l}{2} \sin\left(\dfrac{\pi}{r_1 - l_1} s - \pi\left(\dfrac{1}{2} - \dfrac{l_1}{r_1 - l_1}\right)\right) + \dfrac{r_m + r_l}{2} & \text{if } l_1 < s \leq r_1 \\ r_m & \text{if } r_1 < s \leq l_2 \\ \dfrac{r_m - r_r}{2} \sin\left(\dfrac{\pi}{r_2 - l_2} s + \pi\left(\dfrac{1}{2} - \dfrac{l_2}{r_2 - l_2}\right)\right) + \dfrac{r_m + r_r}{2} & \text{if } l_2 < s \leq r_2 \\ r_r & \text{if } r_2 < s \end{cases}. \tag{2.37}$$

The transitions consist of fitted sine curves and take place between $l_1 = r_c - r_e - r_t$ and $r_1 = r_c - r_e$ for the left transition and between $l_2 = r_c + r_e$ and $r_2 = r_c + r_e + r_t$ for

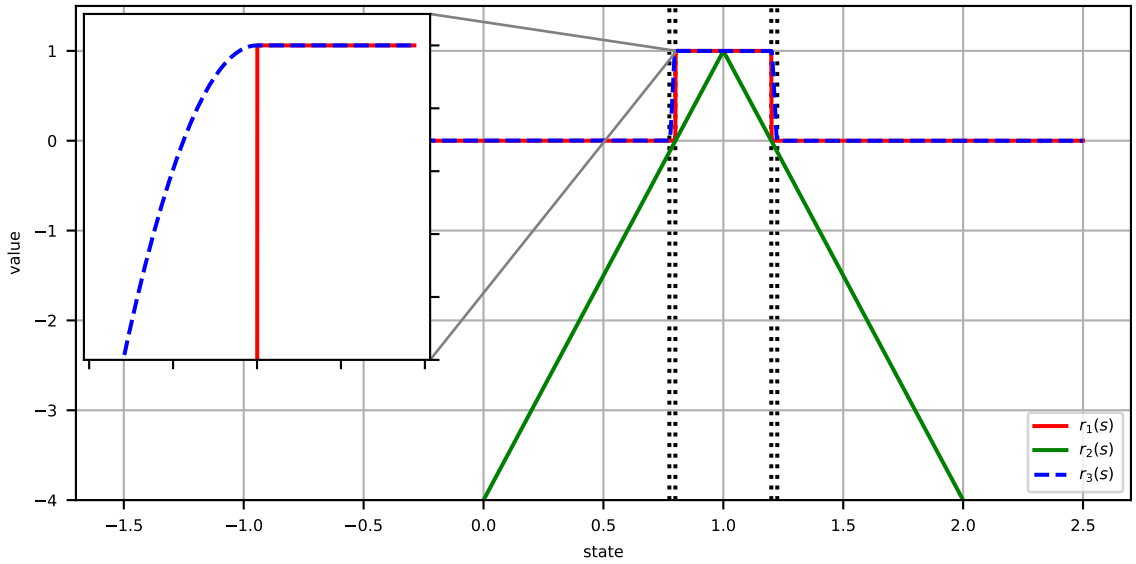


Figure 2.2: The reward functions from Equations (2.35) to (2.37) evaluated on the entire state space $\mathcal{S} = [-3/2, 5/2]$. The enlarged area reveals that r_3 contains indeed a smooth transition between different reward levels, whereas r_1 resembles a square wave signal.

the right one. The new constant $r_t = 0.025$ controls the width of transition regions. For convenience, Figure 2.2 shows all reward functions evaluated on the state space.

Independently of the actual reward function employed, the discount factor for this benchmark is set to $\gamma = 0.9$ such that accumulated and discounted rewards falls into a pleasing range.

2.6.3 Mountain Car

The Mountain Car control problem from [Moore, 1990] is a well-studied dynamical system. An underpowered car is supposed to drive up a hill and reach a goal. Since the engine of the car is not strong enough, an optimal policy consists of swinging up the car in the valley, which can be achieved by alternating the direction of acceleration according to the current velocity. The state of the car is described by its continuous position $x \in [-1.2, 0.6]$ and velocity $\dot{x} \in [-0.07, 0.07]$. Actions live in the range $a \in [-1, 1]$ and denote the fraction of the maximal available acceleration of the engine. Its sign serves as the direction indicator.

A standardised reference implementation of this benchmark, called *MountainCar-v0*, is available in the *OpenAI Gym* package [Brockman et al., 2016]. Unfortunately, it is still necessary to modify the reference implementation to obtain a valid infinite horizon problem.

First, rollouts in this environment are cut after 200 transitions. For discount factors close to one, this becomes problematic, because the approximation of arising from truncated geometric series becomes too coarse. Thus, any manipulation applied to rollouts is disabled. Second, I have replaced the built-in constant reward function. In its original form, this functions assigns independently of a state, an action and its successor state to any transition a punishment of -1. The new reward function favours being in a goal region by not giving

punishments in that area. It follows the pattern

$$r(s) = \begin{cases} 0 & \text{if } s \text{ in goal} \\ -1 & \text{else} \end{cases}, \tag{2.38}$$

where "in goal" indicates, whether the state s is considered to be a goal state. The goal region is defined as $x > 0.45$, where x denotes the position of the car. Lastly, once the car enters the goal region it jumps back to a starting state. Doing so ensures that all states can occur repeatedly even under an optimal policy and, in turn, ergodicity is present. The adjustments allow to use the environment in a non-episodic fashion and, thereby, establish the required formulation to fit into the MDP language. The MSBE in Equation (2.25) and the value function V_π from Equation (2.1) can be computed correctly now. The environment with all changes employed is called *MyMountainCar-v0* in the rest of this document.

For my later experiments, a baseline policy, which can successfully swing up the car, is given by the function

$$\pi(s) = \pi(\begin{bmatrix} x \\ \dot{x} \end{bmatrix}) \begin{cases} 1 & \text{if } \dot{x} \geq 0 \\ -1 & \text{if } \dot{x} < 0 \end{cases}. \tag{2.39}$$

This policy accelerates the car always in the current moving direction and thereby builds up momentum. It will be used for Policy Evaluation experiments whenever *MyMountainCar-v0* is involved.

Since the teleportation component of the state transitions is a rather severe manipulation of the dynamics, another variant of the Mountain Car is constructed. The adapted benchmark is called *MyMountainCar-v1* and is designed to run in an infinite horizon setting without additional manipulations of the system dynamics. The core behaviour remains roughly the same as before. Its differential equation takes the form

$$\begin{aligned} \ddot{x} &= pa - \frac{g}{m}\cos(3x) \\ \dot{x} &= \text{clip}\,(\dot{x} + \Delta t \cdot \ddot{x}) \\ x &= \text{clip}\,(x + \Delta t \cdot \dot{x}) \end{aligned} \tag{2.40}$$

with mass $m = 1\,\text{kg}$, gravity $g = 0.0025\,\text{m/s}^2$ and time step $\Delta t = 1\,\text{s}$. The action is scaled with the dimensionless power $p = 0.0015$ of the engine. The central difference between both versions of the benchmark resides in the clipping operation applied to the updated position. Collisions with boundaries are now perfect inelastic collisions with infinite heavy borders, thus, the velocity of the car flips its direction without changing the magnitude. Furthermore, the environment allows for infinite many transitions without artificial terminal conditions.

Additionally, the reward for the environment *MyMountainCar-v1* is changed to be the position of the car

$$r(s) = r(\begin{bmatrix} x \\ \dot{x} \end{bmatrix}) = x. \tag{2.41}$$

The dynamics and reward together define the task to stand still on top of the right hill. Due to the reflection at borders, an optimal policy also needs to learn to slow the car down. The small gap that appears on the right side behind the summit helps a little.

The policy described in Equation (2.39) is no longer the best solution, as it causes the car to roll down again. Hence, a more sophisticated policy is required as baseline. The following look-up table for actions

$$\pi(s) = \underbrace{\left.\left[\begin{array}{cccccc} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & -1 & 1 & \cdot & \cdot \\ \cdot & \cdot & -1 & 1 & \cdot & \cdot \\ \cdot & \cdot & -1 & 1 & \cdot & \cdot \\ \cdot & \cdot & -1 & 1 & 1 & \cdot \\ \cdot & \cdot & -1 & 1 & 1 & 1 \\ \cdot & \cdot & \cdot & 1 & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & -1 \\ \cdot & \cdot & \cdot & \cdot & -1 & -1 \\ 1 & 1 & 1 & \cdot & -1 & -1 \end{array}\right]\right\}}_{\text{velocity bins}} \text{position bins} , \tag{2.42}$$

where zero actions are represented by dots to emphasize the non-zero actions, achieves this and provides reproducible goal reaching. The policy accelerates the car carefully in the valley and breaks early enough to stop on the right hill in the goal area. To make use of this table, the position is discretised with ten bins and the velocity with six bins. The corresponding action can be retrieved from the responsible cell.

I set the discount factor for all variations of Mountain Car environments to $\gamma = 0.99$, if not stated otherwise during an experiment. Smaller values would blur the fine details of a value function and make it more difficult to distinguish successful policies from mediocre ones based on the accumulated rewards.

The change from *MyMountainCar-v0* to *MyMountainCar-v1* has been motivated by the insights in Section 4.3. Thus, for historical reasons, parts of my work rely still on the first version despite the fact that the second could serve as perfect replacement.

2.6.4 Cart Pole

The Cart Pole control problem [Barto et al., 1983, Florian, 2007] is a frequently used environment in the optimal control community. A pole is placed on a moveable cart and connected with a hinge to its top. The cart can be accelerated along a line, thereby controlling the angular velocity of the pole. With a carefully designed feedback controller, it is possible to balance the pole in an unstable equilibrium. Both the cart and the pole are described as point masses such that the whole dynamical system is defined by the vector $[x, \dot{x}, \theta, \dot{\theta}]^\mathsf{T} \in \mathbb{R}^4$. The position and velocity of the cart are denoted by x and $\dot{x}$. The angle and angular velocity of the pole use the symbol θ and $\dot{\theta}$. The boundaries for each component are given by $\pm[2.4, 12°, 2.0, 2.0]^\mathsf{T}$.

Similarly to the Mountain Car benchmark, there exists the *CartPole-v1* environment of the *OpenAI Gym* package, to which I add some modifications. The resulting benchmark receives the new name *MyCartPole-v0*.

The first modification is for the reward function. Namely, this benchmark also receives a new non-constant reward. Instead of rewarding every step with $+1$, including those states, which are considered to be a failure because the pole fell over or the cart left the allowed range, I only punish with -1 reward the transition into those failure states. Otherwise,

the reward is zero. The reward function can be written compactly but informally as

$$r(s) = \begin{cases} -1 & \text{if } s \text{ is failure state} \\ 0 & \text{else} \end{cases} . \tag{2.43}$$

A state is considered to be a failure if the position of the cart is getting closer than 0.024 units to the boundary. Similarly, the angle of the pole must stay 1.2° away from its maximal values. The second modification alternates the system behaviour. By adding connections from all terminal regions, i.e., the regions where a failure is present, to the start state, the episodic formulation of the balancing task is removed and I restore also here an infinite horizon MDP.

For the same reasons as for the Mountain Car benchmark, I also add another version of the Cart Pole environment, which can work without jumps in the transitions and has an adapted reward function. It is named *MyCartPole-v1*. The core dynamics are identical to the original environment. A new component is a projection operation applied to the position and angle update. Projecting the angle onto the state space allows the pole to rest on the left or right angle limit. Thereby, a full swing up is not required. But simultaneously, starting from the border of the angle range is harder than the original gentle start state at the centre, which is automatically restored once the pole fell over. A position projection keeps the cart on the screen. For the sake of completeness, the following equations describes the unchanged and original acceleration part of the dynamical system

$$\begin{aligned} c &= \frac{l m_p}{(m_p + m_c)} \\ h &= \frac{p}{m_p + m_c} a + c\dot{\theta}^2 \sin(\theta) \\ \ddot{\theta} &= \frac{g \sin(\theta) - \cos(\theta) h}{4/3 l - c \cos(\theta)^2} \\ \ddot{x} &= h - c\ddot{\theta} \cos(\theta), \end{aligned} \tag{2.44}$$

where the mass of the pole is $m_p = 0.1\,\mathrm{kg}$ and that of the cart $m_c = 1\,\mathrm{kg}$. The half length of the pole is $l = 0.5\,\mathrm{m}$, the gravity is approximately that on earth's surface $g = 9.8\,\mathrm{m/s^2}$ and the action input is multiplied by $p = 10$ to bring it in a proper range. The new modifications reside exclusively in the integrator part

$$\begin{aligned} \dot{\theta} &= \mathrm{clip}\left(\dot{\theta} + \Delta t \cdot \ddot{\theta}\right) & \dot{x} &= \mathrm{clip}\left(\dot{x} + \Delta t \cdot \ddot{x}\right) \\ \theta &= \mathrm{clip}\left(\theta + \Delta t \cdot \dot{\theta}\right) & x &= \mathrm{clip}\left(x + \Delta t \cdot \dot{x}\right). \end{aligned} \tag{2.45}$$

The additional clip$(\cdot)$ operations keep the four components inside the state space. The clipping step, which is applied to the position and angle, i.e., the lower line in Equation (2.45), also represents a perfect elastic collision with the boundaries. Once an angle or a position, which is exceeding the limits of the state spaces, is corrected, the corresponding velocity is multiplied by zero. Hence, either the cart position itself or the pole angle is no longer changing. For the Cart Pole benchmark, the integration time step is reduced to $\Delta t = 0.2\,\mathrm{s}$.

The reward signal is no longer a binary indicator, but measures the distance of some state towards an upright pose of the pole. It takes the form

$$r(s) = r(\begin{bmatrix} x & \dot{x} & \theta & \dot{\theta} \end{bmatrix}^\mathsf{T}) = -\left(\theta^2 + 0.01 \cdot \dot{\theta}^2\right). \tag{2.46}$$

A policy needs to learn a stable balancing to receive zero punishment. On top of that, a proper solution also requires to balance the pole towards the middle of the state space, because collision at boundaries interfere with the balancing objective. However, a dedicated position reward in the function is not required.

Lastly, I set the discount factor for all Cart Pole environments also to $\gamma = 0.99$, if not otherwise stated.

Chapter 3

Analysing the Critic: Characterising Critical Points of the Mean Squared Bellman Error

3.1 Introduction

The goal for this chapter is the investigation of algorithms to solve Policy Evaluation, which also corresponds to understanding training of a Critic when working with Actor-Critic algorithms. My investigation addresses two main building blocks. On the one hand, I aim at a full critical point analysis of the MSBE as objective for training MLPs. On the other hand, I am interested in a general application of non-convex optimisation to solve the underlying minimisation problem for Policy Evaluation or Critic training. There are several valid starting points and research directions, which all need to be investigated to arrive at a working algorithm.

For the beginning, there exist subtle but important differences in formulation of the MSBE for discrete and continuous spaces. Discrete states allow for exact learning, whereas continuous spaces always involve sampling and thus approximations. These differences lead to different possible statements and requirements. Furthermore, the sampling setting brings additional problems on its own. A subsequent important design choice for an algorithm is whether to choose only one-step Bellman Operators or whether one involves several consecutive transitions in an environment during training. Once multiple transitions are to be used, again different realisations become possible, namely compound Multistep Bellman Operators and TD(λ)-like methods. For both, one has to classify their impact on the objective and record all implied changes on the optimisation task.

Once a concrete algorithm is available, it is necessary to verify it. With discrete state spaces, it is possible to confirm algorithmic properties and to have a first verification of theoretical results in a controlled setting. Afterwards, I conduct experiments, which cover the continuous sampling based setting. Thus, they are more interesting and also unveil the practical applicability of the algorithm.

Parts of this chapter are based on my published work. The relevant papers are [Gottwald et al., 2018, 2021] and [Gottwald and Shen, 2022]. The notation and introduction in [Gottwald et al., 2021] is also contained partially in Chapter 2.

The remaining chapter is structured as follows. In the next section, Section 3.2, I review the existing work regarding the construction and analysis of Deep Reinforcement Learning algorithms in the context of my own work. I conduct a critical point analysis of the MSBE when using NN-VFA in Section 3.3 and present the results separately for discrete and continuous state spaces in Sections 3.3.1 and 3.3.2, respectively. The variants based on a multistep formulation are covered in Section 3.3.3. Section 3.4 is dedicated to my proposed Gauss Newton Residual Gradient algorithm and addresses details, which are relevant

for implementation and execution. This section also introduces pseudo code and covers experiments in discrete state spaces. Finally, in Section 3.5, I evaluate the performance and generalisation capabilities of my proposed method in several experiments on a larger scale and continuous state spaces. The chapter ends in Section 3.6 with a remark regarding conflicting insights of the critical point analysis and practical requirements.

3.2 Related Work

Recent attempts towards developing efficient NN-VFA methods follow the approach of extending the well-studied LVFA algorithms. These are a general family of gradient-based Temporal Difference algorithms, which have been proposed to optimise either the Mean Squared Bellman Error [Baird III, 1995, Baird III and Moore, 1999] or the Mean Squared Projected Bellman Error [Sutton et al., 2008, 2009]. The work in [Maei et al., 2009] adapts the results of developing the so-called *Gradient Temporal Difference* algorithms to a non-linear smooth manifold setting. The proposed approach requires projections onto some smooth manifold, which is practically infeasible because the so-called geometry of VFA manifolds is in general not available. Similarly, the approach developed in [Silver, 2013] projects estimates of the value function directly onto the vector subspace spanned by the parameter matrices of the NNs. Unfortunately, no further analysis or numerical development exists besides the original work.

Such a difficulty in studying and developing NN-VFA methods is partially due to an incomplete theoretical understanding of training NNs. The main challenge of the underlying optimisation problem is the strong non-convexity. Although there are several efforts towards characterising the optimality of NN-training, e.g., the papers [Kawaguchi, 2016, Nguyen and Hein, 2017, Haeffele and Vidal, 2017, Yun et al., 2018], a complete answer to the question is still missing and demanding. The work in [Shen, 2018a,b, Shen and Gottwald, 2019] addresses training of NNs using theory of differential topology and smooth optimisation. It has highlighted the importance of over-parametrisation to ensure proper convergence to a solution using an Approximated Newton algorithm. In this work, I introduce those techniques to the Neuro-Dynamic Programming domain.

Lately, there has been more interest in Residual Gradient algorithms. As described in [Baird III, 1995], Residual Gradient algorithms possess convergence guarantees, since they use a complete gradient of a well-defined performance objective. Thus, they are eligible for a critical point analysis. Unfortunately, these guarantees are coupled to solving the *Double Sampling* issue, i.e., the requirement of having several possible successors for every state to capture stochastic transitions. In a recent work [Saleh and Jiang, 2019], the authors explore the application of deterministic Residual Gradient algorithms to bypass the *Double Sampling* issue when using the Optimal Bellman Operator. Furthermore, they characterise empirically the impact of stepwise increased noise in environments and can motivate reviving Residual Gradient algorithms. Yin et al. [2022] report negative results regarding Residual Gradient algorithms based on an experimental investigation. However, at the time of writing my thesis, the document also appears to be incomplete and seems to suffer from an imprecise formulation of the RL setting. In this work, I investigate Residual Gradient algorithms in deterministic problems for Policy Evaluation instead of aiming directly at the optimal value function. This allows to use a Policy Iteration scheme as done in [Gottwald et al., 2018] with full control over the learning outcome. Involving a Gauss

Newton Residual Gradient algorithm overcomes limitations regarding convergence speed and a poor quality of the outcome, which have been described by the community.

In the paper [Wen et al., 2021], the authors also formulate the approximation of a Q-function as Residual Gradient algorithm. However, they are forced later-on in their document to rely on a Semi-Gradient method to circumvent the required *Double Sampling*. It is interesting to see that the authors are able to use a Semi-Hessian for their *Stackelberg Actor-Critic* algorithm. In my experiments, I find that a pure Semi-Gradient algorithm with its Semi-Hessian diverges or shows no improvement over time. Thereby, I can motivate both my selected approach via a Residual Gradient formulation and their extension of a Semi-Gradient algorithm towards the *Stackelberg Actor-Critic*.

Dabney and Thomas [2014] explore a natural Critic-Only Residual Gradient algorithm. Instead of constructing a Hessian matrix of the Bellman Residual, they introduce the *Fisher Information Matrix* to enhance the descent behaviour. Despite the similarity between a Hessian and the *Fisher Matrix*, they are not identical. Hence, a one-to-one translation of insights is not possible. The authors need to employ a two-timescale algorithm, whereas I can rely directly on non-convex optimisation. Furthermore, I can provide sophisticated analysis and characterisation of Hessian and, thus, also analyse the descent behaviour as a whole.

Another work [Cai et al., 2019], which is also strongly related to mine, pursues a similar idea, namely the importance of over-parametrisation. However, the authors address Semi-Gradient algorithms and thus work in a different setting. They show that the usage of NNs for NL-VFA with a redundant amount of adjustable parameters is mandatory for achieving good performance. They establish an implicit local linearisation and enable reliable convergence to a global optimum of the Mean Squared Projected Bellman Error. In [Brandfonbrener and Bruna, 2020], reliable convergence under over-parametrisation is also confirmed when treating Semi-Gradient TD-Learning as ordinary differential equation and investigating stationary points of the associated vector field. The Jacobian of over-parametrised networks, when evaluated for all discrete states, can have full rank, which is required for their theoretical investigation. Using an empirical approach as proposed in [Fu et al., 2019], the authors arrive at the conclusion that larger NN-VFA architectures result in smaller errors and boost convergence. In [Liu et al., 2019], the role of over-parametrisation is classified as an important ingredient for a variant of *Proximal Policy Optimisation* [Schulman et al., 2017] to converge to an optimum. Xiao et al. [2022] investigate over-parametrisation with linear function approximation for Residual Gradient algorithms and can derive regularisation terms for the non-linear case.

In my present document and my previously published work, I confirm the role of over-parametrisation when using Residual Gradient algorithms together with Non-Linear Value Function Approximation. Further, I provide insights from the global analysis perspective to characterise the behaviour of a descent algorithm.

3.3 A Critical Point Analysis of the Mean Squared Bellman Error

In the following, I present my theoretical investigation of Residual Gradient algorithms by analysing the critical points of the MSBE. Thereby, I follow the work in [Shen, 2018b] and translate those insights to the Neuro-Dynamic Programming domain. First, I investigate discrete state spaces and derive conditions, under which learning the exact value function

works reliably. Second, I extend my analysis to continuous spaces by changing to a sampling based approximation of the MSBE and adapt previous conditions accordingly. I characterise the importance of over-parametrisation, give design principles for MLPs and unveil a connection to other state of the art algorithms. Third, I include multistep methods in the analysis. By using multiple transitions of a dynamical system, one is able to formulate of a more informative objective without strengthening requirements on the function approximation architecture.

3.3.1 Exact Formulation with Discrete State Spaces

In discrete problems, one has two fundamental choices to approach a critical point analysis. Either one seeks for an approximation architecture, which is exact for each of the $K_{\mathcal{S}}$ states, or one is interested in an architecture, which is exact for only $N \ll K_{\mathcal{S}}$ sampled but unique states. Despite working with sampled and unique states would simplify the analysis and relax restrictions, as I show later for continuous spaces, this also means that I have to address generalisation to states outside of sampled ones. However, since the sampling case with discrete states would be almost identical to my approach for continuous state spaces, I only present the exact learning assumption here. This implies that I am interested in conditions for an MLP that would allow for a perfect solution to any state in a Markov Decision Process.

I consider MLPs of the form $\mathcal{F}(1, n_1, \dots, n_{L-1}, 1)$, i.e., fully connected feed forward networks with arbitrary depth or width but a one dimensional input and output. With discrete and finite spaces, one can evaluate an MLP $f \in \mathcal{F}$ for every available state $s \in \mathcal{S}$ and collect the evaluations of f as vector in $\mathbb{R}^{K_{\mathcal{S}}}$, where $K_{\mathcal{S}}$ is the number of states. As the simplest approach, I encode the discrete states with natural numbers for the single input unit and, thus, can treat $F(\mathbf{W}) \coloneqq [f(\mathbf{W}, 1) \cdots f(\mathbf{W}, K_{\mathcal{S}})]^{\mathsf{T}} \in \mathbb{R}^{K_{\mathcal{S}}}$ as the approximated value function for all states. Next, I define the Bellman Residual vector for a policy $\pi \colon \mathcal{S} \to \mathcal{A}$ through the Bellman Operator in matrix form as

$$\begin{aligned} \Delta_\pi(\mathbf{W}) &= F(\mathbf{W}) - \mathrm{T}_\pi F(\mathbf{W}) \\ &= F(\mathbf{W}) - \Big(R_\pi + \gamma P_\pi F(\mathbf{W})\Big) \\ &= \Big(I_{K_{\mathcal{S}}} - \gamma P_\pi\Big) F(\mathbf{W}) - R_\pi, \end{aligned} \tag{3.1}$$

where the term R_π contains the collection of all one-step rewards suitable for the matrix form of the Bellman Operator and P_π is the transition probability matrix under policy π. The identity matrix with shape $K_{\mathcal{S}} \times K_{\mathcal{S}}$ is denoted by $I_{K_{\mathcal{S}}}$. I use a capital Δ instead of δ to emphasize that I consider all transitions in here instead of just a single one as in Equation (2.17). Using a diagonal matrix $\Xi \coloneqq \operatorname{diag}(\xi_1, \dots, \xi_{K_{\mathcal{S}}})$ consisting of the steady state distributions ξ_i for all states, I can rewrite the norm in Equation (2.26) as the Neural Mean Squared Bellman Error (NMSBE) function

$$\mathcal{J}(\mathbf{W}) \coloneqq \frac{1}{2} \Delta_\pi(\mathbf{W})^{\mathsf{T}} \Xi \Delta_\pi(\mathbf{W}). \tag{3.2}$$

It is important to notice that the NMSBE function is generally non-convex in $\mathbf{W}$, and even worse, it is also non-coercive [Güler, 2010]. Namely, for $\mathbf{W} \to \infty$ one does not necessarily have $\mathcal{J}(\mathbf{W}) \to \infty$. Thus, the existence and attainability of global minima of $\mathcal{J}(\mathbf{W})$ are not

guaranteed in general. Nevertheless, when appropriate non-linear activation functions are employed in hidden layers, exact learning of finite samples can be achieved with sufficiently large architectures [Shah and Poon, 1999]. Thus, for my analysis, it is safe to assume that there exists an MLP able to approximate the value function V_π.

Assumption 1 (Existence of exact approximator)**.** *Let $V_\pi \colon \mathcal{S} \to \mathbb{R}$ be the exact value function under policy π. There exists at least one MLP architecture $\mathcal{F}$ as defined in Equation (2.24) together with a set of parameters $\mathbf{W}^* \in \boldsymbol{\mathcal{W}}$ such that the output of F and V_π coincide, i.e., one has*

$$f(\mathbf{W}^*, s) = V_\pi(s) \quad \forall s \in \mathcal{S}.$$

To conduct a critical point analysis of the NMSBE function, I first need to compute first-order derivatives. Due to the matrix form of the Bellman Operator T_π, I obtain the directional derivative of $\mathcal{J}$ at the point $\mathbf{W} \in \boldsymbol{\mathcal{W}}$ in direction $\mathbf{H} \in \boldsymbol{\mathcal{W}}$ as

$$\mathrm{D}\,\mathcal{J}(\mathbf{W})[\mathbf{H}] = \Delta_\pi(\mathbf{W})^\mathsf{T} \Xi \left(I_{K_\mathcal{S}} - \gamma P_\pi\right) \mathrm{D}\,F(\mathbf{W})[\mathbf{H}], \tag{3.3}$$

with $\mathrm{D}\,F(\mathbf{W})[\mathbf{H}]$ being the differential map of the MLP. As the function $F(\mathbf{W})$ is simply a superposed function evaluated at each state, I just need to compute the directional derivative of the MLP evaluated at a specific state s, i.e., $\mathrm{D}\,f(\mathbf{W}, s)[\mathbf{H}]$. Furthermore, the directional derivative of $f(\mathbf{W}, s)$ is a linear operator, hence the evaluation of directional derivatives of f for all states can be expressed as matrix vector multiplication and results in

$$\mathrm{D}\,F(\mathbf{W})[\mathbf{H}] = \underbrace{\Big[\operatorname{vec}(\nabla_\mathbf{W} f(\mathbf{W}, 1)) \cdots \operatorname{vec}(\nabla_\mathbf{W} f(\mathbf{W}, K_\mathcal{S}))\Big]^\mathsf{T}}_{=:G(\mathbf{W}) \in \mathbb{R}^{K_\mathcal{S} \times N_{net}}} \operatorname{vec}(\mathbf{H}), \tag{3.4}$$

where $\operatorname{vec}(\nabla_\mathbf{W} f(\mathbf{W}, s)) \in \mathbb{R}^{N_{net}}$ is the gradient of f with respect to the parameters evaluated at $\mathbf{W}$ for state s under the Euclidean norm and $\operatorname{vec}(\mathbf{H}) \in \mathbb{R}^{N_{net}}$ the corresponding direction. The operation $\operatorname{vec}(\cdot)$ transforms a matrix into a vector by stacking its columns. It acts on collections of matrices by concatenating the results of each individual vectorisation. The matrix $G(\mathbf{W})$ takes the role of the Jacobian for the evaluation of all states $F(\mathbf{W})$. Now, I can characterise all critical points of the NMSBE function $\mathcal{J}$ from Equation (3.2) by setting its gradient $\nabla_\mathbf{W} \mathcal{J}(\mathbf{W}) \in \mathbb{R}^{N_{net}}$ to zero, i.e., $\nabla_\mathbf{W} \mathcal{J}(\mathbf{W}) = 0$. Combining the results from Equations (3.3) and (3.4) together with Riesz' Representation Theorem yields the critical point condition

$$\nabla_\mathbf{W} \mathcal{J}(\mathbf{W}) \coloneqq G(\mathbf{W})^\mathsf{T} \left(I_{K_\mathcal{S}} - \gamma P_\pi\right)^\mathsf{T} \Xi \Delta_\pi(\mathbf{W}) \overset{!}{=} 0, \tag{3.5}$$

which is the counterpart[1] to Equation (19) in [Shen, 2018b] for the Dynamic Programming setting with exact learning. I derive the following proposition.

Proposition 1 (Suboptimal local minima free condition)**.** *Let an MLP architecture $\mathcal{F}$ satisfy Assumption 1. If the rank of the matrix $G(\mathbf{W})$ as constructed in Equation (3.4) is equal to $K_\mathcal{S}$ for all $\mathbf{W} \in \boldsymbol{\mathcal{W}}$, then any extremum $\mathbf{W}^* \in \boldsymbol{\mathcal{W}}$ of $\mathcal{J}$ realises the true value function V_π, i.e., $f(\mathbf{W}^*, s) = V_\pi(s)$ $\forall s \in \mathcal{S}$. Furthermore, the NMSBE function $\mathcal{J}$ is free of suboptimal local minima.*

[1] In this paper I use G in place of P to avoid confusion with the transition probability matrix

Proof. Since the underlying state space transitions under policy π are required to be Markovian and ergodic, both terms Ξ and $(I_{K_\mathcal{S}} - \gamma P_\pi)$ have full rank. Consequently, the expression $\Xi(I_{K_\mathcal{S}} - \gamma P_\pi)G(\mathbf{W}^*)^\mathsf{T}$ also has rank $K_\mathcal{S}$, if one claims that $G(\mathbf{W}^*)$ has full rank. Hence, there is only the trivial solution left for the linear system in Equation (3.5), meaning that the Bellman Residual $\Delta_\pi(\mathbf{W}^*)$ must be exactly zero for all states. Since the Bellman Residual is only zero for the unique fixed point V_π of the operator T_π, Assumption 1 implies that $\mathbf{W}^*$ corresponds to the true value function. Furthermore, the Bellman Residual appears as factor in the NMSBE. Hence, at any critical point the error vanishes and there are no suboptimal local minima. □

To make use of Proposition 1, I need to investigate, under what conditions the matrix $G(\mathbf{W})$ has full column rank. The first risk of loosing a full rank can be eliminated by choosing proper activation functions without zero derivatives for finite inputs. To see this, one has to look at the structure of $G(\mathbf{W})$. Carrying out the calculations in Equation (3.4) yields

$$G(\mathbf{W}) = \begin{bmatrix} \Psi_1^\mathsf{T}\left(I_{n_1} \otimes \phi_0^{(1)^\mathsf{T}}\right) & \cdots & \Psi_L^\mathsf{T}\left(I_{n_L} \otimes \phi_{L-1}^{(1)^\mathsf{T}}\right) \\ \vdots & \ddots & \vdots \\ \Psi_1^\mathsf{T}\left(I_{n_1} \otimes \phi_0^{(K_\mathcal{S})^\mathsf{T}}\right) & \cdots & \Psi_L^\mathsf{T}\left(I_{n_L} \otimes \phi_{L-1}^{(K_\mathcal{S})^\mathsf{T}}\right) \end{bmatrix}, \tag{3.6}$$

where the Kronecker products result from the layer wise definition of an MLP. The matrices $\Psi_l \in \mathbb{R}^{n_l \times n_L}$ obey the recursive definition $\Psi_l = \Sigma_l \bar{W}_{l+1} \Psi_{l+1}$ with $\Psi_L = 1$. The diagonal matrices Σ_l consist of $\dot{\phi}_l$, which is the output until layer l but using $\dot{\sigma}$ as activation function for layer l. A detailed construction is available in Appendix A. By choosing strictly monotonically increasing functions for σ, I remove one way for parts of $G(\mathbf{W})$ becoming zero. A second way to reduce the rank of $G(\mathbf{W})$ is to have parameter matrices $\bar{W}_l$, which are approaching zero. Due to random initialisation of all W_l, this typically does not happen. Although there are no theoretical guarantees, most matrices have full rank in practical applications, where noise or sampling is present. More demanding are situations, in which rows of $G(\mathbf{W})$ lie in a shared subspace and thus result in a rank deficient matrix. Obviously, one has to design MLPs in such a way that the risk for this situation is minimised. The evident requirement for $G(\mathbf{W})$ is to have enough columns for the given number of rows

$$N_{net} \geq K_\mathcal{S}, \tag{3.7}$$

meaning that I want to employ over-parametrised MLPs. In the DP setting, I have as an additional advantage that each block in $G(\mathbf{W})$ reduces to a single row, because the output layers are always scalar, i.e., $n_L = 1$. For a given amount of samples, there are less possibilities to have linear dependent rows compared to a more general multidimensional regression setting, where I would have $n_L \geq 1$.

Unfortunately, the requirement of using over-parametrised MLPs as in Equation (3.7) prevents a direct application of the theory. If I need as many adjustable parameters in an MLP as there are unique states, I could use a tabular representation in the first place and avoid dealing with Non-Linear Value Function Approximation. Note that the strong condition in Equation (3.7) stems from the assumption of being exact for all states.

In many applications, discrete states are not processed directly by an MLP but passed through an encoding step such as assigning random features. This leads to a potentially

weaker lower bound $\tilde{K} < K_{\mathcal{S}}$, since several states could receive an identical feature vector. But even with $\tilde{K}$ parameters, one ends in the same contradiction. Once it is possible to employ MLPs with $\tilde{K}$ parameters, one could also work directly with tabular methods in the feature space.

If the NMSBE is changed to be an approximation based on sampling, as done for continuous spaces in Section 3.3.2, I could reduce the number of MLP parameters from $K_{\mathcal{S}}$ states down to $N \ll K_{\mathcal{S}}$ samples. But as an immediate consequence, generalisation becomes an issue, because the MLP is trained by design with only a subset of all elements in the state space. Investigating the predictive capabilities for the remaining states of a discrete space, which not necessarily has a proper definition for a metric, is beyond the analysis presented in my work.

In order to obtain an efficient and effective algorithm, one can employ Newton-type optimisation procedures. Furthermore, to ease the work involved in the Hessian, one should aim at an Approximated Newton algorithm. A closer look reveals that the differential map as shown in Equation (3.3) is a candidate for the Gauss Newton approximation as in the non-linear regression setting. Indeed, for the second directional derivative of $\mathcal{J}$ at $\mathbf{W}$ with two directions $\mathbf{H}_1, \mathbf{H}_2 \in \boldsymbol{\mathcal{W}}$, I have

$$\begin{aligned} \mathrm{D}^2 \mathcal{J}(\mathbf{W})[\mathbf{H}_1, \mathbf{H}_2] = \Delta_\pi(\mathbf{W})^\mathsf{T} \Xi \left(I_{K_{\mathcal{S}}} - \gamma P_\pi\right) \mathrm{D}^2 F(\mathbf{W})[\mathbf{H}_1, \mathbf{H}_2] \\ + \mathrm{D} F(\mathbf{W})[\mathbf{H}_1]^\mathsf{T} \left(I_{K_{\mathcal{S}}} - \gamma P_\pi\right)^\mathsf{T} \Xi \left(I_{K_{\mathcal{S}}} - \gamma P_\pi\right) \mathrm{D} F(\mathbf{W})[\mathbf{H}_2], \end{aligned} \tag{3.8}$$

where I see that the first summand from the right hand side vanishes at any critical point $\mathbf{W}^* \in \boldsymbol{\mathcal{W}}$ according to Proposition 1. Thus, the evaluation of the Hessian of the NMSBE function at $\mathbf{W}^*$ is given by

$$\begin{aligned} &\mathrm{D}^2 \mathcal{J}(\mathbf{W}^*)[\mathbf{H}_1, \mathbf{H}_2] = \\ &\quad \mathrm{vec}(\mathbf{H}_1)^\mathsf{T} \underbrace{G(\mathbf{W}^*)^\mathsf{T} \left(I_{K_{\mathcal{S}}} - \gamma P_\pi\right)^\mathsf{T} \Xi \left(I_{K_{\mathcal{S}}} - \gamma P_\pi\right) G(\mathbf{W}^*)}_{=: \mathsf{H}_{\mathbf{W}} \mathcal{J}(\mathbf{W}^*) \in \mathbb{R}^{N_{net} \times N_{net}}} \mathrm{vec}(\mathbf{H}_2). \end{aligned} \tag{3.9}$$

This corresponds to the Gauss Newton approximation for non-linear least squares regression, i.e., defining the Hessian as product of the Jacobian and its transpose. My characterisation of critical points reveals this possibility for approximation as a side benefit. Using naively the product of the MLP's Jacobians $G(\mathbf{W}^*)$ as approximation would ignore the additional structure coming from the Bellman Operator.

To ensure proper behaviour for a GN algorithm, I further need to characterise the Hessian $\mathsf{H}_{\mathbf{W}} \mathcal{J}(\mathbf{W}^*)$ of the NMSBE at all critical points. Its quadratic form leads to the following result for MLPs.

Proposition 2 (Properties of the approximated Hessian)**.** *The Hessian of the NMSBE function $\mathcal{J}$ at any critical point $\mathbf{W}^*$ is always positive semi-definite. Furthermore, its rank is bounded from above by*

$$\mathrm{rank}(\mathsf{H}_{\mathbf{W}} \mathcal{J}(\mathbf{W}^*)) \leq K_{\mathcal{S}},$$

if the MLP satisfies Equation (3.7)*.*

Proof. Positive semi-definiteness of $\mathsf{H}_{\mathbf{W}} \mathcal{J}(\mathbf{W}^*)$ follows from its symmetric definition. As before, the steady state distribution Ξ and the matrix $(I_{K_{\mathcal{S}}} - \gamma P_\pi)$ have full rank. The rank of $G(\mathbf{W}^*)$ is at most $K_{\mathcal{S}}$. Due to $\mathsf{H}_{\mathbf{W}} \mathcal{J}(\mathbf{W}^*)$ being the product of these matrices, one obtains the upper bound on its rank. □

It is interesting to see that the rank condition from Equation (3.7) also allows the Hessian to become positive definite for the special case $K_{\mathcal{S}} = N_{net}$. If the matrix $G(\mathbf{W})$ has full rank, then the Hessian is positive definite. This has significant consequences for the optimisation problem. A positive definite Hessian at all critical points means that they are all local minima, thereby supporting further Proposition 1. There are no saddle points or maxima, where a gradient based optimisation strategy could get stuck. Thus, over-parametrisation of MLPs is not only important for Proposition 1, but also necessary from the algorithmic perspective. I confirm the proposed approximation for the Hessian in discrete state spaces with exact learning numerically in Section 3.4.3.

3.3.2 Sampling Based Approaches For Continuous State Spaces

In higher dimensional continuous state spaces, an exact representation of the value function based on a fine grained partitioning of $\mathcal{S}$ is typically impossible. This is due to the *Curse of Dimensionality* and one is forced to work directly with the continuous space. This causes MLPs to be of the form $\mathcal{F}(K_{\mathcal{S}}, \dots, 1)$ to accept $K_{\mathcal{S}}$ dimensional state vectors as input.

Since there cannot exist a transition probability matrix in continuous spaces and its corresponding discrete steady state distribution ξ, the NMSBE as shown in Equation (3.2) is not available here. Instead, I have to work with a finite number of samples $N \in \mathbb{N}$ to approximate the loss as done in Equation (2.29).

Another limitation for Residual Gradient algorithms is the so-called *Double Sampling* issue. Due to the expectation inside the Bellman Operator, for every single sample $s_i \in \mathcal{S}$ many possible successors s_i' are necessary to approximate this expectation empirically [Baird III, 1995]. The *Double Sampling* issue can be bypassed, if either an accurate model containing a description of stochastic transitions is available or if one has access to a simulator, where the state can be set freely to collect its successors. However, if one wishes to learn in a model-free manner or with rather limited and less powerful simulations, collecting successor samples becomes problematic. In a recent work [Saleh and Jiang, 2019], the authors rediscovered the application of Residual Gradient algorithms in deterministic environments. They are motivated by their observation that many environments and common benchmarks are deterministic or contain only a small amount of noise. Therefore, ignoring the stochastic term in the Bellman Operator does not cause too much harm. The consequence for my analysis is that I follow the same strategy and restrict myself to deterministic MDPs and analyse the algorithm in its purest form. As in [Saleh and Jiang, 2019], a deterministic algorithm is still of practical use. And, as elaborated in Section 2.5.2, if I already need to use analytical models or capable simulators such that the objective for training is defined properly, then the stochasticity in T_π does not pose any interesting challenge.

Therefore, the one-step TD-error from Equation (2.17) now simplifies for the i-th sample to

$$\delta(s_i) \coloneqq V(s_i) - r(s_i, \pi(s_i), s_i') - \gamma V(s_i'), \tag{3.10}$$

where $s_i' = f(s_i, \pi(s_i))$ is the successor of s_i when executing the action $\pi(s_i)$. As before, I collect the evaluation of the MLP for all N sampled states s_i as a vector and denote it by $F(\mathbf{W}) \coloneqq [f(\mathbf{W}, s_1) \cdots f(\mathbf{W}, s_N)]^\mathsf{T} \in \mathbb{R}^N$. Next, I rewrite the loss of Equation (2.29)

accordingly and obtain the sampled NMSBE for continuous state spaces as

$$\begin{aligned}\widetilde{\mathcal{J}}(\mathbf{W}) &\coloneqq \frac{1}{2N}\sum_{i=1}^{N}\Big(\underbrace{f(\mathbf{W},s_i)-r(s_i,\pi(s_i),s_i')-\gamma f(\mathbf{W},s_i')}_{\delta(s_i)}\Big)^2\\ &= \frac{1}{2N}\begin{bmatrix}\delta(s_1) & \cdots & \delta(s_N)\end{bmatrix}\begin{bmatrix}\delta(s_1)\\ \vdots\\ \delta(s_N)\end{bmatrix}\\ &= \frac{1}{2N}\widetilde{\Delta}_\pi(\mathbf{W})^\mathsf{T}\widetilde{\Delta}_\pi(\mathbf{W}),\end{aligned} \tag{3.11}$$

where $\widetilde{\Delta}_\pi(\mathbf{W}) \in \mathbb{R}^N$ now takes the form

$$\widetilde{\Delta}_\pi(\mathbf{W}) = F(\mathbf{W}) - R_\pi - \gamma F'(\mathbf{W}) \tag{3.12}$$

when I denote by F' the evaluation of f for all successor states s_i'. Similar lines of thought apply to this loss as to the discrete setting . But analogously to [Shen, 2018b], I now consider a finite set of sample states, at which in the best case the corresponding value function is approximated exactly. I formulate this situation for Residual Gradient algorithms precisely in the next definition.

Definition 1 (Finite exact approximator)**.** *Let $V_\pi\colon \mathcal{S}\to\mathbb{R}$ be the value function under policy π. Given N sample states $s_i\in\mathcal{S}$, I call an MLP $f\in\mathcal{F}$, which satisfies*

$$f(\mathbf{W},s_i) = V_\pi(s_i) \quad \forall i = 1,\dots,N$$

for some parameters $\mathbf{W}\in\boldsymbol{\mathcal{W}}$, a finite exact approximator of V_π based on the N sample states.

As in the discrete setting, one can choose sufficiently rich MLP architectures $\mathcal{F}$ and, thus, assume also in the continuous setting the existence of such an approximator.

Assumption 2 (Existence of finite exact approximators)**.** *Let $V_\pi\colon \mathcal{S}\to\mathbb{R}$ be the value function of policy π. Given N unique samples $s_i\in\mathcal{S}$, there exists at least one MLP architecture $\mathcal{F}$ as defined in Equation* (2.24) *together with a set of parameters $\mathbf{W}\in\boldsymbol{\mathcal{W}}$, such that the MLP $f(\mathbf{W},\cdot)\in\mathcal{F}$ is a finite exact approximator of V_π according to Definition 1.*

The sampling based approximation of the NMSBE provides a new complication for Definition 1 and Assumption 2, which does not occur in the exact learning setting. Definition 1 only describes the best possible situation. However, there are MLPs, which are not finite exact approximators, but may also have zero NMSBE.

Proposition 3 (The NMSBE includes bad solutions)**.** *Given an arbitrarily expressive MLP architecture $\mathcal{F}$, there can exist some parameters $\mathbf{W}\in\boldsymbol{\mathcal{W}}$, for which $f(\mathbf{W},\cdot)\in\mathcal{F}$ renders the NMSBE as defined in Equation* (3.11) *zero, without f being a finite exact approximator according to Definition 1.*

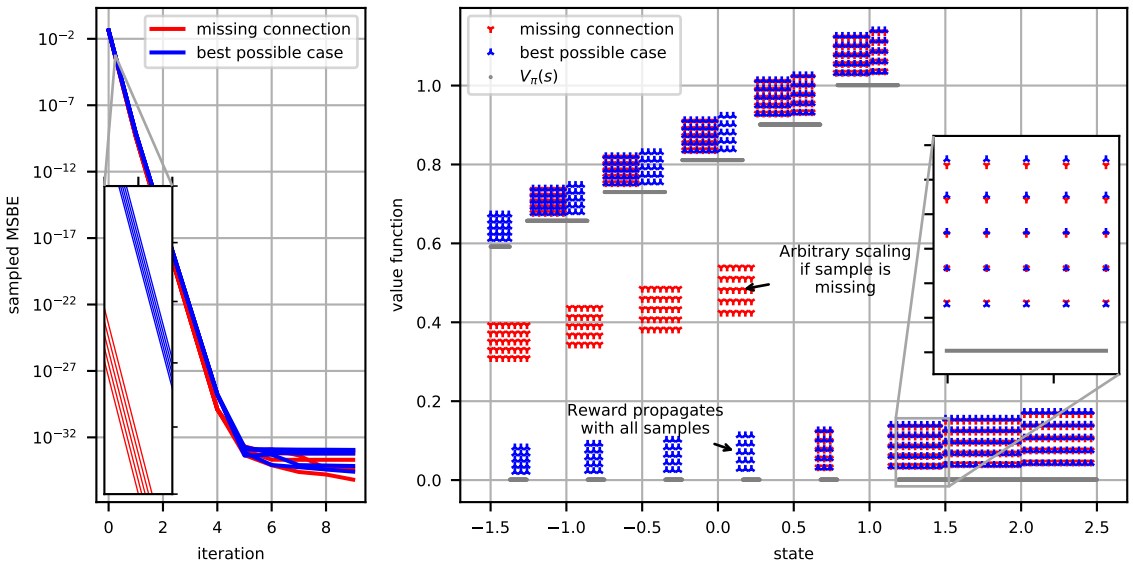
Proof. The NMSBE consists of N squared terms in a sum, which are computed with N unique sample states s_i and their successors. To minimise the NMSBE, an MLP has to shrink the magnitude of each individual $\delta(s_i)^2$. Let $s\in\mathcal{S}$ be one of the s_i and s' be its

successor. The term $\delta(s)^2$ can be thought of the task to align $f(\mathbf{W}, s)$ with $f(\mathbf{W}, s')$ while considering the one-step reward r and discount factor γ. If s' is not part of the sample states s_i, and if additionally its distance to any s_i is so large that the MLP can produce values for s' independently of other states, a new degree of freedom arises to minimise δ for s and s'. The output values of the MLP, which are required to solve the equation $f(\mathbf{W}, s) - \gamma f(\mathbf{W}, s') = c$ for an arbitrary constant c, or, in other words, render $\delta(s)$ zero, are only defined up to scale if s and s' are the free variables. A sufficiently large MLP can produce for almost all sample states s_i the correct value function, but can also create arbitrary values for s and s'. By the nature of δ, the MLP has zero NMSBE. Yet, it is not always a finite exact approximator. □

Whether the issue characterised by Proposition 3 becomes problematic, depends not only on the distribution of samples in the state space, but also on generalisation capabilities of an MLP. It is questionable, whether the issue in Proposition 3 has practical relevance. Of course, one can construct toy examples to demonstrate this issue. But in real engineering applications, the combination of high generalisation capabilities, randomly placed samples, more complicated system dynamics and the usage of multistep algorithms should alleviate the issue altogether. Thus, an empirical investigation should reveal, whether the effect of Proposition 3 is negligible and whether these generalisation capabilities are sufficient. It is part of the experiments in Section 3.5.

To provide an intuitive understanding of Proposition 3, let me construct such a toy example with the help of the one dimensional dynamical system as it is described in Section 2.6.2. Further, I use its reward signal r_1 from Equation (2.35). The policy for evaluation is set to be constant right, i.e., $\pi(s) = 1 \; \forall s \in \mathcal{S}$. To approximate value functions, I use piecewise constant functions, which consist of 32 segments with equal width. These approximation architectures are trained by minimising the sampled MSBE, of course with a Gauss Newton descent algorithm. Training data consists also of 32 equally spaced samples in the state space such that one sample per constant region of the approximation architecture is available. To reveal all possible scaling degrees of freedom, the 12^{th} and 13^{th} state sample can be omitted during training. Doing so creates a gap in the covered state space and cuts the connection between successors. I train five architectures, which use all the training samples. They will serve as a reference for the best possible case one can have. Each architecture has its 32 initial parameters set to the same value, i.e., the initial value function is constant for the complete state space. The initial parameters for the five different architectures are offset to each other and take the values $0.42 + i \cdot 0.025$ with $i \in \{-2, -1, 0, 1, 2\}$. I select those values due to the final range in which V_π is living and to have a proper spread for visualisation. Another collection of five architectures, which employ the same initialisation strategy, are trained in the same setting but with the gap included. Lastly, a ground truth is available in the from of rollouts, which start from a large number of densely packed starting states. Further details regarding the computing of ground truths are available in Section 3.5.1. Alternatively, the computation of V_π could also exploit the tabular nature of a piecewise constant function approximation and apply directly the Bellman Operator under policy π as it can be done in discrete MDPs. The results of this experiment are provided in Figure 3.1.

The toy problem confirms Proposition 3 numerically. Figure 3.1a shows that the sampled MSBE, as it is computed in Equation (2.29), becomes zero when taking machine precision into account. However, the corresponding approximated value functions in Figure 3.1b do



(a) Error over time

(b) Approximated value functions

Figure 3.1: Sampling inside the MSBE leads to problems for training piecewise constant approximation architectures. Their parameters can only be trained up to some scaling constant (blue markers). It is impossible to change parameters away from their initial values, if connections between states are missing (red markers). For both, the sampled MSBE approaches zero over time.

not match the ground truth. Neither the training with artificial connectivity issues nor the training, where all sample states are included, get everywhere close to $V_\pi(s)$. The offsets from initialisation or the initial values themself remain visible. The best possible case, which is shown in blue in Figure 3.1b, follows everywhere the structure from the ground truth, i.e., the approximated value functions have the same steps and offsets between different segments. But the effect from initialisation is not corrected during the training. The initial offset of the parameters remains and results in shifted value functions. This behaviour already implies that the sampling based approach to the MSBE poses a first challenge for realising Policy Evaluation with function approximation via non-convex optimisation. However, it is important to emphasize that an offset for the entire value function does not skew the outcome of Policy Improvement. Hence, the combination of Policy Evaluation and Improvement can still result in a working Approximate Policy Iteration algorithm. Additionally, since the TD-errors are also part of Semi-Gradient algorithms, they suffer from the same problem to a certain extent. The negative aspect of Proposition 3 becomes apparent, once the training process does not include the entire state space any more. By removing the 12^{th} and 13^{th} sample from the training data, one removes the connection between states on the left side of the gap ($s \leq 0.13$) from those on the right side ($s > 0.13$). This allows the MSBE to be zero for arbitrary values of states directly left of the gap. Hence, the value function approximations, which are shown in red, contain in the middle of the state space their initial values. They are not changed during training, because the reward signal is not propagating across the entire state space. States further to the left correctly adapt their values to produce those typical steps as they show up for the ground

truth. This scaling freedom has now a fatal impact on the outcome of Policy Improvement. The solution to $(\mathrm{T}_{\pi'}\tilde{V})(s) = (\mathrm{T}_{\mathrm{g}}\tilde{V})(s)\ \forall s \in \mathcal{S}$ from Equation (2.10), where $\tilde{V}$ denotes the approximated value function, depends entirely on the initialisation of the function approximation architecture.

Unfortunately, there is no feasible method to determine, whether the reward signal is propagating correctly or not. One possible way for compensating the scaling degree of freedom is to enhance the connectivity between consecutive states by employing multistep Bellman Operators. Once there are more connections between sample states and their various successors available, the chance that parts of the state space are not sufficiently connected becomes smaller and the reward signal would propagate more reliably. Hence, an approximation architecture has less freedom to approximate functions, which would shrink the magnitude of the (multistep) Temporal Difference error to zero without becoming close to V_π. Of course, one has to expect that results will still not be perfect, but the distance from approximated value functions to the ground truth should be smaller. Hence, this toy problem provides a first motivation to extend the analysis of this section also to the multistep setting. In Section 3.3.3, I give further motivation and extend the analysis of critical points to multistep Bellman Operators.

Beside the investigation of the sampled MSBE itself, its combination with Multi-Layer Perceptrons, namely the NMSBE, needs to be investigated as well. It is important to know, whether its critical points form a set of solutions matching Definition 1 or whether there are further complications. An analysis of critical points follows the same steps as in Section 3.3.1. First, I need the differential map of Equation (3.11). It takes the form

$$\begin{aligned}\mathrm{D}\,\widetilde{\mathcal{J}}(\mathbf{W})[\mathbf{H}] &= \frac{1}{N}\widetilde{\Delta}_\pi(\mathbf{W})^\intercal\Big(\mathrm{D}\,F(\mathbf{W})[\mathbf{H}] - \gamma\,\mathrm{D}\,F'(\mathbf{W})[\mathbf{H}]\Big)\\ &= \frac{1}{N}\widetilde{\Delta}_\pi(\mathbf{W})^\intercal\Big(G(\mathbf{W}) - \gamma G'(\mathbf{W})\Big)\,\mathrm{vec}(\mathbf{H}),\end{aligned} \tag{3.13}$$

where the definition of $G(\mathbf{W}) \in \mathbb{R}^{N\times N_{net}}$ applies as in the discrete setting. The matrix G' results from using s'_i as input. Using this map, critical points are characterised by setting the gradient to zero

$$\nabla_{\mathbf{W}}\widetilde{\mathcal{J}}(\mathbf{W}) \coloneqq \frac{1}{N}\Big(\underbrace{G(\mathbf{W}) - \gamma G'(\mathbf{W})}_{=:\widetilde{G}(\mathbf{W})\in\mathbb{R}^{N\times N_{net}}}\Big)^\intercal\widetilde{\Delta}_\pi(\mathbf{W}) \stackrel{!}{=} 0. \tag{3.14}$$

Apparently, the critical point condition for the continuous setting based on N unique samples in Equation (3.14) takes a similar form as that of the discrete setting in Equation (3.5). But since I no longer investigate the exact learning scenario, I have to reformulate Proposition 1 and obtain a slightly different version.

Proposition 4 (Suboptimal local minima free condition)**.** *Let an MLP architecture $f \in \mathcal{F}$ satisfy Assumption 2. If the rank of the matrix $\widetilde{G}(\mathbf{W})$ as defined in Equation* (3.14) *is equal to N for all $\mathbf{W} \in \boldsymbol{\mathcal{W}}$, then any extremum $\mathbf{W}^* \in \boldsymbol{\mathcal{W}}$ of $\widetilde{\mathcal{J}}$ achieves zero NMSBE. Furthermore, the NMSBE function $\widetilde{\mathcal{J}}$ of Equation* (3.11) *is free of suboptimal local minima.*

Proof. As before, Equation (3.14) defines a linear equation system in the sampled Bellman Residual vector $\widetilde{\Delta}_\pi(\mathbf{W})$. If one claims that $\widetilde{G}(\mathbf{W})$ has full rank, there is only the trivial solution left for $\widetilde{\Delta}_\pi(\mathbf{W})$. By Assumption 2, one knows that such an MLP exists. Furthermore, the Bellman Residual appears as factor in the sampled NMSBE. Hence, at any critical point the error vanishes completely and there are no suboptimal local minima. □

I now address the requirements for a full rank of $\widetilde{G}(\mathbf{W})$, since this matrix forms the backbone of Proposition 4. As the first requirement, $\widetilde{G}(\mathbf{W})$ has to be non-zero to define a proper equation system and enforce a trivial solution in terms of $\widetilde{\Delta}_\pi(\mathbf{W})$ in Equation (3.14). For both differential maps $G(\mathbf{W})$ and $G'(\mathbf{W})$, the design principles apply individually. Hence, it is unlikely in practice that for N unique sample states simultaneously both matrices vanish elementwise on their own. More troublesome is the distance between a state s and its successor s'. If $\|s - s'\| \to 0$, which happens for example whenever s is getting close to a fixed point of the dynamical system, then the discount factor $\gamma \in (0, 1)$ prevents a perfect cancellation of $G(\mathbf{W})$ with $G'(\mathbf{W})$. Aside from those fixed points in the state space, it is again unlikely to observe perfect cancellation of these two matrices in practice. As the second requirement, the rank of $\widetilde{G}(\mathbf{W})$ is important. Obviously, it is more difficult to make concise statements compared to discrete and exact learning setting. For the rank of the sum of two matrices, a known inequality is

$$\operatorname{rank}(A + B) < \operatorname{rank}(A) + \operatorname{rank}(B), \tag{3.15}$$

which implies that I still have to increase the rank of $G(\mathbf{W})$ and $G'(\mathbf{W})$ individually to push the upper bound for the rank of $\widetilde{G}(\mathbf{W})$ high enough to allow for a full rank. This leads to similar design principles for the MLP as in the discrete setting. Hence, the more complex $\widetilde{G}(\mathbf{W})$ still complies to considerations of the discrete setting regarding linear dependent rows. Of course, these properties and requirements come without any guarantees. In general, I expect that carefully constructed examples (yet almost pathological) exist, where a drop of the rank of $\widetilde{G}(\mathbf{W})$ happens frequently. But in practice, where numerical errors and sampled quantities are present, I do not consider this to become a significant problem. When taking a closer look at $\widetilde{G}(\mathbf{W})$ itself, I find that the overall block structure of $G(\mathbf{W})$ as shown in Equation (3.6) remains. I have

$$\widetilde{G}(\mathbf{W}) = \begin{bmatrix} \widetilde{G}(\mathbf{W})_{11} & \cdots & \widetilde{G}(\mathbf{W})_{1L} \\ \vdots & \ddots & \vdots \\ \widetilde{G}(\mathbf{W})_{N1} & \cdots & \widetilde{G}(\mathbf{W})_{NL} \end{bmatrix} \tag{3.16}$$

with the blocks

$$\widetilde{G}(\mathbf{W})_{ij} = \Psi_j^\mathsf{T} \left(I_{n_j} \otimes {\phi_{j-1}}^{(i)} \right)^\mathsf{T} - \gamma \Psi_j'^\mathsf{T} \left(I_{n_j} \otimes {\phi'_{j-1}}^{(i)} \right)^\mathsf{T}.$$

The vectors $\phi_l \in \mathbb{R}^{n_l}$ result from the evaluation of all layers for state s and ϕ'_l from using the successor s'. Similarly, the matrices Ψ and Ψ' use s and s' for their computation. I provide slightly more detailed construction steps in Appendix A. Unfortunately, one can see from Equation (3.16) that no additional statements or simplifications are possible.

Various authors report a slow convergence of RG algorithms due to the similarity of $G(\mathbf{W})$ and $G'(\mathbf{W})$, e.g., the work in [Baird III, 1995, Bertsekas, 2012, Dabney and Thomas, 2014, Zhang et al., 2020b]. I can identify two remedies here. The first is already visible in Equation (3.14) or Equation (3.16), where the contribution from successors states in $G'(\mathbf{W})$ comes with the prefactor γ. If using n-step returns, as done for example in [Mnih et al., 2016], the discount factor would come with higher powers, thus, more efficiently taking away the cancelling effect of $G'(\mathbf{W})$ onto $G(\mathbf{W})$ if a successor stays after several steps still close to its original state. For large enough lookahead, γ^n would become small

enough to allow for ignoring $G'(\mathbf{W})$ altogether, which also helps with the desired full rank of $\widetilde{G}(\mathbf{W})$. Also, Semi-Gradient issues such as the limited applicability of classic optimisation methods seem to be avoidable, because for long enough lookahead the omitted dependence of derivatives with respect to MLP parameters in the TD target vanishes naturally. Furthermore, it is interesting to see that the extension of n-step returns to a full TD(λ) algorithm is a core component in *Proximal Policy Optimisation* [Schulman et al., 2017] or *Generalized Advantage Estimation* [Schulman et al., 2016]. By using TD(λ), one obtains per sampled state more information without increasing their amount. Hence, the required number of parameters to obtain a full rank for $G(\mathbf{W})$ does not increase. This direction is explored further in Section 3.3.3. A second remedy is related to vanishing gradients of $\widetilde{\mathcal{J}}$, if s and s' are close by. Since this effect is coupled to the geometry of the loss surface, a natural solution is to use second-order gradient descent, which takes the curvature explicitly into account to define a descent direction. Hence, I propose to employ analogously to the discrete setting a Gauss Newton Residual Gradient Algorithm. Other approaches to overcome curvature issues, for example momentum based descent algorithms, are too reliant on the dynamical runtime behaviour as well as on initialisation. They are prone to excessive hyper parameter tuning and frequent restarts. In the worst case, they complicate reproducibility, which is also a reason, why I have decided to use second-order optimisation.

As before, to see the possibility for a GN approximation of the Hessian, I first write down the second-order differential map of $\widetilde{\mathcal{J}}$ at some point $\mathbf{W} \in \boldsymbol{\mathcal{W}}$ for two directions $\mathbf{H}_1, \mathbf{H}_2 \in \boldsymbol{\mathcal{W}}$

$$\begin{aligned} \mathrm{D}^2\, \widetilde{\mathcal{J}}(\mathbf{W})[\mathbf{H}_1, \mathbf{H}_2] = \frac{1}{N} \widetilde{\Delta}_\pi(\mathbf{W})^\mathsf{T} \Big(\mathrm{D}^2\, F(\mathbf{W})[\mathbf{H}_1, \mathbf{H}_2] - \gamma\, \mathrm{D}^2\, F'(\mathbf{W})[\mathbf{H}_1, \mathbf{H}_2] \Big) \\ + \frac{1}{N}\, \mathrm{D}\, \widetilde{\Delta}_\pi(\mathbf{W})[\mathbf{H}_1]^\mathsf{T} \Big(\mathrm{D}\, F(\mathbf{W})[\mathbf{H}_2] - \gamma\, \mathrm{D}\, F'(\mathbf{W})[\mathbf{H}_2] \Big). \end{aligned} \tag{3.17}$$

Since the first summand contains the Bellman Residual as factor, it vanishes at any critical point $\mathbf{W}^*$ of $\widetilde{\mathcal{J}}$ due to the assumption of exact learning at sample states. This removes the contribution of second-order derivatives of the MLP and allows me to simplify the Hessian to

$$\mathrm{D}^2\, \widetilde{\mathcal{J}}(\mathbf{W}^*)[\mathbf{H}_1, \mathbf{H}_2] = \frac{1}{N} \operatorname{vec}(\mathbf{H}_1)^\mathsf{T} \underbrace{\Big(G(\mathbf{W}^*) - \gamma G'(\mathbf{W}^*) \Big)^\mathsf{T} \Big(G(\mathbf{W}^*) - \gamma G'(\mathbf{W}^*) \Big)}_{\mathsf{H}_\mathbf{W}\, \widetilde{\mathcal{J}}(\mathbf{W}^*) \in \mathbb{R}^{N_{net} \times N_{net}}} \operatorname{vec}(\mathbf{H}_2). \tag{3.18}$$

It becomes clear that Proposition 2 applies almost unchanged.

Proposition 5 (Properties of the approximated Hessian)**.** *The Hessian of the NMSBE function $\widetilde{\mathcal{J}}$ from Equation* (3.11) *is at any critical point $\mathbf{W}^*$ always positive semi-definite. Furthermore, its rank is bounded from above by*

$$\operatorname{rank}(\mathsf{H}_\mathbf{W}\, \widetilde{\mathcal{J}}(\mathbf{W}^*)) \leq N.$$

Proof. Positive semi-definiteness follows from the symmetric definition. The rank of the matrix $(G(\mathbf{W}^*) - \gamma G'(\mathbf{W}^*))$ is at most N. For $\mathsf{H}_\mathbf{W}\, \widetilde{\mathcal{J}}(\mathbf{W}^*)$ being the product of these, one gets the upper bound on its rank. □

I want to emphasize that due to the sampling based formulation of the objective, the requirement to employ over-parametrised MLPs

$$N_{net} \geq N \tag{3.19}$$

is far less restricting in terms of the size of the Hessian than in the discrete setting with exact learning. One can control the amount of samples N and select a reasonable size. With nowadays hardware capabilities, computational concerns regarding a second-order optimisation method are no longer that severe. MLPs can become large enough for NDP applications while still allowing for working with Hessians in a reasonable amount of time. I further address practical concerns in Section 3.4 and as part of the experiments.

For $\mathsf{H}_{\mathbf{W}}\,\widetilde{\mathcal{J}}(\mathbf{W}^*)$ there still exists the special case for $N_{net} = N$, which causes the matrix to become positive definite if its rank is full. Hence, the considerations from the discrete setting apply here as well.

The attentive reader might have spotted that Propositions 4 and 5 rely both on Assumption 2 being fulfilled. However, the case, where the assumption does not hold, has not been addressed yet. Doing so leads to the following remark.

Remark 1 (What if finite exact approximators do not exist)**.** *If a finite exact approximator is out of reach, i.e., Assumption 2 is not satisfied, then for MLPs $f \in \mathcal{F}$ the condition*

$$\widetilde{\Delta}_{\pi}(\mathbf{W}) \neq 0 \quad \forall \mathbf{W} \in \boldsymbol{\mathcal{W}}$$

applies. Depending on the rank of $\widetilde{G}(\mathbf{W})$, the objective $\widetilde{\mathcal{J}}$ is either free of critical points if the rank is full, or arbitrary critical points exists, because there are solutions to Equation (3.14) *other than the trivial one. They fall into the null space of $\widetilde{G}(\mathbf{W})$ and thereby create the zero for the critical point condition.*

The typical logic and line of argumentation applies for the rank of $\widetilde{G}(\mathbf{W})$ as in the previous propositions. Namely, one would expect the row rank of $\widetilde{G}(\mathbf{W})$ to be full in typical engineering applications. The empirical results in Section 3.5.4 will support this expectation. Thus, the situation outlined in Remark 1 results in a descent algorithm, which can be run as long as the numerical precision of a computer system permits further progress. There are no critical points, where the iterates would accumulate, such that one can decrease the NMSBE forever and ever with diminishing progress.

In summary, I have two kind of approaches, namely exact learning and sampling based approximation. For discrete state spaces, I have a sound and exact algorithm with verifiable local quadratic convergence as I will show later. For continuous state spaces, only a sampling based algorithm is realisable. As I have shown, it possesses a matching behaviour with only slightly altered propositions. Thus, the only major difference is indeed the formulation of the loss. By using sampling, I allow for broader applications without sacrificing the entire theoretical foundation. The last uncertainty left is, how many sample states are required, i.e., how to select the size of N for a certain MDP and given MLP. I treat this problem concerning sampling complexity mainly as future research, although there will be some empirical statements in that regard.

3.3.3 Multistep Methods For Continuous State Spaces

A key ingredient in well-performing RL algorithms are n-step returns [Mnih et al., 2016] or full TD(λ)-like multistep lookahead mechanisms [Schulman et al., 2016, 2017]. In the previous section, there have been already hints that these capabilities of multistep formulations

could arise due to their effect on the critical point condition from Equation (3.14). Furthermore, multistep methods should be more robust regarding bad positioned state space samples. Longer trajectories consisting of multiple transitions transport more effectively the information about the reward signal through the state space. Hence, the goal of this section is to investigate and analyse the impact of multistep lookahead formulation on RG algorithms, especially when using a GN approximation of the Hessian during optimisation. I start with extending the loss to realise a training algorithm that employs multistep lookahead. Next, I derive its differential map and the related critical point condition. Later, in Section 3.5.4, the influence of larger lookahead on critical points is captured empirically for a Policy Evaluation task. Alternations in behaviour regarding convergence and generalisation performance are described. Additionally, I explore the application of multistep methods for full Policy Iteration at the end of Section 3.5.5.

One obtains a multistep lookahead method by merging k repeated applications of the Bellman Operator T_π in a single operation. The major part of common multistep DRL approaches involve two particular constructions. The first contains the well-known TD(λ)-like formulation, in which infinite many powers of the Bellman Operator are combined with an exponential weighting. The second construction is a variation of the first, which only requires finite many steps into the future and can be combined with an arbitrary weighting. To ease the reading, I call this second construction in my work compound method and also introduce the more compact label TD(k). In the following, I introduce both constructions concisely such that I can tackle afterwards their corresponding critical point conditions.

As the first construction attempt for a multistep method, one sums together infinite many powers of T_π with exponential weighting to obtain the TD(λ) method. This is a common approach and already thoroughly investigated in textbooks, e.g., in [Bertsekas, 2012]. Yet, the results only apply to discrete state spaces and linear function approximation architectures such that a partial goal of my analysis is to bridge the gap towards continuous spaces and non-linear approximation techniques. Nevertheless, I also address discrete state space for the sake of completeness in Section 3.4.4. In [Schulman et al., 2016], a TD(λ) approach is used together with advantage functions to achieve a well-working DRL algorithm. Despite this particular realisation of a DRL algorithm being the main motivation for the investigation of multistep methods in my thesis, I still work directly with value functions. Since the advantage $A(s,a) = Q(s,a) - V(s)$ depends on Q and V, these two functions still need to be learned on their own, for example through minimising the NMSBE. Hence, I focus on V alone to avoid dealing with two optimisation tasks in my work. I denote the multistep Bellman Operator corresponding to TD(λ) by $\mathrm{T}_\pi^{(\lambda)} \colon \mathcal{V} \to \mathcal{V}$. It accepts some $\lambda \in (0,1)$ and takes the form

$$\mathrm{T}_\pi^{(\lambda)} \coloneqq (1-\lambda) \lim_{k\to\infty} \sum_{j=0}^{k} \lambda^j \, \mathrm{T}_\pi^{j+1} \,. \tag{3.20}$$

Of course, the operator $\mathrm{T}_\pi^{(\lambda)}$ always acts on a certain value function $V \in \mathcal{V}$ and cannot exist on its own. The expressions in the sum are simple n-step returns as used in [Mnih et al., 2016]. Thus, TD(λ) can be seen as a more sophisticated and generalised version of n-step returns. For $j = 1$, I have $(\mathrm{T}_\pi^1 V)(s) = (\mathrm{T}_\pi V)(s)$ for some $s \in \mathcal{S}$ when using the shorthand notation from Equations (2.3) and (2.7). Two- and three-step returns can be

written down as

$$(\mathrm{T}_\pi^2 V)(s) = \mathop{\mathbb{E}}_{s',s''}\left[r(s,\pi(s),s') + \gamma r(s',\pi(s'),s'') + \gamma^2 V(s'')\right]$$

and

$$(\mathrm{T}_\pi^3 V)(s) = \mathop{\mathbb{E}}_{s',s'',s'''}\left[r(s,\pi(s),s') + \gamma r(s',\pi(s'),s'') + \gamma^2 r(s'',\pi(s''),s''') + \gamma^3 V(s''')\right],$$

respectively. Higher powers of T_π are defined similarly. Due to the same reasons as for the one-step operator (cf. Section 3.3.2), expectations over successor states are rendered obsolete, since I only work with deterministic environments. The fixed point of $\mathrm{T}_\pi^{(\lambda)}$ is still V_π as defined in Equation (2.1) and yields the condition $V_\pi(s) = (\mathrm{T}_\pi^{(\lambda)} V_\pi)(s)$. Therefore, this operator also allows for a conversion of the fixed point iteration into a root finding problem by defining the difference

$$\delta^{(\lambda)}(s) \coloneqq V(s) - (\mathrm{T}_\pi^{(\lambda)} V)(s). \tag{3.21}$$

A key problem with this condition is that infinite many transitions to the successors s', s'', s''', …, are required for computation. Hence, I exploit the geometric series and terminate the sum early once λ^j is small enough to be ignored without causing larger approximation errors. The available machine precision could serve as a rough guidance. For example, if one uses $\lambda = 0.9$, cutting the infinite sum at $j = 500$ transitions would result in a pre-factors smaller than $0.9^{500} \approx 10^{-23}$ for later terms. Of course, this is not a perfect guarantee that the approximation error is small, because it still depends on the actual value of $(\mathrm{T}_\pi^j V)(s)$. But it provides a good heuristic. Hence, in practical applications, one still collects many successors, but at least just finite many. To do so, one brings the dynamical system under control into the desired current state s and retrieves its k successors s', s'' until $s^{(k)}$ by executing actions according to the current policy. Once a batch of N start states s_i, which are uniformly distributed in the entire state space, and their successors $s_i^{(j)}$ with $j = 1, \ldots, k$ is complete, one can approximate a solution to the root finding problem with an MLP by minimising the multistep Neural Mean Squared Bellman Error (mNMSBE)

$$\begin{aligned}\mathcal{J}^{(\lambda)}(\mathbf{W}) \coloneqq& \frac{1}{2N}\sum_{i=1}^{N}\left(f(\mathbf{W},s_i) - \sum_{j=0}^{\infty}\gamma^j r_{ij} - (1-\lambda)\,\gamma\sum_{j=0}^{\infty}\lambda^j\gamma^j f(\mathbf{W},s_i^{(j+1)})\right)^2 \\ =& \frac{1}{2N}\Delta_\pi^{(\lambda)}(\mathbf{W})^\mathsf{T}\Delta_\pi^{(\lambda)}(\mathbf{W})\end{aligned} \tag{3.22}$$

with any suitable method. The expression $\Delta_\pi^{(\lambda)}(\mathbf{W}) \in \mathbb{R}^N$ now takes the form

$$\Delta_\pi^{(\lambda)}(\mathbf{W}) \coloneqq F(\mathbf{W}) - R_\pi^{(\lambda)} - (1-\lambda)\,\gamma\sum_{j=0}^{\infty}\lambda^j\gamma^j F^{(j+1)}(\mathbf{W}) \tag{3.23}$$

by collecting for all i the evaluation of the MLP $f\colon \boldsymbol{\mathcal{W}} \times \mathcal{S} \to \mathbb{R}$ for the states at the j-th successor as a vector $F(\mathbf{W})^{(j)} \coloneqq [f(\mathbf{W},s_1^{(j)}) \cdots f(\mathbf{W},s_N^{(j)})]^\mathsf{T} \in \mathbb{R}^N$. Furthermore, I use the abbreviation $r_{ij} = r(s_i^{(j)}, \pi(s_i^{(j)}), s_i^{(j+1)})$ and group all accumulated and discounted reward terms in $R_\pi^{(\lambda)}$.

The second construction TD(k) uses only finite many powers by design and combines them as simple weighted average. Despite arbitrary weightings are possible, I only address uniform weightings for the sake of simplicity. Therefore, the corresponding multistep Bellman Operator $\mathrm{T}_\pi^{(k)} \colon \mathcal{V} \to \mathcal{V}$ uses at most k-steps into the future and is given by

$$\mathrm{T}_\pi^{(k)} \coloneqq \frac{1}{k} \sum_{j=1}^{k} \mathrm{T}_\pi^j \,. \tag{3.24}$$

As an example, the compound operator with three-step lookahead results in the expression $\mathrm{T}_\pi^{(3)} = 1/3 \left(\mathrm{T}_\pi^1 + \mathrm{T}_\pi^2 + \mathrm{T}_\pi^3\right)$, where individual terms are defined as for $\mathrm{T}_\pi^{(\lambda)}$. A more generic foundation for this type of multistep formulation, which contains the definition of $\mathrm{T}_\pi^{(k)}$ as special case, is already existing in [Bertsekas, 2012] and called Free-Form sampling. The main idea is to enhance the range of possible applications by being able to incorporate rollouts of different lengths in the implementation of multistep methods. As an example, assume that one has created a rollout consisting of three consecutive transitions. To form $\mathrm{T}_\pi^{(k)}$, one would use only three (sub) rollouts, namely one, two and three consecutive transitions, which all start from the first state. If one wishes to use the entire available data in a rollout of length three, then also the second two-step transition starting from the first successor and all three available one-step transitions would need to be included in the operator. The major benefit of Free-Form sampling would be an increased data efficiency for learning. However, I would face in my work additional restrictions regarding the applicability. Having the critical point condition in mind, I cannot easily add more starting states for the rollouts, as this would increase the amount of rows in $\widetilde{G}(\mathbf{W})$, or, more precisely, in its multistep counterpart. Thus, I do not use the possibilities offered by Free-Form sampling. The operator $\mathrm{T}_\pi^{(k)}$ has been also rediscovered in a more recent work [Yuan et al., 2019] as a seemingly novel approach to data efficient DRL.

Analogously to the TD(λ) setting, the operator $\mathrm{T}_\pi^{(k)}$ also possesses $V_\pi(s)$ as its unique fixed point and, thus, allows to write $V_\pi(s) = (\mathrm{T}_\pi^{(k)} V_\pi)(s)$. Hence, the conversion of the fixed point iteration to an optimisation task follows the same pattern as usual and results in

$$\delta^{(k)}(s) \coloneqq V(s) - (\mathrm{T}_\pi^{(k)} V)(s). \tag{3.25}$$

The main advantage of the compound method over the exponentially weighted one is that it is easier to obtain the required data from an environment. Approximations are not necessary, because only finite many successor s', s'', …, $s^{(k)}$ are involved by design. Next, I can define the compound version of the mNMSBE as

$$\begin{aligned} \mathcal{J}^{(k)}(\mathbf{W}) \coloneqq & \frac{1}{2N} \sum_{i=1}^{N} \left(f(\mathbf{W}, s_i) - \frac{1}{k} \sum_{j=0}^{k-1} (k-j) \gamma^j r_{ij} - \frac{1}{k} \sum_{j=1}^{k} \gamma^j f(\mathbf{W}, s_i^{(j)}) \right)^2 \\ = & \frac{1}{2N} \Delta_\pi^{(k)}(\mathbf{W})^\mathsf{T} \Delta_\pi^{(k)}(\mathbf{W}), \end{aligned} \tag{3.26}$$

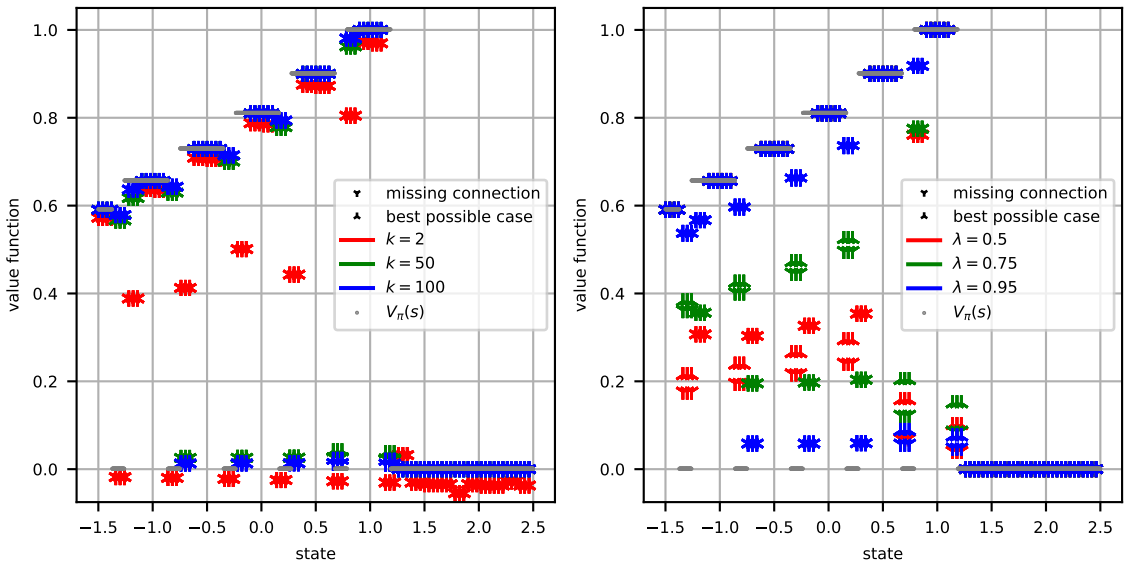
where the expression $\Delta_\pi^{(k)}(\mathbf{W}) \in \mathbb{R}^N$ takes the form

$$\Delta_\pi^{(k)}(\mathbf{W}) \coloneqq F(\mathbf{W}) - \frac{1}{k} R_\pi^{(k)} - \frac{1}{k} \sum_{j=1}^{k} \gamma^j F^{(j)}(\mathbf{W}). \tag{3.27}$$

Previous definitions still apply and all one-step rewards r_{ij} are grouped together in $R_\pi^{(k)}$.

Before addressing critical points of Equations (3.22) and (3.26), consider the following numerical example, which visualises how multistep formulations impact the learning progress, in particular, if the training data contains flaws. I employ the continuous one-dimensional dynamical system described in Section 2.6.2 with its reward function r_1 from Equation (2.35). A ground truth for V_π is easily available by performing rollouts from many starting states. Further details regarding the general method for computing ground truths via rollouts are given in Section 3.5.1. Alternatively, the computation of V_π could also exploit the tabular nature of a piecewise constant function and apply directly the Bellman Operators under policy π as it can be done in discrete MDPs. The policy is set to move one unit to the right in every state. It takes the form $\pi(s) = 1 \ \forall s \in \mathcal{S}$. For the function approximation architecture, I use as in Section 3.3.2 piecewise constant functions comprised of 32 equally spaced segments in the state space and train them by minimising the multistep version of the NMSBE. In total, there are 32 training samples placed equidistantly from left to right by hand in the state space such that they match the nature of the selected approximation architecture. The 12^{th} and 13^{th} sample in this set can be omitted during training with the goal to create an artificial gap and cut the connection between states on the left side of the gap and their successors on the right. The next step is to obtain through various versions of the multistep Bellman Operators approximations to V_π based on the remaining training samples. This means, the function approximation architecture will suffer from this carefully positioned gap. As baseline serves the training with plain one-step Bellman Operator T_π from before. The related experiment is shown in Figure 3.1 and demonstrates clearly the issue regarding missing connections. The multistep methods TD(k) and TD(λ) show how the problems regarding a missing connection in the state space can be alleviated. More precisely, I use $k \in \{2, 50, 100\}$ for the compound method TD(k). For TD(λ), the decay is chosen from $\lambda \in \{0.5, 0.75, 0.95\}$. Each decay is used with 1000 consecutive transitions with the goal to approximate the geometric series with minimal error. Figure 3.2 shows the outcome for the approximated value functions when using multistep operators.

One can see in Figures 3.2a and 3.2b that the scaling effect is not visible any more as it is the case for the one-step approach in Figure 3.1b. Despite the direct connection between sampled states being lost once the gap is included in training, the offset of all approximated value functions is compensated during training. A second important observation is that compound multistep methods with short lookahead ($k = 2$) result for all initialisations at a bad approximation independently of the training data used. The best possible case and the one with missing connections coincide and are offset from the ground truth V_π. For λ-weighted methods with a strong decay ($\lambda = 0.5$), the situation is worse. Both cases do not coincide any more and by including the gap in the training data the approximation becomes even slightly better. This bad behaviour vanishes for both cases when a longer lookahead k or a weak decay λ is employed. For TD(k), the approximated value functions get close to the ground truth and the connectivity issue is successfully compensated. For TD(λ), the same principal behaviour is also there. But the result is not as good as for the compound formulation. The fact that a compound method achieves in this simple example with a fraction of the state space transition ($k = 50$) a similar reduction in the approximation error as the TD(λ) method with $\lambda = 0.9$ and 1000 transitions, raises further the motivation for looking at those methods in more detail.



(a) TD(k) via $\mathrm{T}_\pi^{(k)}$ (b) TD(λ) via $\mathrm{T}_\pi^{(\lambda)}$

Figure 3.2: Multistep operators compensate missing connections between states. The best possible case for training is to use all available training samples. By omitting certain states, connection issues are made explicit during training. **a)** Compound methods for large k can compensate missing connections. Only for $k = 2$, an artefact shows up, where even the best possible case is not matching V_π. This could be explained with the fundamental scaling freedom of the NMSBE such that errors amplify. **b)** A similar behaviour applies to the TD(λ) approach. Yet, the outcomes are worse than for the compound formulation for all λ.

Hence, I extend the approach of Section 3.3.2 and investigate the critical point conditions for both objectives $\mathcal{J}^{(\lambda)}(\mathbf{W})$ and $\mathcal{J}^{(k)}(\mathbf{W})$ from Equations (3.22) and (3.26), respectively. Critical points of $\mathcal{J}^{(\lambda)}(\mathbf{W})$ are characterised by the equation

$$\nabla_{\mathbf{W}} \mathcal{J}^{(\lambda)}(\mathbf{W}) = \frac{1}{N} \underbrace{\Big(G(\mathbf{W}) - (1-\lambda)\,\gamma \sum_{j=0}^{\infty} \lambda^j \gamma^j G^{(j+1)}(\mathbf{W}) \Big)}_{=:\widetilde{G}^{(\lambda)}(\mathbf{W}) \in \mathbb{R}^{N \times N_{net}}}^{\mathsf{T}} \Delta_\pi^{(\lambda)}(\mathbf{W}) \stackrel{!}{=} 0, \tag{3.28}$$

where $G(\mathbf{W})$ is the differential map of F with respect to the parameters $\mathbf{W}$ evaluated at all start states. With $G^{(j)}(\mathbf{W})$, I denote the same quantity but use the j-th successor states as input. A similar expression follows for $\mathcal{J}^{(k)}(\mathbf{W})$. By setting its gradient to zero

$$\nabla_{\mathbf{W}} \mathcal{J}^{(k)}(\mathbf{W}) = \frac{1}{N} \underbrace{\Big(G(\mathbf{W}) - \frac{1}{k} \sum_{j=1}^{k} \gamma^j G^{(j)}(\mathbf{W}) \Big)}_{=:\widetilde{G}^{(k)}(\mathbf{W}) \in \mathbb{R}^{N \times N_{net}}}^{\mathsf{T}} \Delta_\pi^{(k)}(\mathbf{W}) \stackrel{!}{=} 0 \tag{3.29}$$

one obtains the condition for critical points. In both conditions, one can see that the matrices, i.e., $\widetilde{G}^{(\lambda)}(\mathbf{W})$ in Equation (3.28) and $\widetilde{G}^{(k)}(\mathbf{W})$ in Equation (3.29), follow the

structure of the matrix in Equation (3.16) as it appeared in Section 3.3.2. Due to the usage of multiple transitions, there are just more terms being combined to yield $\widetilde{G}^{(\lambda)}(\mathbf{W})$ and $\widetilde{G}^{(k)}(\mathbf{W})$. This means, that the statements regarding the rank of those matrices stay the same as those given around Equation (3.15). I still want to have a rank as large as possible for each $G^{(j)}(\mathbf{W})$. Therefore, one needs to provide a large enough MLP in terms of parameters such that $N_{net} > N$ becomes possible and stick to the already mentioned design choices. Based on Equation (3.15), I would expect that longer lookahead exhibit a positive influence on the rank of either $\widetilde{G}^{(\lambda)}(\mathbf{W})$ or $\widetilde{G}^{(k)}(\mathbf{W})$. In the best case, the rank becomes easily full when involving multiple transitions such that requirements for Proposition 4 and Remark 1 are more likely to be satisfied. But this can only be answered empirically. Numerical investigations for the rank are provided in Section 3.4 for the exact learning case in discrete state spaces. A more sophisticated analysis with experiments for the sampling setting in continuous spaces is part of Section 3.5.

3.4 A Gauss Newton Residual Gradient Algorithm

In this section, I provide details regarding my concrete algorithm and make use of the results from the previous analysis. The focus lies on aspects that are relevant on their own, i.e., decoupled from any experiment, but fit no longer into the previous section as they are only due to the implementation in a computer system. I address the convergence in general and provide pseudo code as it is used in the experiments. Lastly, I verify all components by demonstrating local quadratic convergence of a Gauss Newton algorithm for the discrete and exact learning setting.

3.4.1 Convergence of the Proposed Algorithm

There exists rich literature covering the convergence of Gauss Newton algorithms. More specifically, the Levenberg-Marquardt approach, where a descent direction $\eta \in \boldsymbol{\mathcal{W}}$ is the solution to a regularised linear equation system, provides a framework for articulating the convergence properties of my proposed method. Using the gradient from Equation (3.13) directly and the approximated Hessian from Equation (3.18) combined with an identity matrix times a small scalar factor, I solve

$$\left(\mathsf{H}_{\mathbf{W}}\,\widetilde{\mathcal{J}}(\mathbf{W}) + c\; I_{N_{net}}\right)\eta = \nabla_{\mathbf{W}}\widetilde{\mathcal{J}}(\mathbf{W}) \tag{3.30}$$

for η, where $c > 0$ controls the strength of regularisation. Theorem 4 in [Behling et al., 2019] ensures that the series of distances of iterates to the set of critical points forms a Cauchy sequence if certain assumptions hold. Firstly, in a neighbourhood around critical points, the theorem requires $G(\mathbf{W})$ or $\widetilde{G}(\mathbf{W})$ to be Lipschitz continuous. Secondly, the gradient needs to provide a local bound on the distance of iterates to the set of critical points. Thirdly, the linearisation error has to be small enough. The MLP architectures I am working with satisfy these assumptions. Since I have no full control over the rank of $G(\mathbf{W})$ or $\widetilde{G}(\mathbf{W})$ during the entire optimisation process, I assume that a loss of rank could happen and set the regularisation to $c = 10^{-5}$. All experiments use this value if not otherwise stated. Once the iterates are close enough to critical points, then Propositions 1 and 4, and their related considerations, ensure a proper outcome. Unfortunately, it is hardly possible to calculate for arbitrary control problems a convergence radius beforehand.

Hence, I select the initial parameters for the optimisation process randomly, which may result in additional iterations until one hits the basin of attraction. To remove the risk for harmful initial steps in the parameter space, I combine the descent direction η with a learning rate $\alpha \in (0, 1)$.

3.4.2 The Algorithm

In my numerical analysis, I rely on two common DP procedures, namely Approximate Policy Evaluation and Approximate Policy Improvement. The approximate nature of those algorithms arises from the usage of MLPs as function class. Both algorithms together provide the Approximate Policy Iteration procedure. It can be used to compute sequentially improving policies until the point, where approximation errors prevent any further progress.

The actual realisation of Policy Evaluation depends on the setting at hand. For the exact learning scenario with discrete state spaces, I need to use equations from Section 3.3.1 to arrive at Algorithm 1. The input is the MLP used for representing a value function and the MDP components themselves. The result is an approximation of V_π for all states.

Algorithm 1 Policy Evaluation with Discrete States and Gauss Newton Optimisation

Hyper parameters: $\gamma \in (0, 1)$, $\alpha > 0$, $c = 10^{-5}$, $\epsilon \leq 10^{-25}$, $N_{net} \sim K_{\mathcal{S}}$
Input:
- MLP $f \in \mathcal{F}(1, n_1, \ldots, n_{L-1}, 1)$ with initialised parameters $\mathbf{W} \in \boldsymbol{\mathcal{W}}$
- transition probabilities P_π and rewards R_π

Output: $\mathbf{W}$ such that $f(\mathbf{W}, s) \approx V_\pi(s) \quad s = 1, \ldots, K_{\mathcal{S}}$

1: **do**
2: Evaluate $F(\mathbf{W}) \coloneqq [f(\mathbf{W}, 1) \ldots f(\mathbf{W}, K_{\mathcal{S}})]^\mathsf{T} \in \mathbb{R}^{K_{\mathcal{S}}}$ and its differential map $G(\mathbf{W})$
3: Bellman Residual: $\Delta_\pi(\mathbf{W}) = F(\mathbf{W}) - P_\pi \left(R_\pi + \gamma F(\mathbf{W})\right)$
4: NMSBE: $\mathcal{J}(\mathbf{W}) = \frac{1}{2}\Delta_\pi(\mathbf{W})^\mathsf{T} \Xi \Delta_\pi(\mathbf{W})$
5: Gradient: $\nabla_{\mathbf{W}} \mathcal{J}(\mathbf{W}) = G(\mathbf{W})^\mathsf{T} \Big(I_{K_{\mathcal{S}}} - \gamma P_\pi\Big)^\mathsf{T} \Xi \Delta_\pi(\mathbf{W})$
6: Hessian: $\mathsf{H}_{\mathbf{W}}\, \mathcal{J}(\mathbf{W}) = G(\mathbf{W})^\mathsf{T} \Big(I_{K_{\mathcal{S}}} - \gamma P_\pi\Big)^\mathsf{T} \Xi \Big(I_{K_{\mathcal{S}}} - \gamma P_\pi\Big) G(\mathbf{W})$
7: Solve for η: $\Big(\mathsf{H}_{\mathbf{W}}\, \mathcal{J}(\mathbf{W}) + c I_{N_{net}}\Big)\eta = \nabla_{\mathbf{W}} \mathcal{J}(\mathbf{W})$
(e.g. with Householder QR-Decomposition)
8: Descent step: $\mathbf{W} \leftarrow \mathbf{W} - \alpha\eta$
9: **while** $\mathcal{J}(\mathbf{W}) > \epsilon$

For continuous state spaces and the sampling setting, Policy Evaluation takes the form as shown in Algorithm 2. It makes use of the machinery in Section 3.3.2 to realise a Gauss Newton Residual Gradient algorithm for approximating V_π with a given MLP. Different than before, the required inputs no longer expect the complete MDP description. Only a list of N sample states needs to be provided. Additionally, the policy must be specified explicitly, since it is no longer contained as part of the transition probabilities. The

output of the algorithm consists now of MLP parameters, which correspond to a good approximation in the context of Definition 1.

Algorithm 2 Policy Evaluation with Gauss Newton Residual Gradient Formulation

Hyper parameters: $\gamma \in (0,1)$, $\alpha > 0$, $c = 10^{-5}$, $\epsilon \leq 10^{-5}$, $N \sim N_{net}$
Input:
- MLP $f \in \mathcal{F}(K_{\mathcal{S}}, \dots, 1)$ with initialised parameters $\mathbf{W} \in \boldsymbol{\mathcal{W}}$
- policy $\pi \colon \mathcal{S} \to \mathcal{A}$
- unique sample states $s_i \in \mathcal{S}$ with $i = 1, \dots, N$

Output: $\mathbf{W}$ such that $f(\mathbf{W}, s_i) \approx V_\pi(s_i) \quad \forall i = 1, \dots, N$

1: Construct transition tuples (s_i, r_i, s_i') using π for all i
2: **do**
3: Evaluate $F(\mathbf{W}) \coloneqq [f(\mathbf{W}, s_1) \dots f(\mathbf{W}, s_N)]^\mathsf{T} \in \mathbb{R}^N$ and its differential map $G(\mathbf{W})$
4: Evaluate F' and G' using s_i'
5: Bellman Residual: $\widetilde{\Delta}_\pi(\mathbf{W}) = F(\mathbf{W}) - R_\pi - \gamma F'(\mathbf{W})$
6: NMSBE: $\widetilde{\mathcal{J}}(\mathbf{W}) = \frac{1}{2N} \widetilde{\Delta}_\pi(\mathbf{W})^\mathsf{T} \widetilde{\Delta}_\pi(\mathbf{W})$
7: Gradient: $\nabla_{\mathbf{W}} \widetilde{\mathcal{J}}(\mathbf{W}) = \frac{1}{N} \Big(G(\mathbf{W}) - \gamma G'(\mathbf{W}) \Big)^\mathsf{T} \widetilde{\Delta}_\pi(\mathbf{W})$
8: Hessian: $\mathsf{H}_{\mathbf{W}} \widetilde{\mathcal{J}}(\mathbf{W}) = \frac{1}{N} \Big(G(\mathbf{W}) - \gamma G'(\mathbf{W}) \Big)^\mathsf{T} \Big(G(\mathbf{W}) - \gamma G'(\mathbf{W}) \Big)$
9: Solve for η: $\Big(\mathsf{H}_{\mathbf{W}} \widetilde{\mathcal{J}}(\mathbf{W}) + c I_{N_{net}} \Big) \eta = \nabla_{\mathbf{W}} \widetilde{\mathcal{J}}(\mathbf{W})$
(e.g. with Householder QR-Decomposition)
10: Descent step: $\mathbf{W} \leftarrow \mathbf{W} - \alpha \eta$
11: **while** $\widetilde{\mathcal{J}}(\mathbf{W}) > \epsilon$

To construct a Policy Iteration framework, a new procedure is established, which uses Policy Evaluation in one of the two forms described before and extends it with a Policy Improvement step. Since my experiments only require Policy Iteration for continuous state spaces, there is no dedicated description for a Policy Iteration algorithm in the discrete space setting. However, the required modifications are straightforward and should not allow for any ambiguity. The important new component is Policy Improvement, which is responsible for obtaining a GIP according to Equation (2.15). Despite the fact that Policy Evaluation works equivalently for state-only value functions and Q-factors, the necessity for solving Equation (2.15) renders Q-factors mandatory to use. For discrete action spaces, the related argmax$(\cdot)$ operation is trivial to compute and can be done by brute force search. This approach is suitable for the PI experiments in Section 3.5.5. For continuous action spaces, the computation of a GIP gets more demanding. One can either stick to direct search methods in action space for computing the action corresponding to the largest Q-factor, or one employs parametrised policies to determine the best action in a state. Since continuous action spaces bring additional challenges on their own, this topic is postponed until Chapter 4. After every pair of evaluation and improvement, which is

called sweep in the following, a better performing policy should be available. The complete Policy Iteration algorithm is depicted in Algorithm 3.

Algorithm 3 Policy Iteration for Continuous State Spaces

Hyper parameters: $N \sim N_{net}$, $sweeps > 0$
Input: MLP architecture $f \in \mathcal{F}(K_{\mathcal{S}}, \dots, 1)$ with parameter space $\boldsymbol{\mathcal{W}}$
Output: policy $\pi \colon \mathcal{S} \to \mathcal{A}$

1: Draw $\mathbf{W}$ element-wise uniformly from $[-1, 1]$
2: $\pi(s) \leftarrow \mathrm{GIP}(f, \mathbf{W}, s) \quad \forall s \in \mathcal{S}$
3: Draw s_i for $i = 1, \dots, N$ uniformly from $\mathcal{S}$

4: **for** sweep **in** sweeps **do**
5: **if not** persistent parameter **then**
6: Draw $\mathbf{W}$ elementwise uniformly from $[-1, 1]$
7: **end if**
8: **if not** persistent data **then**
9: Draw s_i for $i = 1, \dots, N$ uniformly from $\mathcal{S}$
10: **end if**

11: $\mathbf{W} \leftarrow \mathrm{PolicyEvaluation}(f, \mathbf{W}, \pi, s_1, \dots, s_N)$
12: $\pi(s) \leftarrow \mathrm{GIP}(f, \mathbf{W}, s) \quad \forall s \in \mathcal{S}$

13: Evaluate π empirically using several rollouts
14: **end for**

A common practice in Policy Iteration is to reuse the MLP parameters of the most recent Policy Evaluation step as initialisation for the next sweep. Although it is theoretically possible to start in every sweep with a freshly initialised MLP, in real world applications one observes that this adjustment stabilises Policy Iteration a lot. In [Sigaud and Stulp, 2019], this practice is called persistent, whereas running the evaluation from scratch in every sweep is referred to as transient. I employ this naming convention here as well and have equipped the Policy Iteration procedure in Algorithm 3 with an optional use of persistent parameters in Lines 5 to 7. Additionally, I extent this pattern to the sampling of states. Lines 8 to 10 provide the mechanism to resample data in every sweep.

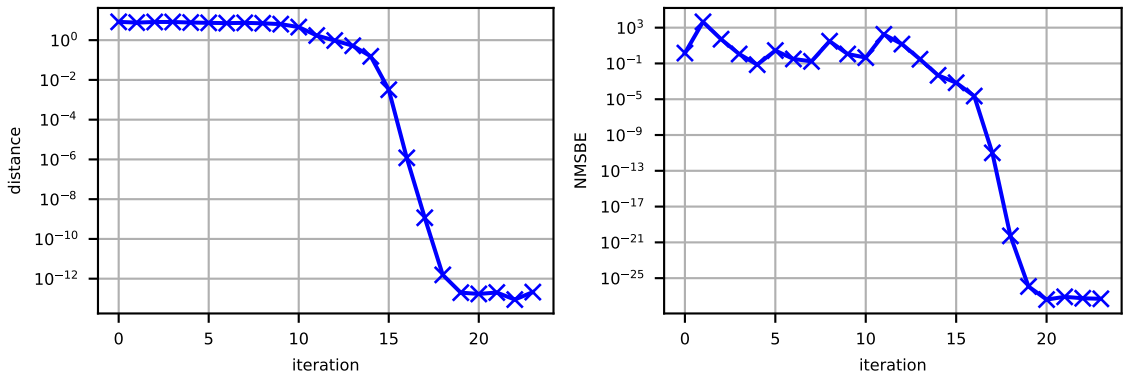
Line 13 performs technically the same operation as Policy Evaluation, but due to the use of independent Monte Carlo rollouts, it is possible to obtain reliable and unbiased performance metrics for the policy. In infinite horizon problems, this is only possible by introducing an early termination of the rollouts. The idea and concept is similar to that of the TD(λ) method and its finite many terms contributing to the geometric series.

Let τ denote a rollout, which may also be called a trajectory, of length $L > 0$. Each τ contains L transitions tuples s_i, a_i, s_{i+1} with $i = 1, \dots, L$ and actions being computed by the policy $a_i = \pi(s_i)$. The first state s_0 is the starting point in the state space and is a free parameter. The discounted return of a trajectory is defined as $R(\tau) = \sum_{i=1}^{L} \gamma^i r(s_i, a_i, s_{i+1})$, which is related but not identical to V_π from Equation (2.1). Firstly, the expectation is missing meaning that τ represents one particular sample from the environment if noise is

present. By combining for a certain start state s_0 the discounted returns of many trajectories as empirical mean, it is possible to get an approximation for the expectation. By the law of large numbers, the approximation error becomes small if the number of trajectories used becomes large. Due to my special case of only considering deterministic environments, even a single trajectory per given start state should be enough. However, machine precision and the resulting numerical errors in the computation still make it necessary to use an average over a couple of trajectories to obtain a more stable performance value of a policy when starting in s_0. Secondly, the discounted return is a mere finite horizon approximation for expected discounted reward. However, due to the required discount factor γ to render the series bounded, there does not arise a too large approximation error, if the length of trajectories is chosen sufficiently long. The required length depends on the environment at hand and the selected discount factor. But also the overall runtime has to be taken into account and might pose restrictions on the largest length possible.

To be able to provide performance curves, which visualise intuitively the learning progress of the algorithm across several sweeps, another processing step is required. Let j denote a sweep and let there be M sweeps in total, i.e., $j = 1, \ldots, M$. Per sweep, there will be N trajectories starting from N different states. They receive the index $k = 1, \ldots, N$. Thus, I now use the term τ_{jk} to refer to a specific trajectory and τ to refer to the set of all trajectories. Consequently, their discounted returns can be interpreted as a matrix $R(\tau) \in \mathbb{R}^{M \times N}$. For visualisation, $R(\tau)$ is now combined in three different ways. The first way is the mean discounted return for all sweeps $\bar{R} = \frac{1}{N} \sum_k R(\tau_{jk}) \in \mathbb{R}^{M \times 1}$. The second one is the minimal discounted return $R^- = \min_k R(\tau_{jk}) \in \mathbb{R}^{M \times 1}$ and the third one is the maximal return $R^+ = \max_k R(\tau_{jk}) \in \mathbb{R}^{M \times 1}$. All three quantities have one value per sweep and can be used to reveal the typical learning progress as well as the best and worst case. If the enveloping curve around $\bar{R}$ is narrow, then this indicates that all rollouts are consistent. Otherwise, a more careful investigation without aggregation of the performance is required.

A frequent concern regarding second-order algorithms is the computational effort involved in models with many parameters. However, typical architectures of MLPs used in DP applications contain around two hidden layers with approximately 50 units in each layer as upper limit on the MLP size. For such MLPs, N_{net} falls in the range of 2000 parameters, for which second-order optimisation is manageable with reasonable time and storage as I demonstrate with my experiments. A second concern is raised about the root seeking behaviour of a Newton's method. By using second-order information in a gradient descent optimisation procedure, one aims directly at any root in the gradient vector field of the NMSBE. Thus, the optimiser would also be attracted by (local) maxima or even saddle points. However, due to the nature of a non-linear least squares problem and as I have already elaborated after Propositions 2 and 5, these types of extrema do not exist if all assumptions are satisfied. Additionally, saddles and maxima are numerically unstable such that approximation errors help to overcome those extrema. A third concern, which people may rise, is to just use an automatic differentiation framework for all kind of derivatives and gradients instead of the tedious manual work. In my implementation, I do not make use of them due to the following reasons. First, such frameworks are not able to introduce approximations to symbolic expressions on their own. As I already have the required differential maps available due to my theoretical investigation, I could realise the optimisation procedure by hand without too much overhead required. Second, to the best of my knowledge, the operations involved in the Hessian are not suited for general purpose



(a) (b)

Figure 3.3: A typical local quadratic convergence behaviour observed for an adapted version of Baird's *Seven State Star Problem*. Within only a couple of descent steps, the error or distance decreases down to machine precision. **a)**: Distance of iterates $\mathbf{W}^{(k)}$ to the accumulation point $\mathbf{W}^*$. **b)**: The NMSBE corresponding to each iterate $\mathbf{W}^{(k)}$.

graphics processing units, hence I expect the performance gain of automatic differentiation frameworks to be minimal. Third, as I have to solve linear equation systems in every descent step, I cannot avoid to use other software libraries as well. This would add additional overhead in the data transfer between computational devices and the main memory such that benefits of these frameworks diminish further. In conclusion, by implementing the algorithm manually, I achieve a full control over all components and also could make use of sophisticated parallelisation in all of them. Of course, recent developments and contributions to automatic differentiation frameworks can render these considerations obsolete.

3.4.3 Demonstration of Local Quadratic Convergence

To evaluate empirically my derived theoretical results in the discrete state space setting together with the assumption of exact learning, I demonstrate local quadratic convergence on the adapted *Star Problem* (cf. Section 2.6.1). Only when all components are working as intended, local quadratic convergence of a second-order gradient descent algorithm can be visible.

I deploy an MLP architecture $\mathcal{F}(2, 7, 1)$ consisting of $N_{net} = 29$ parameters with step size $\alpha = 1$ and use Bent-Id as activation function in hidden layers. Every state receives a unique two-dimensional random feature to embed the discrete states in a vector space for the input layer. Features are drawn from a normal distribution with zero mean and unit covariance.

In Figure 3.3, I visualise the convergence behaviour through the distance of the accumulation point $\mathbf{W}^*$ to all iterates $\mathbf{W}^{(k)}$ as well as by the corresponding NMSBE. I use the last iterate as $\mathbf{W}^*$ and measure the distance by extending the Frobenius norm of matrices to collections of matrices as $\|\mathbf{W}^{(k)} - \mathbf{W}^*\|_F^2 := \sum_{l=1}^{L} \|W_l^{(k)} - W_l^*\|_F^2$.

It is clear from Figure 3.3a that the Gauss Newton algorithm converges locally quadratically fast to a critical point. Judging from the negligibly small final NMSBE as seen in

Figure 3.3b, I conclude that the MLP is an exact approximator of the ground truth value function. Both graphs together imply that the approximated Hessian is exact close to critical points $\mathbf{W}^*$. Due to over-parametrisation, convergence to suboptimal local minima does not happen.

3.4.4 Tracking the Rank of the Jacobian During Optimisation

Aside from a visual inspection of the convergence rate, as I have provided it in the previous section, it is desirable to quantify empirically the convergence behaviour across many repetitions of the entire training process. To get a reliable indicator, which reveals whether the optimisation is really working properly, one needs to measure how the rank of the matrix $G(\mathbf{W})$, as it is defined in Equation (3.6), behaves. Of particular interest is how the rank for freshly initialised MLPs look like and how they behave during training. In the best case, one would observe that initial rank deficient Jacobians recover over time and that no loss of the rank occurs during training.

I employ the typical computation of the rank based on singular values of a matrix. The rank is equal to the amount singular values σ_i, which exceed a predefined threshold[2] $\max_i \{\sigma_i\} \cdot \max\{M, N\} \cdot \epsilon$. The shape of this matrix is given by M and N. The machine precision ϵ is set by the data type.

However, depending on the matrix, this method of defining the rank might be too coarse. Hence, I also make use of the singular values directly and use the inverse condition number

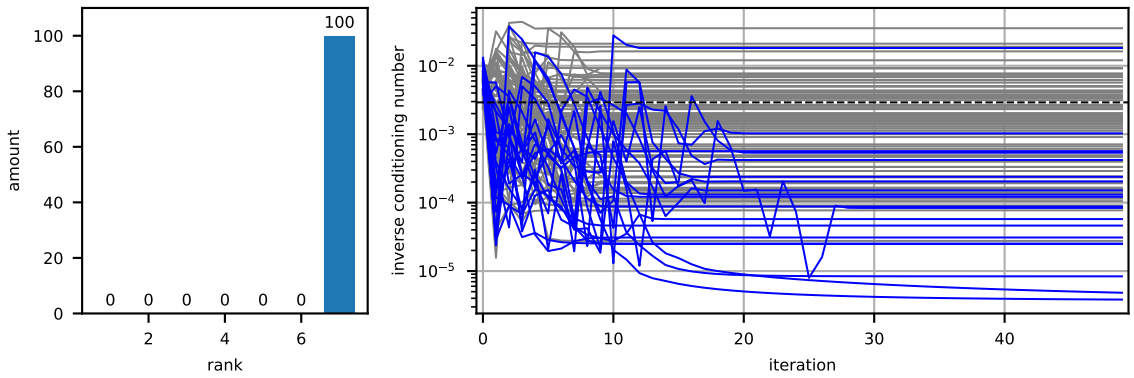
$$\kappa = \frac{\sigma_{min}}{\sigma_{max}} \tag{3.31}$$

to characterise the rank of $G(\mathbf{W})$. Due to the ordering of singular values, the inverse condition number satisfies $0 \leq \kappa \leq 1$. For $\kappa \to 1$, all singular values are close together and the matrix has a full rank (or it is the zero matrix, assuming that the ratio of zeros behaves correctly). If $\kappa \to 0$, the magnitude between the smallest and largest σ is far apart and one needs to treat the matrix as rank deficient. Since singular values of smoothly varying matrices also vary smoothly, it is also possible to track κ during training. Thereby, it is possible to see, whether the optimisation algorithm can recover from situations with bad conditioned initial $G(\mathbf{W})$.

Next, I repeat the experiment from the previous section, namely training an MLP to approximate V_π for the adapted *Star Problem*, for 100 times. The random features for embedding discrete states in a two dimensional space are kept constant and are identical for all repetitions. Every repetition starts with a randomly initialised MLP. In Figure 3.4a, the rank of $G(\mathbf{W})$ without any training happening is presented as a histogram. Figure 3.4b contains the inverse condition numbers for all repetitions during training. For the sake of visualisation, I show the major part of repetitions in as grey lines and draw only a selected subset in another colour. In all figures, the curves drawn in blue belong to those repetitions, where κ requires the most iterations to reach its final value. In this way, it is possible to see the overall long-term behaviour of all repetitions via the grey background as well as the evolution of a few individual runs by the blue foreground.

As expected, Figure 3.4a demonstrates that the rank itself is always full and does not provide a reliable and detailed insight into the nature of $G(\mathbf{W})$. For all executed

[2]This is the common approach in numerical computing libraries such as *NumPy* or the corresponding literature



(a) Direct computation of rank.

(b) The inverse condition number of $G(\mathbf{W})$.

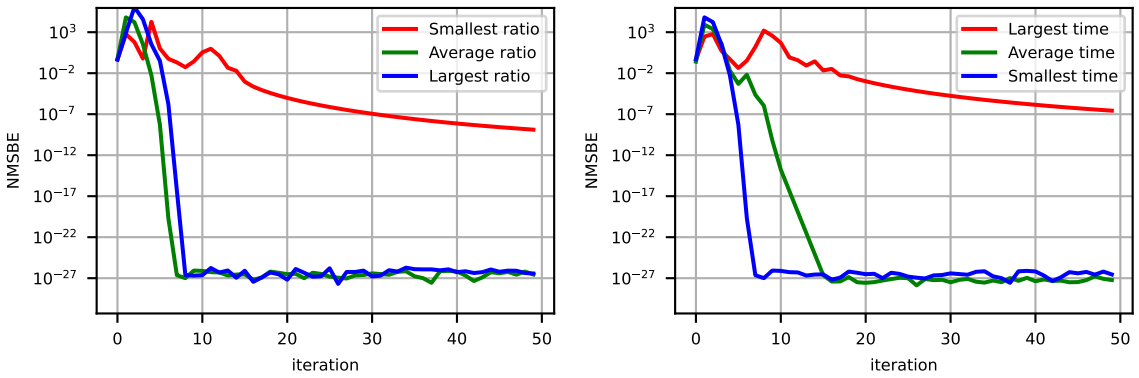
Figure 3.4: An investigation of the matrix $G(\mathbf{W})$ using an adapted version of Baird's *Seven State Star Problem* for 100 repetitions. **a)**: The rank of $G(\mathbf{W})$ at the beginning of training. For all randomly initialised MLPs the rank is full. **b)**: The inverse condition number κ of $G(\mathbf{W})$ during training. In blue, the 20 runs with the longest time until κ settles at its final value are emphasized. The horizontal dashed line represents the average value at the end of training.

optimisation procedures, every singular value of the Jacobian exceeds the predefined threshold. In Figure 3.4b, I can see several important properties. First, at the beginning of training, all MLPs start with a ratio above 10^{-3}, thereby supporting coarse statements about the rank from Figure 3.4a. Second, after some iterations, every training process stabilises and the ratio settles at a certain level. There are no strong jumps for κ towards zero, which could lead to a failure of optimisation, because the critical point condition looses its pleasing properties. Third, all ratios stay in a sound range of approximately 10^{-6} to 10^{-2}. Since I am working with double precision, the inverse condition numbers are far away from regions, where the ratio is close to machine accuracy. Severe issues with $G(\mathbf{W})$ and its rank do not arise such that the optimisation process has to work as desired. The worst effect should be the loss of local quadratic convergence for the runs with smallest final κ.

Hence, since κ appears as a reliable indicator for the rank of $G(\mathbf{W})$, it is reasonable to wonder, whether it can be used to classify the convergence characteristics of the GN algorithm. As a first attempt, I take the final κ of a run and use it as a score. I select three different repetitions out of the 100. The first has the largest ratio at the end of training, the second is on average and the third achieves the smallest κ. Their NMSBE over time is in depicted in Figure 3.5a.

As one can see, the final κ can be used to some extent as a classifier for the convergence pattern. However, the largest κ and average value reports runs with approximately the same convergence behaviour. Only the smallest ratio serves as a proper indicator for bad optimisation behaviour and reveal a missing local quadratic convergence.

To obtain a better indicator, I select another three repetitions out of the 100, but this time they are selected based on the required number of iterations until κ settles at its final value. The first run requires the largest amount of iterations until κ reaches its final value. It belongs to the lowest lines in Figure 3.4b and achieves a final NMSBE of only $2.7 \cdot 10^{-7}$ and $\kappa = 4.8 \cdot 10^{-6}$. The second is selected based on the average time to settle. It has



(a) Repetitions are selected based on the ratio κ at the end of training. The better κ is, the more likely one can observe local quadratic convergence. Only for very small values, the convergence is hindered so strongly that final error is significantly larger.

(b) The time required for the inverse condition number to arrive at its final value is used for slection. It provides a better classifier for the behaviour. The quicker κ settles, the more likely one can observe local quadratic convergence. Only for longest times to settle, convergence is hindered so strongly that final error is significantly larger.

Figure 3.5: Error over time for the adapted *Star Problem*, when repetitions are selected based on the inverse condition number. The convergence behaviour reflects the value of κ.

the final error $5.9 \cdot 10^{-28}$ and ratio $\kappa = 1.1 \cdot 10^{-4}$. The third run is the fastest to settle and is located at the top. Its final error is $2.8 \cdot 10^{-27}$ and the inverse condition number is $\kappa = 2.1 \cdot 10^{-2}$. The error over time for the three repetitions are shown in Figure 3.5b.

It can be seen that for a run with very long settling time there is no local quadratic convergence. This matches the previous attempt, however, another repetition out the 100 is reported. As before, the error at the end of training is much larger than for other repetitions. Different than before, it is now possible to spot different convergence speeds for small and average times to settles. In Figure 3.5, this corresponds to the runs visualised with green and blue graphs. The final error approaches smallest value observed so far at the end of training, but the speed of convergence correlates properly with the indicator. The best case to happen is that of blue lines in Figures 3.5a and 3.5b. Just a couple of descent steps result in the final error.

Overall, the phenomenon of bad initialisation with low performance occurs very rarely in the 100 repetitions of the experiment. When investigating again Figure 3.4b, I see that the run with average settling time in Figure 3.5b belongs to a ratio located at the lower middle spectrum. In other words, all other repetitions have better values for κ and, thus, should convergence reliably with stepwise more pronounced local quadratic convergence.

So far, all tests with the adapted Baird's *Star Problem* converge reliably with single-step lookahead. Hence, one cannot expect to see a significant performance boost onto the final error, if multistep transition data would be incorporated. However, there might be a visible impact on the rank of the matrix and, in turn, on the convergence behaviour. Hence, I employ a multistep formulation in the exact learning setting for Baird's *Seven State Star*

Problem and run the experiments again. Section 3.5.4 then only deals with continuous spaces, since this is the primary application area of multistep methods.

Fortunately, the realisation of multistep methods for small discrete state space tasks is straightforward. By combining the definitions of $\mathrm{T}_\pi^{(k)}$ and $\mathrm{T}_\pi^{(\lambda)}$ from Equations (3.20) and (3.24) with the matrix form of the operators similar to Equation (3.1), one finds that the same procedure as in Algorithm 1 can be used, if different expressions for P_π and R_π are provided. More specifically, to obtain the compound method for exact learning with discrete state spaces, I define the new multistep transition matrix

$$P_\pi^{(k)} = \frac{1}{k} \sum_{j=0}^{k-1} \gamma^j P_\pi^{j+1}$$

and the aggregated reward vector

$$R_\pi^{(k)} = \frac{1}{k} \sum_{j=0}^{k-1} \gamma^j P_\pi^j R_\pi.$$

For the exponentially weighted case, the two terms take a similar form. The transition probabilities are given by

$$P_\pi^{(\lambda)} = (1-\lambda) \sum_{j}^{\infty} \gamma^j \lambda^j P_\pi^{j+1}.$$

The corresponding rewards can be computed as

$$R_\pi^{(\lambda)} = \sum_{j=0}^{\infty} \gamma^j \lambda^j P_\pi^j R_\pi.$$

I exploit in these definitions that the reward in the *Star Problem* depends only on the current state. Next, I write for a discrete state space the multistep Bellman Operators for some value function $V \in \mathbb{R}^K$ compactly as

$$(\mathrm{T}_\pi^{(k)} V) = P_\pi^{(k)} \left(R_\pi^{(k)} + \gamma V \right)$$

for the compound method and as

$$(\mathrm{T}_\pi^{(\lambda)} V) = P_\pi^{(\lambda)} \left(R_\pi^{(\lambda)} + \gamma V \right)$$

for the λ-weighted formulation. An immediate consequence is that multiple transitions only affect the critical point condition indirectly. Since the new optimisation task can be expressed in terms of alternated P and R matrices, the critical point condition still matches Equation (3.5). There are only slightly different terms, which still share an identical behaviour. This also implies that Propositions 1 and 2 works as before and that the definition and computation of $G(\mathbf{W})$ itself stays the same. Different for a multistep approach is that the counterpart of Equation (3.1) now conveys more information. Thus, an MLP might achieve zero NMSBE earlier or adjust its parameters more quickly. This will then influence the Jacobian $G(\mathbf{W})$.

To capture the impact of multistep method empirically, I repeat the previous experiment, but make use of the compound Bellman Operator $\mathrm{T}_\pi^{(k)}$ and exponentially weighted version

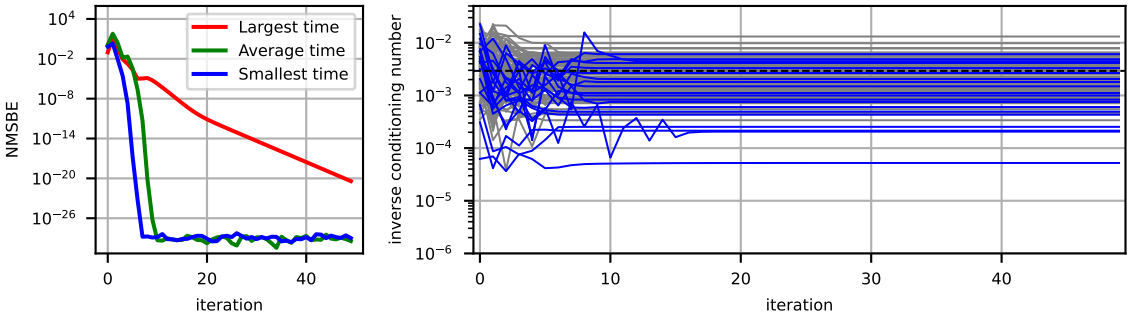
$\mathrm{T}_\pi^{(\lambda)}$. For the compound method, I vary the amount of steps k into the future. More specifically, I test $k \in \{2, 3, \ldots, 10\}$. The value $k = 1$ is omitted, because this is exactly the single-step approach from before. For the TD(λ)-like method, I adjust the decay λ and select it from $\lambda \in \{0.1, 0.5, 0.9, 0.95\}$. The infinite series inside of the operator is approximated with trajectories consisting of 150 consecutive transitions. The error, which is caused by terminating the series after finite many terms, is for all values of λ negligible, if these many transitions are used. I compute the network Jacobians for all configurations after each descent step and collect their inverse condition numbers κ. Similar to before, they are plotted across iterations to reveal the type of descent behaviour. Additionally, the errors over time for repetitions corresponding to a small, medium and large time to settle of κ are provided again. Figure 3.6 depicts the experiments for a subset of the aforementioned values. Omitted parameters share the same characteristics of other parameters and do not reveal more details.

The convergence plots in the left column of Figure 3.6 confirm the hypothesis that multistep methods cannot result in a significantly better descent behaviour. Both the smallest and average time to settle remain unchanged even if multistep methods are employed. The baseline is provided by single-step approaches, which manages to get a final error around 10^{-27} as it can be seen in Figure 3.5b. The exponentially weighted TD(λ) formulation achieves only the same final error, despite relying on multistep transition data with large horizon. On the contrary, a compound method with $k = 10$ steps into future can gain three orders of magnitude and achieve a better result in terms of final NMSBE. Of course, it is questionable whether this achievement is of any use. From an engineer's perspective, both approximated value functions will be indistinguishable from each other.

More important are the changes in the spectrum of κ values during and at the end of training. Figures 3.6b, 3.6d and 3.6f demonstrate that multistep formulations can have a positive effect on optimisation and, furthermore, that a compound Bellman Operator is better than the exponentially weighted variant. The inverse condition numbers at the end of training for $k \geq 3$ have less spread than all λ-weighted versions. Furthermore, the ratios are located consistently at larger values for all k. In particular, for $k = 10$, the ratio stays above 10^{-4}, thereby indicating clearly that the rank of $G(\mathbf{W})$ has to be considered full. More importantly, when comparing against the values for the single-step case in Figure 3.4b, one sees that ratios have improved. In conclusion, the foundation of the theory from Section 3.3.1 is strengthened further.

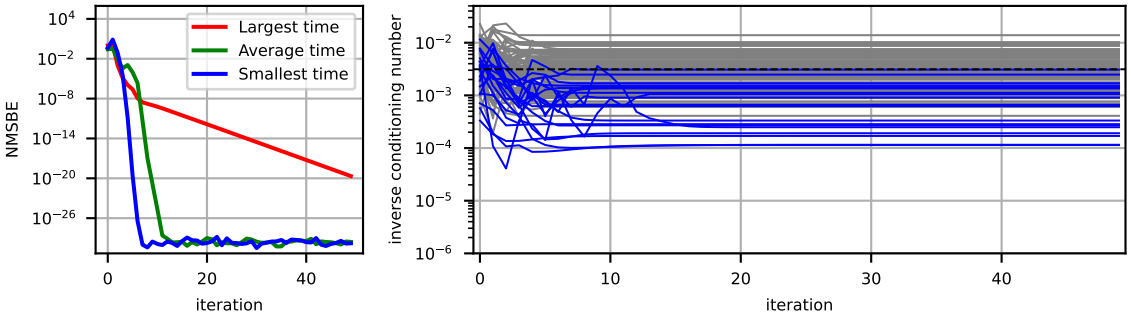
The practical benefit of a multistep approach is visible in Figures 3.6a, 3.6c and 3.6e. Even the worst convergence pattern encountered in all repetitions of the compound methods are now acceptable. Whereas the convergence speed for the worst single-step case is too slow to be of any use (cf. red curve in Figure 3.5b), for $\mathrm{T}_\pi^{(k)}$ with $k = 10$ it is even possible for the optimisation task to converge for bad inverse condition numbers within the considered iterations to the same final error.

TD(λ) methods only demonstrate an unsatisfactory performance. For all tested values of λ, i.e., not only the single value for λ shown in Figure 3.6h, there is no clear advantage over the single-step experiment. Neither the final error nor the convergence speed or the achieved ratio κ are improved. Furthermore, it needs to be stressed that the $\mathrm{T}_\pi^{(\lambda)}$ operator involves 150 transitions in the state space. These are significantly more (s, a, r, s', a') tuples used for training than in the largest compound method. In summary, the increased computational burden without having a positive impact on the quality of value function



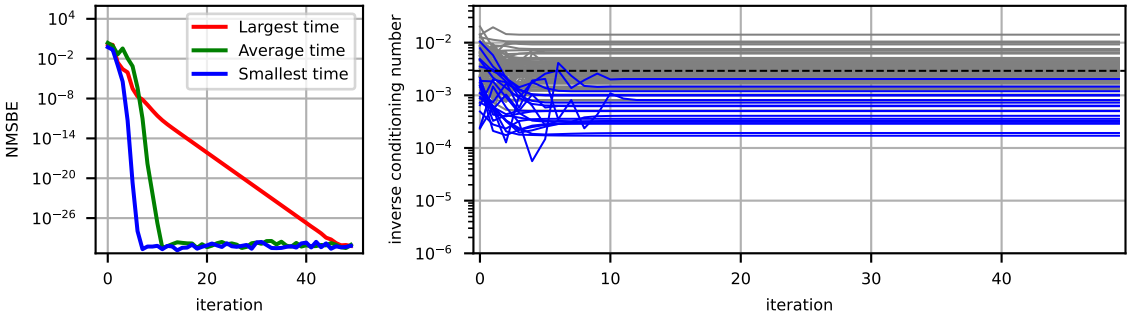
(a) Error over time for $k = 3$.

(b) The inverse condition number of $G(\mathbf{W})$ for $k = 3$.



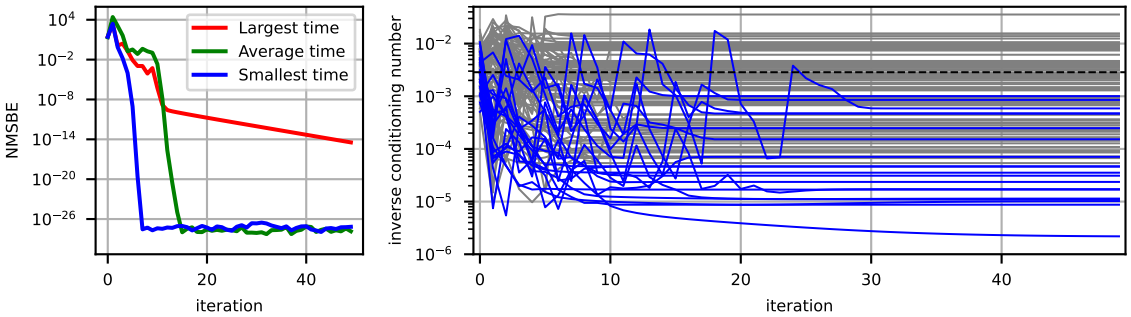
(c) Error over time $k = 5$.

(d) The inverse condition number of $G(\mathbf{W})$ for $k = 5$.



(e) Error over time for $k = 10$.

(f) The inverse condition number of $G(\mathbf{W})$ for $k = 10$.



(g) Error over time for $\lambda = 0.9$.

(h) The inverse condition number of $G(\mathbf{W})$ for $\lambda = 0.9$.

Figure 3.6: The time to settle and inverse condition numbers κ of $G(\mathbf{W})$ for different multistep methods. Experiments use an adapted version of Baird's *Seven State Star Problem*. There are 100 repetitions per choice of k or λ. In blue, the 20 repetitions with the longest time to settle are emphasized.
The compound multistep operator $\mathrm{T}_\pi^{(k)}$ produces better values for κ than $\mathrm{T}_\pi^{(\lambda)}$. Consequently, the convergence is better, even for the worst observed κ.

approximation prevent any justification of their application in discrete state space problems when using MLPs as function approximator and second order methods for training.

3.5 Experiments in Continuous State Spaces

Following my experiments with discrete state spaces, I will now look at continuous problems in detail and investigate my Gauss Newton Residual Gradient algorithm empirically. I outline the general experimental setup first. Second, I introduce the first experiment where I test the convergence behaviour during Policy Evaluation in continuous state spaces. I compare Residual Gradient algorithms with Semi-Gradient formulations and quantify the influence of second-order optimisation on both. Next, I explore the generalisation capabilities of an MLP when trained with a Gauss Newton Residual Gradient algorithm by evaluating the NMSBE on unseen states outside of the training set. In a third experiment, I elaborate on the behaviour of the algorithm when exploiting multiple steps in the transition data. I still work with the Policy Evaluation component and capture empirically the rank of the network Jacobian in a continuous state space setting. Finally, I address the application of a second-order Residual Gradient algorithm in full Policy Iteration and test its performance in a continuous control problem. I investigate, whether a Policy Iteration algorithm based on Gauss Netwon Residual Gradient Policy Evaluation achieves higher rewards compared to its first-order and Semi-Gradient counterparts. Additionally, I measure the impact of multistep formulations on the quality of policies.

3.5.1 Experimental Setup

In all exerperiments, I apply the Gauss Newton Residual Gradient algorithm for Policy Evaluation in finite dimensional and bounded Euclidean state spaces. I provide empirical results for the performance of the Approximated Newton algorithm by minimising the objective of Equation (3.11) in several different scenarios:

- **Convergence of Policy Evaluation** – I analyse the convergence behaviour when evaluating a fixed policy and compare the cases of whether or not considering derivatives of the TD-target. This means, I compare Semi-Gradient and Residual Gradient formulations for both first- and second-order optimisation methods. This draws the connection to existing and important algorithm classes such as DDPG from Lillicrap et al. [2015].

- **Generalisation Capabilities** – I investigate the algorithm in terms of its generalisation capabilities. This is both interesting and important due to a still unexplainable performance of Neural Networks [Lawrence et al., 1997, Zhang et al., 2017, Neyshabur et al., 2017]. Unfortunately, classical measures of generalisation error for MLPs such as Vapnik-Chervonenkis dimension and Rademacher complexity are still not capable of providing sufficient answers [Anthony and Bartlett, 1999, Jakubovitz et al., 2019]. Since it is impossible to include the entire state space via sampling in the optimisation problem, generalisation to unseen states is essential for successful algorithms. Consequently, generalisation capabilities are investigated numerically for different architectures and sampling scenarios.

- **Impact of Multistep Formulations** – I employ a multistep Bellman Operator and capture the effect on the optimisation behaviour of the Gauss Newton Residual Gradient algorithm. Multistep formulations are the tool of choice and appear in many (Deep) Reinforcement Learning implementations. By combining optimisation methods that use approximated second-order information with multistep transition data, I can determine whether they mutually assist one another.
- **Full Policy Iteration** – I combine Policy Evaluation with Q-factors and improve iteratively an initial policy. This reveals the general applicability of a Gauss Newton Residual Gradient algorithm. However, the Policy Iteration algorithm is restricted to the setting employed in this chapter. Namely, the Policy Iteration approach must be realised as Critic-Only method, i.e., there is no explicit policy and one has to retrieve greedy actions directly from a Q-function. To do so, all environments considered for the following experiments possess a discrete action space.

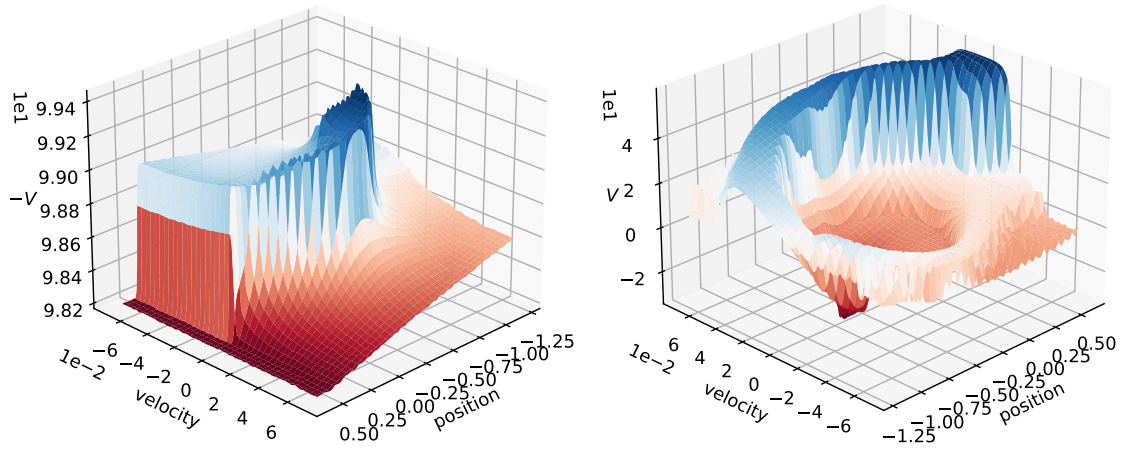
For the experiments, I use the Mountain Car and Cart Pole control problems as benchmarks. These two environments are classic deterministic benchmarks with a typical continuous state space and a manageable complexity, which allows for an extensive investigation and visualisation. Their complete description is available in Sections 2.6.3 and 2.6.4, respectively. I set the discount factor for all environments to $\gamma = 0.99$ if not otherwise specified.

Since the state space in the Mountain Car problem is two dimensional, I am also able to estimate the ground truth value function with Monte Carlo methods based on a fine grained two dimensional grid. Thus, I can evaluate accurately the performance of tested algorithms against a ground truth and also run a qualitative and visual verification of the outcome. Because of the higher dimension of the state space in the Cart Pole environment, I no longer provide visualisation attempts of the value function but rely on the achieved accumulated and discounted reward of a policy as the performance indicator.

Due to historical reasons, I use two different versions of the Mountain Car and Cart Pole environments. The experiments in Sections 3.5.2 and 3.5.3 rely on *MyMountainCar-v0*, whereas in Section 3.5.4 the new version *MyMountainCar-v1* is active. This change has become necessary after my investigation from Chapter 4 has been completed. More details about the reason for this change in the environments is part of the description of benchmarks in Sections 2.6.3 and 2.6.4. Similarly, I perform the experiments related to Policy Iteration in Section 3.5.5 in two different variations. The first uses the original version of the Cart Pole environment, namely *MyCartPole-v0*, and the second employs the new version called *MyCartPole-v1*. Again, details about their differences are given in Section 2.6.4.

The policies, which are used for the Policy Evaluation task, are fixed throughout the experiment but depend on the environment used. For *MyMountainCar-v0*, the policy is given by Equation (2.39) and simply accelerates the car in the direction of the current velocity. When working with *MyMountainCar-v1*, the policy takes the form as described by Equation (2.42). It is capable of steering the car into the goal and stopping there. For these two policies, it is possible to generate a ground truth solution for the value functions through the help of Monte Carlo rollouts. They take the form as shown in Figure 3.7.

To get those ground truths based on rollouts of the respective policies, I create a large number of trajectories and their discounted returns as described at the end of Section 3.4.2. Furthermore, to have estimates close to the solution V_π for the entire state space, I let the trajectories start from states s_0, which are arranged on a regular grid. The resolution of



(a) *MyMountainCar-v0* (b) *MyMountainCar-v1*

Figure 3.7: Ground truths for value function V_π obtained from Monte Carlo rollouts. The policies for evaluation are the predefined ones from the respective environments (cf. Equations (2.39) and (2.42)). Please pay attention to the axis labels and tick values. First, the surfaces are rotated by 180 degree along the z-axis compared to each other. Second, the left surface has the negative z-axis pointing upwards. This manipulation is necessary to reveal for both surfaces the interesting structure, which would otherwise be occluded by the value function itself.

the grid is 500×500 such that there are no artefacts from a low sampling frequency. Per start state there is only one rollout. Since the environments and policies are deterministic, there is no need to combine several trajectories per starting state. And as one can see in Figure 3.7, numerical issues are not noticeable. The trajectories themself consist of 2000 transitions, such that the necessary cut-off in an infinite horizon task does not create strong approximation errors for the discount factor $\gamma = 0.99$. Unfortunately, in my experiments, where I rely on rollouts to assess the quality of a policy during training, I need to reduce the length of trajectories to keep the overall runtime of those experiments feasible. Hence, there will be a slight mismatch between the ground truth and a policy, even if the value function under that policy would be the correct one.

3.5.2 Empirical Convergence Analysis

Setting First, I make use of the Mountain Car problem in the version *MyMountainCar-v0* from Section 2.6.3 and its corresponding predefined fixed policy from Equation (2.39). I investigate the performance of the optimisation problem under the influence of three variants: *ignoring TD targets in derivatives*, *using Hessian based optimisation* and *varying learning rates*. I use four different learning rates $\alpha \in \{10^0, 10^{-1}, 10^{-2}, 10^{-3}\}$ and study all 16 combinations of parameters.

For all tests, I adopt the batch learning scenario. Specifically, I use as training set $N = 100$ transition tuples (s, r, s') collected in prior with actions set by the fixed policy. All states are sampled uniformly from $\mathcal{S}$. Executing the action provided by the policy in each state yields its successor s' and the one-step reward r for that transition. I fix

the transition tuples throughout all convergence experiments to provide a fair comparison between individual runs.

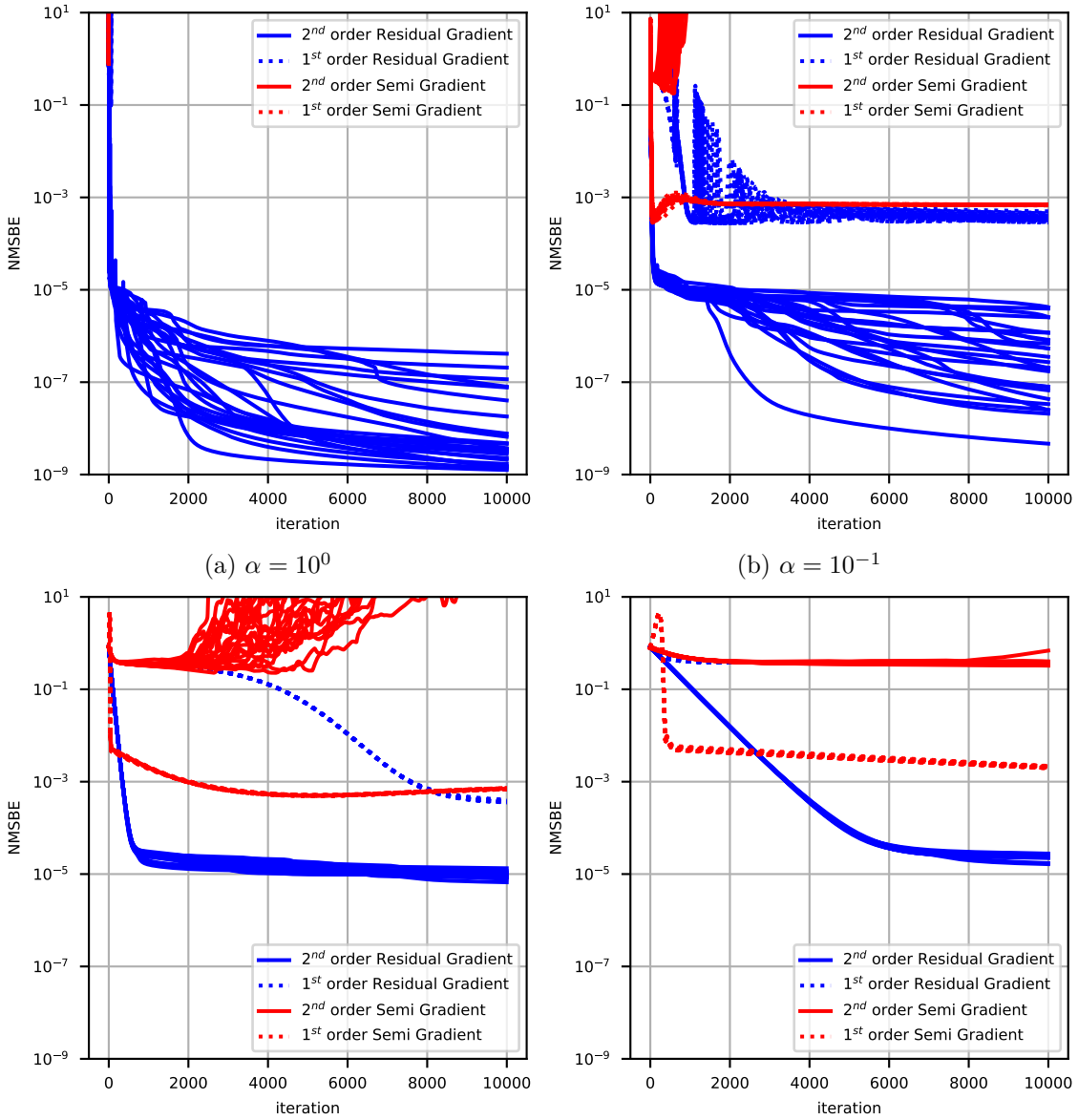
As function space for approximated value functions, I employ the MLP $\mathcal{F}(2, 10, 10, 1)$ with Bent-Id activation functions. This architecture has $N_{net} = 151$ free parameters, i.e., more parameters than the number of samples, and complies to my theoretical considerations. Furthermore, MLPs in this experiment are all initialised with the same weight matrices to further improve the comparability. For initialisation, I use a uniform distribution in the interval $[-1, 1]$.

Although there is no noise or randomness in the environment part involved and I do not make use of exploration mechanisms, the outcomes can still vary. I use an asynchronous and multithreaded implementation such that numerical errors can either accumulate or cancel out over time based on the order of execution. Hence, all experiments are repeated 25 times. My results of the optimisation processes are shown in Figure 3.8.

Results I observe that Semi-Gradient algorithms always diverge for extremely large step sizes as shown in Figure 3.8a with $\alpha = 1$. For smaller step sizes $\alpha \in \{10^{-1}, 10^{-2}, 10^{-3}\}$ as shown in Figures 3.8b, 3.8c and 3.8d, a Semi-Gradient algorithm can converge if using first-order optimisation. Extending it to second-order gradient descent causes it to diverge sooner or later for all step sizes. For small enough learning rates, e.g., $\alpha = 10^{-3}$, second-order Semi-Gradient additionally achieves only the same final NMSBE as first-order Residual Gradient methods, indicating that the computation for the approximated Semi-Hessian is obsolete. However, I want to point out that the resulting solution of Policy Evaluation is not good compared to other possible outcomes.

Figures 3.8b and 3.8c show that first-order gradient descent algorithms with and without ignoring the dependency of the gradient on the TD target perform consistently and achieve almost identical final errors. Looking at the descent behaviour, I can confirm the slow convergence issue of a Residual Gradient formulation. From Figure 3.8d it becomes clear that first-order Semi-Gradient descent combined with carefully selected learning rates $\alpha \in \{10^{-2}, 10^{-3}\}$ can achieve an equal or even lower final error, further explaining its popularity over Residual Gradients.

In contrast stands the Gauss Newton Residual Gradient algorithm, i.e., a Gauss Newton algorithm combined with complete derivatives of the NMSBE. This algorithm works well with all learning rates. In particular, this algorithm works with large learning rates, as it can be seen in Figures 3.8a and 3.8b. Even for an extremely large learning rate $\alpha = 1$, convergence is not a problem. Despite strong initial numerical problems, all repetitions arrive at a sufficiently small NMSBE over time. For all learning rates, the final value for the NMSBE is orders of magnitude smaller than that of other approaches. In other words, building the derivatives of the TD target with respect to the parameters and using (approximate) second-order information of the NMSBE function are important ingredients for designing and implementing efficient NN-VFA algorithms. Modifying the descent direction based on the curvature is crucial to achieve good performance. This insight matches also the empirical evidence in [Gronauer and Gottwald, 2021] that when combining Semi-Gradient algorithms with an annealing scheme on the learning rate, even better performance can be obtained.



(a) $\alpha = 10^{0}$ (b) $\alpha = 10^{-1}$

(c) $\alpha = 10^{-2}$ (d) $\alpha = 10^{-31}$

Figure 3.8: The NMSBE as defined in Equation (3.11) plotted over time for all tested optimisation approaches. Figure a) to d) represent considered learning rates $\alpha \in \{10^{0}, 10^{-1}, 10^{-2}, 10^{-3}\}$, respectively. A Residual Gradient formulation combined with Hessian based optimisation outperforms Semi-Gradient algorithms for all learning rates and stays convergent, even for large values of α. First-order only Residual Gradient algorithms demonstrate the reported slow convergence.

Order	Time (SG)	Time (RG)	Iterations	Error at iteration 1500
1^{st}	8.4 s	7.6 s	10^4	$\approx 3 \cdot 10^{-4}$
2^{nd}	22.4 s	48.0 s	10^4	$\approx 1 \cdot 10^{-5}$

Table 3.1: Runtime requirements with $\alpha = 0.1$.

Computational concerns A severe burden of Newton-type algorithms is the computational complexity involved in evaluating the (approximated) Hessian, especially since the Gauss Newton approximation involves a full sized and dense matrix. If I assume that the gradient direction can be obtained in $O(n)$ operations, then the GN step adds an additional effort of $O(n^3)$ to form the Hessian and solve for Newton's direction. The runtime requirements of the experiments are summarised in Table 3.1.

Calculating the approximated Hessian and Newton's direction using *Eigen*'s Householder QR-decomposition when linked against *OpenBlas* and *OpenLapack* increases the run time significantly. My experiments run on an *AMD 3990X 64-Core* computer, but these execution times are not supposed to be accurate and reproducible measurements. They should only reveal the overall trend. The numbers imply that within 10^4 steps of a first-order Residual Gradient method only around 1500 iterations of a Gauss Newton Residual Gradient algorithm can be performed on the available computer. However, the performance gain in terms of convergence speed and lower error, as seen in Figure 3.8b or Figure 3.8c, still justifies the additional computational effort. A second-order algorithm is in total faster than first-order only methods, because far less iterations are required to reach an already significantly smaller error.

3.5.3 Generalisation Capabilities of MLPs

As mentioned earlier in the critical point analysis, working with finite exact approximators based on N samples from a continuous state space causes generalisation to be an important topic. Since the value function approximator can only be trained to fit a finite set of samples exactly, the generalisation capabilities of an MLP for states in between the collected training samples are essential to DP and worth further investigation. Thus, I evaluate an approximated value function, which I obtain by optimising the NMSBE with the Gauss Newton Residual Gradient algorithm, on parts of the state space, which are not covered by the training data.

Single Architecture

Setting I start with a single architecture and vary the amount of training samples. More specifically, I consider again the MLP $\mathcal{F}(2, 10, 10, 1)$ with learning rate $\alpha = 10^{-2}$ and *Bend-Id* activation function. It possesses $N_{net} = 151$ parameters. For their initialisation, I use a uniform distribution in the interval $[-1, 1]$. In this experiment, the number of training samples N is varied non-uniformly between 25 and 2000. Selected values are given as part of the figure. I compute the NMSBE for a separated test set comprised of $25 \cdot 10^4$ states arranged on a grid. Again, for each N, I use batch training with samples placed uniformly

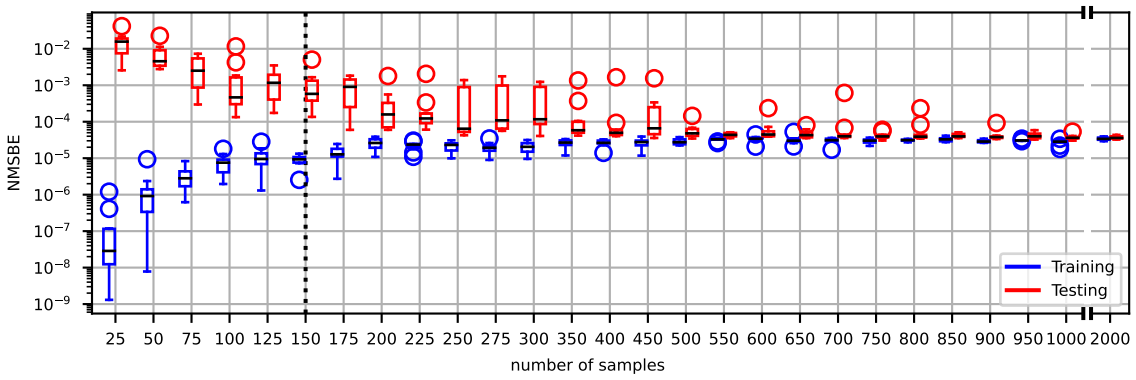


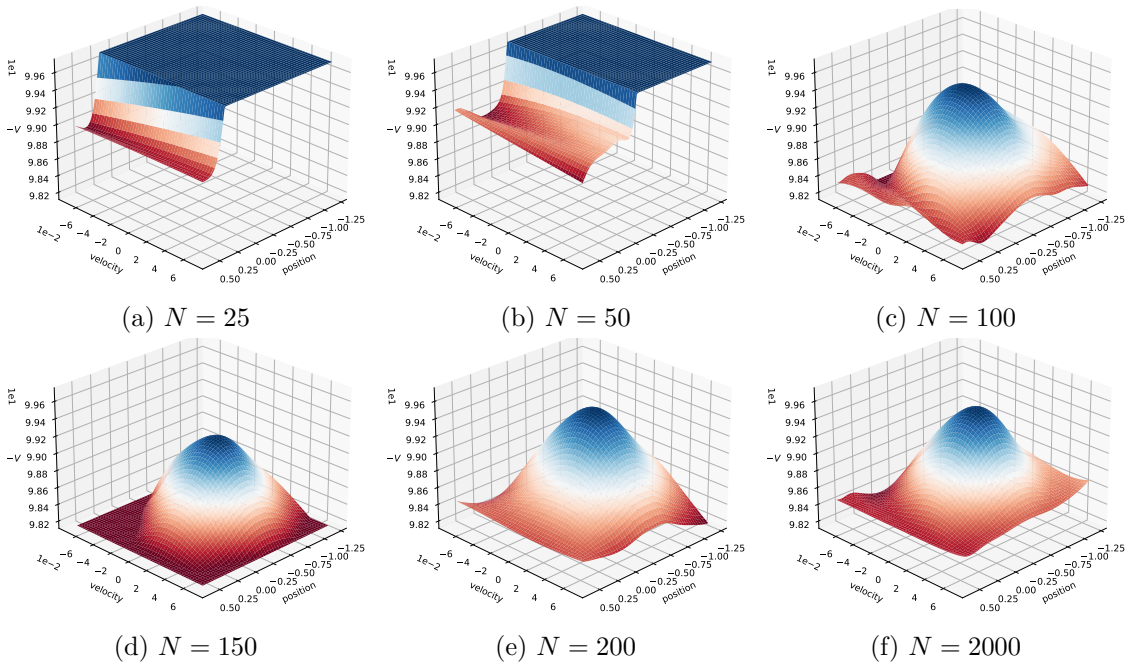
Figure 3.9: The final NMSBE of the MLP $\mathcal{F}(2, 10, 10, 1)$ for different amount N of training samples. The NMSBE using the training samples is shown in blue. That for the test set is drawn in red. The dotted line represents the condition $N = N_{net}$, where it can be seen that the median training and test errors are smaller than the direct neighbours, thus hinting at the important role of over-parametrisation.

in the state space. Every training of MLPs is repeated 10 times. Finally, I visualise the NMSBE as box-plots for the training data and the held-out test set in Figure 3.9.

Results It is clear that for $N = 2000$ samples the training process is always successful with almost ignorable variance. While decreasing the number of randomly placed training samples down to $N = 225$, the median test error becomes larger with more pronounced variances. Only poor results are observed at $N \leq 75$, where the MLP becomes able to fit the training data exactly. But the test error indicates that this is no longer a valid solution.

I observe for the median test and training error that, for $N = 150$ samples, they are smaller or at least equal to their direct neighbours, respectively. Simultaneously, the variance for training errors is reduced compared to the neighbours. Since this number of samples ($N = 150$) is almost identical to the number of parameters in the network ($N_{net} = 151$), these effects could be a numerical confirmation of the role of over-parametrisation as it has been derived from theoretical considerations in Section 3.3. However, I admit that an empirical verification is rather involved. In particular, I require N unique samples in the training set, but there is no practical way to determine, whether those N collected samples are "sufficiently unique".

When evaluating the approximated value function with smallest test error for $N \in \{25, 50\}$ samples visually, i.e., see Figures 3.10a and 3.10b, one can spot a plateau located at around -100 expected discounted reward. This is exactly the solution to the Bellman Equation or, more precisely, to the loss as defined in Equation (3.11), if every transition would yield -1 reward. I conclude that those scarce samples do not allow the reward information to flow and, hence, I have solved implicitly a different MDP. By comparing the ground truth value function as shown in Figure 3.7a to other value functions learned with different sample sizes as shown in Figures 3.10c to 3.10f, I see that training with even only 100 samples starts to fit the shape of the ground truth value function in the correct range. Hence, this experiment suggests that for problems with a continuous state space, NN-VFA methods can still perform well with a relatively small number of sampled



(a) $N = 25$ (b) $N = 50$ (c) $N = 100$

(d) $N = 150$ (e) $N = 200$ (f) $N = 2000$

Figure 3.10: Approximated value functions for different sample sizes N with minimal test NMSBE, evaluated at the same states as for the MC version. A minimal amount of data is required to learn the main structure of the value function. However, increasing the amount of data beyond a certain value does not bring any visible benefit.

interactions. Furthermore, with only $N = 100$ training samples, which corresponds to the over-parametrisation regime, it is possible to learn qualitatively the same approximation of the value function as when using significantly more training data. This implies on the one hand that it is possible to avoid the computational burden regarding the larger training data set. On the other hand, it also demonstrates that defining a reliable error measure between functions is a non-trivial task on its own and must be used with care.

Various Architectures

Setting It is widely believed that the architecture of an MLP has an important influence on its generalisation performance, but the exact impact is still unclear. Therefore, I perform the previous experiment for several MLPs with different architectures $\mathcal{F}(K_{\mathcal{S}}, w \times d, 1)$. Next to varying the number of samples in the same scenario as before, I select as network depths $d \in \{2, 3\}$ and as widths $w \in \{1, 2, 3, \ldots, 20\}$. Concrete choices of architectures and the values of N are available as axis labels in Figure 3.11. I provide the mean test and training error as contour plots in Figure 3.11 with logarithmic scale ($Z = \log_{10}(E)$, where E is the actual error and Z its plotted value). This additional processing step is required to reveal the detailed structure of the surface.

Results To be able to relate intuitively a certain MLP architecture and its number of parameters with the amount of samples used for training, I introduce in all contour plots

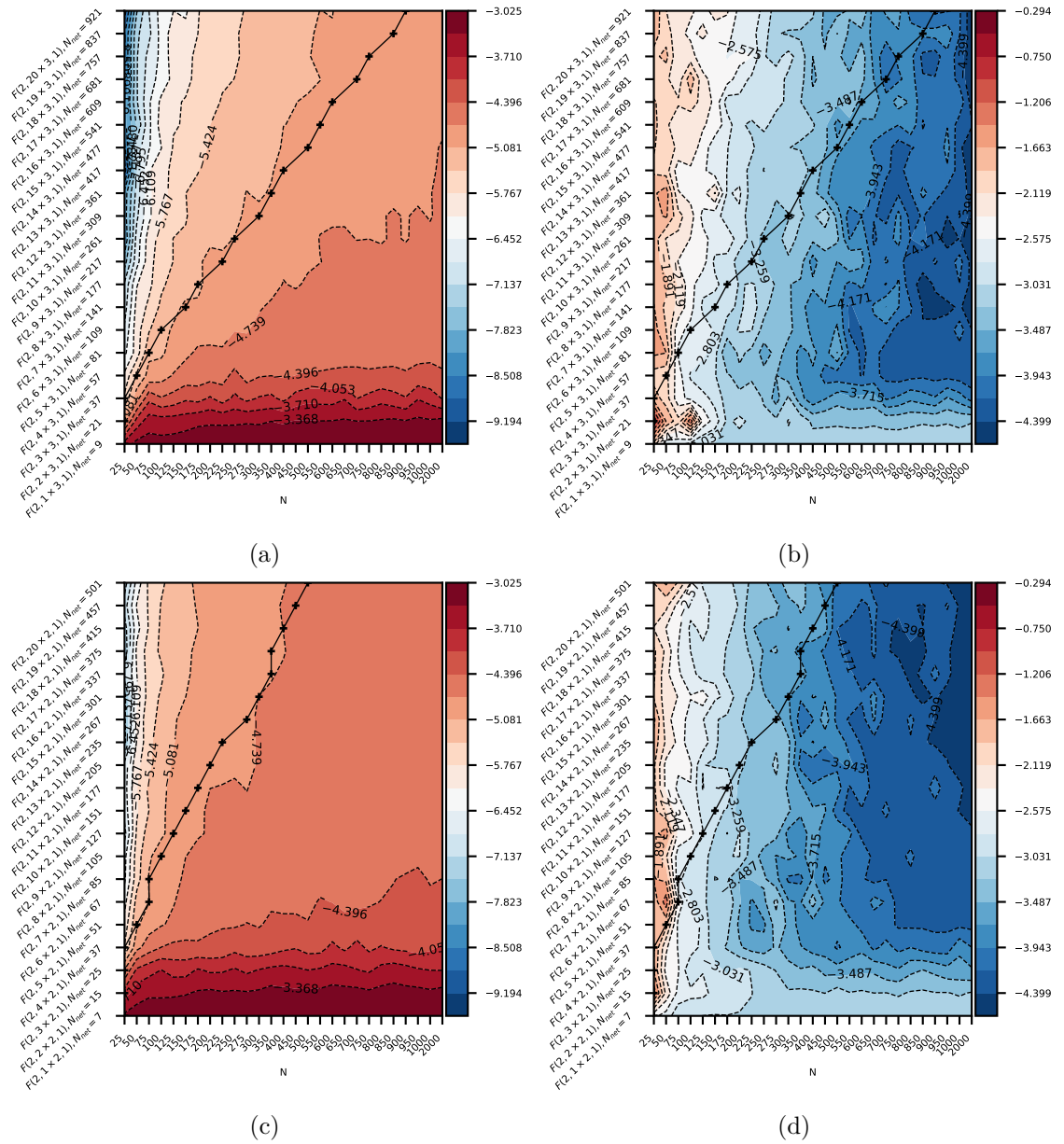


Figure 3.11: The training and test error of different MLP architectures (ordinate) for various sample sizes N (abscissa). I use a logarithmic scale $Z = \log_{10}(E)$, where E is the original error and Z its plotted value. Red indicates higher errors. In all plots, the solid line represents the condition $N_{net} \approx N$. The left column shows training errors, the right provides test errors.
For training, one can see that low error as well as the case of exact fitting happens to the upper left of the black solid line, confirming considerations from Section 3.3. The testing errors reveal that, at least for MLPs with a large number of parameters, the region with smallest test error reaches close to the condition $N_{net} \approx N$. However, for all (small) MLPs the well-known rule "the more data, the better performance" applies.

an auxiliary line called "condition line" to represent the condition $N_{net} \approx N$. I obtain this line by rounding the amount of parameters N_{net} to the closest value of N and joining the corresponding tuples in figures. In the following, I look separately at training and testing errors and start with the training errors, i.e., the contour plots in the left column of Figure 3.11.

The isolines of the error surface consists of almost straight lines starting from the origin. Hence, when I increase the amount of data I also have to increase the depth or width of an MLP to maximise the chance of avoiding suboptimal minima. If the number of parameters is large compared to the number of samples, then the isolines are almost vertical. This implies that the training error is independent of the MLP architecture as long as the architecture is sufficiently rich and over-parametrisation still applies. Only when passing the condition line, it is possible to cross easily isolines, thereby increasing the training error significantly. In each of Figures 3.11a and 3.11c, there exists each at least one isoline, which stays close to the condition line. For MLPs with $d = 2$ this happens at $-4.739 \approx \log_{10}\left(1.82 \cdot 10^{-5}\right)$ and for $d = 3$ at $-5.081 \approx \log_{10}\left(8.30 \cdot 10^{-6}\right)$. Thus, this behaviour suggests that the geometry of the error surface reflects indeed the condition in Equation (3.19) from my theoretical investigation. Also, exact learning of all samples indicated by training errors close to zero happens only above the condition line, i.e., whenever the network has more parameters than the number of training samples.

Next, I address the testing error as shown in the right column of Figure 3.11. For depth three MLPs, where the condition $N_{net} \approx N$ is available for large values of N, I observe that the region with smallest test error extends always to the condition line. This happens for MLPs wider than $\mathcal{F}(2, 15 \times 3, 1)$ at $N = 500$. I see that larger MLPs work more predictable, whereas smaller MLPs still can achieve the smallest test error for several training set sizes N with higher errors in between. In other words, large MLPs do not necessarily increase the threshold of required samples for good performance in my experiment. Small MLPs need samples in a similar scale as the largest MLPs to achieve the smallest test error. Even those MLPs, which one would consider as tiny such as $\mathcal{F}(2, 7 \times 3, 1)$, also achieve the smallest test error with the same amount of samples. Thus, if for small MLPs the condition $N_{net} > N$ is far from being realistic, the well-known statement "the more data, the better performance" applies. One needs roughly a factor of ten more training data than adjustable parameters. In summary, large MLPs concentrate the region with extreme values for the test errors and amplify the effect of the sample size. For $N \approx 75$, the largest architectures produce also almost the highest test errors. Hence, by shrinking the amount of parameters in an MLP such that one moves closer to $N_{net} \approx N$ one can reduce computational complexity of the model without changing the amount of data and the test error. If $N \to 1000$, one has to employ MLPs with a comparable amount of parameters such that the area with smallest test error is present. These MLPs then possess the highest generalisation performance out of all architectures.

3.5.4 Multistep Impact

From an intuitive perspective, one would expect a better performance when switching to multistep methods. But since a Gauss Newton Residual Gradient algorithm converges already without problems, one could also argue that a beneficial impact of multiple transitions depends highly on the algorithm and problem at hand. Hence, I repeat partially my previous experiments, but use the multistep operators $\mathrm{T}_{\pi}^{(k)}$ and $\mathrm{T}_{\pi}^{(\lambda)}$ to define the fixed

point problem. Again, I compare first-order and Hessian based descent algorithms and test, whether different values for amount of transitions per start state create a meaningful difference.

Convergence Behaviour

Setting Training is performed in a batch setting with both a GN and first-order only RG algorithm. For the former, I use a constant learning rate $\alpha = 0.1$. And for the latter $\alpha = 0.01$. Start states are sampled uniformly from the entire space. I use $N = 250$ samples for training. For representing value functions, I take an MLP consisting of two hidden layers with 15 units each and employ Bent-Id activation functions. The input layer accepts two dimensional state vectors. The output is scalar with linear activation. Initial parameters are drawn element-wise uniformly from the interval $[-1, 1]$.

I test the compound Bellman Operator $\mathrm{T}_\pi^{(k)}$ with $k \in \{1, 2, 3, \ldots, 10\}$. The exponentially weighted operator $\mathrm{T}_\pi^{(\lambda)}$ is used with $\lambda \in \{0.5, 0.9, 0.99\}$. The trajectories are stopped after $\{50, 250, 1000\}$ transitions, respectively. The trajectory lengths L are selected such that λ^L is sufficiently small (i.e., in the order of 10^{-5}).

I show results for 15 repetitions, where I randomise the training data and initial network parameters in each run. The visualisation of the convergence behaviour during training for both multistep methods is contained in Figure 3.12. Figure 3.13 emphasizes the final training error for different parameters as box plot.

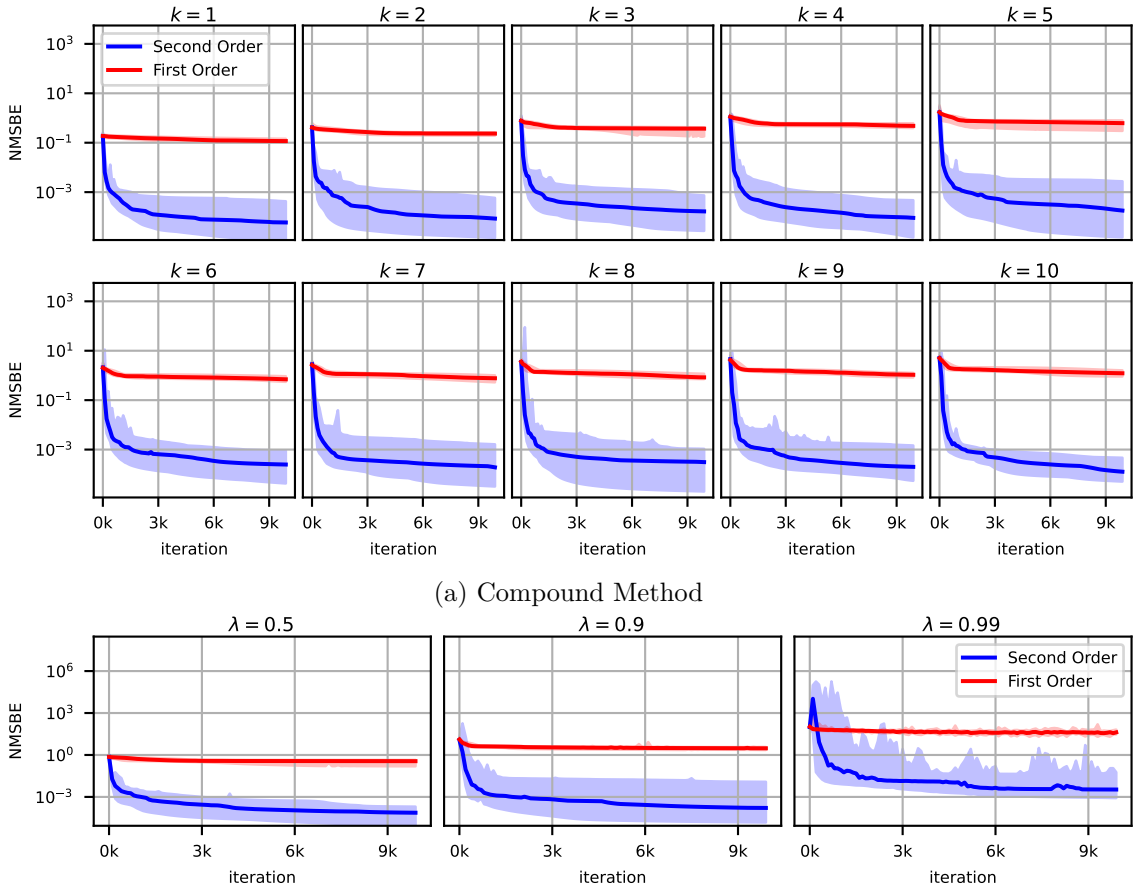
Results Most notably, I observe a missing beneficial impact of $\mathrm{T}_\pi^{(k)}$ and $\mathrm{T}_\pi^{(\lambda)}$ with increasing k or λ for GN based optimisation. Using multiple transitions during training does not clearly help with convergence and the final outcome.

Only when training with a first-order RG algorithm, the convergence speed can be slightly enhanced by increasing k. The known slower convergence of RG algorithms for $k = 1$ vanishes with larger values, at the price of a higher error. However, this effect is rather subtle and not easily visible in Figure 3.12 because of the required range for the y-axis. In Appendix B, an alternated version of the figure is available to demonstrate this behaviour more clearly.

For both first-order and Hessian based optimisation, a stepwise larger k results in worse final training errors. Simultaneously, the descent behaviour for larger lookahead becomes more unstable as well, in particular for $\lambda = 0.99$ with 1000 transitions. This could be explained by the fact that an MLP with identical capacity must fit a more complex objective. The increase of the final error is more pronounced for gradient only optimisation than for the GN method.

A larger MSBE at the end of training would imply that the solutions represented by a trained MLP become worse. Thus, I check also the value functions visually. They are rendered in Figure 3.14.

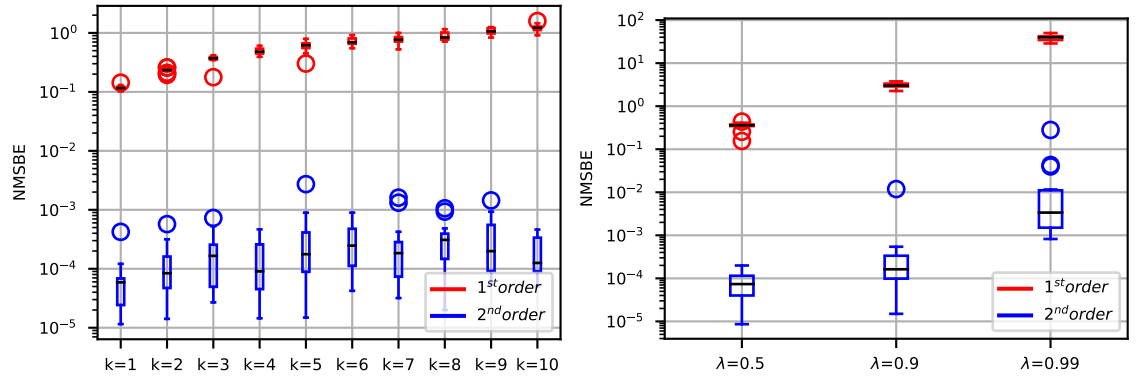
The best approximated value function for $k = 5$, which is shown in Figure 3.14e, possesses a qualitatively better shape despite a higher training error than the best value function for $k = 1$ depicted in Figure 3.14d. This can be interpreted as a confirmation of the initial thoughts in Section 3.3.3. Multistep operators are better in conveying the reward information, thus the MLP is required to learn a better approximation. However, increasing the lookahead further seems to hinder the training again. The value function for $k = 10$ in



(a) Compound Method

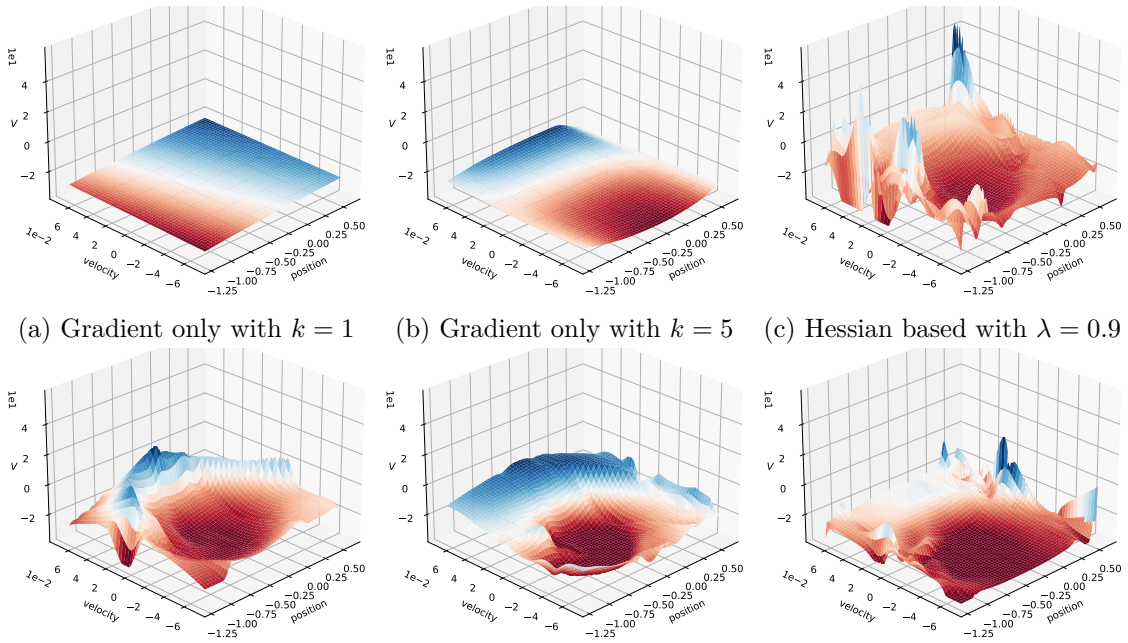
(b) TD(λ)-like Method

Figure 3.12: The impact of multistep lookahead onto first-order and Hessian based RG algorithms. I test both multistep operators $\mathrm{T}_\pi^{(\lambda)}$ and $\mathrm{T}_\pi^{(k)}$. Only first-order optimisation shows better convergence for larger k or λ. Yet, the final achieved NMSBE increases with the lookahead.



(a) Compound Method (b) TD(λ)-like Method

Figure 3.13: Final training errors for different multistep methods and parameters. The box plots unveil the increase of final MSBE with greater lookahead.



(a) Gradient only with $k = 1$ (b) Gradient only with $k = 5$ (c) Hessian based with $\lambda = 0.9$

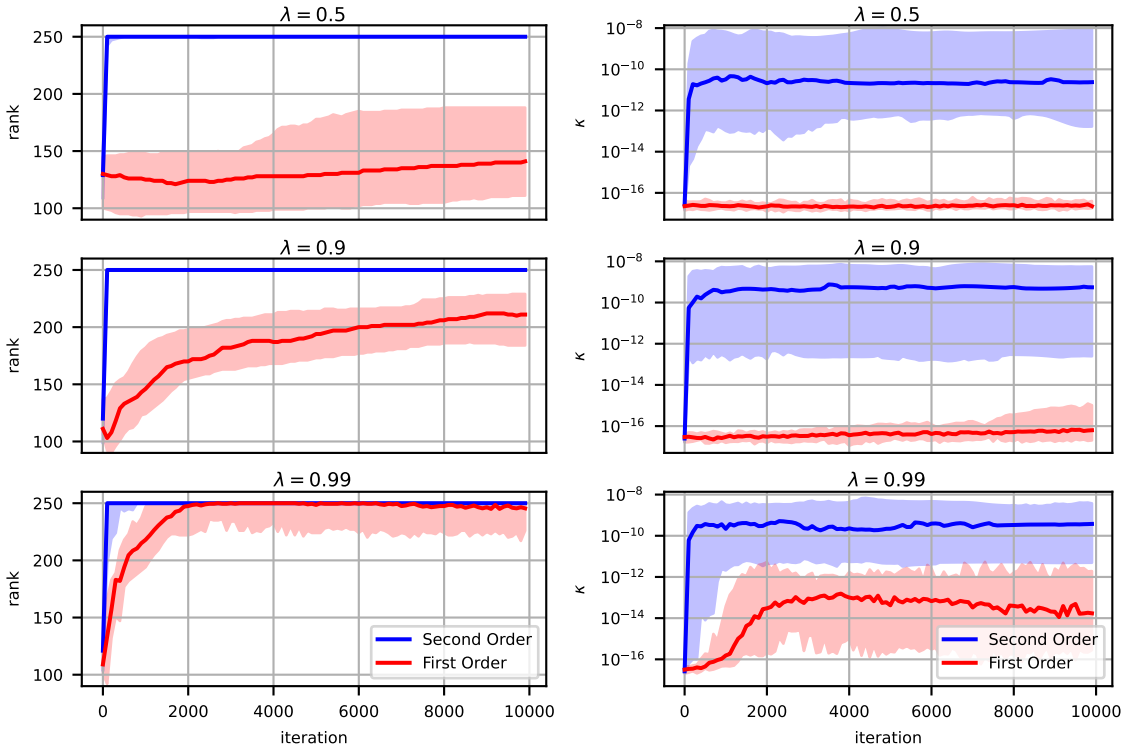
(d) Hessian based with $k = 1$ (e) Hessian based with $k = 5$ (f) Hessian based with $k = 10$

Figure 3.14: Best approximations to V_π according to smallest NMSBE. Training uses first- and second-order optimisation combined with the compound and TD(λ) multistep formulation. In general, one can not obtain a good approximation V_π from a visual perspective, in particular, when comparing against the ground truth in Figure 3.7b. However, with second-order optimisation, one can see for $k = 1$ and $k = 5$ that at least a qualitative resemblance is present.

Figure 3.14f and for $\lambda = 0.9$ in Figure 3.14c appear more jagged. It seems that there is a certain trade-off between the size of the lookahead and the difficulty of the learning task due to a higher complexity of the objective. The value functions for first-order optimisation are almost identical for all tested parameters and multistep methods. They look similar to V_π in Figures 3.14a and 3.14b and do not contain the spiral-like structure. In accordance to the insights about TD(λ) from the discrete setting in Section 3.4.4), the experiment with continuous state space also suggest that the similar quality for the value function approximation does not justify the additional computational effort for the long trajectories, in particular for larger λ.

Rank Analysis

Setting Next, I investigate also the rank of $\widetilde{G}^{(k)}(\mathbf{W})$ and $\widetilde{G}^{(\lambda)}(\mathbf{W})$ during training as I have done in discrete setting. My goal is to see, whether the claims on a full rank of the Jacobians hold, in particular for the single-step case. Furthermore, analysing the rank or inverse condition numbers ultimately reveals, whether one should rely on multistep formulations when incorporating approximated Hessian information or not. I work with the exact same setting from the convergence experiment in Section 3.5.4. For all runs of this experiment, I compute the matrices $\widetilde{G}^{(k)}(\mathbf{W})$ and $\widetilde{G}^{(\lambda)}(\mathbf{W})$ for all iterates $\mathbf{W}$. The rank is determined as described in Section 3.4.4 and singular values follow directly from a



(a) Rank (b) Inverse condition number

Figure 3.15: The inverse condition numbers κ and ranks for $\widetilde{G}^{(\lambda)}(\mathbf{W})$ during training. The MLP satisfies $N_{net} = 301 > 250 = N$ and one can see that rank and condition numbers behave correctly during training for second-order optimisation. First-order optimisation suffers from rank deficient Jacobians such that there are more local minima with poor NMSBE compared to second-order training case. This matches observed higher errors from previous figures.

Singular Value Decomposition of the matrices. The results for the TD(λ) approach with rank and inverse condition numbers of $\widetilde{G}^{(\lambda)}$ are given in Figure 3.15. Figure 3.16 shows the results for the compound Bellman Operator $\mathrm{T}_{\pi}^{(k)}$. The upper part depicts ranks during training for first- and second-order methods for all repetitions. The lower part combines the smallest and largest singular values to yield the inverse condition number κ for all iterations. The parameters k and λ take the values mention before.

Results Opposed to the discrete setting with only a few states, the rank itself becomes useful to some extend on its own. For both multistep approaches with second-order descent, the rank of most repetitions jumps almost immediately to the maximal value. But there are a few cases, where the matrix $\widetilde{G}(\mathbf{W})^{(k)}$ is obviously rank deficient. For the first-order descent algorithm, the situation is apparently bad. Almost all runs do not reach a full rank for their the Jacobian during the entire training time. The only exception is $\mathrm{T}_{\pi}^{(\lambda)}$ with $\lambda = 0.99$. There it happens for some runs to reach a full rank for the Jacobian. The resulting value function is more pronounced and not as flat as others for first-order learning (cf. Figures 3.14a and 3.14b), but has the overall same shape.

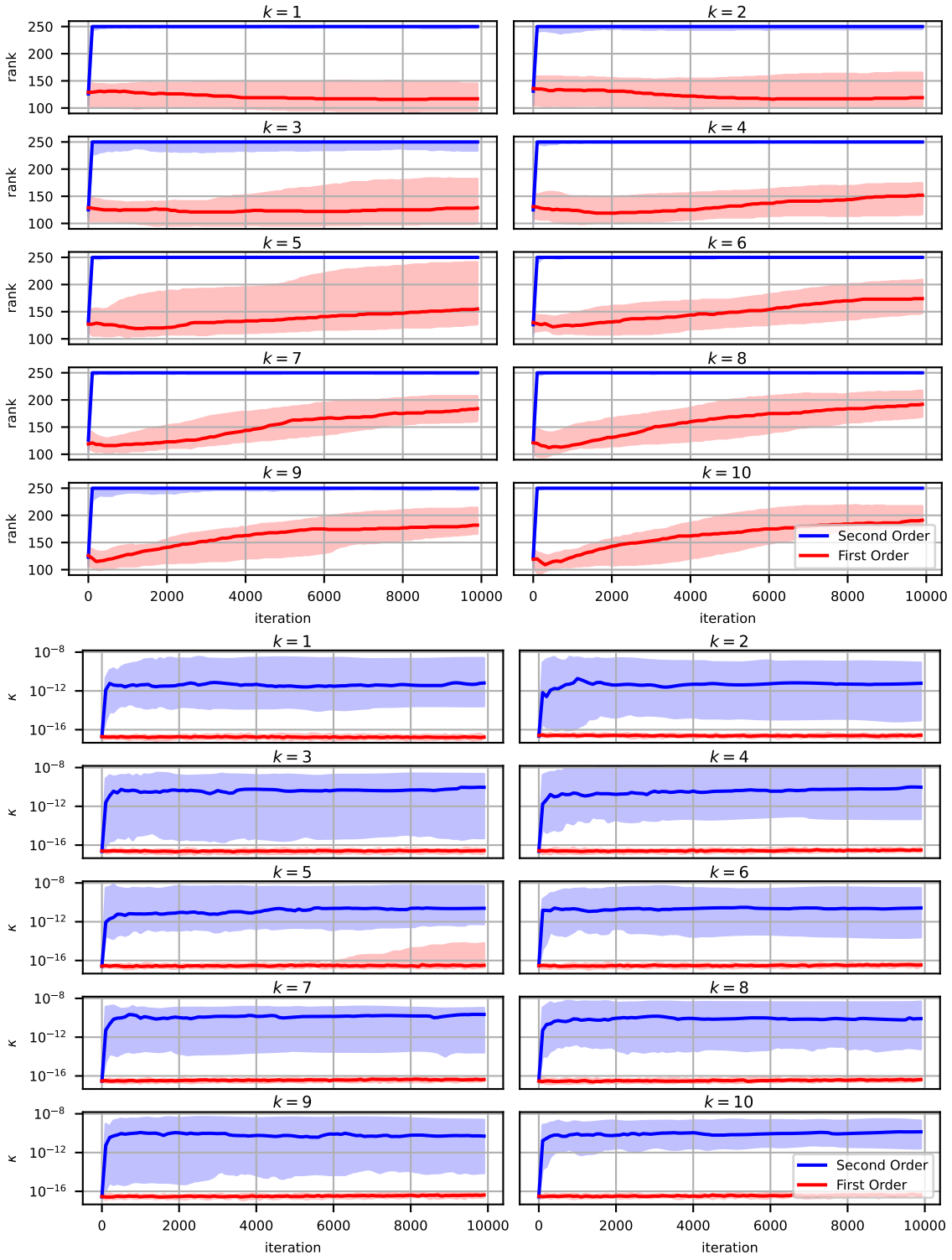


Figure 3.16: The inverse condition numbers κ and ranks for $\widetilde{G}^{(k)}(\mathbf{W})$ during training. The MLP satisfies $N_{net} = 301 > 250 = N$ and one can see that rank and condition numbers behave correctly during training for second-order optimisation. First-order optimisation suffers from rank deficient Jacobians such that there are more local minima with poor NMSBE compared to second-order training case. This matches observed higher errors from previous figures.

When using as before the inverse condition numbers as indicator, a more fine grained classification becomes available. Most notably is that the κ values do not jump as they do in the discrete setting. Hence, there is no need to talk about the time to settle as before. Most runs for the second-order optimisation have healthy values around or above 10^{-12} and keep them consistently. Only some have bad inverse condition numbers, which are sitting close to 10^{-16}. Thus, their Jacobians do not behave well and this cannot be spotted based on the rank alone. But these few runs are not representative for the others. For first-order methods, a different picture becomes visible. Despite the rank seemingly approaching its maximum over time, during the entire training all κ values are located at the bottom of the range without clear trend of improvement.

An important observation in Figures 3.15 and 3.16 is that the initial rank of the MLPs Jacobian is not full. This is either given by computing the rank directly or via the inverse condition number κ. In Appendix B, a variation of the figures with focus on the initialisation is available to make the observation more pronounced. Thus, the Jacobian and Proposition 4 imply that there are many critical points, because the critical point condition allows for a plethora of solutions. The extreme case $\widetilde{G}^{(k)}(\mathbf{W}) \equiv 0$ or $\widetilde{G}^{(\lambda)}(\mathbf{W}) \equiv 0$, which would cause any update direction for parameters being a critical point, does not appear in any repetitions.

On top of the initialisation, first-order optimisation suffers strongly from rank deficient Jacobians during the entire training phase. Hence, according to the reasoning around Proposition 4, there are multiple local minima with poor MSBE present for all iterations . This matches observed higher errors from Figure 3.12a. Using gradients alone for realising a descent algorithm does not work, since no matter what direction is selected, it is too easy to hit arbitrary minima with larger errors. Even worse, since the minima exist mainly due to the null space of $\widetilde{G}^{(k)}(\mathbf{W})$ and $\widetilde{G}^{(\lambda)}(\mathbf{W})$, the minima is not an isolated point but can be a connected set. The iterates then could move freely without causing any improvement for the error.

The second-order training case does not have this issue. The full rank requirement for the Jacobian is satisfied after only a couple of iterations such that the only type of extrema left are global ones. Of course, one has to take Remark 1 into account and consider the case that no extrema exist at all. Since the MSBE is not approaching zero within numerical limitation of a computer, it might be that critical points are out of reach and one can only reduce the error as long as possible.

The loss of rank or small κ values at the beginning of training are not a significant hurdle. Due to the Gauss Newton approach for approximating the Hessian, the descent direction is the gradient with some perturbation. It arises from the regularisation term and also from a bad approximation of the Hessian when being further away from extrema. This also explains, why it is possible to see increasing errors for some iterations during training. Hence, it is rather unlikely (but not impossible) to get attracted by "early" critical points, which exists due to the null space of $\widetilde{G}^{(k)}(\mathbf{W})$ or $\widetilde{G}^{(\lambda)}(\mathbf{W})$. Only once the "true" critical points, i.e., those, which are relevant for Proposition 4, get close by, the Hessian and thus the descent direction becomes correct. Since one can observe that the rank goes up, one can conclude that optimisation obeys theoretical considerations.

Remark

It is important to emphasize that the experiments suggest that multistep methods are not a global remedy for all problems in DP with NN-VFA. In particular, for the second-order training case, one can see that $k = 1$ works also in the continuous setting reliably. Changing to larger lookahead has a beneficial impact on κ, but it is questionable if this is really worth the effort. Judging based on Figure 3.12, there is no clear advantage or disadvantage for larger lookahead for the compound Bellman Operator in terms of final error. The same holds true for λ-weighted lookahead. Larger λ such that rewards from later successors have more impact on the current state only increases the effort without providing better approximations of value functions. The only case, where an advantage is visible, is that for first-order gradient descent and a TD(λ) approach. With increasing λ, the rank becomes almost full and the inverse condition numbers improve for the entire training process. Still, high final errors suggest that the approximation is not a good fit for the actual value function.

3.5.5 Policy Iteration

A First Attempt

Setting To examine the practical applicability of Gauss Newton Residual Gradient optimisation, I change the environment to Cart Pole and test my approach in a Policy Iteration setting. For that purpose, I use Q-factors instead of the state-only value function V and use a greedily induced policy as defined in Equation (2.15). To represent Q-factors, the network input consists of the continuous state and the discrete action index, hence MLPs need $K_{\mathcal{S}} + 1$ units in the input layer. More specifically, I use the MLP $\mathcal{F}(5, 15, 15, 1)$ with $N_{net} = 346$ parameters to approximate the Q-function. Actions are selected by brute force enumeration and evaluation of all possible values in a certain state. The learning rate is set to $\alpha = 0.1$.

Primarily, I am interested in the effect of different sample sizes and how the amount of Policy Evaluation iterations impacts the overall performance. Hence, I select the number of states, which are sampled uniformly from the entire state space before every Policy Evaluation step, in the range $N \in \{100, 200, 300, 400, 500, 1000\}$. The number of descent steps, which also controls the quality of the Policy Evaluation component, is taken from $i \in \{50, 100, 150\}$. Additionally, I also compare the impact of reusing the last parameters of the previous Policy Evaluation step as initialisation for the current sweep. Based on the outcome of experiments in Section 3.5.2, I also wonder, whether a Semi-Gradient formulation can work for a PI setting. Thus, I also test, whether the slow convergence of first-order methods is that severe for full Policy Iteration. With regard to Section 4.3, both versions of the Cart Pole benchmark, i.e., *MyCartPole-v0* and *MyCartPole-v1*, are tested to see, whether a proper system dynamics and reward function are necessary for the success of a PI algorithm.

I measure the performance of a policy after every improvement step as described at the end of Section 3.4.2. I obtain unbiased estimations for the discounted reward by performing several rollouts with 500 transitions, which start from random but kind start states. By the adjective “kind” I denote a region in the centre of the state space, where the cart is located towards the middle of the track with the pole being almost upright. Furthermore, there is only a small but still non-zero initial velocity for both the cart and the pole. I

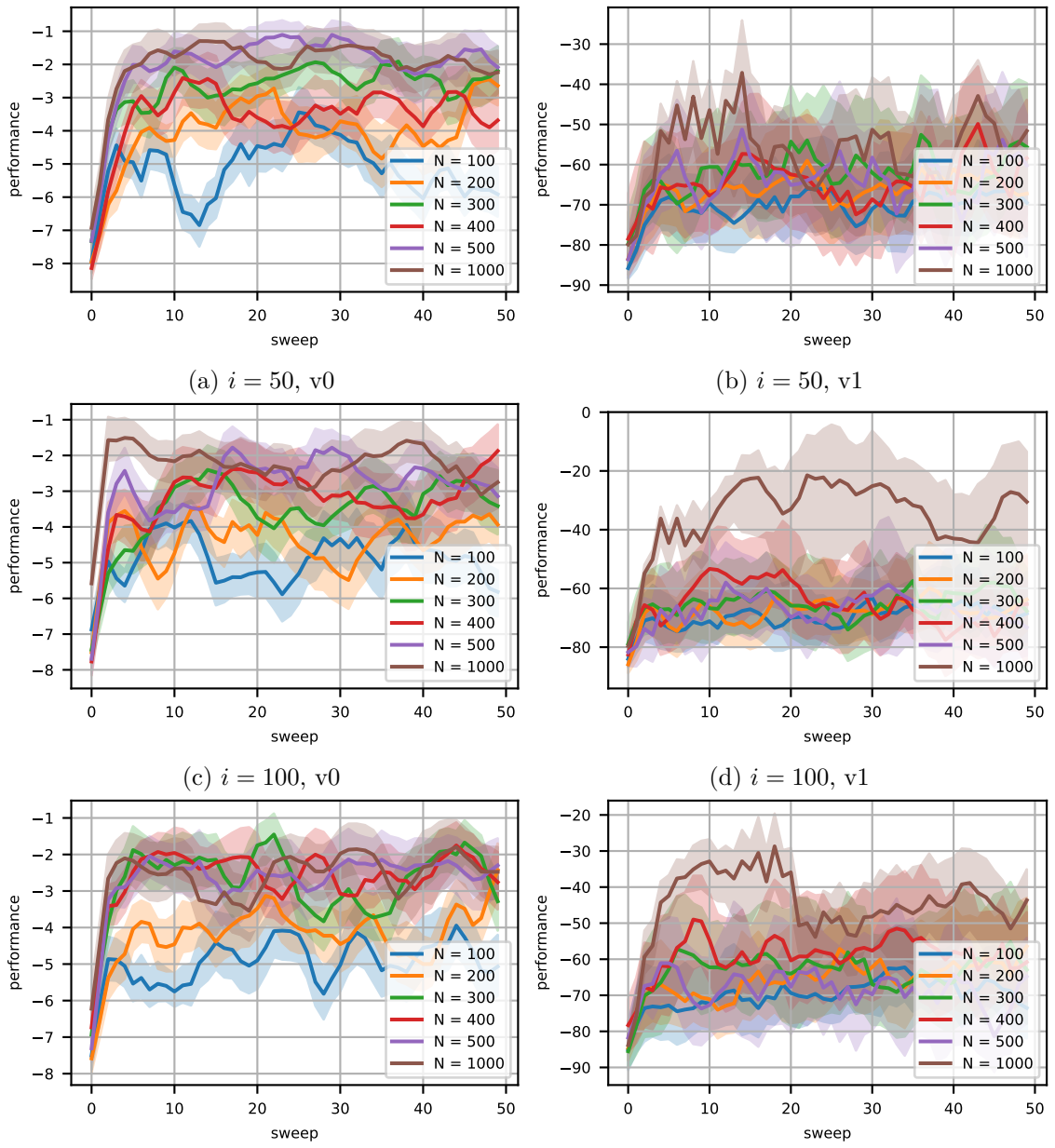
Option	Effect	Comment
Transient parameters	More random training	Progress is reset randomly, $i >> 150$ does not help
1^{st} order Semi-Gradient	Divergence	only tested in *MyCartPole-v0*
2^{nd} order Semi-Gradient	Lack of improvement & Divergence	
1^{st} order Residual Gradient	Divergence	

Table 3.2: Hyper parameter with convergence issues.

have decided to use less transitions in trajectories compared to the computation of the ground truth to reduce the overall runtime of the experiment. Even if this causes the approximation error of expected discounted rewards to be larger, it is still possible to see the learning progress.

In order to provide figures, which enable reliable statements independently of the initialisation for MLP parameters, every parameter combination is executed ten times from scratch. Therefore, the combination of rollouts across all sweeps requires an additional processing step. For each repetition, I obtain one average return $\bar{R}_l$ as well as the minimal and maximal returns, which are denoted by R_l^- and R_l^+, respectively. The index l denotes the repetition and ranges from $l = 1, \ldots, 10$. I combine all repetitions by forming the average of these terms over all l, i.e., I compute $\bar{R}^- = \frac{1}{10}\sum_l R_l^-$, $\bar{R}^+ = \frac{1}{10}\sum_j R_l^+$ and $\bar{R} = \frac{1}{10}\sum_l \bar{R}_l$. The overall average performance $\bar{R}$ can then be visualised as single curve, which indicates the typical return of the policy. The average best cases $\bar{R}^+$ and worst cases $\bar{R}^-$ are drawn as shaded area around $\bar{R}$. The size of the shaded area indicates how stable the process is. All curves receive a uniform filter with size three to remove only the most drastic jumps in the graphs for consecutive sweeps.

Many combinations of all tested parameters result in an immediate divergence of MLP parameters within the first five sweeps. If they do not diverge, then there are still many cases, where only chaotic performance values are present, which tremble around in a minimal to barely improved accumulated reward spectrum. Other choices for parameters do not have a beneficial impact and, for example, render the training simply more random. Table 3.2 summarises the effects of those bad performing parameters and, if required, their interactions. To compensate the seemingly random reset of progress for transient parameters, it seems obvious that one should use significantly more training iterations per sweep. Because every sweep starts with a fresh MLP, the optimisation process needs enough time to arrive at a good approximation of V_π. However, I have seen in my experiments that choosing $i = 5000$ descent steps does not compensate the resetting behaviour. The progress of Policy Iteration still appears to be lost from time to time. This stands in conflict with insights from the convergence experiment (cf. Figure 3.8), which suggest that $i = 5000$ iterations should be enough to realise Policy Evaluation. As a consequence, I work exclusively with persistent parameters and a small amount of iterations as specified above. This also has the pleasing effect that the overall runtime of the Policy Iteration algorithm is reduced. My results for working PI experiments are visualised in Figure 3.17.



(a) $i = 50$, v0

(b) $i = 50$, v1

(c) $i = 100$, v0

(d) $i = 100$, v1

(e) $i = 150$, v0

(f) $i = 150$, v1

Figure 3.17: Combining Gauss Newton Residual Gradient Policy Evaluation with Q-factors and greedily induced policies to obtain full Policy Iteration. As environment serves Cart Pole in versions $v0$ and $v1$. The graphs show the expected reward according to rollouts.
Sequentially improving policies are to some extent visible, especially for the environment $v0$. However, there is no clear trend for the required number of samples or the optimal amount of Policy Evaluation iterations. For all cases, the quality of policies is bad, since an optimal policy with perfect balancing should produce close to zero rewards (i.e., no punishment at all). Without second-order optimisation or when using Semi-Gradients, Policy Iteration does not work or diverges after a couple of sweeps.

Results Experiments with Semi-Gradients for both first- and second-order descent algorithms and tests with a first-order Residual Gradient formulation do not show improving policies over time, or even diverge. Thus, I can only provide results for my Gauss Newton Residual Gradient formulation. Fortunately, this behaviour already implies that second-order information is essential to enable and stabilise Policy Iteration in continuous control problems with Non-Linear Value Function Approximation architectures. It also shows that in order to get the benefits of second-order approaches, the TD target cannot be ignored during the computation of differential maps.

Sequentially improving policies are visible for several parameter combinations in Figure 3.17. But this happens only at the beginning of training and only until a certain reward threshold. A stable and repeatable training of policies from scratch is not available.

In all plots of Figure 3.17, it is hard to find a clear trend for the required number of samples or the optimal amount of Policy Evaluation iterations. Figures 3.17a, 3.17c and 3.17e allow for the statement that once the number of sample states N is getting close to or exceeding the number of MLP parameters $N_{net} = 346$, then the highest rewards are visible. However, since proper balancing would result in zero punishment, i.e., the highest reward, one needs to classify these policies as poor result in terms of solving the task.

With the other version of the environment, performance generally looks worse with one exception. In Figure 3.17d, one can see that $N = 1000$ samples and $i = 100$ descent steps for Policy Evaluation produces for several repetitions well performing policies. The accumulated reward in this environment needs not be necessarily exactly zero to indicate a proper policy. Slight swinging around the balancing point produces a minor negative reward, but the pole is indeed kept upright.

It is interesting that both decreasing and increasing the amount of Policy Evaluation iterations results in worse performance. But based on my available experimental results, I cannot make reliable statements at this point.

Although I observe a slightly chaotic behaviour, I argue that this is to be expected, since function approximation is utilised in a Policy Iteration framework. Thus, there are two unavoidable sources of errors, namely inaccurate Policy Evaluation and erroneous Policy Improvement. First, different than typical RL settings, where samples come from an exploration mechanism, I select samples uniformly distributed in the entire state space. Thus, my control problem and learning algorithm does not have a high resolution focused on visited parts of the state space, but try to find a global solution, which is obviously a more challenging problem. Second, I make use of discrete actions. This means, I demand a smooth function approximation architecture to model jumps in the value function, which must occur for example around the balancing point of the pole.

Remark on Visualisation Providing intuitive visualisations for the training progress of the PI algorithm, without having by accident false claims on the level of performance, is a challenge on its own. To emphasize the difficulty in visualising results of several repetitions of the same experiment, I provide here the raw rollout values for the seemingly best performing parameter combination in Figure 3.17d. Ten individual executions for $N = 1000$ samples with $i = 100$ Policy Evaluations steps are depicted in Figure 3.18.

All repetitions use the same hyper parameters N and i. One can see from the ten individual graphs in Figure 3.18, that the various rollouts per individual repetition of the entire training process produce mostly similar discounted returns. This means, after

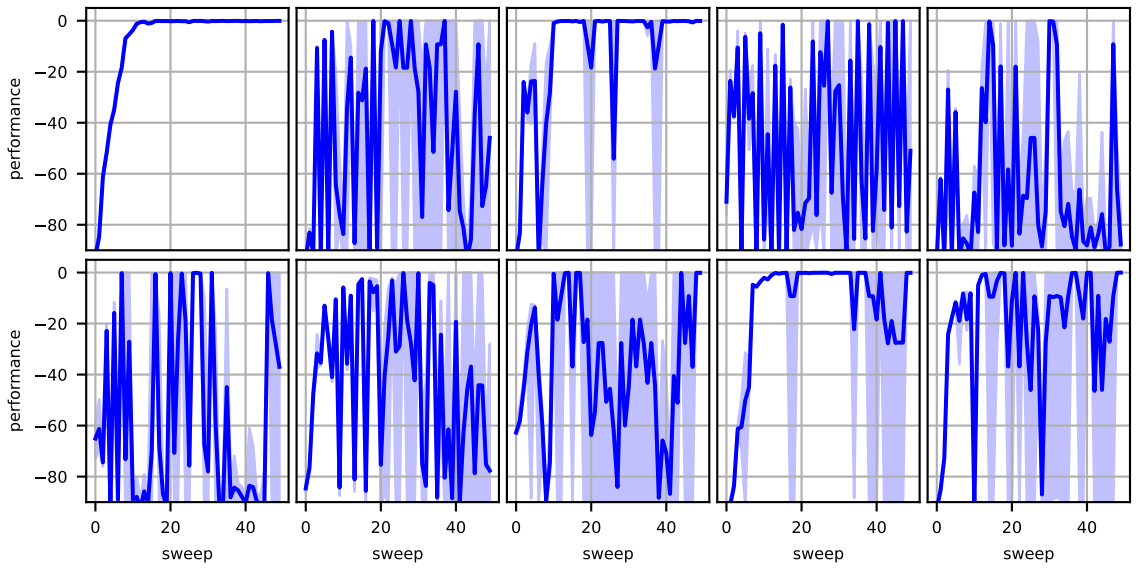


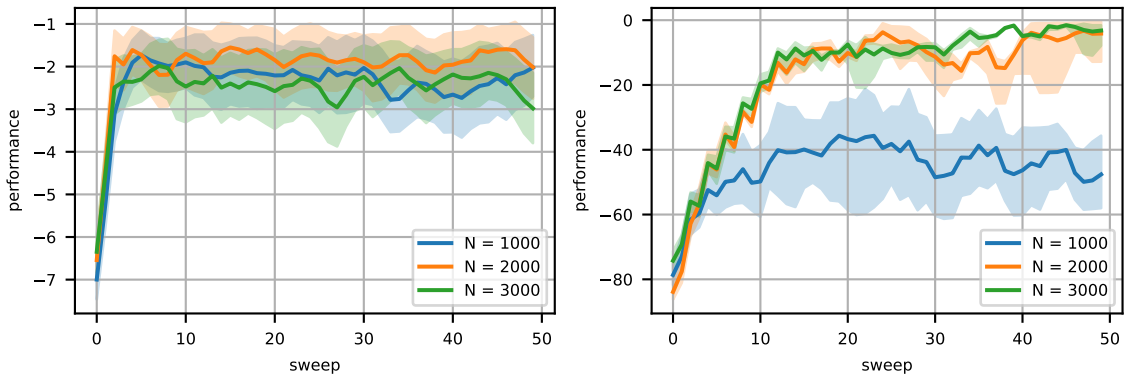
Figure 3.18: The individual performance curves, which result in a single combined line in Figure 3.17d. Whereas repeated rollouts for a certain approximated Q-function achieve often consistent rewards (single graph with shaded area), redoing the overall experiment can result in completely different behaviour (all ten cases compared against each other).

initialising the MLP, the evaluation of the policy produces consistent values. Cases, where the complete shown range of the y-axis is covered by the shaded area, are a due to a single rollout reporting the minimal or maximal accumulated discounted reward. The remaining ones stick to the average value.

Running again the whole experiment produces varying performance curves. This corresponds to comparing individual graphs in Figure 3.18 with each other. Unfortunately, these curves can be rather dissimilar. By averaging the mean accumulated rewards over time for all ten executions, I can highlight the trend for performance over time in the overall experiment. Additionally, by combining the min-max areas as described earlier, it is possible to obtain an intuitive understanding for the algorithm. If the shaded area is still small after the combination, then all repetitions of the experiment perform consistently and the combined graph allows for robust statements. If not, then a more sophisticated investigation is required and the plots shown in Figure 3.17 are not suitable for drawing concrete conclusions. The reader must reproduce the experiment on his or her own and investigate all runs individually.

Leaving the Over-parametrisation Setting

Setting My previous PI experiments have been constructed around the theoretical insights from the critical point analysis in Section 3.3.2. Namely, the MLP size should be synchronised with the amount of training samples. Thus, I have selected the number of samples N around the given number of MLP parameters N_{net}. However, if one keeps the generalisation experiment from Section 3.5.3 in mind and, in particular, the test errors



(a) $i = 300$, v0 (b) $i = 300$, v1

Figure 3.19: Combining the Residual Gradient Gauss Newton Policy Evaluation with Q-factors and greedily induced policies to full Policy Iteration. As environment Cart Pole is employed in both versions. The graphs show the expected reward obtained by performing rollouts.
More data and Policy Evaluation iterations seem to help and solve the struggle from previous experiments. For version $v1$ of Cart Pole, the algorithm achieves consistently working policies, whereas original version $v0$ appears unsolvable. The beneficial effect of additional data stands in conflict with theoretical insights, but matches generalisation experiment.

from Figure 3.11, then the obvious strategy is to incorporate significantly more data into the Policy Evaluation task than parameters of the MLP.

Therefore, I repeat the complete experiment from the previous section with the same settings, expect for a different amount of iterations i and sample states N. The number of iterations is fixed to $i = 300$ and I vary the amount of training data in the set $N = \{1000, 2000, 3000\}$. As usual, I sample start states uniformly from the entire state space. I include more iterations i to reduce further the NMSBE based on idea that a higher quality for Policy Evaluation step cannot cause any harm. However, when taking the overall runtime into account if N is increased, the large amount of iterations from previous experiments (e.g. $i = 5000$) is not possible, because for PI, an MLP is trained every sweep.

Using the same strategy for visualisation as before, I provide the resulting performance of the PI algorithm when using more training samples in Figure 3.19.

Results Most notably, the Gauss Newton Residual Gradient algorithm, when used inside a Policy Iteration algorithm, is now working. All experiments with enough data, i.e., $N \in \{2000, 3000\}$, produce consistently policies in the upper reward range. The sizes of shaded areas in Figure 3.19 are small compared to those in Figure 3.17, indicating that policies can be optimised reliably and independently of the initialisation.

Increasing only the amount of descent steps to get a better Policy Evaluation is not sufficient to enable training progress. This can be seen when comparing the curve for $i = 300$ and $N = 1000$ in Figure 3.19b with the curves corresponding to $i \in \{100, 150\}$ and $N = 1000$ from the previous Figures 3.17d and 3.17f. The discounted reward produced by the policies remains at approximately -40 for all values of i when using only $N = 1000$

samples for training. Only with larger N, it is possible to obtain policies with correct balancing behaviour, i.e., an accumulated reward close to zero.

It is important to emphasize the difference in the resulting quality of policies between the two versions of the Cart Pole benchmark. For the original environment *MyCartPole-v0* with performance curves from Figure 3.19a, the increased amount of data does not solve the previous problem with the maximal achievable expected reward. Even the best policies do not solve the task, not even from a qualitative perspective. Once the formulation of the environment fits natively into the MDP language and the reward signal is replaced with a function, which does not contain extended constant regions, well performing policies are possible. For *MyCartPole-v1* and $N = 3000$, the policies manage to achieve almost zero punishment for all repetitions. This can be seen starting at sweep 40 in Figure 3.19b. Furthermore, those policies exhibit also from a visual perspective a proper balancing behaviour.

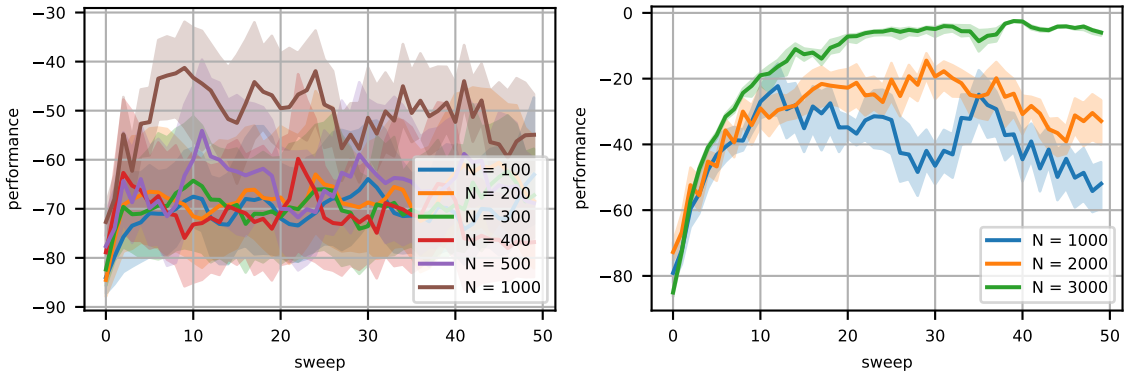
A possible explanation for this behaviour is related to insights from Section 4.3. Namely, it is crucial that the Q-function maintains a proper shape such that a greedily induced policy according to Equation (2.15) has a high chance for selecting the correct action.

Effect of Multistep Operators on Policy Iteration

Setting As the last experiment for a Policy Iteration algorithm, I want to see, whether multistep formulation provide at least a noticeable boost for the performance of polices. Hence, I take the same experimental setting for Policy Iteration as in my experiments from the previous two paragraphs. But this time, I realise Policy Evaluation through the compound multistep operator. I check impact of multistep lookahead for $k \in \{5, 10, 15\}$. Based on my insights from the *Seven State Star Problem* in Section 3.4.4 and from the multistep convergence tests in Section 3.5.4, I ignore completely the TD(λ) method. Furthermore, I also omit the environment *MyCartPole-v0*, because the existing single-step PI experiments suggest that this environment is simply not solvable. I provide my results in Figure 3.20.

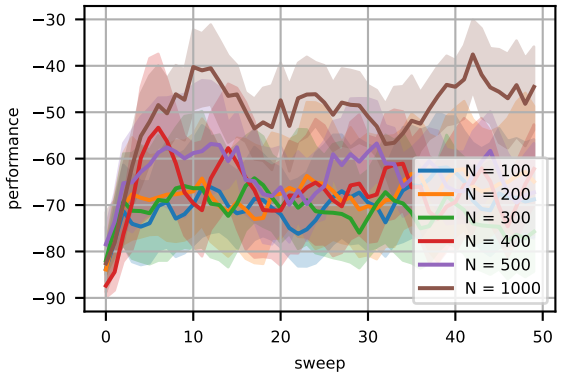
Results The performance curves reveal that the TD(k) multistep method does not help Policy Iteration, at least if second-order optimisation is involved for the Policy Evaluation component. Independently of k, I only observe the same behaviour for the PI progress as in the single-step case with small amounts of data (i.e., $i \leq 1000$). Once enough samples are involved, i.e., $N \in \{2000, 3000\}$, PI starts to work as for $k = 1$. This can be seen from Figures 3.17b, 3.17c and 3.17f. Whereas a lookahead of $k = 5$ does not seem to have any impact, selecting the largest lookahead $k = 15$ even destroys the high quality of policies such that the accumulated reward settles at approximately -20 for $N = 3000$. This effect matches the hindered convergence behaviour for large λ or k in Section 3.5.4.

As a conclusion for the setting I consider, a multistep formulation does not help to increase the performance. If the amount of data is large enough, then the algorithm simply works (right column in Figure 3.20). Also, one cannot overcome the struggle, if only little amount of data is used (left column in Figure 3.20). The performance curves stay similar to that of Figure 3.17d for $k = 1$ and by using a multistep formulation one increases only the computation burden.

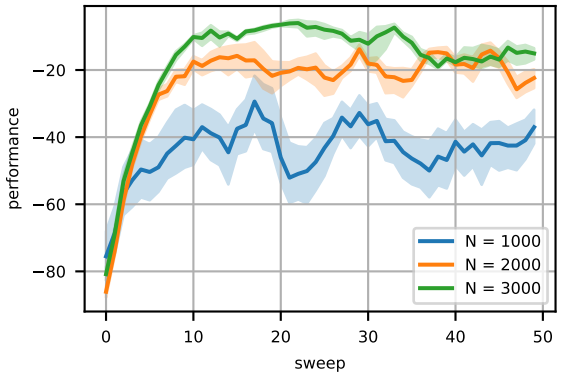


(a) $k = 5, i = 100$

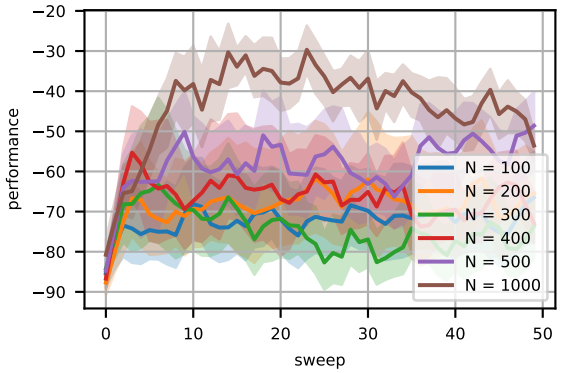
(b) $k = 5, i = 300$



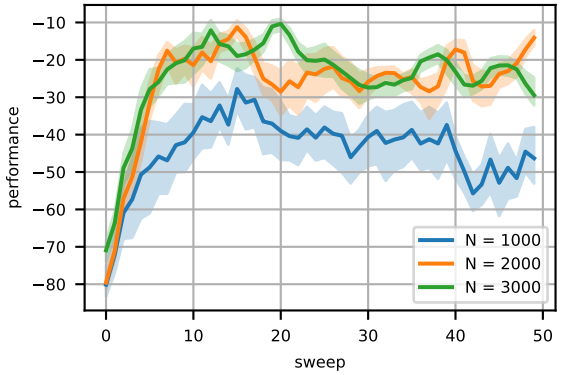
(c) $k = 10, i = 100$



(d) $k = 10, i = 300$



(e) $k = 15, i = 100$



(f) $k = 15, i = 300$

Figure 3.20: Residual Gradient Gauss Newton based Policy Iteration for different sizes of multistep lookahead with compound Bellman Operator $\mathrm{T}_{\pi}^{(k)}$. The environment is *MyCartPole-v1*. All figures show the expected reward of the policies obtained by performing rollouts.
One can see, that TD(k) does not bring an advantage. On the contrary, a bigger lookahead k leads to larger demand of training data N to maintain identical performance. PI executions in the left column do not achieve a proper progress at all and resemble those with $k = 1$.

3.6 Remark: Over- vs. Under-parametrisation

If I compare my theoretical results from Section 3.3 with those from experiments in Section 3.5, then it is apparent that there exists a certain conflict. My analysis of critical points suggests that one should select for given MLP the amount of samples such that $N_{net} \geq N$ holds. In practice however, I find that the relation $N \geq c \cdot N_{net}$ with $c \geq 1$ is needed to enable proper progress for a Policy Iteration application. My goal for this section is to summarise the empirical insights regarding data consumption and also to provide a possible explanation, which justifies the observed behaviour within my analysis of critical points.

The initial question, which needs to be answered, is, whether there is a fundamental problem with my approach through non-convex optimisation. Afterwards, it can be hypothesised, whether there are additional and still undiscovered mechanisms at work. Therefore, I need to test first, whether the MLP I have selected in previous experiments is indeed capable to model the correct value function. Second, I need to ensure that the data itself is not a source of artefacts.

To do so, I run another collection of experiments focused around solving the Policy Evaluation task, but employ this time two fundamentally different learning methodologies. I compare my proposed approach, namely minimising the NMSBE with a Gauss Newton Residual Gradient algorithm, with a pure Supervised Regression algorithm. For both strategies, I use the same MLP architecture and identical start states as training data. Target values, which are required to realise Supervised Regression, are given by Monte Carlo rollouts performed in all starting states. This boils down to using the V_π values shown in Figure 3.7b directly for fitting. To ensure that Supervised Regression does provide an unbiased comparison, I do not make use of my Gauss Newton algorithm as I do later on in the regression task from Section 4.6, but rely on a "typical" setting. This means, I make use of *Torch* [Paszke et al., 2019] as a common Deep Learning framework and use the provided *Adam* optimiser [Kingma and Ba, 2014] and settings according to best practices.

The experimental setup orients itself on that of the previous section. The environment is still *MyMountainCar-v1* such that I can compare value functions visually with those in Figures 3.7 and 3.14. Furthermore, this also allows me to draw a connection to Actor-Critic algorithms and their limitations as described in Section 4.5.4. The MLP consists of two hidden layers with 15 units each and Bent-Identity as activation function. The input is two dimensional matching the state space and the MLP has a single linear output unit to represent the expected discounted reward. Thus, the MLP has $N_{net} = 301$ parameters. Training consists for both methods of 10^4 iterations. For the Gauss Newton Residual Gradient algorithm, I set the learning rate to $\alpha = 0.1$. For Supervised Regression, I use $\alpha = 0.01$ and let *Adam* handle the adaptions. For *Adam* itself, I rely on default parameters as they are given by the authors.

The actual experiment now uses two different amounts of start states N and both learning approaches. For the first amount of training samples, I select $N = 250$, which is the number of start states used in the multistep experiment from Section 3.5.4. With $N = 250$ samples, I comply to the results from the critical point analysis, i.e., there are more parameter N_{net} than samples N. But at the same time, this value for N also has revealed more clearly the issues regarding learning the correct value function. As the second value for N, I pick $N = 3000$. This amount of training samples has been deemed necessary during the Policy Iteration experiments in Section 3.5.5 or also during the experiments with an Actor-Critic

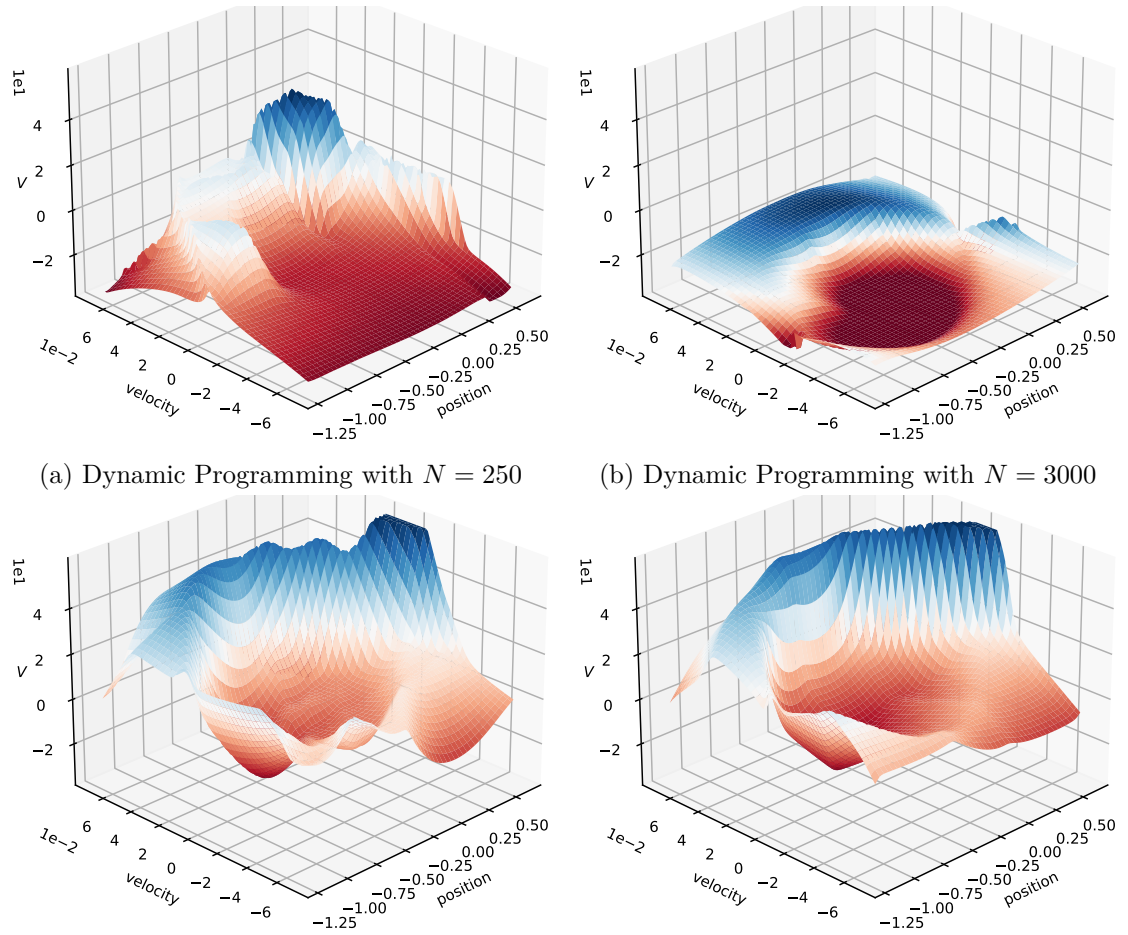
algorithm from Section 4.5.4. Therefore, I investigate four different combinations: *learning methodology* and *over- or under-parametrisation*. Every combination is executed ten times to reduce the impact of initialising the MLP parameters randomly. I show the value functions for the smallest achieved error in Figure 3.21 and the errors for each iteration in Figure 3.22.

Figures 3.21c and 3.21d suggest that the MLP is indeed expressive enough to approximate the ground truth, which is shown in Figure 3.7b. When looking at the red curves in Figures 3.22a and 3.22b, one sees that regression ends with a mean squared error of about 30 or, in other words, the approximated value function is offset by approximately 5 units on average from the ground truth. Thus, the theoretical considerations in Section 3.3.2 can be maintained in numerical applications. In particular, Assumption 2 is not entirely out of reach. Of course, creating a perfect zero would involve MLPs with infinite size.

Next, for the under-parametrisation setting, the final NMSBE achieved by the Gauss Newton Residual Gradient algorithm is around $2 \cdot 10^{-4}$. Since one knows from Figure 3.16 that the rank of the Jacobian is full, this is a strong hint that the situation outlined in Remark 1 applies as is. Optimisation is possible as long as the numerical precision of the computer system is not the limiting factor.

In the over-parametrisation setting, the value function resulting from the DP approach, which is shown in Figure 3.21a, primarily only shares a qualitative resemblance to that of regression (Figure 3.21c) or the ground truth (Figure 3.7b). It does not posses the exact right numerical values and appears to be offset across the entire state space. Here, I can conclude that the NMSBE as objective for a Residual Gradient formulation in continuous spaces is too complicated to obtain easily a good approximation for a value function with an MLP. This emphasizes, why a characterisation of all critical points, as I have done in Section 3.3, is crucial. To come up with a sophisticated Approximated Newton algorithm, all moving parts of an optimisation problem must be identified and understood. Also, my other experiments match the insights from the critical point condition. If working in an over-parametrised configuration, the only full solution from the DP perspective is to have a vanishing TD-error vector for all states. But at the same time, due to a limited number of samples, the approximation of integrals in the MSBE, which is required to arrive at the sampling based formulation for a practical algorithm, is of low quality. Hence, one does not capture the entire requirement or information in the optimisation task for solving the MDP properly. This is where Proposition 3 applies with its negative consequences for training. Namely, solving the MDP works from the algorithmic perspective in the sense that some loss becomes small. But the outcomes is not a solution to the actual engineering problem.

More puzzling, but in a positive way, is the beneficial effect of increasing the amount of samples, i.e., switching to the under-parametrised setting. Whereas the result of regression becomes more sharp (cf. Figure 3.21d), the value function from minimising the NMSBE only reproduces the correct underlying structure. Figure 3.21b shows a coarse spiral-like pattern, but the value function is completely offset when compared against the ground truth. The result of this single experiment is also backed by the investigation of generalisation capabilities from Section 3.5.3. To my understanding, such a puzzling nature of this behaviour stems directly from the critical point condition in Equation (3.14). Once the condition $N > N_{net}$ holds, then there is a guaranteed null space in the matrix $\widetilde{G}(\mathbf{W})$, in which $N - N_{net}$ components of the Bellman Residual $\widetilde{\Delta}_\pi(\mathbf{W})$ can vanish. Thus, one would expect critical points to include many solutions with completely arbitrary value functions. However, since the rank of $\widetilde{G}(\mathbf{W})$ in the under-parametrisation setting is still N_{net} and



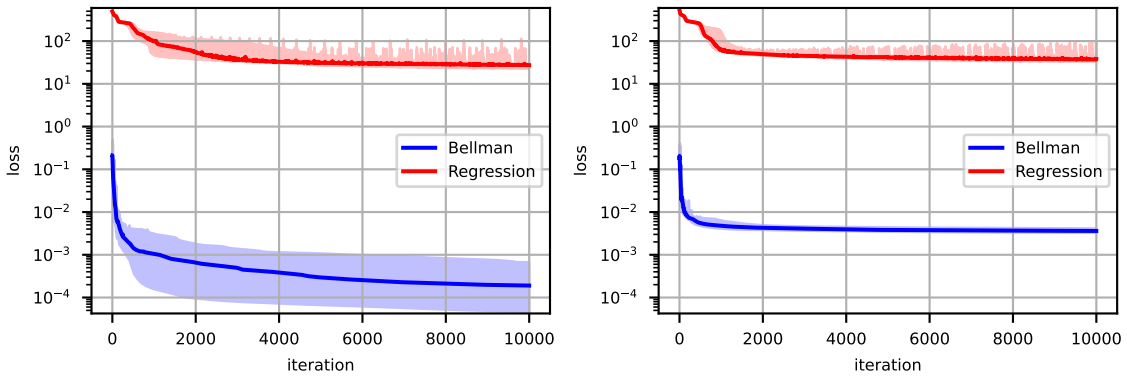
(a) Dynamic Programming with $N = 250$

(b) Dynamic Programming with $N = 3000$

(c) Supervised Regression with $N = 250$

(d) Supervised Regression with $N = 3000$

Figure 3.21: Best approximations to V_π according to the smallest errors. The error is given by Bellman Residual minimisation (top row) or by the MSE for Supervised Regression (bottom row). Training of MLPs is done in over- ($N = 250$, left column) and under-parametrisation ($N = 3000$, right column) setting. One can see that the overall outcome is only qualitatively similar. From a quantitative perspective however, the MLP has a certain freedom in finding the best approximation.



(a) $N = 250$ (b) $N = 3000$

Figure 3.22: Comparison of convergence behaviour for Bellman Residual minimisation and Supervised Regression in the over- ($N = 250$) and under-parametrisation ($N = 3000$) setting. The meaning of the y-axis depends on the method. For regression, it denotes the Mean Squared Error between predictions and target labels. For Bellman Residual minimisation, it is the NMSBE as defined in Equation (3.11).
Supervised Regression remains mostly unaffected by different N, only the width of the shaded area is smaller with more samples. For Bellman Residual minimisation, one can see that the over-parametrisation setting allows for a smaller NMSBE.

therefore as large as it can get, there is still the incentive created by the $N_{net} \times N_{net}$ sub-block in $\widetilde{G}(\mathbf{W})$ to render N_{net} entries in $\widetilde{\Delta}_\pi(\mathbf{W})$ zero. I have conducted experiments similar to that of Section 3.5.4 for larger values of N to see this behaviour of the rank. However, I do not include them in my work due to the required amount of pages to list seemingly identical figures.

I hypothesise that in this mechanism a dependency is residing, which favours a good outcome of optimisation. In the ideal case, the only way for an MLP to render $\widetilde{\Delta}_\pi(\mathbf{W})$ zero is to be the actual value function V_π. This is due to T_π being a contraction in the function space $\mathcal{V}$. Since the usage of more state samples, i.e., larger and larger N, affects positively the approximation of integrals inside the MSBE and the connectivity between states across the entire state space, the issue outlined in Proposition 3 vanishes. The only way to make an MLP learn a wrong function would be to have again significantly more parameters than samples, such that tearing apart the expected rewards for different states becomes possible again. Thus, as long as there are enough parameters N_{net} such that sufficient many unique state samples or components of $\widetilde{\Delta}_\pi(\mathbf{W})$ are enforced to become zero, the Policy Evaluation task can work well enough and, hence, the overall Policy Iteration algorithm is able to produce good policies over time. This line of thought is supported by the training and test errors in Figure 3.11, by the behaviour of Policy Iteration with more training data (cf. Figure 3.19) or the Actor-Critic algorithm (cf. Figure 4.5) in the under-parametrisation setting.

Chapter 4

Analysing the Actor: Extending the Investigation to Parametrised Policies

4.1 Introduction

In this chapter, I extend my analysis of critical points and the application of non-convex optimisation from the MSBE and the Critic-Only approach to the full Actor-Critic regime. The foundation of this chapter is my published paper [Gottwald et al., 2022].

The analysis and possible statements for Actor-Critic algorithms become more complicated compared to Chapter 3, because an Actor uses the Critic, or more precisely, its approximated Q-function, as objective for training. It is important to investigate, what kind of statements are still possible and which components of an algorithm require more care, especially since the Q-function is a non-linear and non-convex function with less pleasing properties than the NMSBE used for training a Critic. For example, by using the Q-function to train an Actor, there will be a more restrictive structure imposed on the reward function. Furthermore, if the Actor is also build around a parametrised policy to handle continuous action spaces, then the domain, on which the task is defined, will cause trouble. Lastly, if a policy is to be trained based on collected states-action tuples, then one obtains a classic regression setting, for which additional challenges for training with respect to the MLP components will show up. Hence, analysing the optimisation task, which forms the Actor-Critic algorithm, and merely writing down the source for troubles is already of great value.

This chapter is organised as follows. Section 4.2 addresses the existing literature. More precisely, Section 4.2.1 gives an overview over possible methods once continuous action spaces are involved. Afterwards, Section 4.2.2 addresses existing realisations and investigations of Actor-Critic algorithms. My insights regarding the Q-function as an objective and its action input are contained in Section 4.3. Section 4.4 extends those insights when involving parametrised policies, i.e., an Actor. Experiments for demonstrating concepts from previous sections numerically are the goal for Section 4.5. I give a quantitative verification for theoretical insights and also show overall capabilities and limitations of an Actor-Critic approach, when using Gauss Newton Residual Gradient optimisation for training the Critic. Finally, in Section 4.6, I address geometrical issues, which I have revealed during the design and investigation of Actor-Critic algorithms. I propose and evaluate a possible solution, which is build around the training of policies as Supervised Regression tasks and exploiting the geometry of a unit ball.

4.2 Existing Methods & Related Work

The purpose of this section is to introduce and define the actual problem, which needs to be faced, once the action space is no longer discrete. In any Dynamic Programming algorithm, one has to be able to extract from a given value- or Q-function an enhanced policy such that one can solve the Policy Improvement step from Equation (2.10). Since there are several equally valid approaches, Section 4.2.1 gives a general overview first. Afterwards, I address in Section 4.2.2 the existing literature. Despite all approaches relying on Q-factors, they naturally split into two categories. Algorithms in the first category tackle the Policy Improvement task by taking a Q-function and searching directly in the action space for optimal values. In the second category, existing work employs a parametrised policy such that it covers the class of Actor-Critic algorithms.

4.2.1 Existing Methods

A main challenge for the realisation of a Dynamic Programming algorithm is the implementation of the Policy Improvement step, which appears either inside Policy Iteration or as the last instruction for Value Iteration. With the help of Q-factors, solving the general form of Policy Improvement as given in Equation (2.10) reduces to the more compact construction $\pi(s) \in \operatorname{argmax}_a Q(s, a)$ as it is defined in Equation (2.15). Given a current Q-function or its approximation, this maximisation is now straightforward to solve from a conceptual viewpoint. Furthermore, if the action space is discrete, Policy Improvement even becomes trivial, as it involves just an $O(|K_{\mathcal{A}}|)$ search. Namely, one iterates over all elements in the set and selects the largest one. Typically, this search is fast enough for reasonable action spaces and in the worst case parallelisation must be employed.

Of course, the obvious approach of discretising continuous action spaces leads to typical combinatorial problems and suffers from an exploding number of combinations. As a numerical example, consider a humanoid robot, which can easily have twenty degrees of freedom. With the coarsest discretisation possible, namely for each joint *turn clockwise*, *idle* and *turn counter clockwise*, this already results in over three billion combinations. Hence, processing the action space directly in its continuous form is mandatory, which in turn renders the Policy Improvement step more demanding. Aside from obtaining optimal continuous actions themself from a Q-function, one also faces new challenges related to the training of parametrised function approximation architectures, which might be needed to represent the space of policies.

Common approaches and algorithms, which can realise a full DP algorithm, form three different categories:

- **Direct-GIP** – One solves the optimisation problem in Equation (2.15) directly at the states one is interested in and obtains for those states optimal actions a^*. The actions can be used directly as action input to a dynamical system or can be inserted in the Optimal Bellman Operator $\mathrm{T_g}$ to realise Q-learning or even Value Iteration in continuous spaces.

- **Fitted-Actor** – The previous approach is employed to create a training set consisting of state-action pairs, which can be used for Supervised Regression of a parametrised policy in a successive but separated step.

- **Actor-Critic** – One introduces a parametrised policy and trains it directly through the (approximated) Q-function. This means, the scalar Q-function is treated as non-linear and non-convex loss surface and is used in conjunction with the chain rule. One creates an improved policy for all states by manipulating the policy parameters with the goal to increase the approximated expected reward.

In the first two scenarios, Direct-GIP and Fitted-Actor, various tools are available, which allow to create a dataset (s_i, a_i^*) with $i = 1, \ldots, N$ for Supervised Regression. A simple yet performant example is derivative free optimisation. The *Cross Entropy Method* (CEM) is a well-known algorithm in that category and can be used to obtain for some given state an optimal action according to the Q-function. An approach which uses CEM treats the Q-function as a black box and returns optimal actions for all states, when given enough computation time. Whereas such a method can work on a large scale, as demonstrated for example with *QT-Opt* in [Kalashnikov et al., 2018], gradient ascent methods should result in better efficiency [Laguna et al., 2006] and are preferable. To exploit gradient information, solving Equation (2.15) is converted to the iteration

$$a \leftarrow a + \alpha \nabla_a Q_\pi(s, a), \tag{4.1}$$

which starts with an initial action a_0 and follows repeatedly a fraction α of the gradient. This has to be done for every state $s \in \mathcal{S}$ one is interested in. The iteration terminates, once a reaches a critical point a^* of Q_π with respect to the action input, assuming one exists. Even second-order optimisation

$$a \leftarrow a + \alpha\big(\mathsf{H}_a\, Q(s, a)\big)^{-1} \nabla_a Q(s, a) \tag{4.2}$$

has been proposed and explored in [Nichols and Dracopoulos, 2014], where the Hessian and Gradient of the Q-function are used together to define a geometry-aware ascent direction. I provide more details on this approach as part of the related work in Section 4.2.2 and during my analysis of critical points of an approximated Q-function in Section 4.3.

The third scenario, Actor-Critic, employs the chain rule to define a gradient with respect to the parameters $\mathbf{U} \in \boldsymbol{\mathcal{U}}$ of a parametrised policy through some Q-function

$$\mathrm{D}_{\mathbf{U}}\, Q\big(s, \pi(\mathbf{U}, s)\big)[h] = \mathrm{D}_2\, Q\big(s, \pi(\mathbf{U}, s)\big) \circ \mathrm{D}_{\mathbf{U}}\, \pi(\mathbf{U}, s)[h]. \tag{4.3}$$

Thus, it expresses equation Equation (4.1) directly in the policy parameter space $\boldsymbol{\mathcal{U}}$. This particular usage of the Q-function is possible and well-behaved as it is described in [Silver et al., 2014] for the *Deterministic Policy Gradient* algorithm. The chain rules is applicable for a Q-function in terms of its definition as expected accumulated discounted return and not only because of the function class used to represent some approximated Q-function. The corresponding iteration then can be written informally as

$$\mathbf{U} \leftarrow \mathbf{U} + \alpha \nabla_{\mathbf{U}} Q_\pi\big(s, \pi(\mathbf{U}, s)\big). \tag{4.4}$$

In all three scenarios, I am interested in an extremum of the (approximated) Q-function with respect to the action input. Hence, it is important to know, how the critical point condition of an approximated Q-function with respect to the action input looks like and what kind of theoretical statements are available. This is my goal for Sections 4.3 and 4.4.

4.2.2 Related Work

Searching for Optimal Actions

As mentioned in Section 4.2.1, a GIP can be realised by searching directly in the action space for optimal actions at a certain state by maximising Q-values. The search process can be realised by computing derivatives of the Q-function with respect to its action input and performing gradient ascent. Lee and Anderson [2014] have explored this approach for first-order gradient ascent. Nichols and Dracopoulos [2014] use second-order optimisation, i.e., Newton's Method, and thus face the typical problems when combining Hessian-based methods with non-convex objectives. Namely, the MLP used to approximate $Q(s, a)$ possesses local minima and even saddle points in $\mathcal{A}$, which will be discovered reliably by Newton's Method. It is required to start from several initial actions and to keep track of the action with largest expected reward observed so far to retrieve proper actions from some Q-function. Furthermore, the optimisation problem is also altered to some extent, because the action space $\mathcal{A}$ has to be a compact subset of a vector space such that DP methods become applicable. To ensure that the action iterates after each ascent step remain in $\mathcal{A}$, it is mandatory to project ascent directions or the iterates themself back to the domain. The authors employ direct component-wise clipping of action vectors with the consequence that there arise new points with artificially zero gradients at the boundary of the action space, where an update direction is perpendicular to the surface and gets removed completely by clipping operations. A follow up work [Nichols, 2016] also compares Newton's Method with other approaches. Among the tested optimisation strategies are derivative based optimisation, discretisation with brute-force search and the Nelder-Mead method. All considered benchmarks contain a one dimensional action space. Thus, neither the required amount of restarts for Newton's Method nor the grid based approach create issues regarding sampling complexity or storage requirements.

For my work, I am also considering derivates of Q-function, but primarily because they arise as part of the chain rule, which is required for implementing Actor-Critic methods. Hence, I face the same problems regarding the existence of gradients or the type of extrema of $Q(s, a)$, which are mentioned by the authors in their work. But different to their paper, in my work these problems are primarily relevant for a parametrised Actor. In the reverse direction, my insights regarding the possibility to apply a Gauss Newton algorithm for training an Actor may carry over to their work. Lastly, I elaborate explicitly on these additional critical points, which arise along the boundary of $\mathcal{A}$, during my description of the Actor-Critic optimisation behaviour and propose a way for handling them.

The idea of solving $\operatorname{argmax}_a Q(s, a)$ directly to obtain a Policy Improvement step has also been explored in various papers. Kalashnikov et al. [2018] introduce with *QT-Opt* a scalable distributed system to build a Reinforcement Learning algorithm, which can learn to grasp objects with a robotic arm based on camera data. The relevant component is their application of *Cross Entropy Method*, which is a black box derivative-free optimisation technique, to retrieve optimal actions for given states according to a Q-function. The authors observe that CEM is already enough to define a policy implicitly through the Q-function and therefore can avoid training an explicit Actor altogether. However, relying on black box optimisation strategies also prevents any form of insight into the optimisation problem itself. Other forms of sampling to retrieve actions have also been explored, for example, *Gibbs Sampling* in [Kimura, 2007].

In [Ryu et al., 2020], *Continuous Action Q-Learning* is proposed, which converts $\max_a Q(s,a)$ into a *Mixed Integer Programming* task to retrieve actions from the Q-function. However, scalability of this technique could remain an issue and prevents a broader application.

Wire-Fitting [Baird and Klopf, 1993] relies on interpolation between pairs consisting of an action input and estimated values of the Q-function for that input. These pairs are called *Wires* and have the pleasing property that the action with largest Q-value is one of the action points with the highest value. The performance of *Wire-Fitting* dependents directly on the number of pairs and has been used by the authors in a memory based RL algorithm. According to [Nichols and Dracopoulos, 2014], one has to trade off the number of *Wires* with the available computational resources. On the one hand, there need to exist sufficiently many *Wires* for an accurate representation of the function. But on the other hand, one also needs to keep the overall computation time feasible. A similar idea is described in [Millán et al., 2002] or [Lazaric et al., 2007]. A weighted combination of actions is maintained and updated according to Q-values with the aim to efficiently retrieve actions with largest Q-values.

Normalised Advantage Functions [Gu et al., 2016] follow the strategy of changing the available function space of the approximation architecture, which is employed to represent Q-factors. The authors render the Policy Improvement step $\operatorname{argmax}_a \tilde{Q}(s,a)$ analytically solvable by imposing a quadratic form on the approximated Q-function, i.e., $\tilde{Q}(s,a) = -1/2(a-\mu(s))^\mathsf{T}\Sigma(s)(a-\mu(s)) + \tilde{V}(s)$. The state dependent functions μ and Σ describe completely the quadratic form and are represented by another MLP next to that for the estimated value function $\tilde{V}$. To compute optimal actions analytically, one just has to evaluate the corresponding terms and define $a^* = \mu(s)$. The matrix Σ allows for more expressiveness during fitting the Q-function, but drops away regarding the action selection such that less computation is required. A similar idea to enforce a quadratic form for Q also has occurred in [Rawlik et al., 2012], but it has been used in a different and unrelated context. Whereas an analytical solution for the optimal action is very pleasing, imposing a quadratic form for the Q-function might result in a bad fit for certain environments.

Input Convex Networks [Amos et al., 2017] provide a general strategy for training NNs. The authors create MLPs $f\colon \boldsymbol{\mathcal{W}} \times \mathcal{X} \to \mathbb{R}$, which possess a convex structure with respect to their arguments $x \in \mathcal{X}$ and therefore allow for an efficient inference over the input domain. In particular, the optimisation task $\operatorname{argmin}_x f(x, \mathbf{W})$ is now convex with all the resulting benefits. A drawback for general applications is that the network can have only a scalar output. Of course, this is not a limitation for the application in RL, where the main task is to estimate Q-factors and to retrieve optimal actions from it. It remains questionable and is problem dependent, whether the achieved convexity is worth the loss of generality for the space of possible function approximations.

Asadi et al. [2021] combine the *Deep Q-Network* from [Mnih et al., 2015] with Radial Basis Functions (RBF) to obtain a DRL method applicable to continuos action spaces. The authors describe how to find approximately optimal actions given the current Q-function. A good approximation for the result of $\operatorname{argmax}_a Q(s,a)$ is defined by the best centroid of the responsible Radial Basis Function component. Furthermore, their method can also be employed together with DDPG. The motivation for their RBF-DQN is that a direct approximation of the Q-function suffers from many local maxima and saddle points. The advantage of the RBF step is, that this addition does not hinder the universal function approximation properties of the underlying Neural Network. Thus, the authors avoid the

limitations of *Normalised Advantage Function* or *Input Convex Networks* and maintain a better applicability.

There is a clear trend in avoiding issues regarding training of an explicit Actor. Many constructions to allow for different kind of RL algorithms have been proposed. But, as pointed out in [Kalashnikov et al., 2018, Lim et al., 2018, Ryu et al., 2020, Asadi et al., 2021], these constructions (e.g., special MLP architectures) come with a loss of generality. Limiting the expressive power of the function approximation method can lead to problems. Hence, I study the optimisation problem when using pure Neural Networks for representing a policy and try to characterise their training behaviour in an unrestricted setting. To remove bad effects on training, one requires a full understanding of the nature of all critical points, which motivates further to investigate the Actor optimisation task from the non-convex optimisation perspective.

Parametrised Policies

The *Deterministic Policy Gradient* (DPG) theorem by Silver et al. [2014] provides the foundation for training Actors directly through a Critic's derivative without the log-likelihood trick. The authors tackle integrals over successor states and introduce regularity requirements with the aim to apply Leibniz's rule for swapping the order of differentiation and integration, and to employ Fubini's theorem for altering the order of integrals. As the result, defining the gradient for Actor parameters through a Critic is possible in general and not only due to the use of a particular function approximation architecture such as Multi-Layer Perceptrons. The extension of DPG towards MLPs is coined *Deep Deterministic Policy Gradient* and investigated from an empirical perspective in [Lillicrap et al., 2015].

By using DPG as foundation for Actor-Critic algorithms, one also avoids using more sophisticated approaches such as *Residual-* and *Stackelberg Actor-Critic* formulations [Wen et al., 2021]. There is no difference between an original loss for the Actor and a varied loss provided by the Critic. Consequently, there is also no need to involve an additional MLP responsible for modelling the difference term between these two losses.

Zhang et al. [2019, 2020a] report a problem with parametrised policies that are trained by maximising value function directly, i.e., the typical definition of infinite horizon expected discounted rewards serves directly as objective. The authors point out shortcomings of the analysis of policy gradient algorithms, and motivate using non-convex optimisation as a toolbox. They elaborate on a direct application of non-convex optimisation for policy parameters and find that it can end up at saddle points or in local extrema with only poor performance. Thus, they also investigate the application of rollouts with randomised lengths and demonstrate that their more sophisticated approach provides an escape mechanism from saddles. In my work, I focus on the setting, where a value- or Q-function is represented by some MLP. Hence, whereas my setting and idea is still related to their work, I face different challenges due to a different objective.

Understanding the behaviour of Actor-Critic algorithms with MLPs for both Critic and Actor is also of high practical demand. In [de Bruin et al., 2015], *Deep Deterministic Policy Gradient*, a DRL algorithm by [Lillicrap et al., 2015], is applied to a robotic arm with two joints. The authors find that if the trainings data does not consist of sample trajectories from the entire state-action space, then the algorithm becomes unstable. This insight confirms my considerations regarding the need of models in Section 2.5.2. Matheron

et al. [2020] address situations, in which the Actor-Critic algorithm DDPG does not show improvement over time, although it is known that DDPG can achieve high performance after careful tuning. The authors work in a toy environment with sparse piecewise constant rewards and outline, why in this scenario DDPG converges sometimes to suboptimal policies. Most importantly, this happens despite a behaviour policy discovering non-zero rewards. In my analysis, I am facing similar problems for sparse and piecewise constant reward signals when analysing the optimisation task for the Actor. Even when working without Semi-Gradients in the Critic and also without exploration mechanisms (i.e., model-based), training is sometimes impossible. Based on my findings for critical points, I propose a basic design principle for the reward function to ensure that a Q-function always maintains a proper geometry to serve as loss for gradient based training of an Actor. This results in even stronger requirements for the reward function than those already necessary according to [Silver et al., 2014].

4.3 Critical Points of the Approximated Q-Function with Respect to Actions

The first step towards extending the critical point analysis from the Critic to an Actor is to address the Q-function alone. Please keep in mind that it is now mandatory to use Q-factors, because relying on a state-only value function would involve derivatives of the reward function and system dynamics with respect to their action inputs. This would open up a new source of potential issues, for example, those described in [Metz et al., 2021] and references therein. Handling these issues is beyond the scope of my thesis.

Since the *Deterministic Policy Gradient* algorithm already provides a foundation for using Q-factors in general for training an Actor, the goal of this section is to gain an understanding for the nature of an approximated Q-function. It is important to make the role of an MLP explicit, because it determines directly how the represented function behaves.

4.3.1 Notation

When working with Actor-Critic algorithms, it becomes necessary to introduce new symbols and notation for the use in this chapter. Most notably, there will be now two MLPs present, where one represents the Q-function and the other serves as policy π. Therefore, all affected expressions come in two variations and receive a corresponding index Q or π. I introduce the new notation already over here, i.e., rather close to the beginning of this chapter, despite it is not required until Section 4.4. My intention is to avoid unnecessary changes in writing throughout this chapter.

First, and as already mentioned, two MLPs can now be involved at the same time. The Q-function is approximated by some $f_Q \in \mathcal{F}(K_\mathcal{S} + K_\mathcal{A}, \ldots, 1)$ with parameters $\mathbf{W} \in \boldsymbol{\mathcal{W}}$. The policy π is represented with $f_\pi \in \mathcal{F}(K_\mathcal{S}, \ldots, K_\mathcal{A})$ and parametrised by $\mathbf{U} \in \boldsymbol{\mathcal{U}}$. Both MLPs have an arbitrary depth or width, which can be different in general. However, for my applications, I keep the amount of hidden layers and their widths synchronised such that the search space for hyper parameters is reduced. Both MLPs use the Bend-Id activation function in hidden layers. The output layer of f_Q is linear, whereas f_π uses normally $tanh(\cdot)$ to match a compact action space. It is up to the environment to scale actions

correctly. In the later document, two differential maps of MLPs will show up. To keep the naming consistent, there will exist some $G_Q(\mathbf{W})$ for the approximated Q-function and a matrix $G_\pi(\mathbf{U})$ for the policy MLP.

Second, the objectives for training MLPs must be distinguished clearly from each other. The NMSBE for continuous state spaces from Equation (3.11) is now denoted by $\mathcal{J}_Q \colon \boldsymbol{\mathcal{W}} \to \mathbb{R}_0^+$ and there will be a new objective for the policy parameters called $\mathcal{J}_\pi \colon \boldsymbol{\mathcal{U}} \to \mathbb{R}$. The image of an objective depends on its usage in an ascent or descent algorithm.

Third, some hyper parameters will show up in two flavours. Currently, this applies to learning rates α, strengths of regularisation for Hessians c and amount of ascent or descent steps i.

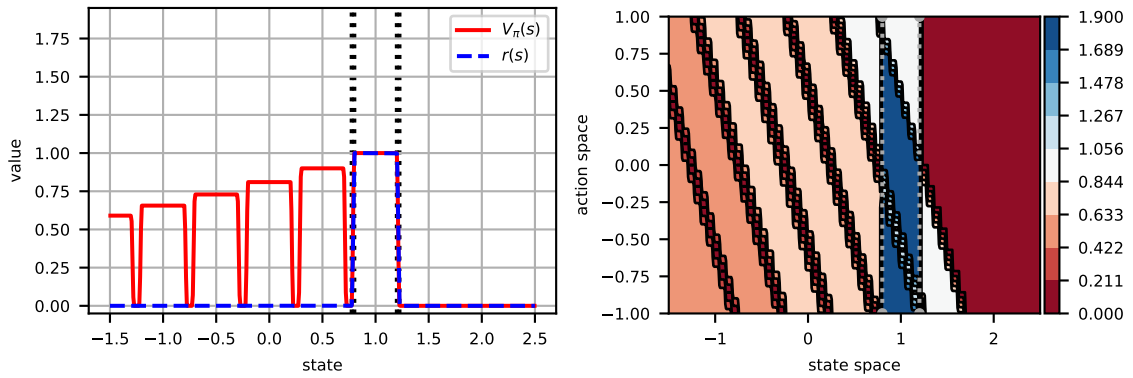
4.3.2 Requirements for the Reward Function

The first component in a DP algorithm, which is responsible for the desired shape and form of a Q-function, is the one-step reward r itself. Matheron et al. [2020] have figured out piecewise constant rewards as a source of problems in DDPG, because Q_π will also be piecewise constant when working in linear and time discrete environments. Their approach is to characterise DDPG's behaviour when working in a full RL setting, which includes exploration mechanisms with behaviour policies, collecting and incorporating off-policy transition tuples and working model-free. Yet, the problem with a piecewise constant Q-function is far more general and exists not only due to using DDPG as an algorithm class. I employ a purely theoretical non-convex optimisation perspective and work model-based to remove the need for exploration. Still, once I have obtained correctly a piecewise constant Q-function, which is matching the reward signal, training an Actor is no longer possible. Hence, the reward function itself must be suitable for MLP training. Normally, when coming from the Dynamic Programming methodology or tabular Reinforcement Learning methods in discrete state-action spaces, there are no important restrictions on the reward function except for being bounded such that Equation (2.11) converges reliably. In my work, where I consider continuous spaces and consequently the use of smooth function approximation in both Actor and Critic, I require proper derivative information from the Critic to adjust the policy MLP parameters. This rules out any sparse and piecewise constant reward function of the form

$$r(s, a, s') = \begin{cases} r_{max} & \text{if} \quad (s, a, s') \in \mathcal{R} \\ 0 & \text{else} \end{cases},$$

where $\mathcal{R}$ denotes a spatially limited reward region in the state space. For such a reward, it can happen that a value- or Q-function is also almost everywhere constant. In particular at the beginning of training, when an initialised policy is producing mostly constant actions, this is likely to happen.

To see this effect, consider the simple one-dimensional linear dynamical system from Section 2.6.2 with the reward function r_3 from Equation (2.37) and the constant policy $\pi(s) = 1$. The reward function is drawn as the blue dashed curve in Figures 2.2 and 4.1a. The value function under the policy is represented by the red line in Figure 4.1a. The contour plot from Figure 4.1b shows the Q-function for the entire state action space. Both V_π and Q_π are ground truths obtained from a Monte Carlo method. Their construction



(a) reward and value function (b) Q-function

Figure 4.1: The value- and Q-function for a piecewise constant reward signal under the policy $\pi(s) = 1$ in a one dimensional (linear) dynamical system. The characteristics of the reward function carries over to the value- and Q-function and prevent the application of gradient based methods for solving $\text{argmax}_a Q(s, a)$. Both V_π and Q_π are ground truths obtained from a Monte Carlo method. Thus, they demonstrate the severity of this issue by showing that also the best possible case is affected negatively by the reward signal.

follows that of ground truths from previous experiments and is described in Section 3.5.1 and also at the end of Section 3.4.2.

Due to the form of the reward function, both V_π and Q_π possess large constant regions. Therefore, it is impossible extract an optimal action for arbitrary states using gradient based methods. In most regions of the state-action space, gradients are zero. As long as the policy does not improve, the value- or Q-function under that policy will not change and in turn, the problem persists. A Policy Iteration algorithm, which is relying on gradient information, is blocked.

The fact that this issue happens even for the ground truth, i.e., no kinds of approximations or an MLP itself are involved, underlines the severity of the issue. The best possible case is that training an MLP works without flaws and that it approximates closely the correct function. Yet, the result is not of any use for solving the Policy Improvement step.

Of course, this also has consequences for Actor-Critic algorithms, where the derivate of a Q-function appears through the chain rule in the training of a policy MLP. When going back to the foundation of deterministic Actor-Critic algorithms [Silver et al., 2014], the reward function r must be continuous such that one can use the chain rule to compute gradients for the Actor via a Q-function. Hence, the application of DDPG in [Matheron et al., 2020] also leaves the realm, where theoretical requirements are satisfied.

Based on the behaviour of ground truth value- or Q-function shown in Figure 4.1, I propose on top of the conditions given in [Silver et al., 2014] to use reward signals, which avoid larger extended regions, in which the reward value stays constant. In one-dimensional environments, this boils down to using (mostly) strictly monotonic functions as the reward. For such rewards, the value- or Q-functions more likely to provide proper improvement directions for actions. Thus, an Actor-Critic training procedure can work more reliably. Unfortunately, it is hardly possible to determine in advance how large a constant region of the reward function may become before it creates a problem for training progress. An

empirical and slightly more sophisticated justification is given as part of the experiments with Actor-Critic algorithms in Section 4.5.2.

4.3.3 Investigation of the Differential Map

A bad reward function on its own can lead to severe problems for Actor-Critic algorithms in terms of having a plethora of critical points without any use. Hence, it is important to know, where critical points of an approximated Q-function with respect to its action input actually originate from and how the critical point condition of the objective in Equation (4.1) looks like. Since the derivative of f_Q will appear later-on as part of the chain rule in the gradient for Actor updates, it is mandatory to avoid frequent loss of information for the Actor due to a bad setting for the Q-function approximation. On top of that, the differential map of the policy MLP f_π will affect significantly the loss surface.

First Steps Let me start with the differential map of the MLP f_Q, which I use to represent a Q-function. Its derivation follows the same steps as in Section 3.3, but this time, the derivative applies to the action input. After defining a direction $h \in \mathcal{A}$, I obtain the familiar pattern

$$\begin{aligned}
\mathrm{D}_a f_Q(\mathbf{W}, s, a)[h] &= \mathrm{D}_2 \Lambda_L(W_L, \phi_{L-1}) \circ \cdots \circ \mathrm{D}_2 \Lambda_1(W_1, \phi_0) \circ \mathrm{D}_a \begin{bmatrix} s \\ a \end{bmatrix} [h] \\
&= \Sigma_L \bar{W}_L^\mathsf{T} \Sigma_{L-1} \bar{W}_{L-1}^\mathsf{T} \cdots \Sigma_1 \bar{W}_1^\mathsf{T} \begin{bmatrix} 0 \\ I_{K_\mathcal{A}} \end{bmatrix} h \\
&= \underbrace{\Psi_1^\mathsf{T}}_{\in \mathbb{R}^{1 \times n_1}} \underbrace{\bar{W}_1^\mathsf{T}}_{\in \mathbb{R}^{n_1 \times (K_\mathcal{S} + K_\mathcal{A})}} \underbrace{\begin{bmatrix} 0 \\ I_{K_\mathcal{A}} \end{bmatrix}}_{\in \mathbb{R}^{(K_\mathcal{S} + K_\mathcal{A}) \times K_\mathcal{A}}} \underbrace{h}_{\in \mathbb{R}^{K_\mathcal{A} \times 1}} \\
&=: C(\mathbf{W}, s, a) h,
\end{aligned} \tag{4.5}$$

where I make use of definitions given after Equation (3.6) or as provided in Appendix A. According to Riesz, I compute the gradient as

$$\mathrm{D}_a f_Q(\mathbf{W}, s, a)[h] = \left\langle \underbrace{\begin{bmatrix} 0 & I_{K_\mathcal{A}} \end{bmatrix} \bar{W}_1 \Sigma_1 \bar{W}_2 \Sigma_2 \cdots \bar{W}_L \Sigma_L}_{=: \nabla_a f_Q(\mathbf{W}, s, a)}, h \right\rangle \tag{4.6}$$

with the corresponding critical point condition

$$\nabla_a f_Q(\mathbf{W}, s, a) = \begin{bmatrix} 0 & I_{K_\mathcal{A}} \end{bmatrix} \bar{W}_1 \Sigma_1 \bar{W}_2 \Sigma_2 \cdots \bar{W}_L \Sigma_L \overset{!}{=} 0. \tag{4.7}$$

When attempting to render Equation (4.7) zero, the design principles of an MLP, which are already required for a proper minimisation of the NMSBE, become active. First, creating a critical point by having any parameter matrix $\bar{W}_i$ becoming zero is not likely to happen. A zero parameter matrix would render the MLP constant, which is not matching a reasonable training task in DP, and because randomly initialised matrices typically have full rank. Second, one cannot rely on Σ_i to create critical points. All Σ_i are diagonal matrices, which contain the output of the truncated MLP on their diagonal. Furthermore, all layers use $\dot{\sigma}$, i.e., the derivative of the activation function. Hence, this second case is directly

prevented by design principles employed for Critic training. Since I choose functions for the non-linearities in an MLP, which are free from vanishing gradients for finite inputs, the individual Σ_i in Equation (4.7) cannot become zero.

Numerical and Visual Confirmation Before spending more effort on investigating all the expressions, I run a quick numerical verification for the existence of critical points. I consider arbitrary Q-functions, which are obtained from MLPs with parameters chosen randomly. More concisely, I take an MLP $f_Q \in \mathcal{F}(4, 10, 10, 1)$ with Bend-Id and a single sampled state s to represent some Q-function. It has four input dimensions to process a hypothetical two dimensional state space and action vectors with two components, thus $K_\mathcal{S} = K_\mathcal{A} = 2$. I evaluate $f_Q(s, a)$ with a selected from a 175×175 grid in the range $[-1.1, 1.1]^2$. This domain is larger than the intended action space $\mathcal{A} = [-1, 1]^2$, which allows to see without problems the behaviour of f_Q near and beyond the boundary. The visualisations of Q-functions as contour plot for two different random initialisations are contained in Figure 4.2. The boundary of the action space is shown as dotted line. Gradients of $f_Q(s, a)$ with respect to a are computed using Equation (4.6) and are drawn in Figures 4.2c and 4.2d with a zoom-in centred around the shortest vectors.

In Figure 4.2a, a candidate for a critical point is present in the region, where isolines are rather far away. Yet, when looking at the vector field on a smaller scale as done in Figure 4.2c, one finds a clear direction towards the interior of $\mathcal{A}$. The gradients are not close to zero, not even from a numerical perspective when taking limited machine precision into account. In such a situation, optimal actions satisfying $\operatorname{argmax}_a Q(s, a)$ are located at the boundary. In Figure 4.2a, those actions are located roughly at the upper middle section. This property gives rise to additional and severe issues for any geometric optimisation approach. The requirement to stay in $\mathcal{A}$ makes the application of a projection operation to the actions or gradients mandatory. However, these projectors also open a new way to create artificially zero gradients. Hence, a gradient based algorithm for Direct-GIP will now have new solutions, which are not consistent with those of an unconstrained task. Since these vanishing gradients originate exclusively from projections, I call them in the following "spurious" critical points and handle them differently than their natural counterparts. An example, where such a spurious critical point would show up, can be seen in Figures 4.2a and 4.2c close to the centre of the upper boundary. I cover the issues related to spurious critical points on their own in Section 4.6.

After some repeated initialisations, I have been able to produce a Q-function approximation, which has its optimal action as maximum in $\mathcal{A}$, as one can see in Figure 4.2b. Since only a compact subset of the input domain to the MLP is covered, it is not possible to decide, whether this action corresponds only to a local or indeed a global maximum. On the other hand, this does not matter for a DP application, since I only need statements about the action space itself and the action with the largest Q-value in that domain. Lastly and as a side note, care must be taken to not find saddle points. I have not observed them so far, but nothing prevents them from occurring.

In conclusion, if proper critical point inside the action space exists, as it can be seen in the numerical example from Figure 4.2b, but the individual expressions are unlikely to vanish on their own, then there must be another source of solutions to Equation (4.7). A not that obvious candidate would be null spaces, which would provide the natural critical points inside $\mathcal{A}$. For the rest of this section and for Section 4.4, I proceed only with these

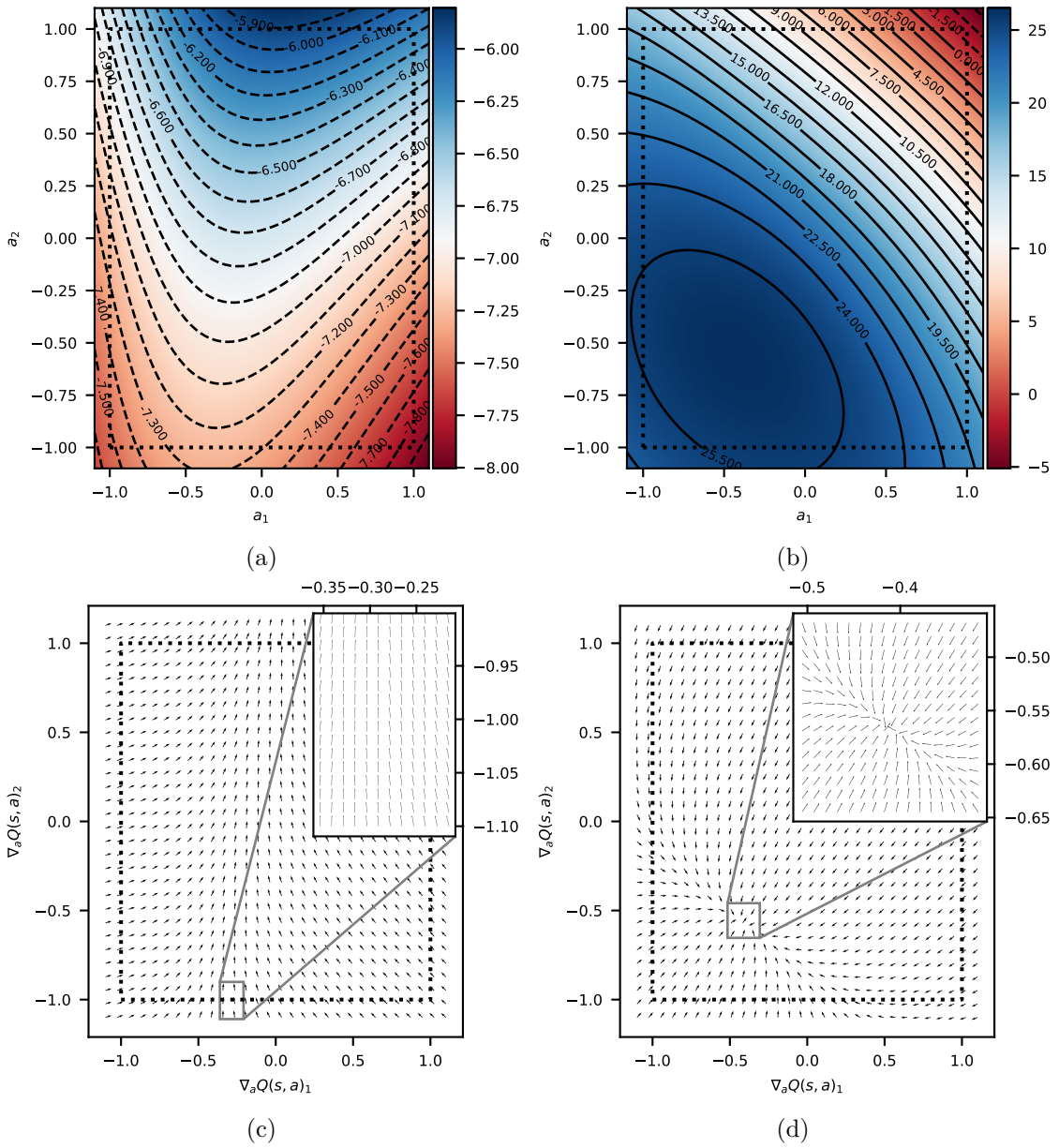


Figure 4.2: The Q-function as error surface. The MLP $\mathcal{F}(K_{\mathcal{S}} + K_{\mathcal{A}}, \ldots, 1)$ is initialised randomly and uses a single fixed state $s \in \mathcal{S}$. The action input is determined by a grid in the range $[-1.1, 1.1]^2$. The black dotted line surrounds the actual action space $\mathcal{A} = [-1, 1]^2$. The evaluation of the MLP is done also outside of $\mathcal{A}$ to reveal the behaviour around the boundary of $\mathcal{A}$. Gradients lengths are upper bounded to enhance visibility. **a) and c)** The Q-function with a critical point outside of $\mathcal{A}$ and the corresponding gradient vector field $\nabla_a Q(s, a)$. **b) and d)** Similarly, but the critical point is part of $\mathcal{A}$.

natural critical points. Section 4.6 is then dedicated to their spurious counterparts and introduces a possible method to handle them correctly.

Critical Points due to Null Spaces A mechanism to render Equation (4.7) zero, which is not that obvious, is that a vector falls into the null space of a matrix. An apparat candidate for a null space is the matrix $\begin{bmatrix} 0 & I_{K_\mathcal{A}} \end{bmatrix}$. For the sake of convenience, I call this term in the following zero-one matrix. This matrix is guaranteed to have a null space, because its rank is only $K_\mathcal{A}$. Hence, the question arises, whether the column vector

$$A \coloneqq \bar{W}_1 \Sigma_1 \cdots \bar{W}_L \Sigma_L \in \mathbb{R}^{(K_\mathcal{S}+K_\mathcal{A})\times 1}$$

is an element of it. To answer it, first introduce the partitioning

$$A = \begin{bmatrix} \overline{A} \\ \underline{A} \end{bmatrix} \begin{matrix} \} \in \mathbb{R}^{K_\mathcal{S}\times 1} \\ \} \in \mathbb{R}^{K_\mathcal{A}\times 1} \end{matrix}$$

such that one obtains the condition

$$\begin{bmatrix} 0 & I_{K_\mathcal{S}} \end{bmatrix} \cdot \begin{bmatrix} \overline{A} \\ \underline{A} \end{bmatrix} = 0\overline{A} + I_{K_\mathcal{A}}\underline{A} = \underline{A} \overset{!}{=} 0.$$

Here, one can see that the only way to satisfy the condition is that the term $\underline{A}$ still needs to vanish on its own. Creating a zero in the trivial way through the parameter matrices or derivatives of activation functions contained in Σ_i is not possible. The aforementioned design principles for MLPs or the typical use cases, i.e., non-zero weight matrices, prevent this method. The zero-one matrix only takes care of the upper part of A, which belongs to the state input. Thus, this is not enough.

The usage of homogenous inputs $\tilde{\phi} = \begin{bmatrix} \phi & 1 \end{bmatrix}$ causes parameter matrices to be non-square even if one uses in all hidden layers the same number of units. In their transposed usage, there is one more column than rows, indicating that there must be a null space in which a vector could fall. But as one can already see in Equation (4.7), only the truncated matrices $\bar{W}_i$ without transpose play a role. Hence, the actual matrices that contribute to the critical point condition do not have a null space just by adding the bias vector to them. Furthermore, hidden layers should be of same size according to design principles in [Shen, 2018b] to prevent additional loss of rank for the matrix $\tilde{G}(\mathbf{W})$. Thus, for all hidden matrices, I have $\bar{W}_l \in \mathbb{R}^{n_{l-1}\times n_l}$ with $n_{l-1} = n_l$. So there is no further null space, if I assume that the matrices themselves are of full rank, as I already have done for the chapter with the theoretical investigation for a Critic. Because the last matrix $\bar{W}_L$ has the shape $n_{L-1} \times 1$, it again ensures that there is no useful null space according to its definition. There are more rows than columns.

The matrices Σ_l are diagonal for all l with non-zero entries due to proper activation functions and compatible shapes. Hence, they do not contribute anything and are out of interest.

The actual important null space is present in the matrix W_1, where the input dimension gets boosted to the hidden layer size. Normally, this null space has to exist, because the first layer should lift the input dimension of the MLP to a higher value. To see how this null space does affect the critical point condition, one should investigate more closely the actual values used to produce Figure 4.2. I increase with $\bar{W}_1 \in \mathbb{R}^{4\times 10}$ the four dimensional state-action input to a ten dimensional latent space, hence there is a six dimensional basis

for a null space. In Figure 4.2d, the critical point is located at $a^* = \begin{bmatrix} -0.41 & -0.56 \end{bmatrix}^\mathsf{T}$. With that action one can compute the column vector $B = \Sigma_1 \bar{W}_2 \dots \bar{W}_L \Sigma_L \in \mathbb{R}^{10\times 1}$ such that one obtains $A = \bar{W}_1 B = \begin{bmatrix} * & * & 0 & 0 \end{bmatrix}^\mathsf{T}$. One can see that B resides indeed in the null space of $\bar{W}_1$ and that the first two non-zero components (denoted by $*$) get removed by the zero-one matrix. Thus, I have critical points due to a null space, which exists in the truncated parameter matrix $\bar{W}_1$.

It is an open question, whether solutions for the critical point condition, which arise from null spaces, can be exploited in the context of DP. Since such solutions are not necessarily defined exactly, one can use a basis for the null space and move around freely without affecting the outcome. This opens up the possibility to pick an arbitrary critical point out of all available ones with other tasks in mind, e.g., as it has been proposed with *Safe Regularization* for general Neural Network training in [Kissel et al., 2020]. But for the application in DP, the MLP parameters are fixed and only the two dimensional action input can be varied. Hence, it is not clear how to move in the null space of $\bar{W}_1$, if one can only control the input to an MLP. Whatever value is inserted gets processed non-linearly to produce this B vector. From the surface plot in Figure 4.2b, there is also a numerical hint that the set of critical points does not consists of an extended region in the action space. Hence, an another interesting question regarding the applicability of *Safe Regularization*, is, whether the set of critical points is indeed a point. However, this question leaves the scope of my thesis.

Hessian Motivated by the work from [Nichols and Dracopoulos, 2014], let me take a look at the Hessian of the MLP f_Q used to approximate a Q-function. If it has been possible to derive and use a Newton's Method (cf. Equation (4.2)), maybe I can introduce further simplifications or even arrive at an Approximated Newton Algorithm. Thus, I start from Equation (4.5) and apply the derivative with respect to the action a second time to obtain the unwieldy term

$$
\begin{aligned}
&\mathrm{D}_a \left(\mathrm{D}_a \, f_Q(\mathbf{W}, s, a)[h_1] \right)[h_2] = \mathrm{D}_a \left(C(\mathbf{W}, s, a) h_1 \right)[h_2] \\
&\quad = \mathrm{D}_a \, \Psi_1(s,a)^\mathsf{T}[h_2] \bar{W}_1^\mathsf{T} \begin{bmatrix} 0 \\ I_{K_\mathcal{A}} \end{bmatrix} h_1 \\
&\quad = \sum_{l=1}^{L} \Psi_{L,l}^\mathsf{T} \, \mathrm{D}_a \, \Sigma_l[h_2] \bar{W}_l^\mathsf{T} \Psi_{l-1,0}^\mathsf{T} \begin{bmatrix} 0 \\ I_{K_\mathcal{A}} \end{bmatrix} \\
&\quad = \dots \\
&\quad = h_2^\mathsf{T} \begin{bmatrix} 0 & I_{K_\mathcal{A}} \end{bmatrix} \sum_{l=1}^{L} \Psi_{l-1,0} \bar{W}_l \dot{\Sigma}_l \operatorname{diag}\left(\Psi_{L,l}^\mathsf{T} \right) \bar{W}_l^\mathsf{T} \Psi_{l-1,0}^\mathsf{T} \begin{bmatrix} 0 \\ I_{K_\mathcal{A}} \end{bmatrix} h_1 \\
&\quad = \langle h_2 \, | \mathsf{H}_a \, Q(\mathbf{W}, s, a) | \, h_1 \rangle \, ,
\end{aligned} \tag{4.8}
$$

where the definitions of $\Psi_{f,i}$ are not inserted for the sake of visibility. Opposed to the original definition of Ψ_l after Equation (3.6), the term $\Psi_{f,i}$ with two indices *final* and *initial* now denotes an arbitrary segment in the sequence of Σ and $\bar{W}$ matrices. It holds

$$
\Psi_{f,i} = \bar{W}_{i+1}^\mathsf{T} \Sigma_{i+1} \cdot \bar{W}_{i+2}^\mathsf{T} \Sigma_{i+2} \cdot \dots \cdot \bar{W}_{f-1}^\mathsf{T} \Sigma_{f-1} \cdot \bar{W}_f^\mathsf{T} \Sigma_f
$$

with Σ_l and $\bar{W}_l$ defined as before. The indices i and f satisfy $i \leq f$. To avoid confusion, consider the following examples

$$\begin{aligned}\Psi_{L,1} &= \bar{W}_2^\mathsf{T}\Sigma_2 \cdots \bar{W}_L^\mathsf{T}\Sigma_L \\ \Psi_{L-1,0} &= \bar{W}_1^\mathsf{T}\Sigma_1 \cdots \bar{W}_{L-1}^\mathsf{T}\Sigma_{L-1} \\ \Psi_{l+1,l-1} &= \bar{W}_l^\mathsf{T}\Sigma_l \cdots \bar{W}_{l+1}^\mathsf{T}\Sigma_{l+1}.\end{aligned}$$

As a special case, it applies $\Psi_{L,L} = I$, where the shape of the identity matrix is given by the MLP architecture. Please also note that $\dot{\Sigma}_l$ in Equation (4.8) contains the second-order derivatives of the activation function on its diagonal for layer l. The counter intuitive notation arises from the original definition of Σ, where it has been easier to abbreviate directly the first derivatives of the activation functions.

Since the only mechanism for creating critical points is using the null space of $\bar{W}_1$, there is no useful way to achieve a simplification of the Hessian at critical points as required for a Gauss Newton approximation. The only hint at a minor approximation would be for layer L, i.e, the last term in the summation.

4.3.4 Interpretation and Implications

During my investigation in this section, I have unveiled the source of critical points with respect to the action inputs of an MLP, which is used to approximate a Q-function. Furthermore, I have outlined how to control them to some extend. One can reduce the amount of critical points by following the design principles for MLPs as they are required for training the Critic by minimising the NMSBE. Additionally, the reward function should avoid having constant areas. Therefore, less suboptimal outcomes for Policy Improvement exist in general, because fewer local maxima are present, where first-order gradient ascent algorithms would get stuck.

Of course, this is not a general guarantee for a performant algorithm, because local maxima are not excluded entirely. There remains the typical struggle of gradient based methods with non-convex objectives. I expect that following the gradient uphill to maximise for some initial action its Q-value is working reliably enough for practical applications.

Additionally, first-order methods will suffer from flat surfaces such that more sophisticated methods like momentum based optimisation should be employed. Alternatively, second-order optimisation works as well to overcome flat regions, but will also seek actively for arbitrary critical points, including minima and saddle points. Since I do not have the same full control over the Q-function as over the loss when minimising the NMSBE, I cannot eliminate completely this bad behaviour of a Newton's Method at this point. The outcome for Policy Improvement depends completely on the Hessian of the Q-function at critical points with respect to the action input and, therefore, on the curvature of the surface. Thus, similar to [Nichols and Dracopoulos, 2014], I expect that multiple restarts are required. In the worst case, the optimisation task is not solvable with second-order optimisation. Because of the Hessian, the optimisation process moves actively away from larger Q-values across the entire action space.

Thanks to the required projections to keep the iterates inside of the action space, one should be able to ensure that there are always critical points available in the action space. Either in the form of natural critical points inside of $\mathcal{A}$, or as spurious ones, which are located on the boundary. Unfortunately, they imply a strong manipulation

of the optimisation task and manipulate the optimisation behaviour. Whereas this is not a limiting factor for the Direct-GIP approach, these considerations carry over to an Actor-Critic formulation and will affect the training of the policy MLP. Hence, for the remaining part of chapter, I am interested in how exactly spurious critical points interact with the parametrised policy. I will extend the critical point analysis from Chapter 3 for the NMSBE to the full Actor-Critic regime and check, whether an Approximated Newton algorithm can be constructed directly in the policy parameter space.

4.4 Combining the Q-function with Parametrised Policies

A common approach in (Deep) RL is to combine an MLP for estimating the Q-function with a second MLP for approximating the policy π. This pattern of an algorithm is called Actor-Critic, which also can be seen as a possible realisation of Approximate Policy Iteration as it is defined in Equation (2.16). An Actor, which corresponds to a parametrised policy, is trained by a Critic or the Q-function to increase its performance. More precisely, optimising the policy MLP is implemented by maximising the Q-function for all states. The Critic is obtained as usual by minimising the NMSBE. It realises the Policy Evaluation step in Approximate Policy Iteration.

Due to the usage of Q-factors, the loss for Critic part is now slightly different compared to the existing analysis from Chapter 3. I now use the TD-error δ_Q from Equation (2.18) for defining the root finding problem and, consequently, evaluate the MLP f_Q for state-action tuples instead of states alone. The resulting vector takes the form $F_Q(\mathbf{W}) := [f_Q(\mathbf{W}, s_1, a_1) \dots f_Q(\mathbf{W}, s_N, a_N)]^\mathsf{T} \in \mathbb{R}^N$. With F_Q, I write the difference of both sides in Equation (2.12) in vector form as

$$\Delta_\pi^Q(\mathbf{W}) = F_Q(\mathbf{W}) - R_\pi - \gamma F_Q'(\mathbf{W}). \tag{4.9}$$

The term F_Q' denotes the evaluation of f_Q for all successor states s_i' and actions a_i'. Successor states originate from the system dynamics evaluate for the current tuple (s, a). The corresponding actions are provided by the current Actor $a_i' = f_\pi(\mathbf{U}, s_i')$. All one-step rewards are contained as usual in R_π. Finally, I can formulate with the help of Equation (4.9) the Neural Mean Squared Bellman Error in Q (NMSBEQ) as

$$\mathcal{J}_Q(\mathbf{W}) = \frac{1}{2N} \Delta_\pi^Q(\mathbf{W})^\mathsf{T} \Delta_\pi^Q(\mathbf{W}). \tag{4.10}$$

This is the loss corresponding to the Critic and required for training the MLP, which represents a Q-function. All insights from Chapter 3 apply as before.

4.4.1 Critical Points for an Actor

For analysing the critical points of the objective used to train an Actor, it is a mandatory first step to define concisely the optimisation task related to the Actor. This optimisation task makes use of a given MLP f_Q and its parameters to define an update direction for parameters sitting in f_π.

Before continuing, please pay attention to an omitted technical complication. For my analysis in the rest of this section and also during the related experiments in Section 4.5, I ignore these spurious critical points, which have been introduced in Section 4.3. For now,

everything is assumed to behave nicely. MLPs have the complete Euclidean space as input domain for actions such that projections to remain inside of the compact subset $\mathcal{A}$ are not required from the implementation viewpoint. The geometrical issues and thus spurious critical points themself are then covered in Section 4.6.2.

Given a set of sample states $s_i \in \mathcal{S}$, the loss for training an Actor is given by

$$\begin{aligned}\mathcal{J}_\pi(\mathbf{U}) =& \frac{1}{N}\sum_{i=1}^{N} f_Q(\mathbf{W}, s_i, f_\pi(\mathbf{U}, s_i)) \\ =& \frac{1}{N}\begin{bmatrix} 1 & \dots & 1 \end{bmatrix}\begin{bmatrix} f_Q(\mathbf{W}, s_1, f_\pi(\mathbf{U}, s_1)) \\ \vdots \\ f_Q(\mathbf{W}, s_N, f_\pi(\mathbf{U}, s_N)) \end{bmatrix} \\ =:& \frac{1}{N}\mathbb{1}^\intercal \cdot F_{all}(\mathbf{W}, \mathbf{U}). \end{aligned} \tag{4.11}$$

The goal is to maximise the approximated expected reward for all sample states s_i according to the MLP f_Q. This can be achieved by gradient ascent in the policy parameter space.

To arrive at the related critical point condition, one has to compute the differential map of Equation (4.11) with respect to the policy parameters $\mathbf{U} \in \boldsymbol{\mathcal{U}}$. I yield

$$\begin{aligned}\mathrm{D}_\mathbf{U}\, \mathcal{J}_\pi(\mathbf{U})[\mathbf{H}] =& \frac{1}{N}\sum_{i=1}^{N} \mathrm{D}_\mathbf{U}\, f_Q(\mathbf{W}, s_i, f_\pi(\mathbf{U}, s_i))[\mathbf{H}] \\ =& \frac{1}{N}\mathbb{1}^\intercal\, \mathrm{D}_\mathbf{U}\, F_{all}(\mathbf{W}, \mathbf{U})[\mathbf{H}], \end{aligned}$$

where $\mathbf{H} \in \boldsymbol{\mathcal{U}}$ is a direction in the policy parameter space. For now, I take a single state s out of all sampled ones and the known differential map of f_Q, as it has been computed in Equation (4.5). Then, I exploit the chain rule and insert everything to obtain

$$\begin{aligned}&\mathrm{D}_\mathbf{U}\, f_Q(\mathbf{W}, s, f_\pi(\mathbf{U}, s))[\mathbf{H}] = \mathrm{D}_a\, f_Q(\mathbf{W}, s, f_\pi(\mathbf{U}, s)) \circ \mathrm{D}_\mathbf{U}\, f_\pi(\mathbf{U}, s)[\mathbf{H}] \\ =& C(\mathbf{W}, s, f_\pi(\mathbf{U}, s)) \underbrace{\begin{bmatrix} \Phi_1^\intercal\, (I_{n_1} \otimes \phi_0)^\intercal & \cdots & \Phi_L^\intercal\, (I_{n_L} \otimes \phi_{L-1})^\intercal \end{bmatrix}}_{=:G_\pi^{(s)}(\mathbf{U}) \in \mathbb{R}^{K_\mathcal{A} \times N_{net}}} \begin{bmatrix} \mathrm{vec}(H_1) \\ \vdots \\ \mathrm{vec}(H_L) \end{bmatrix} \\ =& C(\mathbf{W}, s, f_\pi(\mathbf{U}, s)) G_\pi^{(s)}(\mathbf{U})\, \mathrm{vec}(\mathbf{H}). \end{aligned} \tag{4.12}$$

I denote by Φ_l the same expressions as Ψ_l, but use the parameter matrices U_l of the policy MLP. The definition of Ψ_l is provided in the text after Equation (3.6). Combining Equation (4.12) for all states results in

$$\begin{aligned}\mathrm{D}_\mathbf{U}\, \mathcal{J}_\pi(\mathbf{U})[\mathbf{H}] =& \frac{1}{N}\sum_{i=1}^{N} \underbrace{C(\mathbf{W}, s_i, f_\pi(\mathbf{U}, s_i))}_{=:C_i} \underbrace{G_\pi^{(s_i)}(\mathbf{U})}_{=:G_i}\, \mathrm{vec}(\mathbf{H}) \\ =& \frac{1}{N} \underbrace{\begin{bmatrix} C_1 & \cdots & C_N \end{bmatrix}}_{=:C(\mathbf{W},\mathbf{U}) \in \mathbb{R}^{1 \times N \cdot K_\mathcal{A}}} \underbrace{\begin{bmatrix} G_1 \\ \vdots \\ G_N \end{bmatrix}}_{=:G_\pi(\mathbf{U}) \in \mathbb{R}^{N \cdot K_\mathcal{A} \times N_{net}}} \mathrm{vec}(\mathbf{H}) \\ =& \frac{1}{N} C(\mathbf{W}, \mathbf{U}) G_\pi(\mathbf{U})\, \mathrm{vec}(\mathbf{H}). \end{aligned} \tag{4.13}$$

A critical point condition for the policy parameters now manifests itself in

$$\nabla_{\mathbf{U}} \mathcal{J}_\pi(\mathbf{U}) \coloneqq \frac{1}{N} G_\pi(\mathbf{U})^\mathsf{T} C(\mathbf{W}, \mathbf{U})^\mathsf{T} \stackrel{!}{=} 0. \tag{4.14}$$

As a first insight from Equation (4.14), the condition is fulfilled whenever $C(\mathbf{W}, \mathbf{U})$ becomes zero, i.e., the action produced by f_π is an extremum of f_Q. This is a pleasing property, because finding extrema is the goal for a GIP from Equation (2.15) and, apparently, the objective for the Actor complies to it. Constellations, in which the differential map of the approximate Q-function $C(\mathbf{W}, \mathbf{U})$ vanishes, have already been considered in Section 4.3.3.

Analogously to the analysis in Chapter 3, sufficiently large MLPs allow also the Actor part for exact learning based on a similar reasoning as in Section 3.3.2. Hence, I introduce an alternated version of Definition 1 for the Actor, which makes the interplay between f_Q and f_π concise.

Definition 2 (Extremal policy approximator)**.** *Let $f_Q \in \mathcal{F}(K_\mathcal{S} + K_\mathcal{A}, \ldots, 1)$ be an approximate evaluation of some current policy $\pi \colon \mathcal{S} \to \mathcal{A}$. Given N samples $(s_i, a_i) \in \mathcal{S} \times \mathcal{A}$, one calls an MLP $f_\pi \in \mathcal{F}(K_\mathcal{S}, \ldots, K_\mathcal{A})$, which satisfies*

$$f_\pi(\mathbf{U}, s_i) = a_i \quad \textit{such that} \quad \nabla_a f_Q(s_i, a_i) = 0 \quad \forall i = 1, \ldots, N$$

for some policy parameters $\mathbf{U} \in \boldsymbol{\mathcal{U}}$, an extremal policy approximator of a greedily induced policy based on the N sampled tuples.

This definition simply states that training an Actor corresponds to finding a policy, which produces actions that are critical points of an approximated Q-function. Due to the more complicated nature of the Q-function as opposed to the typical loss from the last chapter, one cannot tell in general, whether the MLP f_π produces indeed the (local) maximum, or just some action, which might be a saddle or local minimum of f_Q. Thus, the considerations given in Section 4.3.4 apply over here as well. Namely, going uphill in the parameter space of the policy MLP typically works and saddles or minima are not a limiting factor from an engineering perspective. There are just no guarantees that a (global) maximum is inside action space and that it is reached. Next, it is possible to introduce a slightly varied formulation of Assumption 2 for a policy MLP.

Assumption 3 (Existence of extremal policy approximator)**.** *Let $f_Q \in \mathcal{F}(K_\mathcal{S} + K_\mathcal{A}, \ldots, 1)$ be an approximate evaluation of some current policy $\pi \colon \mathcal{S} \to \mathcal{A}$. Given N unique sample $(s_i, a_i) \in \mathcal{S} \times \mathcal{A}$, there exists at least one MLP architecture $\mathcal{F}(K_\mathcal{S}, \ldots, K_\mathcal{A})$ as defined in Equation (2.24) together with a set of parameters $\mathbf{U} \in \boldsymbol{\mathcal{U}}$, such that the MLP $f_\pi(\mathbf{U}, \cdot) \in \mathcal{F}(K_\mathcal{S}, \ldots, K_\mathcal{A})$ is an extremal policy approximator for f_Q according to Definition 2.*

Finally, based on Definition 2 and Assumption 3, a new proposition to characterise the nature of the optimisation task in the Actor becomes available.

Proposition 6 (Proper solution condition for an Actor)**.** *Let an MLP architecture $f_\pi \in \mathcal{F}(K_\mathcal{S}, \ldots, K_\mathcal{A})$ satisfy Assumption 3. If the rank of the matrix $G_\pi(\mathbf{U})$ as defined in Equation (4.13) is equal to $N \cdot K_\mathcal{A}$ for all $\mathbf{U} \in \boldsymbol{\mathcal{U}}$, then any extremum $\mathbf{U}^* \in \boldsymbol{\mathcal{U}}$ of $\mathcal{J}_\pi$ corresponds to an extremal policy approximator according to Definition 2.*

Proof. The objective for the Actor defines a linear equation system in terms of $G_\pi(\mathbf{U})$ and $C(\mathbf{W}, \mathbf{U})$, c.f. Equation (4.14). If the rank of $G_\pi(\mathbf{U})$ is full, then the matrix enforces the trivial solution for $C(\mathbf{W}, \mathbf{U})$. Since $C(\mathbf{W}, \mathbf{U})$ is a factor in the differential map of f_Q, any critical point of the $\mathcal{J}_\pi(\mathbf{U})$ implies an extremal policy approximation. □

Unfortunately, the more complex nature of a Q-function, or to be more precise, the MLP used to approximate this Q-function, allows for any type of extremum. Hence, the proposition is weaker than its counterparts for the Critic in Section 3.3. There are no longer global statements about the quality of the outcome of optimisation. One only gets actions, for which it is known that they are critical point of f_Q in the best case. This is necessary to solve $\operatorname{argmax}_a Q(s,a)$, but not itself sufficient.

A full rank is possible as soon as I employ over-parametrised MLPs, since I need enough columns for the given amount of rows. But because the policy has a vector valued output, it is a harder problem than the approximation of a scalar Q-function. Instead of having only more parameters than samples ($N_{net} > N$, c.f. Equation (3.19)), I also have to take the dimension of the action space into account and require f_π to satisfy

$$N_{net} \geq N \cdot K_{\mathcal{A}}. \tag{4.15}$$

I do no longer have a full control over the outcome of Actor training as when learning a Critic. This is due to the Q-function, which imposes as non-linear and non-convex function an arbitrarily complicated geometry on the loss surface. Therefore, let me continue with the analysis of the curvature to see what kind of statements are still possible. As for the Q-function, the term $G_\pi(\mathbf{U})$ is the Jacobian of the MLP representing a policy when evaluated for all sample states. Its structure and definition is analogue to Equation (3.6). The auxiliary expression $C(\mathbf{W},\mathbf{U})$ contains derivatives of the Critic, or, in other words, the change of the Q-function with respect to the action inputs. Using these abbreviations, I can go on with the second-order differential maps. My goal is to introduce a Gauss Newton approximation of the Hessian also in the Actor part, but due to using the more complex Q-function as objective, I expect this to be a bit less informative. I have

$$\begin{aligned}\mathrm{D}_{\mathbf{U}}\Big(\mathrm{D}_{\mathbf{U}}\,\mathcal{J}_\pi(\mathbf{U})[\mathbf{H}_1]\Big)[\mathbf{H}_2] =& \frac{1}{N}\Big(\mathrm{D}_{\mathbf{U}}\,C(\mathbf{W},\mathbf{U})[\mathbf{H}_2]G_\pi(\mathbf{U})\,\mathrm{vec}(\mathbf{H}_1)\\ &+ C(\mathbf{W},\mathbf{U})\,\mathrm{D}_{\mathbf{U}}\left(G_\pi(\mathbf{U})\,\mathrm{vec}(\mathbf{H}_1)\right)[\mathbf{H}_2]\Big).\end{aligned} \tag{4.16}$$

Here, I already see that the term $C(\mathbf{W},\mathbf{U})$, which sits in front of the second-order differentials of the policy, still vanishes at critical points $\mathbf{U}^*$ by design, if I assume that the policy MLP is rich enough to achieve exact learning and that Proposition 6 holds. Hence, I am able to proceed with the GN approximation at critical points

$$\mathrm{D}_{\mathbf{U}}\Big(\mathrm{D}_{\mathbf{U}}\,\mathcal{J}(\mathbf{U}^*)[\mathbf{H}_1]\Big)[\mathbf{H}_2] = \frac{1}{N}\,\mathrm{D}_{\mathbf{U}}\,C(\mathbf{W},\mathbf{U}^*)[\mathbf{H}_2]G_\pi(\mathbf{U}^*)\,\mathrm{vec}(\mathbf{H}_1). \tag{4.17}$$

To obtain the Hessian with respect to $\mathbf{U}$, I need the expressions

$$\mathrm{D}_{\mathbf{U}}\,C(\mathbf{W},s,f_\pi(\mathbf{U},s))[\mathbf{H}] = \mathrm{D}_a\,C(\mathbf{W},s,f_\pi(\mathbf{U},s)) \circ \mathrm{D}_{\mathbf{U}}\,f_\pi(\mathbf{U},s)[\mathbf{H}],$$

where $\mathrm{D}_a\,C(\mathbf{W},s,a)[h]$ is computed in Equation (4.8) and $\mathrm{D}_{\mathbf{U}}\,f_\pi(\mathbf{U},s)\mathbf{H}$ follows the same steps as in Equation (4.12). Once they are inserted in the corresponding sum of Equation (4.13), I can complete Equation (4.17) and get

$$\begin{aligned}\mathrm{D}_{\mathbf{U}}\Big(\mathrm{D}_{\mathbf{U}}\,\mathcal{J}_\pi(\mathbf{U}^*)[\mathbf{H}_1]\Big)[\mathbf{H}_2] &= \mathrm{vec}(\mathbf{H}_2)^\mathsf{T}\underbrace{\frac{1}{N}G_\pi(\mathbf{U}^*)^\mathsf{T}H(\mathbf{U}^*)G_\pi(\mathbf{U}^*)}_{=:\mathsf{H}_{\mathbf{U}}\,\mathcal{J}_\pi(\mathbf{U}^*)}\mathrm{vec}(\mathbf{H}_1)\\ &= \mathrm{vec}(\mathbf{H}_2)^\mathsf{T}\,\mathsf{H}_{\mathbf{U}}\,\mathcal{J}_\pi(\mathbf{U}^*)\,\mathrm{vec}(\mathbf{H}_1),\end{aligned} \tag{4.18}$$

where the symmetrical block diagonal matrix $H(\mathbf{U}^*)$ contains the Hessians of f_Q for all sample states and actions

$$H(\mathbf{U}^*) = \begin{bmatrix} \mathsf{H}_a\, f_Q(\mathbf{W}, s_1, f_\pi(\mathbf{U}^*, s_1)) & & \\ & \ddots & \\ & & \mathsf{H}_a\, f_Q(\mathbf{W}, s_N, f_\pi(\mathbf{U}^*, s_N)) \end{bmatrix}.$$

Although a Gauss Newton approximation is possible for training the Actor, if the technical complications regarding the input domain and spurious critical points are ignored for now, I am no longer able to ensure a "global" applicability of a Gauss Newton algorithm as I can do for the Critic training in Propositions 2 and 5. The type of an extremum depends completely on the geometry imposed by the approximated Q-function. In practice, this means that one cannot start at an arbitrary point in the parameter space $\boldsymbol{\mathcal{U}}$ and expects the Actor to improve. A second-order optimisation process will get attracted by any type of extremum according to the Q-function. Whatever is closest, in terms of the Newton's direction and the basin of attraction around critical points, is reached, potentially moving the policy parameters actively towards smaller Q-values.

A possible solution might reside in the Levenberg-Marquardt heuristic, where first- and second-order methods are combined. One follows the gradient as long as the Newton's direction points away from an optimum and switches gradually to second-order optimisation to benefit from good convergence properties of a second-order method. Alternatively, for an Actor-Critic algorithm, it might be possible to exploit an easier strategy. First, fast gradient ascent steps are preformed until the error is no longer changing. If required, one can include some small disturbances to escape from saddles and tiny local maxima. Minima and saddles are by their nature always numerically unstable and should not impose a significant problem for gradient directions. After enough gradient ascent steps, one should be close enough to a (local) maximum such that the Hessian in Equation (4.18) is negative definite. Therefore, as the second step, one makes use of a GN algorithm to refine the solution. But to be on the safe side, one should employ some form of test, whether the gradient and Newton's direction point into the same half space.

4.4.2 Impact of Advantage Functions

In the existing literature such as [Schulman et al., 2015, 2016] or in commonly used textbooks, for example [Bertsekas and Tsitsiklis, 1996], a frequently employed instrument are the advantage functions. The so-called advantage of an action under some policy is defined informally as $A(s,a) = Q(s,a) - V(s)$ and corresponds to a state-wise centring of the Q-function at zero. Actions produced by the policy then have an advantage value of zero. Better actions for a given state are those with positive advantage, worse actions possess a negative value. The main motivation for using advantage functions is that the image set of the function $A(s,a)$ is centred around zero and thus more suitable for non-linear function approximation architectures such as MLPs. However, an apparent disadvantage is the requirement to either estimate both the Q- and value function, or to involve the additional computational effort to compute the value function from the Q-function whenever it is needed. Since the task of learning Q and V is still present and the output range of a scalar function should not affect an optimisation problem at all, I wonder whether advantage functions are really beneficial for the optimisation problem

insides the Actor. Based on my empirical insights so far, I would argue that there is no urgent need to rely on advantage functions. MLPs appear to be able to learn value- or Q-functions in the correct range, which implies that a centring around zero is not required. Furthermore, for the application in Actor-Critic algorithms, it seems that there does not arise a benefit from the changed output range of the Critic MLP for the Actor.

This argumentation would also match the structure of differential maps. One can see this structure by defining two policies π_1 and π_2, where π_2 is represented by an MLP and has the parameters $\mathbf{U} \in \boldsymbol{\mathcal{U}}$. One wants to train this second policy π_2 with the goal that it performs better than the current policy π_1, which might be the policy of a previous sweep or some behaviour policy used for exploration. Next, take a look at the definition of the advantage function $A\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ with the two policies already inserted

$$\begin{aligned} A_{\pi_1}(s, \pi_2(\mathbf{U}, s)) &= Q_{\pi_1}(s, \pi_2(\mathbf{U}, s)) - V_{\pi_1}(s) \\ &= Q_{\pi_1}(s, \pi_2(\mathbf{U}, s)) - Q_{\pi_1}(s, \pi_1(s)). \end{aligned}$$

The second line makes use of the relations between value- and Q-functions and emphasizes the importance of not mixing up indices of both policies. To compute the differential map with respect to the policy parameters, I follow my typical approach and obtain

$$\mathrm{D}_{\mathbf{U}}\, A_{\pi_1}(s, \pi_2(\mathbf{U}, s))[\mathbf{H}] = \mathrm{D}_2\, A_{\pi_1}(s, \pi_2(\mathbf{U}, s)) \circ \mathrm{D}_{\mathbf{U}}\, \pi_2(\mathbf{U}, s)[\mathbf{H}].$$

Since the value function is independent of the action, i.e, there is no dependence of π_1 on $\mathbf{U}$, derivatives of the advantage function simplify to

$$\mathrm{D}_2\, A_\pi(s, a)[h] = \mathrm{D}_2\, Q_\pi(s, a)[h] - \underbrace{\mathrm{D}_2\, V_\pi(s)[h]}_{=0}$$

such that I arrive at the same situation as in Equation (4.13). Advantage functions seem to have no direct impact on the learning outcome, at least from the optimisation perspective.

Lastly, the benign property to reduce the variance of gradients by subtracting a baseline from the Q-function (cf. [Sutton et al., 1999]), i.e., switching to advantage functions, is not applicable in my setting, because I make use of deterministic policies. Furthermore, due to my usage of model-based algorithms, I also do not need the mechanisms to express the advantage of one policy over another based on estimates, as it is done for example in [Schulman et al., 2015] close to their first equation. This also frees the effort in algorithmic design choices to ensure that an improved policy is still close to its predecessor. Once a Q-function is available, a GIP can be constructed by just considering the objective from Equation (4.11) and an initial choice of policy parameters.

4.4.3 An Actor-Critic Algorithm with a Gauss Newton Residual Gradient Critic

The realisation of an Actor-Critic algorithm follows the pattern of the PI procedure in Algorithms 2 and 3 from Chapter 3, but partially breaks open the clear separation between Policy Evaluation and Policy Improvement. The training of both MLPs f_Q and f_π becomes to some extend a simultaneous and ongoing task. The exact procedure is outlined in Algorithm 4.

The Actor-Critic algorithm receives two MLPs and their hyper parameters as input and returns improved versions as output. In the ideal case, the MLP f_Q approximates closely

Algorithm 4 Actor-Critic Algorithm with Gauss Newton Optimisation for the Critic

Hyper parameters: $\gamma \in (0,1)$, $\alpha_Q > 0$, $\alpha_\pi > 0$, $c = 10^{-5}$, $\epsilon \leq 10^{-5}$, $N \sim N_{net}$
Input:
- MLP $f_Q \in \mathcal{F}(K_\mathcal{S} + K_\mathcal{A}, \ldots, 1)$ with initialised parameters $\mathbf{W} \in \boldsymbol{\mathcal{W}}$
- MLP $f_\pi \in \mathcal{F}(K_\mathcal{S}, \ldots, K_\mathcal{A})$ with initialised parameters $\mathbf{U} \in \boldsymbol{\mathcal{U}}$

Output:
- $\mathbf{U}$ such that $f_\pi(\mathbf{U}, s)$ is a trained policy
- $\mathbf{W}$ such that $f_Q(\mathbf{W}, s, a) \approx Q_{f_\pi}(s, a)$

for sweep **in** sweeps **do**

// Preparation

Draw (s_i, a_i) for $i = 1, \ldots, N$ uniformly from $\mathcal{S} \times \mathcal{A}$

Construct transition tuples $(s_i, a_i, r_i, s'_i, a'_i = f_\pi(\mathbf{U}, s'_i))$ for all i

// Policy Evaluation : Critic

for i **in** i_Q **do**

Evaluate $F_Q(\mathbf{W}) \coloneqq [f_Q(\mathbf{W}, s_1, a_1) \ldots f_Q(\mathbf{W}, s_N, a_N)]^\mathsf{T} \in \mathbb{R}^N$

Evaluate $F'_Q(\mathbf{W}) \coloneqq [f_Q(\mathbf{W}, s'_1, a'_1) \ldots f_Q(\mathbf{W}, s'_N, a'_N)]^\mathsf{T} \in \mathbb{R}^N$

Compute $G_Q(\mathbf{W})$ and $G'_Q(\mathbf{W})$.

Bellman Residual in Q: $\Delta^Q_\pi(\mathbf{W}) = F_Q(\mathbf{W}) - R_\pi - \gamma F'_Q(\mathbf{W})$

NMSBE: $\mathcal{J}_Q(\mathbf{W}) = \frac{1}{2N} \Delta^Q_\pi(\mathbf{W})^\mathsf{T} \Delta^Q_\pi(\mathbf{W})$

Gradient: $\nabla_\mathbf{W} \mathcal{J}_Q(\mathbf{W}) = \frac{1}{N} \Big(G_Q(\mathbf{W}) - \gamma G'_Q(\mathbf{W}) \Big)^\mathsf{T} \Delta^Q_\pi(\mathbf{W})$

Hessian: $\mathsf{H}_\mathbf{W}\, \mathcal{J}_Q(\mathbf{W}) = \frac{1}{N} \Big(G_Q(\mathbf{W}) - \gamma G'_Q(\mathbf{W}) \Big)^\mathsf{T} \Big(G_Q(\mathbf{W}) - \gamma G'_Q(\mathbf{W}) \Big)$

Solve for η: $(\mathsf{H}_\mathbf{W}\, \mathcal{J}_Q(\mathbf{W}) + c I_{N_{net}})\, \eta = \nabla_\mathbf{W} \mathcal{J}_Q(\mathbf{W})$
(e.g. with Householder QR-Decomposition)

Descent step: $\mathbf{W} \leftarrow \mathbf{W} - \alpha_Q \eta$

end for

// Policy Improvement : Actor

for i **in** i_π **do**

Evaluate $F_{all}(\mathbf{W}, \mathbf{U}) \coloneqq [f_Q(\mathbf{W}, s_1, f_\pi(\mathbf{U}, s_1)) \ldots f_Q(\mathbf{W}, s_N, f_\pi(\mathbf{U}, s_N))]^\mathsf{T} \in \mathbb{R}^N$

Compute $C(\mathbf{W}, \mathbf{U})$ and $G_\pi(\mathbf{U})$

Objective: $\mathcal{J}_\pi(\mathbf{U}) = \frac{1}{N} \mathbb{1}^\mathsf{T} F_{all}(\mathbf{W}, \mathbf{U})$

Gradient: $\nabla_\mathbf{U} \mathcal{J}_\pi(\mathbf{U}) = \frac{1}{N} G_\pi(\mathbf{U})^\mathsf{T} C(\mathbf{W}, \mathbf{U})^\mathsf{T}$

Ascent step: $\mathbf{U} \leftarrow \mathbf{U} + \alpha_\pi \nabla_\mathbf{U} \mathcal{J}_\pi(\mathbf{U})$

end for

// Training Progress

Evaluate f_π empirically using several rollouts

end for

the Q-function of the policy f_π and f_π is itself close to an optimal policy. In a more typical case, the policy should perform better than the one implied by the initial parameters and the Critic captures only the most important aspects of the resulting policy.

Based on the empirical insights from Chapter 3, I remove the toggle for transient or persistent parameters and data sets. The Actor-Critic algorithm always employs persistent MLPs, meaning that the training progress is kept across sweeps. Opposed to that, the data collection is now always performed transient. In each sweep, new transition data is collected, such that the whole Actor-Critic algorithm resembles a mini batch stochastic gradient ascent or descent optimisation. Despite introducing some noise in training, I argue that this strategy is not a problem. In every sweep, there are still always N state-action samples. Thus, every sweep matches individually the setting from the theoretical investigation. As long as the states are sampled uniformly without skewing or bias, they approximate in each sweep the NMSBE in the same manner. Thus, critical points for the Q-function, which I am aiming for, are the same and the optimisation of f_Q works as intended in the long term. Since the training of f_π only depends on f_Q and arbitrary samples from the complete state space, it will work as well. The Critic MLP f_Q is static from the Actor's perspective and its critical points with respect to action inputs remain unchanged from the sampling process.

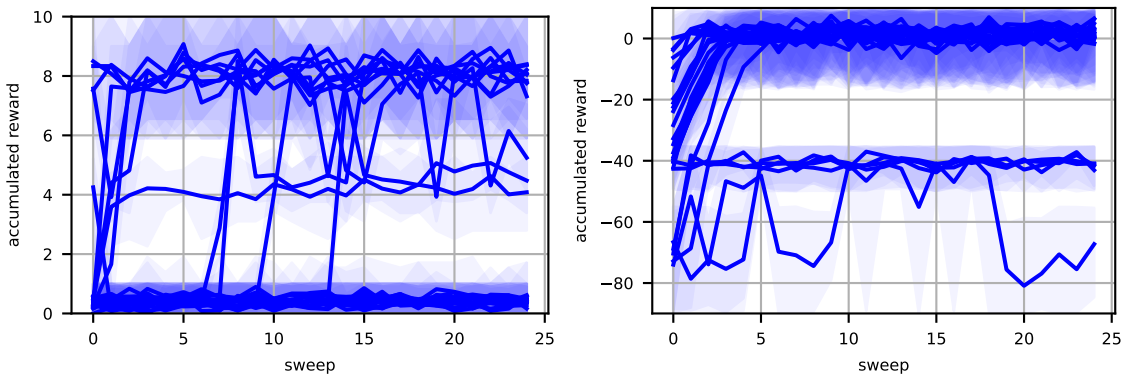
4.5 Experiments Regarding Actor-Critic Algorithms

In the following, I provide numerical confirmations to some of my theoretical insights from Sections 4.3 and 4.4. I describe the overall experimental setting first. Afterwards, I address requirements for the reward function and the impact of over-parametrisation. Lastly, I evaluate the Actor-Critic algorithm itself and outline its limitations.

For the experiments in this section, I follow the common practice of the RL community and still ignore the issues regarding the input domain. They result from differences between the desired action input to the Critic MLP and the actual compact subset of an Euclidean vector space, which has to be used as $\mathcal{A}$. The origin of this issue has been described to some extend in Sections 4.3 and 4.4 and is the main topic for the subsequent Section 4.6.

4.5.1 Experimental Setup

There are two MLPs for f_Q and f_π with the size of input and output layers set according to the environment. They use Bent-Identity in hidden layers. The Critic has a linear output and the Actor contains $tanh(\cdot)$ to match $\mathcal{A}$. The width and depth of the MLPs varies for each experiment. Parameters for f_Q and f_π are initialised element-wise and uniformly in the interval $[-1, 1]$. Furthermore, they are persistent, i.e., each sweep of PI starts with the parameters from the last round. I run in each PI sweep $i_Q = 2 \cdot 10^3$ Approximated Newton steps for the Critic and $i_\pi = 5 \cdot 10^3$ gradient ascent steps for the Actor. For both algorithms, I pick a constant learning rate $\alpha_Q = \alpha_\pi = 10^{-2}$. Furthermore, I set the regularisation for the approximated Hessian of the NMSBE to $c_Q = 10^{-5}$. With these parameters, the Critic is able to converge and the Actor is training fast enough. Since I use a more sophisticated descent algorithm for the Critic compared to the Actor, it is important to ensure that the Actor has enough time to realise a proper improvement of the policy, hence it can use more descent steps than the Critic. Training samples consist of uniformly distributed tuples in



(a) Piecewise constant sparse reward function (b) Reward function without constant regions

Figure 4.3: Actor-Critic performance behaviour for r_1 and r_2 in the one dimensional toy benchmark. Only with the reward r_2, the major part of all repetitions settle at high values over time. The constant areas in r_1 cause the Actor improvement to fail.

the state-action space. I collect them once at the beginning of a sweep and use them in a batch ascent/descent setting.

By running rollouts as described at the end of Section 3.4.2, I evaluate the performance of policies. Similar to the Critic-Only Policy Iteration experiment from Section 3.5.5, I create in each sweep 10 trajectories starting randomly in $\mathcal{S}$. Each trajectory consists of 500 transitions. Furthermore, the entire experiment is repeated several times from scratch to remove the impact from initialisation. But opposed to previous Policy Iteration experiments, in this section I do not combine all the repetitions of an experiment in a single graph. The results, which will be shown in the next sections, are too different to each other to be merged. Hence, I depict average performance curve and their respective min-max enveloping curves semi-transparently for each repetition and rely on blending to reveal how often a certain outcome is achieved across all repetitions.

4.5.2 Reward Issues

To emphasize the effect of reward functions on the learning progress, consider the training behaviour of an Actor-Critic algorithm in the one dimensional benchmark problem (cf. Section 2.6.2) for the two reward functions r_1 and r_2 as defined in Equations (2.35) and (2.36), respectively. I use $N = 250$ training samples (s, a) and train 25 times a policy from scratch. The MLPs f_Q and f_π have width 15 and two hidden layers. Figure 4.3a shows results for r_1 and Figure 4.3b for r_2.

I observe the expected behaviour. When using the reward function r_1, a major part of all repetitions settle at close to zero accumulated and discounted reward. This can be seen in Figure 4.3a. Except for a few cases which recover after several sweeps, the Actor-Critic algorithm produces consistently policies without any performance. Those policies do not manage to reach and stay in the reward region, thereby generating the value zero. Due to the loss of information when computing the gradient for updating Actor parameters, it is not possible to change the policy. Only if one has luck during initialisation, it is possible yet unlikely to observe good performing policies.

Once I replace the reward function with r_2, the performance of the Actor-Critic algorithm changes drastically. Figure 4.3b shows that the major part of repetitions approaches a high reward level and only a couple stabilise around an intermediate performance level located at -40. Policies with around zero accumulated reward are those, which move an agent successfully towards the state with highest one-step reward ($\max_s r_2(s) = 1.0$) and can also stabilise it at this state. If a policy only reaches the intermediate performance level, then this corresponds to an agent, which oscillates with a stable orbit around the state with maximal one-step reward.

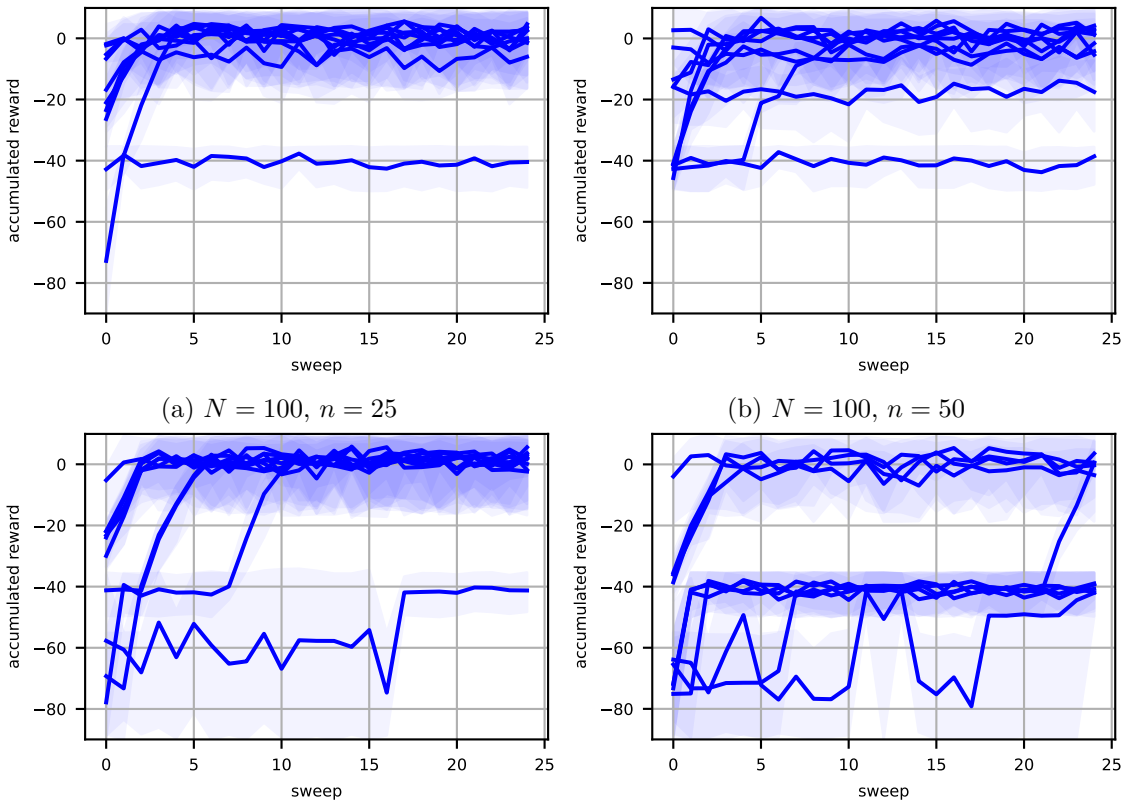
From this experiment, one can see that even in trivial dynamical systems an inappropriate reward can lead to failure. To allow Actor-Critic algorithms to work, it is mandatory to use reward functions without extended constant areas whenever possible. Unfortunately, this will open up issues regarding reward shaping. Namely, the more information an engineer encodes in the reward function, the more likely it is for the algorithm to come up with solutions, which do not match the expectations or intentions of the engineer.

4.5.3 Over-parametrised Actor

To characterise the influence of over-parametrisation, I use in this experiment again the toy problem and but only the reward r_2. I run the Actor-Critic algorithm with a varying amount of training data and different MLPs such that the impact on the critical point conditions and the over-parametrisation requirement becomes accessible. I set $N \in \{100, 200\}$ and use two MLP widths $n \in \{25, 50\}$. In this experiment, MLPs only possess one hidden layer to ease the computational burden. Furthermore, there are only 10 repetitions. The MLPs have $N_{net}^{crit} \in \{101, 201\}$ parameters for the Critic and $N_{net}^{act} \in \{76, 151\}$ for the Actor. The factor two for network widths n and number of samples N is just a coincidence. I select the width for a given number of samples with the combined absolute distance $d = \left|N_{net}^{crit} - N\right| + \left|N_{net}^{act} - N\right|$. When the number of samples N is set to the aforementioned values, the distance d is smallest for the widths mentioned above. Actually, I have tested the values $n \in \{10, 15, 20, \ldots, 100\}$, but only report the subset mentioned above to maintain the reading flow. My results are contained in Figure 4.4.

I notice that increasing the amount of samples for both MLP widths has a negative impact on performance. Whereas there still exist some repetitions, which settle at the highest possible reward, for both values of n the amount of bad performing repetitions grows for larger N. For the small MLP in Figures 4.4a and 4.4c, the time for convergence is increased. Some runs take more PI sweeps to reach at the largest accumulated reward. A single bad run keeps a lower performance level over a longer time span (in terms of sweeps when compared against its counter part in Figure 4.4a) before it reaches the suboptimal level of -40. This level is visible in all four combinations in Figure 4.4 and belongs to a policy, where the state is not stabilised close to the reward centre but oscillates around it. For those policies, MC rollouts collect frequently lower one-step rewards. For the bigger MLP architecture, the effects are more pronounced. The asymptotic performance for half of the runs is significantly lower and only arrives at this -40 level when using more data.

I conclude that the size of MLPs should be related to the amount of samples used for training. Of course, using more data than parameters can work sometimes. I have seen this with other hyper parameter combinations of n and N, which also matches insights from other experiments in my thesis. But my results in Figure 4.4 show that there are configurations, where more data can be harmful to the learning outcome of Actor-Critic



(a) $N = 100$, $n = 25$

(b) $N = 100$, $n = 50$

(c) $N = 200$, $n = 25$

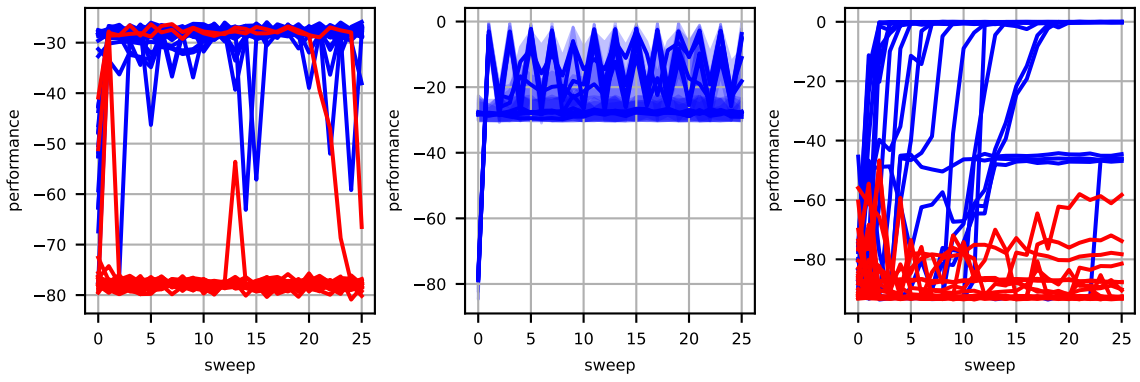
(d) $N = 200$, $n = 50$

Figure 4.4: Actor-Critic performance behaviour for different amount of training data and MLP sizes. Adding more data is not necessarily beneficial. In my experiments, I observe that maintaining $N \approx N_{net}$ is also important.

algorithms and may result in policies with lower overall performance. Even in a toy problem, one has to pay attention to algorithmic details and cannot simply expect it to work. By using my theoretical insights, I provide at least a partial guidance for constructing an Actor-Critic algorithm, which uses non-linear function approximation architectures, and for formulating a well-behaving optimisation problem.

4.5.4 Limitations of the Actor-Critic Approach

As the last experiments in this section, I want to demonstrate also some limitations of Actor-Critic algorithms. They become visible once I switch from the one dimensional toy problem to the environments *MyMountainCar-v1* and *MyCartPole-v1*. Since there is a parametrised Actor, I use only version *v1* of both environments. Only this version brings the correct reward function for the usage with Actor-Critic approaches. Since the two environments posses a higher dimensional state space with a more complex Q-function, I use again two hidden layers and width 15. Next, I also employ a different setting for the optimisation tasks, namely those from Section 3.5.5. This is done such that I can compare the Actor-Critic performance with the PI experiments build around a Critic-Only approach. The Critic MLP f_Q receives now only $i_Q = 300$ iterations of the Gauss Newton descent



(a) (b) (c)

Figure 4.5: Performance of the Actor-Critic approach for different environments. Bad performing runs are drawn in red. **a)**: *MyMountainCar-v1* cannot be solved, because the Actor learns the function $\pi(s) = 1$ in the best case, or shows no improvement in the worst case. **b)**: The same environment with Critic-Only algorithm, i.e., the PI approach from 3.5.5, as reference. It is possible to solve the environment. Hence, the struggle is indeed due to using parametrised Actors. **c)**: *MyCartPole-v1* can be solved, and if so, with high quality policies. But there are also too many bad performing repetitions to label it a successful algorithm.

algorithm with a larger learning rate $\alpha_Q = 0.1$. This change is also motivated by my insight, that in those environments a Critic converges sufficiently well within that amount of iterations if the learning rate is increased. Furthermore, this also reduces computational effort. The Actor f_π now performs $i_\pi = 5 \cdot 10^3$ iterations for maximising f_Q. The larger amount of iterations does not increase runtime significantly, since it is a first-order only algorithm, but reduces the risks of relying on an "incomplete" Policy Improvement step. Its learning rate stays the same as before and uses the value $\alpha_\pi = 0.01$. Lastly, I increase the amount of training samples to $N = 3000$. Of course, by doing so, the over-parametrisation requirement is violated. But the empirical evidence from the Policy Iteration experiment in Section 3.5.5 suggest that the additional training data is indeed required. Also, the thoughts and explanations from Section 3.6 apply, for the Critic directly and for Actor indirectly through the Supervised Regression analogy. Lastly, due to a strong variation in the outcome of training, I increased again the number of repetitions to 50. Figure 4.5 shows the training behaviour of the Actor-Critic algorithm for both environments and also the outcome of a Critic-Only approach (i.e., that of Section 3.5.5) for *MyMountainCar-v1*.

Figure 4.5a shows that the Actor-Critic approach is not able to solve the environment *MyMountainCar-v1*. In the best case, the Actor learns a policy similar to $\pi(s) = 1$ for all states, which is capable of producing some reward but is not the best possible outcome. In the worst case, the Actor does not improve at all. The resulting policy keeps the car down in the valley. So far, I have not observed a single close to optimal policy produced by an Actor-Critic algorithm, which could be a hint, that the environment itself is containing an issue.

Since I have not used a PI algorithm together with *MyMountainCar-v1* in Chapter 3, it is important to check, whether a solution to this control problem is learnable. To

remove the Actor as possible source of errors, I employ a Critic-Only approach as part of this experiment. Figure 4.5b demonstrates, that the environment is indeed solvable and, consequently, that the Actor-Critic algorithm causes more trouble than a Critic-Only approach. When using a Critic alone and its implicitly defined policy, one can produce at least for single sweeps a policy, which approaches zero accumulated reward. Such a policy drives the car via a swing-up in the goal area and also breaks early enough to stay there. As a human, one would classify this policy as optimal. Furthermore, among all repetitions, there is no longer a bad performing one, which would not create any reward at all. One can also see that the Critic-Only method also suffers from oscillations between two types of policies, namely a perfect one and a sub-optimal policy. This could be explained by the typical behaviour of an Approximated Policy Iteration algorithm.

Lastly, the results shown in Figure 4.5c confirm that an Actor-Critic can work. At least in the sense that the policies for a reasonable amount of repetitions achieve high accumulated rewards. The results also demonstrate, why a combined visualisation similar to that for Critic-Only PI experimentsis in the previous chapter is not possible over here. For an Actor-Critic algorithm, this would lead to completely skewed statements. Several repetitions demonstrate that a close to optimal policy (i.e., a perfect balancing of the pole with minimal movement of cart) can be learned. Some runs of the experiment settle at intermediate performance, which corresponds to a slightly swinging of pole, but without falling over. Some struggle completely and the corresponding policies are unable to keep the pole upright.

The central difference between the environments, which I consider for this section, is whether or not they make a strongly non-linear policy class necessary to achieve an optimal control. The environment *MyCartPole-v1* and the one dimensional toy problem can be solved with an almost linear policy of the form $\pi(s) = \text{clip}(A \cdot s)$, where A denotes some matrix of matching shape for the state vector s and $\text{clip}(\cdot)$ projects the vector component-wise into $\mathcal{A}$. Due to the clipping operation, large angles of the pole always produce a large value for the action. Once the balancing point is close by, the clipping is no longer required and the policy becomes linear. The same would be true for *MyMountainCar-v0* with the original task of just getting fast. Opposed to that, the current reward signal in *MyMountainCar-v1* demands a policy to go to the rightmost position and stop there. In this environment, there does not exist an immediate balancing task and it also does not require actions with frequently switching signs as it is needed in the toy benchmark. Hence, a policy cannot be expressed as a linear function of the state and also not as an affine mapping. Thus, a possible explanation for the bad outcome for learned policies is that in *MyMountainCar-v1* it is too easy for an Actor to produce a policy similar to $\pi(s) = 1$ for all states. This "solution" is producing higher rewards than a the random initialisation and, therefore, represents a learning progress. Furthermore, a constant function for the Actor would correspond to a rank deficient $G_\pi(W)$. Therefore, this kind of policy is a perfectly valid outcome of the optimisation task, since many, if not all, parameters in the parameter space of f_π form a critical point.

I hypothesise that training the MLP f_π with first-order only gradient ascent in Actor might be too weak. The ascent directions are not good enough to lead early enough to a correct policy. In particular, for *MyMountainCar-v1*, this implies that one cannot avoid the suboptimal solution, which is rather easy to represent with an arbitrary MLP. Unfortunately, employing second-order methods with (approximated) Hessian information is not possible for Actor-Critic methods as hinted at the end of Section 4.4.1. Thus, these

experiments are an empirical motivation for a different approach towards training an Actor. One needs to be able to handle geometrical issues explicitly, which is the topic for the next section.

4.6 Fitted-Actors to Handle Spurious Critical Points

This section covers the spurious critical points of the Critic with respect to its action input, which arise from enforcing geometrical requirements for the action space. First, I elaborate on their origin and the related problems for a naive formulation of an optimisation task. Second, I describe how so-called Fitted-Actors could be used to avoid issues related to spurious critical points. Since they realise the training of Actors based on a separated Supervised Regression task, one has the full control over the construction of the training dataset and how optimal actions are extracted from a Q-function. Lastly, a new and unexpected complication for Fitted-Actors needs to be solved. The training of an Actor in a supervised manner struggles to learn fine details according to a Q-function as long as the output activation of the MLP is $\tanh(\cdot)$. An easy solution is to make use of linear outputs, which do not include a squashing behaviour. But this causes the Actor to produce values outside of the action space. As an attempt to overcome this problem, I propose and investigate a Fitted-Actor algorithm, which works with the unit balls as target domain for the action space.

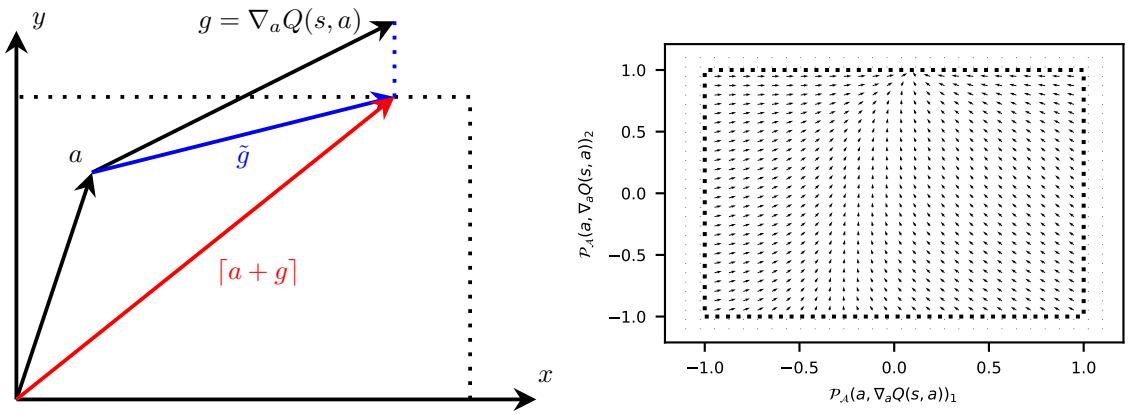
4.6.1 Critic's Expected Action Input vs. the Actual Action Space

The MLPs used for approximating a Q-function expect a full Euclidean vector space as domain. On the contrary, for DP applications, one is forced to work with compact subsets for the state space $\mathcal{S}$ and action space $\mathcal{A}$ to ensure the boundedness of rewards and their accumulation. This opens up the possibility that critical points regarding the action input for f_Q are located outside of $\mathcal{A}$. Thus, the behaviour of algorithms to solve $\operatorname{argmax}_a Q(s,a)$, when they operate at boundaries of $\mathcal{A}$, plays an important role and one should handle these geometrical issues explicitly. The simplest scenario possible is an action space, which is given by a hypercube of the form $\mathcal{A} \coloneqq [-1,1]^{K_\mathcal{A}} \subset \mathbb{R}^{K_\mathcal{A}}$. At the same time, it is the typical action space encountered in many RL benchmarks. Thus, it serves as a representative foundation for the remainder of this section.

Having optimisation on subspaces in mind, any manipulation applied to a point in the action space, which ensures that $\mathcal{A}$ is not left, should be expressed through a projection operation. An operator, which manipulates gradients in the action space such that one stays in $\mathcal{A}$, takes the form

$$P_\mathcal{A}(a,g) = \lceil a+g \rceil - a, \tag{4.19}$$

where $\lceil \cdot \rceil$ clips a vector to $\mathcal{A}$ component-wise. In particular, for a Direct-GIP method of the form given in Equation (4.1), this operator corresponds to clipping the action iterates after every ascent step to the action space. It should match the approach used in [Nichols and Dracopoulos, 2014] or [Nichols, 2016]. The expression $P_\mathcal{A}$ is indeed a projection, since a straightforward computation reveals that the idempotency property $(P_\mathcal{A} \circ P_\mathcal{A})(a,g) = P_\mathcal{A}(a,g)$ holds. Of course, this operator is neither an orthogonal projection nor a linear one. One can verify this visually with the help of Figure 4.6.



(a) Visualisation of $P_{\mathcal{A}}(a, g)$.

(b) Projection applied to gradient vector field.

Figure 4.6: The projection operator from Equation (4.19) applied to the gradient vector field. One can see a new spurious critical point at the top boundary. Gradients for actions outside of $\mathcal{A}$ are omitted for better visibility and because $P_{\mathcal{A}}(a, g)$ is not designed for that area. Furthermore, gradient lengths are upper bounded to remove visual clutter. Still, whenever an original gradient vanishes also the corresponding arrow in the visualisation vanishes.

Figure 4.6a depicts, how the projector in Equation (4.19) is acting on direction vectors such as the gradient, which would leave the space when applied to a certain action. In Figure 4.6b, the outcome of projection, when applied to the previous gradient vector field, is shown. It is easy to see that by using this projection spurious critical points now show up at the boundary whenever gradients are perpendicular to the border and point outwards. Other components do not exist for such gradients and the remaining ones are suppressed by the application of $P_{\mathcal{A}}$. Hence, one is left with a new type of critical point at the equilibriums $\langle \nabla_a Q(s,a), n_a \rangle = \|\nabla_a Q(s,a)\|$, where n_a refers to the normal vector of the boundary at the point of interest a. A good effect is that due to $P_{\mathcal{A}}$, there is always at least one critical point, where an ascent algorithm would stop. Either as a natural critical point inside of $\mathcal{A}$, or as a spurious one on the boundary. The mismatch between the expected and actual action domain for the Critic now gives rise to two complications.

The minor complication applies to derivative based Direct-GIP methods, where one just needs to keep the current action inside of $\mathcal{A}$. For implementing Direct-GIP via gradient ascent in action space (cf. Equation (4.1)), any method needs to be aware of the boundary. This can be achieved easily for an action space, which takes the form of a hypercube, by transforming the gradient for the current action with the help of Equation (4.19). As the result, the next action iterate is manipulated and the accumulation point is not necessarily a true natural critical point any more. Hence, my analysis of critical points from Section 4.3 does not apply as is to such Direct-GIP algorithms and less statements about the optimisation behaviour are possible.

The major complication is related to the Actor-Critic formulation, where policy parameters get changed via the Critic through the chain rule (cf. Equation (4.4)). For this approach, there is no longer a clear way to include the projector $P_{\mathcal{A}}$ into training. There are two effects, where the difference in domains is noticeable for Actor-Critics. First, the policy MLP has to use $tanh(\cdot)$ as activation function in the output layer to comply to

the action space such that values outside of $\mathcal{A}$ cannot be produced by design. Then, one can only hope that solutions to Equation (4.14) correspond to values for the Actor inside $\mathcal{A}$. Second, the Gauss Newton approximation as described in Sections 4.4 and 4.5 is not realisable in general, because critical points of the Q-function approximator with respect to the action input might be located outside of the action space. Thus, the GN approximation for the Hessian is not only harmful since I have no control over the curvature, it can also be a completely bad approximation if omitted second derivatives contribute too much when being away from potentially unreachable natural critical points.

In conclusion, when employing a Critic as objective, one is forced to handle these geometrical issues explicitly. The main goal is to stay inside of the action space $\mathcal{A}$. Actor-Critic algorithms would require careful work to include the geometry of the action space in the training process of both the Actor and Critic. However, there is no clear strategy how to include projections on the hypercube during the optimisation of both. The projector $P_{\mathcal{A}}$, as it is defined in Equation (4.19), is a non-linear oblique projection operator, which prevents additionally the usage of classic results for projected gradients if a function's input domain (in this case that of the Q-function or its approximation f_Q) gets restricted to some smaller Hilbert space. On top of that, one does not even have a proper Hilbert space for $\mathcal{A}$ any more. Actually, the space $\mathcal{A}$ does not even have the structure of a vector space. Despite $\mathcal{A} \subset \mathbb{R}^{K_{\mathcal{A}}}$ still applies in my example, the boundedness of $\mathcal{A}$ and the typical definition as the hypercube $\mathcal{A} = [-1, 1]^{K_{\mathcal{A}}}$ prevents the definition of an inner product and also interferes with the vector space conditions. Thus, the adaption of Actor-Critic algorithms remains open for now.

Fortunately, Direct-GIP methods are more flexible and allow for a reliable recovery of optimal actions according to some Q_π. Thus, I propose to focus on Fitted-Actor algorithms in the following, where Actors are trained as a separated Supervised Regression task. By doing so, one can avoid the challenges and limitations of Actor-Critic training.

4.6.2 Actor Training as Supervised Regression Task

If I combine the concerns from Section 4.6.1 and also keep in mind recent effective algorithms such as *QT-Opt*, which also scale to higher dimensional action spaces according to the authors, it is obvious that training the Actor as separated Supervised Regression task appears as promising alternative. Hence, I follow now a new strategy. First, I solve Equation (2.15) with any suitable (global) optimisation approach and generate a dataset consisting of (s, a^*) pairs. Several choices for the optimiser exists as it can be seen in Section 4.2.2. I use the *Cross Entropy Method* as described in [Kalashnikov et al., 2018]. The method is easy to realise, can handle sufficiently high dimensional action spaces and has a pleasing performance in practice. Second, I address and solve the Supervised Regression task

$$\widetilde{\mathcal{J}}_\pi(\mathbf{U}) = \frac{1}{N} \sum_{i=1}^{N} \left\| f_\pi(\mathbf{U}, s_i) - a_i^* \right\|^2 \tag{4.20}$$

with typical derivative based optimisation. This decoupling of the Critic and Actor brings several advantages.

First, the shape and curvature of the Q-function approximation architecture is no longer a limiting factor. For creating (s, a^*) tuples, derivative free random search methods can be employed such that the type of extrema does not matter. Furthermore, flat regions without

gradient information, which would pose a challenge for an hypothetical Actor or gradients with respect to the action input, are harmless. On top of that, my previously introduced and more restrictive requirements regarding the reward signal can be removed, because there is no longer the need to maintain a proper slope of the approximated Q-function.

Second, input space limitations regarding the position and nature of critical points are not problematic. During the search for optimal actions in a certain state, constraints on the state and action spaces can be easily ensured by the use of clipping or projection operations applied to the samples themselves. The search space consists by design only of the considered action space.

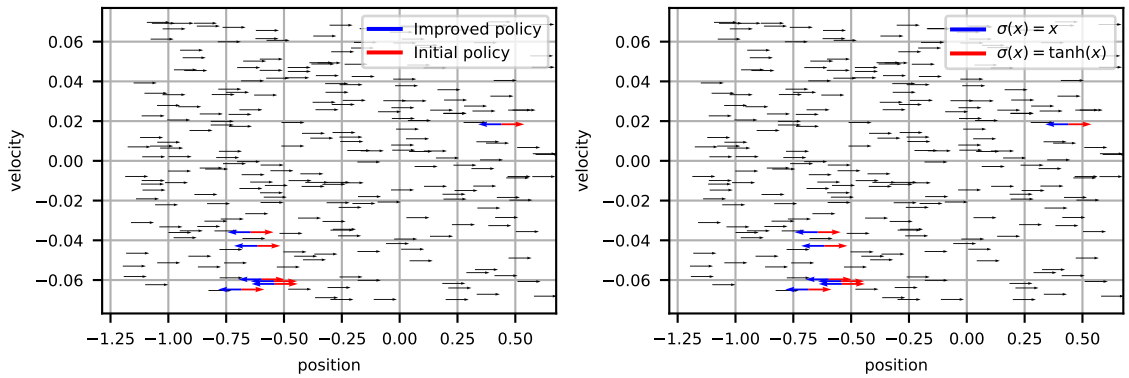
Third, the computational effort inside of the full Hessian of the Q-function with respect to the input drops away. Additionally, it is rather convenient to map into the correct range for the action space $\mathcal{A}$ by using $\tanh(\cdot)$ as activation function for the output layer.

To minimise the objective for the Actor in Equation (4.20), I can now employ a Gauss Newton algorithm and benefit from all theoretical insights in [Shen, 2018b]. Namely, fitting the Actor is expected to work reliably once MLP with proper activation functions are used. If over-parametrisation is available, I can even eliminate suboptimal local minima.

The entire Fitted-Actor algorithm is contained in Algorithm 5 and will be described in a later section once all relevant components have been described. For the rest of this section, I need to introduce an unexpected complication first. As it turns out, Supervised Regression with $\tanh(\cdot)$ activation functions for the output layer in the MLP f_π gives rise to a new and unsolved challenge for training. It becomes visible in the Mountain Car control problem *MyMountainCar-v1* and is also in line with the limitations of an Actor-Critic method in that environment. First, I set up by hand the first sweep of a Policy Iteration algorithm. More precisely, I take the initial policy $\pi(s) = 1$ and create its evaluation Q_π with any suitable method. As the next step, the PI algorithm needs to come up with a second policy, which needs to perform better than the initial constant policy. In the Mountain Car problem and for the selected reward, this means that the new policy must produce for some parts of the state space actions with negative values to gain higher velocities than those appearing under a constant policy. Next, I can test the realisation of Policy Improvement with a Fitted-Actor. I create a training set of (s_i, a_i^*) pairs with $i = 1, \ldots, N$ through the *Cross Entropy Method* (cf. Algorithm 6). With that method, I search for all existing sampled states s those actions, which satisfy sufficiently well $\operatorname{argmax}_a Q(s, a)$. Once this is completed, I fit the policy MLP f_π to this dataset by running a Gauss Newton Non-Linear Least Squares algorithm until convergence. Results and failure cases are visible in Figure 4.7.

Please note that both quiver plots in Figure 4.7 appear to be identical, but this is not the case in terms of numerical values in the computer. Because the length of action arrows is upper bounded to enable the presented visualisation, the differences between the two figures get lost. When looking directly at the numerical values for the coloured actions, I find different values for the policy output.

Clearly visible in Figure 4.7a is the successful application of CEM. It is possible to obtain from a given Q-function a dataset consisting of state action tuples, which convey the semantically correct information to enable Policy Improvement. Whereas the most actions after the improvement step still point in the same direction as the original values (black arrows), there is indeed an acceleration happening down in the valley in the interval $s = [-0.5, -0.06]^\mathsf{T}$. The related actions are highlighted as red (initial policy) and blue (improved actions) arrows.



(a) Initial policy and improved one via CEM. (b) Result of regression for both activations.

Figure 4.7: Fitting a policy MLP f_π to state action samples is not working correctly in *MyMountainCar-v1*. Arrows correspond to accelerating in the indicated direction. Due to a necessary clipping of arrow lengths, both figures appear to be identical. **a)**: The state action samples used for training and optimal actions samples obtained from the Critic f_Q with the *Cross Entropy Method*. Those actions, which change the sign after the Policy Improvement step, are highlighted in red and blue, respectively. All others are drawn in black. **b)**: An MLP f_π with $\tanh(\cdot)$ in its output layer is not able to adapt to the few actions samples, where the action now has to be negative. Switching to a linear activation for the output layer allows for a correct regression. Affected actions are drawn in different colours.

The struggle to solve the regression task for a Fitted-Actor becomes obvious in Figure 4.7b. The MLP f_π prefers to produce a policy, which is mapping the entire state space to a constant value. For the training data at hand, this constant action is positive one, which corresponds to the most frequent action value in the dataset as produced by the initial policy $\pi(s) = 1$. By using $\tanh(\cdot)$ to comply with the action space, the MLP f_π struggles to learn fine details regarding changed signs for the few actions (i.e., the arrows drawn in red and blue as before). Therefore, a successful Policy Improvement is not possible and the overall Fitted-Actor Policy Iteration algorithm is prone to become blocked.

As it can already be seen in Figure 4.7b, the experiment also contains a setting for the regression task, which involves a plain linear output for the MLP f_π. The difference in sign between red and blue arrows in that figure belongs precisely to the type of activation function used for f_π and suggests that this bad behaviour is mainly due to the usage of $\tanh(\cdot)$ in the output layer. Once a linear output is used for the policy approximation architecture, the regression outcome improves and now carries the correct qualitative information to be used inside a PI algorithm. Of course, a linear output is not suitable for a DP application, since the MLP output is now residing outside of the action space. Normally, one is forced to choose $\tanh(\cdot)$ as activation function in the output layer to comply to the hypercube as action space. But my single execution of a Supervised Regression task already suggests that this practice needs to be reworked.

In the following, I want to provide more resilient statements regarding the outcome of regression. Thus, I repeat the regression task from before 50 times and collect the values of several performance indicators to arrive at empirical results. I follow the exact

same procedure as described above and only provide the remaining details to complete the experimental setting. I work with $N = 250$ sampled states, which are distributed uniformly in $\mathcal{S}$. The Q-function is approximated with the MLP $f_Q \in \mathcal{F}(3, 15, 15, 1)$ with a linear output. The MLP $f_\pi \in \mathcal{F}(2, 15, 15, 1)$ represents a policy and uses once $\tanh(\cdot)$ for the output and once the identity function. Both have a Bent-Identity in hidden layers, use the learning rates $\alpha_Q = \alpha_\pi = 0.1$ and $i_Q = i_\pi = 1000$ iterations for descending. As the performance indicators, I use four different measurements. As the first, I collect the final fitting errors (mean squared ℓ_2 error) for the Fitted-Actor. The mean reward of the improved policies obtained from rollouts as described at the end of Section 3.4.2 provides the second indicator. The third and fourth indicators are defined by the span of action values and the angle between all actions to a reference point. Both rely on a one dimensional action space and need to be adapted before they can be used in arbitrary spaces. I compute the span of action values by evaluating for the improved policy π the largest possible difference between a pair of actions $\delta a = \max_{s\in\mathcal{S}} \pi'(s) - \min_{s\in\mathcal{S}} \pi'(s)$. For a successful improvement with $\tanh(\cdot)$ activation, the span δa approaches two, i.e., some actions are changed from $+1$ to -1. With a linear activation, the span should be larger than two, because actions outside of the space can be produced. The angle span provides a more strict and binary statement about the behaviour of actions. I convert the output of a policy to an angle towards a reference action first. Namely, I set $\beta(s) = \arccos(\pi(s), a_{ref})$ with the reference action $a_{ref} = 1$. The angle can take two different values $\beta \in \{0, 180°\}$, depending on whether the action points to the right or left[1]. Lastly, I compute in a similar fashion the angle span by $\delta\beta = \max_{s\in\mathcal{S}} \beta(s) - \min_{s\in\mathcal{S}} \beta(s)$. If this value is zero, then all seemingly improved actions still point in the same direction as the reference action. Since the reference is matching the output of the initial policy, this means the improvement has not been successful. Once the angle span jumps to 180°, at least some actions have changed their orientation. The evaluation of the four indicators for all repetitions is shown as box plots in Figure 4.8.

The results demonstrate empirically the struggle in training an MLP, which employs $\tanh(\cdot)$ in its output layer, when compared to an MLP with a linear output. Performing a successful regression of the training data is in many cases not possible with an MLP that relies on $\tanh(\cdot)$ in its output layer. The leftmost indicator in Figure 4.8 shows that small final errors can occur, but are not the typical case. In many cases, not all improved actions are represented correctly by the policy MLP, which leads to problems in Policy Improvement as shown in Figure 4.7b. Switching to linear outputs allows to learn more fine grained details. This results as expected in smaller final errors and overall lower errors compared to the case with $\tanh(\cdot)$.

The mean reward of the improved policy, which corresponds to the second indicator in Figure 4.8, reveals that a linear output with a mandatory clipping operation before actions are fed into the dynamical system is not an option. Whereas the policy produced by f_π with $\tanh(\cdot)$ at least can achieve consistently an expected reward of -30, which corresponds to driving right as far as possible, the performance of f_π with linear output is even worse. However, it is debatable, whether the collected reward is a reliable tool for assessment at the very beginning of a PI algorithm.

The indicators responsible for the action and angle span in Figure 4.8 unveil clearly the struggle for training when using $\tanh(\cdot)$. Both indicators report zero as the major value

[1] I am not using radians such that the confusion $\pi \approx 3.1415$ does not arise.

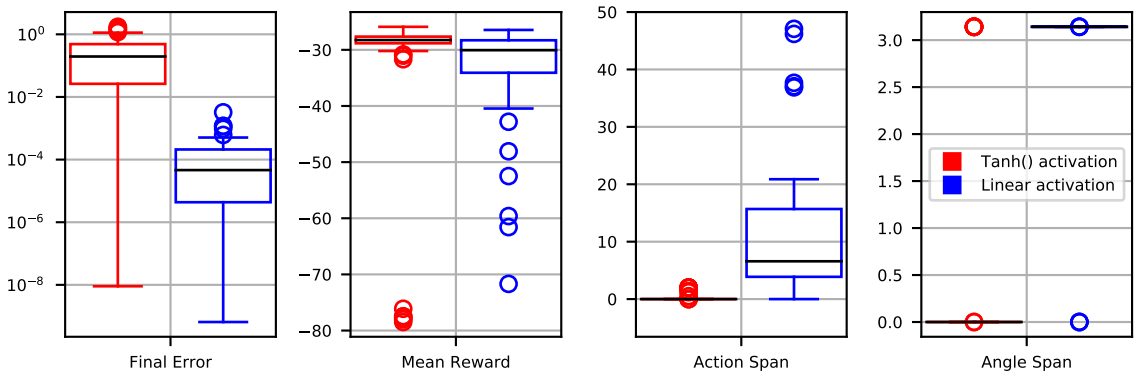


Figure 4.8: Empirical investigation for fitting MLPs to (s, a^*) tuples. The experiment uses the same setting as for Figure 4.7, but involves more repetitions.
The four indicators demonstrate the challenge in fitting a policy to the data, if $\tanh(\cdot)$ is employed in the output (red box plots). The MLP is not able to decrease sufficiently the training error and does not produce correct actions as indicated by the action and angle spans. Thus, the new policy after the improvement step is not able to capture the essence of the reward function and blocks the PI algorithm. With a linear output in the policy MLP, learning becomes possible. But the main limitation is that the action space is left (third indicator) and, consequently, that the empirical performance of the policy is worse according to rollouts (second indicator).

among all repetitions, meaning that all actions point in the same direction and thus are identical. The new policy represented by the MLP with $\tanh(\cdot)$ after performing the Policy Improvement step is not able to capture the essence of the reward function. The indicators match the selected situation Figure 4.7b, where one can see that all actions take the value 1. An application of this MLP in a Policy Iteration algorithm is not possible. Switching to a linear output for the MLP has the desired effect for training, but comes at the price of exceeding the action space limits. When using a linear activation function, the angle span consists for most but not all repetitions of the largest possible value, indicating that actions change their sign. However, the action span demonstrates concisely their main issue. The largest action space, which can fit in the action space $\mathcal{A} = [-1, 1]$, is two. Thus, in all repetitions bad policies have been learned, which cannot be used naturally in the environment.

Judging from the empirical investigation, an obvious solution to ease the problems is to change the policy MLP to use a linear activation functions for its output. However, this creates the issue with a wrong image of the function, which must be avoided since the main task is to stay in $\mathcal{A}$ to be able to use the policy within the environment. Hence, a possible conclusion would be to combine a linear output of the MLP with some component-wise clipping operation, which moves its action output into the available action space. This clipping strategy would thus follow the same idea as the Direct-GIP approach and their projectors. A disadvantage of this approach is that clipping causes actions to get "corrected" automatically, which has the negative consequence that an MLP and its optimisation task are not aware that large action values outside of $\mathcal{A}$ are bad solutions. Clipping operations simply destroy information.

To move the action output of the MLP f_π inside a restricted set and also correct the case, where action values are too large, a more sophisticated approach than direct clipping is necessary. As possible solution, I propose to train the policy MLP with its output constrained to the unit ball and include a corresponding projection operation in the training process. The action space is therefore reduced from a hypercube to the smaller volume of a ball, but this should not impose practical limitations. The unit ball is a subset of the hypercube $[-1, 1]^{K_\mathcal{A}}$ and satisfies all requirements for DP algorithms, but also adds some pleasing properties regarding optimisation. Introducing Fitted-Actors with the unit ball as image results in the approach described in the next section.

4.6.3 Fitted-Actors that Live in the Unit Ball

By introducing a Fitted-Actor algorithm and an MLP f_π, which can use the unit ball as its image, it is possible to remove the $\tanh(\cdot)$ in the output layer of the MLP, which is the limiting factor during training policies via Supervised Regression. Hence, the policy MLP f_π now uses a linear output similar to the Critic and gets combined with a new projection operation to work in the $K_\mathcal{A}$-Ball $\mathcal{B}_{K_\mathcal{A}}$. Therefore, I introduce another projector $P_\mathcal{B} \colon \mathbb{R}^{K_\mathcal{A}} \to \mathcal{B}_{K_\mathcal{A}}$, which is given by

$$P_\mathcal{B}(x) = \frac{x}{\max\{1, \|x\|_A\}}. \tag{4.21}$$

This projector moves elements from the full vector space $\mathbb{R}^{K_\mathcal{A}}$ down to the unit ball $\mathcal{B}_{K_\mathcal{A}}$. There might be a minor conflict in the way to count dimensions, in particular, when talking about spheres embedded in vector spaces. Namely, an n sphere typically lives in an $n+1$ dimensional vector space. But since the interior of the ball is still the original (vector) space, it is easier to have the same index in my document and ignore the off-by-one issue from other literature or common textbooks. Furthermore, for the sake of convenience, it is also possible to omit the dimension index for the ball altogether in the remaining document. It will always be $K_\mathcal{A}$.

The first question that arises, is what is gainend from working in the unit ball. The ball consists of two parts, namely, the interior and its surface. The interior does not provide any technical difficulty, because it behaves during the optimisation just like an unrestricted Euclidean space, The surface of the ball however, makes optimisation on the sphere necessary. But here also resides the advantage of this construction. Opposed to the non-linear and oblique projection operator as defined in Equation (4.19), the projector on the surface of a sphere is linear and even orthogonal. Hence, I can benefit of the sphere structure on the surface and have a proper and well defined behaviour for the optimisation problem with existing solution techniques. It should be possible to switch on the fly between the interior and surface case. Whereas this will not be mathematically rigorous, it should still be an acceptable realisation for practical applications.

The possibility to exploit optimisation techniques on the sphere also carries over from the Fitted-Actor constructions back to a full Actor-Critic algorithm. Once the Actor is producing values inside of the unit ball, also the Critic must be aware of the new input domain. Hence, the projection steps would appear as part of the chain rule used to train the Actor through the Critic. Only by relying on $P_\mathcal{B}$ from Equation (4.21) as an orthogonal operator (when dealing with the surface of $\mathcal{B}$), it is possible to express gradients for the Actor in the tangent space and avoid the creation of spurious critical points. This would

result in a complex and challenging approach to DP with NN-VFA, which will require a sophisticated analysis on its own. Hence, I leave this as future research and focus on exploiting the Fitted-Actor approach.

Since the training of a Critic and an Actor is already decoupled for Fitted-Actors, working inside the ball is conceptually simple as I will demonstrate in the rest of this section. It is directly possible to alternate the Fitted-Actor approach by making $P_{\mathcal{B}}$ a part of the MLP and by replacing the clipping operation to stay within $\mathcal{B}_{K_{\mathcal{A}}}$ instead of $[-1,1]^{K_{\mathcal{A}}}$. The following text summarises some of the important notation and concepts regarding Supervised Regression from [Shen, 2018a,b]. Although the cited documents have all required expressions, they do not appear in the exact right form as they are needed for my work. This is most notably for the inclusion of the projector from Equation (4.21) and its differential map in these terms. As the first new concept for Supervised Regression, there exists now an explicit error function $E\colon \mathcal{A}\times\mathcal{A}\to\mathbb{R}_0^+$, which measures the distance between the output of f_π and the desired action value. A common choice for Euclidean spaces is the general form of a squared norm

$$E(a_1,a_2)=\frac{1}{2}\big\langle a_1-a_2,a_1-a_2\big\rangle_A^2=\frac{1}{2}\big\|(a_1+th-a_2)\big\|_A^2, \tag{4.22}$$

where A denotes a positive definite symmetric matrix with shape $K_{\mathcal{A}}\times K_{\mathcal{A}}$ and full rank. The two possible action inputs are denoted by $a_1,a_2\in\mathcal{A}$. This error function has the differential

$$\begin{aligned}
&\mathrm{D}_1E(a_1,a_2)[h]=\\
&\quad=\frac{1}{2}\frac{d}{dt}\bigg|_{t=0}(a_1+th-a_2)^{\mathsf{T}}A(a_1+th-a_2)\\
&\quad=\frac{1}{2}\frac{d}{dt}\bigg|_{t=0}\Big((a_1-a_2)^{\mathsf{T}}A(a_1-a_2)+t(a_1-a_2)^{\mathsf{T}}Ah+th^{\mathsf{T}}A(a_1-a_2)+t^2h^{\mathsf{T}}Ah\Big)\\
&\quad=(a_1-a_2)^{\mathsf{T}}Ah=\langle a_1-a_2,h\rangle_A\,,
\end{aligned} \tag{4.23}$$

where an arbitrary direction is denoted as usual by $h\in\mathcal{A}$. The second-order derivative of the error function reads as

$$\begin{aligned}
\mathrm{D}_1^2\,E(a_1,a_2)[h_1,h_2]&=\frac{d}{dt}\bigg|_{t=0}(a_1+th_2-a_2)^{\mathsf{T}}\,Ah_1\\
&=\frac{d}{dt}\bigg|_{t=0}\Big(a_1^{\mathsf{T}}Ah_1+th_2^{\mathsf{T}}Ah_1-a_2^{\mathsf{T}}Ah_1\Big)\\
&=h_2^{\mathsf{T}}Ah_1=\langle h_2,h_1\rangle_A\,.
\end{aligned} \tag{4.24}$$

To arrive at an optimisation problem, the parameters $\mathbf{U}$ of the policy MLP f_π, the sample states s_i with their corresponding action labels a_i^* and the error function define together the objective

$$\widetilde{\mathcal{J}}_\pi(\mathbf{U})\coloneqq\frac{1}{N}\sum_{i=1}^{N}E(\cdot,a_i^*)\circ f_\pi(\mathbf{U},s_i), \tag{4.25}$$

which of course is identical to Equation (4.20), but possesses the required shape to be able to insert the projection. The differential map is now the thing of interest. For some

direction $\mathbf{H} \in \boldsymbol{\mathcal{U}}$, it takes the form

$$\mathrm{D}_{\mathbf{U}}\, \widetilde{\mathcal{J}}_\pi(\mathbf{U})[\mathbf{H}] = \frac{1}{N} \sum_{i=1}^{N} \mathrm{D}_1\, E(f_\pi(\mathbf{U}, s_i), a_i^*) \circ \mathrm{D}_{\mathbf{U}}\, f_\pi(\mathbf{U}, s_i)[\mathbf{H}]. \tag{4.26}$$

Finally, inserting everything yields the expressions required to realise the optimisation task

$$\begin{aligned} \mathrm{D}_{\mathbf{U}}\, \widetilde{\mathcal{J}}_\pi(\mathbf{U})[\mathbf{H}] =& \frac{1}{N} \sum_{i=1}^{N} \left(f_\pi(\mathbf{U}, s_i) - a_i^*\right)^{\mathsf{T}} A \\ & \underbrace{\begin{bmatrix} \Phi_1^{\mathsf{T}} \left(I_{n_1} \otimes \phi_0\right)^{\mathsf{T}} & \dots & \Phi_L^{\mathsf{T}} \left(I_{n_L} \otimes \phi_{L-1}\right)^{\mathsf{T}} \end{bmatrix}}_{=: G_\pi^{(s_i)}(\mathbf{U}) \in \mathbb{R}^{K_{\mathcal{A}} \times N_{net}}} \underbrace{\begin{bmatrix} \mathrm{vec}(H_1) \\ \vdots \\ \mathrm{vec}(H_L) \end{bmatrix}}_{=: \mathrm{vec}(\mathbf{H}) \in \mathbb{R}^{N_{net} \times 1}}. \end{aligned} \tag{4.27}$$

Now I can apply the required changes to this expression to arrive at a Fitted-Actor algorithm, which is working in the unit ball. There are two possible realisations. The first is to alternate the MLP such that its output is inside the unit ball. This would be realised by applying the projector $P_{\mathcal{B}}$ from Equation (4.21) to the output vector after the last layer. The second would leave the MLP unchanged but would combine the corresponding input of the error function with $P_{\mathcal{B}}$. Thus, a new virtual error function $\widetilde{E}(a_1, a_2) = E(P_{\mathcal{B}}(a_1), a_2)$ is created, which possess a more complex geometry that will be clearly visible in its Hessian. For no particular reason, I have decided to include $P_{\mathcal{B}}$ through the error function. In both cases, the projector will change the differential map of the objective $\widetilde{\mathcal{J}}_\pi$, thus, one has to compute the differential of the projector first. It is given by

$$\mathrm{D}\, P_{\mathcal{B}}(x)[h] = \begin{cases} \left\langle \frac{1}{\|x\|_A} \left(I - \frac{Axx^{\mathsf{T}}}{\langle x, x \rangle_A}\right), h \right\rangle & \text{if } x \notin \mathcal{B} \\ \langle I, h \rangle & \text{else} \end{cases}. \tag{4.28}$$

Next, it has to be added as new term in the differential map of the objective. Due to the chain rule, the new term appears in Equation (4.27) between the differential of the error function and the leftmost term in each Ψ. This underlines the skewing effect on the error function. The differential $\mathrm{D}\, P_{\mathcal{B}}(x)[h]$ is independent of the parameters $\mathbf{U}$ and is always tied towards the matrix A used in the squared norm.

Lastly, I need a quick calculation for the Gauss Newton Approximation of the Hessian to be able to see the impact of the virtual error function. If I assume as usual that the policy MLP f_π is rich enough to allow for exact learning, then at any critical point $\mathbf{U}^* \in \boldsymbol{\mathcal{U}}$ the Hessian takes the form

$$\begin{aligned} \mathrm{D}_{\mathbf{U}}^2\, \widetilde{\mathcal{J}}_\pi(\mathbf{U}^*)[\mathbf{H}_1, \mathbf{H}_2] =& \frac{1}{N} \sum_{i=1}^{N} \Big(\left(\mathrm{D}_{\mathbf{U}} (f_\pi(\mathbf{U}^*, s_i) - a_i^*)[\mathbf{H}_2] \right)^{\mathsf{T}} A G_\pi^{(s_i)}(\mathbf{U}^*)\, \mathrm{vec}(\mathbf{H}_1) \\ & + \underbrace{\left(f_\pi(\mathbf{U}^*, s_i) - a_i\right)^{\mathsf{T}}}_{=0} A\, \mathrm{D}_{\mathbf{U}}\, G_\pi^{(s_i)}(\mathbf{U}^*)\, \mathrm{vec}(\mathbf{H}_1) \Big) \end{aligned} \tag{4.29}$$

$$\begin{aligned} =& \frac{1}{N} \sum_{i=1}^{N} \Big(\mathrm{vec}(\mathbf{H}_2)^{\mathsf{T}} (G_\pi^{(s_i)}(\mathbf{U}^*))^{\mathsf{T}} A G_\pi^{(s_i)}(\mathbf{U}^*)\, \mathrm{vec}(\mathbf{H}_1) \Big) \\ =& \mathrm{vec}(\mathbf{H}_2)^{\mathsf{T}} \frac{1}{N} G_\pi^{\mathsf{T}}(\mathbf{U}^*) \tilde{A} G_\pi(\mathbf{U}^*)\, \mathrm{vec}(\mathbf{H}_1), \end{aligned} \tag{4.30}$$

where the symmetrical block diagonal matrix $\tilde{A} = \text{diag}(A, \ldots, A)$ contains the Hessian of E for all samples. If the Hessian of E vanishes to have the default inner product for Euclidean spaces, then A is reduced to the identity matrix. In the middle between the two Jacobians $G_\pi^{(s_i)}(\mathbf{U})$, the changes to the error function due to the projector would appear. The skewing by $\mathrm{D}\, P_{\mathcal{B}}(x)[h]$ happens once from the left of A and once from the right and creates the symmetrical pattern $\text{Jac}_1\, P_{\mathcal{B}}(x)^\mathsf{T} A\, \text{Jac}_1\, P_{\mathcal{B}}(x)$.

The sharp eyed reader might already spot an issue with the rank of the projector's differential. It has an effect on the statements about the Jacobian from [Shen, 2018b]. In particular, for the one dimensional action space considered in this section, the differential is zero once the value is outside of $\mathcal{B}$, otherwise one. This seems to be an issue, but one can argue that this is not too harmful in practice. The contribution to Gradients and Hessians for input state samples, whose corresponding action output value lies outside of desired range, is zero. However, if policy parameters change due to other elements in the training data, these samples will contribute again to all expressions. Also, the critical point condition for the Actor MLP f_π is to some extend weaker, because the rank can be deficient due to $\mathrm{D}\, P_{\mathcal{B}}(x)[h]$. Yet, the additional solutions in parameter space, which become possible through the loss of rank of the projectors differential, correspond to MLPs with large action values. They get corrected by the projector anyway. For fitting a proper policy to the training data sampled in the unit ball, action values must be as well inside of the boundaries or need to be located at least on the boundary. Hence, the projection on the unit ball is not active in that regime such that solutions for critical points correspond to proper policies. For action spaces with two or more dimensions, the differential of the projection has full rank for most input values. Only the basis vectors result in a reduced rank. Since the set of unit vectors is small compared to the entire action space, this will not be a severe problem. Thus, properties of objective are not affected significantly and it is possible to realise a Gauss Newton algorithm for training.

To see the effect of working on the ball, I repeat the empirical verification from Section 4.6.2. The experimental setting is the same as before, but I now consider additionally policy MLPs, which produces outputs in the unit ball. Hence, I compare cases with $tanh()$ output for f_π, a plain linear output, which relies on clipping to produces actions inside $\mathcal{A}$, and the approach with linear output but using the ball projector. I show the results next to the those from the previous experiment in Figure 4.9.

The indicators reveal that the approach based on the unit ball can work partially and shows the desired properties, but also demonstrate certain issues. I will focus in Figure 4.9 only on the results for linear activation functions and the projector (green box plots). Box plots shown in red or blue are identical to Figure 4.8 and are repeated for convenience.

To the positive effects belong a slightly enhanced final fitting error and the maintained action span. Whereas the regression works better than having $\tanh(\cdot)$ in the output, the method constructed around the unit ball cannot achieve the same consistent small errors as for the unmodified linear output. The action span is in the range from zero to two, indicating that the method overcomes the limitations of regressions with $\tanh(\cdot)$ without suffering from leaving the action space.

The downside is visible with the average reward of the improved policy and the angle span. Similar to the pure linear activation function, the achieved expected rewards after the improvement cover a large range of values and performs worse than $\tanh(\cdot)$. The angles

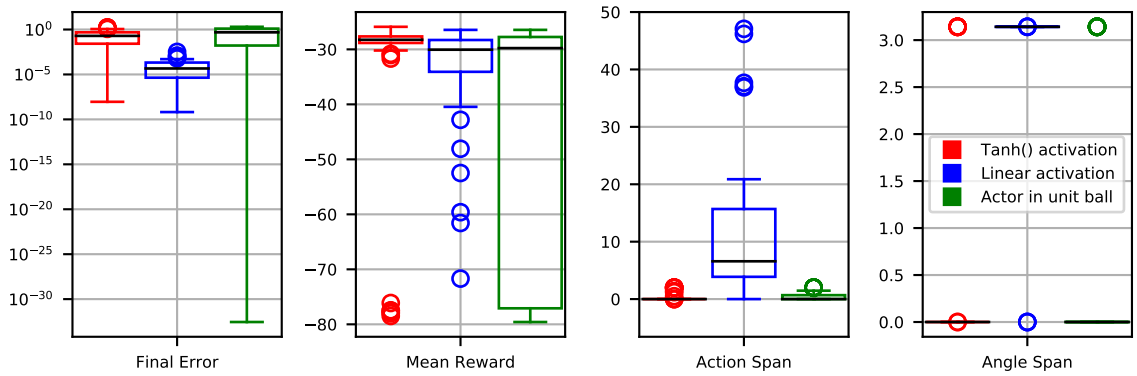


Figure 4.9: The empirical investigation for fitting MLPs to (s, a^*) tuples from Figure 4.8 extended with the method that works in the unit ball. When working with the unit ball, the outcome for final errors and the action span is improved when compared against $\tanh(\cdot)$. However, the average reward of the improved policy suffers and the distribution of angles between all actions and a reference value is compressed at zero. This would imply that the method constructed around the unit ball has struggle to achieve high performance in a PI algorithm.

between actions sit according to the box plot almost all at zero, which would imply that the method cannot overcome the change of sign for important actions.

Despite these results, I will still explore a full Fitted-Actor PI algorithm, which works on the unit ball. I expect that there will be performance issues, but it could serve as a possible solution to the original problem, namely avoiding spurious critical points by realising a reliable Policy Improvement based on Supervised Regression.

4.6.4 A Fitted-Actor Algorithm with Gauss Newton Optimisation

A Fitted-Actor algorithm resembles in many parts the procedure for an Actor-Critic approach from Algorithm 4 in Section 4.4.3. In fact, the most important change is that the Policy Improvement step is realised now as two separated tasks. The first task is the application of CEM to extract for all sample states s_i the corresponding best actions $a_i^* \in \mathcal{B}$ according to the current approximation f_Q of the Q-function. The second one encompasses the Supervised Regression to improve the policy MLP f_π by fitting the dataset. A minor change resides in the transition data used to execute Policy Evaluation. Algorithm 5 provides the entire Fitted-Actor procedure.

Line 3 emphasizes that the result of *Cross Entropy Method* is also used directly for training the Critic. The policy MLP f_π is no longer active in conjunction with the successor state and gets replaced by the action produced with CEM. Please note that $a_i'^*$ is not an optimal action in the sense that it belongs to Q^*, it is just an approximation for the best outcome of $\operatorname{argmax}_a f_Q(\mathbf{W}, s, a)$. By not involving f_π at all in the Critic, one obtains the advantage that the Policy Evaluation step has a higher quality, because $a_i'^*$ is more likely to be the correct value according to f_Q than the action produced by the Actor, i.e., $a_i'^* \neq a_i' = f_\pi(\mathbf{U}, s_i')$ in general. Furthermore, the fitted policies only serve for the empirical verification and visualisation of the training progress across all sweeps. Instead of executing the random search for actions every time a value is needed, one "compiles" the dataset

Algorithm 5 Fitted-Actor with Gauss Newton Optimisation

Hyper parameters: $\gamma \in (0,1)$, $\alpha_Q > 0$, $\alpha_\pi > 0$, $c = 10^{-5}$, $\epsilon \leq 10^{-5}$, $N \sim N_{net}$
Input:
- MLP $f_Q \in \mathcal{F}(K_\mathcal{S} + K_\mathcal{A}, \ldots, 1)$ with initialised parameters $\mathbf{W} \in \boldsymbol{\mathcal{W}}$
- MLP $f_\pi \in \mathcal{F}(K_\mathcal{S}, \ldots, K_\mathcal{A})$ with initialised parameters $\mathbf{U} \in \boldsymbol{\mathcal{U}}$

Output:
- $\mathbf{U}$ such that $f_\pi(\mathbf{U}, s)$ is a trained policy
- $\mathbf{W}$ such that $f_Q(\mathbf{W}, s, a) \approx Q_{f_\pi}(s, a)$

for sweep **in** sweeps **do**

// Preparation

Draw (s_i, a_i) for $i = 1, \ldots, N$ uniformly from $\mathcal{S} \times \mathcal{A}$

Construct transition tuples $(s_i, a_i, r_i, s'_i, a'^*_i)$ for all i

// Policy Evaluation : Critic

for i **in** i_Q **do**

Evaluate $F_Q(\mathbf{W}) \coloneqq [f_Q(\mathbf{W}, s_1, a_1) \ldots f_Q(\mathbf{W}, s_N, a_N)]^\mathsf{T} \in \mathbb{R}^N$

Evaluate $F'_Q(\mathbf{W}) \coloneqq [f_Q(\mathbf{W}, s'_1, a'_1) \ldots f_Q(\mathbf{W}, s'_N, a'_N)]^\mathsf{T} \in \mathbb{R}^N$

Compute $G_Q(\mathbf{W})$ and $G'_Q(\mathbf{W})$

Bellman Residual in Q: $\Delta^Q_\pi(\mathbf{W}) = F_Q(\mathbf{W}) - R_\pi - \gamma F'_Q(\mathbf{W})$

NMSBE: $\mathcal{J}_Q(\mathbf{W}) = \frac{1}{2N} \Delta^Q_\pi(\mathbf{W})^\mathsf{T} \Delta^Q_\pi(\mathbf{W})$

Gradient: $\nabla_\mathbf{W} \mathcal{J}_Q(\mathbf{W}) = \frac{1}{N} \Big(G_Q(\mathbf{W}) - \gamma G'_Q(\mathbf{W}) \Big)^\mathsf{T} \Delta^Q_\pi(\mathbf{W})$

Hessian: $\mathsf{H}_\mathbf{W} \mathcal{J}_Q(\mathbf{W}) = \frac{1}{N} \Big(G_Q(\mathbf{W}) - \gamma G'_Q(\mathbf{W}) \Big)^\mathsf{T} \Big(G_Q(\mathbf{W}) - \gamma G'_Q(\mathbf{W}) \Big)$

Solve for η: $(\mathsf{H}_\mathbf{W} \mathcal{J}_Q(\mathbf{W}) + c_Q I_{N_{net}})\, \eta = \nabla_\mathbf{W} \mathcal{J}_Q(\mathbf{W})$

Descent step: $\mathbf{W} \leftarrow \mathbf{W} - \alpha_Q \eta$

end for

// Direct-GIP : Construction of Dataset for Actor

$a^*_i \leftarrow \operatorname{argmax}_{a \in \mathcal{A}} f_Q(\mathbf{W}, s_i, a)$ for $i = 1, \ldots, N$

// Policy Improvement : Fitted-Actor

for i **in** i_π **do**

Objective: $\widetilde{\mathcal{J}}_\pi(\mathbf{U}) = \frac{1}{N} \sum_{i=1}^N E(f_\pi(\mathbf{U}, s_i), a^*_i)$

Gradient: $\nabla_\mathbf{U} \widetilde{\mathcal{J}}_\pi(\mathbf{U}) = \frac{1}{N} \sum_{i=1}^N G_\pi(\mathbf{U}, s_i)^\mathsf{T} \nabla_1 E(f_\pi(\mathbf{U}, s_i), a^*_i)$

Hessian: $\mathsf{H}_\mathbf{U} \widetilde{\mathcal{J}}_\pi(\mathbf{U}) = \frac{1}{N} \sum_{i=1}^N G_\pi(\mathbf{U}, s_i)^\mathsf{T} \mathsf{H}_1 E(f_\pi(\mathbf{U}, s_i), a^*_i) G_\pi(\mathbf{U}, s_i)$

Solve for η: $\Big(\mathsf{H}_\mathbf{U} \widetilde{\mathcal{J}}_\pi(\mathbf{U}) + c_\pi I_{N_{net}} \Big) \eta = \nabla_\mathbf{U} \widetilde{\mathcal{J}}_\pi(\mathbf{U})$

Descent step: $\mathbf{U} \leftarrow \mathbf{U} - \alpha_\pi \eta$

end for

// Training Progress

Evaluate f_π empirically using several rollouts

end for

s_i, a_i^* into a policy, which is defined on the entire state space. This blurs the distinction between the pure Critic-Only approach from Chapter 3 and the Fitted-Actor algorithm of this section.

The major difference of Fitted-Actors compared to Actor-Critic methods starts with Line 15. Instead of using the Critic directly for training the Actor, a dataset of state-action tuples is created. The explicit solving of $\operatorname{argmax}_{a\in\mathcal{A}} f_Q(\mathbf{W}, s, a)$ directly in the action space is done via CEM. A detailed description follows later in the text. In Lines 17 to 19, the components for training an Actor are implemented as an ordinary Supervised Regression problem. As outlined in Section 4.6.2, a non-linear least squares problem is solved with Gauss Newton optimisation. Consequently, there are now two linear equation systems to define descent directions for both the Actor and Critic in Lines 12 and 20. Both use as before Householder QR-Decomposition for solving.

To switch from the normal setting, i.e., using a hypercube for $\mathcal{A}$, to an Actor, which works with the unit ball, one just need to replace in Lines 17 to 19 the error function E with its varied counterpart $\tilde{E}$. This applies to the objective itself and also to its derivatives such that the differential map of the projector as given in Section 4.6.3 becomes active.

To search for optimal actions in Lines 3 and 15, the *Cross Entropy Method* is employed. Finally, Algorithm 6 contains the complete description of my realisation of CEM. For the sake of simplicity, I formulate the algorithm in a more general setting. Namely, the method is presented for solving the maximisation of an arbitrary function $f\colon \mathcal{X} \to \mathbb{R}$. This allows to use typical notation inside the algorithm without having to embed everything in the DP context of the remaining document.

Algorithm 6 Cross Entropy Method for maximising a scalar function

Hyper parameters:
- $i_{max} = 50$, $\epsilon = 10^{-10}$, $c_0 = 10$
- $N_{total} = 500$, $N_{keep} = 10\%\ N_{total}$, $N_{noisy} = 2\%\ N_{total}$

Input: function $f\colon \mathcal{X} \to \mathbb{R}$

Output: solution $x^* \in \mathcal{X}$ such that $f(x^*) \approx \max_{x\in\mathcal{X}} f(x)$

1: Initialise: $mean \sim \mathcal{U}$, $cov = c_0 \cdot I$, $i = 0$
2: **do**
3: $\quad x_i \sim \mathcal{N}(mean, cov)$ with $i = 1, \ldots, N_{total} - N_{noisy}$
4: $\quad x_i \sim \mathcal{U}()$ with $i = N_{noisy}, \ldots, N_{total}$
5: $\quad x_i = P_{\mathcal{X}}(x_i)$ for all $i = 1, \ldots, N_{total}$
6: $\quad s_i = f(x_i)$ for all $i = 1, \ldots, N_{total}$
7: $\quad$ Sort all x_i in descending order according to s_i
8: $\quad mean = Mean(x_1, \ldots, x_{N_{keep}})$, $cov = Cov(x_1, \ldots, x_{N_{keep}})$
9: $\quad i = i + 1$
10: **while** $i < i_{max}$ **and** $\|cov\|_\infty > \epsilon$

The initialisation in Line 1 produces a vector and matrix with a shape matching $\mathcal{X}$. For the application in Fitted-Actors, the space $\mathcal{X}$ can be the action space in its original form $\mathcal{A} = [-1, 1]^{K_\mathcal{A}}$ or the subset corresponding to the unit ball $\mathcal{B}$. An initial mean is drawn from the uniform distribution $\mathcal{U}$. The covariance matrix starts with a scaled identity matrix I, which posses a compatible shape. Lines 3 and 4 draw particles from a normal

distribution $\mathcal{N}$, which is specified by the current mean and covariance, and also from a uniform distribution $\mathcal{U}$ to escape bad local solutions. Line 5 ensures that all particles remain in $\mathcal{X}$, especially if drawn from $\mathcal{N}$, by applying an appropriate clipping or projection operation. Next, all the scores s_i of all particles x_i are computed (Line 6). Afterwards, the particles are sorted according to their quality (Line 7). The last important step in Line 8 uses the best performing particles to compute a new mean and covariance for the next iteration. The *Cross Entropy Method* terminates, if either the covariance becomes sufficiently small or if the maximal amount of iterations is reached. The last computed mean is then used as the best value in $\mathcal{X}$, i.e., it is assumed to provide a solution for $\max_{x \in \mathcal{X}} f(x)$. The hyper parameters specified in the beginning of the algorithm work reasonable well in practice for my application. Furthermore, with the selected values for all three variants of N, the resulting CEM is so fast to not be noticeable compared to other algorithmic components. Therefore, I have not spent sophisticated work to optimise them further.

4.6.5 Behaviour of a Fitted-Actor Policy Iteration Algorithm

With my last experiments, I want to test my approach for training an Actor on the unit ball within a full PI loop. I want to gain empirical insights on whether this method can be used and how it performs compared to the Actor-Critic method. The overall structure of the experiment is based on that from Section 4.5.1. I only have to include small adjustments due to the Supervised Regression. Based on the experience gained so far from all experiments, I restrict the search space of certain parameters.

I employ two MLPs $f_Q \in \mathcal{F}(K_\mathcal{S} + K_\mathcal{A}, 15, 15, 1)$ and $f_\pi \in \mathcal{F}(K_\mathcal{S}, 15, 15, K_\mathcal{A})$ to represent the Q-function and policy, respectively. The input dimension of MLPs need to match the environment. Hidden layers rely on the Bent-Identity, whereas the output layers depend on the usage. The Critic has always a linear output. The Actor either uses $\tanh(\cdot)$, the identity as activation combined with clipping or a linear output followed by the ball projector. Both optimisation tasks make use of a Gauss Newton descent algorithm. In the two optimisation tasks, learning rates are identical and set to $\alpha_Q = \alpha_\pi = 0.1$. The regularisation for the Hessian is set for both to $c_Q = c_\pi = 10^{-5}$. The MLPs f_Q and f_π are trained with each $i_Q = i_\pi = 1000$ descent steps. All parameters are initialised uniformly in the range $[-1, 1]$ before the first sweep. Furthermore, all parameters are persistent, i.e., the last parameters from the previous sweep serve as initialisation for the next one. I use for training $N = 3000$ state samples, which are placed uniformly in $\mathcal{S}$. They are transient, meaning I resample the training data in every sweep. Due to my results for the Actor-Critic algorithm and the Critic-Only PI experiment, I no longer investigate smaller values for N, which would allow for exploiting effects from over-parametrisation.

The PI algorithm starts from a random policy and improves it iteratively over 25 sweeps. Policies are evaluated as usual after every sweep with Monte Carlo rollouts. I create ten trajectories with each 500 transitions starting randomly in the state space. Details given in the experimental setup for Actor-Critic experiments from Section 4.5.1 apply.

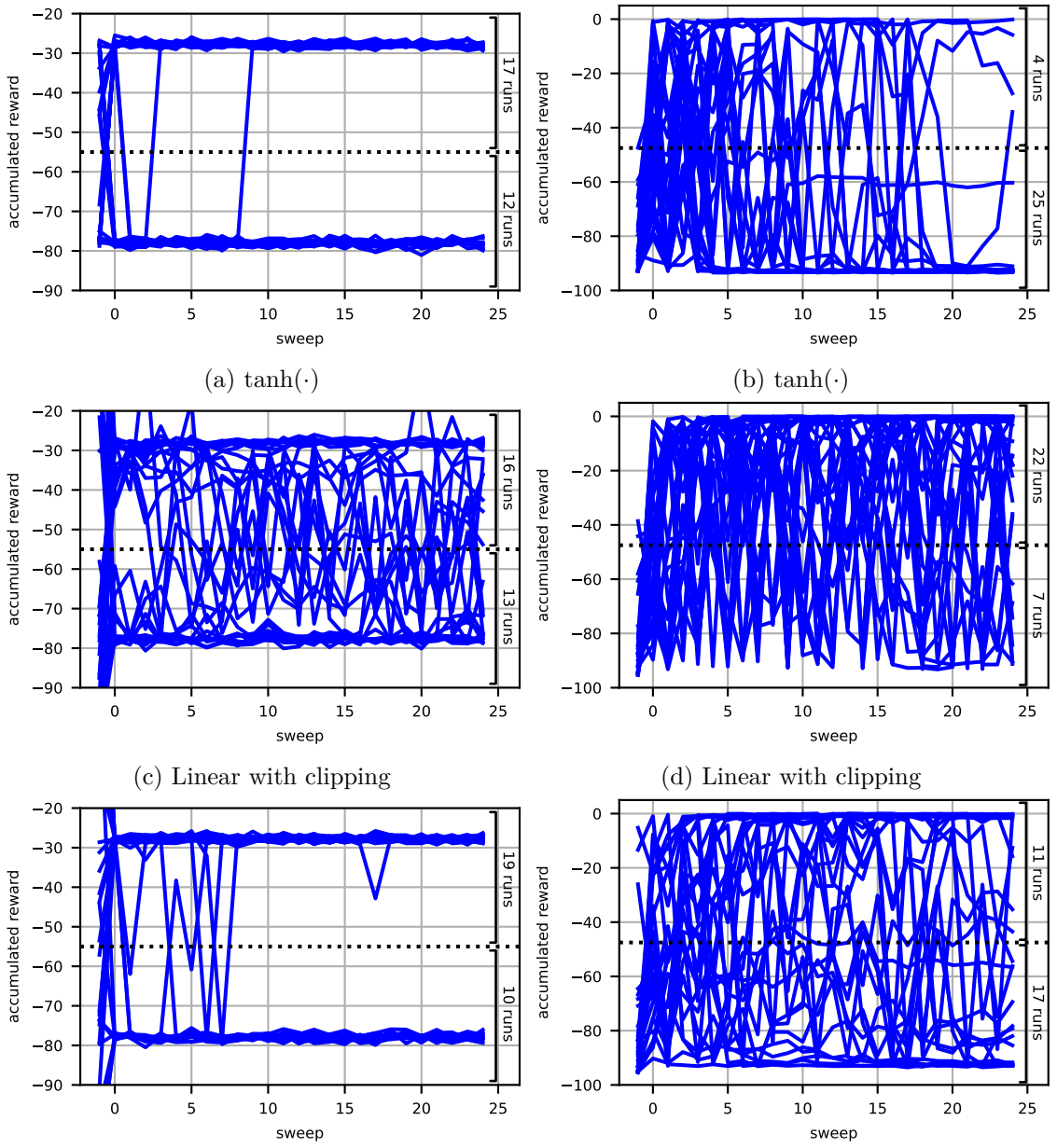
For the environments, I select *MyMountainCar-v1* from Section 2.6.3 and *MyCartPole-v1* from Section 2.6.4. Except for the necessary adjustments for the input and output dimensions of the MLPs, I use the same setting for optimisation in all environments. I vary the output activation of the policy MLP f_π and whether or not the training is performed on the unit ball. The entire training process is repeated ≈ 30 times from scratch.

Unfortunately, some repetitions are missing due to technical complications. Figures 4.10a, 4.10c and 4.10e show the results for *MyMountainCar-v1*, where Actors use $\tanh(\cdot)$ in the output, the identity with clipping and a linear function followed by the ball projector, respectively. Figures 4.10b, 4.10d and 4.10f contain counterparts for the other environment *MyCartPole-v1*.

The performance curves for the environment *MyMountainCar-v1*, i.e., the left column in Figure 4.10, reveal together two important facts. First, using an MLP f_π with linear output and component-wise clipping, which is not included during the training process, is to some extend harmful. The fitting of policies works unreliable, which results in a chaotic behaviour across all sweeps. Policies with all types of qualities show up, such that they cover the full spectrum of expected rewards (cf. Figure 4.10c). Second, adding the projection on the unit ball to MLPs, which use a linear output, results in a proper training process. This can be seen by the similarity of Figures 4.10a and 4.10e. Both plots show qualitatively the same structure without the strong jumps in performance as in Figure 4.10c. Unfortunately, the Fitted-Actor approach with a projection on the unit ball is still not able to overcome the last hurdle. Only suboptimal policies are produced as the best possible case. Policies, which achieve average discounted expected reward between -25 and -30, correspond to accelerating constantly to the right instead of swinging up. I conclude that the environment *MyMountainCar-v1* simply favours too strongly learning this kind of policy, since this happens for different approaches. The required level of detail, which is needed to create a working Policy Improvement step, are too subtle for an Actor-Critic or Fitted-Actor algorithm to be learned reliably. It seems, that a Critic-Only formulation, which computes actions on the fly, always results in a more robust algorithm compared to using an additional Actor with a second non-linear function approximation architecture. This can be seen with the experiment in Figure 4.5b, where a Critic-Only method is indeed able to solve the *MyMountainCar-v1* environment, because it achieves for some sweeps policies with zero accumulated rewards.

The right column in Figure 4.10, i.e., the results for the environment *MyCartPole-v1*, confirms mostly the statements from above. Again, one can see in Figures 4.10b and 4.10d that a linear output layer combined with clipping demonstrates a stronger chaotic behaviour than the usage of $\tanh(\cdot)$ as activation. A pure clipping of action values outside of the desired range destroys information. Yet, a linear output with clipping is able to produce from time to time working policies. Other than for the *MyMountainCar-v1* environment, Figures 4.10b and 4.10f are no longer that similar. Indeed, one can see in the figures that the amount of repetitions, where the MLP f_π can learn successfully a proper balancing policy at the end of training, is higher when working on the unit ball than for using $\tanh(\cdot)$ in the output layer. However, in this environment a linear output with clipping works better in terms of how often a good policy shows up, even if the training process itself is rather unstable.

The results for *MyCartPole-v1* match also the Actor-Critic method regarding the general ability to solve that environment as shown in Figure 4.5c. However, based on the available empirical insights, one has to conclude that using the Actor-Critic method is the better strategy, at least for this particular control problem.



(a) tanh(·)

(b) tanh(·)

(c) Linear with clipping

(d) Linear with clipping

(e) Linear with ball projector

(f) Linear with ball projector

Figure 4.10: The Fitted-Actor algorithm in *MyMountainCar-v1* (left column) and *MyCartPole-v1* (right column). Each figure shows the expected rewards for the Actor with different choices for its output layer. All runs are grouped into two categories based on final reward value. **a) and b):** The reference behaviour with tanh(·) as activation, which suffers from constant policies. **c) and d):** A linear output with clipping shows chaotic behaviour during training, because leaving $\mathcal{A}$ is not punished. Yet, it is still possible to learn a good policy by chance. **e) and f):** Projection on the unit ball enables usage of linear outputs, but its performance depends on the environment. For *MyMountainCar-v1*, one only restores the behaviour of tanh(·). For *MyCartPole-v1*, one can enhance the quality of policies compared to tanh(), but a linear output with clipping results more often in well performing policies.

4.6.6 Remark

Unfortunately, it is difficult to draw a clear conclusion regarding whether Fitted-Actors or Actor-Critics should be used. The approach based on the unit ball can work, but it is not a universal remedy. When investigating individual components, one can see that the limitations of $\tanh(\cdot)$ as activation function in the output layer or the mismatch between expected and actual action spaces can be overcome. But as soon as all components are assembled together to form a full PI algorithm, Fitted-Actors do not result in a method, which performs significantly better than a direct Actor-Critic approach. Still, a positive aspect is that using the projector on the unit ball restores the correct behaviour for a linear output. It is a sound method to achieve a restriction to $\mathcal{A}$. A negative property is that this projection introduces more moving parts in the algorithm, which have to play together smoothly and therefore create more work in setting up a proper application. Hence, one can also argue to use a plain Critic-Only approach to avoid entirely all kinds of problems related to an Actor. Yet, this statement is tied towards being able to extract fast enough actions from a Q-function to steer the system under control properly.

In the end and once NN-VFA is involved, realising a DP algorithm remains an art on its own. I can incorporate all the insights from my analysis, but finding the right experimental setup is non-trivial. Even if an algorithm is well understood and analysed, there are so many moving parts which all need careful tuning such that an working algorithm is the result. As an example that covers all the components of an Actor-Critic algorithm, consider the different behaviour of the same algorithm for the two versions of the Cart Pole benchmarks. Despite seemingly harmless alternations, one version allows for a successful application of the method, the other does not.

Chapter 5

Conclusion

Dynamic Programming with Neural Network based Value Function Approximation has become one of the most powerful learning paradigms in both research and application over the recent years. Despite the superior performance, its training and convergence analysis has remained challenging due to an incomplete theoretical understanding how MLPs affect the common DP setting. Namely, most previous theoretical analysis of DP with VFA approach these challenges from the perspective of minimising the Mean Squared Projected Bellman Error and Linear Value Function Approximation architectures. Hence, they are arguably incompatible with recent successful algorithms. My work bridges this gap by working with tools from non-convex optimisation and by investigating the critical point conditions of the related optimisation problems.

For DP algorithms, there are two important components. The first is the computation of a Q-function for some given policy. The second is the improvement of this policy, once a Q-function is available. Both components need to be addressed both individually and as entangled entity.

I tackle the first component in DP, namely the computation of value- or Q-functions, by employing non-convex optimisation for minimising the MSBE. More precisely, I work with the Neural Mean Squared Bellman Error, which is the objective resulting from combining MLPs as the Non-Linear Value Function Approximation architecture with the ordinary MSBE. I address both the discrete and continuous state space setting. Within discrete spaces, exact learning is possible and one can minimise the NMSBE directly. Obtained solutions correspond to the ground truth. However, for continuous spaces, a sampling based calculation of the NMSBE and the use of a Residual Gradient formulation become necessary. This leads to a more complicated objective and the important insight that this objective now also allows for bad solutions. By using an approach based on non-convex optimisation this finding has become possible. A second important outcome, which underlines why non-convex optimisation should be used, is that my analysis unveils the possibility to utilise approximated second-order information of the cost function for both discrete and continuous state space formulations. My investigation results in an efficient Approximated Newton method, namely a Gauss Newton Residual Gradient algorithm, which possesses the typical and pleasing convergence properties.

For computing improved policies, i.e., the second component in DP, I analyse the class of Actor-Critic algorithms. In every variation of Policy Improvement, a first building block is the extraction of optimal actions from a given Q-function or its approximation. To enable the usage of non-convex optimisation, an important result of my work are additional constraints on the reward signal, which ensure its compatibility with derivative based optimisation. A second block in Actor-Critic algorithms are parametrised policies themselves. They are trained by maximising an (approximated) Q-function for all states in

the state space. Ensuring a proper outcome is getting much more complicated, because the Q-function is in general a non-linear and non-convex objective, but due to my critical points analysis of the objective, the complexity of the optimisation process becomes manageable. As part of my investigation through non-convex optimisation, I have unveiled an additional issue residing in the geometry of the action space. Namely, the Critic expects input values from a different space than it actually receives. To overcome these geometrical complications, I propose to decouple Actor and Critic training by switching to a Fitted-Actor method. Despite this method realising overall a sound Policy Iteration procedure, which handles the geometry of the action space correctly, it does not always produce well-performing policies. Further effort needs to be spent to address the training of parametrised policies, which can use the existing foundation provided by my work.

For all cases, non-convex optimisation is a promising methodology, which takes the geometry of a problem explicitly into account. It can answer open issues in Neuro-Dynamic Programming and, thus, also in Deep Reinforcement Learning. Furthermore, it unveils important details for the implementation of efficient algorithms. Relying on non-convex optimisation renders difficulties in the formulation of an optimisation task visible and provides guidance for design choices. My work also establishes a foundation for incorporating other Neural Network architectures such as convolutional networks or transformers into Neuro-Dynamic Programming. Additionally, other loss functions could be investigated as well through non-convex optimisation. This will require additional and significant work, but will also outline their limitations and advantages. Non-convex optimisation is a useful toolset when applied correctly and will always reveal details about the task. However, it is not rendering the application and implementation of such algorithms trivial. A thorough analysis and investigation of the concrete problem at hand is always required.

Appendix A

Step by Step Calculations

This chapter contains calculations with detailed steps and instructions. I have not included them in the main text, because their value and insights do not justify an increased complexity of the storyline. My work over here falls in the category "tedious but straightforward".

Differential Map of the Error Function For the error function $E\colon \mathbb{R}^K \to \mathbb{R}$ and some evaluated function $F \in \mathbb{R}^K$, I have

$$\begin{aligned}
E(F) =& \frac{1}{2}\Big(F - P_\pi\big(R_\pi + \gamma F\big)\Big)^\mathsf{T} \Xi \Big(F - P_\pi\big(R_\pi + \gamma F\big)\Big) \\
\mathrm{D}\, E(F)[h] =& \frac{1}{2}\left.\frac{d}{dt}\right|_{t=0} \left[\Big((F+th) - P_\pi\big(R_\pi + \gamma(F+th)\big)\Big)^\mathsf{T} \Xi \Big((F+th) - P_\pi\big(R_\pi + \gamma(F+th)\big)\Big)\right] \\
=& \frac{1}{2}\Big[\Big(h - P_\pi \gamma h\Big)^\mathsf{T} \Xi \Big((F+th) - P_\pi\big(R_\pi + \gamma(F+th)\big)\Big) \\
&+ \Big((F+th) - P_\pi\big(R_\pi + \gamma(F+th)\big)\Big)^\mathsf{T} \Xi \Big(h - P_\pi \gamma h\Big)\Big]_{t=0} \\
=& \frac{1}{2}\left[\Big(h - P_\pi \gamma h\Big)^\mathsf{T} \Xi \Big(F - P_\pi\big(R_\pi + \gamma F\big)\Big) + \Big(F - P_\pi\big(R_\pi + \gamma F\big)\Big)^\mathsf{T} \Xi \Big(h - P_\pi \gamma h\Big)\right] \\
=& \Big(F - P_\pi\big(R + \gamma F\big)\Big)^\mathsf{T} \Xi \Big(I_K - \gamma P_\pi\Big) h \\
=& \left\langle \Big(I_K - \gamma P_\pi\Big)^\mathsf{T} \Xi \Big(F - P_\pi\big(R + \gamma F\big)\Big), h \right\rangle .
\end{aligned}$$

Thus, according to Riesz, the gradient of E with respect to F is

$$\nabla_F E(F) = \Big(F - P_\pi\big(R + \gamma F\big)\Big)^\mathsf{T} \Xi \Big(I_K - \gamma P_\pi\Big).$$

Differential Map of an MLP Consider an MLP $f \in \mathcal{F}(n_0, n_1, \ldots, n_{L-1}, n_L)$ and an input $s \in \mathcal{S}$. To calculate the differential map of f for s at the point $\mathbf{W} \in \boldsymbol{\mathcal{W}}$ and a direction $\mathbf{H} \in \boldsymbol{\mathcal{W}}$, first start with a single layer l of the MLP. I have

$$\mathrm{D}_{W_l} f(\mathbf{W}, s)[H_l] = \mathrm{D}_2\, \Lambda_L(W_L, \phi_{L-1}) \circ \ldots \circ \mathrm{D}_2\, \Lambda_{l+1}(W_{l+1}, \phi_l) \circ \mathrm{D}_1\, \Lambda_l(W_l, \phi_{l-1})[H_l],$$

where $\mathrm{D}_1\, \Lambda_l(W_l, \phi_{l-1})[H_l]$ and $\mathrm{D}_2\, \Lambda_l(W_l, \phi_{l-1})[h_{l-1}]$ refer to the derivative of layer mapping Λ_l with respect to the first and the second argument, respectively. For the layer definition

in Equation (2.21), I obtain

$$\begin{aligned}
\mathrm{D}_1\,\Lambda_l(W_l,\phi_{l-1})[H_l] &= \left.\frac{d}{dt}\right|_{t=0} \begin{bmatrix} \vdots \\ \sigma\left((W_{l,k}+t\cdot H_{l,k})^\intercal \cdot \begin{bmatrix} \phi_{l-1} \\ 1 \end{bmatrix}\right) \\ \vdots \end{bmatrix} \\
&= \begin{bmatrix} \vdots \\ \dot\sigma(\cdots) H_{l,k}^\intercal \begin{bmatrix} \phi_{l-1} \\ 1 \end{bmatrix} \\ \vdots \end{bmatrix}_{t=0} \\
&= \mathrm{diag}\left(\dot\phi_l\right) H_l^\intercal \begin{bmatrix} \phi_{l-1} \\ 1 \end{bmatrix} \\
&=: \Sigma_l \cdot H_l^\intercal \cdot \tilde\phi_{l-1}
\end{aligned}$$

and

$$\begin{aligned}
\mathrm{D}_2\,\Lambda_l(W_l,\phi_{l-1})[h_{l-1}] &= \left.\frac{d}{dt}\right|_{t=0} \begin{bmatrix} \vdots \\ \sigma\left(W_{l,k}^\intercal \cdot (\begin{bmatrix} \phi_{l-1} \\ 1 \end{bmatrix} + t\cdot \begin{bmatrix} h_{l-1} \\ 0 \end{bmatrix})\right) \\ \vdots \end{bmatrix} \\
&= \begin{bmatrix} \vdots \\ \dot\sigma(\cdots) W_{l,k}^\intercal \begin{bmatrix} h_{l-1} \\ 0 \end{bmatrix} \\ \vdots \end{bmatrix}_{t=0} \\
&= \mathrm{diag}\left(\dot\phi_l\right) W_l^\intercal \begin{bmatrix} h_{l-1} \\ 0 \end{bmatrix} \\
&=: \Sigma_l \cdot \bar W_l^\intercal \cdot h_{l-1}.
\end{aligned}$$

The term $\Sigma_l \in \mathbb{R}^{n_l \times n_l}$ is a diagonal matrix with its entries being the derivatives of the activation function with respect to the input $\dot\phi_l$. Inside of $\dot\phi_k$, the derivative of the activation function $\dot\sigma(\ldots)$ is contained for all units in layer l. The input to $\dot\phi_l$ is the unmodified output ϕ_{l-1} of the truncated MLP. By writing $\bar W$, I indicate that the last row is cut off due to the multiplication by zero. The notation $\tilde\phi$ implies that the layer output is extended with an additional 1. This resembles homogenous coordinates as they are used with the special Euclidean group $SE(3)$ for computer vision applications. Inserting these parts yields for the differential map

$$\mathrm{D}_{W_l} f(\mathbf{W}, s)[H_l] = \Sigma_L \bar W_L^\intercal \cdot \Sigma_{L-1} \bar W_{L-1}^\intercal \cdots \Sigma_{l+1} \bar W_{l+1}^\intercal \cdot \Sigma_l H_l^\intercal \tilde\phi_{l-1}.$$

To shorten this expression let me construct a sequence of matrices for all $l = L-1, \ldots, 1$ as

$$\Psi_l \coloneqq \Sigma_l \bar W_{l+1} \Psi_{l+1} \in \mathbb{R}^{n_l \times n_L},$$

with $\Psi_L \equiv 1$ due to the activation function in the last layer being the identity function. Now I can write compactly

$$\mathrm{D}_{W_l} f(\mathbf{W}, s)[H_l] = \Psi_l^\mathsf{T} H_l^\mathsf{T} \phi_{l-1}.$$

To arrive at the expression shown in the main text consider a matrix $A \in \mathbb{R}^{n\times m}$ and a compatible column vector $b \in \mathbb{R}^{n\times 1}$. When denoting by $A_1, \cdots, A_m$ the m columns of A, one can show by straightforward computation the identity

$$A^\mathsf{T} \cdot b = \begin{bmatrix} A_1^\mathsf{T} b \\ \vdots \\ A_m^\mathsf{T} b \end{bmatrix} = \begin{bmatrix} b^\mathsf{T} A_1 \\ \vdots \\ b^\mathsf{T} A_m \end{bmatrix} = \begin{bmatrix} b^\mathsf{T} & & \\ & \ddots & \\ & & b^\mathsf{T} \end{bmatrix} \begin{bmatrix} A_1 \\ \vdots \\ A_m \end{bmatrix} = \left(I_{m\times m} \otimes b^\mathsf{T}\right) \cdot \mathrm{vec}(A).$$

By setting $A = H_l$ and $b = \phi_{l-1}$, I get

$$\mathrm{D}_{W_l} f(\mathbf{W}, s)[H_l] = \Psi_l^\mathsf{T} \left(I_{n_l} \otimes \phi_{l-1}^\mathsf{T}\right) \mathrm{vec}(H_l).$$

Finally, I can combine the expressions for all layers and produce the full differential map with respect to all parameters

$$\mathrm{D}_{\mathbf{W}} f(\mathbf{W}, s)[\mathbf{H}] = \underbrace{\begin{bmatrix} \Psi_1^\mathsf{T} \left(I_{n_1} \otimes \phi_0^\mathsf{T}\right) & \dots & \Psi_L^\mathsf{T} \left(I_{n_L} \otimes \phi_{L-1}^\mathsf{T}\right) \end{bmatrix}}_{\in \mathbb{R}^{n_L \times N_{net}}} \cdot \underbrace{\begin{bmatrix} \mathrm{vec}(H_1) \\ \vdots \\ \mathrm{vec}(H_L) \end{bmatrix}}_{\in \mathbb{R}^{N_{net}\times 1}},$$

where the MLP input ϕ_0 is just the input s. For the application with the MSBE in my work, I always have $n_L = 1$, because the value function maps to a scalar value. When using all N inputs at once, I arrive at the expression $G(\mathbf{W}) \in \mathbb{R}^{N\cdot n_L \times N_{net}}$ as shown in Equation (3.4)

$$\mathrm{D}_{\mathbf{W}} F(\mathbf{W})[\mathbf{H}] = \underbrace{\begin{bmatrix} \Psi_1^\mathsf{T} \left(I_{n_1} \otimes \phi_0^{(1)\mathsf{T}}\right) & \dots & \Psi_L^\mathsf{T} \left(I_{n_L} \otimes \phi_{L-1}^{(1)\mathsf{T}}\right) \\ \vdots & \ddots & \vdots \\ \Psi_1^\mathsf{T} \left(I_{n_1} \otimes \phi_0^{(N)\mathsf{T}}\right) & \dots & \Psi_L^\mathsf{T} \left(I_{n_L} \otimes \phi_{L-1}^{(N)\mathsf{T}}\right) \end{bmatrix}}_{=: G(\mathbf{W}) \in \mathbb{R}^{N\cdot n_L \times N_{net}}} \cdot \begin{bmatrix} \mathrm{vec}(H_1) \\ \vdots \\ \mathrm{vec}(H_L) \end{bmatrix}.$$

The superscript $(\cdot)^{(i)}$ indicates that the layer outputs ϕ_l arise from the i-th state in the input layer.

Definition of $\widetilde{G}(\mathbf{W})$ Equation (3.16) originates directly from the difference of $G(\mathbf{W})$ and $G'(\mathbf{W})$. I have

$$\begin{aligned} \widetilde{G}(\mathbf{W}) =& G(\mathbf{W}) - \gamma G'(\mathbf{W}) \\ =& \begin{bmatrix} G(\mathbf{W})_{11} & \cdots & G(\mathbf{W})_{1L} \\ \vdots & \ddots & \vdots \\ G(\mathbf{W})_{N1} & \cdots & G(\mathbf{W})_{NL} \end{bmatrix} - \gamma \begin{bmatrix} G'(\mathbf{W})_{11} & \cdots & G'(\mathbf{W})_{1L} \\ \vdots & \ddots & \vdots \\ G'(\mathbf{W})_{N1} & \cdots & G'(\mathbf{W})_{NL} \end{bmatrix} \\ =& \begin{bmatrix} \widetilde{G}(\mathbf{W})_{11} & \cdots & \widetilde{G}(\mathbf{W})_{1L} \\ \vdots & \ddots & \vdots \\ \widetilde{G}(\mathbf{W})_{N1} & \cdots & \widetilde{G}(\mathbf{W})_{NL} \end{bmatrix} \end{aligned}$$

with the blocks

$$\widetilde{G}(\mathbf{W})_{ij} = \underbrace{\Psi_j^{\mathsf{T}} \left(I_{n_j} \otimes {\phi_{j-1}}^{(i)} \right)^{\mathsf{T}}}_{G(\mathbf{W})_{ij}} - \gamma \underbrace{\Psi_j'^{\mathsf{T}} \left(I_{n_j} \otimes {\phi'_{j-1}}^{(i)} \right)^{\mathsf{T}}}_{G'(\mathbf{W})_{ij}}.$$

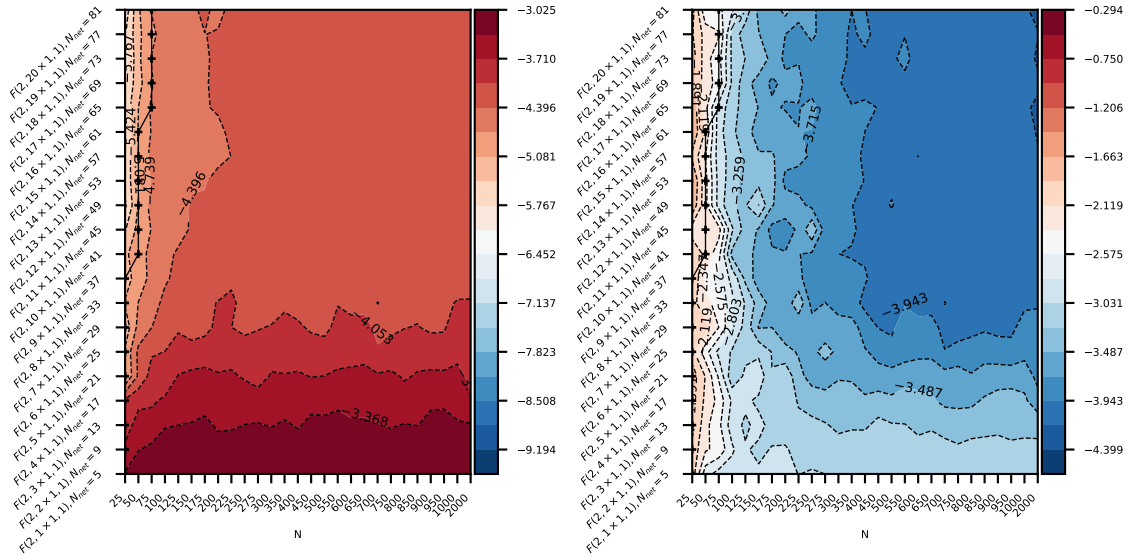
by pairing each block in the matrices. No further simplifications, which would allow for more insights, are possible.

Appendix B

Skipped Figures and Results

This chapter contains figures and results, which I have not included in the main text. Their value and insight does not justify an increased complexity of the reading flow.

Generalisation Experiments The following two contour plots belong to Figure 3.11 and show the outcome for MLPs with only one hidden layer. The parameters and setting is that of the generalisation experiment in Section 3.5.3.

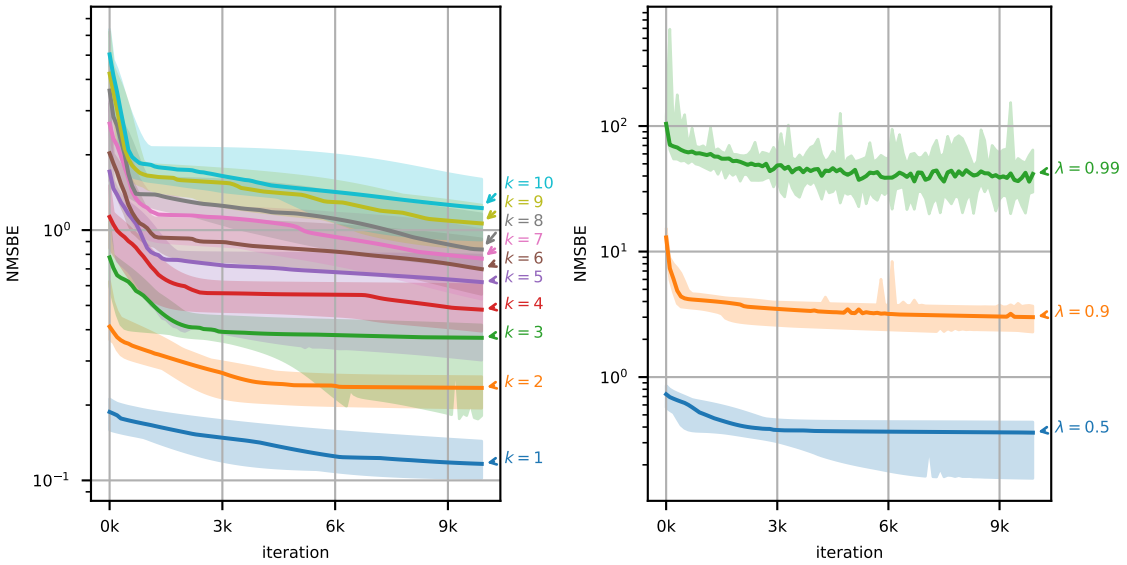


(a) (b)

Figure B.1: The training and test error of different MLP architectures (ordinate) for various sample sizes N (abscissa). I use a logarithmic scale $Z = \log_{10}(E)$, where E is the original error and Z its plotted value. Red indicates higher errors. In all plots the solid line represents the condition $N_{net} = N$. Please refer to Figure 3.11 for a complete description. **Left column:** Training error. **Right column:** Test error.

Due to the small number of parameters in the MLP, it is only for $N \leq 75$ possible to observe the effect of over-parametrisation. However, the tiny MLPs are not rich enough to represent the value function and do not achieve sufficiently low error.

Multistep Experiments This is an alternative version for showing the first-order only gradient descent with different sizes of lookahead. Figure B.2 aims to complement the figures in the convergence investigation from Section 3.5.4.



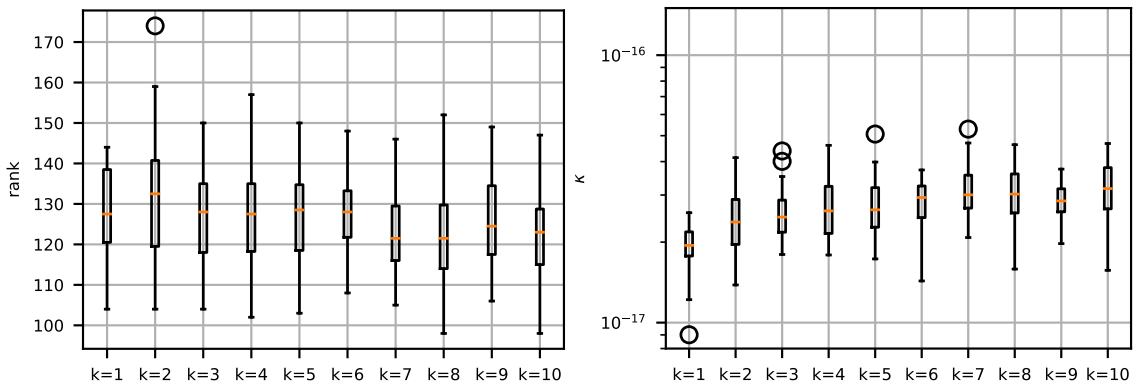
(a) Compound Method

(b) TD(λ)-like Method

Figure B.2: Larger lookahead can have a beneficial impact onto the convergence speed. This plots shows the same data as in Section 3.5.4, but since only the first-order methods are shown for different k or λ, it is possible to see the beneficial impact on the convergence speed at the beginning of training.

Due to the missing second-order optimisation, the limits for the y-axis become more suitable for the first-order gradient descent experiments. It is possible to see the enhanced convergence speed, once the lookahead k is increased. This is the known desirable behaviour for multistep lookahead methods, at least for first-order gradient descent. But one can also observe the price to pay in Figure B.2a. Namely, the training error also increased with larger k. Despite the faster decay of the error, the overall outcome is worse. The TD(λ) method behaves a bit differently. Here, the choice of λ is a trade-off. This can be seen in Figure B.2b and matches the literature, e.g., [Bertsekas, 2012].

The following figure shows the initial ranks and inverse condition numbers for the experiments from Section 3.5.4. Although it is possible to extract the same information directly from the figures in that section, it is rather sophisticated due to the size of those figures. Thus, Figure B.3 depicts the distribution of initial ranks and values for κ as box plot for the compound operator $\mathrm{T}_\pi^{(k)}$.



(a) Initial ranks

(b) Initial κ values

Figure B.3: These box plots emphasize the initial rank or inverse condition numbers of Jacobians for various MLPs before any training happened. This figure is essentially a magnified version of all corresponding figures in Section 3.5.4.

The randomly initialised MLPs do not have a Jacobian with full rank. The behaviour for the TD(λ) variation is similar, i.e., the Jacobian of the loss behaves the same, even if more terms contribute to it or a different weighting is involved.

The important property is, that for second-order methods the rank quickly goes up, whereas for first-order training, the loss of a full rank remains an issue. This can see in figures from Section 3.5.4.

Bibliography

P. A. Absil, R. Mahony, and R. Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, 2008.

M. Adibi and J. Van der Woude. Secondary frequency control of microgrids: An online reinforcement learning approach. *IEEE Transactions on Automatic Control*, 2022.

B. Amos, L. Xu, and J. Z. Kolter. Input convex neural networks. In *Proceedings of the* 34^{th} *International Conference on Machine Learning*, 2017.

M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, Pieter A., and W. Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

M. Anthony and P. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, 1999.

K. A. Asadi, N. Parikh, R. E. Parr, G. D. Konidaris, and M. L. Littman. Deep radial-basis value functions for continuous control. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 2021.

L. C. Baird and A. H. Klopf. Reinforcement learning with high-dimensional continuous actions. Technical report, Wright Laboratory, Wright-Patterson Air Force Base, Tech. Rep. WL-TR-93-1147, 1993.

L. C. Baird III. Residual algorithms: Reinforcement learning with function approximation. In *Proceeding of the* 12^{th} *International Conference on Machine Learning*, 1995.

L. C. Baird III and A. W. Moore. Gradient descent for general reinforcement learning. In *Advances in Neural Information Processing Systems*, 1999.

A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5), 1983.

R. Behling, D. S. Gonçalves, and S. A. Santos. Local convergence analysis of the levenberg-marquardt framework for nonzero-residue nonlinear least-squares problems under an error bound condition. *Journal of Optimization Theory and Applications*, 183(3), 2019.

R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause. Safe model-based reinforcement learning with stability guarantees. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

D. P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, 4^{th} edition, 2012.

D. P. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

D. P. Bertsekas, V. Borkar, and A. Nedić. Improved temporal difference methods with linear function approximation. In *Learning and Approximate Dynamic Programming*. IEEE Press, 2004.

N. Bhat, V. Farias, and C. C. Moallemi. Non-parametric approximate dynamic programming via the kernel method. In *Advances in Neural Information Processing Systems*, volume 25, 2012.

W. Böhmer, S. Grünewälder, Y. Shen, M. Musial, and K. Obermayer. Construction of approximation spaces for reinforcement learning. *Journal of Machine Learning Research*, 14(1), 2013.

H. Bojun. Steady state analysis of episodic reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, 2020.

D. Brandfonbrener and J. Bruna. Geometric insights into the convergence of nonlinear TD learning. In *International Conference on Learning Representations*, 2020.

G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *Computing Research Repository*, arXiv:1606.01540, 2016.

Q. Cai, Z. Yang, J. D. Lee, and Z. Wang. Neural temporal-difference learning converges to global optima. *Computing Research Repository*, arXiv:1905.10027, 2019.

A. I. Cowen-Rivers, D. Palenicek, V. Moens, M. A. Abdullah, A. Sootla, J. Wang, and H. Bou-Ammar. Samba: Safe model-based & active reinforcement learning. *Machine Learning*, 111(1), 2022.

W. Dabney and P. Thomas. Natural temporal difference learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014.

T. de Bruin, J. Kober, K. P. Tuyls, and R. Babuska. The importance of experience replay database composition in deep reinforcement learning. In *Deep Reinforcement Learning Workshop*, 2015.

J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de las Casas, C. Donner, L. Fritz, C. Galperti, A. Huber, J. Keeling, M. Tsimpoukelli, J. Kay, A. Merle, J. Moret, S. Noury, F. Pesamosca, D. Pfau, O. Sauter, C. Sommariva, S. Coda, B. Duval, A. Fasoli, P. Kohli, K. Kavukcuoglu, D. Hassabis, and M. Riedmiller. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897), 2022.

J. Dodge, T. Prewitt, R. Tachet des Combes, E. Odmark, R. Schwartz, E. Strubell, A. S. Luccioni, N. A. Smith, N. DeCario, and W. Buchanan. Measuring the carbon intensity of ai in cloud instances. In *ACM Conference on Fairness, Accountability, and Transparency*, 2022.

B. Eysenbach, R. R. Salakhutdinov, and S. Levine. Search on the replay buffer: Bridging planning and reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

R. V. Florian. Correct equations for the dynamics of the cart-pole system. Technical report, Center for Cognitive and Neural Studies (Coneural), Romania, 2007.

J. Fu, A. Kumar, M. Soh, and S. Levine. Diagnosing bottlenecks in deep q-learning algorithms. In *Proceedings of the* 36^{th} *International Conference on Machine Learning*, volume 97, 2019.

J. García and F. Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(42), 2015.

M. Geist and O. Pietquin. Algorithmic survey of parametric value function approximations. *IEEE Transactions on Neural Networks and Learning Systems*, 24(6), 2013.

M. Gottwald and H. Shen. On the compatibility of multistep lookahead and hessian approximation for neural residual gradient. In *The Multi-disciplinary Conference on Reinforcement Learning and Decision Making*, 2022.

M. Gottwald, D. Meyer, H. Shen, and K. Diepold. Learning to walk with prior knowledge. In *International Conference on Advanced Intelligent Mechatronics*, 2017.

M. Gottwald, M. Guo, and H. Shen. Neural value function approximation in continuous state reinforcement learning problems. In *European Workshop on Reinforcement Learning*, 2018.

M. Gottwald, S. Gronauer, H. Shen, and K. Diepold. Analysis and optimisation of bellman residual errors with neural function approximation. *Computing Research Repository*, arXiv:2106.08774, 2021.

M. Gottwald, H. Shen, and K. Diepold. A critical point analysis of actor-critic algorithms with neural networks. In 6^{th} *IFAC Conference on Intelligent Control and Automation Sciences*, 2022.

M. Granzotto, R. Postoyan, L. Buşoniu, D. Nešić, and J. Daafouz. Exploiting homogeneity for the optimal control of discrete-time systems: application to value iteration. In 60^{th} *IEEE Conference on Decision and Control*, 2021.

S. Gronauer and M. Gottwald. The successful ingredients of policy gradient algorithms. In *Proceedings of the* 30^{th} *International Joint Conference on Artificial Intelligence*, 2021.

S. Gronauer, M. Kissel, L. Sacchetto, M. Korte, and K. Diepold. Using simulation optimization to improve zero-shot policy transfer of quadrotors. In *International Conference on Intelligent Robots and Systems*, 2022.

S. Gu, T. Lillicrap, I. Sutskever, and S. Levine. Continuous deep q-learning with model-based acceleration. In *Proceedings of The 33rd International Conference on Machine Learning*, 2016.

O. Güler. *Foundations of Optimization*. Springer, 2010.

B. D. Haeffele and R. Vidal. Global optimality in neural network training. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

P. Henderson, J. Hu, J. Romoff, E. Brunskill, D. Jurafsky, and J. Pineau. Towards the systematic reporting of the energy and carbon footprints of machine learning. *Journal of Machine Learning Research*, 21(248), 2020.

M. Holzleitner, L. Gruber, J. Arjona-Medina, J. Brandstetter, and S. Hochreiter. *Convergence Proof for Actor-Critic Methods Applied to PPO and RUDDER*, pages 105–130. Springer Berlin Heidelberg, 2021.

R. Islam, P. Henderson, M. Gomrokchi, and D. Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *Computing Research Repository*, arXiv:1708.04133, 2017.

D. Jakubovitz, R. Giryes, and M. R. D. Rodrigues. *Generalization Error in Deep Learning*, pages 153–193. Springer International Publishing, 2019.

D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *Computing Research Repository*, arXiv:1806.10293, 2018.

K. Kawaguchi. Deep learning without poor local minima. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

H. Kimura. Reinforcement learning in multi-dimensional state-action space using random rectangular coarse coding and gibbs sampling. In *Society of Instrument and Control Engineers Annual Conference*, 2007.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *Computing Research Repository*, arXiv:1412.6980, 2014.

M. Kissel, M. Gottwald, and K. Diepold. Neural network training with safe regularization in the null space of batch activations. In *International Conference on Artificial Neural Networks*, 2020.

M. Laguna, A. Duarte, and R. Martí. The accelerated cross entropy method: An application to the max-cut problem. available online, last accessed 05.01.2023, 2006.

S. Lawrence, C. L. Giles, and A. C. Tsoi. Lessons in neural network training: Overfitting may be harder than expected. In *Proceedings of the* 14^{th} *National Conference on Artificial Intelligence and* 9^{th} *Conference on Innovative Applications of Artificial Intelligence*, 1997.

A. Lazaric, M. Restelli, and A. Bonarini. Reinforcement learning in continuous action spaces through sequential monte carlo methods. In *Advances in Neural Information Processing Systems*, 2007.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521, 2015.

M. Lee and C. W. Anderson. Convergent reinforcement learning control with neural networks and continuous action search. In *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, 2014.

S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39), 2016.

Y. Li, K. H. Johansson, J. Mårtensson, and D. P. Bertsekas. Data-driven rollout for deterministic optimal control. In 60^{th} *IEEE Conference on Decision and Control*, 2021.

T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *Computing Research Repository*, arXiv:1509.02971, 2015.

S. Lim, A. Joseph, L. Le, Y. Pan, and M. White. Actor-expert: A framework for using action-value methods in continuous action spaces. *Computing Research Repository*, arXiv:1810.09103, 2018.

L. J. Lin. *Reinforcement Learning for Robots Using Neural Networks*. PhD thesis, Carnegie Mellon University, 1993.

D. Lind and B. Marcus. *An Introduction to Symbolic Dynamics and Coding*. Cambridge University Press, 1995.

B. Liu, Q. Cai, Z. Yang, and Z. Wang. Neural proximal/trust region policy optimization attains globally optimal policy. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

H. R. Maei, C. Szepesvári, S. Bhatnagar, D. Precup, D. Silver, and R. S. Sutton. Convergent temporal-difference learning with arbitrary smooth function approximation. In *Advances in Neural Information Processing Systems*, volume 22, 2009.

H. Mania, A. Guy, and B. Recht. Simple random search provides a competitive approach to reinforcement learning. *Computing Research Repository*, arXiv:1803.07055, 2018.

G. Matheron, N. Perrin, and O. Sigaud. Understanding failures of deterministic actor-critic with continuous action spaces and sparse rewards. In *Artificial Neural Networks and Machine Learning*, 2020.

L. Metz, C. D. Freeman, S. S. Schoenholz, and T. Kachman. Gradients are not all you need. *Computing Research Repository*, arXiv:2111.05803, 2021.

J. D. R. Millán, D. Posenato, and E. Dedieu. Continuous-action q-learning. *Machine Learning*, 49 (2-3), 2002.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *Computing Research Repository*, arXiv:1312.5602, 2013.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Peterson, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518, 2015.

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the* 33^{rd} *International Conference on Machine Learning*, 2016.

T. M. Moerland, J. Broekens, and C. M. Jonker. Model-based reinforcement learning: A survey. *Computing Research Repository*, arXiv:2006.16712, 2020.

A. W. Moore. Efficient memory-based learning for robot control. Technical Report UCAM-CL-TR-209, University of Cambridge, Computer Laboratory, 1990.

A. Nedić and D. P. Bertsekas. Least squares policy evaluation algorithms with linear function approximation. *Discrete Event Dynamic Systems: Theory and Applications*, 13(1-2), 2003.

B. Neyshabur, S. Bhojanapalli, D. McAllester, and N. Srebro. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*, 2017.

Q. Nguyen and M. Hein. The loss surface of deep and wide neural networks. In *Proceedings of the* 34^{th} *International Conference on Machine Learning*, 2017.

B. D. Nichols. A comparison of action selection methods for implicit policy method reinforcement learning in continuous action-space. In *International Joint Conference on Neural Networks*, 2016.

B. D. Nichols and D. C. Dracopoulos. Application of newton's method to action selection in continuous state-and action-space reinforcement learning. In *Proceedings of the European Symposium on Artificial Neural Networks*, 2014.

R. Parr, C. Painter-Wakefield, L. Li, and M. Littman. Analyzing feature generation for value-function approximation. In *Proceedings of the* 24^{th} *International Conference on Machine Learning*, 2007.

R. Parr, L. Li, G. Taylor, C. Painter-Wakefield, and M. L. Littman. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *Proceedings of the* 25^{th} *International Conference on Machine Learning*, 2008.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

A. Ramaswamy, S. Bhatnagar, and D. E. Quevedo. Asynchronous stochastic approximations with asymptotically biased errors and deep multiagent learning. *IEEE Transactions on Automatic Control*, 66(9), 2021.

K. Rawlik, M. Toussaint, and S. Vijayakumar. On stochastic optimal control and reinforcement learning by approximate inference. In *Robotics: Science and Systems VIII (RSS 2012)*, 2012.

M. Riedmiller. Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method. In *Proceedings of the* 16^{th} *European Conference on Machine Learning*, 2005.

M. Ryu, Y. Chow, R. M. Anderson, C. Tjandraatmadja, and C. Boutilier. Caql: Continuous action q-learning. In *Proceedings of the* 8^{th} *International Conference on Learning Representations*, 2020. to appear.

E. Saleh and N. Jiang. Deterministic bellman residual minimization. In *Optimization Foundations for Reinforcement Learning Workshop*, 2019.

T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized experience replay. In *Proceedings of the* 4^{th} *International Conference on Learning Representations*, 2016.

J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *Proceedings of the* 32^{nd} *International Conference on Machine Learning*, volume 37, 2015.

J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the* 4^{th} *International Conference on Learning Representations*, 2016.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *Computing Research Repository*, arXiv:1707.06347, 2017.

J. V. Shah and C. S. Poon. Linear independence of internal representations in multilayer perceptrons. *IEEE Transactions on Neural Networks*, 10(1), 1999.

H. Shen. A differential topological view of challenges in learning with feedforward neural networks. *Computing Research Repository*, arXiv:1811.10304, 2018a.

H. Shen. Towards a mathematical understanding of the difficulty in learning with feedforward neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018b.

H. Shen and M. Gottwald. Demystification of flat minima and generalisability of deep neural networks. In *Workshop on Understanding and Improving Generalization in Deep Learning*, 2019.

O. Sigaud and F. Stulp. Policy search in continuous action domains: An overview. *Neural Networks*, 113, 2019.

D. Silver. Gradient temporal difference networks. In *Proceedings of the* 10^{th} *European Workshop on Reinforcement Learning*, volume 24, 2013.

D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the* 31^{st} *International Conference on Machine Learning*, volume 32, 2014.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529, 2016.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge. *Nature*, 550, 2017.

E. Strubell, A. Ganesh, and A. McCallum. Energy and policy considerations for deep learning in NLP. *Computing Research Repository*, arXiv:1906.02243, 2019.

R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. The MIT Press, 2^{nd} edition, 2020.

R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, volume 12, 1999.

R. S. Sutton, C. Szepesvári, and H. R. Maei. A convergent $O(n)$ algorithm for off-policy temporal-difference learning with linear function approximations. In *Advances in Neural Information Processing Systems*, volume 21, 2008.

R. S. Sutton, H. R. Maei, D. Precup, S. Bhatnagar, D. Silver, C. Szepesvári, and E. Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the* 26^{th} *International Conference on Machine Learning*, 2009.

G. Taylor and R. Parr. Kernelized value function approximation for reinforcement learning. In *Proceedings of the* 26^{th} *International Conference on Machine Learning*, 2009.

N. C. Thompson, K. Greenewald, K. Lee, and G. F. Manso. Deep learning's diminishing returns: The cost of improvement is becoming unsustainable. *IEEE Spectrum*, 58, 2021.

J. N. Tsitsiklis and B. van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5), 1997.

H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double Q-learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 1992.

J. Wen, S. Kumar, R. Gummadi, and D. Schuurmans. Characterizing the gap between actor-critic and policy gradient. In *Proceedings of the* 38^{th} *International Conference on Machine Learning*, 2021.

C. Xiao, B. Dai, J. Mei, O. A. Ramirez, R. Gummadi, C. Harris, and D. Schuurmans. Understanding and leveraging overparameterization in recursive value estimation. In *International Conference on Learning Representations*, 2022.

X. Xu, D. Hu, and X. Lu. Kernel-based least squares policy iteration for reinforcement learning. *IEEE Transactions on Neural Networks*, 18(4), 2007.

S. Yin, T. Luo, P. Liu, and Z. J. Xu. An experimental comparison between temporal difference and residual gradient with neural network approximation. *Computing Research Repository*, arXiv:2205.12770, 2022.

D. Yu and L. Deng. *Automatic Speech Recognition: A Deep Learning Approach*. Springer-Verlag, London, 2015.

Y. Yuan, Z. L. Yu, Z. Gu, Y. Yeboah, W. Wei, X. Deng, J. Li, and Y. Li. A novel multi-step q-learning method to improve data efficiency for deep reinforcement learning. *Knowledge-Based Systems*, 175, 2019.

C. Yun, S. Sra, and A. Jadbabaie. Global optimality conditions for deep neural networks. In *The* 6^{th} *International Conference on Learning Representations*, 2018.

C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. In *The* 5^{th} *International Conference on Learning Representations*, 2017.

K. Zhang, A. Koppel, H. Zhu, and T. Başar. Convergence and iteration complexity of policy gradient method for infinite-horizon reinforcement learning. In 58^{th} *IEEE Conference on Decision and Control*, 2019.

K. Zhang, A. Koppel, H. Zhu, and T. Başar. Global convergence of policy gradient methods to (almost) locally optimal policies. *SIAM Journal on Control and Optimization*, 58(6), 2020a.

S. Zhang, W. Boehmer, and S. Whiteson. Deep residual reinforcement learning. In *Proceedings of the* 19^{th} *International Conference on Autonomous Agents and Multi Agent Systems*, 2020b.