A Neural Network for Evaluating King Danger in Shogi

Reijer Grimbergen
Department of Information Science, Saga University
Honjo-machi 1, Saga-shi, 840-8502 Japan
E-mail: grimbergen@fu.is.saga-u.ac.jp

Abstract

Neural networks are a promising tool for automatically tuning parts of game programs. For neural networks to be useful, it must be clear which features need to be weighted. There also must be a way to decide automatically if the output of the neural network is correct or not. King danger in shogi meets both these requirements. In this paper a simple neural network is given that uses 161 features as input units. This network is trained with 500 positive and 500 negative examples of king danger. The learning curves for this neural network are not ideal, but for large training sets the learning behaviour is good, giving a prediction accuracy of over 90%. Furthermore, a self-play experiment shows that it is likely that the resulting network can be used in a normal evaluation function.

1 Introduction

Neural networks have been applied in different areas of Artificial Intelligence. The NETtalk program for pronouncing written text [9], networks for handwritten character recognition [6] and ALVINN, a neural network to drive a car [7] are examples of successful AI applications using neural networks.

In games, there also have been a number of applications using neural networks. Most famous examples are Tesauro's Neurogammon [11] and TD-gammon [10] systems. Neurogammon used only a neural network and expert feedback, while TD-gammon used a combination of temporal difference learning and a neural network. TD-gammon no longer required any intervention from outside and learned to play world class backgammon by self-play only.

Neural networks have been tried in other games as well. However, in chess [5], Go [8] and shogi [12] the results have been far less spectacular than in backgammon. The main problem of neural networks is that they are black boxes. A number of features is chosen and these features are weighted to give an output value. Updating weights is done automatically by training the network with a set of examples and their corresponding output values. If the updating does not produce satisfying results, it is hard to find the problem. It could be caused by a flawed feature set, problems in the updating or a bad network design.

Furthermore, automatic updating of weights requires that there is some way of laying the blame of a wrong output value on a subset of the features, i.e. there must be a training set with positive and negative examples (defined by feature sets) with their correct output value. For games this is a problem, as it requires continuous feedback of an expert (by pointing out mistakes, preferably explained by a set of features that are the culprit) or a very clear understanding of the relation between the output value and the features. It is hard to use the limited feedback at the end of the game (win, loss or draw) to determine where the program went wrong. Therefore, rather than neural networks, TD-learning methods have been used in games to use changes in evaluation function values during the game for learning [1, 2, 3].

Despite the shift to TD-learning, neural networks can still play a role in game research if two basic requirements are met. First, the feature set must be more or less agreed upon. Second, there must be well-defined training set of positive and negative examples. If these two requirements are met, it is possible that an evaluation function based on a neural network is more accurate than a hand-tuned evaluation function.

In this paper, a neural network will be presented for evaluating a major component of an evaluation function for shogi: king danger. It will be shown that king danger meets the two requirements for neural network use. It will also be shown that the neural network for this task is not trivial. The network needs to balance 161 features. The next section explains why king danger is a good candidate for using a neural network. Also, the design of the neural network for this task will be given.

2 Design of the Neural Network

From an AI point of view, the goal of neural network research is to design a neural network that can perform a human task, i.e. play a game like shogi. Game programs usually have a search component and an evaluation component. There has been research into automatically guiding the search [4], but this part of the problem is generally considered to be the more difficult of the two. Most machine learning research in games focuses on learning (parts of) the evaluation function.

A general evaluation function in shogi consists of two major components: material and king danger (of course there are many minor features that play a role in the overall evaluation as well). In this paper, the first step in the design of a neural network that learns how to evaluate king danger will be given. King danger has been chosen because it meets the two requirements for neural network design that have been given in the introduction. Firstly, most of the features that make up king danger are reasonably well understood. For example, attacked squares near the king, discovered attacks, open squares that can be used for drops, and pins are part of the evaluation function of every strong shogi program. Secondly, it is quite easy to make a well-defined set of positive and negative examples for learning king danger. The positive examples can be taken from tsume shogi problems (if there is mate, the king is in danger) and the negative examples can be taken from next move problems that use middle game positions. Therefore, king danger meets the two requirements outlined above and can be considered a potential candidate for using a neural network instead of a hand-tuned evaluation.

2.1 Input features of the network

As said, there is consensus on most of the features contributing to king danger. For the neural network, the following features were used for the input units:

- Features for each of the eight squares adjacent to the king (64 features)
  - The square is a board edge
  - There is a defending piece on the square
  - There is a attacking piece on the square
  - The square is an empty square
  - The defender controls the square
  - The attacker controls the square
  - Neither the defender or the attacker controls the square

    - The opponent covers the square

- The opponent has pieces in hand (45 features)
    - 0, 1 or 2 rooks
    - 0, 1 or 2 bishops
    - 0, ... , 4 golden generals
    - 0, ... , 4 silver generals
    - 0, ... , 4 knights
    - 0, ... , 4 lances
    - 0, ... , 18 pawns
- There are open lines to the king (32 features)
    - Open diagonals of length 0, 1, 2 or 3 in each of the four possible directions
    - Open rank or file of length 0, 1, 2 or 3 in each of the four possible directions
- Discovered attack (8 features)
    - Discovered attack on each of the four diagonals to the king
    - Discovered attack on each of the four ranks and files to the king
- Potential knight attack (4 features)
    - The knight square to the left of the king is free for a move or drop
    - The opponent can move a knight to the knight square on the left of the king
    - The knight square to the right of the king is free for a move or drop
    - The opponent can move a knight to the knight square on the right of the king
- Pinned pieces (8 features)
    - There is a pinned piece on one of the four diagonals to the king
    - There is a pinned piece on one of the four ranks and files to the king

As a result, there are 161 features in total. This is a relatively large number, making it unlikely that hand-tuning the weights between these features will lead to an optimal result. King danger is therefore not only a likely candidate for neural network learning, it is also a challenging part of the evaluation function and it can be expected that the use of a well-trained neural network will improve the performance.

2.2 The Perceptron

The implementation of a neural network for king danger that will be given in this paper uses the most simple network type: the *perceptron.* A perceptron consists of a number of input units, an output unit and weights between the input units and the output unit (see Figure 1). General neural networks can also have a number of hidden layers between input units and output units. In this paper, a perceptron is used to test the potential of the neural network approach. Another reason for using a perceptron is that the final goal of this research is to have a complete evaluation function for shogi implemented as a neural network. The topology of this network (the number of nodes and layers) is expected to be less complex if the components of the network

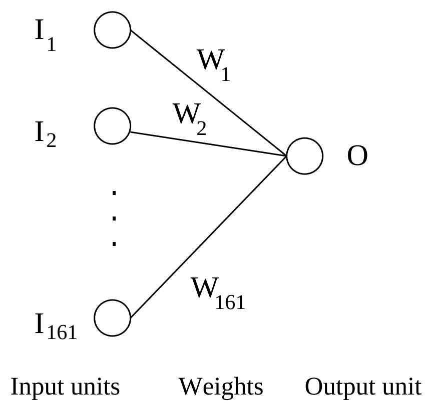

Figure 1: Perceptron with 161 input units (I_j), 161 weights (W_j) and 1 output unit (O).

(for example king danger) have less nodes and layers. Complex neural networks with many nodes and many different layers are known to be difficult to train. Having a complex neural network for king danger increases the possibility that the neural network for a full evaluation function will be impossible to train.

The perceptron of Figure 1 has 161 input units and one output unit. The value of the output unit is the sum of the weights of the input units that are active (the king danger features of the current position). If this output value exceeds a certain threshold, the king is considered to be in danger. If the output value is below the threshold, the king is safe. The output value given by the network can be wrong in two ways:

1. The example is positive (i.e. the king is in danger), but the output value is below the threshold
2. The example is negative (i.e. the king is safe), but the output value is above the threshold

In the first case, the weights of the active features must be increased to bring the output value above the threshold. In the second case the weights of the active features must be decreased to bring the output value below the threshold. The increase and decrease is determined by the *learning rate* (usually written as α). The general update rule is:

$$W_j \leftarrow W_j \pm \alpha$$

The network is trained by a number of positive examples and negative examples, called the *training set*. This training set is used to update the weights so that all the examples give the correct output value. In general, updating the weights so that all the examples give the correct output value can not be done in a single update cycle with the rule above. A number of update cycles (called *epochs*) is needed to get the correct output value for all the examples. Therefore, the number of epochs used to train the network is also important for the learning behaviour of the network. After training the network with the training set, a *test set* of positive and negative examples (of course different from the training set) is used to evaluate how many correct predictions the network can make. Learning behaviour can then be shown by an increased number of correct predictions when more examples are used for training.

2.3 Adding constraints to the neural network

In general, each of the weights of the perceptron network will be increased or decreased to fit the training examples. Therefore, it is possible that features that will never contribute positively to king danger get a positive weight or that features that will always increase king danger get a negative weight. To avoid these clear mistakes in the feature weights, the following constraints have been added to the updating of the perceptron weights:

- Always negative weight
  - The defender controls the square adjacent to the king
  - The opponent has no rook/bishop/gold/silver/knight/lance/pawn in hand
- Weight must always be greater or equal to zero
  - The opponent has at least one rook/bishop/gold/silver/knight/lance/pawn in hand
  - Discovered attack
  - Potential knight attack
  - Pinned pieces

3 Results

Two test were performed to evaluate the perceptron network. First, the learning behaviour was compared for different epoch numbers and different learning rates for a large set of positive and negative examples that were split into a training set and a test set. Second, the behaviour of the trained perceptron in actual game play was tested by plugging the perceptron into the evaluation function of the shogi playing program SPEAR, comparing the playing strength of the versions of the program with and without the perceptron.

3.1 Learning Curves

To train the perceptron, 500 positive examples and 500 negative examples of king danger have been used. The 500 positive examples are tsume shogi problems (shogi mating problems), while the 500 negative examples are middle game positions that have been taken from the next move problem section of the weekly shogi magazine Shukan Shogi. The positive examples and negative examples are used alternately. Initial weights between the input units and the output unit are assigned randomly. The king is considered to be in danger if the sum of the weights of the features of the example exceeds a certain threshold. In this experiment the threshold is set to 0.8. This value was chosen because it had a linear connection with the king danger threshold used in SPEAR (the value of this threshold is 800). Other values for the threshold were tested but did not lead to significant differences in learning behaviour.

Network training was done by dividing the examples into a training set and a test set. The training set was increased from 1 example to 999 examples and the test set was decreased correspondingly from 999 examples to 1 example. The network was trained using 5000, 10000, 15000, 20000, 25000 and 50000 epochs. For the learning rate α the values 0.01 and 0.001 were tried. The complete results of these experiments are shown in Figure 2.

To interpret the learning curves of Figure 2, one must realise that in a perfect learning experiment the network quickly finds the correct feature weights. Starting with a correct prediction rate of the test examples that is about 50%, the learning curve will quickly rise to a value near 100% and stay there until the end of the experiment. Looking at Figure 2, it is clear that the learning curves do not follow this ideal pattern. Even though the learning curves quickly rise from a correct prediction rate of 50% to 70%, the curves stay around 70% for a long time. Furthermore, when the training set contains about 640 examples, the network no longer stabilizes within the number of epochs and the learning curve suddenly drops to a correct prediction rate

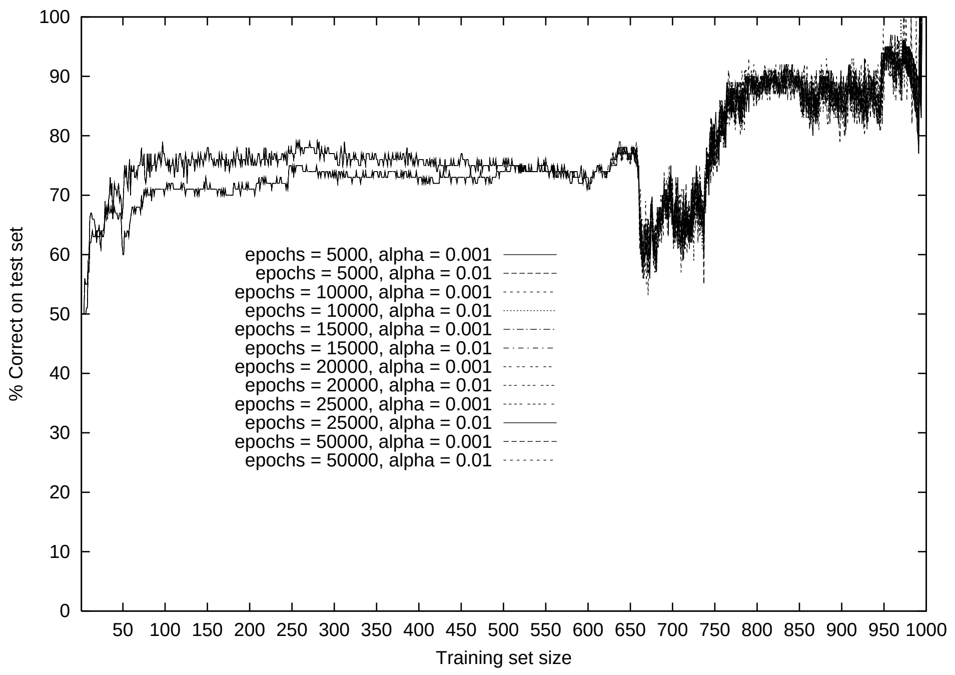

Figure 2: Learning curves for the perceptron. Number of epochs ranging from 5000 to 50000, learning rate α is 0.01 or 0.001.

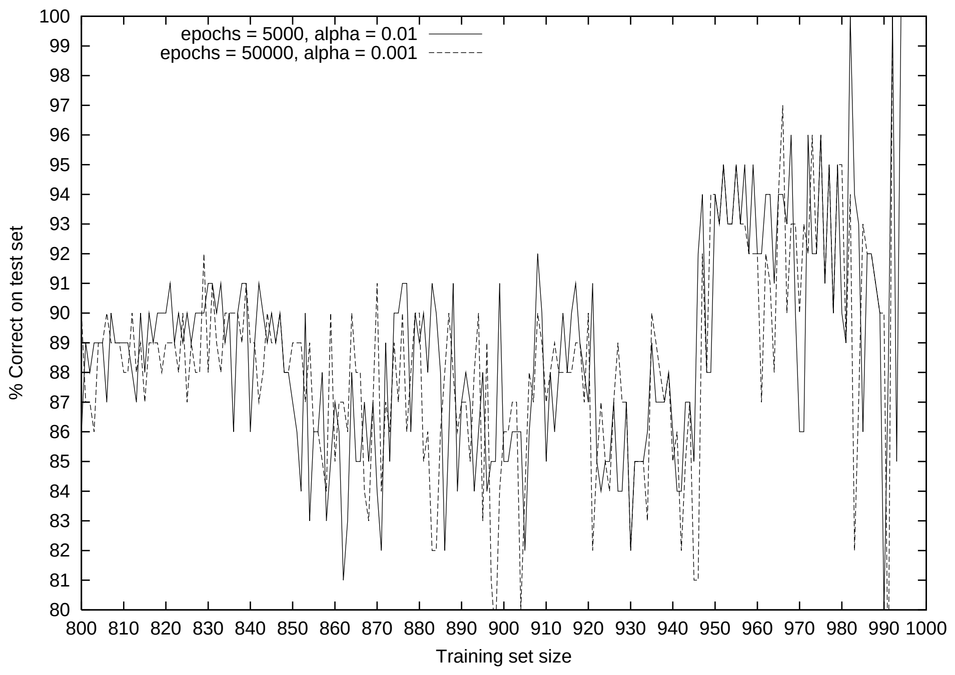

Figure 3: Detail of the perceptron learning curve. Compared are the learning curve using 5000 epochs and $\alpha = 0.01$ with the learning curve using 50000 epochs and $\alpha = 0.001$.

that is less than 60%. After this sharp drop, the learning behaviour improves and shows stable prediction rates of over 80% correct after training sets of about 780 examples or more.

It seems that the features that were defined in Section 2.1 are unable to define king danger. For example, the features do not include the concept of an escape route for the king. Focussing on the eight squares adjacent to the king has the risk of missing some vital information about ways the king can escape from an attack. Furthermore, it is also possible that there is a problem in the sets of positive and negative examples around example 640. The positive and negative examples were checked for consistency, but there may be other hidden problems in the set of examples.

Despite these problems, the perceptron shows clear learning behaviour and in the end is able to accurately predict more than 90% of the examples in the test set. This is a general pattern independent of epoch numbers and learning rates. In Figure 3, parts of the learning curves for 5000 epochs with $\alpha = 0.01$ and 50000 epochs with $\alpha = 0.001$ are given. These two learning curves are supposed to be the most different, but from the figure it is hard to tell them apart. However, it is clear that both achieve accuracy rates of well over 90% for a number of training set sizes.

3.2 Self-play experiments

The main issue is of course if the perceptron can be used in a shogi playing program. To test this, the perceptron that uses 50000 epochs for training and has a learning rate α of 0.001 has been plugged into the evaluation function of the shogi program SPEAR. Instead of using a hand-tuned evaluation for king danger, the activation of the output unit of the perceptron network is used. A number of matches of 20 games were played between the normal version of SPEAR and different versions of P-SPEAR, which uses the perceptron for king danger.

Using the perceptron in the evaluation function is not a trivial plug-in. King danger is only one of the components of the evaluation function and this component must be tuned with the other components. For example, the value of a pawn in SPEAR is 100 points, while the perceptron network will evaluate king danger with an activation of the output unit in a range between approximately -1.0 and 1.0. Unfortunately, tests showed that simply multiplying the output value by 100 did not produce a reasonable evaluation function (P-SPEAR lost this match 18-2). Because of the tuning problems, only a limited number of conversions of the output unit activation to a general evaluation value have been tried. However, the best version of P-SPEAR was able to give SPEAR good opposition, losing the 20 game match by 12 games to 8.

4 Conclusions and Future Work

In this paper a simple type of neural network called perceptron has been described that can be used to evaluate king danger in shogi. It was shown that king danger has a challenging number of features and that a perceptron can produce a high percentage of correct predictions of king danger. Using the perceptron in the evaluation function of a shogi playing program also indicated that the performance of the perceptron can be made comparable to a hand-tuned evaluation of king danger.

To make a perceptron a viable alternative for a hand-tuned evaluation of king danger, a number of improvements are necessary. Further analysis of the features contributing to king danger, adding more constraints to the feature weights, analysis of the set of examples and improving the balance between the output value of the perceptron network and the other components of the evaluation function are areas for future research.

There is also the possibility that a perceptron is unable to learn king danger and that neural networks with hidden layers are necessary for good results. In the feature set given in Section 2.1, there are features that have some clear relation. For example, opponent pieces in hand are related to empty squares near the king where they can be dropped. Making these relations explicit by introducing hidden layers that connect these features might improve the performance. In the near future, network topologies different from the perceptron will be tested.

References

[1] J. Baxter, A. Tridgell, and L. Weaver. Experiments in Parameter Learning Using Temporal Differences. *ICCA Journal*, 21(2):84–99, June 1998.

[2] D. Beal and M. Smith. First Results from using Temporal Difference Learning in Shogi. In H.J.van den Herik and H.Iida, editors, *Computers and Games: Proceedings CG'98. LNCS 1558*, pages 113–125. Springer-Verlag, Berlin, 1999.

[3] D.F. Beal and M.C. Smith. Learning Piece-square Values Using Temporal Differences. *ICCA Journal*, 22(4):223–235, December 1999.

[4] Y. Björnsson and T. Marsland. Learning Search Control in Adversary Games. In H.J.van den Herik and B.Monien, editors, *Advances in Computer Games 9*, pages 157–174. Van Spijk, Venlo, The Netherlands, 2001.

[5] G.M.C. Haworth and M. Velliste. Chess endgames and neural networks. *ICCA Journal*, 21(4):211–227, December 1998.

[6] Y. Le Cun, L. D. Jackel, B. Boser, and J. S. Denker. Handwritten digit recognition: Applications of neural network chips and automatic learning. *IEEE Communications Magazine*, 27(11):41–46, 1989.

[7] D. A. Pomerleau. *Neural Network Perception for Mobile Robot Guidance.* Kluwer, Dordrecht, The Netherlands, 1993.

[8] N. Sasaki, Y. Sawada, and J. Yoshimura. A neural network program of tsume-go. In H.J.van den Herik and H.Iida, editors, *Computers and Games: Proceedings CG'98. LNCS 1558*, pages 167–182. Springer-Verlag, Berlin, 1999.

[9] T. J. Sejnowski and C.R. Rosenburg. Parallel networks that learn to pronounce english text. *Complex Systems*, 1:145–168, 1987.

[10] G. Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8(3–4):257–277, 1992.

[11] G. Tesauro and T.J. Sejnowski. A parallel network that learns to play backgammon. *Artificial Intelligence*, 39(3):357–390, 1989.

[12] H. Tokuda and Y. Kotani. Experiment on generation of a evaluation function using neural network in shogi. In *Game Programming Workshop in Japan '95*, pages 47–56, Kanagawa, Japan, 1995. (In Japanese).