A Study of Machine Learning and Neural Networks in Strategic Games

MAGNUS JÄVERBERG

KTH Computer Science and Communication

Master of Science Thesis
Stockholm, Sweden 2007
A Study of Machine Learning and Neural Networks in Strategic Games

MAGNUS JÄVERBERG

Master’s Thesis in Computer Science (20 credits) at the School of Computer Science and Engineering
Royal Institute of Technology year 2007
Supervisor at CSC was Örjan Ekeberg
Examiner was Anders Lansner

TRITA-CSC-E 2007:094
ISRN-KTH/CSC/E--07/094--SE
ISSN-1653-5715

Royal Institute of Technology
School of Computer Science and Communication

KTH CSC
SE-100 44 Stockholm, Sweden

URL: www.csc.kth.se
Abstract

A study of machine learning and neural networks in strategic games

The domain of complex games is generally considered to be a good testing ground for learning algorithms. Such games offer a tremendous complexity at expert play level, and both the problem and performance measures are clearly defined.

One class of learning algorithms that has been thoroughly tested in this domain is artificial neural networks, ANN. For some games ANN have been found to perform remarkably well, and for others it performs very poorly. This work aims to explore the following area: Which features should a game possess in order to be suitable for neural networks and self learning?

In order to do this exploration games where ANN works well, such as backgammon, was compared with games where ANN works poorly, such as chess and go. Further, an ANN based program for the game street soccer was developed. This program proved to be rather successful, and since street soccer shares some features with chess and some features with backgammon this success may play an important role for gaining a better understanding of the suitableness of ANN for games.

Sammanfattning

En studie av maskininlärning och neurala nätverk i strategiska spel

Domänen komplexa spel betraktas generellt som ett lämpligt testområde för lärande algoritmer. Sådana spel erbjuder en oerhörd komplexitet på expertnivå, och både problemet och mått på framgång är klart definierade.

En typ av lärande algoritmer som har blivit grundligt testade inom denna domän är artificiella neurala nätverk, ANN. För vissa spel har ANN visat sig mycket lämpliga, och för andra mindre lämpliga. Detta arbete syftar till att utforska följande område: Vilka egenskaper bör ett spel inneha för att vara lämpligt för neurala nätverk och självinlärning.

För att göra denna utforskning kommer spel där ANN lämpar sig väl, så som backgammon, att jämföras med spel där ANN lämpar sig sämre, så som schack och go. Vidare blev ett ANN baserad program för street soccer utvecklat. Detta program visade sig vara relativt framgångsrikt, och eftersom street soccer har vissa egenskaper gemensamt med schack och vissa egenskaper gemensamt med backgammon kan denna framgång vara en viktig del i en bättre förståelse av lämpligheten av ANN till spel.
Background........................................................................................................................................1
  Short introduction to ANN ................................................................................................................2
    Training ........................................................................................................................................3
Method..............................................................................................................................................5
  Street soccer, a short introduction .....................................................................................................5
    Rules ..........................................................................................................................................5
    Symmetries .................................................................................................................................6
Simple soccer ....................................................................................................................................7
  Rules of simple soccer ......................................................................................................................7
  An exact solution ...............................................................................................................................7
Less simple soccer ...............................................................................................................................8
Preparations for an ANN solution .........................................................................................................8
  Number of hidden layers ...................................................................................................................9
  Number of hidden neurons ...............................................................................................................9
  Input representation .........................................................................................................................9
  Training method ..............................................................................................................................9
  Over learning ................................................................................................................................9
  Measures of performance in simple soccer ....................................................................................10
Short presentation of the experiments ...............................................................................................10
Results ............................................................................................................................................11
  Simple Soccer .................................................................................................................................11
    Experiment 1 .................................................................................................................................11
    Experiment 2 .................................................................................................................................12
    Experiment 3 .................................................................................................................................14
    Experiment 4 .................................................................................................................................16
Less simple soccer .............................................................................................................................18
  Experiment 5 .................................................................................................................................18
  Experiment 6 .................................................................................................................................19
Discussion........................................................................................................................................21
Background

In the early days of AI learning programs such as Samuels checker playing program [1] (1959) and Donald Michie’s MENACE tic-tac-toe learner [2] achieved considerable attention. Later the interest for learning programs diminished when it became obvious that such programs could be outperformed by hand made programs, which due to faster evaluation algorithms could search deeper in the game trees and thereby obtained a stronger level of play.

This suddenly changed in 1992, when Gerry Tesauro wrote the remarkably successful backgammon playing program Neurogammon, based on expert learning and artificial neural networks (ANN1). In 1992 it won the First Computer Olympiad with a perfect score of 5 wins and no loss, thereby becoming the first learning program ever to win any tournament [3].

The success of Neurogammon inspired Tesauro to try a new training method called TD(λ), developed a few years earlier by Sutton[4], resulting in TD-gammon.

The main advantage of TD(λ) is that it enables the network to learn by self play, without taking advantage of possibly erroneous human domain specific knowledge. In TD(λ) each position reached during self play is trained towards the value of either the succeeding position (TD(0)), the final position (TD(1)), or something in between (TD(λ), where 0≤λ≤1). The intuitive justification is that the closer a position is to the final position the more precisely it can be evaluated.

The indata in the first versions of TD-gammon was raw board data, and later versions used a hand crafted set of precalculated features as well. As output the ANN gave an estimation of the equality2 of the position. Since TD-gammon could evaluate only positions, and not moves, it had to evaluate the resulting position of all possible moves in order to pick the best move (the move resulting in the position with highest equality).

In strength TD-gammon greatly surpassed Neurogammon, and was very close to the very best human players. Kit Woolsey, then the world’s fifth best player, claimed that except in a few minor areas TD-gammon widely surpasses all human players [5].

The successful application of TD(λ) to backgammon have lead to several attempts to apply TD(λ) to other games. Unfortunately these attempts in general have not been very successful, neither in chess[6][7], go[8] nor Othello[9]. There have been several attempts to explain this[10], and most agree that the most important reason is that the backgammon is non-deterministic.

This makes sense, since if you let a program play a deterministic game against itself it tends to play very similar matches each time. In a non-deterministic game, such as backgammon, the random element makes it very unlikely that the same game will ever be played twice. Hence, in a non-deterministic game the ANN will have the opportunity to learn from many more unique positions. By using a random setup while training an ANN in a deterministic game the problem can somewhat be reduced, but in general this is not enough.

---

1 In this work “ANN” can stand for one or several artificial neural networks.
2 Equality is a measure of goodness, commonly used in backgammon and other games. It is defined as the expected economical outcome of a game where the looser pays 1 unit to the winner. E.g. if Player A is to move in a 1 $ game, and from the current position would earn 0.5 $ in average, the position is said to have an equality of 0.5. This corresponds to a winning chance of 75 % (If you win 1 $ in 75% of the cases and lose 1 $ in the rest of the cases, you end up with 0.5 $ in average).
Another difference between backgammon and chess is how the game ends. In backgammon you can only win by removing all your checkers from the board. This is something that happens gradually, you usually remove one or two checkers at a time, and while you do this the complexity of the games is slowly reduced. The number of possible ending states in backgammon is considerably less than the number of possible mid game states, and every state close to the final state shares an easy identifiable feature: At least one of the players has only a few checkers left.

In chess the final position does usually contain fewer pieces than the start position, but there is no gradual reduction in complexity. The final position can look in many different ways, and there is no certain way to easily determine whether a position is close to the final position. Othello and go do lack this gradual reduction of complexity as well.

It is possible to argue whether this feature is of importance for an ANN. Fewer possible states in the end of the game make it easier for the ANN to gain an understanding of the endgame, and with a good understanding of an end game you have a foothold that greatly simplifies the task of understanding the mid game.

However, in order to gain a better understanding of the importance of this feature it will be tested experimentally. The tool for the current experiment is the game street soccer, a game that is neither deterministic nor has a gradual reduction of complexity. If ANN would prove to be suitable for street soccer this would suggest that the gradual reduction of complexity is a less important feature, and vice versa.

To the best of my knowledge there have been no previous attempts to apply ANN to street soccer, but a non-ANN based program, “Loco”, was developed by Ronald Lokers 2004. This program played with good result against human opponents at the online community “Little Golem” (littlegolem.net). As a part of the current work work an ANN based player was developed and tested at this community.

**Short introduction to ANN**

An artificial neural network (ANN) is a computational structure capable of function approximation and classification. The building stones of ANN are called neurons (fig. 1).

![Figure 1. A schematic overview of a neuron.](image)

Each neuron has a number of inputs and one output, and each input is associated with a parameter called weight, which is a real number. The weights and input determine the output according to eqv. 1.
\[ y = \varphi \left( \sum_{j=1}^{m} w_j x_j \right) \quad \text{Eqv. 1.} \]

Where
\( x_j \) is input \( j \),
\( w_j \) is the weight associated with input \( j \), and
\( \varphi \) is the activation function.

\[ \varphi(v) = \tanh(v) \quad \text{Eqv. 2.} \]

The choice of the activation function is based on factors such as input range, and desired output. In this work the activation in eqv. 2 is used.

An ANN consists of a set of neurons, usually divided in one or more layers where all neurons in each layer are connected to all neurons in the next layer (fig. 2). Note that the so-called input layer does not consist of neurons. To avoid ambiguity ANN:s are classified by the number of hidden layers.

![Figure 2. An example of an ANN with 4 inputs, 5 hidden neurons in one layer, and one output.](image)

With correct weights and enough neurons an ANN can approximate any function with arbitrary precision [11]. To determine the optimal weights is, however, far from trivial. Except for a few trivial cases no analytic solution is known, and we are forced to use iterative methods based on examples (sets of inputs and desired outputs). This is called supervised training.

**Training**

There are several different training algorithms. In this work I used the “backpropagation” algorithm, a well known algorithm based on gradient decent [12]. However, if there are no examples with known desired outputs available this method have to be combined with a method for unsupervised training. A popular method is TD(0):

Suppose we have a set of states, \( S \), and a Markov process, traversing \( S \), that will terminate in either state \( S_A \) or state \( S_B \). The probability that we from \( s_0 \) eventually will reach \( s_i \) is denoted...
p(s_0, s_1), and the state visited at step t is denoted s'. The goal is an ANN that is capable of estimating p(s, s_A) for any s ∈ S.

Let us start from any state (usually a random one) and for each encountered state s' ∉ {s_A, s_B} we train the network using p(s'|A) as an approximation of p(s', A). An intuitive justification is that the network first will learn the values for states close to the terminal states and then propagate this knowledge to states further away.

An alternative method for unsupervised training is Monte Carlo simulation. This gives a better estimate of p(s, a), but does also require more computation.
Method

Before developing a complete ANN based street soccer program several experiments were performed on simple soccer, a simplified version of street soccer I developed specifically for this study. Simple soccer is simple enough to allow the calculation of an exact solution, but still complex enough to be a worthy ground to try out different ANN designs. Obviously, when performing experiments it is a considerable advantage to be able to compare with a known solution.

As a second step the final ANN design from simple soccer was tested in a much more complex, but still simplified, version of street soccer. In the third, and final, step a program that can handle the complete rules was developed and tested against human opponents.

Street soccer, a short introduction

Street soccer is a modern board game, and has gained in popularity. The game is published by Cwali and can be bought in any well sorted game shop. It is also possible to play over Internet.

Rules

The rules of street soccer are reasonably simple, and are summarised below. Some of the details (such as setup rules, and anti blocking rules) are left out. For complete rules see appendix 1.

The game is played between two players, yellow and red. The purpose of the game is to score more goals than your opponent during the predefined playing time. Each player has 5 pieces, one of which is a goalkeeper. The board consists of 8x12 squares, and the playing field of 6x10 squares.

Position 1A. Yellow to move

Position 1B. Yellow to move
In the beginning of each turn (except during the setup rules, see appendix 1) the player to move rolls a die, and gets one movement point (MP) for each eye of the die. He then chooses one of his pieces to move. Each step with this piece costs one MP. Allowed steps are all none-diagonal steps that end up in an adjacent square that is either empty or contains the ball. The player keeps moving the same piece until he is either out of MP or until the ball is reached.

If the ball is reached the remaining MP plus one extra MP is transferred to the ball. The player then chooses a direction to kick the ball (horizontally, vertically or diagonally). After the ball is kicked the player can change direction of the ball once, and then only with 45 degrees (curve). For each square the ball moves it loses 1 MP. When the ball is out of MP it stops. If the ball reaches a piece of the same team it gets one extra MP and can be kicked off in any direction, with a new curve allowed. The ball is not allowed to get outside the field (except when scoring a goal), or reach a square occupied with a piece belonging to the opposite team.

A goal is scored if the ball reaches one of the two squares behind the goal. After a goal the goalkeeper kicks the ball back into the field and the game continues.

The game normally ends after 25 rounds for each player. If the score is equal the game continues until next goal, but not longer than 10 extra moves for each player.

**Symmetries**

When training ANN, symmetries can often be used to simplify the task. Street soccer has two symmetries: Reflection along the centre vertical line symmetry, and the “side swapping symmetry”.

**Reflection along the centre vertical line symmetry**

If a position is reflected along the central vertical line, the value of the position is not changed. This is reasonable obvious, and an example is shown in position 1A and 1B.
**Side swapping symmetry**

Since the board is symmetric and both players play by the same rules we also have the side swapping symmetry that exists in most two persons zero sum games. For example, the side to move has equal winning chances in the position 1A and 1C.

**Simple soccer**

The purpose of simple soccer is to be a game simple enough to allow an exact solution, but still being as similar to street soccer as possible.

![Position 2](image)

*Position 2. An example of a simple soccer position.*

**Rules of simple soccer**

- The playing field is 6x6 squares inside the lines and no squares outside
- Each player has 2 pieces, of which none is goal keeper.
- The die has three sides, with 1, 2 and 3 eyes.
- The game is started from a random position.
- First goal wins (unlimited playtime)
- All other rules are the same as in street soccer.

Rule 1 and 2 are intended to reduce the number of possible positions, and with a smaller board it makes sense with less movement (hence the three sided die). The random placement and the first goal win rule is intended to simplify the learning. With a random starting position it’s likely that a much larger number of positions will be explored, and with first goal win the ANN does not have to take the score or the amount of time remaining into account. Position 2 shows an example of a simple soccer position.

**An exact solution**

For each valid position an exact solution has to contain either an optimal action, or a value for the goodness of the position. In this case the second option is to prefer, and as a measure
of goodness I used the winning chance for yellow, assuming perfect play from both sides. From such a table it’s trivial to obtain the optimal move for a given position by comparing the goodness of all possible successive positions.

Thanks to the symmetries mentioned above we only have to include positions where yellow is to move and where the ball is on the left half of the field. If we want to know the winning chance for a position where the ball is on the right side we mirror the position, and the winning chance in position where red is to move is 1 minus the winning chance in the “side-swapped” position.

In total the table contains 3,437,446 legal and unique positions.

In order to construct the solution table I treated the problem as a standard Markov Decision Problem (MDP) and used the well-known solution algorithm “Value Iteration” [13], slightly modified to fit our problem:

Initialize \( V \) to \( V(s) = 0 \), for all \( s \in S \)

Repeat

\[
\Delta \leftarrow 0
\]

For each \( s \in S \)

\[
V \leftarrow V(s)
\]

For each \( d \in \{1, 2, \ldots, d_{\text{max}}\} \)

\[
V_d(s) \leftarrow \max_{i \in M(s, d)} (1 - V(R(s')))
\]

\[
V(s) \leftarrow \sum_d V_d(s) / d_{\text{max}}
\]

\[
\Delta \leftarrow \max(\Delta, |V - V(s)|)
\]

until \( \Delta < \theta \) (a small positive number)

Algorithm 1. A MDP solution method. \( d_{\text{max}} \) is the maximal die (3 in simple soccer), \( M(s, d) \) is the group of all states directly reachable from state \( s \) with die \( d \), and \( R(s) \) is the “side-swapped” state of \( s \).

After 48 iterations, each iterations taking approximately one hour on a 3GHz Pentium computer, an error margin of less than 1/100,000 was obtained.

**Less simple soccer**

In an attempt to establish whether the results from simple soccer are valid on a bigger board I constructed the game “less simple soccer”. This game has the same size of the playing field and same amount of pieces as street soccer, but otherwise has the same rules as simple soccer. The number of legal positions in less simple soccer is approximately \( 1 \times 10^{15} \), compared to \( 3.4 \times 10^6 \) in simple soccer. Hence, computation of an exact solution is utterly impossible.

**Preparations for an ANN solution**

Next step was to design an ANN that is capable of playing simple soccer, using the exact solution. The natural choice for input is the current state, and for output the best move. Unfortunately there is no easy way to make the ANN output a move, and we are forced to use a value of goodness instead. As goodness measure the choice fell on equality instead of
the winning chance, since equality has a range of [-1, 1] and hence gives a more balanced network [14].

**Number of hidden layers**

The by far most common choice is one hidden layer, fully connected with both the input and the output layer. I could not find any compelling reason for an alternative design.

**Number of hidden neurons**

A small number of hidden neurons simplifies the learning. If nothing else is mentioned all ANN have 10 hidden neurons.

**Input representation**

This is often one of the most difficult decisions, which usually requires experimenting. In this work three different input representations were tried.

**Training method**

In all attempts the backpropagation training algorithm was used when using supervised training, and TD(0) for unsupervised training. One problem with these algorithms is to decide the learning parameters: learning rate and momentum. There is no simple method to decide which parameters yield the best result, so for each experiment below I made several attempts with different parameters. However, I only present the best result for each experiment.

**Over learning**

Since the training is going to be done on exact values from the entire domain it’s tempting to believe that over training can not happen. Unfortunately this is not entirely true, over training can always happen if the performance of the network is measured by something else than how well it performs on the training set.

In our case the network will be trained to approximate the equality (for random positions), but the measure of performance we really are interested in is how well it plays. In order to play well you don’t have to be able to give a good approximation of equality (or any other measure of goodness), as long as you from a set of positions can pick the most beneficial. An example is shown in Table 1.

For example, suppose you can choose between two possible moves, M_A and M_B, resulting in the positions P_A and P_B.

<table>
<thead>
<tr>
<th>True equality</th>
<th>Network 1</th>
<th>Network 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_A</td>
<td>0.20</td>
<td>-0.43</td>
</tr>
<tr>
<td>P_B</td>
<td>0.22</td>
<td>-0.31</td>
</tr>
</tbody>
</table>

*Table 1. An example of over learning.*

As table 1 shows, network 1 is really bad at estimating the equality (MSE ≈ 0.7) while network 2 does it almost perfectly (MSE ≈ 1/1000). Still network 1 manages to choose the
best move, while network 2 does not. MSE (Mean Square Error) is in other words not a very reliable measure of performance, and we have to look for other alternatives.

**Measures of performance in simple soccer**

One of the alternatives to MSE is MMSE (Mean move selection error). Given a position P, the chosen move m, and the optimal move $m_{opt}$, the move selection error is defined as the difference in equality between the positions resulting from move m and from move $m_{opt}$. In the example above Network 1 would have a move selection error of 0 (since the optimal move was selected) and Network 2 would have an error of 0.02 (since the move selected was 0.02 worse than the optimal move).

MMSE has a strong relation to playing strength, but note that the MMSE is measured for random positions, and only truly random positions in a match is the first one. Hence a network with a low MMSE is not necessarily stronger than a network with high MMSE.

As another measure of playing strength I use 3 reference players:

- **Untrained**, a randomly initiated network without any training.
- **Rules**, a simple non-ANN player, based on a few simple rules. See appendix 2 for a closer description.
- **Perfect**, this player is using the exact solution and hence plays perfect.

Playing test matches between the network and the reference players can give a good measure of strength, but since a large number of matches are required for the result to be reliable I do not measure performance with help of the reference players during the training.

**Short presentation of the experiments**

On simple soccer four different ANN experiments were performed:

**Experiment 1**: Only the position of the ball was used as input.

**Experiment 2**: The entire board was used as input.

**Experiment 3**: The entire board was used as input, but it was represented in a different way.

**Experiment 4**: Same input representation as in experiment 3, but the exact solution was not used for the training.

On less simple soccer the following two ANN experiments were performed:

**Experiment 5**: Same setup as in experiment 4.

**Experiment 6**: Same setup as in experiment 4, except 25 hidden neurons were used instead of 10.
Results

In this section six experiments are presented, four was made on simple soccer and two was made on less simple soccer. Each experiment resulted in an ANN based player.

Simple Soccer

Experiment 1

Since the final result of each game is decided by the position of the ball it is reasonable to assume that this is an important fact for judging the chance of winning. As a first experiment I therefore used this fact as the only input, ignoring the players. The ball can be in 18 different positions, and hence the ANN has 18 Boolean inputs. The hidden layer consists of 3 neurons, and the output layer of 1 neuron.

The network was trained on random positions, with the equality of the positions as desired values. Since it is computationally expensive to verify the training network against all 3.4 million positions I used 100,000 random positions as validation set, both for MSE and MMSE.

With only 18 unique inputs it is, of course, no need to use an ANN. The main purpose with this experiment was rather to test the the ANN implementation.

Diagram. 1. Training progress with a simple ANN that only uses the position of the ball as input
Table 2. This table shows the relative playing strength between all players. Each square contains the winning statistics and average number of moves. For example: in the top right square you see that Untrained wins 9.3% against Perfect, and the matches last for an average of 6.8 moves. Each entry is based on 100 000 matches, and the standard deviation is in all cases less than 1.6 x 10^{-3}.

Table 2 shows that after the training the ANN wins 79.2% of the matches against the untrained network, indicating that even a simple ANN is substantially better than a random player.

Table 3. Comparison between MSE and MMSE for all players. Based on 100 000 random positions.

As can be seen from table 3 the MSE for Rules is rather high. This is due to the fact that Rules was designed to choose good moves, not to be good at estimating equality.

**Experiment 2**

Can the result be improved by taking the position of the pieces into consideration? In experiment 2 I dedicated 12 inputs for the ball and for each piece, making a total of 12 x 5 = 60 inputs.

For every piece, and ball, each of the 12 dedicated inputs were assigned to a unique column or row. The inputs assigned to the row and column where the piece is located are set to true, and all other inputs are set to false.
As expected the ANN from this experiment is substantially better than the previous ANN. This can been seen in diagram 2, table 4 and table 5.

In diagram 2, note that the MMSE after 250 000 training positions is approximately 0.11, and does never go much lower. But in a test match between the network after 250 000 training positions and 5 000 000 training positions the later won 53.1 % of the matches. This indicates that MMSE is far from a perfect measure of playing strength.
Table 4. The relative playing strength between the players. The information not earlier presented is shadowed.

<table>
<thead>
<tr>
<th></th>
<th>Untrained</th>
<th>Player 1</th>
<th>Player 2</th>
<th>Rules</th>
<th>Perfect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Untrained</td>
<td>20.8%</td>
<td>11.5%</td>
<td>10.1%</td>
<td>9.3%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>63.9</td>
<td>47.8</td>
<td>12.1</td>
<td>7.6</td>
<td>6.8</td>
</tr>
<tr>
<td>Player 1</td>
<td>79.2%</td>
<td>26.6%</td>
<td>22.4%</td>
<td>16.2%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>47.8</td>
<td>44.4</td>
<td>15.6</td>
<td>9.3</td>
<td>8.0</td>
</tr>
<tr>
<td>Player 2</td>
<td>88.5%</td>
<td>73.4%</td>
<td>45.6%</td>
<td>26.1%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12.1</td>
<td>15.6</td>
<td>14.6</td>
<td>10.5</td>
<td>11.9</td>
</tr>
<tr>
<td>Rules</td>
<td>89.9%</td>
<td>77.6%</td>
<td>54.4%</td>
<td>26.2%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7.6</td>
<td>9.3</td>
<td>10.5</td>
<td>10.1</td>
<td>9.8</td>
</tr>
<tr>
<td>Perfect</td>
<td>90.7%</td>
<td>83.8%</td>
<td>73.9%</td>
<td>73.8%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6.8</td>
<td>8.0</td>
<td>11.9</td>
<td>9.8</td>
<td>14.7</td>
</tr>
</tbody>
</table>

Table 5. Comparison between MSE and MMSE. The information not earlier presented is shadowed.

<table>
<thead>
<tr>
<th></th>
<th>MSE</th>
<th>MMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Untrained</td>
<td>0.404</td>
<td>0.387</td>
</tr>
<tr>
<td>Player 1</td>
<td>0.186</td>
<td>0.230</td>
</tr>
<tr>
<td>Player 2</td>
<td>0.023</td>
<td>0.106</td>
</tr>
<tr>
<td>Rules</td>
<td>0.174</td>
<td>0.074</td>
</tr>
<tr>
<td>Perfect</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Experiment 3**

In experiment 3, three Boolean inputs were dedicated for each square on the left half of the field. First input is true if the square contains a yellow piece, second if it contains a red piece, and third if it contains the ball. The squares on the right side of the field only contain inputs for red and yellow pieces. If the ball is on the right side of the field the “reflection along the centre vertical line”-symmetry is used before applying the ANN. Total number of input neurons is 90. The true value was set to 1 and the false value was set to a small negative value, chosen so that the average input for each neuron is 0 (over all possible inputs).
Diagram 3. Training progress for an ANN that takes the entire board into consideration.

Table 6. The relative playing strength between the players. The information not earlier presented is shadowed.
Table 7. Comparison between MSE and MMSE.

<table>
<thead>
<tr>
<th></th>
<th>MSE</th>
<th>MMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Untrained</td>
<td>0.404</td>
<td>0.387</td>
</tr>
<tr>
<td>Player 1</td>
<td>0.186</td>
<td>0.230</td>
</tr>
<tr>
<td>Player 2</td>
<td>0.023</td>
<td>0.106</td>
</tr>
<tr>
<td>Player 3</td>
<td>0.015</td>
<td>0.031</td>
</tr>
<tr>
<td>Rules</td>
<td>0.174</td>
<td>0.074</td>
</tr>
<tr>
<td>Perfect</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Experiment 4

In all previous experiments I used the known solution to train the network. When attempting less simple soccer such solution is not available. Instead the network was trained according to the TD(0) method. To get an estimate how this method performs I will in experiment 4 use TD(0) to train the same design as in experiment 3.

Diagram 4. Training by TD(0).
Table 8. A complete table of the relative playing strength between the players of simple soccer. The information not earlier presented is shadowed.

<table>
<thead>
<tr>
<th>Untrained</th>
<th>Player 1</th>
<th>Player 2</th>
<th>Player 3</th>
<th>Player 4</th>
<th>Rules</th>
<th>Perfect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Untrained</td>
<td>20.8%</td>
<td>11.5%</td>
<td>10.3%</td>
<td>10.1%</td>
<td>10.1%</td>
<td>9.3%</td>
</tr>
<tr>
<td></td>
<td>63.9</td>
<td>47.8</td>
<td>12.1</td>
<td>7.3</td>
<td>7.9</td>
<td>7.6</td>
</tr>
<tr>
<td>Player 1</td>
<td>79.2%</td>
<td>26.6%</td>
<td>19.1%</td>
<td>18.7%</td>
<td>22.4%</td>
<td>16.2%</td>
</tr>
<tr>
<td></td>
<td>47.8</td>
<td>44.4</td>
<td>15.6</td>
<td>8.8</td>
<td>8.7</td>
<td>9.3</td>
</tr>
<tr>
<td>Player 2</td>
<td>88.5%</td>
<td>73.4%</td>
<td>38.1%</td>
<td>33.2%</td>
<td>45.6%</td>
<td>26.1%</td>
</tr>
<tr>
<td></td>
<td>12.1</td>
<td>15.6</td>
<td>14.6</td>
<td>15.5</td>
<td>13.2</td>
<td>10.5</td>
</tr>
<tr>
<td>Player 3</td>
<td>89.7%</td>
<td>80.9%</td>
<td>61.9%</td>
<td>43.7%</td>
<td>60.6%</td>
<td>34.1%</td>
</tr>
<tr>
<td></td>
<td>7.3</td>
<td>8.8</td>
<td>15.5</td>
<td>16.4</td>
<td>15.0</td>
<td>10.9</td>
</tr>
<tr>
<td>Player 4</td>
<td>89.9%</td>
<td>81.3%</td>
<td>66.8%</td>
<td>56.3%</td>
<td>65.7%</td>
<td>37.4%</td>
</tr>
<tr>
<td></td>
<td>7.9</td>
<td>8.7</td>
<td>13.2</td>
<td>15.0</td>
<td>14.0</td>
<td>11.2</td>
</tr>
<tr>
<td>Rules</td>
<td>89.9%</td>
<td>77.6%</td>
<td>54.4%</td>
<td>39.4%</td>
<td>34.3%</td>
<td>26.2%</td>
</tr>
<tr>
<td></td>
<td>7.6</td>
<td>9.3</td>
<td>10.5</td>
<td>10.9</td>
<td>11.2</td>
<td>10.1</td>
</tr>
<tr>
<td>Perfect</td>
<td>90.7%</td>
<td>83.8%</td>
<td>73.9%</td>
<td>73.8%</td>
<td>62.6%</td>
<td>73.8%</td>
</tr>
<tr>
<td></td>
<td>6.8</td>
<td>8.0</td>
<td>11.9</td>
<td>9.8</td>
<td>12.9</td>
<td>9.8</td>
</tr>
</tbody>
</table>

Table 9. Comparison between MSE and MMSE.

<table>
<thead>
<tr>
<th></th>
<th>MSE</th>
<th>MMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Untrained</td>
<td>0.404</td>
<td>0.387</td>
</tr>
<tr>
<td>Player 1</td>
<td>0.186</td>
<td>0.230</td>
</tr>
<tr>
<td>Player 2</td>
<td>0.023</td>
<td>0.106</td>
</tr>
<tr>
<td>Player 3</td>
<td>0.015</td>
<td>0.031</td>
</tr>
<tr>
<td>Player 4</td>
<td>0.028</td>
<td>0.036</td>
</tr>
<tr>
<td>Rules</td>
<td>0.174</td>
<td>0.074</td>
</tr>
<tr>
<td>Perfect</td>
<td>0.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>

Note that player 4 plays stronger than player 3, even though the training in experiment 4 was done without using the known solution. This may sound odd, but has a simple explanation:

While in the previous experiment all ANN were trained from random positions, the ANN in experiment 4 trained from self play, and during self play you learn from the positions you encounter. The more likely a position is to occur the more likely the network is to learn from it. This explanation is supported by the fact that player 4 is worse than player 3 both in
evaluating random positions and picking good moves in a random position, as can be seen in table 9.

It is possible that a training method combining self play and the exact values would produce still a stronger player, but falls outside the aim of this work.

**Less simple soccer**

**Experiment 5**

In experiment 5 the same ANN design as in experiments 3 and 4 will be used. The increased size of the board and the extra pieces lead to an increase of the number of input neurons from 90 to 150.

Diagram 5 contains two sets of data, MSE and the chance of winning against Rules. Since the exact result is not available the MSE is based on the difference between the training value (the desired value) and the networks estimation of this value before the positions is trained.

![Diagram 5. TD(0) training in less simple soccer. 10 hidden neurons.](image)

<table>
<thead>
<tr>
<th>Rules</th>
<th>Player 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rules</td>
<td>44.0%</td>
</tr>
<tr>
<td>Player 5</td>
<td>56.0%</td>
</tr>
</tbody>
</table>
Table 10. Comparison of the playing strength between Rules and Player 5. Based on 250 000 matches.

The uneven curves in diagram 5 can be explained by imprecise measurements and high learning rate. The values for the chance of winning against Rules is based on only 10 000 games (the 95% confidence interval is approximately 1.0 %), and the MSE is only based on one epoch of 50 000 matches.

The learning rate is 30 times higher than the learning rate usually used in the previous experiments. With a lower learning rate the network tends to stabilize on about 52 % wins against Rules (an obvious interpretation is that it hits a local minimum).

I also tried to gradually decrease the learning rate, but failed to reach any better results than with a constant high learning rate. Instead I had to resort to making more precise test of the network at some points in time when it seemed to have a high percentage of wins against Rules. The ANN proved to be as best after being trained for 3.6 million matches. It then won 56.3 % against Rules (based on 100 000 matches).

**Experiment 6**

Next, experiment 5 was repeated with 25 hidden neurons (instead of 10). A higher number of neurons should make it possible for the network to learn more features, but will also make the training slower and increase the risk of over learning.

![Diagram 6. TD(0) training in less simple soccer. 25 hidden neurons.](image)

The best performance for player 6 I got after 6.4 million training matches, after which it wins 57.6 % against Rules (based on 250 000 matches).
<p>|</p>
<table>
<thead>
<tr>
<th>Rules</th>
<th>Player 5</th>
<th>Player 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rules</td>
<td>44.0%</td>
<td>42.4%</td>
</tr>
<tr>
<td>Player 5</td>
<td>56.0%</td>
<td>48.6%</td>
</tr>
<tr>
<td>Player 6</td>
<td>57.6%</td>
<td>51.4%</td>
</tr>
</tbody>
</table>

*Table 11. Comparison of the playing strength. Based on 250 000 matches.*

Heads up player 6 wins against player 5 in 51.4 % of the matches.
Discussion

The experiments show that an ANN based player can outperform a simple rule based player, even with unsupervised training. This is a very promising result.

However, in the last experiment it was shown that more hidden neurons only give a slightly stronger player. This was a bit of a disappointment, since an ANN with more hidden neurons should be able to learn more advanced game concepts. Using more hidden neurons does however also make the training more difficult, and since the training was done with a comparably high learning rate it is likely that the network did not reach its full potential. More experiments on this, preferably with alternative learning methods such as conjugate gradient descent, would be very interesting. Unfortunately time does not allow this as a part of this project.
Real life results

The ultimate measurement of strength for a game playing program is how well it performs against opponents of known strength. For street soccer there are at least two communities available on Internet, the most popular is “Little Golem” (www.littlegolem.net).

Little Golem has about 300 active street soccer players, and offers several tournaments with ranked matches. Each match is played with 36 hours thinking time per move.

Construction of NanoBrain

To be able to evaluate the network at Little Golem I created a network based player called “NanoBrain”. In order to do so the following problems had to be considered:

Problem 1: The ANN is trained for random setup

Street soccer rules: The yellow player places his pieces first, and then the red player does the same. After the placement the yellow player does the kick off.

Since the ANN is only trained for complete positions (positions with one ball and 10 players on the field), it cannot do any evaluation before the kick off, and due to the large amount of possibilities it is not possible to let the network evaluate all possible combination of placements and kick offs within a reasonable amount of time. It may be possible to create an opening library within a few weeks of computation time, but I decided to just let NanoBrain place its pieces randomly.

Problem 2: The ANN is trained for a first goal win

Street soccer rules: A match is always played for at least 52 moves. After a goal the goalkeeper kicks the ball back into play.

Since the network is only trained for one goal matches it lacks understanding of two important game concepts:

Sacrificing short term advantages for long term advantages. In some cases it is more important to keep your players in good positions than hunting the ball, even if it reduces your chance to make the next goal. This is especially true early in the game.

Play on the result. If you are one goal down with just a few moves left you should not concern yourself very much about your defence, and vice versa.

Especially the ability to play on the result is very important, and it is reasonable to assume that the lack of understanding of this concept results in a significant weaker player. In the chapter “Possible improvements” I suggest some possible ways to handle this problem, but NanoBrain ignores it.

Problem 3: The ANN can’t tell the goalkeeper apart from the other players.

The goalkeeper is the only piece allowed inside the goal area, and it is generally a good idea to keep him close to the goal. Since the ANN can’t tell the goalkeeper apart from the other
pieces it sometimes moves him into the opponents side of the field. This happens rather rarely, and I don’t judge it to have a big impact on the playing strength.

**Problem 4: The network can’t handle positions where any piece is outside the board.**

In some, rather rare, situations it can be a good idea to move a piece out of the field. Since the squares outside the field are not part of the networks input such positions can’t be evaluated, and are hence never generated by the move generator. The fact that the network never makes such moves, even when they are sound, is a rather small problem. A more serious problem is when such moves are made by the opponent. The best solution I could come up with was to let Rules, handle all such positions. But I would like to point out that of all positions NanoBrain encountered, approximately 17 000, this only happened twice.

**Search depth**

On a modern computer it only takes a small fraction of a second for the network to select how to move in a given position. Humans on the other hand use anything from a few seconds to over an hour for the same task.

In my opinion a fair comparison of playing strength should be done with equal amount of thinking time, and I therefore allow NanoBrain to look ahead 3 plies. This means it considers all moves it can make, all moves the opponent can make in return, and all moves it can make in return to these moves.

Roughly NanoBrain evaluates 1 500 000 positions for each move, which takes about 30 seconds. At little golem each player has 36 hours per move, but since it is my belief that few players spend more than 10 minutes active time it would not be fair with a 4 plies search (takes approximately 1.5 hours).

This game tree search results in a considerably stronger playing strength. When playing with 1 ply against 2 plies over 80% of the matches are won by the 2 plies version.
Results

During an extended period of time I let NanoBrain 1.0 play rated tournaments against human opponents at little golem (www.littlegolem.net). At little golem each player start with the rating 1500, and each time a player gains rating points the opponent loses same amount of points. Hence, the average rating always remains at 1500. The very best players have a rating around 1900.

![Diagram 7. The rating for NanoBrain 1.0 against human opponents at little golem.](image)

As can be seen in diagram 7, the rating of NanoBrain 1.0 fluctuates rather much and ends up just above 1700. This means that NanoBrain is better than about 80% of the players at Little Golem. Unfortunately I didn’t have the opportunity to test NanoBrain against Loco, the only other computer player at Little Golem, before it retired 2005 with a final rating of 1602.

Note that NanoBrain 1.0 is based on an older ANN, and is inferior to NanoBrain 2.0 which is based on the ANN from experiment 6. Heads up NanoBrain 2.0 wins 65% of the matches against NanoBrain 1.0, but since they share the same weaknesses it’s difficult to translate this into a rating improvement against human opponents. Unfortunately time did not allow testing of NanoBrain 2.0 at Little Golem within this project.
Possible improvements

The main weakness of NanoBrain is, as mentioned, the inability of playing on the result. Unfortunately this is also one of the most difficult problems to solve. The obvious solution, to train the net on the entire game with additional inputs for current score and time would most likely not be very successful, since the task would be too complex.

A less ambitious approach would be to train an additional net to play defensive. If in lead, and the time is running out, base the moves on the defensive net. If the lead is smaller, or there is more time left, perhaps the evaluations from the normal net and the defensive net could be weighted together. A similar net could also be trained for attacking.

Another way to increase the strength is to improve the net, within it is limitations. This could possibly be done by adding extra inputs, adding more hidden neurons, or alternative learning algorithms. Especially adding extra inputs could make a considerable improvement, as it did for TD-gammon [5]. Unfortunately it is not obvious which inputs would be useful.

Other, non-ANN based, improvements would be an opening library and a better game tree search algorithm. NanoBrain makes a full width 3 plies search, maybe a more shallow search could pick candidate moves for a deeper search.
Conclusions

The final ANN based player did rather well against human opponents, and it is not unreasonable to assume that the strength can be further improved by implementing some of the suggestions in “Possible Improvements”. Maybe it then could rival even the very best human players.

The first and obvious conclusion from this is that street soccer is indeed well suited for ANN, which leads to the question of why. As earlier mentioned, backgammon works well while chess and go do not. Street soccer shares the non-deterministic feature with backgammon, and the wide range of possible final positions with chess and go.

Hence the randomness seems to be of big importance, while the large number of possible final positions is not.
References


Appendix

Appendix 1, Street soccer rules

Street soccer is a turn based two persons game. Each player (person) controls a team of five team pieces, of which one is a goalkeeper. The teams are traditionally called yellow and red, and the players who control the teams are called yellow player and red player.

Placing phase

After setting up the start positions, see position 3, and deciding who plays which team, the yellow player places his remaining pieces. He has to place at least one piece on the opponents half, and he’s not allowed to place any pieces in the middle circle or directly in front of any of the goals. Neither is he allowed to place more than one of his remaining pieces in his penalty area, or to place on an occupied square. When the yellow player is done the red player places his pieces according to the same rules.

Position 3. The starting position is Street Soccer, before the placing phase.

Position 4. A goal can be scored in case A, but not in case B.
Kick-off phase
The yellow player rolls one die, and remembers the result. He then rolls the same die until it shows a different result then the first time. The difference between the rolls is the initial movement of the ball. He is then required to move the ball until it runs out of movement, according to the rules below.

Ball movement
The ball is allowed to move horizontally, vertically or diagonally. When one player has started to move the ball in one direction the direction is not allowed to be changed more than once, and then only with 45 degrees. This corresponds to curving the ball. Each time the ball enters a square a movement point is lost, and when it runs out of movement it is next player’s turn to move.

If a player succeeds to move the ball to a square occupied by one of his own pieces he has completed a pass. When a pass is completed the ball gets one more movement, and the player is allowed to continue the movement in any direction and gets a new chance to curve the ball.

The ball is never allowed to go outside the lines (except when scoring a goal, see below) or move into a square occupied by one of the opponent’s pieces. Moves with such results are illegal. The ball is not allowed to come to rest until it is out of movement or a goal is scored.

Playing phase
After the kick off the red player makes the first normal move. During a normal move the player to move first rolls a die. The result is his movement points.

He first picks a piece to move. He can move it horizontally or vertically, but not diagonally. Each time the piece enters a new square one movement point is lost. The player keeps moving the same piece until he either runs out of movement points, or the piece reaches the ball. If the piece reaches the ball the remaining movement plus one is transferred to the ball.

You are never allowed to have more than one none goalkeeper piece in your own penalty area, and except the goal keepers no pieces are allowed directly in front of the goals. Note: The goalkeepers are allowed to stand directly in front of their own goal, but not directly in front of the opponent’s goal. However, there is one exception to the rule about no players directly in front of the goal: If the ball lays directly in front of a goal any piece is allowed to move to it. After the move is completed that piece is moved away from the goal to the first free square in the same column.

Scoring a goal
A goal is scored if the ball reaches inside the opponent’s goal. Moving the ball diagonally into the opponent’s goal is only allowed if the ball directly before passes a square directly in front of the goal. See position 4.

Goal kick
After scoring a goal the player who let the goal in moves his defending goalkeeper to one of the squares directly in front of his goal, according to his own choice. He then moves the ball to the same square and rolls a die. The result from the die is the movement of the ball.

Anti blocking rules
You are not allowed to block the ball so that the opponent can’t reach the ball without passing over the lines.

Match length and points
In tournaments you can in each game earn 0 to 5 points. If one player has scored more goals than the other after move 52 (including placing and kick off), the winner gets 5 points and looser 0 points.
If the score is equal after move 52 the game is continued until a goal is scored, but never for more than a total of 72 moves. If a goal is scored before move 72 is competed the winner gets 4 points and the loser 1 point. If the score is still equal after move 72 the player who scored the last move gets 3 points, and the other player 2 points. If the game ends without any goals both players get 2 points.
Appendix 2, The player “Rules”

The player Rules is based on an algorithm that only uses the row of the ball (counted from the bottom), and the distance between the ball and the closest player of each colour (bird way), and a small random value.

The evaluation score is: $(10 \times \text{“row of the ball”} + \text{“red distance to the ball”} - \text{“yellow distance to the ball”} + \text{“random value between -0.01 and 0.01”}) / 45.0$

In other words, the most important feature is the row of the ball, and the second most important feature is who is closest to the ball. The purpose of the devisor is to get evaluation score in the range [-1 1].