

Reinforcement Learning with Interacting Continually Running Fully Recurrent Networks

In Proc. INNC International Neural Network Conference, Paris, volume 2, pages 817-820, 1990.

Jürgen Schmidhuber*
Institut für Informatik
Technische Universität München
Arcisstr. 21, 8000 München 2, Germany
schmidhu@tumult.informatik.tu-muenchen.de

Abstract

We describe an on-line learning algorithm for attacking the fundamental credit assignment problem in non-stationary reactive environments. Reinforcement and pain are considered as special types of input to an agent living in the environment. The agent's only goal is to maximize cumulative reinforcement and to minimize cumulative pain. This simple goal may require to produce complicated action sequences. Supervised learning techniques for recurrent networks serve to construct a differentiable model of the environmental dynamics which includes a model of future reinforcement. While this model is adapted, it is concurrently used for learning goal directed behavior. The method extends work done by Munro, Robinson and Fallside, Werbos, Widrow, and Jordan.

Introduction

Consider an agent whose movements are controlled by the output units of a neural network, called the control network, which also receives the agent's sensory perception by means of its input units. The agent potentially is able to produce actions that may change the environmental input (external feedback caused by the 'reactive' environment). By means of recurrent connections in the network the agent is also potentially able to internally represent past events (internal feedback).

The agent sometimes experiences different types of negative reinforcement or 'pain' by means of so-called *reinforcement units* or *pain units* that become activated in moments of 'pain' (e.g. the experience of bumping against an obstacle with an extremity). The agent's only goal is to minimize cumulative pain. The agent is autonomous in the sense that no intelligent external teacher is required to provide additional goals or subgoals for it.

A pain unit is treated as a special type of input unit which possesses conventional outgoing connections to other units. Unlike normal input units pain units can have desired activation values at every time. For the purpose of this paper we say that the desirable activation of a pain unit is zero for all times. In the sequel we assume a discrete time environment with 'time ticks'. At a given time the quantity to be minimized is $\sum_{t,i} y_i(t)$ where $y_i(t)$ is the activation of the i th pain unit at time t , and t ranges over all remaining time ticks still to come.

Pain corresponds to negative reinforcement. The reinforcement learning agent faces a very general spatio-temporal credit assignment task: No external teacher provides knowledge about e.g. desired outputs or 'episode boundaries'. In this paper we demonstrate how the agent can employ a combination of two recurrent *self-supervised* learning networks in order to satisfy its goal.

As Munro [3] has pointed out in the case of stationary environments and feedforward networks, one does not necessarily have to employ a 'pure' reinforcement learning algorithm for reinforcement

*This work was supported by a scholarship from SIEMENS AG

learning. ('Pure' reinforcement learning algorithms (or *reinforcement comparison* algorithms) for temporal credit assignment in non-stationary environments have been described in [1], [11], [7] and [8].) A *supervised* learning algorithm can be applied to build a model of the relationships between environmental inputs, output actions of the agent, and corresponding reinforcement. An adaptive model network representing the model can be used to propagate gradient information back into the control network in order to maximize reinforcement.

Robinson and Fallside described an extension of Munro's static approach to dynamic recurrent networks in time-varying environments [6]. (Nguyen and Widrow [4], Jordan [2], and Werbos [10] also use model networks for constructing a mapping from output actions of a control network to their effects in in 'task space' [2]. The same principle as used in Munro's work serves to provide error signals for the control network, in order to improve performance on a given control task.)

As in Munro's approach, the only aspect of the external world which is explicitly described by Robinson and Fallside's recurrent model network is the reinforcement's dependency on past inputs and outputs. There is no model for the dependency of (non-reinforcement) inputs on past outputs (or on past inputs which again may have been caused by past outputs). This makes the model for the reinforcement itself incomplete: Paths for credit assignment leading 'through the environment' can not be considered.

The system described in the next section (see also [9]) employs an adaptive model of the environmental dynamics for computing gradients of the control network's pain. Both the control network and the model network are fully recurrent.

Unlike Robinson and Fallside's approach our approach includes credit assignment passes that lead from pain units back to output units back to all input units and so on. There are also credit assignment paths that lead from input units back to the input units themselves, and from there to the output units. The latter paths are important in the common case when the environment can change even if there are no recent output actions.

The On-Line Algorithm

The discrete time algorithm below *concurrently* adjusts the fully recurrent model network and the fully recurrent control network. Williams and Zipser's on-line version [12] of Robinson and Fallside's Infinite-Input-Duration learning algorithm for fully recurrent networks [5] is used for training both networks. The algorithm is a particular instantiation of a more general form and is based on the logistic activation function for all non-input units. Notation (the reader may find it convenient to compare with [12]):

C is the set of all units of the control network, A is the set of its output units, I is the set of its 'normal' input units, P is the set of its pain units, M is the set of all units of the model network, O is the set of its output units, $O_P \subset O$ is the set of all units that predict pain, $H = M \cup C \setminus (I \cup P)$, W_M is the set of variables for the weights of the model network, W_C is the set of variables for the weights of the control network, $y_{k_{new}}$ is the variable for the updated activation of the k th unit from $M \cup C$, $y_{k_{old}}$ is the variable for the last value of $y_{k_{new}}$, w_{ij} is the variable for the weight of the directed connection from unit j to unit i , $p_{ij_{new}}^k$ is the variable which gives the current (approximated) value of $\frac{\partial y_{k_{new}}}{\partial w_{ij}}$, $p_{ij_{old}}^k$ is the variable which gives the last value of $p_{ij_{new}}^k$, α_C is a positive constant, the learning rate for the control network, α_M is a positive constant, the learning rate for the model network.

$|I \cup P| = |O|$, $|O_P| = |P|$. For each $k \in O \setminus O_P$ there is exactly one $i \in I$ such that $y_{k_{new}}$ predicts the value of $y_{i_{new}}$, which also is called $x_{k_{new}}$. For each $k \in O_P$ there is exactly one $i \in P$ such that $y_{k_{new}}$ predicts the value of $y_{i_{new}}$, which also is called $x_{k_{new}}$. Each unit from $I \cup P \cup A$ has one forward connection to each unit from H . Each unit from M is connected to each other unit from M . Each unit from $C \setminus (I \cup P)$ is connected to each other unit from this set. Each weight of a connection leading to a unit in M is said to belong to W_M . Each weight of a connection leading to a unit in $C \setminus (I \cup P)$ is said to belong to W_C . Each weight $w_{ij} \in W_M$ needs p_{ij}^k -values for all $k \in M$. Each weight $w_{ij} \in W_C$ needs p_{ij}^k -values for all $k \in H$.

First we will describe the algorithm, then some comments will be given.

INITIALIZATION:

For all $w_{ij} \in W_M \cup W_C$: begin $w_{ij} \leftarrow$ random, for all possible k : $p_{ij_{old}}^k \leftarrow 0, p_{ij_{new}}^k \leftarrow 0$ end,
for all $k \in H$: $y_{k_{old}} \leftarrow 0, y_{k_{new}} \leftarrow 0$.

For all $k \in I \cup P$: Set $y_{k_{old}}$ by environmental perception, $y_{k_{new}} \leftarrow 0$.

FOREVER REPEAT:

1. A. For all $i \in H$: $y_{i_{new}} \leftarrow \frac{1}{1+e^{-\sum_j w_{ij} y_{j_{old}}}}$,
for all $i \in I \cup P$: Set $y_{i_{new}}$ by environmental perception.
B. Execute all motoric actions based on activations of units in A.
2. A. For all $w_{ij} \in W_M, k \in M$: $p_{ij_{new}}^k \leftarrow y_{k_{new}}(1 - y_{k_{new}})(\sum_{l \in M} w_{kl} p_{ij_{old}}^l + \delta_{ik} y_{j_{old}})$
B. For all $w_{ij} \in W_M$: $w_{ij} \leftarrow w_{ij} + \alpha_M \sum_{k \in O} (x_{k_{new}} - y_{k_{new}}) p_{ij_{new}}^k$.
3. A. For all $k \in O$ begin $y_{k_{new}} \leftarrow x_{k_{new}}$, for all $w_{ij} \in W_M$: $p_{ij_{new}}^k \leftarrow 0$ end.
B. For all $w_{ij} \in W_C, k \in H$: $p_{ij_{new}}^k \leftarrow y_{k_{new}}(1 - y_{k_{new}})(\sum_{l \in H, w_{kl} \text{ exists}} w_{kl} p_{ij_{old}}^l + \delta_{ik} y_{j_{old}})$
C. For all $w_{ij} \in W_C$: $w_{ij} \leftarrow w_{ij} - \alpha_C \sum_{k \in O_P} p_{ij_{new}}^k$.
4. For all $k \in M \cup C$: $y_{k_{old}} \leftarrow y_{k_{new}}$,
For all $w_{ij} \in W_M \cup W_C$ and for all possible k : $p_{ij_{old}}^k \leftarrow p_{ij_{new}}^k$.

General comments on the algorithm. 1. In step 2 the model network is updated in order to better predict the input (including pain) for the controller. Since the control network continues activation spreading based on the actual inputs instead of using the predictions of the model network, ‘teacher forcing’ [12] is used in the model network (step 3.A).

2. In step 3 the weights of the control network are updated in order to minimize the cumulative activations of the pain units. In the version above no teacher forcing is used for the control network. Here the philosophy is that a little pain may be informative for the agent, and may have an explicit influence on future actions.

3. The algorithm assumes that from one time tick to the next the environment changes in a fashion that is predictable by linearly separable mappings from past states. If there is a ‘higher degree of environmental non-linearity’ then the algorithm has to be modified in a trivial manner such that the involved networks tick at a higher frequency than the environment. In any case it suffices if there are four network ticks for each environmental tick. This is due to the fact that 4-layer-operations in principle are enough to arbitrarily approximate any desired mapping.

Comments on the on-line nature of the algorithm. Since we want an on-line learning procedure we deviate from true gradient descent in several respects:

1. Instead of accumulating contributions to weight changes over time and actually changing the weights after activation spreading, the weights are changed immediately. Immediate weight changes allow to renounce on information about ‘episode boundaries’ [12].

2. The weight changing mechanism of the controller acts as if the model network already was a perfect predictor (with fixed weights) which could replace the environment. However, the model may be imperfect:

2A. Jordan [2] as well as Robinson and Fallside [6] note that a model network does not need to be perfect to allow increasing performance of a control network. If the error for the control network is not given by the difference of the desired input for the control network and the model output but by the difference of the desired input and the actual input of the control network, then the minima of this difference still are fixpoints of the weight changing mechanism, as long as the model network already has reached a local minimum. The zero-points of the controller’s error are fixpoints even if the model network has not yet found a local minimum. The minima of the error for the control network can be found if the inner products of the approximated gradients for the control network’s weights and the exact gradients (according to a perfect model) tend to be positive.

2B. Note that the p_{ij}^k ’s of the model network change independently from the p_{ij}^k ’s of the control network. A situation where the control network experiences pain and where its weights are based on an inaccurate model will not remain stable, as long as not both the model network and the control network are trapped in local minima. If we assume that the model network always finds a

zero-point of its error function (which means that it sooner or later always will correctly predict future inputs no matter how the controller behaves), then over time we can expect the control network to perform gradient descent in pain according to a perfect model of the visible parts of the real world. As long as the model is inaccurate the controller partly functions as a random explorer who rather uninformedly causes situations that help the model network to collect new data about the environmental dynamics, in order to ‘make the relevant dynamics of the world differentiable’.

Experiments with a difficult control task. The algorithm is currently being tested on a complicated pole balancing problem (the differential equations modelling the cart-pole system described in [1] are employed). Unlike with previous pole balancing tasks no prewired decoder is used to pre-process the inputs from the environment. Additionally, unlike with previous pole balancing tasks no information is provided about temporal derivatives of the environment’s state variables (pole velocity, etc.). The agent is forced to extract this kind of information by itself, by means of the recurrent connections of its model network. An additional difficulty is that no external teacher provides information about ‘trial boundaries’. Thus the agent faces a complex and realistic spatio-temporal credit assignment task. The results of preliminary test runs are very encouraging, however, the experiments have not yet been completed.

Extensions to the system. In [9] it is discussed how the two recurrent networks can be used for planning future action sequences by performing gradient descent in predicted pain instead of actual pain. It is also hinted at the possibility of using probabilistic output units for the controller, thus providing the agent with explicit explorative capabilities. Furthermore it is noted that a perfect model which also predicts the controller’s output can be used for ‘meta-learning’ (‘learning how to learn’).

Concluding Remarks

The weights of a network with fixed topology may be considered as its program. One of the most interesting aspects of many connectionist algorithms is that program outputs are differentiable with respect to programs. A simple program generator (the gradient descent procedure) allows to produce increasingly successful programs, if the desired outputs are known.

In typical reinforcement learning situations the environment is not *a priori* represented in a differentiable form. So the main reason for connectionist world models in the style above can be seen in ‘making the world differentiable’. Thus even *program inputs* can become differentiable with respect to programs. A differentiable world model allows the program generator an informed search for better goal directed programs.

The degree of informedness of this search for suitable programs is a main difference between the very general approach presented in this paper and other reinforcement learning algorithms.

References

- [1] 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, 834-846, 1983.
- [2] M.I. Jordan. Supervised learning and systems with excess degrees of freedom. Technical Report COINS TR 88-27, Massachusetts Institute of Technology, 1988.
- [3] P.W. Munro. A dual back-propagation scheme for scalar reinforcement learning. *Proceedings of Ninth Annual Conference of the Cognitive Science Society, Seattle, WA*, 1987.
- [4] Nguyen and B. Widrow. The truck backer-upper: An example of self learning in neural networks. In *IJCNN International Joint Conference on Neural Networks, Vol 2*, 1989.
- [5] A. J. Robinson and F. Fallside. Static and dynamic error propagation networks with application to speech coding. *Proceedings of Neural Information Processing Systems, American Institute of Physics*, 1987.
- [6] T. Robinson and F. Fallside. Dynamic reinforcement driven error propagation networks with application to game playing. In *Proceedings of the 11th Conference of the Cognitive Science*

Society, Ann Arbor, 1989.

[7] J. H. Schmidhuber. The neural bucket brigade. In R. Pfeifer, Z. Schreter, Z. Fogelman, and L. Steels, editors, *Connectionism in Perspective*, Amsterdam: Elsevier, 1988.

[8] J. H. Schmidhuber. Temporal-Difference-Driven Learning in Recurrent Networks. In *ICNC International Conference on Parallel Processing in Neural Systems and Computers, Düsseldorf*, 1990.

[9] J. H. Schmidhuber. Making the world differentiable: On using supervised learning recurrent neural networks for dynamic reinforcement learning and planning in non-stationary environments. FKI-Report, Institut für Informatik, Technische Universität München, 1990.

[10] P. J. Werbos. Building and understanding adaptive systems: A statistical/numerical approach to factory automation and brain research. *IEEE Transactions on Systems, Man, and Cybernetics*, 17, 1987.

[11] R. J. Williams. Toward a theory of reinforcement-learning connectionist systems. Technical Report NU-CCS-88-3, College of Comp. Sci., Northeastern University, Boston, MA, 1988.

[12] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent networks. Technical Report ICS Report 8805, Univ. of California, San Diego, La Jolla, 1988.