ADAPTIVE LOOK-AHEAD PLANNING

SEBASTIAN THRUN†‡ · KNUT M"OLLER‡ · ALEXANDER LINDEN†

†German National Research Center for Computer Science
D–5205 St. Augustin, Postfach 1240, F.R.G.
e-mail: st@gmdzi.uucp, al@gmdzi.uucp

‡University of Bonn
Department of Computer Science
D–5300 Bonn, R"omerstr. 164, F.R.G.

Abstract

We present a new adaptive connectionist planning method. By interaction with an environment a world model is progressively constructed using the backpropagation learning algorithm. The planner constructs a look-ahead plan by iteratively using this model to predict future reinforcements. Future reinforcement is maximized to derive suboptimal plans, thus determining good actions directly from the knowledge of the model network (strategic level). This is done by gradient descent in action space.

The problem of finding good initial plans is solved by the use of an "experience" network (intuition level). The appropriateness of this planning method for finding suboptimal actions in unknown environments is demonstrated with a target tracking problem.

Keywords: planning, reinforcement learning, temporal credit assignment problem, gradient descent, target tracking

Introduction

Undoubtedly planning is an important and powerful concept in problem solving [12]. Planning concerns the synthesization of a sequence of actions to achieve a specific goal.

Connectionist approaches so far rely on an associative mapping, which selects good actions given environmental state descriptions. These are based on the interaction of a world model and an action generating network [1, 2, 6, 10, 11, 14, 15, 20]. We recognize three major problems with these approaches:

1. Since no explicit consideration of the future is made, future effects of actions must be directly encoded into this world model mapping, thus model learning becomes complicated.
2. After learning the integration of additional constraints is not possible.
3. While the model network is an essential part of the training of the action network, the learning of the latter lags behind that of the former.

In this paper we present a new connectionist planning procedure. Our model network learns one-step predictions by observing the environmental mapping [2, 6, 11]. In this way training information is immediately available and model learning is easier and faster. With such a model network a look-ahead plan is constructed and subsequently optimized. We demonstrate the performance of this planning procedure through simulations on a target tracking problem.

Reinforcement Learning

In common supervised learning tasks usually an explicit target pattern is known for each situation. E.g. if we train a network to find optimal actions in a controlling task, supervised learning can only be used if these optimal actions are known to the teacher. However, when supervised learning is used for tasks where only a simple reinforcement signal is received, then the optimal actions (the targets) are not known a priori. Instead this reinforcement signal evaluates all past actions: the better the actions of the past,

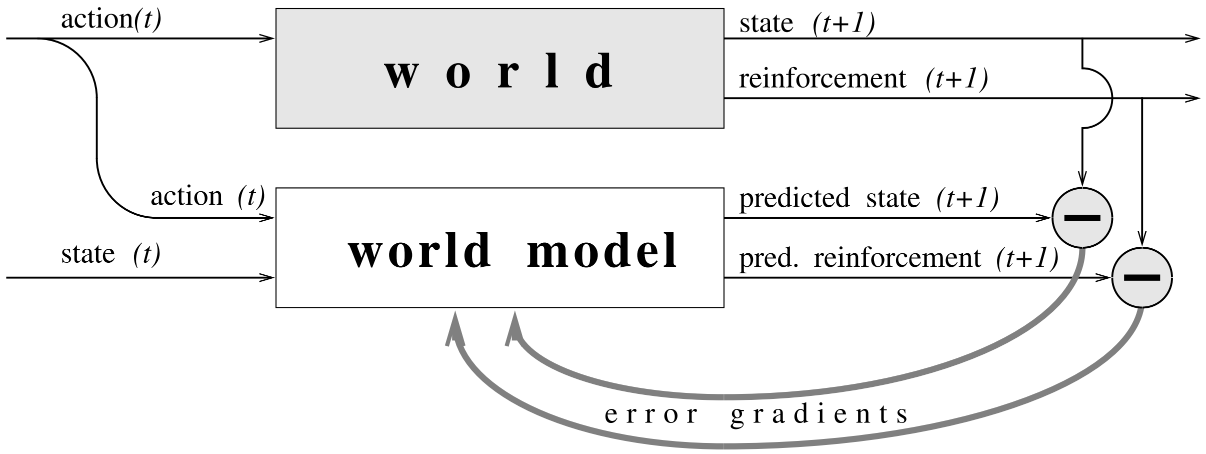

Figure 1: Training of the world model by error comparison and backpropagation. The model is learning to predict subsequent world states and reinforcements.

the better the reinforcement. Since the reinforcement is an unstructured overall signal, the problem with reinforcement learning tasks is the assignment of particular reinforcements to particular actions in the past. This problem is called the *temporal credit assignment problem.*

Many approaches use a network which solves this problem directly [1, 11, 17]. This network, the controller network, learns to generate actions that optimize the whole future reinforcement. Obviously the quality of actions depends strongly on future actions. Thus if the controller generates an action, future actions are implicitly contained in this decision.

The planning procedure presented in this paper does not solve the temporal credit assignment problem directly, although the experience network described below can be considered as a solution for this problem. Instead of assigning a quality value to each possible action, actions are optimized by a look-ahead planning procedure which optimizes actions with respect to the next N reinforcements. This implies the assumption that the effect of a certain action to the reinforcement occurs in the next N time steps – similar assumptions are also made in [1, 17]. N can be an arbitrary number – the computational costs of the optimization steps are linear in N. Moreover, N can be determined at planning time dynamically.

The World and the World Model

Planning is a hypothetical process, in which future states and future actions are involved. If we consider future events for finding optimal actions, it is not sufficient to work with the real world only. Indeed, some kind of a world model is demanded, which learns gradually to mimic the qualities of the real world. The training of this world model is a system identification task.

In this paper, we use a multilayer differentiable, non-recurrent connectionist network for modeling the world (c.f. figure 1). This network is trained by backpropagation to predict the behavior of the world [1, 2, 6, 10, 14, 20].

Formally, the world considered in this paper is defined as a mapping, which maps an action vector $\vec{a}(t)$ with a current state $\vec{s}(t)$ to a subsequent state $\vec{s}(t+1)$ and reinforcement $\vec{r}(t+1)$. Before training, the mapping of the world is unknown. Hence by exploring the world we obtain training information for the world model: if we change the world's state at time $\vec{s}(t)$ by an arbitrary action $\vec{a}(t)$ (e.g. random action), we obtain a subsequent state vector $\vec{s}(t+1)$ and a corresponding reinforcement $\vec{r}(t+1)$. These signals are used as a teacher signal for training the world model. At the same time we use the model network for predicting the state $\vec{s}_{pred}(t+1)$ and reinforcement $\vec{r}_{pred}(t+1)$. If we compare predicted and real state and reinforcement, we can compute an error gradient, which is used for adapting the internal parameters of the model network – namely the weights and the biases – in order to decrease the prediction error (c.f. figure 1). This is done by propagating the error back through the network using the backpropagation algorithm [16, 18].

Adaptive Look-Ahead-Planning

The planning procedure presented in this paper is an approximation procedure, which starts with a initial plan and improves this plan stepwise by gradient descent in order to maximize the next N reinforcements. Let us assume we have such an initial N-step look-ahead plan. This is a sequence of proposed actions for

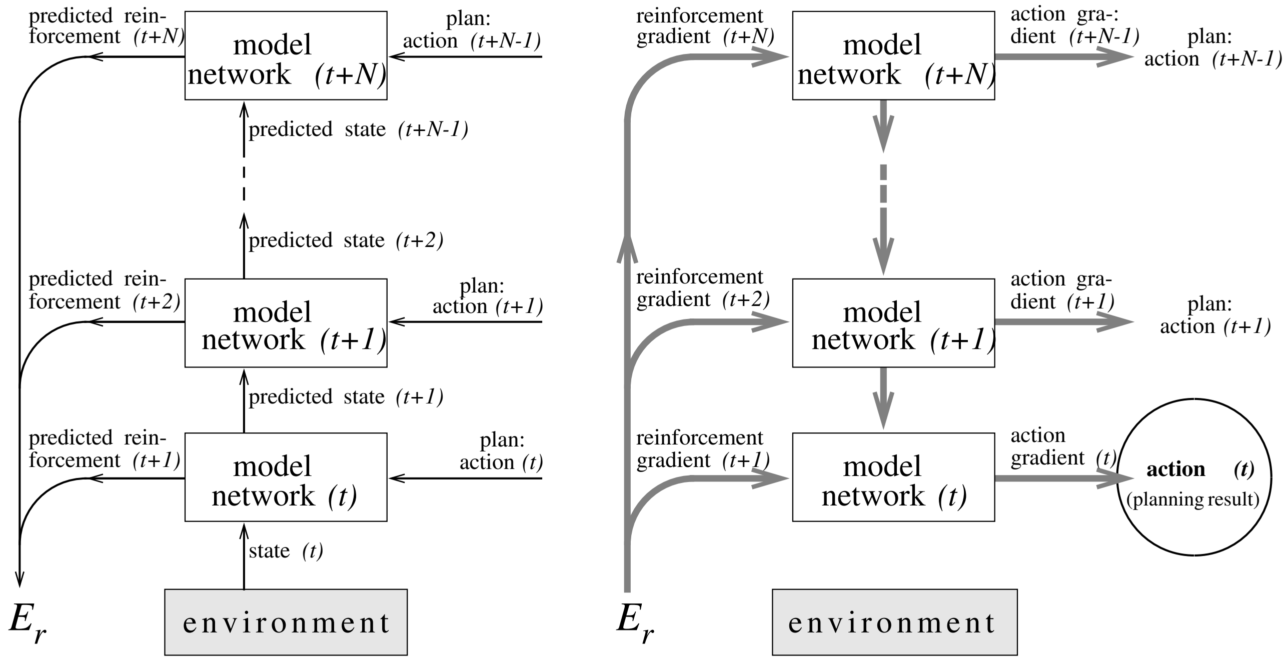

Figure 2: Planning with a chain of adaptive model networks. The black arrows indicate the activation stream, the grey arrows the energy gradient flow. With the actual world state the reinforcement for the next N steps is predicted with respect to a plan. Then the plan is changed in small steps in order to maximize the reinforcement.

the next N time steps starting at the actual time t:

$$\vec{a}_{plan}(t), \vec{a}_{plan}(t+1), ..., \vec{a}_{plan}(t+N-1).$$

(Some heuristics for obtaining initial plans are explained below.)

Now we can use the model network for predicting the subsequent state $\vec{s}_{pred}(t+1)$ and reinforcement $\vec{r}_{pred}(t+1)$. If we assume that this prediction is a good approximation of the real state, we can build up a chain of N copies of the model network for forecasting the next $N-1$ states and reinforcements (c.f. figure 2). Thus we obtain a prediction for all reinforcements

$$\vec{r}_{pred}(t+1), \vec{r}_{pred}(t+2), ..., \vec{r}_{pred}(t+N)$$

in this look-ahead window.

For improving the actions with respect to the predicted reinforcement we use a gradient descent algorithm in action space [7, 8, 19], which will be derived in detail in the next section. It computes the gradients of the reinforcement with respect to the plan, which give us the information, how to change the plan in small steps in order to improve its performance.

The whole procedure described above is to be repeated. After a fixed time or if convergence is observed the algorithm is terminated and the first action of the resulting plan $\vec{a}_{plan}(t)$, the result of the planning procedure, is executed from the environment

$$\vec{a}(t) \quad = \quad \vec{a}_{plan}(t).$$

This action is a (sub-)optimal action, i.e. which yields (sub-)optimal reinforcement regarding the current model network.

In the next section we derive a feed-forward algorithm for computing the desired gradients and then we discuss how to obtain initial plans.

The Feed-Forward Algorithm for Gradient Search in Action Space

As mentioned above the environment is modeled by a non-recurrent multilayer backpropagation network. This restriction is sufficient for our simulation results – the extension of the algorithm to recurrent networks [3, 4, 5, 9, 13, 14, 15, 21] is straightforward and shown in [18].

The external input of the world model network is a state vector $\vec{s}(t)$ and an action vector $\vec{a}(t)$. Both state and action vector are the external input $\vec{I}(t)$ of the model network; for all non-input units this external input is 0. The output of the network is the predicted state and the predicted reinforcement, thus the number of input units, which receive state values, is equal to the corresponding output units – this is crucial for concatenating the world models.

Let the activation function of each unit be given by

$$x_k(t) = \sigma_k(net_k(t)) + I_k(t) \quad \text{with} \quad net_k(t) = \sum_j w_{kj} x_j(t) + \theta_k. \tag{1}$$

Here $x_k(t)$ denotes the activation value of unit k at time t, w_{kj} the weight from unit j to unit k, θ_k the bias of unit k, $I_k(t)$ the external input of unit k at time t and σ_k denotes an arbitrary differentiable squashing function, usually $\sigma_k(net_k(t)) = (1 + e^{-net_k(t)})^{-1}$.

Let us assume that we have some initial plan. As described in figure 2, the actual state of the world $\vec{s}(t)$ and the actions of the plan are propagated through the chain of models using the activation function (1). Note that the external input $I_i(t{+}s)$ for each state input unit i of the sth copy of the model network $(1 \leq s \leq N{-}1)$ is fed with the activation $x_{i'}(t{+}s{-}1)$ of the corresponding state output unit i' of the preceding model copy. The external action input $I_j(t{+}s)$ is fed with the corresponding action $a_{j'}(t{+}s)$ of the plan.

Unlike the state predictions, which are used directly in the plan evaluation chain, the reinforcement predictions $\vec{r}_{pred}(t{+}\tau)$ $(1 \leq \tau \leq N)$ are used for optimizing the performance of the plan. In order to improve these future reinforcements we define a *reinforcement energy function* E_r:

$$\begin{aligned} E_r &= \tfrac{1}{2} \sum_{\tau=1}^{N} (\vec{r}_{opt} - \vec{r}_{pred}(t{+}\tau)) \cdot I\vec{g}(\tau-1) \cdot (\vec{r}_{opt} - \vec{r}_{pred}(t{+}\tau))^T \\ &= \tfrac{1}{2} \sum_{\tau=0}^{N-1} \sum_k g_k(\tau)(r_{opt,k} - x_k(t{+}\tau))^2 \end{aligned} \tag{2}$$

(this holds since the predicted reinforcement $\vec{r}_{pred}(t{+}\tau)$ is the activation of some output units $x_k(t{+}\tau{-}1)$, c.f. figure 2.) Here I is the identity matrix, $\vec{g}$ is a weighting function, in the simplest case $\vec{g} \equiv (1, ..., 1)$, and $\vec{r}_{opt}$ is the *optimal* reinforcement, e.g. $\vec{r}_{opt} = (1, ..., 1)$.

In the sequel we show how to compute the gradients of E_r with respect to the plan. These tell us how to change the actions of the plan in order to improve the predicted reinforcement, thus how to optimize the plan with respect to our world model.

One way of computing the gradients of E_r with respect to the actions is using backpropagation through the spatial unfolded time-structure [16, 19]. Since no dynamical determination of the plan length N is possible by using this backpropagation-in-time technique, we will derive a pure feed-forward algorithm for computing these gradients.

Let us define the gradient of each activation x_k with respect to the external input $I_i(t{+}s)$ by

$$\xi_{is}^k(\tau) \equiv \frac{\partial x_k}{\partial I_i(t{+}s)}(t{+}\tau). \tag{3}$$

If we know these gradients for all input units i, all reinforcement prediction units k and all time steps $s, \tau \in \{0, 1, ..., N{-}1\}$ we can compute the desired gradients for changing the actions with the stepsize $\eta > 0$:

$$\begin{aligned} \Delta I_i(t{+}s) &= -\eta \frac{\partial E_r}{\partial I_i(t{+}s)} \\ &= \eta \sum_{\tau=0}^{N-1} \sum_k g_k(\tau)(r_{opt,k} - x_k(t{+}\tau)) \cdot \frac{\partial x_k}{\partial I_i(t{+}s)}(t{+}\tau) \\ &= \eta \sum_{\tau=0}^{N-1} \sum_k g_k(\tau)(r_{opt,k} - x_k(t{+}\tau)) \cdot \xi_{is}^k(\tau) \end{aligned} \tag{4}$$

It remains to show how to compute these $\xi_{is}^k(\tau)$.

Obviously $\xi_{is}^k(\tau) = 0$ holds for all $s > \tau$, since no activation depends on a future action. For all τ, for *all* units k and for all $s \le \tau$ $\xi_{is}^k(\tau)$ can be propagated forward through the network by[1]

$$\begin{aligned}
\xi_{is}^k(\tau) &\overset{(1)}{=} \frac{\partial \sigma_k(net_k(t+\tau))}{\partial I_i(t+s)} + \frac{\partial I_k(t+\tau)}{\partial I_i(t+s)} \\
&= \sigma_k'(net_k(t+\tau)) \sum_j w_{kj} \frac{\partial x_j}{\partial I_i(t+s)}(t+\tau) + \delta_{ik}\delta_{s\tau} \\
&= \sigma_k'(net_k(t+\tau)) \sum_j w_{kj} \xi_{is}^j(\tau) + \delta_{ik}\delta_{s\tau}
\end{aligned} \tag{5}$$

Since at non-recurrent multilayer networks input units receive only external input (i.e. $net_k \equiv 0$) and the remaining units, the hidden and output units, receive only internal input ($I_k \equiv 0$), (5) reduces to:

$$\begin{aligned}
k \text{ action input unit:} \quad \xi_{is}^k(\tau) &= \delta_{ik}\delta_{s\tau} \\
k \text{ no input unit:} \quad \xi_{is}^k(\tau) &= \sigma_k'(net_k(t+\tau)) \sum_j w_{kj} \xi_{is}^j(\tau)
\end{aligned} \tag{6}$$

So far, we have derived a rule for propagating gradients through one copy of the model network, namely to derive the gradients of the output activations from those of the input units. It remains to state a propagation rule for propagating these gradients forward through the whole chain of model networks. Since the state prediction $\vec{s}_{pred}(t+\tau)$ is used as the state input $\vec{s}(t+\tau+1)$ for the next time step, for each state input unit k and the corresponding state output unit k' the gradients are equivalent:

$$k \text{ state input unit:} \quad \xi_{is}^k(\tau) = \xi_{is}^{k'}(\tau-1) \tag{7}$$

With the equations (6) and (7) the gradients of the reinforcement energy function with respect to all actions of the plan are computed. According to (4) these actions are changed in small steps in order to decrease E_r:

$$\vec{a}_i(t+s) \longleftarrow \vec{a}_i(t+s) - \eta \frac{\partial E_r}{\partial I_{i'}(t+s)} \tag{8}$$

for all action vector components i, the corresponding input units i' of the model network and all plan steps $s \le N-1$.

Variable Plan Lengths

The advantage of our feed-forward algorithm unlike backpropagation-through-time is the possibility to determine plan lengths dynamically at planning time. During construction of the model chain we propagate both activations and gradients ξ forward through the network. Thus after each look ahead into future we know by definition (3), how an infinitesimal small change of the first action of the plan $\vec{a}_{plan}(t)$ effects the actual state predictions $\vec{s}_{pred}(t+\tau)$.

Instead of the real world we use the world model for planning and predicting future effects. Hence this model does not match the world exactly but approximates its behavior, look-ahead planning can be cut as soon as the estimated error of the current model network is larger than (a constant c times) the maximal estimated effect of $\vec{a}_{plan}(t)$ to activations of the actual copy of the world model. Then the model is too inaccurate – further look ahead is not expected to turn out precise new information. One obtains an estimation for the error from the model training procedure: for example, the minimal or average observed error can be taken as a lower bound of this error.

Initial Plans – The Experience Network

There are a lot of different strategies for finding initial plans. They can be derived by heuristics like

- random, last or average action (if exists),
- rest of the last plan (if exists) or

[1] δ denotes the Kronecker delta.

Figure 3: Target tracking – basic strategy: The system (boxes, arrow) moves always into the target's (crosses) direction.

- local optimization: Starting with one of the above initial actions only the one-step reinforcement is optimized by gradient descent. This strategy produces suboptimal actions with respect to the immediate next predicted reinforcement.

If some planning took place before, one should use the rest of the previous plan, $N{-}1$ actions, as the first $N{-}1$ actions of the new plan. This reduces the problem of finding an initial plan to finding the last action of the initial plan only.

Despite of those fast static strategies it is interesting to investigate adaptive modules for determining initial actions. One adaptive way of finding initial plans is the use of an *experience network* in addition to the world model network. This network is trained in a supervised manner (e.g. with backpropagation) to compute the resulting action of the training procedure from the current state. The experience network is similar to the *control network* described in [1, 17], but it is used in a different way. Its output is optimized by our planning procedure before it is given to the world.

The advantage of using an experience network is that the time-consuming planning procedure is shifted gradually to the *experience* in the experience network. This decreases the whole planning computation time.

Simulation Results: Target Tracking

We tested our planning method on a target tracking task. The system tried to reach a target in a two-dimensional space. The target's policy was not to flee in a fixed direction, but to move always 90° to the current direction of motion of the system. The system had to learn the policy of the target for reaching the target as quick as possible.

It was sufficient to use a one layer world model network for predicting states and reinforcements. The state input and output consisted of the actual coordinates of system and target object. The action was also a two-dimensional vector, which pointed into the movement direction of the system – the length and thus the speed of a movement was fixed. The goal was to minimize the euclidian distance between system and target. Corresponding to the two dimensions of the plane the reinforcement was split into a horizontal and a vertical component, each of which was simply defined as the difference between target and system coordinate. The reinforcement error function E_r described above measured the euclidian distance, such that minimizing E_r was equivalent to minimizing this distance. In addition, we used a two-layered experience network with four hidden units for learning the planning results and proposing good initial plans.

For reaching the target in as few steps as possible, the immediate reinforcement should not be optimized due to later reinforcements. This is illustrated in figure 4: if the system uses the simple strategy to maximize

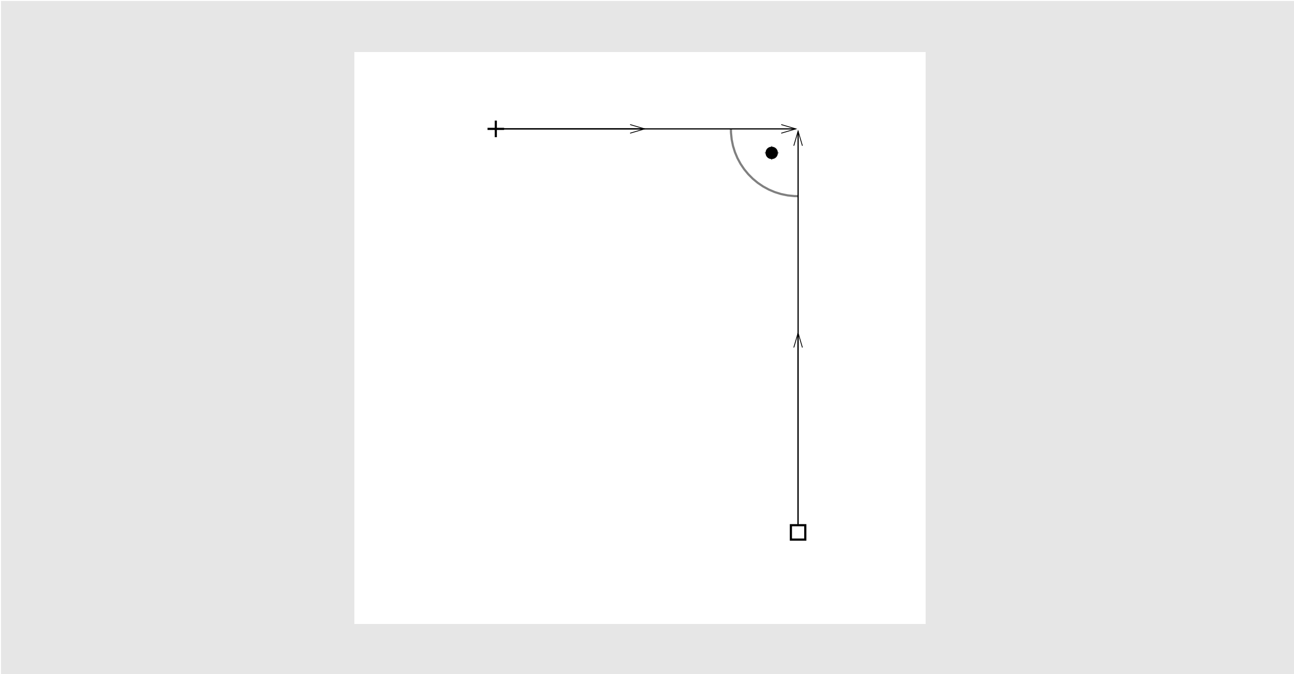

Figure 4: Target tracking – the analytical optimum

the next reinforcement – this would be a non-planning system expected to do – the system will always move in the direction of the target. Obviously this is not the optimal trajectory. The theoretical optimum is shown in figure 5. In this case neither system nor target changes its direction.

The planning process was conducted with a look-ahead $N \leq 7$. The reinforcement weighting function was $g_k(\tau) = 2^\tau$ and the adaptation rate depended also on τ: $\eta = \frac{1}{2} \cdot 2^{-\tau}$. The learning rate of the model and experience network were $\eta_{\text{model}} = 0.1$ and $\eta_{\text{exp}} = 0.05$.

After about 3500 training cycles we observed the trajectories shown in figure 6. The planner always found a close to optimal solution. This demonstrates the appropriateness of the method for finding suboptimal actions at the target tracking task.

Discussion

The optimization technique used in this paper is a gradient search procedure. Since look-ahead planning, as it is presented in this paper, does not depend on a special optimization method, one can use arbitrary numerical optimization like genetic algorithms etc. as well. In using gradient search techniques we have made certain assumptions about the environment. One is continuity, i.e. similar actions imply similar subsequent states and reinforcements. This has also been assumed in the use of continuous connectionist networks for modeling the environment. Another is that the initial plan is in the E_r-valley of the resulting plan. Therefore the choice of the initial plan is essential for the quality of the planning result.

On control problems usually many solutions exist for achieving a certain goal. E.g. inverse kinematics are often characterized by infinitely many solutions with different properties. Many connectionist approaches reduce this one-to-many mapping to a one-to-one mapping [1, 17, 11, 2, 6, 10, 11], since the action generator, the control network, is a mathematical function. Therefore often a marginal constraint like smoothness etc. is also optimized. The planning procedure, as it is presented above, is able to perform one-to-many mappings as well. For example, this can be done if the selection of initial plans is a probabilistic process or if the adaptation of the plan's action might be combined with a probabilistic search procedure, such that the planner is able to find different results at the same configuration. Up to now no simulation results are available – maybe we will present a probabilistic planning procedure in a later paper.

Acknowledgements

The authors wish to thank Frank Śmieja for fruitful discussions. The preparation of this paper was supported in part by grant ITR 8800 L7 from the German Federal Ministry for Scientific Research and

Figure 5: Target tracking by planning with an adaptive world model. The system always moves very close to the optimal direction.

Technology (BMFT).

References

[1] C. W. Anderson. Learning and problem solving with multilayer connectionist systems. Technical Report COINS TR 86-50, Dept. of Computer and Information Science, University of Massachusetts, Amherst, MA, 1986.

[2] A. G. Barto. Connectionist learning for control: An overview. Technical Report COINS TR 89-89, Dept. of Computer and Information Science, University of Massachusetts, Amherst, MA, September 1989.

[3] J. L. Elman. Finding structure in time. Technical Report CRL Technical Report 8801, Center for Research in Language, University of California, San Diego, 1988.

[4] M. Gherrity. A learning algorithm for analog, fully recurrent neural networks. In *Proceedings of the First International Joint Conference on Neural Networks, Washington, DC*, San Diego, 1989. IEEE, IEEE TAB Neural Network Committee.

[5] M. I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Conference on Cognitive Science*, 1986.

[6] M. I. Jordan. Generic constraints on unspecified target constraints. In *Proceedings of the First International Joint Conference on Neural Networks, Washington, DC*, San Diego, 1989. IEEE, IEEE TAB Neural Network Committee.

[7] J. Kindermann and A. Linden. Inversion of neural nets. *Journal of Parallel Computing*, 1990. (to appear).

[8] A. Linden and J. Kindermann. Inversion of multilayer nets. In *Proceedings of the First International Joint Conference on Neural Networks, Washington, DC*, San Diego, 1989. IEEE.

[9] M. C. Mozer. A focused backpropagation algorithm for temporal pattern recognition. Technical Report CRG-TR-88-3, Depts. of Psychology and Computer Science, University of Toronto, Toronto, Jun 1988.

[10] P. Munro. A dual backpropagation scheme for scalar-reward learning. In *Ninth Annual Conference of the Cognitive Science Society*, pages 165–176, Hillsdale, NJ, 1987. Cognitive Science Society, Lawrence Erlbaum.

[11] D. Nguyen and B. Widrow. The truck backer-upper: An example of self-learning in neural networks. In *Proceedings of the First International Joint Conference on Neural Networks, Washington, DC*, San Diego, 1989. IEEE, IEEE TAB Neural Network Committee.

[12] N. J. Nilsson. *Principles of Artificial Intelligence*. Springer Verlag, Berlin, 1982.

[13] B. A. Pearlmutter. Learning state space trajectories in recurrent neural networks. Technical Report CMU-CS-88-191, Carnegie Mellon University, 1988.

[14] A. J. Robinson. *Dynamic Error Propagation Networks*. PhD thesis, Cambridge University Engineering Dept., Cambridge, UK, February 1989.

[15] A. J. Robinson and F. Fallside. Dynamic reinforcement driven error propagation networks with application to game playing. to be presented at the Eleventh Annual Conference of the Cognitive Science Society, Ann Arbor, 1989.

[16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing. Vol. I + II*. MIT Press, 1986.

[17] R. S. Sutton. *Temporal Credit Assignment in Reinforcement Learning*. PhD thesis, University of Massachusetts, 1984.

[18] S. Thrun. A general feed-forward algorithm for gradient-descent in neural networks. Technical Report In press, GMD, Sankt Augustin, FRG, 1990.

[19] S. Thrun and A. Linden. Inversion in time. In *Proceedings of the EURASIP Workshop on Neural Networks, Sesimbra, Portugal, February 15-17*. EURASIP, 1990.

[20] 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*, SMC-17:7–19, 1987.

[21] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. Technical Report ICS Report 8805, Institute for Cognitive Science, University of California, San Diego, CA, 1988.