

# TD Learning of Game Evaluation Functions with Hierarchies of Adaptive Experts

Marco A. Wiering and Ben J.A. Kröse

Faculty of Mathematics and Computer Science  
University of Amsterdam

## 1 Introduction

Expert Systems for playing games require an evaluation function which returns the expected payoff from a position given optimal future play of both sides. When the evaluation function for a game is known accurately, it can be used to compare all possible moves in a position after which the move which results in a position with the highest evaluation can be selected.

Since neural networks are universal approximators [Cybenko89] they are able to represent the evaluation function. Learning can be supervised (a human expert or a perfect computer program) or by self-play [Tesauro92, Boyan92, Schraudol94] in which the temporal difference method [Sutton88, Dayan94] is used to generate learning samples.

Games provide domains where the evaluation function can differ drastically for similar positions (e.g. tic-tac-toe, draughts, chess). To solve the problems of having to represent discontinuities and storing large amounts of incoherent knowledge in one single neural network, we propose to use multiple neural networks. In section 2 the modular architectures are described. In section 3 the learning procedures are discussed and in section 4 the experiments with the game of tic-tac-toe and the endgame of backgammon are described.

## 2 Modular Architectures

Modular architectures have been proposed consisting of a number of small expert neural networks which co-operate to learn the desired function [Jordan92, Hampshire89]. Typically they contain ‘gating’ networks which learn to divide the input space into a number of subregions and ‘expert’ networks which learn a specific part of the function. Architectures are hierarchical (figure 1) in which each propagate node uses the normalized output vectors of a gating network to propagate the output vectors of the experts to higher levels. The highest level propagate node will give the final output.

### 2.1 Hierarchical Mixtures of Experts

[Jordan92] developed a modular gating architecture for task decomposition. The system works as follows. The gating networks are linear neural networks and are used to blend the outputs of the experts. The output of a propagate node is a weighted sum of the gates  $g_i$  and the outputs  $y_i$  of the experts :  $y = \sum_i g_i y_i$ .

For understanding the learning algorithm, we must give the hierarchy a probabilistic interpretation. For this the propagate nodes act like single stochastic switches. The gate-value  $g_i$  determines the probability that the propagate node decides to select the output  $y_i$  of the  $i^{th}$  expert network. We must maximize the probability  $P$  that the architecture generates the desired output  $d$  when the input vector  $\vec{x}$  is given. If we use a Gaussian density

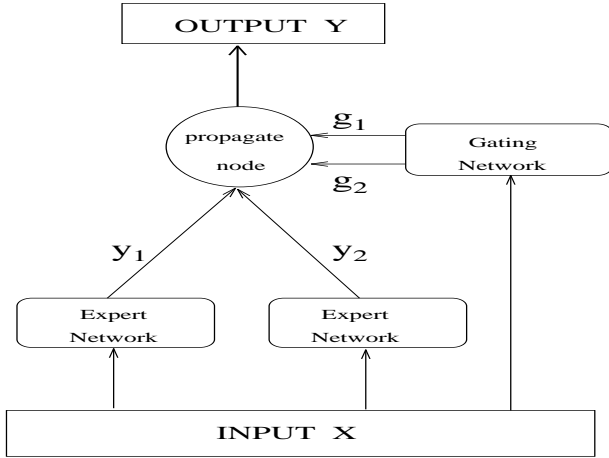


Figure 1: A one level hierarchy of adaptive experts. All networks receive the same input.

function to model the probabilistic component that an expert produces the desired output, the probability that the architecture generates the desired output is

$$P(d|\vec{x}) = \sum_i g_i e^{-\frac{1}{2}(d-y_i)^2}$$

A learning algorithm is developed by using gradient ascent and back-propagation to maximize the log likelihood function given by  $\ln P(d|\vec{x})$ . When we maximize this function, expert networks are made more responsible in regions where they out-compete the other experts.

## 2.2 Meta-Pi Network Architecture

The Meta-Pi network as presented by [Hampshire89] has the same structure as given in figure 1, but instead of competing networks they use co-operating networks; the Meta-Pi gating network learns to make experts more responsible in regions where they can be used to minimize the error of the architecture as a whole. The propagate nodes in a Meta-Pi architecture use the gates of the gating networks to propagate a weighted sum of the output vectors of the experts or cluster of experts to higher levels.

The Meta-Pi network uses a squared error cost-function which is to be minimized. The error of the whole architecture on a given example is given by

$$E = \frac{1}{2} (d - \sum_i g_i y_i)^2$$

We use gradient descent and back-propagation to minimize this error-function.

## 2.3 Selection threshold

For both architectures, a selection threshold can be used by the propagate nodes to speed up the learning process. The essence of a selection threshold is that experts are only invoked when their gate is high enough, e.g. .3. This can save a lot of time, especially when many expert networks are used. The selection threshold  $ST$  is used in the following equation as an intermediate step before calculating the final gates

$$\begin{array}{lll} \text{if } (g_i > ST) & \text{then} & s_i = g_i \\ & \text{otherwise} & s_i = 0 \end{array}$$

### 3 Training samples

The modular neural architecture representing the evaluation function has to be trained with examples. In our experiments we used two methods to generate learning samples:

- Heuristic dynamic programming or reinforcement learning, using the temporal difference (TD( $\lambda$ )) method [Sutton88, Dayan94].
- For the endgame of backgammon we also generated learning samples from a dynamic programming program.

The latter method constructs explicitly learning samples of the evaluation function for each position, which can be used in a supervised learning procedure. However, the method is computationally expensive since all states have to be evaluated repeatedly.

## 4 Experiments

### 4.1 Tic-Tac-Toe

#### 4.1.1 Experiment Design

In the first experiment the architectures are compared in learning (by TD method) to play tic-tac-toe against a fixed opponent. Identical to the TTT knowledge base used by Boyan [Boyan92] the opponent used the following rules:

- 1) IF a given move wins the game THEN play this move.
- 2) ELSE IF a given move for the opponent would win the game, THEN block this move by playing on this field.
- 3) ELSE play a random move.

The maximal obtainable performance when playing against TTT is about 0.614. This performance (match-equity) stands for (wins - losses)/games when one plays a match of a large number of games with both white and black against the opponent TTT.

When a position has to be evaluated by the networks in which black has to move, the colors are inverted. The advantage of this is that the input vector implicitly encodes the fact that the opponent has to move next. The input of the networks consists of 9 units : each unit encodes one field. The activation of an input unit will be : +1 for a circle (white), -1 for a cross (black) and 0 for an empty field. The output  $V$  of the networks lies between  $+1 \iff P(\text{white wins}) = 100\%$  and  $-1 \iff P(\text{white loses}) = 100\%$ .

In order to speed up learning, expert networks were trained with extended back-propagation [Sperduti92], which multiplies the input of each neuron with an adaptive "neuron sensitivity", which are initially set to a high value (3.0) so that the activation functions become steeper.

We studied architectures consisting of single neural networks with 30, 50 and 80 hidden units. For the hierarchical approaches about the same number of parameters was used as the 80 hidden units single network. The chosen HME and Meta-Pi architectures used only two expert networks, because larger hierarchies did not perform better. We also used an architecture which uses symbolic rules to select an expert network. The symbolic rules select for each move a different expert network to evaluate that move. Finally we used lookup tables to store the evaluation of each position in a different table-entry.

#### 4.1.2 Results and Discussion

Table 1 shows that larger single networks performed better than smaller single networks. The largest single network with 80 hidden units performed a little bit better than the HME

architectures, but this difference is not significant and the HME architecture with a selection threshold of 0.3 saves a lot of time by not invoking one of its experts in 98% of the times!

Architecture	h.u.	equity	SD	time
Single	30	0.526	.030	51m
Single	50	0.577	.014	85m
Single	80	0.600	.010	136m
HME ST=.0	2*40	0.592	.011	138m
HME ST=.3	2*40	0.592	.009	78m
Meta-Pi ST=.0	2*40	0.588	.011	138m
Meta-Pi ST=.3	2*40	0.587	.016	78m
Symbolic	9*20	0.590	.013	35m
Symbolic	9*30	0.598	.004	51m
Lookup	4560*	0.606	.007	102m

Table 1: The average maximal match equities obtained by the single networks and hierarchical architectures. Simulations were repeated 10 times, the required time for one simulation is given in minutes.

The results show that architectures with a lot of parameters perform better than smaller architectures. The selection threshold is an efficient method to use more parameters without decreasing the propagating speed of the architecture. The hierarchical architectures which uses symbolic rules to select a different expert network for evaluating the merits of playing different moves gives very good performance. These architectures have a lot of parameters, but each time only one expert needs to be invoked so they perform very fast.

The lookup table used about 4560 tables to store all evaluations of the states it had visited. Thus it needs much more parameters than the neural networks, but it obtains the best performance.

Almost all simulations reached a match equity which proximates the maximal obtainable match equity of 0.614. When we compare these results with [Boyan92], who attained a maximal match equity of 0.474, they are impressive. Except for a different input encoding, the use of large neuron sensitivities could be the reason for the difference between the results. The activation functions with steep slopes have less problems with learning discontinuities than normal sigmoids have. This was also confirmed by our obtained results when we trained the networks on a discontinuous function.

## 4.2 The Endgame of Backgammon

### 4.2.1 Experimental Design

In this section we used the endgame (bear-off) of backgammon to compare TD learning to supervised learning. The endgame considers all positions with a maximum of 14 against 14 stones in the two home-tables. This makes a total of  $1.5 * 10^9$  positions.

The networks used 68 inputs, 56 inputs encode the possible fields 0-6 where the stones for both players are allowed to stand. The maximal number of stones that can be on a particular field is 14, and so the number of stones on a field is binary encoded by 4 inputs. For the other 12 inputs some features are used which are hard to learn for the network itself.

We compared temporal difference learning with supervised learning. For supervised learning, perfect learning samples consisting of the evaluation  $V$  for a given state  $\vec{x}$  are generated by dynamic programming. Two learning set sizes have been studied: 500 samples and 5000 samples. The architecture is trained on the learning sets for 800000 iterations.

An experiment with temporal difference learning consisted of 130000 games of self-play starting with randomly drawn positions in the endgame. We also studied using a combina-

tion of TD and supervised learning. For this, TD learning on 100 games was interchanged with supervised learning on 500 perfect examples during a simulation.

For both learning strategies, an independent test-set of 5000 randomly drawn examples generated by dynamic programming was used. After every 40000 iterations, the RMS error was computed. The lowest obtained RMS error was recorded as the final result of a simulation.

After a coarse search through the parameter space, we decided to keep a single network with 10 hidden units. Neuron sensitivities were initialized at 1.0.

#### 4.2.2 Results and Discussion

Method	nr games or epochs	RMS	SD
500 examples	1600	0.101	0.006
5000 examples	160	0.044	0.002
TD learning	130000	0.080	0.002
TD + 500 examples	65000 + 800	0.068	0.003

Table 2: The results of training a single network with 10 hidden units on supervised learning and TD learning. The method specifies if supervised learning is used, TD learning is used, or a mixture of TD and supervised learning is used. The simulations were repeated 5 times.

Table 2 shows that using a large set of learning samples works best. Supervised learning on 500 examples has a much higher RMS; good generalization performance cannot be attained with so few training examples. In figure 2 we can see that overtraining occurs.

This problem does not occur with TD learning or the mixture of TD learning and supervised learning. Both methods obtain better results than supervised learning on the small learning set.

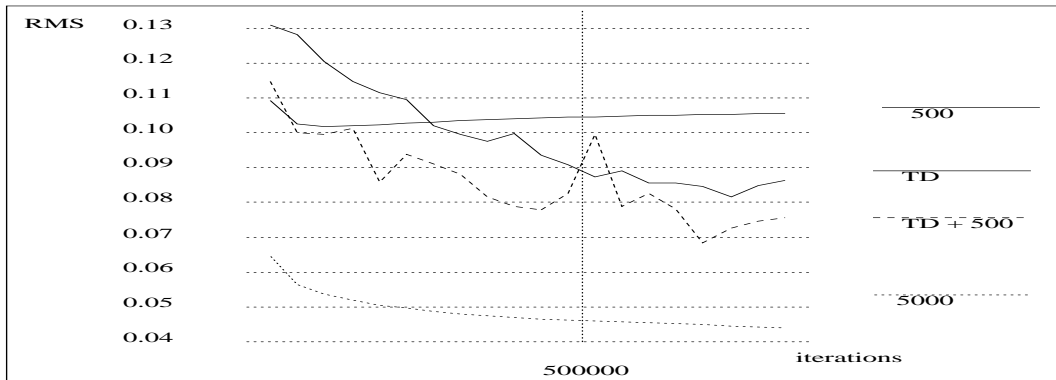


Figure 2: The learning curves for a single network with 10 hidden units when the following learning methods are used : supervised learning on 500 and 5000 perfect examples, TD learning and a mixture of TD and supervised learning. The learning curves are averaged over 5 simulations.

## 5 Conclusion

We have presented two learning algorithms for task decomposition. These methodologies are combined with temporal difference learning to be able to learn game evaluation functions

with modular neural network architectures. Results show that TD learning is an efficient way to learn a control policy for an agent. The obtained results show that by using architectures which contain many adjustable parameters, we can get very close to the maximal performance of playing tic-tac-toe against a fixed imperfect opponent. When more parameters are used in an architecture, the learning performance improves. Using multiple expert networks is an efficient way to use many parameters, because we can select independent neural networks for evaluating different moves and positions, which is much faster than always invoking one large single network.

Experiments with the endgame of backgammon show that TD learning is a viable alternative for learning on a training set. Overtraining of the architectures does not occur, and it is to be expected that the generalization error will gradually decrease. The results also showed that a combination of TD and supervised learning can be advantageous.

In the future, we want to research the efficiency of TD learning and modular neural network architectures to learn to play more difficult games, so that the true power of this combination can be validated.

## 6 Acknowledgements

Thanks to Jan Wortelboer for placing enough computer power at our disposal. Also thanks to Patrick v.d. Smagt for introducing graphic tools and thanks to Sander Bosman, Joris v. Dam, Anuj Dev and Gerard Schram for helping in one way or the other.

## References

- [Boyan92] J. Boyan. *Modular Neural Networks for learning Context-Dependent Game Strategies*. Thesis report B.S. , University of Chicago, 1992.
- [Cybenko89] G. Cybenko. Approximation by Superpositions of a Sigmoidal Function. *Math. Control Signals Systems*, 2, (303-314), 1989.
- [Dayan94] P. Dayan & T.J. Sejnowski. TD( $\lambda$ ) Converges with Probability 1. *Machine Learning*, 14, (295-301), 1994.
- [Hampshire89] J.B. Hampshire & A. Waibel. *The Meta-Pi network: Building distributed knowledge representations for robust pattern recognition*. Tech. report CMU-CS-89-166, Carnegie Mellon University, August 1989.
- [Jordan92] M.I. Jordan & R.A. Jacobs. Hierarchies of adaptive experts. In J. Moody, S. Hanson & R. Lippmann (eds.), *Advances in Neural Information Processing Systems*, 4, (985-993), San Mateo, CA: Morgan Kaufmann, 1992.
- [Schraudolph94] N.N. Schraudolph, P. Dayan, T.J. Sejnowski : Temporal difference learning of Position evaluation in the Game of Go. In J.D. Cowan, G. Tesauro & J. Alspector (eds.), *Advances in Neural Information Processing*, 6, San Francisco, Morgan Kaufmann, 1994.
- [Sperduti92] A. Sperduti. *Speed Up Learning and Network Optimization With Extended Back Propagation*. Tech. report TR-10/92, University of Pisa, May 1992.
- [Sutton88] R. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3, (9-44), 1988.
- [Tesauro92] G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8(3/4), (257-277), Kluwer Academic Publishers, May 1992.