Investigations on Improvised Neural Network Chess Engine for Augmenting Topologies

M Suresh Babu[1,*], E. Keshava Reddy[2]
[1]*Research Scholar, JNTUA, Ananthapuramu, AP, India*
[2]*Professor, JNTUA, Anathapuramu, AP, India*

ABSTRACT

In this paper, reinforcement learning in a chess engine with the help of genetic algorithm was investigated. Thereinforcement learning methodology is used in the chess engine to get trained in to play games. The engine initially starts with null information about the game, however, every further moves in the game gets stored within the chess engine. Hencechess engine gains experience to discharge best performance with continuing games. The engine uses temporal difference leaf learning approach to improve the skill set in every level. The engine performs a brute force search of all possible positions that result from the given position on the board. The positions are evaluated using an evaluation function. The evaluation function incorporated with present engine is a neural network which returns a value for each position indicating its potential for either black or white. The neural network learns the function whose values are provided by the temporal difference leaf learning algorithm.

Key Words*: Reinforcement Learning, Gene Chess Engine, NEAT, Neural Network.*

1. INTRODUCTION

The number of all possible chess positions is $2.4\times e^{54}$or roughly10^{43}. This makes it difficult to evaluate a given chess position all the way to the way to the end of the game. Humans generally use their experience to evaluate a given chess position and identify the various features of the position. A chess playing program uses hard coded knowledge preprogrammed by an expert in chess to evaluate a given chess position. The program does a brute force search, checking all possible chess positions that can result from the given position up to a certain depth in a game search tree and uses an evaluation function to evaluate the resulting nodes to find the best move. In this paper we have used temporal difference learning to train the evaluation function. The program plays a number of games and is only given the final result of the game, a win, a loss or a draw. Temporal difference learning allows the program to draw the necessary conclusions of the game. The conclusions drawn may be inaccurate at first but as the program plays more and more games it gets better and the evaluation function becomes more and more accurate at evaluation the chess position. We have used a neural network to represent the evaluation function and allowed the network to evolve over time to have a topology that has a better accuracy by using evolutionary algorithms.

2. METHODOLOGY

a) Reinforcement learning

Reinforcement learning is branch of machine learning where the system learns how to solve problems that require a sequence of actions to reach the goal. This involves giving the agent a pre-defined reward when it reaches goal states. When the agent reaches a desired goal state it is given a positive reward, if it reaches an undesirable goal state it is given a negative reward. The agent then uses reinforcement learning algorithm to learn a value function for the various non-goal states to help it decide the next action. The value function is generally stored in look up table but in cases of games with an exponential number of states a function approximator like a neural network is used. TD-learning is a reinforcement learning algorithm that assigns utility values to states alone instead of state-action pairs. The desired values of the states are updated by the following function[6]:

$$V(st) = V(st) + \alpha \bullet (rt + \gamma \bullet V(st+1) - V(st)) \quad (1)$$

where:

• α is the learning rate

• rt is the received scalar reward of state t

• γ is the discount factor

• V (st) is the value of state t

• V (st+1) is value of the next state

• V (st) is the desired value of state t

For TD (λ)- learning is similar to the TD learning algorithm which takes into account the results of astochastic process and the prediction of the result by the next state. The desired value of the terminal state stend is for a board game.

The desired values of the other states are given by the following function:

$$V(st) = \lambda \cdot V(st+1) + \alpha \cdot ((1 - \lambda) \cdot (rt + \gamma \cdot V(st+1) - V(st))) \quad (2)$$

• $0 \leq \lambda \leq 1$ controls the feedback of the desired value of future states

If λ is 1, the desired value for all states will be the same as the desired value for the terminal state. If λ is 0, the desired value of a state will receives no feedback of the desired value of the next state. With λ set to 0, formula 2 is the same as formula 1. Therefore normal TD-learning is also called TD(0)-learning[6].

b) Genetic algorithm on neural networks

Genetic Programming is an evolutionary based methodology inspired by biological evolution to optimize computer programs, in particular game playing programs. It is a machine learning technique used to optimize a population of programs, for instance to maximize the winning rate versus a set of opponents, after modifying evaluation weight parameter. We use NEAT's genetic encoding scheme is designed to allow corresponding genes to be easily lined up when two genomes cross over during mating. NEAT (short for Neuro Evolution of Augmenting Topologies) and has been developed by Kenneth Stanley Owen and Risto Miikkulainen at the University of Texas[12]. It uses node-based encoding to describe the network structure and connection weights, and has a nifty way of avoiding the competing convention problem by utilizing the historical data generated when new nodes and links are created. NEAT also attempts to keep the size of the networks it produces to a minimum by starting the evolution using a population of networks of minimal topology and adding neurons and connections throughout the run. The NEAT structure contains a list of neuron genes and a list of link genes. A link gene, contains information about the two neurons it is connected to, the weight attached to that connection, a flag to indicate whether the link is enabled, a flag to indicate if the link is recurrent, and an innovation number. A neuron gene describes that neuron's function within the network—whether it be an input neuron, an output neuron, a hidden neuron, or a bias neuron. Each neuron gene also possesses a unique identification number [11].

3. PROPOSED SOLUTION

a) Introduction

In our experiments we used the TD(λ)-learning algorithm to learn from the occurred board positions in a game. In order to learn an evaluation function for the game of chess we made use of a database which contains games played by human experts. The games are stored in the file format Portable Game Notation (PGN). We wrote a program which converts a game in PGN format to board positions. A board position is propagated forward through a neural network with the output being the value of the position. The error between the value and the desired value of a board position is called the TD-error [1] :

$$\text{TD-error} = V(st) - V(st) \quad (3)$$

This error is used to change the weights of the neural network during the backward pass. We will repeat this learning process on a huge amount of database games. It's also possible to learn by letting the program play against itself. Learning on database examples has two advantages over learning from self-play.

Firstly, self-play is a much more time consuming learning method than database training. With self-play a game first has to be played to have training examples. With database training the games are already played as seen in Fig 2.
Secondly, with self-play it is hard to detect which moves are bad. If a blunder move is made by a player in a database game the other player will mostly win the game. At the beginning of self-play a blunder will often not be punished. This is because the program starts with randomized weights and thus plays random moves. Lots of games therefore are full of awkward looking moves and it is not easy to learn something from those games.
However, self-play can be interesting to use after training the program on database games. Since then a bad move will be more likely to get punished, the program can learn from its own mistakes. Some bad moves will never be played in a database game. The program may prefer such a move above others which actually are better. With self-play, the program will be able to play its preferred move and learn from it. After training solely on database games, it could be possible that the program will favor a bad move just because it hasn't had the opportunity to find out why it is a bad move [3].

b) Brief Solution

For the purpose of investigating our hypothesis (producing a reasonably good Chess Engine without injecting any human expert knowledge),the following evolutionary algorithm is used:

1. For the present experiments, each chess board was represented by a vector of length 64, with each component corresponding to an available position on the board.
2. Components in the vector were elements from {-K,-1,0,+1,+K}, where 0 corresponded to an empty square,1 with count was the value of a regular pawn, and K was the number assigned for a king. Instead, the value of K was evolved by the algorithm. The sign of the value indicated whether or not the piece belonged to the player (positive) or the opponent (negative).
3. When a board was presented to a neural network for evaluation, the scalar output of the network was interpreted as the worth of the board from the perspective of the player whose pieces were denoted by positive values. The closer the output was to 1.0, the better the evaluation of the corresponding board. Similarly, that closer the output was to –1.0, the worse the board. All positions that were wins for the player (e.g., no remaining opposing pieces) were assigned a value of exactly 1.0, and likewise all losing positions were assigned a value of –1.0.
4. The evolutionary algorithm began with a randomly created population of 10 artificial neural networks (also described as strategies), P_i, i = 1, …, 10, defined by the weights and biases which are initialized randomly.
5. The piece values for the various pieces on the chessboard are assigned to their material values. But some amount of variability is assigned to the piece values as their values can change depending on the nature of the position, as the network evolves it is able to evolve the value of the pierces depending on the nature of the position.
6. In this we have given 3 data folds to evaluate per candidate. 40 number of epochs to make each fold. 70% of data is used for training and rest will be validation.50 Number of candidates in a batch, to evaluate in parallel.

The value of the kings are assigned to an arbitrarily high value which only changes a little during the evolution of the network. The initial population of the networks being considered consist of 64 input nodes followed by a layer consisting of the combination of the various sub squares in order to extract both local information involving the squares immediately surrounding the square in question and information involving more number of squares, this information is captured in the sub square layer. The initial population is than allowed to evolve by using the temporal difference algorithm.

$$V(st) = \lambda \bullet V(st+1) + \alpha \bullet ((1 - \lambda) \bullet (rt + \gamma \bullet V(st+1) - V(st))) \quad (2)$$

Each network plays 10 games with the user or another chess engine and its results are recorded. The fitness value for each network is assigned by the number of wins, draws and losses. The networks than crossover and evolve by the methods described above. The new networks are than evaluated base on their playing record [1].

4. RESULTS AND ANALYSIS

The Genechess architecture was tested with established chess engines to analyze its performance. After the networks were evolved for 200 generations the program does fairly well. Fig 3 and 4 are Evaluation on the Test Set and Exporting the Best Network. The program does a lot of mistakes on the openings and endgames because it lacks explicit knowledge of these phases of the game. The engine's knowledge of the openings come from the limited number of games it has played involving the vast number of openings and the temporal difference algorithm makes imprecise conclusions about the various moves both in the openings and the endgames. The algorithm does fairly well in the middle game but mistakes in the opening and the endgame make it difficult to hold onto any concrete advantage against engines who have opening book and endgame tables encoded into them. The engine was allowed to play and learn against GNU-chess with a limited opening and endgame database to evolve its skill. The engine also played many games with humans who were asked to limit their opening preparation. Fig 5 is the Gene Chess Engine, The engine thus won 39 games, drew 70 and lost 69 games out of a total 178 games against gnu-chess.

PIECES	WHITE VALUE	BLACK VALUE
ROOK	+5	-5
KNIGHT	+3	-3
BISHOP	+3	-3
QUEEN	+9	-9
KING	+1000	-1000
PAWN	+1	-1

Table 1 is the value of each pieces on the chess board.

5. CONCLUSION

In this paper, we have proposed[2] a method for selection of candidate evaluation functions and also a learning mechanism that utilizes the dynamics of a population in order to control mutation amounts. Mutating by a random proportion of the standard deviation of parameters is beneficial, as the population controls mutation not the user. An experiment was conducted and the resulting evaluation function was examined through competition with Chess master. After learning, the performance of the population was greatly improved over the initial population and it is interesting that the values developed, both with and without seeding, are of similar nature.

Finally making this Gene Chess online where everyone can play against the chess engine helps it evolve it's network and makes the chess engine strong and powerful. The ability for non-experts to produce competent programs that exceed the expertise of the creator, with the use of evolutionary learning methods, is a very powerful approach that we would encourage people to use when domain specific knowledge is lacking. The engine can be further improved by doing explicit training on openings and endgames or encoding such databases into the engine. The engine can be trained on only opening and endgame studies to get better results to improve the probability of winning.

6. WORK CITED

[1]. Kenneth O. Stanley,Risto, Miikkulainen, Evolving Neural Networks through Augmenting Topologies, 10(2):99-127, 2002
[2]. Sebastian Thrun , Learning To Play the Game of Chess. Advances in Neural Information Processing Systems 7 1995.
[3]. HenkMannen, October 2003, Learning To Play Chess Using Reinforcement Learning With Database Games.
[4]. Philipp Koehn, 1994, Combining Genetic Algorithms and Neural Networks: The Encoding Problem.
[5]. PetrAksenov, 2008, Genetic algorithms for optimizing chess position scoring.
[6]. Jonathan Baxter, Andrew Tridgell ,LexWeaver "KnightCap: A chess program that learns by combining TD(λ) with game-tree search " , CiteSeerXβ 5 Jan 1999.
[7]. David B. Fogel, Evolving an Expert Checkers Playing Program, Published in IEEE August 2001.
[8]. A. L. Samuel, "Some studies in machine learning using the game of checkers,"IBMRes. Dev., vol. 3, pp. 210-219, 1959.
[9] Evaluation of chess position by Modular Neural Network generate by genetic algorithm,7th European conference, EuroGP 2004,Coimbra,Portugal,April 2004
[10]. David b. Fogel, fellow, ieee, Timothy j. hays, Sarah l. hahn, and James quon, A Self-Learning Evolutionary Chess Program, IEEE-2004.
[11]. by Simon M. Lucas, Computational Intelligence and Games: Challenges and Opportunities. International Journal of Automation and Computing (2008), volume 5, pages: 45 – 57.
[12]. AI Techniques For Game Programming, 2002, Prima Tech.ISBN-13: 978-1931841085