
ANALYSIS AND OPTIMISATION OF BELLMAN RESIDUAL ERRORS WITH NEURAL FUNCTION APPROXIMATION

A PREPRINT

Martin Gottwald *
martin.gottwald@tum.de

Sven Gronauer *
sven.gronauer@tum.de

Hao Shen †
shen@fortiss.org

Klaus Diepold *
kldi@tum.de

March 15, 2022

ABSTRACT

Recent development of Deep Reinforcement Learning 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, development of reliable and efficient numerical algorithms to minimise the Bellman Error is still of great scientific interest and practical demand. Such a challenge is partially due to the underlying optimisation problem being highly non-convex or using incomplete gradient information as done in Semi-Gradient algorithms. In this work, we analyse the Mean Squared Bellman Error from a smooth optimisation perspective and develop an efficient Approximate Newton's algorithm.

First, we 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. We 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. We confirm theoretical properties of this algorithm such as being locally quadratically convergent to a global minimum numerically. The second employs sampling and can be used in the continuous setting. We demonstrate feasibility and generalisation capabilities of the proposed algorithm empirically using continuous control problems and provide a numerical verification of our critical point analysis. We outline the difficulties of combining Semi-Gradient approaches with Hessian information. To benefit from second-order information complete derivatives of the Mean Squared Bellman Error must be considered during training.

Keywords Critical Point Analysis · Dynamic Programming · Gauss Newton Algorithm · Local Quadratic Convergence · Mean Squared Bellman Error · Residual Gradient

1 Introduction

Reinforcement Learning (RL), or more general *Dynamic Programming* (DP), is a general purpose framework to solve sequential decision making or optimal control problems. To handle problems with large or even continuous state spaces *Value Function Approximation* (VFA) is used as an important and effective instrument [Bertsekas, 2012, Sutton and Barto, 2020]. Such VFA methods can be categorised 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]. Despite their significant simplicity and convergence stability, the performance of LVFA methods depends heavily on construction and selection of linear features, which is a time consuming and hardly scalable process [Parr et al., 2007, Böhmer et al., 2013]. Therefore, recent research efforts have focused more on *Non-Linear Value Function Approximation* (NL-VFA) methods.

*Chair for Data Processing, Technical University of Munich, Arcisstr. 21, 80333 Munich, Germany

†fortiss, Forschungsinstitut des Freistaats Bayern, Guerickestr. 25, 80805 Munich, Germany

As a popular non-linear mechanism, kernel tricks have been successfully adopted to VFA, and 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]. Impressive successes of NNs in solving challenging problems in pattern recognition, computer vision, and 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, e.g. *Atari* games [Mnih et al., 2015] or the game *Go* [Silver et al., 2016, 2017]. Despite these advances, development of more efficient NN-VFA based algorithms is still of great demand for even more challenging applications. So far, these impressive successes are generally only possible, if a plethora of training samples is available.

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 e.g. [Riedmiller, 2005], and is recently referred to as *Semi-Gradient* (SG) algorithms [Sutton and Barto, 2020]. Reasons for choosing Semi-Gradients over Residual Gradients include inferior learning speed [Baird III, 1995], limitation with non-Markovian feature space [Sutton et al., 2008] and non-differentiable operators, e.g. the max-operator involved in Q-learning. Residual Gradients are favoured for their convergence guarantees and applicability of “classic” gradient based optimisation techniques.

In this work, we study the problem of minimising the *Mean Squared Bellman Error* (MSBE) with NN-VFA from the global analysis perspective. More specifically, we work in the framework of geometric optimisation [Absil et al., 2008]. We conduct a critical point analysis of the MSBE including its Hessian and also derive a proper approximation for it. We 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 and by ensuring certain properties of the optimisation objective. Furthermore, our analysis leads to an efficient and effective *Approximate Newton’s* (AN) algorithm. It is further investigated for a continuous state space setting 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. We outline how to overcome the convergence speed issues of Residual Gradients 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. Finally, we conduct several experiments to confirm the results of our critical point analysis numerically and also investigate generalisation capabilities of NN-VFA methods empirically.

The rest of this paper is arranged as follows. In the next section, we review the existing work regarding the construction and analysis of algorithms. In Section 3, we give an introduction to RL with VFA and provide the necessary notational conventions around NNs since a concise notation is required for our analysis in later sections. We conduct a critical point analysis of the MSBE when using NN-VFA in Section 4 and present our results separately for discrete and continuous state spaces in Sections 4.1 and 4.2, respectively. Section 5 is dedicated to our proposed Approximate Newton’s Residual Gradient algorithm and addresses details relevant for implementation. In Section 6, we evaluate performance and generalisation capabilities of the proposed method in several experiments. Finally, a conclusion is given in Section 7.

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* (GTD) algorithms to a non-linear smooth VFA manifold setting. The proposed approach requires projections onto some smooth manifold, which is practically infeasible because the 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 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. [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, 2018b,a, 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. In this work, we introduce those techniques to the RL 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. In this work, we investigate Residual Gradient algorithms in deterministic problems for Policy Evaluation instead of aiming directly at the optimal value function. This allows us to use a Policy Iteration scheme as done in [Gottwald et al., 2018].

Another work [Cai et al., 2019], which is also strongly related to ours, 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 proposed in [Fu et al., 2019], the authors arrive at the conclusion that larger NL-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. In the present work, we confirm the role of over-parametrisation when using Residual Gradient algorithms and provide further insights from the global analysis perspective.

3 Notation and Technical Preliminaries

This section contains various definitions and provides a concise foundation for Section 4. First, we outline the Reinforcement Learning setting in general. Second, we introduce the usage of function approximation architectures and formulate the optimisation problem we want to investigate. Lastly, we define the function class and its components used for the NL-VFA method we are going to analyse.

3.1 Reinforcement Learning

As common approach, we model the decision making problem as a *Markov Decision Process* (MDP) by defining the tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$. As state space \mathcal{S} we consider both finite countable sets of K discrete elements $\mathcal{S} = \{1, 2, \dots, K\}$ as well as compact subsets of finite dimensional Euclidean vector spaces $\mathcal{S} \subset \mathbb{R}^K$. Depending on the state space, we denote with slight abuse of notation by $K := |\mathcal{S}|$ either the cardinality of the finite set or by $K := \dim(\mathcal{S})$ the dimension of the subspace. The action space \mathcal{A} is always a finite set of discrete actions an agent can choose from. The conditional transition probabilities $P: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ are either available directly in the form of analytic models or can be collected by using simulators and performing roll-outs. For discrete spaces, they define the probability $P(s'|s, a)$ for transiting from state s to s' when executing action a . In continuous state spaces, P takes the role of a probability density function which must be integrated over. A scalar *reward* function $r: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [-M, M]$ 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 of an agent is to learn a *policy* π , which maximises the expected discounted reward. It is sufficient to consider deterministic policies of the form $\pi: \mathcal{S} \rightarrow \mathcal{A}$, as the space of *history independent Markov policies* can be proven³ to contain an optimal policy.

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

$$V_\pi: \mathcal{S} \rightarrow \mathbb{R}, \quad s \mapsto \lim_{T \rightarrow \infty} \mathbb{E}_{s_1, s_2, \dots, s_T} \left[\sum_{t=0}^T \gamma^t r(s_t, \pi(s_t), s_{t+1}) \mid s_0 = s \right]. \quad (1)$$

³e.g. chapter one and two in [Bertsekas, 2012], applies also to other statements here

The computation of expectations depends on the type of the state space. For discrete sets, the expression takes the form of a weighted sum. In continuous spaces, we need to compute an integral over the successor states. It is well known that for a given policy π the value function V_π satisfies the *Bellman equation*, i.e., we can write for all states $s \in \mathcal{S}$

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_{s'} \left[r(s, \pi(s), s') + \gamma V_\pi(s') \right] \\ &= \sum_{s'} P(s, \pi(s), s') \left(r(s, \pi(s), s') + \gamma V_\pi(s') \right), \end{aligned} \quad (2)$$

where s' is the successor state of s obtained by executing action $a = \pi(s)$. The second equality in Equation (2) is only available if transition probabilities are known and exist due to discrete state and action spaces. 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) induces the *Bellman Operator* under the policy π and is denoted by $T_\pi: \mathcal{V} \rightarrow \mathcal{V}$. It can be shown that T_π is a contraction mapping with modulus γ and, consequently, that the value function V_π is its unique fixed point. Hence, we can write

$$V_\pi(s) = (T_\pi V_\pi)(s) \quad \forall s \in \mathcal{S}. \quad (3)$$

Finally, given an arbitrary $V \in \mathcal{V}$, the difference of both sides in Equation (3) is known as the *one-step Temporal Difference error*

$$\delta(s) := V(s) - (T_\pi V)(s) \quad \forall s \in \mathcal{S}. \quad (4)$$

The term *Temporal Difference* (TD) emphasizes the occurrence of the current state in $V(s)$ and the successor state s' in $(T_\pi V)(s)$. In this context one also calls the application of the Bellman operator *TD target*. The TD error is non-zero for all but the correct value function. Thus, we can use δ to convert the fixed point iteration into a root finding problem. By combining the squared TD-error for all states we obtain a performance objective that can be minimised. We will use this objective together with function approximation architectures to conduct our critical point analysis in a later section.

To avoid extensive computation when working with policies and state-only value functions, it is possible to define similar to Equation (1) so-called *Q-factors* as

$$Q_\pi: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}, \quad s, a \mapsto \lim_{T \rightarrow \infty} \mathbb{E}_{s_1, s_2, \dots, s_T} \left[\sum_{t=0}^{T-1} \gamma^t r(s_t, \pi(s_t), s_{t+1}) \mid s_0 = s, a_0 = a \right] \quad (5)$$

and the corresponding *Bellman equation in Q*

$$Q_\pi(s, a) = \mathbb{E}_{s'} \left[r(s, a, s') + \gamma Q_\pi(s', \pi(s')) \right] = (T_\pi Q_\pi)(s, a) \quad (6)$$

for all $(s, a) \in \mathcal{S} \times \mathcal{A}$. While all algorithms work with *Q-factors* as they do for state-only value functions, it is now possible to define a *greedily induced policy* (GIP) compactly as $\pi': \mathcal{S} \rightarrow \mathcal{A}$ with

$$\pi'(s) \in \operatorname{argmax}_{a \in \mathcal{A}} Q_\pi(s, a). \quad (7)$$

The new GIP π' satisfies $\pi' \geq \pi$ in the sense that $Q_{\pi'}(s, a) \geq Q_\pi(s, a)$ for all states and actions with the equality being true for optimal policies.

3.2 Reinforcement Learning with Function Approximation

If a state space is finite but too large to be stored in memory, or even has to be treated as infinite due to its size, an exact representation of the value function $V_\pi(s)$ is practically impossible. This phenomenon is often referred to as the *curse of dimensionality*. 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 the current policy π in any computer system.

Let $F: \mathcal{S} \rightarrow \mathbb{R}$ be an arbitrary value function approximation and let us now denote by \mathcal{V} the space of approximations considered. If we further restrict the decision making problem to be *ergodic*, there exists a *steady state distribution* $\xi_i \in (0, 1)$ for each⁴ state s_i , which allows us to create a quality assessment for the approximation when combined with the aforementioned objective Equation (4). The combination of a ξ -weighted norm and the TD-error yields the *Mean Squared Bellman Error*

$$\text{MSBE}(F) = \frac{1}{2} \left\| F - (T_\pi F) \right\|_\xi^2, \quad (8)$$

⁴for continuous spaces ξ is also a probability density function and must be used with integrals

which accepts a value function approximation $F \in \mathcal{V}$ and returns a scalar value. The factor $1/2$ is included for convenience. As the MSBE contains a norm over a function space, the realisation of $\|\cdot\|_\xi$ comes in two different ways, depending on the type of state space. In discrete spaces we can enumerate all states and write down directly

$$\text{MSBE}(F) = \frac{1}{2} \left\| F - (\mathbb{T}_\pi F) \right\|_\xi^2 = \frac{1}{2} \sum_{s \in \mathcal{S}} \xi(s) \left(F(s) - (\mathbb{T}_\pi F)(s) \right)^2. \quad (9)$$

Since the steady state distribution is a probability, we can treat the summation also as expectation and approximate it by Monte Carlo sampling. We have

$$\text{MSBE}(F) = \frac{1}{2} \mathbb{E}_{s \in \mathcal{S}} \left[\left(F(s) - (\mathbb{T}_\pi F)(s) \right)^2 \right] \approx \frac{1}{2N} \sum_{i=1}^N \left(F(s_i) - (\mathbb{T}_\pi F)(s_i) \right)^2, \quad (10)$$

where samples s_i are drawn according to the environment. In continuous spaces, the ξ -norm is typically based on integrals

$$\text{MSBE}(F) = \frac{1}{2} \left\| F - (\mathbb{T}_\pi F) \right\|_\xi^2 = \frac{1}{2} \int_{\mathcal{S}} \xi(s) \left(F(s) - (\mathbb{T}_\pi F)(s) \right)^2 ds. \quad (11)$$

Here, ξ takes the role of a normalised probability density function such that we can make use of Monte Carlo Integration. It allows us to write integrals as summations over samples

$$\text{MSBE}(F) = \frac{1}{2} \int_{\mathcal{S}} \xi(s) \left(F(s) - (\mathbb{T}_\pi F)(s) \right)^2 ds \approx \frac{1}{2N} \sum_{i=1}^N \left(F(s_i) - (\mathbb{T}_\pi F)(s_i) \right)^2. \quad (12)$$

From Equations (10) and (12) 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 $(F(s) - (\mathbb{T}_\pi F)(s))$ in Equation (11) is square integrable, i.e., an element of L^2 . This is not a problem for the definition of V_π as the infinite discounted sum in Equation (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 \mathcal{S} . However, this stands in conflict with the assumptions required for RL and also can affect in turn the square integrability of V_π . We leave these theoretical 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 input to F .

Reinforcement Learning with Function Approximation now manifests itself in the optimisation problem

$$F_\pi \in \underset{F \in \mathcal{V}}{\text{argmin}} \text{MSBE}(F), \quad (13)$$

where F_π is an optimal approximation to V_π in terms of minimising the MSBE. In general we have $F_\pi \neq V_\pi$ for at least some states $s \in \mathcal{S}$ and only aim to be close enough to V_π . By the choice of ξ as weighting for the norm, we also maintain for linear function approximation architectures the contraction properties of \mathbb{T}_π when combining it with a linear projection onto that function space. The accuracy of the solution F_π as defined in Equation (13) is known to be bounded by

$$\|F_\pi - V_\pi\|_\xi \leq \frac{1 + \gamma}{1 - \gamma} \inf_{F \in \mathcal{V}} \|F - V_\pi\|_\xi. \quad (14)$$

Obviously, the challenge is to find a function space \mathcal{V} which we can optimise over and which still contains a good approximation for the desired function V_π .

3.3 Multi Layer Perceptrons

To approximate value functions, we deploy in this paper the classic feed forward fully connected neural network, a.k.a. *Multi-Layer Perceptron* (MLP). We summarise in the following 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 us 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 $l = 1, \dots, L$. By $l = 0$ we refer to the input layer with n_0 units. The value of n_0 depends on the state space and

its type. For discrete spaces, we have $n_0 = 1$ such that a single state can be processed directly as natural number by the network. In a continuous setting, we use $n_0 = K$ units matching the K -dimensional state vectors. We always restrict the number of nodes in the output layer $l = L$ to $n_L = 1$.

Let $\sigma: \mathbb{R} \rightarrow \mathbb{R}$ be a unit activation function and denote by $\dot{\sigma}: \mathbb{R} \rightarrow \mathbb{R}$ 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*⁵. In this work, we 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 in the last layer. The latter two are used in this work.

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

$$\Lambda_{l,k}(w_{l,k}, b_{l,k}, \phi_{l-1}) := \sigma(w_{l,k}^\top \phi_{l-1} - b_{l,k}), \quad (15)$$

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, we can define the l -th *layer mapping* by stacking all *unit mappings* of layer l as

$$\begin{aligned} \Lambda_l(W_l, b_l, \phi_{l-1}) &:= [\Lambda_{l,1}(w_{l,1}, b_{l,1}, \phi_{l-1}) \quad \dots \quad \Lambda_{l,n_l}(w_{l,n_l}, b_{l,n_l}, \phi_{l-1})]^\top \\ &= \sigma(W_l^\top \phi_{l-1} + b_l), \end{aligned} \quad (16)$$

with $W_l := [w_{l,1}, \dots, w_{l,n_l}] \in \mathbb{R}^{n_{l-1} \times n_l}$ being the l -th parameter matrix and $b_l := [b_{l,1}, \dots, b_{l,n_l}] \in \mathbb{R}^{n_{l-1}}$ the l -th bias vector. 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, we can write Equation (16) equivalently as

$$\Lambda_l(W_l, \phi_{l-1}) := \sigma\left(W_l^\top \cdot \begin{bmatrix} \phi_{l-1} \\ 1 \end{bmatrix}\right) \quad (17)$$

by using a larger parameter matrix $W_l \in \mathbb{R}^{(n_{l-1}+1) \times n_l}$. Next, let us define the overall function represented by the MLP. First, denote by $\phi_0 \in \mathbb{R}^{n_0}$ the network input. Then, 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 employs the identity map as activation function and is thus only an affine mapping. Finally, by denoting the set of all parameter matrices in the MLP by $\mathcal{W} := \mathbb{R}^{(n_0+1) \times n_1} \times \dots \times \mathbb{R}^{(n_{L-1}+1) \times 1}$, we can compose all layer-wise mappings to define for a set of parameters $\mathbf{W} \in \mathcal{W}$ the overall *network mapping* as

$$f: \mathcal{W} \times \mathbb{R}^{n_0} \rightarrow \mathbb{R}, \quad (\mathbf{W}, \phi_0) \mapsto \Lambda_L(W_L, \cdot) \circ \dots \circ \Lambda_1(W_1, \phi_0), \quad (18)$$

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

$$N_{net} = \sum_{l=1}^L (n_{l-1} + 1) \cdot n_l. \quad (19)$$

With this construction, we can define the set of parametrised functions belonging to an MLP by writing

$$\mathcal{F} := \{f(\mathbf{W}, \cdot): \mathbb{R}^{n_0} \rightarrow \mathbb{R} \mid \mathbf{W} \in \mathcal{W}\}. \quad (20)$$

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

3.4 Optimisation Approaches

For solving the optimisation problem in Equation (13), i.e., obtaining the approximated value function with smallest MSBE, two fundamental approaches using gradient information exist:

- *Direct Algorithms* [Baird III, 1995], a.k.a. *Semi-Gradient* [Sutton and Barto, 2020];
- *Residual Gradient* algorithms [Baird III, 1995].

⁵also abbreviated as *Bent-Id*

Both approaches are descent algorithms. The difference between them resides in the choice of descent directions. For Residual Gradient algorithms, the complete gradient is calculated. In Semi-Gradient algorithms, one ignores the dependence on the parameters through the value function inside the Bellman Operator. We refer to [Zhang et al., 2020] and references therein for a comparison of Semi-Gradient and Residual Gradient algorithms.

Next, we proceed with the analysis of critical points of Equation (8) for discrete and continuous spaces. We only investigate Residual Gradient algorithms as there is no insight to be gained from a critical point analysis, when we use a direction for descending, which is not always pointing in the same half space as the gradient.

4 A Critical Point Analysis of the MSBE

In the following, we present our theoretical investigation of Residual Gradient algorithms by analysing the critical points of the MSBE. Thereby, we follow the work in [Shen, 2018b] and translate those insights to the RL domain. First, we investigate discrete state spaces and derive conditions under which learning the exact value function works reliably. Second, we extend our analysis to continuous spaces by changing to a sampling based approximation of the MSBE and adapt our conditions accordingly. We characterise the importance of over-parametrisation, give design principles for MLPs and unveil a connection to other state of the art algorithms.

4.1 For Discrete State Spaces

In discrete problems, we have two choices to approach a critical point analysis. Either we seek for an approximation architecture that is exact for each of the K states or we are interested in an architecture, which is exact for only $N \ll K$ sampled but unique states. While the second case simplifies the analysis and relaxes restrictions as shown later for continuous spaces, this also means that we have to address generalisation to states outside of the sampled ones. As the sampling case with discrete states would match almost completely our approach for continuous state spaces, we present here only the exact learning assumption. This implies that we are interested in conditions for an MLP that would allow for a perfect solution to any state.

We consider MLPs of the form $\mathcal{F}(1, n_1, \dots, n_{L-1}, 1)$, i.e., fully connected feed forward networks with a one dimensional input and output and arbitrary depth or width. With discrete and finite spaces we 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 , where K is the number of states. As the simplest approach, we encode the discrete states with natural numbers for the single input unit and, thus, can treat $F(\mathbf{W}) := [f(\mathbf{W}, 1) \dots f(\mathbf{W}, K)]^\top \in \mathbb{R}^K$ as the approximated value function for all states. Next, we define the *Bellman Residual* vector for a policy π in matrix form as

$$\Delta_\pi(\mathbf{W}) = F(\mathbf{W}) - T_\pi F(\mathbf{W}) = F(\mathbf{W}) - P_\pi \left(R_\pi + \gamma F(\mathbf{W}) \right), \quad (21)$$

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 . We use a capital Δ instead of δ to emphasize that we consider all transitions in here instead of just a single one as in Equation (4). Using a diagonal matrix $\Xi := \text{diag}(\xi_1, \dots, \xi_K)$ consisting of the steady state distributions ξ_i for all states, we can rewrite the norm in Equation (9) as the *Neural Mean Squared Bellman Error* (NMSBE) function

$$\mathcal{J}(\mathbf{W}) := \frac{1}{2} \Delta_\pi(\mathbf{W})^\top \Xi \Delta_\pi(\mathbf{W}). \quad (22)$$

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 $\mathbf{W} \rightarrow \infty$ we do not necessarily have $\mathcal{J}(\mathbf{W}) \rightarrow \infty$. 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 our 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 $V_\pi: \mathcal{S} \rightarrow \mathbb{R}$ be the exact value function under policy π . There exists at least one MLP architecture \mathcal{F} as defined in Equation (20) together with a set of parameters $\mathbf{W}^* \in \mathcal{W}$ such that the output of F and V_π coincide, i.e., we have*

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

To conduct a critical point analysis of the NMSBE function we first compute the derivatives. By the linearity of the Bellman Operator T_π , we obtain the directional derivative of \mathcal{J} at the point $\mathbf{W} \in \mathcal{W}$ in direction $\mathbf{H} \in \mathcal{W}$ as

$$D \mathcal{J}(\mathbf{W})[\mathbf{H}] = \Delta_\pi(\mathbf{W})^\top \Xi (I_K - \gamma P_\pi) D F(\mathbf{W})[\mathbf{H}], \quad (23)$$

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

$$DF(\mathbf{W})[\mathbf{H}] = \underbrace{\left[\text{vec}(\nabla_{\mathbf{W}} f(\mathbf{W}, 1)) \dots \text{vec}(\nabla_{\mathbf{W}} f(\mathbf{W}, K)) \right]}_{=: G(\mathbf{W}) \in \mathbb{R}^{K \times N_{net}}} \text{vec}(\mathbf{H}), \quad (24)$$

where $\text{vec}(\nabla_{\mathbf{W}} f(\mathbf{W}, s)) \in \mathbb{R}^{N_{net}}$ is the gradient of f with respect to the parameters evaluated at \mathbf{W} for state s under the Euclidean norm. The operation $\text{vec}(\cdot)$ 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, we can characterise critical points of the NMSBE function \mathcal{J} from Equation (22) by setting its gradient to zero, i.e., $\nabla_{\mathbf{W}} \mathcal{J}(\mathbf{W}) = 0$. Combining the results from Equations (23) and (24) together with *Riesz' Representation Theorem* yields the critical point condition

$$\nabla_{\mathbf{W}} \mathcal{J}(\mathbf{W}) := G(\mathbf{W})^T (I_K - \gamma P_{\pi})^T \Xi \Delta_{\pi}(\mathbf{W}) \stackrel{!}{=} 0, \quad (25)$$

which is the counterpart⁶ to Equation (19) in [Shen, 2018b] for the Reinforcement Learning setting with exact learning. We 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 (24) is equal to K for all $\mathbf{W} \in \mathcal{W}$, then any extremum $\mathbf{W}^* \in \mathcal{W}$ 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.*

Proof. Since the underlying state transitions under policy π are required to be Markovian and ergodic, both terms Ξ and $(I_K - \gamma P_{\pi})$ have full rank. Consequently, the expression $\Xi(I_K - \gamma P_{\pi})G(\mathbf{W}^*)^T$ also has rank K , if we claim that $G(\mathbf{W}^*)$ has full rank. Hence, there is only the trivial solution left for the linear system in Equation (25), 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. \square

To make use of Proposition 1, one needs to investigate, under what conditions the matrix $G(\mathbf{W})$ has full column rank. The immediate risk of being exactly zero can be eliminated by choosing proper activation functions without zero derivatives for finite inputs. To see this we have to look at the structure of $G(\mathbf{W})$, which takes the form

$$G(\mathbf{W}) = \begin{bmatrix} \Psi_1^T \left(I_{n_1} \otimes \phi_0^{(1)T} \right) & \dots & \Psi_L^T \left(I_{n_L} \otimes \phi_{L-1}^{(1)T} \right) \\ \vdots & \ddots & \vdots \\ \Psi_1^T \left(I_{n_1} \otimes \phi_0^{(K)T} \right) & \dots & \Psi_L^T \left(I_{n_L} \otimes \phi_{L-1}^{(K)T} \right) \end{bmatrix}, \quad (26)$$

where the matrices $\Psi_l \in \mathbb{R}^{n_l \times n_L}$ and Kronecker products result from the layer wise definition of an MLP. The Ψ_l are products of diagonal matrices containing the derivative of σ and the parameter matrices of various layers, hence a proper choice for σ prevents $G(\mathbf{W})$ from becoming zero. A detailed construction is available in the appendix.

Maintaining the rank of $G(\mathbf{W})$ means to choose MLPs in such a way that one minimises the risk for rows of the matrix to lie in a shared subspace. As in the original multidimensional regression setting, the overall block structure of $G(\mathbf{W})$ remains for RL applications. However, we have in the RL setting the 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 the same amount of sampled inputs, there are less possibilities to have linear dependent rows. Although there are no theoretical guarantees, most matrices have full rank in practical applications, where noise or sampling is present.

Unfortunately, the requirement of using over-parametrised MLPs, i.e.,

$$N_{net} \geq K, \quad (27)$$

⁶In this paper we used G in place of P to avoid confusion with the transition probability matrix

prevents a direct application of the theory. If we need as many adjustable parameters in an MLP as there are unique states, we could use a tabular representation in the first place and avoid dealing with non-linear function approximation. Note that the strong condition in Equation (27) stems from the assumption of being exact for all states. Typically, 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 $\bar{K} < K$, since several states could receive an identical feature vector.

If the NMSBE is changed to be an approximation based on sampling, as done for continuous spaces in Section 4.2, we can reduce the number of MLP parameters from K down to $N \ll K$ samples. But as an immediate consequence, generalisation becomes an issue as a result of the MLP being only accurate by design for finitely many points 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 this 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, we aim at *Approximated Newton's* (AN) algorithms. A closer look reveals that the differential map as shown in Equation (23) is a candidate for the *Gauss Newton* (GN) approximation as in the non-linear regression setting. Indeed, for the second directional derivative of \mathcal{J} at \mathbf{W} with two directions $\mathbf{H}_1, \mathbf{H}_2 \in \mathcal{W}$ we have

$$\begin{aligned} \mathrm{D}^2 \mathcal{J}(\mathbf{W})[\mathbf{H}_1, \mathbf{H}_2] &= \Delta_\pi(\mathbf{W})^\top \Xi (I_K - \gamma P_\pi) \mathrm{D}^2 F(\mathbf{W})[\mathbf{H}_1, \mathbf{H}_2] \\ &\quad + \mathrm{D} F(\mathbf{W})[\mathbf{H}_1]^\top (I_K - \gamma P_\pi)^\top \Xi (I_K - \gamma P_\pi) \mathrm{D} F(\mathbf{W})[\mathbf{H}_2], \end{aligned} \quad (28)$$

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

$$\mathrm{D}^2 \mathcal{J}(\mathbf{W}^*)[\mathbf{H}_1, \mathbf{H}_2] = \mathrm{vec}(\mathbf{H}_1)^\top \underbrace{G(\mathbf{W}^*)^\top (I_K - \gamma P_\pi)^\top \Xi (I_K - \gamma P_\pi) G(\mathbf{W}^*)}_{\mathbf{H}_{\mathbf{W}} \mathcal{J}(\mathbf{W}^*) \in \mathbb{R}^{N_{net} \times N_{net}}} \mathrm{vec}(\mathbf{H}_2). \quad (29)$$

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. Our characterisation of critical points reveals this possibility for approximation as a side benefit. Using naively the product of the model'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, we need to characterise the Hessian $\mathbf{H}_{\mathbf{W}} \mathcal{J}(\mathbf{W}^*)$ 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*

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

if the MLP satisfies Equation (27).

Proof. Positive semi-definiteness of $\mathbf{H}_{\mathbf{W}} \mathcal{J}(\mathbf{W}^*)$ follows from its symmetric definition. As before Ξ and $(I_K - \gamma P_\pi)$ have full rank. The rank of $G(\mathbf{W}^*)$ is at most K . For $\mathbf{H}_{\mathbf{W}} \mathcal{J}(\mathbf{W}^*)$ being the product of these matrices we get the upper bound on its rank. \square

It is interesting to see that the rank condition from Equation (27) also allows the Hessian to become positive definite if the matrix $G(\mathbf{W})$ has full column rank. 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. We confirm the proposed approximation for the Hessian in discrete state spaces with exact learning numerically in Section 5.3.

4.2 For Continuous State Spaces

In high dimensional continuous state spaces, say $K > 6$, 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 we are forced to work directly with the continuous space, which causes MLPs to be of the form $\mathcal{F}(K, \dots, 1)$ to accept K 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 (22) is not available here. We must work with a finite number of samples $N \in \mathbb{N}$ to approximate the loss as done in Equation (12).

Another limitation for Residual Gradient algorithms is the so-called *Doubling 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 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 RL testbeds are deterministic or possess only a small amount of noise. Thus, for our analysis, we follow the same strategy and restrict ourselves to deterministic MDPs and analyse the algorithm in its purest form. The one-step TD-error from Equation (4) now simplifies for the i -th sample to

$$\delta(s_i, s'_i) := V(s_i) - r(s_i, \pi(s_i), s'_i) - \gamma V(s'_i), \quad (30)$$

where s'_i is the successor of s_i when executing $\pi(s_i)$. As before, we collect the evaluation of the MLP for all N sampled states s_i as vector and denote it by $F(\mathbf{W}) := [f(\mathbf{W}, s_1) \dots f(\mathbf{W}, s_N)]^\top \in \mathbb{R}^N$. Next, we rewrite the loss of Equation (12) accordingly and obtain the sampled NMSBE for continuous state spaces as

$$\begin{aligned} \tilde{\mathcal{J}}(\mathbf{W}) &:= \frac{1}{2N} \sum_{i=1}^N \left(\underbrace{f(\mathbf{W}, s_i) - r(s_i, \pi(s_i), s'_i) - \gamma f(\mathbf{W}, s'_i)}_{\delta(s_i, s'_i)} \right)^2 \\ &= \frac{1}{2N} \begin{bmatrix} \delta(s_1, s'_1) & \dots & \delta(s_N, s'_N) \end{bmatrix} \cdot \begin{bmatrix} \delta(s_1, s'_1) \\ \vdots \\ \delta(s_N, s'_N) \end{bmatrix} = \frac{1}{2N} \Delta_\pi(\mathbf{W})^\top \Delta_\pi(\mathbf{W}), \end{aligned} \quad (31)$$

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

$$\Delta_\pi(\mathbf{W}) = F(\mathbf{W}) - R_\pi - \gamma F'(\mathbf{W}) \quad (32)$$

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

Definition 1 (Finite exact approximator). *Let $V_\pi: \mathcal{S} \rightarrow \mathbb{R}$ be the value function under policy π . Given N sample states $s_i \in \mathcal{S}$ we 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 $\mathbf{W} \in \mathcal{W}$, a finite exact approximator of V_π based on the N sample states.

As in the discrete setting, we 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 approximator). *Let $V_\pi: \mathcal{S} \rightarrow \mathbb{R}$ 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 (20) together with a set of parameters $\mathbf{W}^* \in \mathcal{W}$ such that the MLP $f(\mathbf{W}^*, \cdot) \in \mathcal{F}$ is a finite exact approximator of V_π according to Definition 1.*

A key challenge of Definition 1 and Assumption 2 resides in the sampling based approximation of the NMSBE. With only finitely many samples, it can happen that the connection between states and their successors is lost in some parts of the state space. Whether this happens solely depends on the MDP and the distance of samples. Hence, the NMSBE might possess undesired degrees of freedom and Equation (30) no longer applies to all sample states. Unfortunately, Assumption 2 ensures, that we can always find an MLP f with parameters \mathbf{W}^* to make $\Delta_\pi(\mathbf{W}^*)$ exactly zero. However, in such a scenario, this MLP will not be a finite exact approximator according to Definition 1.

Proposition 3 (The NMSBE includes bad solutions). *The NMSBE as shown in Equation (31) contains minima $\mathbf{W}^* \in \mathcal{W}$, for which the MLP f is not a finite exact approximator according to Definition 1.*

Proof. Denote by $\mathcal{W}^* := \{\mathbf{W} \in \mathcal{W} \mid \Delta_\pi(\mathbf{W}) = 0\}$ the set of all MLP with minimal NMSBE. It coincides with critical points as we see later. By $\mathcal{W}' := \{\mathbf{W} \in \mathcal{W} \mid f(\mathbf{W}, s_i) = V_\pi(s_i) \forall i\}$ we denote the set of all finite exact approximators. Both sets are non-empty due to Assumption 1. In general we have $\mathcal{W}' \subset \mathcal{W}^*$ such that $\mathcal{W}^* \setminus \mathcal{W}' \neq \{\}$, because if for a single sample state the connection to its successors is not present, then the MLP can approximate any value for that state without affecting the NMSBE. Thus, there exist critical points of the NMSBE for which at some states the MLP is arbitrarily far away from V_π . \square

Whether the mismatch of both sets becomes problematic, depends on generalisation capabilities of MLPs. Thus, these capabilities are crucial and we investigate them empirically as part of the experiments.

Our critical point analysis follows the same steps as in Section 4.1. First, we need the differential map of Equation (31). It takes the form

$$\begin{aligned} D\tilde{\mathcal{J}}(\mathbf{W})[\mathbf{H}] &= \frac{1}{N}\Delta_\pi(\mathbf{W})^\top \left(DF(\mathbf{W})[\mathbf{H}] - \gamma DF'(\mathbf{W})[\mathbf{H}] \right) \\ &= \frac{1}{N}\Delta_\pi(\mathbf{W})^\top \left(G(\mathbf{W}) - \gamma G'(\mathbf{W}) \right) \text{vec}(\mathbf{H}), \end{aligned} \quad (33)$$

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

$$\nabla_{\mathbf{W}}\tilde{\mathcal{J}}(\mathbf{W}) := \frac{1}{N} \left(\underbrace{G(\mathbf{W}) - \gamma G'(\mathbf{W})}_{=: \tilde{G}(\mathbf{W}) \in \mathbb{R}^{N \times N_{net}}} \right)^\top \Delta_\pi(\mathbf{W}) \stackrel{!}{=} 0. \quad (34)$$

Apparently, the critical point condition for the continuous setting based on N unique samples in Equation (34) takes a similar form as that of the discrete setting in Equation (25). But since we no longer investigate the exact learning scenario, we 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 $\tilde{G}(\mathbf{W})$ as defined in Equation (34) is equal to N for all $\mathbf{W} \in \mathcal{W}$, then any extremum $\mathbf{W}^* \in \mathcal{W}$ achieves zero NMSBE. Furthermore, the NMSBE function $\tilde{\mathcal{J}}$ of Equation (31) is free of suboptimal local minima.*

Proof. As before, Equation (34) defines a linear equation system in the sampled Bellman Residual vector Δ_π . If we claim that $\tilde{G}(\mathbf{W})$ has full rank, there is only the trivial solution left for $\Delta_\pi(\mathbf{W})$. By Assumption 2 we know 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. \square

We now address the requirements for $\tilde{G}(\mathbf{W})$ since this matrix forms the backbone of Proposition 4. As the first requirement, $\tilde{G}(\mathbf{W})$ has to be non-zero to define a proper equation system and enforce a trivial solution in terms of $\Delta_\pi(\mathbf{W})$ in Equation (34). 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 $\|s - s'\| \rightarrow 0$, which happens for example whenever s is getting close to a fixed point of the system dynamics, 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 $\tilde{G}(\mathbf{W})$ 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, the only known inequality is

$$\text{rank}(A + B) < \text{rank}(A) + \text{rank}(B),$$

which implies that we still have to increase the rank of $G(\mathbf{W})$ and $G'(\mathbf{W})$ individually to push the upper bound for the rank of $\tilde{G}(\mathbf{W})$ high enough to allow for full rank. This leads us to similar design principles for the MLP as in the discrete setting. Hence, the more complex $\tilde{G}(\mathbf{W})$ still complies to considerations of the discrete setting regarding linear dependent rows. Of course these properties and requirements come without any guarantees, i.e., we expect that carefully constructed examples (yet almost pathological) exist, where a drop of the rank of $\tilde{G}(\mathbf{W})$ happens. But in practice, where numerical errors and sampled quantities are present, we do not consider this to become a problem. When taking a closer look at $\tilde{G}(\mathbf{W})$ itself, we find that the overall block structure of $G(\mathbf{W})$ as shown in Equation (26) remains. We have

$$\tilde{G}(\mathbf{W}) = \begin{bmatrix} \Psi_1^\top \left(I_{n_1} \otimes \phi_0^{(1)} \right)^\top - \gamma \Psi_1'^\top \left(I_{n_1} \otimes \phi_0'^{(1)} \right)^\top & \dots & \Psi_L^\top \left(I_{n_L} \otimes \phi_{L-1}^{(1)} \right)^\top - \gamma \Psi_L'^\top \left(I_{n_L} \otimes \phi_{L-1}'^{(1)} \right)^\top \\ \vdots & \ddots & \vdots \\ \Psi_1^\top \left(I_{n_1} \otimes \phi_0^{(N)} \right)^\top - \gamma \Psi_1'^\top \left(I_{n_1} \otimes \phi_0'^{(N)} \right)^\top & \dots & \Psi_L^\top \left(I_{n_L} \otimes \phi_{L-1}^{(N)} \right)^\top - \gamma \Psi_L'^\top \left(I_{n_L} \otimes \phi_{L-1}'^{(N)} \right)^\top \end{bmatrix}. \quad (35)$$

The vectors $\phi_l \in \mathbb{R}^{n_l}$ result from the evaluation of all layers for state s and ϕ'_l from using s' . Similarly, the matrices Ψ and Ψ' use s and s' for their computation. We provide slightly more detailed construction steps in the appendix. Unfortunately, we see from Equation (35) that no additional statements 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. [Zhang et al., 2020, Baird III, 1995]. We can identify two remedies here. The first is already visible in Equation (34) or Equation (35), where the contribution of the successor $G'(\mathbf{W})$ 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 the 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 $\tilde{G}(\mathbf{W})$. 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]. 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. The second remedy is related to vanishing gradients of \tilde{J} if s and s' are close by. Here, the natural solution is to use second-order gradient descent, which takes the curvature into account to define a descent direction. Hence, we propose to employ again a *Gauss Newton Residual Gradient Algorithm*. Other approaches to overcome curvature issues such as 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 we have decided to use second-order optimisation.

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

$$\begin{aligned} \mathrm{D}^2 \tilde{J}(\mathbf{W})[\mathbf{H}_1, \mathbf{H}_2] &= \frac{1}{N} \Delta_\pi(\mathbf{W})^\top \left(\mathrm{D}^2 F(\mathbf{W})[\mathbf{H}_1, \mathbf{H}_2] - \gamma \mathrm{D}^2 F'(\mathbf{W})[\mathbf{H}_1, \mathbf{H}_2] \right) \\ &\quad + \frac{1}{N} \mathrm{D} \Delta_\pi(\mathbf{W})[\mathbf{H}_1]^\top \left(\mathrm{D} F(\mathbf{W})[\mathbf{H}_2] - \gamma \mathrm{D} F'(\mathbf{W})[\mathbf{H}_2] \right). \end{aligned} \quad (36)$$

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

$$\mathrm{D}^2 \tilde{J}(\mathbf{W}^*)[\mathbf{H}_1, \mathbf{H}_2] = \frac{1}{N} \mathrm{vec}(\mathbf{H}_1)^\top \underbrace{\left(G(\mathbf{W}^*) - \gamma G'(\mathbf{W}^*) \right)^\top \left(G(\mathbf{W}^*) - \gamma G'(\mathbf{W}^*) \right)}_{\mathbf{H}_{\mathbf{W}} \tilde{J}(\mathbf{W}^*) \in \mathbb{R}^{N_{net} \times N_{net}}} \mathrm{vec}(\mathbf{H}_2). \quad (37)$$

It becomes clear that Proposition 2 applies almost unchanged.

Proposition 5 (Properties of the approximated Hessian). *The Hessian of the NMSBE function \tilde{J} from Equation (31) is at any critical point \mathbf{W}^* always positive semi-definite. Furthermore, its rank is bounded from above by*

$$\mathrm{rank}(\mathbf{H}_{\mathbf{W}} \tilde{J}(\mathbf{W}^*)) \leq N.$$

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 $\mathbf{H}_{\mathbf{W}} \tilde{J}(\mathbf{W}^*)$ being the product of these we get the upper bound on its rank. \square

The question, whether $\mathbf{H}_{\mathbf{W}} \tilde{J}(\mathbf{W}^*)$ is positive definite or only semi-definite, is the same as in the discrete setting. We just have to use $\tilde{G}(\mathbf{W}^*)$ in place of $G(\mathbf{W}^*)$ alone and still arrive at

$$N_{net} \geq N \quad (38)$$

to allow the rank to become full. We want to emphasize that this requirement for the Hessian is far less restricting than in the discrete setting with exact learning. Also, computational concerns regarding a second-order optimisation method are no longer that severe with nowadays hardware capabilities. MLPs can become large enough for RL applications while still allowing for working with Hessians in a reasonable time. We further address practical concerns in Section 5 and as part of the experiments.

In summary, we have two kind of approaches, namely exact learning and sampling based approximation. For discrete state spaces we have a sound and exact algorithm with verifiable local quadratic convergence as shown later. Furthermore, this algorithm can be converted to sampling based loss definition at any time without extra effort. For continuous state spaces only a sampling based algorithm is realisable. As we have shown, it possesses a matching behaviour with only

slightly altered propositions. Thus, the only major difference is the formulation of the loss. By using sampling we allow for broader applications without sacrificing theoretical properties. The last uncertainty left is how many sample states are required, i.e., how to select the size of N for a certain MDP. We treat this problem concerning sampling complexity as future research.

5 A Gauss Newton Residual Gradient Algorithm

In this section, we provide details regarding our concrete algorithm and make use of the results from our 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. We give pseudo code as it is used in the experiments and demonstrate numerically theoretical properties of a Gauss Newton algorithm using the discrete exact learning setting.

5.1 Second-Order Algorithms

A frequent concern regarding second-order algorithms is the computational effort involved in models with many parameters. Typical architectures of MLPs used in RL applications contain around two hidden layers with approximately 50 units in each layer. 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 we demonstrate with our experiments.

More severe is the root seeking behaviour of a Newton’s method. By using second-order information in a gradient descent optimisation procedure, we aim directly at any root in the gradient vector field of the NMSBE. Thus, the optimiser is also attracted by (local) maxima or even saddle points. Fortunately, as we have already elaborated after Propositions 2 and 5, these types of extrema do not exist if our assumptions are satisfied. On top of that, only at critical points of $\tilde{\mathcal{J}}(\mathbf{W})$ our approximated Hessian is exact. This makes it necessary to define the Approximated Newton’s step $\eta \in \mathcal{W}$ for an arbitrary point in the parameter space as the solution to a regularised linear equation system. Using the gradient from Equation (33) directly and the Hessian from Equation (37) combined with an identity matrix times a small factor we solve

$$\left(H_{\mathbf{W}} \tilde{\mathcal{J}}(\mathbf{W}) + c I_{N_{net}} \right) \eta = \nabla_{\mathbf{W}} \tilde{\mathcal{J}}(\mathbf{W}) \quad (39)$$

for η , where $c = 10^{-5}$ controls the strength of regularisation. All experiments use this value if not otherwise stated. The regularisation is important, because the approximated Hessian is only valid in a restricted neighbourhood around critical points. Outside of critical points, the regularisation ensures that the gradient direction is always part of the Newton’s step. Thus, this disturbance would also help with avoiding saddle points, as the descent direction is not coinciding completely with a correct Newton’s direction. Additionally, for first-order only information saddles and maxima are numerically unstable.

In our implementation, we do not make use of automatic differentiation frameworks due to the following reasons. First, they are not able to introduce approximations to symbolic expressions on their own. As we already have the required differential maps available due to our theoretical investigation, we could realise the optimisation procedure by hand without too much overhead. Second, to the best of our knowledge, the operations involved in the Hessian are not suited for general purpose graphics processing units, hence the performance gain of automatic differentiation frameworks is minimal. Third, as we have to solve linear equation systems, we cannot avoid to use other libraries as well. This would add additional overhead in the data transfer and further neglect benefits of these frameworks. In conclusion, by implementing the algorithm manually we achieved the full control over all the components and also could make use of sophisticated parallelisation. Of course, recent developments and contributions to automatic differentiation frameworks can render these considerations obsolete.

5.2 Pseudocode

In our numerical analysis we investigate two RL procedures. The first one is *Policy Evaluation* as shown in Algorithm 1. It makes use of a Gauss Newton Residual Gradient algorithm to approximate V_{π} with a given MLP. The required inputs are initial parameters for the MLP as well as a policy to evaluate. As output one obtains the optimal parameters. The second algorithm is normal *Policy Iteration*, which makes use of the first and extends it with a *Policy Improvement* step. After every sweep consisting of evaluation and improvement a better performing policy should be available. It is depicted in Algorithm 2.

A common practice in Policy Iteration is to reuse the most recent evaluation outcome for the next iteration. In [Sigaud and Stulp, 2019], this practice is called *persistent* whereas running the evaluation from scratch in every sweep is referred to as *transient*. We employ this naming convention here as well.

Algorithm 1 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 $f \in \mathcal{F}(n_0, \dots, 1)$ with initialised parameters $\mathbf{W} \in \mathcal{W}$
- policy π
- 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 $F(\mathbf{W}) := [f(\mathbf{W}, s_1) \dots f(\mathbf{W}, s_N)]^\top \in \mathbb{R}^N$ and its differential map $G(\mathbf{W})$.
- 4: Evaluate F' and G' using s'_i .
- 5: NMSBE: $\tilde{\mathcal{J}}(\mathbf{W}) = \frac{1}{2N} \Delta_\pi(\mathbf{W})^\top \Delta_\pi(\mathbf{W})$
- 6: Gradient: $\nabla_{\mathbf{W}} \tilde{\mathcal{J}}(\mathbf{W}) = \frac{1}{N} \left(G(\mathbf{W}) - \gamma G'(\mathbf{W}) \right)^\top \Delta_\pi(\mathbf{W})$
- 7: Hessian: $\mathbf{H}_{\mathbf{W}} \tilde{\mathcal{J}}(\mathbf{W}) = \frac{1}{N} \left(G(\mathbf{W}) - \gamma G'(\mathbf{W}) \right)^\top \left(G(\mathbf{W}) - \gamma G'(\mathbf{W}) \right)$
- 8: Solve for η : $\left(\mathbf{H}_{\mathbf{W}} \tilde{\mathcal{J}}(\mathbf{W}) + cI_{N_{net}} \right) \eta = \nabla_{\mathbf{W}} \tilde{\mathcal{J}}(\mathbf{W})$ (e.g. with Householder QR-Decomposition)
- 9: Descent step: $\mathbf{W} \leftarrow \mathbf{W} - \alpha \eta$
- 10: **while** $\tilde{\mathcal{J}}(\mathbf{W}) > \epsilon$

Algorithm 2 Policy Iteration

Hyper parameters: $\gamma \in (0, 1)$, $\alpha > 0$, $c = 10^{-5}$, $\epsilon \leq 10^{-5}$, $N \sim N_{net}$, $sweeps > 0$

Input: MLP $f \in \mathcal{F}(n_0, \dots, 1)$ with parameters $\mathbf{W} \in \mathcal{W}$,

Output: policy π

- 1: Draw \mathbf{W} elementwise uniformly from $[-1, 1]$
- 2: $\pi(s) \leftarrow \text{GIP}(f, \mathbf{W}, s)$ for any $s \in \mathcal{S}$
- 3: Draw s_i for $i = 1, \dots, N$ uniformly from \mathcal{S}
- 4: **for** sweep in sweeps **do**
- 5: **if not** persistent **then**
- 6: Draw \mathbf{W} elementwise uniformly from $[-1, 1]$
- 7: **end if**
- 8: $\mathbf{W} \leftarrow \text{PolicyEvaluation}(f, \mathbf{W}, \pi, s_1, \dots, s_N)$
- 9: $\pi'(s) \leftarrow \text{GIP}(f_\pi, \mathbf{W}, s)$ for all $s \in \mathcal{S}$
- 10: Evaluate π' empirically using several roll-outs
- 11: $\pi \leftarrow \pi'$
- 12: **end for**

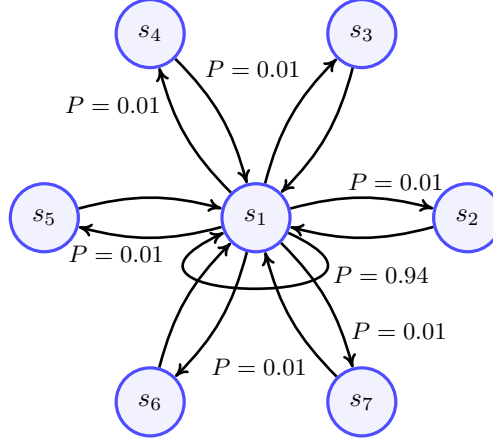


Figure 1: Adapted version of Baird’s *Seven State Star Problem* [Baird III, 1995]. We have added transitions with low probabilities from the central node back to the six outer states to obtain an infinite horizon problem such that the NMSBE is well-defined.

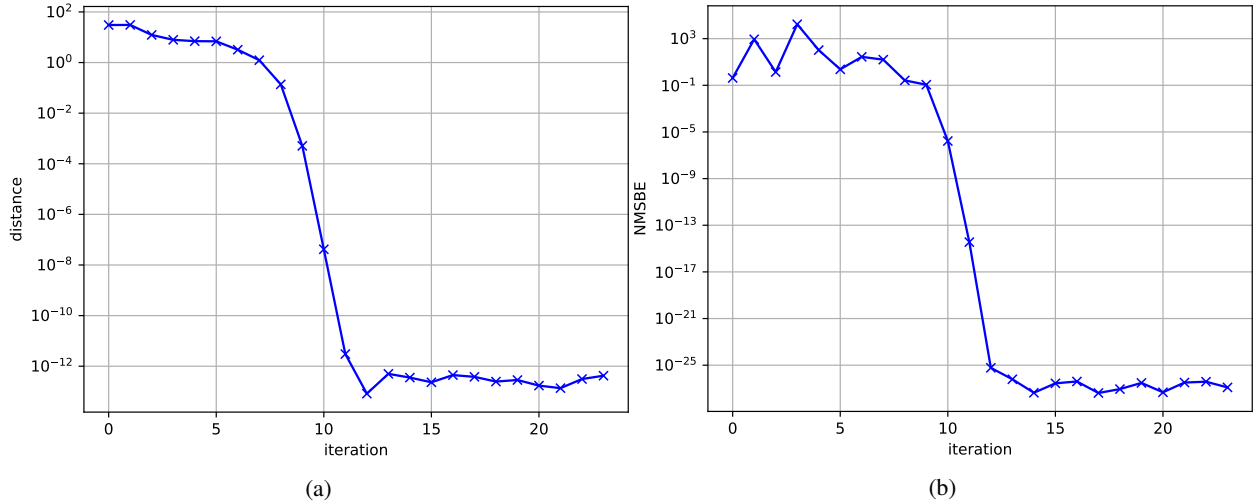


Figure 2: A typical local quadratic convergence behaviour for an adapted version of Baird’s *Seven State Star Problem*. **a)**: Distance of iterates $\mathbf{W}^{(k)}$ to the accumulation point \mathbf{W}^* . **b)**: The NMSBE corresponding to each iterate.

5.3 Demonstration of Local Quadratic Convergence

To evaluate empirically our derived theoretical results in the discrete state space setting together with the assumption of exact learning, we demonstrate local quadratic convergence on Baird’s *Seven State Star Problem* [Baird III, 1995]. Only if all components work as intended, local quadratic convergence of a second-order gradient descent algorithm can be visible. As we require a problem, where all states occur infinite many times for the NMSBE to be well-defined, we extend the Star Problem with transitions from the central node to all others. In Figure 1, all transitions and their probabilities are shown. A policy for evaluation is defined implicitly by setting the transition probabilities to fixed values. The reward is present at the central node with value one. As discount factor we use $\gamma = 0.99$.

We 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 2, we visualise convergence behaviour with the distance of the accumulation point \mathbf{W}^* to all iterates $\mathbf{W}^{(k)}$ as well as by the corresponding NMSBE. We 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 2a that the AN algorithm converges locally quadratically to a minimiser. Judging from the negligibly small final NMSBE as seen Figure 2b, we 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 \mathbf{W}^* and additionally that its rank is full. Due to over-parametrisation, convergence to suboptimal local minima does not happen.

6 Experiments in Continuous State Spaces

For our empirical investigation of the algorithm we first outline the experimental setup. Second, we test the convergence behaviour in continuous state spaces. We compare Residual Gradients with Semi-Gradients and quantify the influence of second-order optimisation. Next, we 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. Finally, we address the application of a second-order Residual Gradient algorithm in full Policy Iteration and test it in a continuous control problem.

6.1 Experimental Setup

We apply the Gauss Newton Residual Gradient algorithm for Policy Evaluation in finite dimensional and bounded Euclidean state spaces. We provide empirical results for the performance of the Approximated Newton’s algorithm by minimising the objective of Equation (31) in several different scenarios:

1. **Convergence of Policy Evaluation** by evaluating the value function of a fixed policy;
2. **Generalisation Capabilities** by characterising the influence of different MLP architectures on the generalisation performance;
3. **Full Policy Iteration** by combining Policy Evaluation and Q -factors to improve iteratively an initial policy.

Investigating performance of the algorithm with generalisation in mind is both interesting and important due to still-unexplainable capabilities of neural networks [Lawrence et al., 1997, Zhang et al., 2017, Neyshabur et al., 2017]. Testing in a full Policy Iteration setting is important for the general applicability of a Gauss Newton Residual Gradient algorithm.

In our Policy Evaluation experiments, except for generalisation, we compare the cases of whether or not considering derivatives of the TD-target. This means we compare Semi-Gradient and Residual Gradient formulations for both first and second order optimisation methods.

For the experiments, we use the *Mountain Car* [Moore, 1990] and the *Cart Pole* control problems [Barto et al., 1983] as environments. These two are classic deterministic Reinforcement Learning benchmarks with a typical continuous state space and a manageable complexity that allows for an extensive investigation and visualisation. More specifically, we employ the *MountainCar-v0* environment of the *OpenAI Gym* package [Brockman et al., 2016], and include important changes to the environment to obtain infinite horizon problems. First, we replace the built-in constant reward function, which assigns independently of a state, an action and its successor state to any transition a punishment of -1 , with a function that rewards being in a goal region by not giving punishments in that area. We define the goal region as $x > 0.5$, where x denotes the position of the car. Second, once the car enters the goal region we teleport it back to a starting state. These two adjustments allow us to use the environment in a non-episodic fashion and, thereby, establish the required formulation of the MDP to work with uniformly sampled state transitions. Without these adjustments ergodicity cannot be present and the objective would not be well-defined. Similarly, the *CartPole-v1* environment of *OpenAI Gym* receives a new non-constant reward and a new transition. 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, we only punish with -1 reward the transition into those failure states. Otherwise, the reward is zero. By adding connections from terminal region to the start state, we convert the episodic formulation of the balancing task and restore also here an infinite horizon MDP. We set the discount factor for all environments to $\gamma = 0.99$ if not otherwise stated.

6.2 Empirical Convergence Analysis

Setting We first make use of the Mountain Car problem, which has a continuous state space and discrete actions, and investigate the performance when evaluating a predefined policy. The complexity and dimensionality of the problem is manageable, which enables us to estimate the ground truth value function with Monte Carlo methods based on a fine grained two dimensional grid. Thus, we can evaluate accurately the performance of tested algorithms against the ground truth. The policy, whose value function is being evaluated, is fixed to accelerate the car in the direction of the current velocity.

We investigate the performance of the algorithm under the influence of three variants: *ignoring TD targets in derivatives*, *using Hessian based optimisation* and *varying learning rates*. We use four different learning rates $\alpha \in \{10^0, 10^{-1}, 10^{-2}, 10^{-3}\}$ and study all 16 combinations.

For all tests, we adopt the batch learning scenario. Specifically, we use as training set 100 transition tuples (s, r, s') collected in prior with the fixed policy. All states are sampled uniformly from \mathcal{S} . Note that changing the ξ -weighted norm to an equivalent one, for example the uniform distribution used here, can only change the steepest descent direction. The types and locations of critical points remain the same. Executing the action provided by the policy in each state yields its successor s' and the one-step-reward r for that transition. We fix the transition tuples throughout all convergence experiments to provide a fair comparison between individual runs.

As function space for approximated value functions, we 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 our theoretical findings. Furthermore, MLPs in this experiment are all initialised with the same weight matrices to further improve the comparability. For initialisation we use a uniform distribution in the interval $[-1, 1]$.

Although there is no noise or randomness in the environment part involved and we do not make use of exploration mechanisms, the outcomes can still vary. We use an asynchronous 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. Our results of the optimisation processes are shown in Figure 3.

Results We observe that a Semi-Gradient algorithm always diverges for extremely large step sizes as shown in Figure 3a with $\alpha = 1$. For smaller step sizes $\alpha \in \{10^{-1}, 10^{-2}, 10^{-3}\}$ as shown in Figures 3b, 3c and 3d, 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, we want to point out that the resulting solution of Policy Evaluation is not good compared to other possible outcomes.

Figures 3b and 3c 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, we can confirm the slow convergence issue of a Residual Gradient formulation. From Figure 3d 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 3a and 3b. Even for extremely large learning rates $\alpha = 1$ 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 that when combining Semi-Gradient algorithms with an annealing scheme on the learning rate, even better performance can be obtained [Gronauer and Gottwald, 2021].

Computational concerns A severe burden of Newton-type algorithms is the computational complexity involved in evaluating the Hessian, especially since the Gauss Newton approximation involves a full sized and dense matrix. In this experiment, first-order only methods require roughly 8.4 and 7.6 seconds on average for 10^4 iterations of Semi- and Residual Gradients, respectively. Our 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. Calculating the Hessian and Newton’s direction⁷ increases the run time to 22.4 and 48.5 seconds, respectively. 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 our computers. However, the performance gain in terms of convergence speed and overall lower error, as seen in Figure 3b or Figure 3c, 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. The performance could be further enhanced if we would make use of concepts such as the *Levenberg-Marquardt* heuristic or put additional effort in hyper parameter tuning.

⁷using *Eigen*’s Householder QR-decomposition when linked against *OpenBlas* and *OpenLapack*

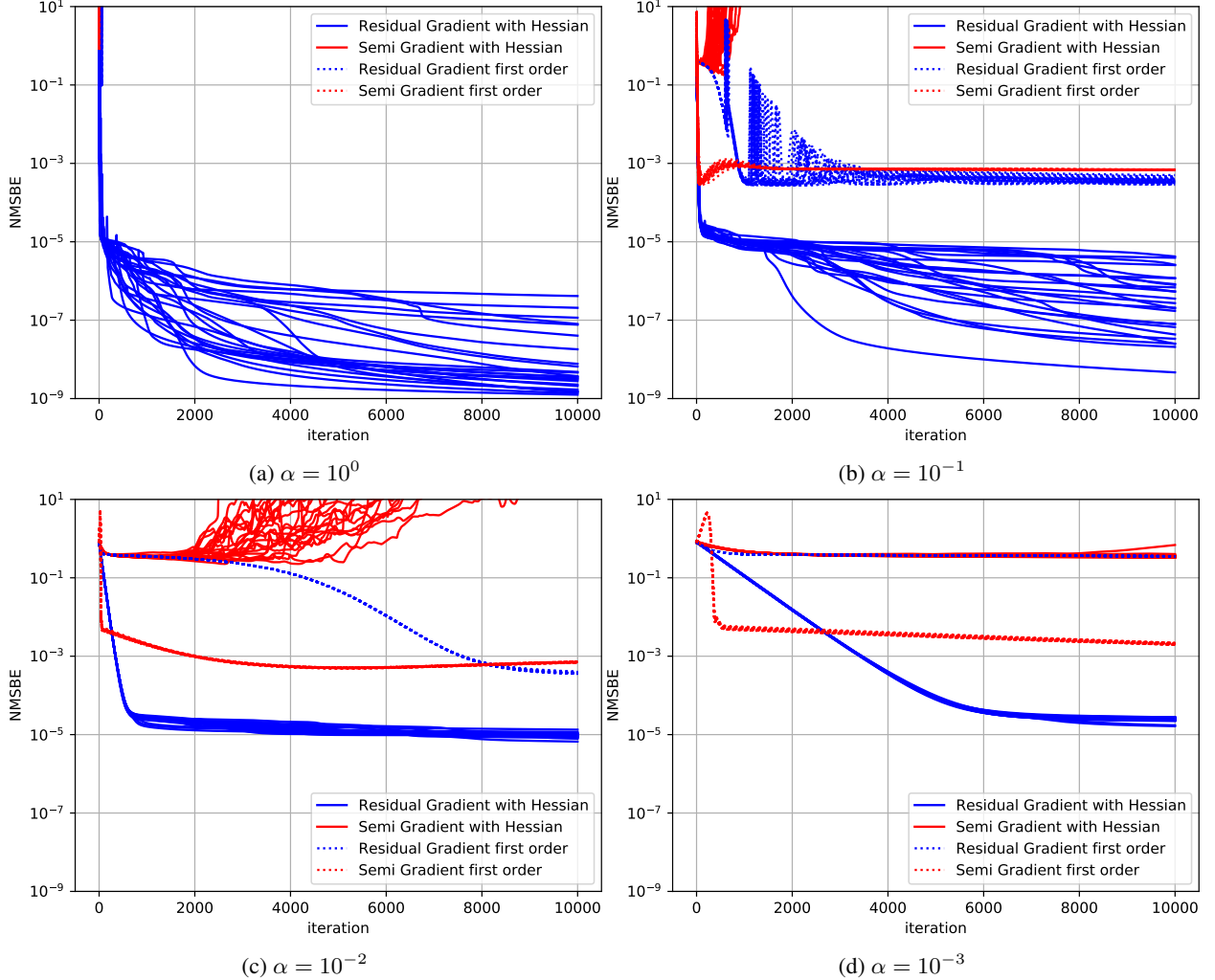


Figure 3: The NMSBE as defined in Equation (31) plotted over time for all tested optimisation approaches. Figure a) to d) represent considered learning rates $\alpha \in \{10^0, 10^{-1}, 10^{-2}, 10^{-3}\}$. A Residual Gradient formulation combined with Hessian based optimisation outperforms Semi-Gradient 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.

6.3 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 RL and worth further investigation. Thus, we evaluate an approximated value function, which we obtain by optimising the NMSBE with the Gauss Newton Residual Gradient algorithm, with states that were not part of the training data.

6.3.1 Single Architecture

Setting We start with a single architecture and vary the amount of training samples. More specifically, we consider again the MLP $\mathcal{F}(2, 10, 10, 1)$ with learning rate $\alpha = 10^{-2}$ and *Bend-Id* activation function. For the initialisation we 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 and the NMSBE is computed for a separated test set comprised of $25 \cdot 10^4$ states arranged on a grid. Again, we repeat for each N the training of MLPs 25 times and visualise the NMSBE as box-plots in Figure 4a for the training data and in Figure 4b for the held-out test set.

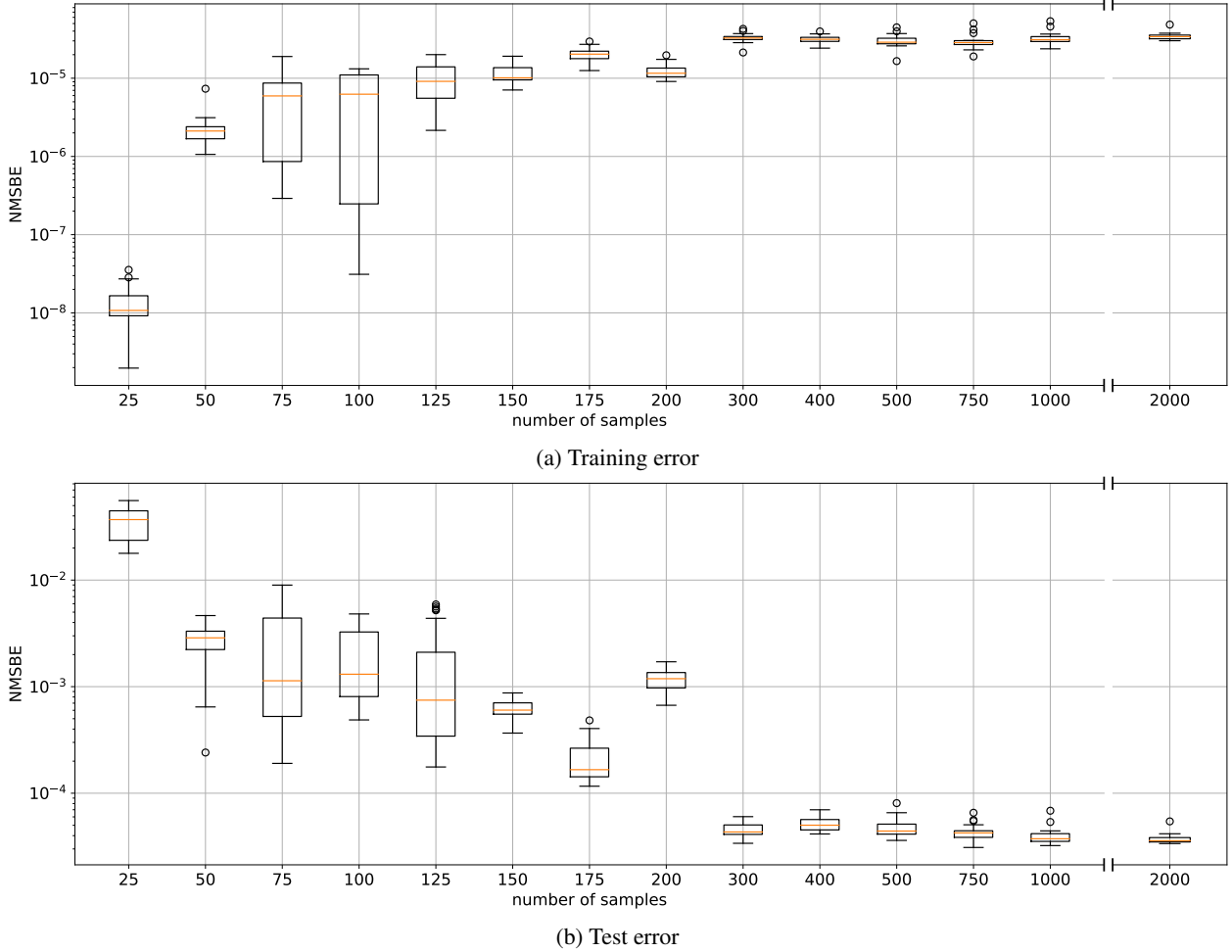


Figure 4: The final NMSBE of the MLP $\mathcal{F}(2, 10, 10, 1)$ for different amount N of training samples. **Top:** NMSBE using the training samples. **Bottom:** NMSBE for the test set.

Results It is clear that at $N = 2000$ the training process is always successful with almost ignorable variance. While decreasing the number of randomly placed training samples down to $N = 300$, the mean error becomes slightly larger with more pronounced variances. Furthermore, only poor results are observed at $N = 25$. Here, the MLP is able to fit the training data exactly, but the test error indicates that this is no longer a valid solution.

We observe for the test error that the variance, when using $N = 150$ samples, is smaller than that of its direct neighbours. Simultaneously, there is a sharp reduction in the variance for training errors starting at $N = 125$. Since these numbers of samples are almost identical or at least close to the number of parameters in the network ($N_{net} = 151$) this could be a numerical confirmation of the theoretical condition as derived earlier. However, we admit that an empirical verification is rather involved. In particular, we 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 $N = 25$ samples visually, i.e., see Figure 5b, one can spot a plateau located at around -100 expected discounted reward. This is exactly the solution to Bellman’s equation or, more precisely, to the loss as defined in Equation (31) if every transition would yield -1 reward. We conclude that those scarce samples do not allow the reward information to flow and hence we have solved implicitly a different MDP. By comparing the ground truth value function as shown in Figure 5a, which we obtain by running rollouts from every grid state in the test set, to other value functions learned with different sample sizes as shown in Figures 5c to 5f, we see that training with even only 50 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 interactions.

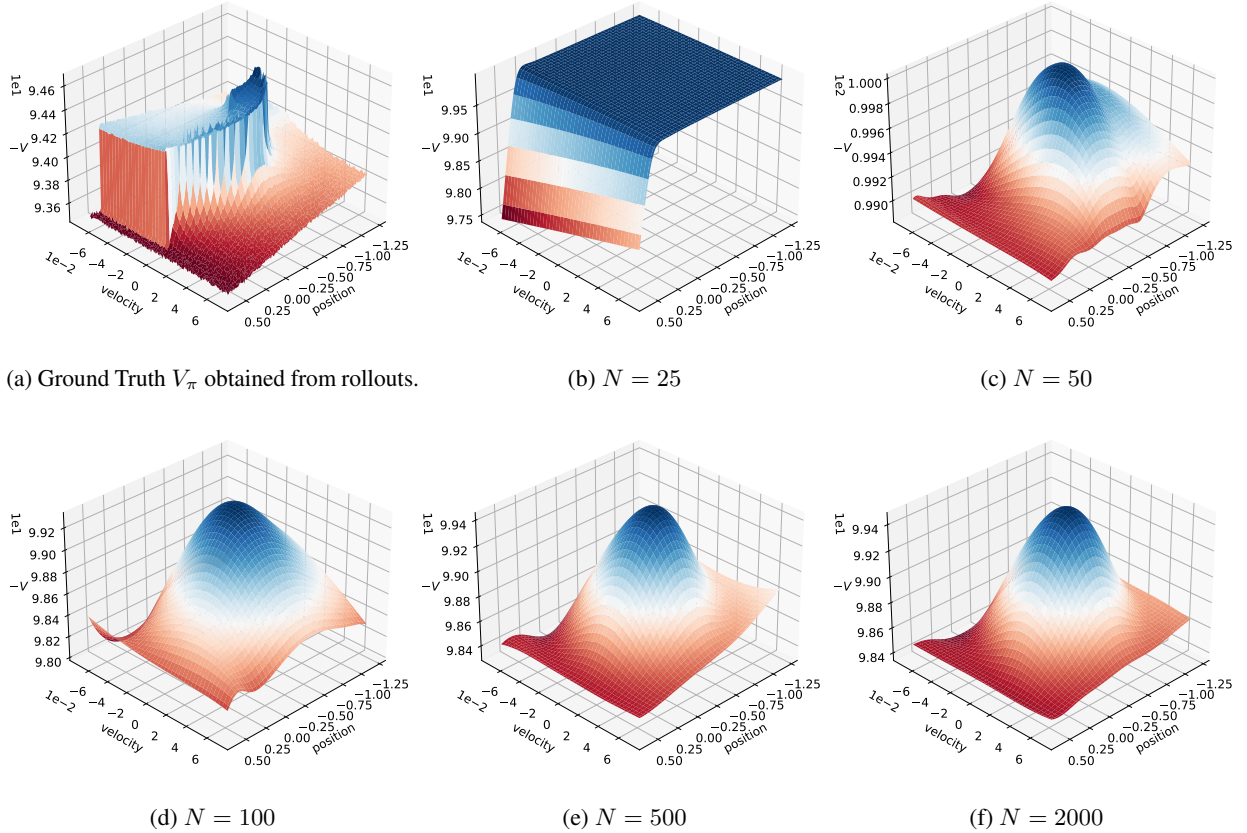


Figure 5: Approximated value functions for different batch sizes N with minimal test NMSBE, evaluated at the same states as for the MC version.

6.3.2 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, we train several MLPs with different architectures and a varying number of samples in the same scenario as before. We refer to the axis labels in Figure 6 for concrete choices of architectures and the values of N . Due to the huge amount of computation involved in deeper networks with large sample sizes, we reduce the number of repetitions from 25 to ten. We provide the mean test and training error as contour plots in Figure 6 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 In the following, we look separately at training and testing errors and start with the training errors, i.e., the contour plots in the left column of Figure 6. We see that the shapes of isolines for all MLPs but the smallest ones match the shape of the line $N_{net} \approx N$, which we call *condition line* in the following. We obtain the condition line by rounding the amount of parameters N_{net} to the closest value of N and joining those points. The shape implies that the area with same training error follows the condition in Equation (38) from our theoretical investigation. When we increase the amount of data, we also have to increase the depth or width of an MLP, and vice versa, to maximise the chance of avoiding suboptimal minima. Between Figures 6a and 6c the shapes of isolines stay consistent and furthermore, the condition line is present at roughly the same training error of -4.994 to -4.697 . Exact learning of all samples indicated by training errors close to zero happens only above the solid black line, i.e., whenever the network has more parameters than the number of training samples.

Next, we address the testing error as shown in the right column of Figure 6. For larger MLPs, where the condition $N_{net} \approx N$ is available for large values of N , we observe that the region with smallest test error extends always to the condition line. This happens e.g. for the MLP $\mathcal{F}(2, 15 \times 3, 1)$ at $N = 500$ or for $\mathcal{F}(2, 20 \times 3, 1)$ at $N = 900$. We see that larger MLPs work more predictable where smaller MLPs can achieve the smallest test error for several training set

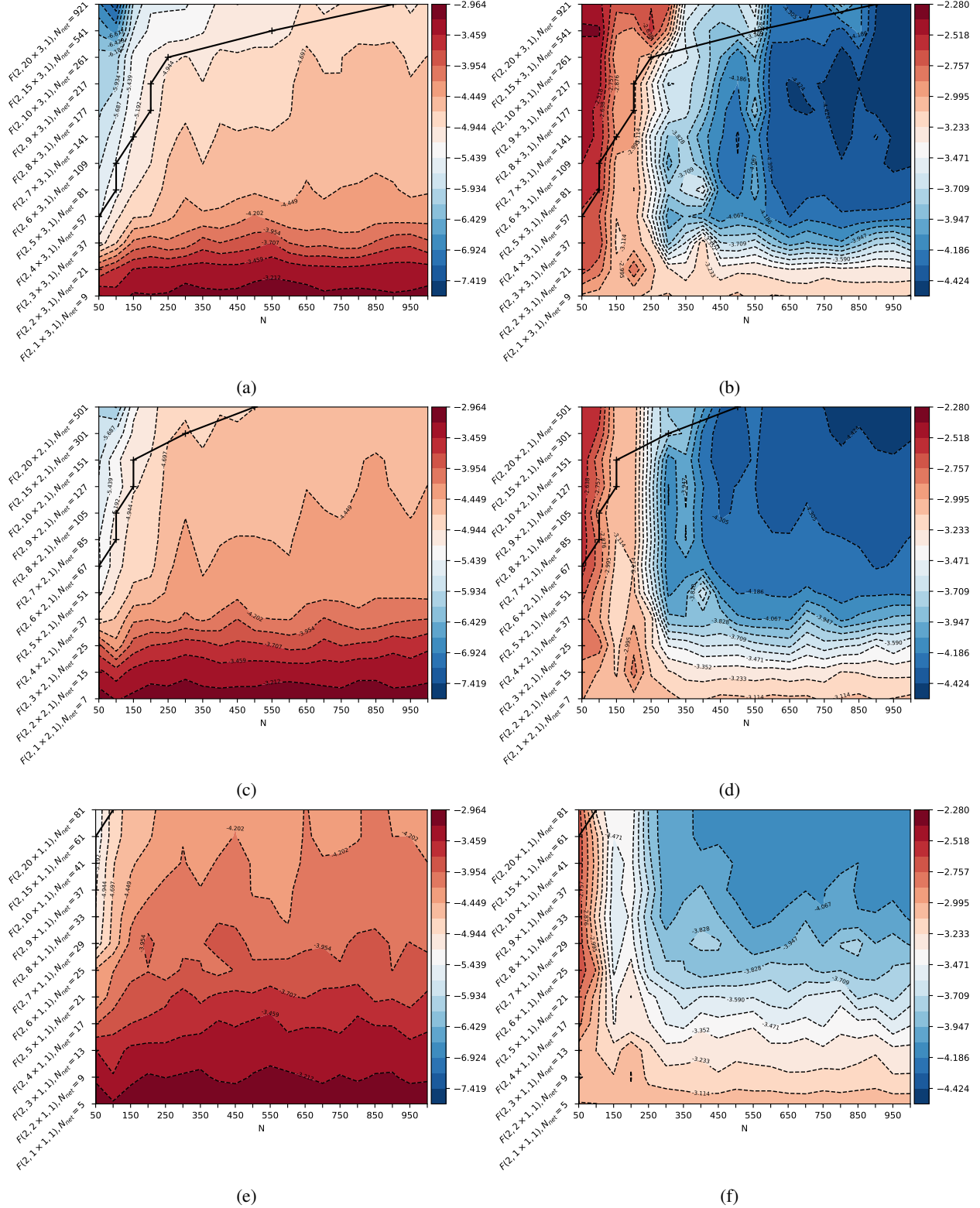


Figure 6: The training and test error of different MLP architectures (ordinate) for various sample sizes N (abscissa). We use a logarithmic scale $Z = \log_{10}(E)$, where E is the original error and Z its plotted value. Red indicates higher errors. In all plots the solid line represents the condition $N_{net} = N$. **Left column:** Training error. **Right column:** Test error. For training errors one can see that shape and position of isolines follow roughly the condition $N_{net} = N$. Lower training error as well as the case of exact fitting happens to the upper left of the black solid line, confirming our considerations from Section 4. 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 $N_{net} = N$. However, for all networks the well-known rule “the more data, the better performance” applies, presumably because not all sample states are far apart.

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 our 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 $\mathcal{F}(2, 6 \times 3, 1)$, 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. We need 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 $N \rightarrow 50$, the largest architectures produces out of all runs the highest test error. Hence, by shrinking the amount of parameters in an MLP such that $N_{net} \approx N$ applies again one can reduce the test error without changing the amount of data. If $N \rightarrow 1000$, 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 highest generalisation performance out of all architectures.

6.4 Policy Iteration

Setting For the final experiment, we change the environment to Cart Pole and test our approach in a Policy Iteration setting. For that purpose, we use Q -factors instead of $V_\pi(s)$ and use a greedily induced policy as defined in Equation (7). To represent Q -factors, the network input consists of the continuous state and the discrete action index, hence MLPs must have $K + 1$ units in the input layer. More specifically, we use the MLP $\mathcal{F}(5, 10, 10, 1)$ with $N_{net} = 181$ parameters to approximate the Q -function.

We test different sample sizes $N \in \{100, 181, 300, 500\}$, i.e., the number of states sampled before every Policy Evaluation, and investigate different amounts of policy evaluation steps $i \in \{500, 1500, 2500, 3500, 4500\}$, i.e., the number of descent steps. Finally, we also compare the effect of reusing the last parameters of the previous Policy Evaluation step as initialisation for the current one and refer to this by *transient* and *persistent* as in [Sigaud and Stulp, 2019].

We measure the performance of a policy by performing ten rollouts with each 500 steps after every improvement step and combine the rollouts as mean, minimal and maximal value. Furthermore, we repeat every experiment five times such that we can provide averages of expected returns at each sweep as a summarising impression of the Policy Iteration processes. Shaded areas represent the mean of all individual spreads of discounted returns. Our results are available in Figure 7.

Results Since experiments with Semi-Gradients diverged for both first and second order descent algorithms and a first-order Residual Gradient formulation did not show improving policies over time, we only provide results for the Gauss Newton Residual Gradient formulation. Thus, this experiment already implies that second-order information is essential to enable and stabilise Policy Iteration in continuous problems. 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 in most plots. Starting with a fresh MLP for each evaluation increases the number of sample states to see a more pronounced improvement over time. When using persistent evaluations, i.e., reusing the MLP parameters corresponding to the last policy, clear improvements are visible also for small sample sizes. Furthermore, the maximum achieved performance is higher than for transient sweeps.

In all plots of Figure 7, it is hard to find a clear trend for the required number of samples or the optimal amount of Policy Evaluation iterations. As an example, in Figure 7d both $i = 1500$ and $i = 4500$ have a similar high performance around sweep 175. In Figure 7e, using only $N = 100$ samples result for $i = 3500$ reliably in significant better performing policies than when increasing the amount of Policy Evaluation steps to $i = 4500$.

The fact that for $N = 181 = N_{net}$ samples and when using persistent evaluations of policies all repetitions of the experiment come up with good performing policies for the largest amount of Policy Evaluation steps is worth further investigation. But based on our currently available experimental results we cannot make reliable statements at this point. Although we observe a slightly chaotic behaviour, we argue that this is to be expected, since function approximation is utilised in a Policy Iteration framework. Thus, there are two unavoidable sources of error, namely inaccurate Policy Evaluation and erroneous Policy Improvement. First, different than typical settings where samples come from an exploration mechanism, we select samples uniformly distributed in the whole state space. Thus, our optimisation problem does not have a high resolution focused on visited parts of the state space but tries to find a global solution, which is obviously a more challenging problem. Second, we make use of discrete actions. This means, we ask a smooth function approximation architecture to model jumps in the value function, which must occur for example around the balancing point of the pole.

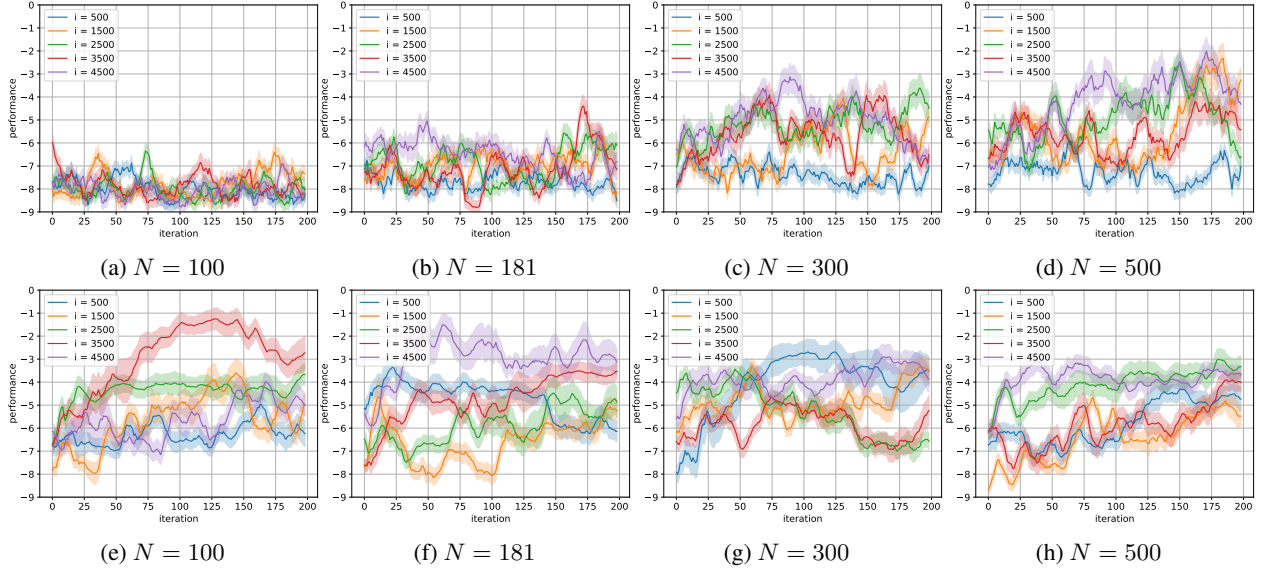


Figure 7: Combining the Residual Gradient Gauss Newton Policy Evaluation with Q -factors and GIP policies to full Policy Iteration. As environment we use *Cart Pole* and plot the expected reward obtained by performing roll-outs from 5 repetitions. From left to right we vary the number of sampled states. In each Figure the number of Policy Evaluation iterations is changed. In the top row Policy Evaluation starts with a new MLP in every sweep. For the bottom row we reuse the last evaluation of the previous sweep as initialisation for the current one.

Sequentially improving policies are visible, especially when reusing the MLP, however, there is no clear trend for the required number of samples or the optimal amount of Policy Evaluation iterations. Without second-order optimisation or when using Semi-Gradients Policy Iteration did not work at all.

7 Conclusion

RL with NN-VFA has become one of the most powerful RL paradigms in both research and application in the recent years. Despite the superior performance, its training and convergence analysis remains challenging due to an incomplete theoretical understanding how MLPs affect the common RL setting. In particular, most previous theoretical analysis of this problem approaches the challenge from the perspective of minimising Mean Squared Projected Bellman Error which is incompatible with recent successful applications.

This work bridges the gap with a concise critical point analysis of the NMSBE when using MLPs. We address both the discrete and continuous state space setting for a Residual Gradient formulation, i.e., using the complete gradient of the NMSBE. We derive conditions on MLPs to ensure a proper behaviour of the optimisation procedure. Over-parametrisation of the MLP is required next to some design principles for MLPs to eliminate suboptimal local minima. Furthermore, full rankness of the differential maps of the MLP enable pleasing convergence properties of gradient descent algorithms. Our analysis unveils the possibility to utilise approximated second-order information of the cost function, resulting in an efficient Approximate Newton’s method, namely a Gauss Newton Residual Gradient algorithm. As part of our work, we also see, why multistep lookahead can help Semi-Gradient algorithms. As the MLP in the TD target gets multiplied with the discount factor with larger powers the effect of ignoring the dependency vanishes naturally. Furthermore, we point out a source of error when using sampling based approximations to the NMSBE. Critical points also contain solutions with zero NMSBE, which not necessarily correspond to good approximations of the value function. The optimisation problem can have undesired degrees of freedom.

In several experiments, we investigate empirically the minimisation of the NMSBE using a Gauss Newton Residual Gradient algorithm. First, we ensure the correctness of the approximated Hessian close to critical points by demonstrating quadratic convergence on an adapted version of Baird’s Seven State Star Problem.

Next, in continuous state space problems, we provide consistent and stable convergence properties of Residual Gradient algorithms combined with second-order information for a wide range of learning rates. Semi-Gradient algorithms are observed to diverge except for some learning rates. Carefully selecting the learning rate is thus important. First-order only Residual Gradients are shown to converge slowly, confirming the known behaviour which explains their unpopularity. Using an approximated Hessian can solve these issues and is essential to develop stable and efficient RL

algorithms, which also achieve better final errors. Our experiments also serve as proof of concept that with modern computer systems second-order optimisation is a possible approach in Reinforcement Learning applications.

Furthermore, we investigate the generalisation capabilities of MLPs when training with approximated second-order Residual Gradient algorithms. Training and test errors follow our theoretical insights, meaning that the number of parameters should be synchronised with the number of training samples. We find that deeper architectures do not necessarily increase the number of samples required for good performance, they rather amplify the extreme cases for errors.

Finally, we demonstrate that the application of Residual Gradient methods can work in a full Policy Iteration setting. A well performing policy can be learned by alternating between Policy Evaluation and Policy Improvement, starting from a random one. However, our empirical findings did not reveal trends on the number of samples or Policy Evaluation steps needed. Our hypothesis is that the discrete action space stands in conflict with the capabilities of a smooth function approximators such as MLPs. Solving the RL task involves representing jumps in the value function, which might not be possible. Hence, including a continuous action space in our analysis is an important next step for our future work.

In conclusion, considering derivatives of the TD-target as done in Residual Gradient algorithms allows us to perform a sophisticated analysis and also provides the foundation for reliable NL-VFA algorithms. Smooth optimisation is a promising methodology to answer open issues in Deep RL. It unveils important details for the construction of efficient algorithms and outlines difficulties in the formulation of the optimisation task. Hence, next steps for future work are to extend this approach to policy gradient methods and the actor-critic frameworks to allow for continuous action spaces in the analysis.

Acknowledgements

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

References

- P. A. Absil, R. Mahony, and R. Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, 2008.
- L. C. Baird III. Residual algorithms: Reinforcement learning with function approximation. In *Proceeding of the 12th International Conference on Machine Learning*, pages 30–37, 1995.
- L. C. Baird III and A. W. Moore. Gradient descent for general reinforcement learning. In *Advances in neural information processing systems*, pages 968–974, 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):834–846, 1983.
- D. P. Bertsekas. *Dynamic Programming and Optimal Control: Approximate Dynamic Programming*, volume 2. Athena Scientific, 4th 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*, pages 231–235. IEEE Press, 2004.
- N. Bhat, V. F. Farias, and C. C. Moallemi. Non-parametric approximate dynamic programming via the kernel method. In *Advances in Neural Information Processing Systems 25*, pages 386–394. The MIT Press, 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):2067–2118, 2013.
- 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](https://arxiv.org/abs/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](https://arxiv.org/abs/1905.10027), 2019.
- J. Fu, A. Kumar, M. Soh, and S. Levine. Diagnosing bottlenecks in deep q-learning algorithms. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 2021–2030, 2019.
- M. Geist and O. Pietquin. Algorithmic survey of parametric value function approximations. *IEEE Transactions on Neural Networks and Learning Systems*, 24(6):845–867, 2013.
- M. Gottwald, M. Guo, and H. Shen. Neural value function approximation in continuous state reinforcement learning problems. In *European Workshop on Reinforcement Learning*, 2018.
- S. Gronauer and M. Gottwald. The successful ingredients of policy gradient algorithms. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence*, pages 2455–2461, 2021.
- 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 (CVPR)*, pages 7331 – 7339, 2017.
- K. Kawaguchi. Deep learning without poor local minima. In *Advances in Neural Information Processing Systems*, volume 29, pages 586–594, 2016.
- S. Lawrence, C. L. Giles, and A. C. Tsoi. Lessons in neural network training: Overfitting may be harder than expected. In *Proceedings of the 14th National Conference on Artificial Intelligence and 9th Conference on Innovative Applications of Artificial Intelligence*, pages 540–545, 1997.
- Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–444, 2015.
- L.-J. Lin. *Reinforcement Learning for Robots Using Neural Networks*. PhD thesis, Carnegie Mellon University, 1993.
- B. Liu, Q. Cai, Z. Yang, and Z. Wang. Neural proximal/trust region policy optimization attains globally optimal policy. *Advances in Neural Information Processing Systems*, 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, pages 1204–1212, 2009.
- 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*, 518:529–533, 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 33rd International Conference on Machine Learning*, pages 1928–1937, 2016.
- 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):79–110, 2003.
- B. Neyshabur, S. Bhojanapalli, D. McAllester, and N. Srebro. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems 30*, pages 5947–5956, 2017.
- Q. Nguyen and M. Hein. The loss surface of deep and wide neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.
- R. Parr, C. Painter-Wakefield, L. Li, and M. Littman. Analyzing feature generation for value-function approximation. In *Proceedings of the 24th international conference on Machine learning*, pages 1–8, 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 25th international conference on Machine learning*, pages 752–759, 2008.
- M. Riedmiller. Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method. In *Proceedings of the 16th European Conference on Machine Learning*, pages 317–328, 2005.
- E. Saleh and N. Jiang. Deterministic bellman residual minimization. In *Optimization Foundations for Reinforcement Learning Workshop*, 2019.
- J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the 4th 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](https://arxiv.org/abs/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):10–18, 1999.
- H. Shen. A differential topological view of challenges in learning with feedforward neural networks. *Computing Research Repository*, [arXiv:1811.10304](https://arxiv.org/abs/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 (CVPR)*, pages 811–820, 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*, 113:28–40, 2019.
- D. Silver. Gradient temporal difference networks. In *Proceedings of the 10th European Workshop on Reinforcement Learning*, volume 24, pages 117–130, 2013.
- 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*, 529:484–489, 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 with deep neural networks and tree search. *Nature*, 550:354–359, 2017.
- R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. The MIT Press, 2nd edition, 2020.
- 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 21*, volume 21, pages 1609–1616, 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 26th International Conference on Machine Learning (ICML)*, pages 993–1000, 2009.
- G. Taylor and R. Parr. Kernelized value function approximation for reinforcement learning. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, pages 1017–1024, 2009.
- 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, pages 2094–2100, 2016.
- X. Xu, D. Hu, and X. Lu. Kernel-based least squares policy iteration for reinforcement learning. *IEEE Transactions on Neural Networks*, 18(4):973–992, 2007.
- D. Yu and L. Deng. *Automatic Speech Recognition: A Deep Learning Approach*. Springer-Verlag, London, 2015.
- C. Yun, S. Sra, and A. Jadbabaie. Global optimality conditions for deep neural networks. In *The 6th 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 5th International Conference on Learning Representations*, 2017.
- S. Zhang, W. Boehmer, and S. Whiteson. Deep residual reinforcement learning. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, page 1611–1619, 2020.

A Step by Step Calculations

Differential map of error function For the error function $E: \mathbb{R}^K \rightarrow \mathbb{R}$ and some $F \in \mathbb{R}^K$ we have

$$\begin{aligned}
 E(F) &= \frac{1}{2} \left(F - P_\pi(R_\pi + \gamma F) \right)^\top \Xi \left(F - P_\pi(R_\pi + \gamma F) \right) \\
 \text{D} E(F)[h] &= \frac{1}{2} \frac{d}{dt} \Big|_{t=0} \left[\left((F + th) - P_\pi(R_\pi + \gamma(F + th)) \right)^\top \Xi \left((F + th) - P_\pi(R_\pi + \gamma(F + th)) \right) \right] \\
 &= \frac{1}{2} \left[\left(h - P_\pi \gamma h \right)^\top \Xi \left((F + th) - P_\pi(R_\pi + \gamma(F + th)) \right) \right. \\
 &\quad \left. + \left((F + th) - P_\pi(R_\pi + \gamma(F + th)) \right)^\top \Xi \left(h - P_\pi \gamma h \right) \right]_{t=0} \\
 &= \frac{1}{2} \left[\left(h - P_\pi \gamma h \right)^\top \Xi \left(F - P_\pi(R_\pi + \gamma F) \right) + \left(F - P_\pi(R_\pi + \gamma F) \right)^\top \Xi \left(h - P_\pi \gamma h \right) \right] \\
 &= \left(F - P_\pi(R_\pi + \gamma F) \right)^\top \Xi \left(I_K - \gamma P_\pi \right) h.
 \end{aligned}$$

Thus, according to Riesz, the gradient is

$$\nabla_F E(F) = \left((F - P_\pi(R_\pi + \gamma F))^\top \Xi (I_K - \gamma P_\pi) \right)^\top,$$

Differential map of MLP Consider an MLP $f \in \mathcal{F}(n_0, n_1, \dots, n_{L-1}, n_L)$ and an input $s \in \mathcal{S}$. To calculate the differential map of f for s at the point $\mathbf{W} \in \mathcal{W}$ and a direction $\mathbf{H} \in \mathcal{W}$ first start with a single layer l of the MLP. We have

$$\text{D}_{W_l} f(\mathbf{W}, s)[H_l] = \text{D}_2 \Lambda_L(W_L, \phi_{L-1}) \circ \dots \circ \text{D}_2 \Lambda_{l+1}(W_{l+1}, \phi_l) \circ \text{D}_1 \Lambda_l(W_l, \phi_{l-1})[H_l],$$

where $\text{D}_1 \Lambda_l(W_l, \phi_{l-1})[H_l]$ and $\text{D}_2 \Lambda_l(W_l, \phi_{l-1})[h_{l-1}]$ 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 (17) we obtain

$$\begin{aligned}
 \text{D}_1 \Lambda_l(W_l, \phi_{l-1})[H_l] &= \frac{d}{dt} \Big|_{t=0} \left[\sigma \left((W_{l,k} + t \cdot H_{l,k})^\top \cdot \begin{bmatrix} \phi_{l-1} \\ 1 \end{bmatrix} \right) \right] = \left[\dot{\sigma}(\dots) H_{l,k}^\top \begin{bmatrix} \phi_{l-1} \\ 1 \end{bmatrix} \right]_{t=0} \\
 &= \text{diag}(\dot{\phi}_l) H_l^\top \begin{bmatrix} \phi_{l-1} \\ 1 \end{bmatrix} =: \Sigma_l \cdot H_l^\top \cdot \tilde{\phi}_{l-1} \\
 \text{D}_2 \Lambda_l(W_l, \phi_{l-1})[h_{l-1}] &= \frac{d}{dt} \Big|_{t=0} \left[\sigma \left(W_{l,k}^\top \cdot \left(\begin{bmatrix} \phi_{l-1} \\ 1 \end{bmatrix} + t \cdot \begin{bmatrix} h_{l-1} \\ 0 \end{bmatrix} \right) \right) \right] = \left[\dot{\sigma}(\dots) W_{l,k}^\top \begin{bmatrix} h_{l-1} \\ 0 \end{bmatrix} \right]_{t=0} \\
 &= \text{diag}(\dot{\phi}_l) W_l^\top \begin{bmatrix} h_{l-1} \\ 0 \end{bmatrix} =: \Sigma_l \cdot \bar{W}_l^\top \cdot h_{l-1}
 \end{aligned}$$

where $\Sigma_l \in \mathbb{R}^{n_l \times n_l}$ is a diagonal matrix with its entries being the derivatives of the activation function with respect to the input, i.e., $\dot{\phi}_l$ containing $\dot{\sigma}(\dots)$ 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} we indicate that the last row is cut off due to the multiplication by zero and $\tilde{\phi}$ shows 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

$$\text{D}_{W_l} f(\mathbf{W}, s)[H_l] = \Sigma_L \bar{W}_L^\top \cdot \Sigma_{L-1} \bar{W}_{L-1}^\top \cdots \Sigma_{l+1} \bar{W}_{l+1}^\top \cdot \Sigma_l H_l^\top \tilde{\phi}_{l-1}.$$

To shorten this expression let us construct a sequence of matrices for all $l = L-1, \dots, 1$ as

$$\Psi_l := \Sigma_l \bar{W}_{l+1}^\top \Psi_{l+1} \in \mathbb{R}^{n_l \times n_L},$$

with $\Psi_L \equiv 1$ due to the activation function in the last layer being the identity function. Now we can write compactly

$$\text{D}_{W_l} f(\mathbf{W}, s)[H_l] = \Psi_l^\top H_l^\top \phi_{l-1}.$$

To arrive at the expression shown in the paper 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, \dots, A_m the m columns of A , one can show by straightforward computation the identity

$$A^\top \cdot b = \begin{bmatrix} A_1^\top b \\ \vdots \\ A_m^\top b \end{bmatrix} = \begin{bmatrix} b^\top A_1 \\ \vdots \\ b^\top A_m \end{bmatrix} = \begin{bmatrix} b^\top & & \\ & \ddots & \\ & & b^\top \end{bmatrix} \begin{bmatrix} A_1 \\ \vdots \\ A_m \end{bmatrix} = (I_{m \times m} \otimes b^\top) \cdot \text{vec}(A).$$

By setting $A = H_l$ and $b = \phi_{l-1}$, we get

$$D_{W_l} f(\mathbf{W}, s)[H_l] = \Psi_l^\top (I_{n_l} \otimes \phi_{l-1}^\top) \text{vec}(H_l).$$

Finally, we can combine the expressions for all layers and produce the full differential map with respect to all parameters

$$D_{\mathbf{W}} f(\mathbf{W}, s)[\mathbf{H}] = \underbrace{\begin{bmatrix} \Psi_1^\top (I_{n_1} \otimes \phi_0) & \dots & \Psi_L^\top (I_{n_L} \otimes \phi_{L-1}) \end{bmatrix}}_{\in \mathbb{R}^{n_L \times N_{net}}} \cdot \underbrace{\begin{bmatrix} \text{vec}(H_1) \\ \vdots \\ \text{vec}(H_L) \end{bmatrix}}_{\in \mathbb{R}^{N_{net} \times 1}},$$

where the MLP input ϕ_0 is just the input s . For the application in our work we always have $n_L = 1$ because the value function maps to a scalar value. Using all N inputs at once we arrive at the expression $G(\mathbf{W}) \in \mathbb{R}^{N \cdot n_L \times N_{net}}$ as shown in Equation (24)

$$D_{\mathbf{W}} F(\mathbf{W})[\mathbf{H}] = \underbrace{\begin{bmatrix} \Psi_1^\top (I_{n_1} \otimes \phi_0^{(1)\top}) & \dots & \Psi_L^\top (I_{n_L} \otimes \phi_{L-1}^{(1)\top}) \\ \vdots & \ddots & \vdots \\ \Psi_1^\top (I_{n_1} \otimes \phi_0^{(N)\top}) & \dots & \Psi_L^\top (I_{n_L} \otimes \phi_{L-1}^{(N)\top}) \end{bmatrix}}_{=: G(\mathbf{W}) \in \mathbb{R}^{N \cdot n_L \times N_{net}}} \cdot \begin{bmatrix} \text{vec}(H_1) \\ \vdots \\ \text{vec}(H_L) \end{bmatrix}.$$

The superscript $(\cdot)^{(i)}$ indicates that the layer outputs ϕ_l arise from the i -th state in the input layer.

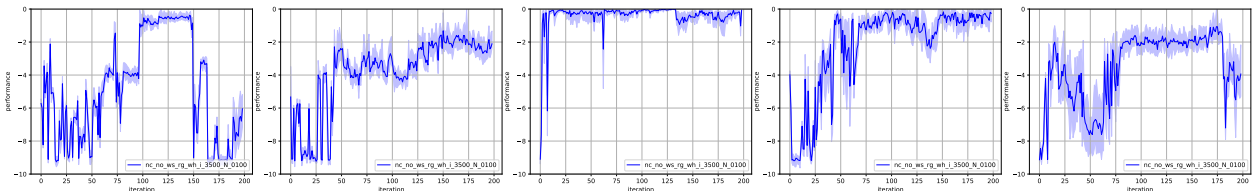
Definition of $\tilde{G}(\mathbf{W})$ Equation (35) originates directly from the difference of $G(\mathbf{W})$ and $G'(\mathbf{W})$. We have

$$\begin{aligned} \tilde{G}(\mathbf{W}) &= G(\mathbf{W}) - \gamma G'(\mathbf{W}) \\ &= \begin{bmatrix} \Psi_1^\top (I_{n_1} \otimes \phi_0^{(1)\top}) & \Psi_L^\top (I_{n_L} \otimes \phi_{L-1}^{(1)\top}) \\ \Psi_1^\top (I_{n_1} \otimes \phi_0^{(N)\top}) & \Psi_L^\top (I_{n_L} \otimes \phi_{L-1}^{(N)\top}) \end{bmatrix} - \gamma \begin{bmatrix} \Psi_1'^\top (I_{n_1} \otimes \phi_0'^{(1)\top}) & \Psi_L'^\top (I_{n_L} \otimes \phi_{L-1}'^{(1)\top}) \\ \Psi_1'^\top (I_{n_1} \otimes \phi_0'^{(N)\top}) & \Psi_L'^\top (I_{n_L} \otimes \phi_{L-1}'^{(N)\top}) \end{bmatrix} \\ &= \begin{bmatrix} \Psi_1^\top (I_{n_1} \otimes \phi_0^{(1)\top}) - \gamma \Psi_1'^\top (I_{n_1} \otimes \phi_0'^{(1)\top}) & \dots & \Psi_L^\top (I_{n_L} \otimes \phi_{L-1}^{(1)\top}) - \gamma \Psi_L'^\top (I_{n_L} \otimes \phi_{L-1}'^{(1)\top}) \\ \vdots & \ddots & \vdots \\ \Psi_1^\top (I_{n_1} \otimes \phi_0^{(N)\top}) - \gamma \Psi_1'^\top (I_{n_1} \otimes \phi_0'^{(N)\top}) & \dots & \Psi_L^\top (I_{n_L} \otimes \phi_{L-1}^{(N)\top}) - \gamma \Psi_L'^\top (I_{n_L} \otimes \phi_{L-1}'^{(N)\top}) \end{bmatrix} \end{aligned}$$

by pairing each block in the matrices.

B Raw Data

To emphasize the difficulty in visualising the results of our Policy Iteration experiment, we provide here the raw output of all five individual repetitions for $N = 100$ samples with $i = 3500$ Policy Evaluations steps when using the persistent setting.



All repetitions of the roll-outs per run produce reliably similar discounted returns, but running again the whole experiment produces varying performance curves. Unfortunately, these curves can be rather dissimilar. Thus, by averaging these curves, we can highlight the trend for the performance over time in the overall experiment and produce the plots shown in Figure 7.