

Training a Back-Propagation Network with Temporal Difference Learning and a database for the board game Pentec

Valentijn Muijers

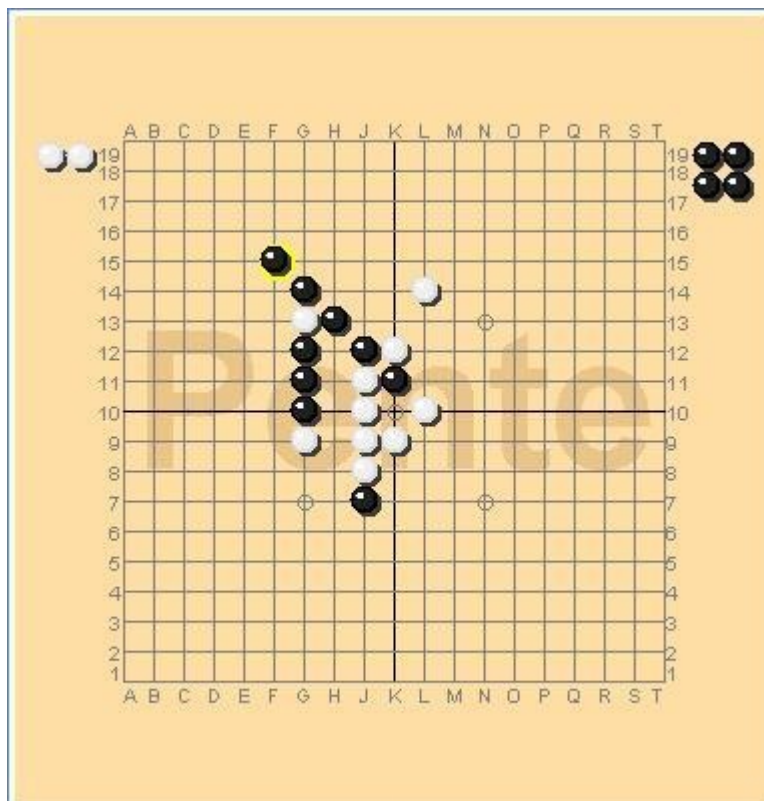
3275183

Valentijn.Muijers@phil.uu.nl

Supervisor: Gerard Vreeswijk

7,5 ECTS

22 januari 2011



Abstract

This paper will give a view on how to make a back-propagation network which can play Pente by learning from database games with temporal difference learning. The aspects of temporal difference learning and neural networks are described. The method of how a neural network can learn to play the game Pente is also explained. An experiment is described and analyzed and shows how the neural network has learned the game of Pente and in what kind of behavior the learning strategy resulted. The best performance after training on one single game was reached with a learning rate of 0.001. After learning 300000 games from the database the network did not perform very well. A third experiment with 4000 self-play games was also conducted and showed that the network placed the game more to the middle of the board than to the sides which is some form of intelligence. First this paper will give a view of the aspects of temporal difference learning in neural networks and which choices in this project were made during testing. Then the game of Pente will be explained and the way of how this game can be learned by a neural network will be justified. At the end of this paper the experiments of this project will be explained and the results will be evaluated. Then the conclusions will be discussed and analyzed.

Contents

1. Introduction	4
2. Relevance for Artificial Intelligence	4
3. Temporal Difference Learning in a Back-propagation Network	5
3.1 How can a neural network play a board game?	5
3.2 Feed-forward Neural networks	5
3.3 Back-propagation Neural Networks	6
3.4 Temporal Difference Learning	6
3.5 Ways of Learning	7
3.6 Features	8
4. Pente	9
4.1 Why Pente?	9
4.2 Rules	9
4.3 Representation	9
4.4 Database Games	10
4.5 Self Play	10
5. Experiments and Analysis	11
6. Discussion and Possible Improvements	14
7. Conclusion And Future Work	16
8. References	17

1. Introduction

In order to make a computer play a board game it is useful for the computer to have a way to play the board game. Therefore the computer needs a way to learn the game and to become a stronger player after playing a good amount of games. A way of learning a board game is by observing a lot of already played games from a database. Another way of learning is by playing a lot of games against another opponent or against oneself. The way of learning a game to a computer is similar to how a human being would learn to play a game. By watching the moves played in a game and remembering the outcome of the game a human player can use this information in later games. If a human player would see a board position which the player remembers as having a good outcome the player will try to get to this position more often in order to win more games. In the same way the player will try to avoid board positions which the player remembers as having a bad outcome. The goal of this paper is to show this naturally intuitive way of learning by a computer which learns with a neural network and temporal difference learning. Also this paper will show how this works for the game Pente and how a network can improve generalizing from other games.

2. Relevance for Artificial Intelligence

Since Tesauro (1995) [1] introduced temporal difference learning for backgammon, a lot of research has been done in the field of machine learning combining these temporal difference methods with neural networks[1,2,3,4,5]. This has lead to many board game playing machines especially for games which have a large search space. Pente also has a very large branching factor and is therefore an excellent subject to test the temporal difference learning method on. From an artificial intelligence point of view it is interesting to see whether or not the method can be applied to this game and how well the network can learn to play the game. There is very little research done on Pente in general and no research done on Pente learning with a neural network. Therefore this project could give some insight in how Pente can be learned by artificial intelligent machines and whether it actually is a good approach to use neural networks for this game.

3. Temporal Difference Learning in a Back-propagation Network

In the next section an overview will be given of how the learning of a back propagation network in combination with temporal difference learning works.

3.1 How can a neural network play a board game?

The goal of this project is to make a computer application which can play a game of Pente by training a neural network. The general idea for the computer to select a move given a board state is to first evaluate all possible successive board positions of the current position and then select the successive board position with the highest evaluation value. This board position is most likely to lead to a good outcome of the game for the player (i.e. the computer). In order to make a good player it is necessary to have a good evaluation function. A neural network can be used to approximate such an evaluation function.

3.2 Feed-forward Neural networks

A feed-forward neural network consists of two or more layers. Every layer has an amount of nodes which represent input values. Inputs in a feed-forward neural network can only be evaluated in a one directional way unlike recurrent neural networks. This means that nodes or neurons only fire to nodes in a higher layer. The layers are connected by weights. A simple neural network consists of an input layer and an output layer. The inputs of the input layer are passed through the weights to get the inputs for the output layer. A two layer neural network can approximate linear functions but in order to make a good evaluation function we need an extra layer, a so called hidden layer. This is used by a so called Multi Layer Perceptrons (MLP) network [7]. A neural network with hidden layers can approximate almost any function and can therefore be very useful to evaluate board positions. In this project a neural network with three layers was used, one input layer, one hidden layer and one output layer. The input layer and the output layer have only input values where as the hidden layer has an input value and an output value.

The hidden layer makes use of an activation function or sigmoid function to map the input of the hidden layer to the output of the hidden layer. The used sigmoid function for a hidden input node H_j is $\text{sig}(H_j) = 1/(1+e^{-H_j})$ which gives an output between 0 and 1.

The input of a node H_j in the hidden layer is the sum of the input nodes times their weights connecting the input node and the node in the hidden layer:

$$H_j = \text{sum}(I_i * W_{ij})$$

where I_i is the input of node i in the input layer and W_{ij} is the weight between input node i and a input node in the hidden layer j .

Since the network has to give an evaluation value for a given board position, the output layer has only one node which is calculated by the sum of the hidden nodes times the weight connecting each individual node and the output node.

$$\text{Output } o = \text{sum}(\text{sig}(H_j) * W_{jo})$$

where $\text{sig}(H_j)$ is the output of the hidden node H_j in the hidden layer and W_{jo} is the weight from H_j to the input node in the output layer. The output o at a given time t is $V(t)$.

3.3 Back-propagation Neural Networks

Back-propagation is a supervised learning method. This means that given a target value, the network will try to give the same output as this target. At first all the weights of the network are initialized with random numbers between -0.5 and 0.5. In order to get to the target value the network has to adjust the weights after each observation pattern (an observation pattern is a board position at a given time t , where t is the number of turns). This is done by computing the error i.e. the difference between the target or desired value at time t and the output value of the network at time t .

$$E(t) = D(t) - V(t)$$

where $D(t)$ is the desired value at time t and $V(t)$ is the output at time t . $D(t)$ for the last board position is the same as the game result. In this project $D(t\text{-end})$ is 1 if the game result is a win for the network and -1 if the network lost. After calculating the error the weights can be adjusted using a weight update rule. To compute the total network error after a series of board positions (i.e. a whole played game), sum all the squares of the individual errors. In order for the network to learn a given sequence of patterns this error should converge to zero.

$$\text{Total Error} = 0.5 * \text{sum}(E(t) ^2)$$

In board games it is sometimes hard to predict what the target value of a given board position is at that given time and therefore an algorithm is needed to predict this target value. This algorithm is called temporal difference learning and can be combined with back-propagation networks to learn the game.

3.4 Temporal Difference Learning

Given a sequence of observations (board positions), it can be difficult to immediately see which moves were good and which were bad. Therefore a temporal difference is used to give every state a temporal difference credit. This means that the value of the state depends on later states in the played game. The desired value of a board position at time t is $D(t)$ [5]. Note that the $D(t)$ for the final board position of a sequence is the same as the game outcome and all other desired values depend on this value.

$$D(t) = \text{lambda} * D(t+1) + \text{alpha} * ((1 - \text{lambda}) * (R(t) + \text{gamma} * V(t+1) - V(t)))$$

Where $D(t+1)$ is the desired value of the next state, alpha is the learning rate, gamma is the decaying factor, $V(t+1)$ is the value of the next state and $V(t)$ is the value of the

current state, $R(t)$ is the direct reward at state t given by decay from the resulting state, and λ is a factor between zero and one indicating how much feedback the desired state t gets from future states.

If λ equals one, every desired state is the same as the final desired state i.e. the game result. If λ is zero a desired value for a state t will receive no feedback from future desired states.

By combining the back-propagation algorithm and the temporal difference learning method an update rule for the weights can be calculated.

For weights between the hidden layer and the output layer, the update rule is as follows[5]: $\mathbf{W}_{jo-new} = \mathbf{W}_{jo-old} + \alpha * \mathbf{E}(t) * \mathbf{F}(\mathbf{H}_j)$

For weights between the input layer and the hidden layer, the update rule is as follows

$$\mathbf{W}_{ij-new} = \mathbf{W}_{ij-old} + \alpha * (\mathbf{E}(t) * \mathbf{W}_{jo} * \mathbf{F}'(\mathbf{H}_j)) * \mathbf{l}_i$$

Where $F'(x)$ is the derivative of the sigmoid function $F(x)$.

In the case that $F(x) = 1/(1+e^{-x})$, the derivative is $F'(x) = F(x) * (1-F(x))$.

3.5 Ways of Learning

There are two ways of learning, the first one is to first approach is to process all the information of a sequence of patterns and then update all the weights at once at the end. This is called Batch-learning [6]. The second approach is to observe a pattern and then update the weights repeating this process until all patterns of a sequence have been processed. This is called Incremental learning. In this project the batch-learning method was used, because it is useful to wait with updating until the evaluation of the end state is known of a played game because the end state determines all previous desired state values..

Then there is a distinction in offline and online learning.

In offline learning all the data is accessible at all times because it is first computed and stored before updating any weights. Batch learning is always offline.

In online learning after every step the data is discarded and the weights are updated. Incremental learning can be either online or offline. In this project offline learning was used since batch learning is always offline. The idea of learning is now in such a way that first of all the board positions of a game are observed and for every position an output is calculated. Then after these output calculations, the desired value for every single board position is calculated starting at the ending position. Since the desired values are dependant of the outcome of the game. After these values have been processed, the updating can begin. All the delta-updates for the individual weights are summed over the board positions and updated at the end.

3.6 Features

In most board games there are certain positions or patterns which are more likely to result in a win for one of the players. When a player can get to these patterns the player is considered a better player since these patterns indirectly make the player win the game. These patterns are described by features. Features denote certain advantages or disadvantages of a given board position. When learning a game, the programmer can give the network a few extra input nodes which denote features. When the network learns a lot of games, the network will come to understand that these features are important aspects of the game in order to win. A way of making sure a network will become an intermediate player, it is often given some features to give a better understanding of the game. Though this is not always the best way to learn, because there may be patterns which humans may not easily observe throughout the games. By letting a network explore a lot of games by itself (i.e. self play), a network can explore the features of the game for itself and may become an even stronger player[1]. In this project the raw board material was used (i.e. how many stones are on the board from either side) and how many white and black stones were captured. These two features are the most basic features in the game. The idea behind this was that first the network could learn from database games with these two basic features and then discover more features of the game by playing against itself.

4. Pente

The goal in this project was to learn a computer to play Pente. Pente is a strategic board game created in 1977 by Gary Gabrel [8]. It is an Americanized variation on the Japanese board game ninuki-renju, which is a variation of renju or gomoku (connect-five).

4.1 Why Pente?

Pente is an easy to learn and hard to master kind of game. This means that the rules are simple but the game can be quite challenging. The game is played on a 19 x 19 nodes board which means that for normal game tree search techniques the branching factor would be too large to calculate in a reasonable amount of time. Neural networks on the other hand can converge to an evaluation function which can be used to give values to board positions. This evaluating of a given board position takes less computation than a game tree search and therefore a neural network would be a good approach to learn Pente. Another advantage of neural networks is that they can generalize. This means that if the network encounters a board position which he has never encountered before during training, the network can still give an accurate evaluation value to the board position when it was trained on similar board positions.

4.2 Rules

Pente is played on a 19 x 19 nodes board. On every ply (a turn taken by one player) a player places one of their stones on one of the 19 x 19 nodes of the board. The goal of the game for a player is to connect five stones (or more) of their colour (vertically, diagonally or horizontally). An additional rule is that it is possible to capture stones of the opponent by placing stones of your colour at the end of two adjacent stones of the opponent. The two stones between your two surrounding stones are captured and removed from the board. It is not possible to sacrifice your own stones by placing it between two opposing surrounding stones in the way described above. An alternative way to win the game is to capture five pairs of stones of the opposing player. Pente is a very competitive game and to make it more balanced for either player, the opening move is always at the very centre of the board. After this move the second player may place his first stone anywhere on the board. Then the first player has to place his second stone anywhere on the board except for a 3x3 square around the middle. After these opening restriction both players can place their stones on any free node on the board.

4.3 Representation

To evaluate a board position an input must be given to the neural network. This input is a representation of the board position to be evaluated. In the neural network which was used in this project, an input exists of 364 nodes which represent the game and 1 bias node (365 in total). Every single node on the board is given a value: 1 if the node is occupied by a stone of the player's colour, -1 if the node is occupied by a stone of the opposing player and 0 if the node is free. There are 19x19 nodes for a give board position so 361 nodes represent the board in this way. There are three additional nodes: one for the amount of white stones captured divided by ten, one for the amount of black stone

captured divided by ten (this gives a value ranging from 0 to 1) and one to show which player it is to make a move (1 for the player, -1 otherwise).

4.4 Database Games

There are many ways to make a neural network learn to play a game. One of them is by showing the network a lot of already played games, i.e. database games. The network learns from these games and then uses the learned experience to determine evaluations of new board positions. The advantage of learning from database games is that it is a lot faster (i.e. less games need to be played) than by learning the game from selfplay. A disadvantage on the other hand is that after learning from the database games, the network is as good as the learned games. So if the learned games are not of high performance quality, the network also won't be a challenging opponent. Wiering and Patist [4] showed that it takes less time to learn from a database and that it is better than learning from a random move player.

The database which was used in this project was taken from Pente.org. All the games were selected from tournaments played in 2000, 2001, 2002 and 2003. Since all the games were played on the internet, a lot of the games were not fully played games i.e. a player left before the game had ended. To make sure that at least a possible interesting game was played, all games with less than 9 moves were not evaluated since it would not be possible for a game to end in 9 or less moves.

4.5 Self Play

Self play is a very intuitive way of playing a game and may lead to great results. Tesauro's Backgammon [1] network is a good example of this strategy for the game of Backgammon. Tesauro managed to make a neural network that could compete with experts. For backgammon Tesauro showed that the network could play at expert level after playing 1000000 games against itself, but this is not necessarily true for every game. Backgammon is a stochastic game which means that there is an element of chance involved i.e. dice. Pente is a deterministic game which means that there is no chance what so ever. Both backgammon and pente have full information for both players which means that all the information in the game is on the board and fully observable and known to both players at any time in the game. Since Pente is a deterministic game it could be that learning from self play could lead to local optimums, since every game is nearly the same as the last game with just a slight difference. This means that for Pente to learn from self play a lot more played games are needed. To speed things up a little, it is possible to make a network that first learns from database games and then progresses further by playing against itself. A combination of database games and self play is useful because the database games can give the network an idea of how the game is played and after this initial learning the network can explore for itself what the good and bad strategies are.

5. Experiments and Analysis

To get a good idea of finding out whether Pente can be learned by a neural network, an experiment is needed. The experiment's goal is to show what a neural network is capable of after learning from database games as opposed to learning from self play. In this section three experiments were set up: The first experiment was merely to test whether the implemented network could actually learn something and at the same time to test what the best value for the learning rate was regarding to Pente.

The idea of the experiment in this project was to first let the neural network learn from approximately 3000 different games and show every one of those 3000 database game 100 times, so that in total the network would have learned from 300.000 games. This gives about the same results as learning from 300.000 different games [12] .

First the network was tested to learn from just one game to see if the network actually succeeds in learning just one sequence of board positions. The network learned the game 10.000 times in a row with different values for alpha and lambda set to 0.8[4]. The hidden layer was set to 80 units and the gamma factor was set to 0.9 [3].

10.000 games	Games needed before convergence to total error less than 0.1	Total Network Error after 10.000 games
Alpha = 0.1	Did not converge	2.777778
Alpha = 0.01	Did not converge	2.7761912
Alpha = 0.001	1740	0.011060873
Alpha = 0.0001	Did not converge	0.19699441

The peculiar thing was that after only a few cycles of games the weights diverged to extremely large values i.e. both positive and negative. But then after 100 or so more cycles the weights began to convert again so that the total network error per game dropped from a very large number to around three. An interesting observation was that the network had learned to get quite a good result (i.e. a low error) by always trying to give an output close to zero for every board position. This works fine for the first ten or so moves the game had to offer but the error for the final board position was the same as the game result. This can be explained because the output was about zero and so the error was the same as the game result because the error is the difference between the target and the output. The conclusion which can be drawn from these observations is that it is necessary to have a very small (i.e. 0.001) parameter for the learning rate or the network won't be able to get a good grasp of the game at hand. The learning should also not be too small or otherwise it takes too much games for the network to converge.

With these conclusions a second experiment emerged: to train on a larger amount of different games with a lambda value of 0.8 and a set learning rate of 0.005 and a decay rate of 0.9. The network learns from a database of 3000 different games and every game is taken into account a hundred times, which gives a total learning database of 300000 games.

To learn from 300000 database games the network needed approximately three hours. After the learning session the network was tested to play against itself. This showed that

the network at the beginning of a match played considerably random. An interesting thing to note was that the network often chose to make a move in the corner of the board, which is generally a bad move. The corner moves in the trained database are non-existent which should have lead to not-choosing such moves. But since the evaluation function doesn't know anything about these corner-moves he won't consider them as bad moves and apparently gives them a high evaluation score. But since testing it by playing against itself it does show whether the network actually understands anything of the game at hand, which in this case was very little.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
A	-	-	o	-	-	o	o	-	-	-	o	-	-	o	-	-	-	-	o
B	o	o	o	-	-	-	-	-	-	-	-	-	-	-	-	-	-	o	-
C	o	-	-	-	-	-	-	-	-	-	-	-	-	-	-	o	o	o	o
D	-	-	o	-	-	-	-	-	-	-	-	-	-	-	o	o	-	-	-
E	-	-	-	-	-	-	-	-	o	-	-	-	-	-	x	-	x	-	-
F	-	-	x	x	-	o	x	-	-	-	-	x	-	-	x	x	-	-	-
G	-	-	-	-	-	-	-	-	-	x	-	-	x	-	-	-	x	-	x
H	-	-	-	-	o	x	-	-	-	-	-	-	-	-	-	-	-	x	-
J	-	-	-	-	-	-	o	x	x	-	x	-	-	-	x	-	-	x	x
K	o	-	-	-	-	x	o	x	-	x	-	-	-	-	-	-	-	-	-
L	x	-	x	-	x	-	-	-	-	x	-	x	-	-	-	-	-	-	-
M	-	-	-	x	-	-	-	-	-	-	-	-	-	-	-	-	-	-	x
N	o	-	-	x	-	x	-	-	-	-	x	-	-	-	-	-	-	-	-
O	-	x	-	-	x	-	x	-	-	-	-	-	-	-	-	-	-	-	-
P	x	x	-	-	-	-	-	-	-	-	-	-	o	-	-	-	o	-	-
Q	-	-	-	o	-	o	-	-	-	-	-	-	-	o	-	-	o	-	-
R	-	-	-	o	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
S	-	o	-	-	-	-	-	-	-	-	-	-	-	o	-	-	o	o	-
T	o	-	-	-	-	-	-	o	-	x	-	-	-	-	-	-	-	-	-

White pawns captured 0.0
 Black pawns captured 0.0
 Games won by 'x' is: 1
 Games won by 'o' is: 0
 Learned from: 1 Games

Figure 1. A game from two database players after 300000 games of training

After the disappointing results from the database network, a third experiment emerged: Training from scratch with self-play. Of course this would take way longer to learn for the network because every game first has to be played and then learned. But it would be interesting to see whether the network would show different behavior in playing and hopefully have a little more understanding of the game. Hypothetically it would take many more games to learn from than from a database, but it could be interesting to see the placement of the stones for either player and whether the network could learn that another player is trying to stop him from winning. The same parameters as in the last experiment were used, but now the network learned from 4000 games against itself. After

seventeen hours of learning the network showed that of the 4000 played games 2836 games were won by the black player (the player which starts), and 1164 games were won by the white player. From this the observation can be made that with random play it is more advantageous to be the player that makes the first move. An interesting thing to note was that in waves the a player would find a little advantage in the game and exploit this feature to win the game many times in a row before the opposing player would learn how to defend against the strategy. Though only 4000 games were learned which is not quite enough to learn from self-play, it is interesting to note that there is more stones in the middle of the board than to the sides and corners of the board. In a real game the most stones are played in the middle of the game because the first stone is placed in the middle and this gives the starting player an advantage.

```

      1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19
A  o  -  -  -  -  -  -  -  x  o  x  x  -  o  -  -  -  -  -
B  o  -  -  -  -  -  -  -  -  -  o  o  -  x  -  -  -  -  -
C  o  x  -  -  -  -  -  -  o  x  -  -  -  x  o  o  -  x  o
D  x  x  -  o  -  -  -  -  -  -  -  x  -  -  -  -  x  -  -
E  -  o  -  -  -  x  -  -  -  -  -  x  -  -  x  x  o  -  -
F  x  -  x  -  -  -  x  -  x  -  o  -  o  -  x  -  o  -  -
G  x  -  -  -  -  -  -  -  -  o  x  -  -  x  -  x  o  o  x
H  x  o  x  x  -  -  -  o  o  o  -  -  o  o  -  x  x  o  x
J  -  o  x  -  x  -  o  -  x  x  -  -  o  -  x  -  -  -  -
K  -  -  x  -  o  -  x  x  -  x  x  -  -  o  -  -  o  -  o
L  -  -  -  -  -  -  x  -  o  o  -  x  o  -  -  -  -  o  -
M  x  -  -  -  -  -  o  o  o  -  x  -  -  -  -  -  -  -
N  -  x  -  -  x  x  -  -  o  -  o  -  x  o  x  o  x  -  o
O  o  -  -  -  -  o  -  -  -  -  o  -  -  -  o  o  -  -  x
P  o  -  -  x  -  o  -  -  -  x  o  o  x  -  x  x  -  -  -
Q  x  -  o  -  -  -  -  -  -  x  -  -  -  o  -  -  -  -
R  -  -  -  -  -  -  -  o  -  x  -  -  -  o  -  -  o  -  x
S  -  -  x  -  x  x  -  x  -  -  -  -  -  -  -  -  x  x  -
T  -  -  -  o  o  -  o  o  -  -  -  -  -  -  -  x  -  o  x  -
White pawns captured 4.0
Black pawns captured 0.0
De winnaar is x!

```

Figure 2. A game from two self-play networks after 4000 games of training

6. Discussion and Possible Improvements

It is interesting to note that the networks described above did not learn the game as well as was expected. Since little research is done on Pente in general [14] and no research in Pente combined with temporal difference learning, it is hard to establish the problems of playing. In this section several reasons are discussed and analyzed to explain the level of play from the trained networks in this paper, generalizing from other games.

Jan Peter Patist and Marco Wiering (2004)[4] found that after training from 200000 database games of Draughts, the trained neural network gained an intermediate level of play i.e. it could beat a good draughts AI and sometimes drew a very good draughts AI. An interesting thing to note is that the network was trained on 200000 different games, in contrast to the experiment described in this paper which taught a neural network from 3000 different games but showing them repeatedly. For the experiment the author assumed this would lead to the same results as learning from 300000 different games. Ragg[12] described that learning from fewer games more than once gives the same result, at least for the game of Nine Men's Morris. Since Nine Men's Morris is far less more complex than Pente (only 24 board positions), the question remains whether this efficiency rule holds for Pente as well. Arguably there are a lot more board positions in Pente and with fewer different games, the amount of board positions is not as greatly explored as with 300000 games. This could lead to a less accurate evaluation function for board positions which are not found in the database, but better accuracy for the board positions which are in the database since the learning rate of the network is slow and seeing the same board positions more often would lead to a better understanding of these positions.

A second interesting aspect which was found in by Patist and Wiering[4], was that the order in which the database games are learned, at least for draughts, did not increase the level of play for the trained network. They showed the network three types of games, with opening, mid-game and late-game positions. The idea behind this was to overcome unpleasant inference. In Pente the distinction between early, mid, and late game is not as big as with draughts, but it would be interesting to see what the difference would be for a network which would learn from specific periods in the game in contrast to learning a whole game at once.

Tesauro[11] showed that learning with a 2-ply search instead of a 1-ply search drastically improved the level of play for the Backgammon network. A 3-ply network was also constructed which was even stronger than the 2-ply search although slower. A 4-ply or n-ply network would most likely give better results Tesauro argues, but decreases playability because of computation. These same rules most likely apply to Pente. A look-ahead applet made by Mark Mammel [13] used look-ahead search up to 18 moves. The amount of time it takes for up to 9-ply moves is 300 seconds per move which is very slow since there is a time clock of 600 seconds per player. A combination of look-ahead and a neural network would most likely come to very high levels of play even for Pente.

Imran Ghory[3] shows that there are many factors influencing the game learning properties. He describes divergence for board games as the difference between two successive board positions divided by the amount of possible successive board positions for a give board position. The divergence rate for chess and backgammon is low, but for

Go and Pente the divergence is exceedingly higher because of the size of the board and in which way captures can occur and change the board position rapidly. The divergence is of importance to the evaluation function in such a way that for a low divergence rate the error of an evaluation function is less important. The idea behind this is that for a low divergence rate an error which would be made by an evaluation function of a network would be the same for similar board positions which means that the network would rank these board positions the same as though there was no error. Since Pente has a medium divergence rate, the errors in the evaluation do count for a less accurate evaluation function. This could account for the fact that similar board positions in Pente are not evenly ranked by the evaluation function.

Ghory[3] also showed from an experiment with tic-tac-toe that the decay-parameter is not of great importance as long as it is less than one. The actual influence of such a parameter is different for every game, in the way the author has implemented Pente in this paper, the gamma is of influence when computing the direct reward at a given time. Since it only decays the direct reward given at a certain time to increase the desired value of a state by a very small factor, the influence of the gamma is of importance but not as influential as the lambda factor, since the lambda determines the feedback from the resulting state of a played game.

Overall, the above described factors combined are sufficient to clarify the level of play of the database network. Also most of these deficiencies could be overcome by learning from more games and more different games. Testing for different parameter settings for lambda and even increasing the amount of hidden nodes will probably lead to a better game playing neural network as well.

7. Conclusion And Future Work

This paper showed how a neural network can learn to play the game Pente by using temporal difference learning. By learning one game 10000 times in a row, the network showed that the learning rate strongly influences how well the given patterns are learned and that a learning rate should be very small (0.005) to get the best results of learning. Furthermore the network was able to train on 3000 different database games but still was not able to come up with a decent strategy. This could be improved by using two-ply search instead of one-ply search. Also the network can be improved by learning from more different games (i.e. a larger database) and by playing against itself.

A network was also trained to learn the game of Pente from scratch by self-play, after 4000 games the network learned not very much but it was remarkable that the play was more set in the middle of the board than on the sides and corners.

It could be interesting to see what happens when the neural network would learn from scratch to expert level by self play for the game Pente. Since there are many patterns and a wide variety of tactics this could lead to some insights which would not have been found before for this type of branching factor game. It would also be possible to add features to the neural network and make the network learn with those features and see how the network will improve.

8. References

- [1] Gerald Tesauro (1995) Temporal Difference Learning and TD-Gammon. Communications of the ACM 38, (<http://www.research.ibm.com/massive/tdl.html>)
- [2] Richard S. Sutton (1988) Learning to predict by the methods of temporal difference. Machine Learning 3: 9-44
- [3] Imran Ghory (2004) Reinforcement learning in board games. Department of Computer Science – Publications. 9-13, 21-30.
- [4] Jan Peter Patist and Marco Wiering (2004) Learning to Play Draughts using Temporal Difference Learning with Neural Networks and Databases. 3
- [5] Henk Mannen (2003) Learning to play chess using reinforcement learning with database games. Phil.uu.nl/preprints/ckiscripties. 11-34.
- [6] <http://www.faqs.org/faqs/ai-faq/neural-nets/part2/section-2.html> – explains the concept of batch learning
- [7] <http://www.nnwj.de/backpropagation.html> - Information on back-propagation in neural networks
- [8] Pente – Rules and Definition of Pente
- [9] Tom M. Mitchell (1997) Machine Learning. 81-108, 367-388
- [10] Pente.org- for the database games
- [11] Gerald Tesauro (2002) Programming Backgammon using self-teaching neural nets IBM Thomas J. Watson Research Center, 181-199
- [12] Thomas Ragg, Heinrich Braunn and Johannes Feulner (1994) Improving Temporal Difference Learning for Deterministic Sequential Decision Problems. Proceedings of the International Conference on Artificial Neural Networks - ICANN '95, 117-122
- [13] Mark Mammel (2002) Pente playing Applet “Pente version 10.4”
- [14] Jacob Schrum (2006) Neuro-Evolution in Multi-Player Pente, Department of Computer Sciences University of Texas at Austin