to appear in: Proceedings of the IEEE International Conference on
Neural Networks ICNN 96, Washington, 3.-6.6.1996

Application of Sequential Reinforcement Learning to Control Dynamic Systems

Martin Riedmiller, riedml@ira.uka.de

ABSTRACT

The article describes the structure of a neural reinforcement learning controller, based on the approach of asynchronous dynamic programming [BBS93]. The learning controller is applied to a well-known benchmark problem, the cart-pole system. In crucial difference to previous approaches, the goal of learning is not only to avoid failure, but moreover to stabilize the cart in the middle of the track, with the pole standing in an upright position. The aim is to learn high quality control trajectories known from conventional controller design, by providing only a minimum amount of a priori knowledge and teaching information.

1. Introduction

In many tasks to be solved by learning controllers we are faced with the following situation: An unknown system has to be manipulated by an agent or more technically, by a controller, to show a desired behavior. Often, this can only be done by a sequence of control decisions or actions, and the result of the control strategy can only be judged at the end of the task, when the final outcome may be observed. The underlying problem is often called the *temporal credit assignment problem*: Which action in a sequence has to be changed in which way to get a satisfactory outcome at the end? This is the sort of problems we focus in this article. It can be characterized in the following way: Both an appropriate control strategy and the behavior of the system to be controlled are a priori unknown. The only external information provided to the learning control system is a judgment of the final state, reached at the end of the sequence. The task is to incrementally learn a control strategy, that transfers the system of any potential start state to the desired goal state, guided only by the final judgment.

2. A self learning controller

2.1. Basic Concepts

Barto et.al [BBS93] showed the close relationship between a certain class of reinforcement learning problems and the theory of dynamic programming. Dynamic programming (DP) covers a collection of techniques and methods to solve control problems where temporal aspects play a central role. The basic problem is to find a sequence of appropriate control decisions that transfers a system from a current state to a specified goal state. This means that a current control decision cannot be viewed isolated - future consequences of this action have to be considered as well. A control decision with a promising current result may cause insolvable troubles some times later. In DP this problem is solved by reducing the global optimization problem to a local one by the aid of a cost-to-go estimation for each state.

2.2. The model based control architecture

The basic structure of the control architecture is shown in figure 1. Three main components can be distinguished: a neural model network, a neural evaluation network and an algorithmic action selection module. The principle working scheme can be described as follows: The controller observes a certain situation of the plant x_t via its sensory inputs. Then, a candidate action is chosen from a discrete set of available actions $\mathcal{A}(x_t)$. The action is applied to an internal neural model of the plant, which computes a prediction $\hat{x}_{t+1} = F(x_t, a_i)$ of the plant's successor state, when action a_i is applied. Next, the estimated

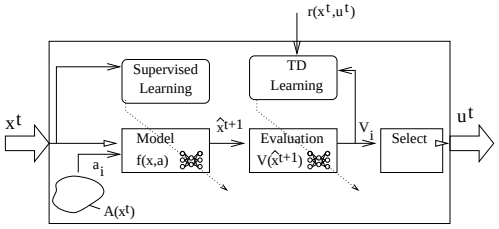


Fig. 1: Principle structure of the model based reinforcement controller

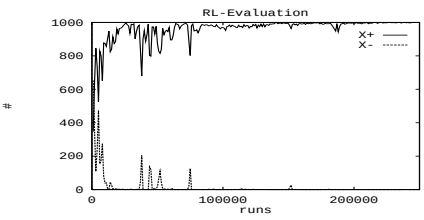


Fig. 2: Training curve

costs to reach the final goal from the potential successor state $\hat{x}_{t+1}$ are computed by the neural evaluation network. This is repeated for all actions in the action set, until the estimated costs for all potential successor states are determined. The actual action selection can now be performed by stepping through the set of evaluated successor states and selecting the action which yields the minimum expected costs to go. When the reaction of the plant is observed, both the neural model and the neural evaluation network are updated. In the following, the components are discussed in more detail.

2.3. Training the neural model

The task of the neural model is to compute the predicted successor state $\hat{x}_{t+1}$ of the plant, when a certain action a_i is applied: $\hat{x}(t+1) = F(x_t, a_i)$. This is a typical supervised learning situation: The input consist of current state and action information, the target is the resulting state of the plant that can be observed, when the action was applied. There are several ways of generating training examples. The method used here was to apply several sequences of control inputs to the plant and to observe the resulting behavior. The neural model was then trained using the superior supervised learning method Rprop [RB93],[Rie94a], which has the advantage of fast and robust convergence.

2.4. Training the evaluation network

2.4.1. Mathematical Framework: The Dynamic Programming Approach

In the following we consider a system which's behavior can be described in terms of a transfer function $x_{t+1} = f(x_t, u_t)$. Here, $x_t \in X$ describes the state of the system at time t, whereas x_{t+1} is the successor state of the system, when control $u_t \in U$ was applied. The goal of a decision making system is to find a policy $\pi : X \mapsto U$ that maps a given state x_t to an appropriate system input u_t, such that a given control objective is fulfilled.

The problem of sequential decision making can be formulated mathematically as the minimization of a cumulative temporal cost function

$$V^{\pi}(x_0) = \sum_{t=0}^{t=N-1} r(x_t, \pi(x_t)) + r(x_N) \tag{1}$$

where $r(x_t, \pi(x_t))$ denotes the scalar cost function or the *immediate payoff* for applying a certain control decision in state x_t, and $r(x_N)$ gives the cost for the final state x_N. N is called the horizon of the problem. Thus, $V^{\pi}(x_0)$ denotes the costs for getting from the start state x_0 to the final state x_N, when following the current policy π.

The key idea of dynamic programming is to decompose the N step 'global' optimization problem into N single step or 'local' optimization tasks. This is done by an iterative improvement of a so-called cost-to-go estimation function. Under the assumption, that the optimal cost-to-go evaluation $V^{\pi^*}(x_t)$ for a certain state x_t was known, the decision making problem reduces to a local minimization problem [BBS93]:

$$\pi^*(x_t) = \min_{u_i}(r(x_t, u_i) + V^{\pi^*}(x_{t+1})) \tag{2}$$

Using the dynamic programming approach for a learning controller, two main aspects have to be considered: First, the control objective has to be appropriately formulated in terms of a temporal cost function. Second, a cost-to-go evaluation function $V^{\pi}(x)$ for a continuous state space must be found. In the controller architecture this is done by incrementally learning the evaluation function by a neural network: In an iterative process the control decision is based on the current evaluation function. After the application the system moves to its successor state x_t, and the evaluation function of the previous state x_{t+1} is updated:

$$V(x_t) \stackrel{!}{=} r(x_t, u_t) + V(x_{t+1}) \tag{3}$$

The corresponding learning-rule for the weights in the neural network is based on Sutton's TD-learning algorithm [Sut88].

2.4.2. The flexible horizon approach

The approach we took here is a slight variation of the above modeling. Instead of taking a number of predefined steps N, we consider the task to be finished, whenever a state within a set of final states is reached. That means, that the number of steps of the sequential decision task may vary (hence the name *'flexible horizon'*). The set of final states $\mathcal{X}^f$ is a part of the set of possible states $\mathcal{X}$. $\mathcal{X}^f$ is further subdivided into a set of 'good' or positive final states, $\mathcal{X}^+$, and a set of 'bad' or negative final states, $\mathcal{X}^-$.

The semantics of the evaluation function changes from being the 'cumulated costs until a predefined number of steps is taken' to the 'cumulated costs until a specified final state is reached'. However, in order to avoid the occurrence of infinitely long sequences, it is necessary in practical use to stop a run when a certain amount of steps N_{max} has been exceeded.

3. Application to cart pole balancing

3.1. Cart Pole Balancing

The cart pole balancing benchmark is again used in this paper - not only for historical reasons (many reinforcement learning architectures have been tested on this benchmark), but also because it is a typical example of an instable single-input multi-output dynamic system. In substantial difference to many previous reinforcement learning approaches on the cart pole benchmark [BSA83] [Rie94b] the controller should not only learn to balance the pole *somehow* for a certain amount of time (avoiding failure), but should furthermore learn to stabilize the instable system while moving the cart to the center of the track. In other words, our effort concentrates on incrementally learning a good control quality, comparable to that of analytical controller design, by using only minimal information on success or failure of the current strategy, as assumed in the framework of reinforcement learning (for a discussion of this important point see also [GS93]).

To summarize, the learning situation is as follows: the controller observes current state information of the plant (provided by appropriate sensors), computes an action according to its current policy, and after some time it gets information about the success or failure of its strategy.

3.2. Experimental setup

A run was terminated when the controlled system reached either a positive or a negative final state or when the number of steps exceeded $N_{max} = 500$. In the following, $x \in \mathcal{X}^+ \Leftrightarrow |angle| < 1^o$ and $|position| < 0.05m$, $x \in \mathcal{X}^- \Leftrightarrow |angle| > 45^o$ or $|position| > 2.4m$.

The final payoffs were chosen to $r^- = 1$ and $r^+ = 0$, whereas the immediate payoffs are uniquely set to $r(x,u) = \frac{r^-}{N_{max}+1}$. The neural evaluator is a multilayer perceptron consisting of four input, 8 hidden units with symmetric activation function and one output neuron using a logistic activation function. The learning method was temporal difference learning as described in the previous section, combined with an ordinary gradient descent rule for the computation of the weight update. The set of available actions was set to $\mathcal{A}(x_t) = \{-10N, +10N\}$. The neural model of the plant was trained off-line using Rprop [RB93].

3.3. Training the controller

When examining several variations and parameter settings of the basic approach, three questions are of special interest:

- how fast does the controller learn to achieve its goal?
- how stable are the results in the long term?
- which final control quality can be achieved?

Figure 2 shows the learning behavior of the controller. The number of successful runs (X^+) sampled over 1000 random starting positions are plotted versus the overall number of training runs. The performance improves very quickly and rests stable when training is continued. This is important with respect to the verification of the approach to exclude random success. After sufficient training, the rate of failure (X^-) decreases to zero. The trained controller needed about 100 steps to move the system from an initial starting position to a positive final state. With two elements in the action set, this corresponds to a number of 2^{100} different policies. Regarding the relative small number of actual training runs, this gives some hint to the generalization quality of the neural evaluation network.

3.4. Evolution of controller capabilities

after 1,000 runs

after 30,000 runs

Fig. 3: Evolution of controller capabilities during the learning process for six different starting positions (marked by a circle)

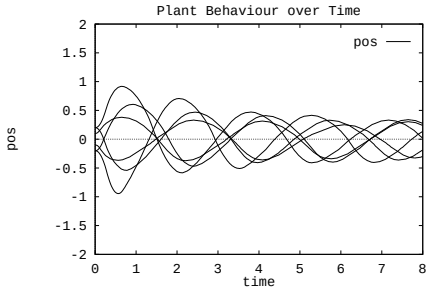
A crude impression of the development of the learned controller strategy can be seen in figure 3. The trajectories of six examples of difficult starting positions (angle of pole up to 20°) are shown in the angle-position-plane. After 1000 runs, the controller has already learned to keep the angle small. Unfortunately it tries to do this by moving the cart towards the end of the track, resulting in a failure. After 30,000 runs, a very good controller performance is reached: All example positions are managed; moreover the controller has developed an efficient policy to achieve its goal very quickly: First, the cart is moved away from the center, forcing the top of the pole towards the center of the track. Then, a failure of the pole is avoided, by accelerating towards the center, where cart and pole are finally balanced and stabilized.

3.5. Controller quality

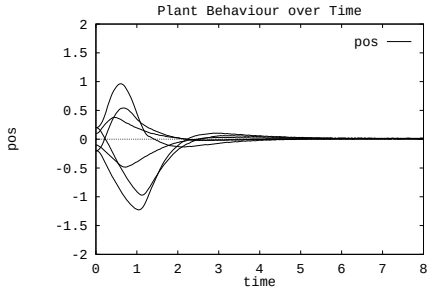
Figure 4 shows the improvement of the controller quality while learning proceeds. After 10,000 runs, the controller has already learned to balance the pole (not shown) and to keep the cart round the center of the track. Both cart and pole are oscillating. After 30,000 runs, all starting positions can be controlled, and the system is quickly moved to the middle of the track, keeping the pole upright.

3.6. Comparison to conventional controller design

If one compares the final control capabilities of a learning controller to an analytical approach, one should always keep in mind the considerable amount of a priori knowledge used by the conventional approach.

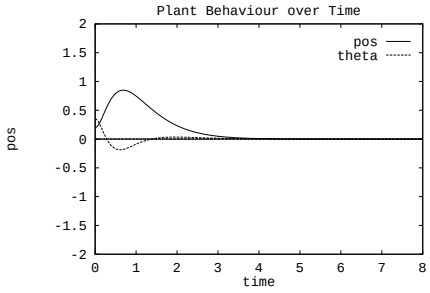


a) after 10,000 runs

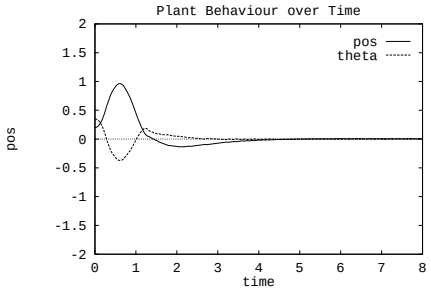


b) after 30,000 runs

Fig. 4: Incremental improvement of the controller's policy on a number of initial starting positions



a) pole assignment controller

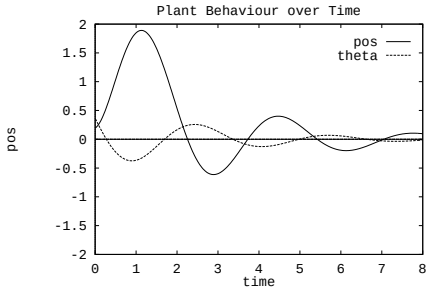


b) neural controller

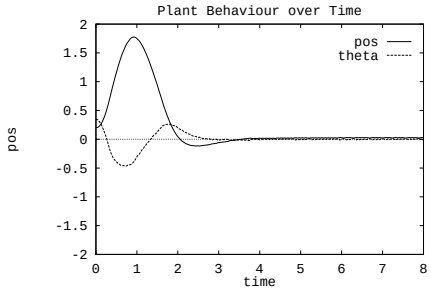
Fig. 5: Comparison of neural and conventional controller on the original plant

However, we used a pole assignment approach and tuned the controller until an acceptable performance was achieved (figure 5). As can be seen on an example trajectory, the neural controller, trained only with a final judgment signal, compares very well to the analytical approach.

The capabilities of the learning controller are stressed, when the parameters of the plant are changed. The pole assignment controller looses a considerable amount of its strength (this can be expected, for it was especially designed to control the original plant). The neural architecture again uses the signals on success or failure to learn to control the modified plant very accurately.



a) pole assignment controller



b) neural controller

Fig. 6: Comparison of neural and conventional controller on a modified plant

4. Conclusion

The article shows the feasibility of learning high quality control trajectories of unknown systems by a neural network based architecture. The controller mainly consists of two neural modules, an identification network, which identifies plant behavior and an evaluation network to solve the temporal credit assignment problem.

Future research will focus on techniques to make the approach suited for practical application. Apparently, the main problem will be the long training times, which we plan to tackle by pretraining on simulated models and by the integration of available a priori knowledge. The results so far are very promising and may hopefully lead to an alternative controller design in situations, where conventional controller design is difficult or infeasible.

References

[BBS93] A. G. Barto, S. J. Bradtke, and S. P. Singh. Learning to act using real-time dynamic programming. *submitted to: AI Journal special issue on Computational Theories' of Interaction and Agency*, 1993.

[BSA83] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuron-like adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13:834–846, 1983.

[GS93] S. Geva and J. Sitte. A cartpole experiment benchmark for trainable controllers. *IEEE Control Systems*, pages 40–51, October 1993.

[RB93] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In H. Ruspini, editor, *Proceedings of the IEEE International Conference on Neural Networks (ICNN)*, pages 586 – 591, San Francisco, 1993.

[Rie94a] M. Riedmiller. Advanced supervised learning in multi-layer perceptrons - from backpropagation to adaptive learning algorithms. *Int. Journal of Computer Standards and Interfaces*, 16:265–278, 1994. Special Issue on Neural Networks.

[Rie94b] M. Riedmiller. Aspects of learning neural control. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, San Antonio, Texas, October 1994.

[Sut88] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, (3):9–44, 1988.