Pattern Based Chess Engine

A Project Report Submitted in
Partial Fulfillment of the Requirements for the
8^{th} Semester B.Tech. Project

by

Anuvesh Kumar	15-1-5-004
Haripriya Gummadi	15-1-5-016
Boddinagula Sanjeeth	15-1-5-020
Tulrose Deori	15-1-5-056

Under the Supervision of
Dr. Samir Kumar Borgohain

Computer Science & Engineering Department
NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR

May, 2019

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR

Declaration

Thesis Title: **Pattern Based Chess Engine**
Degree for which the Thesis is submitted: **Bachelor of Technology**

We declare that the presented thesis represents largely our own ideas and work in our own words. Where others ideas or words have been included, We have adequately cited and listed in the reference materials. The thesis has been prepared without resorting to plagiarism. We have adhered to all principles of academic honesty and integrity. No falsified or fabricated data have been presented in the thesis. We understand that any violation of the above will cause for disciplinary action by the Institute, including revoking the conferred degree, if conferred, and can also evoke penal action from the sources which have not been properly cited or from whom proper permission has not been taken.

Anuvesh Kumar
(15-1-5-004)

Haripriya Gummadi
(15-1-5-016)

Sanjeeth Boddinagula
(15-1-5-020)

Tulrose Deori
(15-1-5-056)

Date:

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR

It is certified that the work contained in this thesis entitled **Pattern Based Chess Engine** submitted by **Anuvesh Kumar**, Registration no (15-1-5-004), **Haripriya Gummadi**, Registration no (15-1-5-016), **Boddinagula Sanjeeth**, Registration no (15-1-5-020), **Tulrose Deori**, Registration no (15-1-5-056) for the B.Tech. End Semester Project Examination May, 2019 is absolutely based on their own work carried out under my supervision.

The work presented here has not been submitted elsewhere for award of any degree.

Place:
Date:

(Dr. Samir Kumar Borgohain)
Computer Science & Engineering
National Institute of Technology Silchar

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR

We understand that plagiarism defined as any one or the combination of the following: Uncredited verbatim,copying of individual sentences,Paragraphs or illustrations(such as graphs,diagrams etc.) from any source published or unpublished including the internet. Uncredited improper paraphrasing of pages or paragraphs (changing a few words or phrases,or rearranging the original sentence order) credited verbatim, copying of a major portion of a paper(or thesis chapter) without clear delineation of who did or wrote what(Source:IEEE). We have to the best of my abilities ensured that all the concepts,ideas,text,expressions,graphs,diagrams etc,which are not the outcome of my research have been credited to those who own the same. Elaborate sentences used verbatim from published work have been clearly identified and coded. We also affirm that no part of this thesis can be considered as plagiarism to the best of my knowledge and understanding and take complete responsibility if any complaint arises. We are fully aware that my project supervisor is not in a position to check for any possible instances of plagiarism within this submitted work.

Anuvesh Kumar
(15-1-5-004)

Haripriya Gummadi
(15-1-5-016)

Sanjeeth Boddinagula
(15-1-5-020)

Tulrose Deori
(15-1-5-056)

Date:

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR

This is to certify that the report titled **Pattern Based Chess Engine** submitted by **Anuvesh Kumar**, Registration no (15-1-5-004), **Haripriya Gummadi**, Registration no (15-1-5-016), **Boddinagula Sanjeeth**, Registration no (15-1-5-020), **Tulrose Deori**, Registration no (15-1-5-056) under the supervision of **Dr. Samir Kumar Borgohain** for the partial fulfilment of the requirements for award of the degree of Bachelor of Technology in Computer Science and Engineering, embodies the bonafide work done by them during the session 2018-19.

Place: **(Head of Department)**
Date: **Department of Computer Science & Engineering**
National Institute of Technology Silchar

Abstract

The game of chess can be perceived as a set of sequence, using which a player wins or loose the game. The past moves provides the context for the current move. These sequences are encoded in PGN. We develop a method which learns to evaluate sequences by assigning it the probability of winning. The sequence evaluation function is then used to assign scores to possible sequences that originate during a game and choose the best move accordingly. Our test results shows that the model is able to learn and exploit sequences to either win or loose the game.

Acknowledgements

We take this opportunity to express our sincere gratitude and heartily thanks to our supervisor **Dr. Samir Kumar Borgohain**, Department of Computer Science and Engineering, National Institute of Technology Silchar for his continuous inspiration and valuable guidance at every stage of my project work.

We are also grateful to **Prof. Sivaji Bandyopadhyay**, Director, NIT Silchar, **Dr. Arup Bhattacharjee**, HOD of Department of Computer Science and Engineering and **Dr. Dalton Meitei Thounaojam**, BTech Project Cordinator of Department of Computer Science and Engineering for their guidance and encouragement throughout our project work. Their concern and support have been invaluable to us in the completion of our project work.

We would also like to thank all the faculty members of the Computer Science and Engineering of National Institute of Technology Silchar, for their administrative support during various phases of this work. We would also like to thank our parents whose continuous support helped us to devote more time towards our project.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Over the years, the problem of automating a chess agent have been studied very deeply. As a result, the scientific literature is powered with many different approaches to solve the problem each having its own caveats. Over more than 70 years since the inception of the idea, the field of chess AI has been influenced by the technology of the time.

This range from standard rule based systems which works on countless conditions to self learning agents with little to no prior knowledge about chess.

The traditional techniques used to in chess AI typically included heuristics manually designed by Humans. These heuristics helped to achieve a good amount of success. But its success is limited to human understanding of the game and includes the biases alongside.

We give away the with lot of standards of developing chess AI and instead treat chess as a language. This converts the problem of chess as learning chess such that a valid sentence is a sequence of moves that helps to win the game. We tested both discriminative and generative models and choose to go with the discriminative model. Our Chess AI is able to make good moves taking previous moves as a context. Extracting knowledge from the raw sequence of moves cannot be done by classical machine learning algorithms. Therefore model these sequences using Neural Networks.

1.1 Motivation

We want to create a chess engine which can exploit the information encoded in chess sequences. As such instead of understanding the board position, we want our AI to understand sequences and use those which could lead to it winning the game.

1.2 Problem Statement

If we consider Chess as a sequence of moves, the problem is to build an engine which can understand and differentiate sequences on the basis of the probability of the sequence that could lead to a win.

1.3 Objectives

The objectives of the project are as follows:

1. To make chess engine which can generalize what sequence of moves lead to winning.
2. Create a sequence evaluation function which takes as input a sequences and give a probability of winning the game using the sequence.
3. Use the evaluation function to decide best move given the past N moves taken by either player.

1.4 Organisation of the report

- In Chapter 2, We discuss the intuition made by referring various research papers in which the same subject is discussed.

- In Chapter 3, We discuss how we can model chess using language modelling and the appropriate approaches and discuss in brief about Word Embeddings.
- In Chapter 4, We discuss the design and codes of implementation of the engine.
- In Chapter 5, We discuss the tests conducted, their respective results, observations, benefits and drawbacks.
- In Chapter 6, We discuss the conclusions drawn and the future work ought to be done.

CHAPTER 2

Literature Survey

The problem of chess AI: Computer playing chess, has been in the minds for more than 70 years. The first major revolution came when Nobert Wiener in his book 'Cybernetics' proposed the idea of depth limited min max search using an evaluation function. Since then, chess was broken down in two ideas, the search algorithm and board evaluation function. John McCarthy invents the alpha-beta search algorithm. The evolution continued. Stephen J. Edwards issued the first Portable Game Notation (PGN) allowing people and programs to share games. We use this extensively in our chess Engine.

The turning point was when DeepBlue defeated Garry Kasparov.

Since then Machine learning has allowed us to develop chess engines which 'learns' the game and it's evaluation functions without any explicit programming. DeepChess is an end to end neural network solution that learns the game of chess exploiting the knowledge base. It creates a board evaluation function by encoding the bit board representation of a game state using autoencoders and makes a board evaluation function which instead of assigning a value to a board, it instead compares two board positions. Latest versions of AlphaZero requires are deep reinforcement learning based methods which learns to play chess with no knowledge base using self play.

However all these approaches have their base in common, a search technique and a board evaluation function.

There are different approaches for language modelling and their drawbacks leading to the reason for opting LSTM. Understanding the structure of a language and to be able to model it becomes a major concern in Language processing tasks as a natural language evolves rather than having a fixed structure or semantics(Some out of ordered sentences can still be understood by Humans).

The traditional static probabilistic model (LM) gives the probability of a word or sequence of words occurring next in the sentence based on a given vocabulary. It's drawbacks include the lack of generalisation and the limited source of vocabulary , which leads to Neural networks being the most popular approach which provides generalisation by finding similarities and dissimilarities between words by representing them as vectors(word embeddings).

Traditional feed forward networks work on fixed length of inputs treating them as independent entities which opposes the characteristics of a natural language with words or sentences of large and variable lengths that must be arranged in a specific order to make sense(All inputs are dependent on each other). This drawback leads us to the use of recurrent neural networks that are widely popular and well known for sequence modelling leading them to take context of much longer lengths of input. But the well known problem of the loss of information while propagating through layers and also the difficulty in training leads us to search for a better optimizing algorithm. This is when LSTM comes into the picture as it preserves the required information using a longterm memory. LSTM unit consists of gates which help it to retain long term memory. These gates will tell how much of past information should be forgotten and how much should be kept. LSTM along with some clustering techniques have been tested on sample language modelling tasks and the results showed relatively a significant rise in performance when compared to traditional Language Modelling(LM).

CHAPTER 3

Background Knowledge

3.1 Language Modelling for Chess Games

Since a chess and natural language share many similarities, we focus on chess as sequence of moves. As such, we use a recurrent model which takes a sub-sequence and tells the probability of winning if the game contains that sub-sequence.

Ex : A sub-sequence of a chess game be .... E6E4 F6F8 F2A2 ....

This sub-sequence is sent as an input to the model and the model will tell the probability that the game will win if the game has this sub-sequence. When we train these kind of models, the function tries to map the relationships between the moves, using past information as the context

3.1.1 Approaches

3.1.1.1 RNN

A major characteristic of most neural networks is that they have no memory. Each input shown to them gets processed independently, with no state kept in between inputs. They cant process a sequence or a temporal series of data points. Recurrent

Neural Networks (RNNs) adopt the same principle, they process sequences by iterating through the sequence elements and maintaining a "state" containing information relative to what they have seen so far. Each member of the output is a function of the previous members of the output. s(t)= f(s(t-1),p) p=parameter .

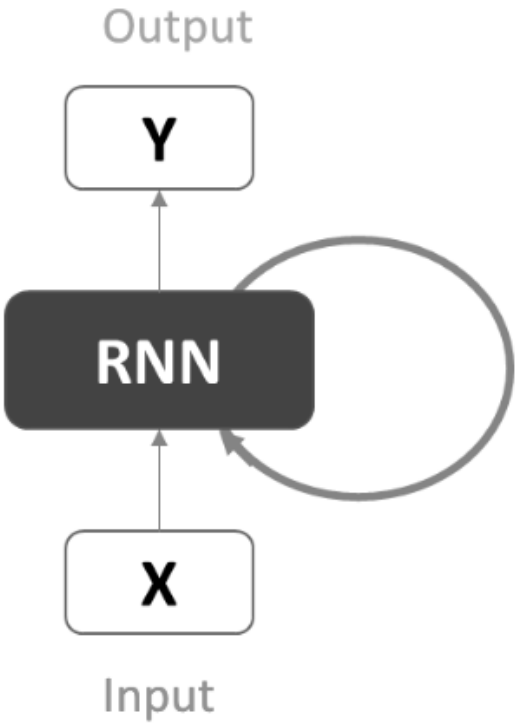

FIGURE 3.1: RNN Unit

Parameter sharing makes it possible to extend and apply the model to examples of different forms (different lengths, here) and generalize across them. Each member of the output is produced using the same update rule applied to the previous outputs. Easy enough: in summary, a RNN is just a loop that reuses quantities computed during the previous iteration of the loop.

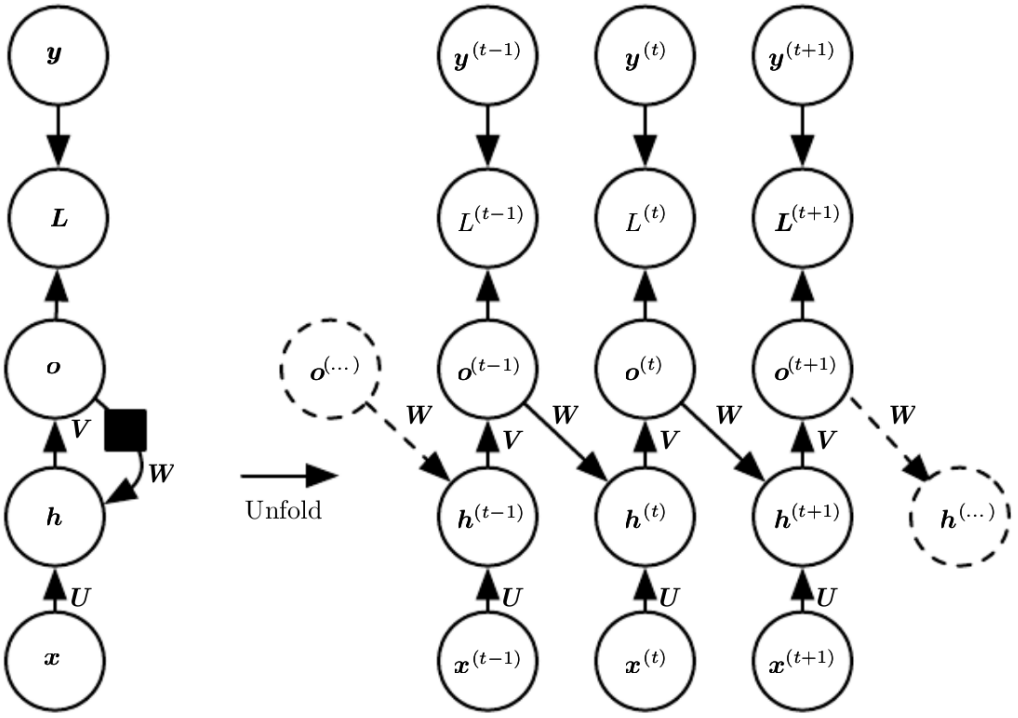

FIGURE 3.2: RNN Unit- Unrolled in time

each timestep is the output of the loop at time t and contains information about timesteps from 0 to t in the input sequenceabout the entire past

3.1.1.2 LSTM

Like leaky units, gated RNNs are based on the idea of creating paths through time that have derivatives that neither vanish nor explode. Leaky units did this with connection weights that were either manually chosen constants or were parameters. Gated RNNs generalize this to connection weights that may change at each time step. Leaky units allow the network to accumulate information (such as evidence for a particular feature or category) over a long duration. However, once that information has been used, it might be useful for the neural network to forget the old state.

The clever idea of introducing self-loops to produce paths where the gradient can flow for long duration is a core contribution of the initial long short-term memory (LSTM)

model . A crucial addition has been to make the weight on this self-loop conditioned on the context, rather than fixed. By making the weight of this self-loop gated (controlled by another hidden unit), the time scale of integration can be changed dynamically. In this case, we mean that even for an LSTM with fixed parameters, the time scale of integration can change based on the input sequence, because the time constants are output by the model itself.

Instead of a unit that simply applies an elementwise non-linearity to the affine transformation of inputs and recurrent units, LSTM recurrent networks have LSTM cells that have an internal recurrence (a self-loop), in addition to the outer recurrence of the RNN.

LSTM networks have been shown to learn long-term dependencies more easily than the simple recurrent architectures, first on artificial data sets designed for testing the ability to learn long-term dependencies then on challenging sequence processing tasks where state-of-the-art performance was obtained.

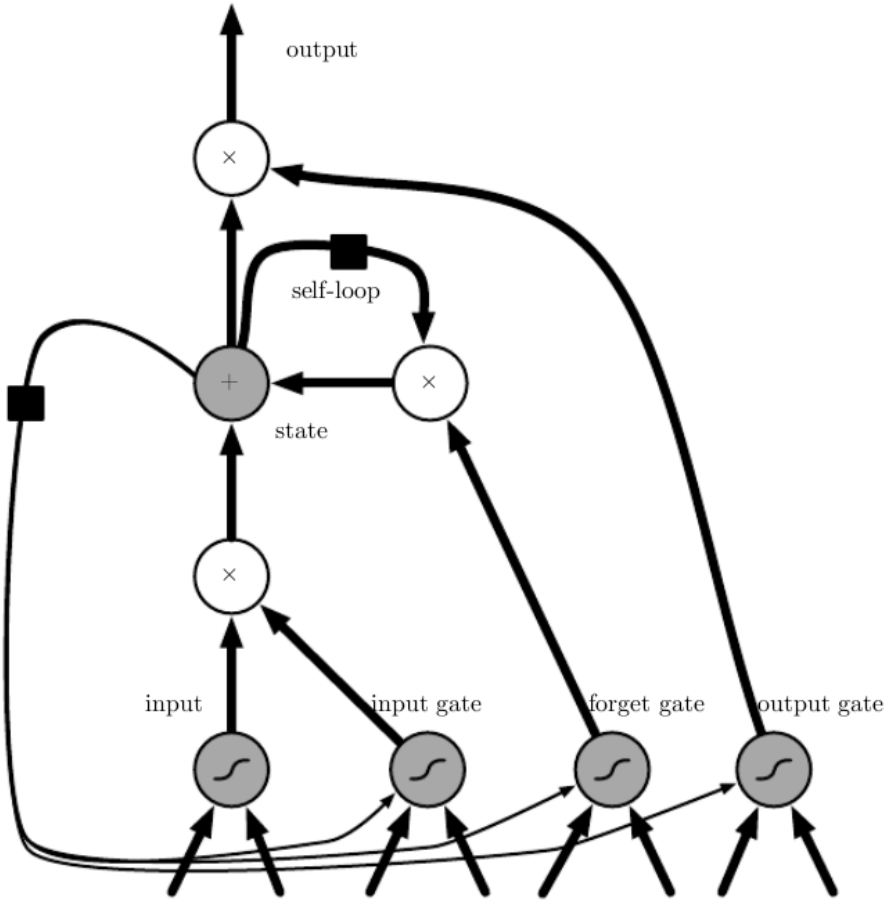

FIGURE 3.3: LSTM unit

3.1.2 Why we choose LSTM ?

- Chess is sequential and a normal neural network will not preserve the order and see the time window as combination of different moves. LSTM helps preserve the temporal dependency. Further, LSTM does not suffer from vanishing/exploding gradient problem that makes the RNN not as appealing.
- The form of data in training set given the selected model will enable the AI to train over all possible sequences that could occur during a game, allowing it to be used as the core of our chess Engine.

3.2 Word Embeddings

While the vectors obtained through one-hot encoding are binary, sparse (mostly made of zeros) and very high-dimensional (same dimensionality as the number of words in the vocabulary), "word embeddings" are low-dimensional floating point vectors (i.e. "dense" vectors, as opposed to sparse vectors).Unlike word vectors obtained via one-hot encoding, word embeddings are learned from data.Word embeddings pack more information into far fewer dimensions.

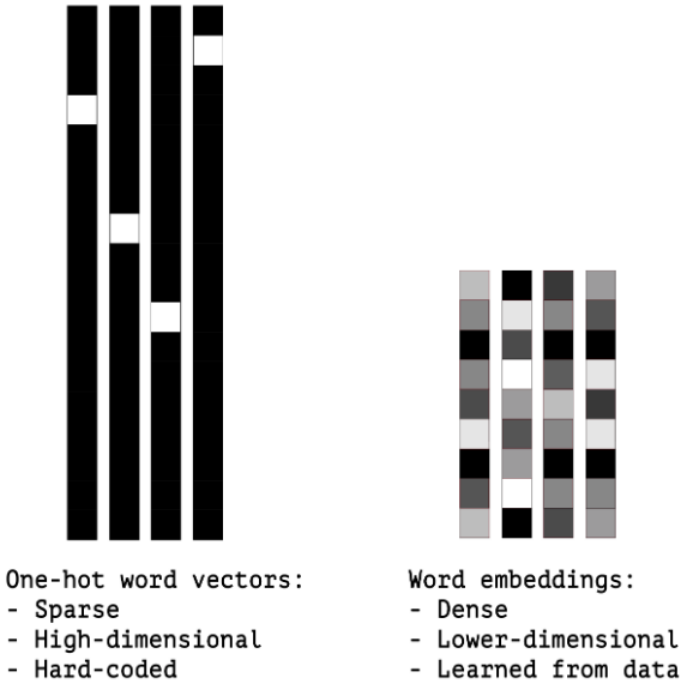

FIGURE 3.4: one-hot encoding vs word embedding

Two ways to obtain word embeddings:

- Learnt jointly with the main task.(Start with random word vectors, then learn your word vectors in the same way that you learn the weights of a neural network)
- Pre-computed using a different machine learning task than the one you are trying to solve("pre-trained word embeddings").

Word embeddings are meant to map human language into a geometric space. In a reasonable embedding space, synonyms are embedded into similar word vectors and words meaning very different things would be embedded to points far away from each other, while related words would be closer.
We would be learning a task-specific embedding of our input tokens, which is generally more powerful than pre-trained word embeddings when lots of data is available.

From raw data to word embeddings:

- Download the raw data.
- Tokenize the data.
- Learning the embeddings.

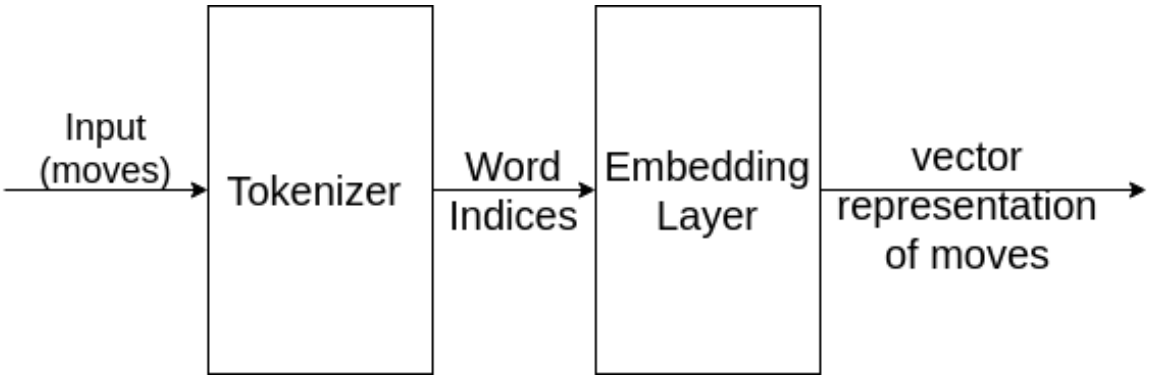

FIGURE 3.5: Process of converting a move to vector representation using word embedding

CHAPTER 4

Design Methodology

4.1 The Basic Idea

Since we are using sequences to evaluate 'how the game is going', we first need to create a function that can determine which sequences are worth playing. The next part is using this function in an ongoing game. Section 7.2 and 7.3 lays down the approach of learning such a function and utilizing it in a gameplay.

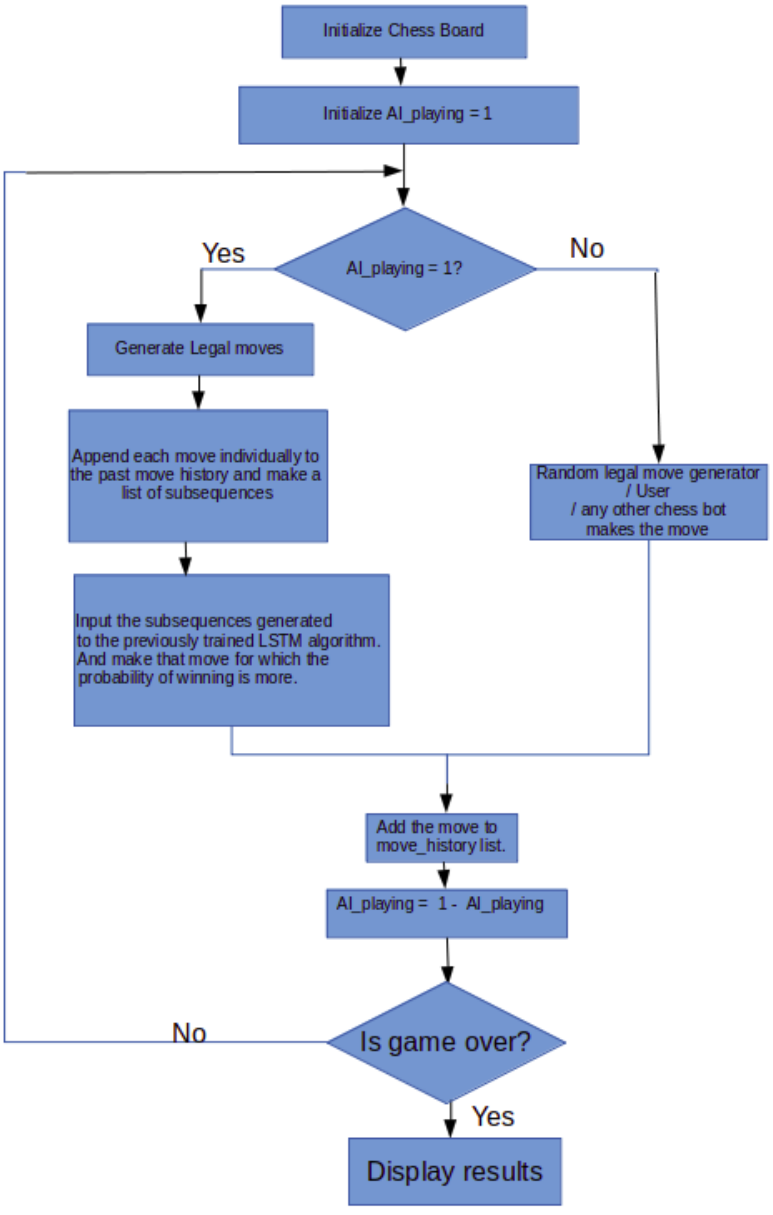

FIGURE 4.1: Design of Chess engine

4.2 Data

We use CCRL(Computer Chess Rating Lists) 40/40 Dataset. The dataset consists of 1,044,560 games played between expert chess engines encoded in PGN(portable game notation) format.

```
[Event "CCRL 40/40"]
[Site "CCRL"]
[Date "2005.12.20"]
[Round "1.1.1"]
[White "Gandalf 6"]
[Black "Fritz 9"]
[Result "1-0"]
[ECO "D85"]
[Opening "Gruenfeld"]
[Variation "modern exchange variation"]
[PlyCount "83"]
[WhiteElo "2633"]
[BlackElo "2743"]

1. d4 Nf6 2. c4 g6 3. Nc3 d5 4. cxd5 Nxd5 5. e4 Nxc3 6. bxc3 Bg7 7. Nf3 c5 8.
Rb1 O-O 9. Be2 cxd4 10. cxd4 Qa5+ 11. Qd2 Qxd2+ 12. Bxd2 b6 13. O-O Bb7 14. Bd3
Rd8 15. Be3 Bxd4 16. Nxd4 e5 17. Nb5 Rxd3 18. Nc7 Nc6 19. Nxa8 Bxa8 20. Rfd1
Rc3 21. Rbc1 Rxc1 22. Bxc1 Kg7 23. f3 Kf6 24. Ba3 Nd4 25. Bb2 Nc6 26. Rd5 Ke6
27. Ba3 f5 28. Rd6+ Kf7 29. exf5 gxf5 30. h4 a5 31. h5 b5 32. Rd7+ Kf6 33. Rxh7
b4 34. Rh8 bxa3 35. Rg8 Kf7 36. Rxa8 Nd4 37. Kf2 f4 38. Rxa5 Nc6 39. Rxa3 Nb4
40. Ra7+ Kg8 41. a4 Nd3+ 42. Kg1 1-0
```

FIGURE 4.2: Dataset

4.2.1 Analysing the Data

One of the issues that we will encounter forming a dictionary of moves based on the games the engine is trained on doesn't necessarily implies that it will encounter all the possible moves. Especially if the number of moves keeps increasing.

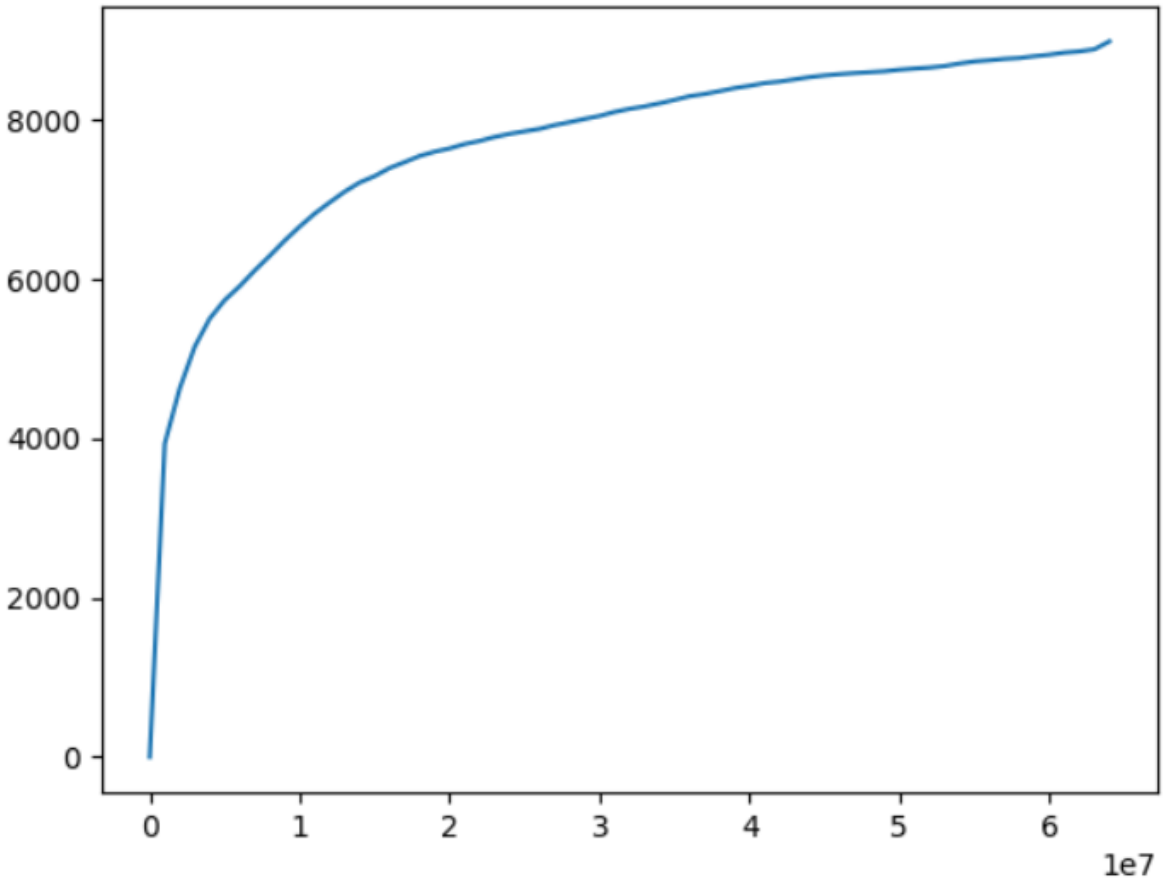

FIGURE 4.3: X: Number of Games, Y: Number of Unique Moves

As is apparent from the inverse logarithmic curve in the graph, the number of unique moves is limited. If the dataset is varied and large enough, it will capture most moves played in any chess game.

The next problem is choosing a size of the window, which allows us to capture information of past moves.

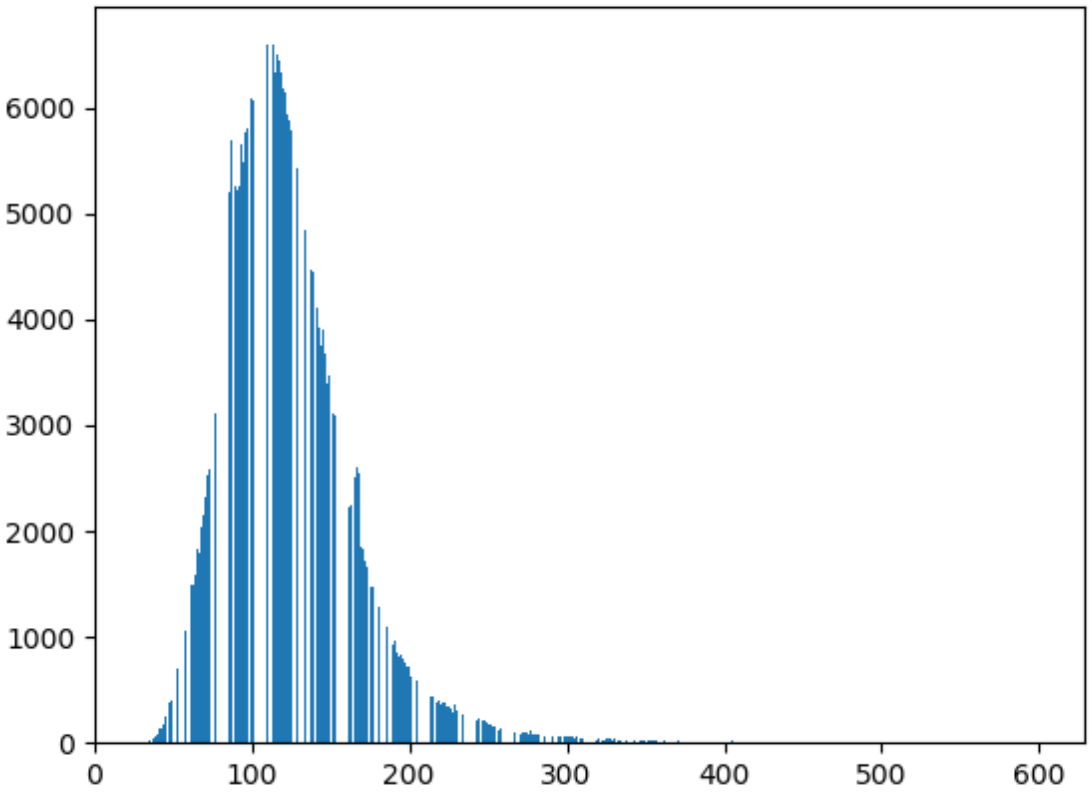

FIGURE 4.4: X: Length Of Game, Y: Number of Games

We see that most game length are between 90-200 moves. We test performance with different length of of window size.

4.2.2 Preprocessing Data

The next task is to extract sequences from the PGN file. We use the following code to automate the task.

```
import re
def get_general_data(stored_game):
    outcomes = []
    games = []
    for i in range(len(stored_game)):
        if stored_game[i][-1] == '0' or stored_game[i][-1] == '1': # or stored_game[i][-1] == '1':
```

```
            game = re.sub(r'\d*\. ?', "", stored_game[i][:-3].rsplit(' ', 1)[0])
            outcome = stored_game[i][-3:].split('-')[0]
            outcomes.append(outcome.split('-')[0])
            games.append(game)

    file = open('dataset.txt', 'w+')
    for i in range(len(games)):
        file.write(games[i] + ' ' + outcomes[i] + '\n')

    file.close()

    return(len(games))

if __name__ == '__main__':
    file = open('CCRL.pgn', 'r')
    game = ''
    stored_game = []
    for i in range(12500000):
        line = file.readline()
        if line[0] != '\n' and line[0] != '[':  # found game sequence
            game = line
            while True:
                forseq = file.readline()
                if forseq[0] != '[':
                    game += forseq
                else:
                    break
            stored_game.append(game.replace('\n', ' ')[:-2])
            game = ''

    file.close()
    print(get_general_data(stored_game))
```

The data is now lines of games, each consisting of a sequence of moves followed by the outcome of the game
Example: Nf3 c5 e4 d6 d4 cxd4 Nxd4 Nf6 Nc3 a6 Bc4 e6 O-O b5 Bb3 b4 Na4 Nxe4 Re1 Nc5 Nxc5 dxc5 Ba4+ Bd7 Nxe6 fxe6 Qh5+ g6 Rxe6+ Kf7 Qd5 Bxe6 Qxd8 1

4.3 Sequence Evaluation Function

The sequence evaluation function is to take as input a subsequence in a game, map it to a probability [0, 1]. We do this by training a model on subsequences in chess and supervising the training by the output of the game.

4.3.1 Methodology

1. Dataset: CCRL 40/40 PGN dataset is used which consists of 1 million games.
2. Filter only the sequences that lead to a win or loss.
3. Select a window size: number of moves in a subsequence.
4. We pickup a random subsequence from a chess game, randomly selected from the dataset until we have the desired number of samples with their corresponding output of games(1/0)
5. These sequences are converted to tokens using a tokenizer.
6. These tokenized sequences are used to train the LSTM network while supervising the training by the observed output of the game.
7. This LSTM network is trained until a reasonable accuracy is achieved when evaluating subsequence that may be helpful in winning the game. I outputs the probability of the sequence leading to a win.

4.3.2 Sequence Evaluation Function: Code

```
file = open('dataset.txt', 'r')
games = []
outcomes = []
for line in file.readlines():
    x = line.strip('\n').split(' ')
    games.append(x[:-1])
    outcomes.append(int(x[-1]))

#adding context to dataset


for i in range(len(games)):
    for j in range(0, len(games[i])):
        if j % 2 == 1:
            games[i][j] = '~' + games[i][j]
        else:
            games[i][j] = '@' + games[i][j]

file.close()
maxlen = 50
```

```
print(maxlen)

# preprocessing
import pandas as pd
import pickle

data = pd.DataFrame({'game': games, 'outcome': outcomes})
X, y = (data['game'].values, data['outcome'].values)
from keras.preprocessing.text import Tokenizer
# from keras.preprocessing.sequence import pad_sequences
tk = Tokenizer(filters=       , lower=True)
tk.fit_on_texts(X)
f = open('tokenized.pkl', 'wb')
pickle.dump(tk, f)
X_seq = tk.texts_to_sequences(X)

# X_pad = pad_sequences(X_seq, maxlen=maxlen, padding='post', truncating='post')
# print(X_seq[0])
import numpy as np
import random
no_games = 0

X_padded = []
Y_padded = []
while no_games < 1000000:
    this_game = random.randint(0, len(X_seq)-1)
    game = X_seq[this_game]
    # move = random.choice(range(maxlen, len(game) - 1))
    # game_strip = game[move-maxlen: move]
    move = random.choice(range(0, len(game) - 1))     #maxlen

    if move - maxlen < 0:
        game_strip = [0] * maxlen
        for i in range(move + 1):
            game_strip[maxlen - i - 1] = game[move - i]
    else:
        game_strip = game[move - maxlen:move]
    # print(str(game_strip) + ' ' + str(y[this_game]))
    X_padded.append(game_strip)
    Y_padded.append(y[this_game])
    no_games += 1


from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(np.array(X_padded), np.array(Y_padded), test_s


batch_size = 64

# Generating the Model
from keras.models import Sequential
```

```
from keras.layers import Embedding, LSTM, Dense, CuDNNLSTM, CuDNNGRU
vocabulary_size = len(tk.word_counts.keys())+1
max_words = maxlen
embedding_size = 64
model = Sequential()
model.add(Embedding(vocabulary_size, embedding_size, input_length=max_words))
model.add(CuDNNLSTM(64, return_sequences=True))
model.add(CuDNNLSTM(64, return_sequences=True))
model.add(CuDNNLSTM(64, return_sequences=True))
model.add(CuDNNLSTM(64, return_sequences=True))
model.add(CuDNNLSTM(64, return_sequences=True))
model.add(CuDNNLSTM(64, return_sequences=False))
# model.add(CuDNNLSTM(128, return_sequences=True))
# model.add(CuDNNLSTM(128, return_sequences=False))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.summary()

# Validation & Early Stopping
from keras.callbacks import ModelCheckpoint, EarlyStopping

filepath = 'discriminate_weights_good1mil50_64_64_6x64.hdf5'
early_stop_checkpoint = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=2, min_d
save_checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True, sa
callbacks_list = [early_stop_checkpoint, save_checkpoint]

# Training the Model
foo = model.fit(X_train, y_train, validation_split=0.05, batch_size=batch_size, epochs=1, callback
print(foo)
# Testing the Model
scores = model.evaluate(X_test, y_test, batch_size=128, verbose=0)
print("Test accuracy", scores[1])
```

4.4 Chess Engine

Given a board state, we have the past sequence of actions. From this point, we can generate a depth limited game tree, and merge it with past sequence. This will generate all the possible sequences. The Sequence Evaluation Function takes as input all the possible sequences and returns the sequences with maximum probability of winning. The corresponding move is then played by the AI.

4.4.1 Methodology

1. Initialize the chess board. Choose AI as white.
2. Initialize the sequence list (Initially empty of size T: [0] * T)
3. If ChanceAI = 1:
4. Chess engine finds out all the legal moves of current board state.
5. Sequences are created based on previous moves and enqueing current possible moves in the game tree upto a certain depth[we choose depth = 1]. Left padding is done when necessary to keep sequences of fixed size, generally during starting of the game.
6. These sequences are given as input to the Sequence Evaluation Funtion. The function returns the sequence that gives the highest probability of winning the game.
7. The move corresponding to best sequence is selected and moved.
8. Else:
9. The user decides their best move(in case of random agent, its random move will be considered the best move.
10. Check the board for possible check mate, stale mate, move by 50 etc. If any of the conditions are satisfied, we end the game and declare the result, else continue,
11. Switch the chance. Move to step 3.

4.4.2 Chess Engine: Code

```
import pickle
maxlen = 100
weight_path = 'discriminate_weights_good1mil50_64_64_6x64.hdf5'
from keras_preprocessing.text import Tokenizer
f = open('tokenized.pkl', 'rb')
tk = pickle.load(f)

from keras.models import Sequential
```

```
from keras.layers import Embedding, LSTM, Dense, CuDNNLSTM
import numpy as np

vocabulary_size = len(tk.word_counts.keys())+1
max_words = maxlen
embedding_size = 64
model = Sequential()
model.add(Embedding(vocabulary_size, embedding_size, input_length=max_words))
model.add(CuDNNLSTM(64, return_sequences=True))
model.add(CuDNNLSTM(64, return_sequences=True))
model.add(CuDNNLSTM(64, return_sequences=True))
model.add(CuDNNLSTM(64, return_sequences=True))
model.add(CuDNNLSTM(64, return_sequences=True))
model.add(CuDNNLSTM(64, return_sequences=False))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.load_weights(weight_path)

# file = open('random_agent_data.txt', 'a')
def find_best_sequence(move_list):
    test = np.array(move_list)
    score = model.predict(test)
    best_index = np.argmax(score)
    return best_index, move_list[best_index]

def find_worst_sequence(move_list):
    test = np.array(move_list)
    score = model.predict(test)
    best_index = np.argmin(score)
    return best_index, move_list[best_index]


AI = 0
Random = 0
import chess
import random
import re
import time
for x in range(10000):
    game_sequence = ''
    board = chess.Board()
    # print(board)
    end = 0
    past_moves = [0] * maxlen # maxlen, no. of moves to consider from past
    AIisW = random.choice(range(2))
    chanceAI = AIisW

    while not end:
        legal_moves = re.sub(',', '', str(board.legal_moves).split('(')[1][:-2]).split(' ')
        # print(legal_moves)
        past_moves = past_moves[1:]
```

```
# print(past_moves)
current_move = ''

if chanceAI == 1:
    possible_seq = []
    if AIisW == 1:     # white
        for i in range(len(legal_moves)):
            a = '@' + legal_moves[i].lower()
            if a in tk.word_index.keys():
                possible_seq.append(past_moves + [tk.word_index[a]])
        best_move_index, past_moves = find_best_sequence(possible_seq)
    else:               # black
        for i in range(len(legal_moves)):
            a = '~' + legal_moves[i].lower()
            if a in tk.word_index.keys():
                possible_seq.append(past_moves + [tk.word_index[a]])
        best_move_index, past_moves = find_worst_sequence(possible_seq)
    best_move = legal_moves[best_move_index]
    # print("ChessAI: " + best_move)

else:
    #Random Agent
    best_move = random.choice(legal_moves)
    if AIisW == 0:  # Random is white
        a = '@' + best_move.lower()
    else:           # Random is black
        a = '~' + best_move.lower()
    if a in tk.word_index.keys():
        past_moves.append(tk.word_index[a])
    else:
        past_moves.append(0)
    # print("Random: " + best_move)

game_sequence += best_move + ' '
board.push_san(best_move)
# print(board)

if board.is_game_over():
    if chanceAI == 1:
        end = 1
        AI += 1
        print(str(x + 1) + ". Game Over: AI Wins " + str(AI))
        if AIisW:
            outcome = 1
        else:
            outcome = 0
    else:
        end = 1
        Random += 1
        print(str(x + 1) + ". Game Over: Random Win " + str(Random))
        if not AIisW:
```

```
                    outcome = 0
                else:
                    outcome = 1

            # file.write(game_sequence + str(outcome) + '\n')
            break
        chanceAI = 1 - chanceAI


print("AI : ", AI)
print("Random: ", Random)
```

CHAPTER 5

Results and Observations

5.1 Results: Sequence Evaluation Function

The model is trained on 74.5%, validated on 0.05% and tested on 25% of data. The network was trained on a system with following configuration:

1. Intel i7 7700HQ
2. Nvidia GTX 1050 4GB DDR5 RAM
3. 16 GB DDR4 RAM

Name	Samples & CI	Window Size	Architecture	Accuracy
Agent 1	1mil	50	64 + 6x64	69.44
Agent 2	1mil	100	64 + 6x64	70.37
Agent 3	5mil	50	64 + 6x64	73.61

Table 5.1: Model Test Accuracy

5.2 Results: The Chess Engine

This chess engine is pit against an agent that makes random moves.

Agent	Win Rate
Agent 1	67.66
Agent 2	69.84
Agent 3	66.40

TABLE 5.2: Win Rate (1000 games)

5.3 Findings

- The results proves that the engine knows the difference between good and bad sequences and exploits this to either win or loose the game. The difficulty of the engine can be modified easily
- The difficulty of the engine can be modified easily by instead of selecting the best(argmax) sequences, we choose the 2nd, 3rd .... best sequences to lower the difficulty.

5.4 Benefits

- The engine is capable of extracting patterns from games between any two players so a dataset can be generated of all the games between player 1 and 2 and train on the dataset there by either imitating or learning patterns allowing it to win.
- Since the engine basically evaluates sequences it can also be extended to evaluate a game tree at higher depth.
- Credit Assignment Problem. It follows almost the same structure as the model we created. [a1, a2, a3, a4. aN] is a set of actions performed in some environment resulting in a reward R. A dataset of such actions sequences and rewards, can then be modeled in a similar fashion, such than it can assign a credit score to an action, given the past actions forming the context.

5.5 Drawbacks

- The engine was trained on the data of the players at grand master levels. These players follow precise logical sequences in contrast to new(noob) players as such which is why the engine doesnt perform as good when playing against random players.It is unable to generalise what it has learned in professional game plays to simple game plays.

- The current form of implementation takes into account only the past moves to make a decision as compared to a standard min-max approach which takes into account the future possible board positions.

CHAPTER 6

Conclusion and Future Work

We conclude that representation of Chess in terms of PGN notation encodes enough information about the chess, however the extent of its usability is pretty limited. Also we conclude that accuracy is constrained in the context of the type of players on which the model was trained on and therefore accuracy isn't the best judge of measuring the performance/capability of the model.

Further improvements to the Chess Engine can be made by training on more varied dataset. We could also improve the chess engine by incorporating an ensemble of LSTM models trained for different lengths of subsequences.

References

[1] Mikolov, Tom / Karafit, Martin / Burget, Luk / ernock, Jan / Khudanpur, Sanjeev (2010): "Recurrent neural network based language model", INTERSPEECH-2010

[2] Martin Sundermeyer / Ralf Schlter / Hermann Ney / "LSTM Neural Networks for Language Modeling" INTERSPEECH-2012

[3] Omid E. David / Nathan S. Netanyahu / Lior Wolf /. (2016). "DeepChess: End-to-End Deep Neural Network for Automatic Learning in Chess" arXiv:1711.09667

[4] Deep Learning written by Yoshua Bengio, Ian Goodfellow, Aaron Coreville

[5] Deep Learning with Python, Franois Chollet