Networks Adjusting Networks

Technical Report FKI-129-90, 1990

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

Revised November 1990

Abstract

This paper describes extensions of previous 'adaptive critics' which have been one-dimensional, acyclic, and suited only for feed-forward controllers. The extensions address the following issues:

1. Feed-forward adaptive critics for fully recurrent probabilistic control nets.
2. Recurrent adaptive critics.
3. Vector-valued adaptive critics based on a system identification component.

Furthermore an idea is described for approximating recurrent back propagation with a 3-network method which is local in time.

In one experiment a linear adaptive critic adjusts a recurrent network such that it solves a non-linear task (a 'delayed XOR'-problem). In another experiment a four-dimensional adaptive critic quickly learns to solve a complicated pole balancing task.

Introduction

Note: This is the revised and expanded version of an earlier report from February 1990.

Reinforcement learning can be applied in cases for which supervised learning is not designed, namely, when there is no well-informed instructive teacher to provide target outputs for each time tick, but only an evaluative teacher who sometimes states whether a system controlled by the network is in a desireable state or not.

A few methods are designed for reinforcement learning with recurrent neural networks, including 'extended REINFORCE' algorithms [20], the 'Neural Bucket Brigade Algorithm' [9], and model-building recurrent neural controllers based on system identification (e.g. [11]). None of these algorithms includes an *adaptive critic* component, although adaptive critics have proven to be very useful in the case of feed-forward networks [3][14][1]. Adaptive critics are based on 'Temporal Difference (TD-) Methods' [15] and are closely related to concepts from *dynamic programming.* TD-methods use differences of successive predictions about future

[*]Research supported by a scholarship from SIEMENS AG

events to generate error signals for the predictor. Thus TD-methods are an example of 'self-supervised' learning. With adaptive critics, the TD-error signals also serve as reinforcement signals for a neural controller.

One of the adaptive critic extensions below may be viewed as an application of TD-methods [15] to the temporal evolution of recurrent networks. Another extension of the adaptive critic principle combines *'gradient descent through a frozen model network'* with a *vector-valued* adaptive critic. (Previous adaptive critics have been *scalar*). The scheme is intended to provide better scaling for multi-dimensional actions and sensory 'pain' or 'pleasure' perceptions (e.g. for robot control).

It must be noted that no theory of adaptive critics exists so far. (The existing theory of TD-methods does not address the on-line learning problems of reinforcement learning systems *based* on TD-methods.) Thus it remains unclear under which conditions on-line algorithms based on the adaptive critic principle actually converge to the desired solutions. However, first steps towards a theory of on-line learning actor/critic systems have been made recently [21].

Locality in Space and Time

An important aspect of the algorithms described in this paper is their applicability to on-line learning tasks. There is no need for storing past activations except for the most recent ones. Yet the credit assignment process during training can in principle bridge arbitrary time lags.

Locality in Space and Time. In this paper a learning algorithm for dynamic neural networks is said to be *local in time* if for given network size (measured in number of connections) during on-line learning the peak computation complexity at every time step is $O(1)$, for *arbitrary* durations of sequences to be learned.

A learning algorithm for dynamic neural networks is said to be *local in space* if during on-line learning for limited durations of learned sequences and for *arbitrary* network sizes (measured in number of connections) and for *arbitrary* network topologies the peak computation complexity per connection at every time step is $O(1)$.

A learning algorithm for dynamic neural networks is said to be *local* if during on-line learning for *arbitrary* durations of sequences to be learned and for *arbitrary* network sizes (measured in number of connections) and *arbitrary* network topologies the peak computation complexity per connection at every time step is $O(1)$.

For example, the 'unfolding in time' method (e.g. [7]) is not local in time. The IID-Algorithm [6] (also called the RTRL-Algorithm in [22]) is local in time but not in space.

The first algorithm to be considered below performs only computations local in space and time. The second algorithm employs a supervised learning algorithm for recurrent networks and is local in time, but not in space. The approach to supervised learning in recurrent networks again is local in space and time.

A TD-algorithm for Reinforcement Learning in Dynamic Recurrent Networks

Here we describe the discrete time version of an algorithm for adjusting a recurrent network in order to let it solve tasks by delayed reinforcement learning (i. e. tasks where an external teacher indicates only once in a while whether the system is in a desireable state or not, *without providing detailed knowledge about the desired outputs at each time tick*).

The algorithm can be viewed as an application of 'Temporal Difference Methods' (TD-methods) [15] to the temporal evolution of recurrent networks. The fully recurrent *control network* consists of linear input units and binary probabilistic non-input units. Its state vector at time t is called $x(t)$. We consider the case where the learning phase is dividable into 'episodes'. An episode starts with the initialization of the system's activations and is finished when the final reinforcement, R, becomes known. Here is a description of the algorithm:

First all weights are randomly initialized with real values.

For all episodes:

In the beginning of each episode, at the first time tick, the activations of input units of the recurrent control network are initialized with values determined by sensory perceptions from the environment, and the activations of hidden and output units are initialized with 0. *For all following time ticks, until there is external reinforcement R (a real number) indicating failure or success :*

At a given time tick t:

1. The critic (a static network with one output) receives as input the complete activation vector $x(t-1)$ of all units of the control network. The dimensionality of the input vector of the critic is therefore equal to the number of units in the control network. Its one-dimensional output, r, is interpreted as a prediction of the final reinforcement to be received in the future [3][14][1]. In the case of a linear critic, its output is given by $r = x^T(t-1)v(t)$, where $v(t)$ is the critic's current weight vector.

2. The control network performs one update-step: Each probabilistic non-input unit i sums its weighted inputs. This sum is passed to the logistic function $l(x) = \frac{1}{1+e^{-x}}$ which gives the probability that the activation $x_i(t)$ becomes 1 *or* 0. *Each unit i also stores its last activation $x_i(t-1)$. Output units may cause an action in the environment, this may lead to new activations for the input units. So besides the internal feedback, there may exist external feedback through the environment.*

3. If there is external reinforcement R (which indicates the end of the current episode) then the variable r' is set equal to R.

Otherwise r' is defined to be a new estimation of final reinforcement, obtained by letting the critic evaluate the new state of the control network. In case of a linear critic $r' = x^T(t)v(t)$.

Using its static learning algorithm (e.g. the generalized delta rule) the critic associates the last activation vector $x(t-1)$ of the recurrent network with r', thus 'transporting expectation back in time' for one time step. So the critic's error is given by $r' - r$. Its weight vector is updated according to the rules of gradient descent; the result is the new weight vector $v(t+1)$.

4. Each directed weight $w_{ij}(t)$ from unit i to unit j of the recurrent network is immediately altered according to $\Delta w_{ij}(t) = \lambda(r' - r)x_i(t-1)x_j(t)$ (with λ being a positive constant), thus encouraging (or discouraging) the last transition.

The differences computed by the critic determine the learning rate for a Hebb-like rule [13]. State transitions from states associated with low expectation of reinforcement leading to states with a higher evaluation are encouraged. State transitions from states associated with high expectation of reinforcement leading to states with a lower evaluation are discouraged. So the learning algorithm implements Samuel's principle for delayed reinforcement, as described in the context of learning to play checkers [8]: "We are attempting to make the score, calculated for the current board position, look like that calculated for the terminal board position of the chain of moves which most probably will occur during actual play."

To use the terminology of continuous time: *The temporal derivative of the expectation of future reinforcement is the actual reinforcement.*

It should be noted that the learning rule of step 4 is only the most simple representative of a set of applicable reinforcement learning rules. For instance, the learning rule may be modified such that unlikely transitions are credited more strongly [1]:

$$\triangle w_{ij}(t) = \lambda(r' - r)x_i(t-1)(x_j(t) - P(x_j = 1 \mid x(t-1), w(t-1))),$$

where $w(t-1)$ is the last weight vector. Another candidate for the learning rule in step 4 is Barto and Anandan's A_{R-P} rule [2].

An Experiment: Delayed XOR with Stationary Inputs

In a simple experiment we wanted a completely recurrent network to solve a delayed XOR task with static inputs. The critic consisted of a single *linear* unit adapted by the delta-rule. For the duration of one episode, two of three input units of the control network were clamped with randomly chosen stationary binary values, the third input unit was set to 1, thus serving as a threshold provider. The task for the control network was to run for a predefined number, k, of time ticks and then to emit the XOR of the two random inputs in a single output unit. The learning scheme depicted above led to successful learning of this task, for various numbers of hidden units and for various time delays. For instance, test runs were conducted with a random weight initialization between -0.1 and 0.1, $k = 3$, 3 hidden units, and with both the learning rate for the critic and λ set equal to 0.2. In some test runs, about 2000 training episodes per pattern led to about 99 percent correct classifications. In these cases, by making a deterministic network out of the stochastic network (i.e. by modifying the activation rule after the learning phase such that always the most likely activations were selected) 100 percent correct classifications were obtained.

Note that a *linear* critic was sufficient to achieve this result, even though the task to be solved was of the 'non-linearly separable' type. This is possible because the task of the critic is somewhat easier than the task of the recurrent non-linear network. The non-linear network has to implement the XOR-mapping, while the critic only has to implement the mapping from network states to future reinforcement. In general the mapping from network states to reinforcement can be a non-linearly separable function by itself. However, if the recurrent network already has learned to produce the correct responses to a subset of the possible input patterns, the critic's task becomes easier. In fact, after a perfect solution has been found, the critic's output becomes trivial: In that case, as long as the recurrent network runs, the critic's output always equals the final reinforcement, which is 1. This final trivial mapping can be implemented with simply a heavy-weighted connection from the unit that is always on to the critic. In fact, this kind of connection was exactly what was observed in some cases after the training phase.

Differences to Anderson's system.

It is worth noting some differences to Anderson's system [1]. In contrast to Anderson we did not use a back-propagation network but a single *linear* unit for the critic. We also did not use a static feed-forward network for computing output actions, but a continually running recurrent network. We also did not use different learning rules for hidden and output units.

Finally, while Anderson's time ticks involve multi-layer activation spreading and back propagation of errors, the system described above performs only one-layer operations within one tick. (Here we view the update of the recurrent network as a one-layer operation.) So the basic operations performed by our system are much simpler than the basic operations of Anderson's. Note that there is a delay of at least 2 time ticks between inputs that have to be transformed in a non-linear fashion, and the corresponding actions. With reactive environments this means that a new input can be available before the response to the last one is computed.

Recurrent Critics

There are tasks where a linear or a static feed-forward critic is not sufficient. But why should not the critic's own output *directly* depend on past states of the recurrent reinforcement learning system? Although the control network is able to memorize information about past states by means of its recurrent links, one should expect advantages by introducing a continually running, self-supervised recurrent critic. We now describe one scenario for such a system consisting of two interacting recurrent networks. The basic principle is similar to the one of the algorithm described in the first section. However, the algorithm aims at maximizing the *cumulative* sum of reinforcement to be received at all future times [14].

There is a continually running recurrent control network with external and internal feedback, and a critic whose task is to predict the cumulative sum of (discounted) reinforcement to be received in the future. However, now the critic itself is a continually running recurrent network whose input at a given time is the complete current state of the control network (including the current environmental input). One of the critic's non-input units is interpreted as the predictive output.

The critic again learns in a 'self-supervised' manner. Its learning algorithm should be local in time, so the on-line version [22] of the IID-Algorithm [6] can be applied. According to TD-methods, the desired value for the critic's output unit at a given time tick is given by the sum of the external reinforcement and its own (discounted) output at the next time tick.

The critic's error is also the reinforcement for the reinforcement learning algorithm of the control network. The latter needs to consider only the last and the present state, as above, but it also might be an on-line version of Williams' 'extended REINFORCE' algorithms [20].

Here is the description of the algorithm:

First, all weights are randomly initialized with real values.

For all episodes:

In the beginning of each episode, at the first time tick, the activations of input units of the reinforcement learning control network are initialized to values determined by sensory perceptions from the environment, and the activations of the probabilistic hidden and output units are initialized to 0*. All unit activations of the recurrent critic also are initialized with* 0*. The dimensionality of the input vector of the critic is equal to the number of units in the control network. The critic receives as input the complete activation vector of all units of the recurrent network, and performs one initializing update step: Each non-input unit sums the weighted activations of its source units and passes the sum through a sigmoid function to obtain its activation. One of the critic's units is called its output unit. The variable r is set* *equal to the activation of this output unit and is interpreted as a prediction of the cumulative discounted reinforcement to be received in the future.*

For all following time ticks:

At a given time tick t:

1. The control network performs one update-step: Each probabilistic non-input unit i sums its weighted inputs, this sum is passed to the logistic function which gives the probability that the activation $x_i(t)$ becomes 1 or 0. Each unit i also stores all information needed by the reinforcement learning algorithm to be applied in step 3. Output units may cause an action in the environment, this may lead to new activations for the input units.

2. The critic performs one update step. Its new activations depend on $x(t)$ and on past activations of its hidden and output units.

The variable r' is defined to be the sum of the current external reinforcement $R(t)$ and the new estimation of final reinforcement, obtained by multiplying the value $P(t)$ of the critic's output unit with a discount factor $0 < \gamma < 1$: $r' := \gamma P(t) + R(t)$.

The error for the critic's output at time $t-1$ is given by $r' - r$. Credit assignment for the critic takes place immediately according to Williams and Zipser's version [22] of Robinson's supervised learning algorithm [6].

3. The critic's error at the same time is the reinforcement for the reinforcement learning algorithm of the reinforcement learning network.

4. r is set equal to $P(t)$.

Again the computation of error signals for the critic's output is very much inspired by Sutton's TD-methods. TD methods, however, require two successive predictions during the same time tick in order to remove dependencies on weight changes. Since the recurrent critic's output already depends on past states (by means of its internal feedback) and also on past weights, the scheme described above makes only one critic update at a time. In [12] (in the section on 'useful extensions') a more complicated recurrent critic based on 'gradient descent through a frozen model network' is described (see also the next section).

Vector-Valued Adaptive Critics and System Identification

The critic above as well as adaptive critics described by other authors are *one-dimensional.* Their prediction refers to a single scalar value, namely, the cumulative future reinforcement. One single internal reinforcement signal is used to modify *all* controller weights in an unspecific fashion. *There are no 'individually tailored reinforcement signals'* [20].

No difference is made between different kinds of reinforcement. This seems to contrast with the reinforcement signals of biological systems. The latter usually make use of a wide variety of 'pain' and 'pleasure' sensors. We will now introduce a *vector-valued* adaptive critic system which includes an adaptive model of the dependency of internal reinforcement *vectors* on (possibly multi-dimensional) ouput actions.

Which are the advantages to be expected by such a system? To speak intuitively: A *detailed* model of the expected consequences of certain actions should allow their *informed* modification. If one knows how much influence a particular output node had on which components of the internal reinforcement vector, one can use this knowledge for 'individually tailored' modifications of the controller weights. Furthermore, it may be *easier* to learn a mapping from states/actions to vector-valued reinforcement than to learn a mapping from states/actions to the corresponding scalar reinforcement.

Three Interacting Networks

One possibility for implementing a detailed model of actions and corresponding internal reinforcement is the following one: We introduce a *third* feed-forward network M [10] which at a given time step sees the input vector and the output vector of the controller C. The system to be identified by M is the process which maps state/action pairs to *internal* reinforcement vectors. At a given time M is trained to predict *the difference between the current and the next prediction of the critic.* This difference is equal to the current internal reinforcement vector. Using back-propagation, the difference between the *desired* and the actual internal reinforcement vector is propagated back through M and through C's output units down into C. C's output units are thereby considered to be the hidden units of the model/controller combination (C's outputs are identified with the corresponding inputs of M.) Only C's weights change, M's weights remain fixed. (This is the approach of 'gradient descent through frozen model networks', see e.g. [4] and [18]). However, to be able to use the back-propagation method we have to get rid of the 'all-or-nothing'-character of the probabilistic units used above, so we make C a feed-forward back-propagation network with semilinear units. Conventional back-propagation networks are deterministic. Since we need explorative capabilities, we introduce a differentiable probability distribution for C's outputs: Each output unit is replaced by two units, one computing the mean and the other computing the variance of a random number generator which provides the final output of the corresponding probabilistic unit. Now we may apply Williams' method of 'back-propagation through random number generators' [19].

Note that both C and M may be replaced by recurrent networks.

Making Two Networks out of Three

To simplify the whole system we may collapse the three-network system above into a similar two-network system. We introduce a network MAC which at the same time fulfills the task of the of the *M*odel network and the *A*daptive *C*ritic above. MAC receives as an input the current input and output of C. Instead of predicting differences between successive critic predictions, MAC learns to predict the critic output itself, by looking at its own next prediction (as feed-forward adaptive critics always should do): MAC's error function at time t is

$$P_{t,v(t)} - \gamma P_{t+1,v(t)} - R(t+1),$$

where $P_{t,v(s)}$ is MAC's prediction based on the controller input and output at time t and MAC's weight-vector $v(s)$ at time s, $R(t)$ is the external reinforcement *vector* at time t, and $0 < \gamma < 1$ is the discount factor for avoiding predictions of infinite sums. (Thus MAC takes over the function of the critic). Errors for the controller are generated analogously to the three-network system described in the last subsection. Since MAC does not evaluate just a state but a state/action pair, it is similar to the approaches described in [16] and [5].

MAC and C may be recurrent: Here is the point where the current report and [12] converge. In [12] (in the section on 'useful extensions') a detailed description of an extension of the C/MAC approach is given which is based on two interacting fully recurrent networks.

One of these networks is used partly for predicting the next controller inputs and partly for predicting the sum of future cumulative reinforcement vectors.

Pole Balancing with a Vector-Valued Adaptive Critic

The task we chose was to test the ideas of the preceeding section to a pole balancing task described in [1]. Programming and tests were conducted by Klaus Bergner, a student at TUM.

The outputs of the control network served to control forces applied to a cart to which a rigid pole was hinged. The cart was able to move on a one-dimensional track. The cart pole system was modeled by the equations given in the appendix. The task was to learn to balance the pole as long as possible without hitting the edges of the track.

Unlike with many other pole balancing tasks, there was *no teacher to give the desired outputs at given time ticks.* The only goal information available to the system was negative reinforcement whenever one of the critical conditions above was violated, which also meant the end of the current 'episode'. Within an episode the external reinforcement vector was equal to 0, so the system faced a spatio-temporal credit assignment task.

Following [1] we made the task more difficult than the similar task described in [3], where a prewired decoder was used to provide binary 162-dimensional input to a single-unit 'network', with all components being zero except for one. Instead the input was real-valued, and additionally scaled in an asymmetric manner (see appendix), in order to force the system to discover a non-trivial internal representation by itself. (Using the input variables directly, without scaling, makes the task easier [1]. Anderson identifies the reason as a symmetry of optimal actions referring to positive and negative values of the state variables.)

Both C and MAC were standard 3-layer feed-forward networks. C had 4 input units for the 4 'visible' scaled state variables $\bar{x}, \bar{\dot{x}}, \bar{\theta}, \bar{\dot{\theta}}$ (defined in the appendix). In addition, C had 5 logistic hidden units and one output unit. C's output unit was probabilistic and consisted of one linear unit (with slope 1) for mean generation, one linear unit (with slope 1) for variance generation, and a random number generator. At a given time, the contribution of the variance generator to the final output was its current activation multiplied by

$$-ln(\frac{1}{rnd} - 1),$$

where rnd was a random variable uniformly distributed between $\frac{1}{2^{16}}$ and 1. At a given time, the activation of the output unit was interpreted as the force (measured in Newtons) to be applied to the cart.

MAC had 5 input units (one for C's output, 4 for the scaled state variables), 5 logistic hidden units and 4 linear output units (with slope 1) for predicting four different kinds of 'pain': 'cart bumps against left edge', 'cart bumps against right edge', 'pole angle exceeds maximal value', and 'pole angle below minimal value'. In case of failure the 'pain contribution' for the corresponding prediction was 1.0. An additional 'true' unit which was always on was connected to all non-input-units of the C/MAC-system in order to provide a modifiable bias. For scaling reasons there was a connection with a fixed weight of 0.1 between C's output unit and MAC's corresponding input unit. Of course, this fixed weight was taken care of during the error-propagation phases from MAC down into C.

At the beginning of each episode, x was randomly initialized between -2.4m and +2.4m, θ was randomly initialized between -0.21 and 0.21, $\dot{x}$ was randomly initialized between -1.5m/s and +1.5m/s, and $\dot{\theta}$ was randomly initialized between -2.0/s and +2.0/s. Between two time steps C's input changed according to a simulation of the cart-pole system by Euler's method with a time step of 0.02s.

C's learning rate was equal to 100, MAC's learning rate was equal to 0.2, the discount factor γ was equal to 0.95. Weights were randomly initialized between -0.05 and 0.05. Five test runs were conducted. The episodes needed to achieve the first episode of more than 30000 time ticks were counted. (If output actions were selected randomly, then the average time until failure was less than 20 time steps. The longest run reported by Anderson [1] took 28407 time steps, more than 7000 failures had to be experienced to achieve that result.)

The results of the five test runs were 713, 486, 536, 614, and 513.

Within less than 800 failures the system always produced an episode with more than 30000 time steps balancing time. *Similar results with a* one-dimensional *critic could not be obtained.*

Using the input variables directly (without scaling) led to even better results: Here the corresponding five numbers were 174, 180, 144, 119, and 155.

It is expected that the concept of multi-dimensional MAC's will prove to be superior, particularily when it comes to complex tasks where there are multi-dimensional action vectors and multi-dimensional 'pain' or 'pleasure' vectors. We have started to apply vector-valued MAC's to industrial robot control.

An Approach to Local Supervised Learning in Recurrent Networks

In this section we propose a local learning scheme for supervised learning in continually running recurrent networks, where each unit at each time receives an individual error signal. The method is based on back-propagation (BP) [17] in recurrent networks unfolding in time [7]. The global error measure to be minimized is the sum of all errors received at the output units over time. The important difference will be that the method is local in space and time, while conventional BP is not. In conventional BP each unit needs a stack for remembering past activations which are used to compute contributions to weight changes during the error propagation phase. Starts and ends of sequences have to be indicated by an external teacher.

Instead of allowing unlimited storage capacities in the form of stacks, we introduce a second adaptive but static network (again termed the 'critic'). Its task is to associate states of the recurrent (primary) network with error-vectors.

The behavior of both interacting networks can be described like this: Activations spread through the primary network in the same manner as with conventional BP. At each discrete time tick the critic receives as input the state vector of the non-input units of the primary network. The sum of the critic's output and the error observed at certain output-units is used as an error-vector. This error-vector is propagated backwards through the primary network, but only one step 'back into time'. (So each unit of the primary network has to store its last activation.) The involved weights are changed immediately afterwards, assuming that the learning rate is sufficiently small to avoid instabilities. (Immediate weight changes are also employed by Williams and Zipser [22] who tested another learning algorithm for fully recurrent networks - first described by Robinson and Fallside [6] - which is local in time but not in space.) Immediate weight changes at the expense of deviating from true gradient descent make it unnecessary to accumulate a sum of weight changes for each weight, The new error-vector received at the non-input units after the one-step back-propagation phase, becomes associated with the last state of the primary network. This association has to be done by the static learning algorithm of the critic, which can be a Boltzmann machine, or a feed-forward BP network, or something else.

A critical assumption of this scheme is that the state of the non-input units at a given time uniquely represents the history which led to this state. Two different histories leading to the same internal state cannot be distinguished by the critic. In such cases it is likely that incorrect error vectors are one-step-back-propagated during further training. A self-healing effect could be that weight modifications caused by this process lead to new errors which in turn split 'critical' states into two or more distinguishable states representing different histories. However, the precise nature of the interactions between two networks like those described above is currently unclear.

The advantage of the scheme is that it is both local in space and local in time: At every time tick the system in principle performs the same local operations, there is no need for storing past activations (except for the last ones), and there is no such thing as epoch boundaries.

For several reasons the method does not implement exact gradient descent. Two of them have been mentioned above: There are continuous weight changes, and different histories leading to the same state will cause incorrect error vectors. Another (pragmatic) reason is that the critic often will not exactly mirror the relations between primary states and error vectors, since its learning algorithm will not be perfect either. 'Similar' primary states will produce 'similar' error-vectors, where the measure of similarity depends on the complexity of the critic. It remains to be verified whether such a learning scheme will face serious problems or whether the inertia of the static network could even lead to beneficial effects, comparable to the effects induced by momentum terms in conventional BP. In some preliminary experiments with a constrained *linear* critic (modified with the delta-rule) the system sometimes learned, but more often failed to learn a dynamic task (the dynamic delayed XOR problem as described in [22]). An interesting point is, again, that the linearity of the critic did not necessarily prevent the recurrent network from eventually solving its task. Yet it is expected that a non-linear critic will lead to better performance, since in general the error is a non-linearily separable function of the primary system's states.

Conclusion

The common aspect of the methods described above is that they all include a component which learns to associate states of a control network with appropriate error information, in order to allow goal directed weight changes in the control network.

A main motivation behind the presented ideas was the desire for learning algorithms local in space and time. A related step in that direction was undertaken in [9], where a completely local learning method for neural networks based on Holland's bucket brigade was described. Unlike the methods described above, the 'Neural Bucket Brigade' does not depend on explicit evaluation of complete activation states at any time. A potential drawback of this 'weaker' approach was a great sensibility to fluctuations of activation at the unit level. The introduction of a network which judges the whole state of a recurrent network was partly motivated by the desire to escape the instabilities caused by such unit-level fluctuations.

I believe that the concept of network-adjusting networks can be helpful in a variety of contexts. The main idea is: A system which has to learn to perform some task should build a model of what is wrong with its current performance. It should use the hypotheses generated by the model to change its behavior. A model which does not lead to improved performance has to be discarded or at least modified such that it generates better hypotheses concerning the successes or failures to be expected.

Acknowledgements

Thanks to Klaus Bergner for conducting the pole balancing experiments. Thanks to Mark Ring for providing useful comments on a draft of the paper.

Appendix: Details of Cart-Pole Simulation

The cart-pole system, taken from [3], [14], and [1], was modeled by the equations

$$\ddot{\theta} = \frac{g sin\theta + cos\theta \frac{-F - ml\dot{\theta}^2 sin\theta + \mu_c sgn(\dot{z})}{m_c + m} - \frac{\mu_p \dot{\theta}}{ml}}{l(\frac{4}{3} - \frac{m cos^2\theta}{m_c + m})},$$

$$\ddot{z} = \frac{F + ml(\dot{\theta}^2 sin\theta - \ddot{\theta} cos\theta) - \mu_c sgn(\dot{z})}{m_c + m}$$

where $-0.21 < \theta < 0.21$ (angle of pole with the vertical), $-2.4m < z < 2.4m$ (position of cart on track), $g = 9.8\frac{m}{s^2}$ (gravitational acceleration), $m_c = 1kg$ (mass of cart), $m = 0.1kg$ (mass of pole), $l = 0.5m$ (half pole length), $\mu_c = 0.0005$ (coefficient of friction of cart on track), $\mu_p = 0.000002$ (coefficient of friction of pole on cart), $F \in [-25N, 25N]$ (force applied to cart's center of mass, parallel to track). (Note that there is a typing error in the equations given in [3], [14], and [1]: There the gravitational constant is given as $g = -9.8\frac{m}{s^2}$).

The scaled input variables were $\bar{z} = \frac{z+2.4}{4.8}$, $\bar{\dot{z}} = \frac{\dot{z}+1.5}{3}$, $\bar{\theta} = \frac{\theta+0.21}{0.42}$, $\bar{\dot{\theta}} = \frac{\dot{\theta}+2}{4}$.

References

[1] C. W. Anderson. *Learning and Problem Solving with Multilayer Connectionist Systems.* PhD thesis, University of Massachusetts, Dept. of Comp. and Inf. Sci., 1986.

[2] A. G. Barto and P. Anandan. Pattern recognizing stochastic learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, 15:360–375, 1985.

[3] 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.

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

[5] M. I. Jordan and R. A. Jacobs. Learning to control an unstable system with forward modeling. In *Proc. of the 1990 Connectionist Models Summer School, in press.* San Mateo, CA: Morgan Kaufmann, 1990.

[6] A. J. Robinson and F. Fallside. The utility driven dynamic error propagation network. Technical Report CUED/F-INFENG/TR.1, Cambridge University Engineering Department, 1987.

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press, 1986.

[8] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, 3:210–229, 1959.

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

[10] J. Schmidhuber. Additional remarks on G. Lukes' review of Schmidhuber's paper 'Recurrent networks adjusted by adaptive critics'. *Neural Network Reviews*, 4(1):43, 1990.

[11] J. Schmidhuber. Learning algorithms for networks with internal and external feedback. In D. S. Touretzky, J. L. Elman, T. J. Sejnowski, and G. E. Hinton, editors, *Proc. of the 1990 Connectionist Models Summer School*, pages 52–61. San Mateo, CA: Morgan Kaufmann, 1990.

[12] J. Schmidhuber. Making the world differentiable: On using fully recurrent self-supervised neural networks for dynamic reinforcement learning and planning in non-stationary environments. Technical Report FKI-126-90 (revised), Institut für Informatik, Technische Universität München, November 1990. (Revised and extended version of an earlier report from February.).

[13] 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.

[14] R. S. Sutton. *Temporal Credit Assignment in Reinforcement Learning.* PhD thesis, University of Massachusetts, Dept. of Comp. and Inf. Sci., 1984.

[15] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[16] C.J.C.H Watkins. *Learning from Delayed Rewards.* PhD thesis, King's College, Oxford, 1989.

[17] P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard University, 1974.

[18] 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.

[19] 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.

[20] 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.

[21] R. J. Williams and Leemon C. Baird. Draft: A mathematical analysis of actor-critic architectures for learning optimal controls through incremental dynamic programming. Technical report, College of Comp. Sci., Northeastern University, Boston, MA, 1990.

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