In J. A. Meyer and S. W. Wilson, editors, Proc. of the International Conference on Simulation of Adaptive Behavior: From Animals to Animats, pages 222-227. MIT Press/Bradford Books, 1991.

A Possibility for Implementing Curiosity and Boredom in Model-Building Neural Controllers

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

Abstract

This paper introduces a framework for 'curious neural controllers' which employ an adaptive world model for goal directed on-line learning.

First an on-line reinforcement learning algorithm for autonomous 'animats' is described. The algorithm is based on two fully recurrent 'self-supervised' continually running networks which learn in parallel. One of the networks learns to represent a complete model of the environmental dynamics and is called the 'model network'. It provides complete 'credit assignment paths' into the past for the second network which controls the animats physical actions in a possibly reactive environment. The animats goal is to maximize cumulative reinforcement and minimize cumulative 'pain'.

The algorithm has properties which allow to implement something like *the desire to improve the model network's knowledge about the world.* This is related to *curiosity.* It is described how the particular algorithm (as well as similar model-building algorithms) may be augmented by dynamic *curiosity* and *boredom* in a natural manner. This may be done by introducing (delayed) reinforcement for actions that increase the model network's knowledge about the world. This in turn requires the model network *to model its own ignorance*, thus showing a rudimentary form of *self-introspective* behavior.

1. Introduction

In the sequel first an on-line algorithm for reinforcement learning in non-stationary reactive environments is described. The algorithm heavily relies on an *adaptive model of the environmental dynamics.* The main contribution of this paper (see the second section) is to demonstrate how the algorithm may be naturally augmented by *curiosity* and *boredom*, in order to improve the world model in an on-line manner.

Consider an 'animat' whose movements are controlled by the output units of a neural network, called the control network, which also receives the animat's sensory perception by means of its input units. The animat 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 animat is also potentially able to internally represent past events (internal feedback).

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

Reinforcement units and pain units are similar to other input units in the sense that they possess conventional outgoing connections to other units. However, unlike normal input units they can have *desired activation values* at every time. For the purpose of this paper we say that the desired activation of a pain unit is zero for all times, other reinforcement units may have positive desired values. In the sequel we assume a discrete time environment with 'time ticks'. At a given time the quantity to be minimized by the learning algorithm is $\sum_{t,i}(c_i - y_i(t))^2$ where $y_i(t)$ is the activation of the ith pain or reinforcement unit at time t, t ranges over all remaining time ticks still to come, and c_i is the desired activation of the ith reinforcement or pain unit for all times.

The reinforcement learning animat faces a very general spatio-temporal credit assignment task: No external teacher provides knowledge about e.g. desired outputs or 'episode boundaries' (externally defined temporal boundaries of training intervals). In the sequel it is demonstrated how the animat may employ a combination of two recurrent *self-supervised* learning networks in order to satisfy its goal.

Munro [2], Jordan [1], Werbos [12], Robinson and Fallside [6], and Nguyen and Widrow [4] used *'model networks'* for constructing a mapping from output actions

*This work was supported by a scholarship from SIEMENS AG

of a control network to their effects in in 'task space' [1]. The general system described below (an improved version of the variants described in [8] and [10]) also employs an adaptive model of the environmental dynamics for computing gradients of the control network's inputs with respect to the controller weights. The model network is trained to simulate the environment by making predictions about future inputs, including pain and reinforcement inputs. Training of the controller works as follows: Since we cannot propagate input errors (e.g. differences between actual pain signals and desired zero pain signals) 'through the environment', we propagate them through the combination of model network and controller. Only the controller weights change during this phase, the weights of the model network have to remain fixed. (Actually, we do not really propagate errors through the networks. We use an algorithm which is functionally equivalent to 'back-propagation through time' but uses only computations 'going forward in time'. So there is no need for storing past activations.)

Both the control network and the model network are fully recurrent. The algorithm differs from algorithms by other authors in at least some of the following issues: It aims at on-line learning and *locality in time*, it does not care for 'epoch-boundaries', it needs only reinforcement information for learning, it allows different kinds of reinforcement (or pain), and it allows both internal and external feedback with theoretically arbitrary time lags. Unlike with Robinson and Fallside's approach (which is the one that bears the most relationships to the one described here) 'credit assignment paths' are provided 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 were no recent output actions.

The discrete time algorithm below *concurrently* adjusts the fully recurrent model network and the fully recurrent control network. An on-line version of Robinson and Fallside's Infinite-Input-Duration learning algorithm for fully recurrent networks [5] (first implemented by Williams and Zipser [14]) is used for training both the model network and the combination of controller and model network. The algorithm is a particular instantiation of a more general form and is based on the logistic activation function for all non-input units.

In step 1 of the main loop of the algorithm actions in the external world are computed. Due to the internal feedback, these actions are based on previous inputs and outputs. For all new activations, the corresponding derivatives with respect to all controller weights are updated.

In step 2 actions are executed in the external world, and the effects of the current action and/or previous actions may become visible.

In step 3 the model network tries to predict these effects without seeing the new input. Again the relevant gradient information is computed.

In step 4 the model network is updated in order to better predict the input (including pain) for the controller. Finally, the weights of the control network are updated in order to minimize the cumulative differences between desired and actual activations of the pain and reinforcement units. Since the control network continues activation spreading based on the actual inputs instead of using the predictions of the model network, 'teacher forcing' [14] is used in the model network.

One can find various improvements of the systems described in [8] and [10]. For instance, the partial derivatives of the controller's inputs with respect to the controller's weights are approximated by the partial derivatives of the corresponding predictions generated by the model network. Furthermore, the model sees the last input and current output of the controller at the same time.

Notation (the reader may find it convenient to compare with [14]):

C is the set of all non-input 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 and reinforcement 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 or reinforcement, 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 kth unit from $M \cup C \cup I \cup P$, $y_{k_{old}}$ 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, $p^k_{ij_{new}}$ is the variable which gives the current (approximated) value of $\frac{\partial y_{k_{new}}}{\partial w_{ij}}$, $p^k_{ij_{old}}$ is the variable which gives the last value of $p^k_{ij_{new}}$, if $k \in P$ then c_k is k's desired activation for all times, α_C is the learning rate for the control network, α_M is the learning rate for the model network.

$| I \cup P | = | O |$, $| O_P | = | P |$. If $k \in I \cup P$, then kpred is the unit from O which predicts k. Each unit from $I \cup P \cup A$ has one forward connection to each unit from $M \cup C$. Each unit from M is connected to each other unit from M. Each unit from C is connected to each other unit from C. 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 is said to belong to W_C. Each weight $w_{ij} \in W_M$ needs p^k_{ij}-values for all $k \in M$. Each weight $w_{ij} \in W_C$ needs p^k_{ij}-values for all $k \in M \cup C \cup I \cup P$.

INITIALIZATION:
For all $w_{ij} \in W_M \cup W_C$*:*
begin $w_{ij} \leftarrow$ *random,*
for all possible k*:* $p^k_{ij_{old}} \leftarrow 0, p^k_{ij_{new}} \leftarrow 0$ *end.*
For all $k \in M \cup C : y_{k_{old}} \leftarrow 0, y_{k_{new}} \leftarrow 0.$
For all $k \in I \cup P$ *:*
Set $y_{k_{old}}$ *by environmental perception,* $y_{k_{new}} \leftarrow 0.$

FOREVER REPEAT:
1. *For all* $i \in C : y_{i_{new}} \leftarrow \frac{1}{1+e^{-\sum_j w_{ij} y_{j_{old}}}}.$
   For all $w_{ij} \in W_C, k \in C$*:*
   $p^k_{ij_{new}} \leftarrow y_{k_{new}}(1 - y_{k_{new}})(\sum_l w_{kl} p^l_{ij_{old}} + \delta_{ik} y_{j_{old}})$
   For all $k \in C$*:*
   begin $y_{k_{old}} \leftarrow y_{k_{new}}$*,*
   for all $w_{ij} \in W_C$ *:* $p^k_{ij_{old}} \leftarrow p^k_{ij_{new}}$ *end .*
2. *Execute all motoric actions based on activations of units in A. Update the environment.*
   For all $i \in I \cup P$*:*
   Set $y_{i_{new}}$ *by environmental perception.*
3. *For all* $i \in M : y_{i_{new}} \leftarrow \frac{1}{1+e^{-\sum_j w_{ij} y_{j_{old}}}}.$
   For all $w_{ij} \in W_M \cup W_C, k \in M$*:*
   $p^k_{ij_{new}} \leftarrow y_{k_{new}}(1 - y_{k_{new}})(\sum_l w_{kl} p^l_{ij_{old}} + \delta_{ik} y_{j_{old}}).$
   For all $k \in M$*:*
   begin $y_{k_{old}} \leftarrow y_{k_{new}}$*,*
   for all $w_{ij} \in W_C \cup W_M$ *:* $p^k_{ij_{old}} \leftarrow p^k_{ij_{new}}$ *end.*
4. *For all* $w_{ij} \in W_M$*:*
   $w_{ij} \leftarrow w_{ij} + \alpha_M \sum_{k \in I \cup P}(y_{k_{new}} - y_{kpred_{old}}) p^{kpred}_{ij_{old}}.$
   For all $w_{ij} \in W_C$*:*
   $w_{ij} \leftarrow w_{ij} + \alpha_C \sum_{k \in P}(c_k - y_{k_{new}}) p^{kpred}_{ij_{old}}.$
   For all $k \in I \cup P$*:*
   begin $y_{k_{old}} \leftarrow y_{k_{new}}$*,* $y_{kpred_{old}} \leftarrow y_{k_{new}}$*,*
   for all $w_{ij} \in W_M : p^{kpred}_{ij_{old}} \leftarrow 0,$
   for all $w_{ij} \in W_C : p^k_{ij_{old}} \leftarrow p^{kpred}_{ij_{old}}$ *end.*

By employing probabilistic output units for C and by using 'gradient descent through random number generators' [13] we can introduce explicit explorative random search capabilities into the otherwise deterministic algorithm. In the context of the IID algorithm, this works as follows: A probabilistic output unit k consists of a conventional unit $k\mu$ which acts as a mean generator and a conventional unit $k\sigma$ which acts as a variance generator. At a given time, the probabilistic output $y_{k_{new}}$ is computed by

$$y_{k_{new}} = y_{k\mu_{new}} + z y_{k\sigma_{new}},$$

where z is distributed e.g. according to the normal distribution. The corresponding $p^k_{ij_{new}}$ have to be updated according to the following rule:

$$p^k_{ij_{new}} \leftarrow p^{k\mu}_{ij_{new}} + \frac{y_{k_{new}} - y_{k\mu_{new}}}{y_{k\sigma_{new}}} p^{k\sigma}_{ij_{new}}.$$

By performing more than one iteration of step 1 and step 3 at each time tick, one can adjust the algorithm to environments that change in a manner which is not predictable by semilinear operations (theoretically three additional iterations are sufficient for any environment).

The algorithm is local in time, but not in space. See [8] for a justification of certain deviations from 'pure gradient descent through time', and for a description of how the algorithm can be used for planning action sequences. See [7] and [9] for two quite different entirely local methods.

2. Implementing Dynamic Curiosity and Boredom

Only if the model network is a good predictor of the environmental dynamics we can expect the controller to converge. In the current section we motivate the introduction of the 'explicit desire to improve the world model' and show a possibility for implementing it in on-line model-building systems as the one described in the last section.

Many biological learning systems, particularily the more complex ones, show an interplay of goal-directed learning and explorative learning. In addition to certain permanent goals (like avoiding pain), goals are generated whose immediate purpose solely is to increase knowledge about the world. So far this interplay has not been addressed at all in the connectionist literature.

The explorative side of learning (related to something that usually is called *curiosity*) is not completely unsupervised, as it is sometimes assumed. Curiosity helps to learn how the world works, which in turn helps to satisfy certain goals. However, the goal-directedness of curiosity is less obvious than the goal-directedness of the algorithm described above (and of less general algorithms described in other papers on goal-directed learning).

Curiosity is related to what one already knows about the world. One gets curious as soon as one *believes that there is something that one does not know.* However, the goal of learning how the world works is dominated by other goals (like avoiding pain): One does not know *exactly* how it feels putting one's hand into the meat grinder. However, one does not *want* to know.

Since curiosity makes sense only for systems that can have *dynamic influence on what they learn*, and since curiosity aims at minimizing a dynamically changing value, namely, the degree of ignorance about something, it makes sense only in on-line learning situations where there is some sort of *dynamic attention.*

Thus the precondition of curiosity is something like our on-line learning algorithm described above. This algorithm builds a world model in order to use the world model for goal-directed learning of the controller. The controller's potential for *dynamic attention* is given by

the external feedback. The world model adapts itself to whatever the controller focusses on (see [11] for an application of similar adaptive control techniques to the problem of learning selective attention). The direct goal of curiosity and boredom is to improve the world model. The indirect goal is to ease the learning of new goal-directed action sequences. The contribution of this section is to show one possibility for augmenting the algorithm by curiosity and by its counterpart, which is *boredom*.

The basic idea is simple: We introduce an additional reinforcement unit for the controller (see figure 1.). This unit, hereafter called the *curiosity unit*, gets activated by a process which at every time step measures the Euclidian distance between reality and prediction of the model network. The activation of the curiosity unit is a function of this distance. Its desired value is a positive number corresponding to the *ideal mismatch* between belief and reality. The effect of the algorithm described in the first section is that there is *positive reinforcement* whenever the model network *fails* to correctly predict the environment. Thus the usual credit assignment process for the controller encourages certain past actions in order to repeat situations similar to the mismatch situation.

As soon as the model network has learned to correctly predict the environment in former 'mismatch situations', actions leading to such situations automatically are de-reinforced. This is because the activation of the curiosity unit goes back to zero. *Boredom* becomes associated with the corresponding situations.

The important point is: The same complex mechanism which is used for 'normal' goal-directed learning is used for implementing curiosity and boredom. There is no need *for devising a* separate *system which aims at improving the world model.*

The controller's credit assignment process is aimed at repeatedly entering situations where the model network's performance is not optimal. *It is important to observe that this process itself makes use of the model network!* The model network has to predict the activations of the curiosity unit. Thus the model network partly has to model its own ignorance, it has *to learn to know that it does not know* certain details.

What is the *ideal mismatch* mentioned above? In conventional AI the saying goes that a system can not learn something that it does not already *almost know*. If we want to adopt this view, then a consequence is that the function that translates mismatches into reinforcement is not a linear one. *Zero reinforcement should be given in case of perfect matches, high reinforcement should be given in case of 'near-misses', and low reinforcement again should be given in case of strong mismatches.* This corresponds to a notion from 'esthetic information theory' which tries to explain the feeling of 'beauty' by means of the quotient of 'subjective complexity' and 'subjective order' or the quotient of 'unfamiliarity' and 'familiarity' (measured in an information-theoretic manner). This quotient should achieve a certain ideal value. (See Nake [3] for an overview of approaches to formalizing 'esthetic information'. Interestingly, the number $\frac{1}{e}$ plays a significant role in at least some of these approaches.) However, at the moment the precise nature of a good mapping between (mis)matches and reinforcement is unclear and subject of ongoing research.

Currently some experimental research is going on in order to answer the following questions: What are useful learning rates (it is assumed that the model network should learn clearly faster than the controller)? What are useful relative strengths of pure goal-directed reinforcement and 'curiosity reinforcement'? And what are the properties of a good mapping from mismatches to reinforcement?

Although these questions are still open, in some preliminary experiments with a *linear* mapping from mismatches to reinforcement it already has been demonstrated that errors of the model network can be reduced by generating curiosity reinforcement in an on-line manner.

Concluding Remarks

The basic idea of implementing curiosity and boredom is not limited to the particular algorithm described in the first section. *Every* model-dependent on-line algorithm for learning goal directed behavior might be augmented by a similar implementation of 'the desire to improve the world model'. The basic motivation is: Instead of using some separate mechanism for improving the world model, we want to make use of the capabilities of the goal-directed learning algorithm itself.

The interesting side effect is: Since the learning algorithm depends on the model network, the model network has to make a prediction about its own current prediction capabilities. The *activations* of the model network are (partly) interpreted as a statement about the current *weights* of the model network. Note that this is already a rudimentary form of *self-introspective* behavior! The author believes that extensions of these rudimentary forms of introspective neural algorithms will be the key to learning systems which are much more sophisticated than the ones we know so far.

Figure 1. An animat controlled by a recurrent control network is shown. For simplicity, only one 'normal' input unit (IN), one 'normal' reinforcement input unit (R), one hidden unit and one output unit (OUT) are depicted. A recurrent model network is trained to emulate the environmental dynamics by predicting the control network's input ($PRED_{IN}$ and $PRED_R$).
An additional reinforcement unit CUR for the control network gets activated by 'ideal mismatches' between expectations of the model network and reality. The model network needs an additional output unit ($PRED_{CUR}$) for predicting CUR. It models its own ignorance, thus showing a rudimentary form of self-introspective behavior. The model network helps to encourage action sequences of the controller which lead to unfamiliar situations.

References

[1] M. I. Jordan. Supervised learning and systems with excess degrees of freedom. Technical Report COINS TR 88-27, Massachusetts Institute of Technology, 1988.

[2] P. W. Munro. A dual back-propagation scheme for scalar reinforcement learning. *Proceedings of the Ninth Annual Conference of the Cognitive Science Society, Seattle, WA*, pages 165–176, 1987.

[3] F. Nake. *Ästhetik als Informationsverarbeitung*. Springer, 1974.

[4] Nguyen and B. Widrow. The truck backer-upper: An example of self learning in neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, pages 357–363. IEEE Press, 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*, pages 836–843, 1989.

[7] J. Schmidhuber. The Neural Bucket Brigade: A local learning algorithm for dynamic feedforward and recurrent networks. *Connection Science*, 1(4):403–412, 1989.

[8] J. Schmidhuber. An on-line algorithm for dynamic reinforcement learning and planning in reactive environments. In *Proc. IEEE/INNS International Joint Conference on Neural Networks, San Diego*, volume 2, pages 253–258, 1990.

[9] J. Schmidhuber. Recurrent networks adjusted by adaptive critics. In *Proc. IEEE/INNS International Joint Conference on Neural Networks, Washington, D. C.*, volume 1, pages 719–722, 1990.

[10] J. Schmidhuber. Reinforcement learning with interacting continually running fully recurrent networks. In *Proc. INNC International Neural Network Conference, Paris*, volume 2, pages 817–820, 1990.

[11] J. Schmidhuber and R. Huber. Learning to generate focus trajectories for attentive vision. Technical Report FKI-128-90, Institut für Informatik, Technische Universität München, 1990.

[12] P. J. Werbos. Backpropagation and neurocontrol: A review and prospectus. In *IEEE/INNS International Joint Conference on Neural Networks, Washington, D.C.*, volume 1, pages 209–216, 1989.

[13] R. J. Williams. On the use of backpropagation in associative reinforcement learning. In *IEEE International Conference on Neural Networks, San Diego*, volume 2, pages 263–270, 1988.

[14] R. J. Williams and D. Zipser. Experimental analysis of the real-time recurrent learning algorithm. *Connection Science*, 1(1):87–111, 1989.