Convolutional Neural Networks learn to play Chess

July 15, 2015

Ashudeep Singh
ashudeep@iitk.ac.in

Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur
India

Outline

- **1950**: Claude Shannon published “Programming a Computer for playing Chess”

- **1950**: Claude Shannon published "Programming a Computer for playing Chess"
- **1951**: Alan Turing published a program, developed on paper, that was capable of playing a full game of chess.

- **1950**: Claude Shannon published “Programming a Computer for playing Chess”
- **1951**: Alan Turing published a program, developed on paper, that was capable of playing a full game of chess.
- **1956**: John McCarthy invented the alpha-beta search algorithm.

- **1950**: Claude Shannon published “Programming a Computer for playing Chess”
- **1951**: Alan Turing published a program, developed on paper, that was capable of playing a full game of chess.
- **1956**: John McCarthy invented the alpha-beta search algorithm.
- **1957**: Alex Bernstein and a group of Russian programmers separately developed programs capable of playing a full game of chess.

- **1950**: Claude Shannon published “Programming a Computer for playing Chess”
- **1951**: Alan Turing published a program, developed on paper, that was capable of playing a full game of chess.
- **1956**: John McCarthy invented the alpha-beta search algorithm.
- **1957**: Alex Bernstein and a group of Russian programmers separately developed programs capable of playing a full game of chess.
- **1978**: David Levy wins the bet made 10 years earlier, defeating Chess 4.7 in a six-game match by a score of $4\frac{1}{2} - 1\frac{1}{2}$. The computer’s victory in game four is the first defeat of a human master in a tournament.

- **1950**: Claude Shannon published "Programming a Computer for playing Chess"
- **1951**: Alan Turing published a program, developed on paper, that was capable of playing a full game of chess.
- **1956**: John McCarthy invented the alpha-beta search algorithm.
- **1957**: Alex Bernstein and a group of Russian programmers separately developed programs capable of playing a full game of chess.
- **1978**: David Levy wins the bet made 10 years earlier, defeating Chess 4.7 in a six-game match by a score of $4\frac{1}{2} - 1\frac{1}{2}$. The computer's victory in game four is the first defeat of a human master in a tournament.
- **1996**: Deep Blue is defeated by Garry Kasparov.

- **1950**: Claude Shannon published “Programming a Computer for playing Chess”
- **1951**: Alan Turing published a program, developed on paper, that was capable of playing a full game of chess.
- **1956**: John McCarthy invented the alpha-beta search algorithm.
- **1957**: Alex Bernstein and a group of Russian programmers separately developed programs capable of playing a full game of chess.
- **1978**: David Levy wins the bet made 10 years earlier, defeating Chess 4.7 in a six-game match by a score of $4\frac{1}{2} - 1\frac{1}{2}$. The computer’s victory in game four is the first defeat of a human master in a tournament.
- **1996**: Deep Blue is defeated by Garry Kasparov.
- **1997**: Deep Blue defeats Garry Kasparov.

- **1950**: Claude Shannon published “Programming a Computer for playing Chess”
- **1951**: Alan Turing published a program, developed on paper, that was capable of playing a full game of chess.
- **1956**: John McCarthy invented the alpha-beta search algorithm.
- **1957**: Alex Bernstein and a group of Russian programmers separately developed programs capable of playing a full game of chess.
- **1978**: David Levy wins the bet made 10 years earlier, defeating Chess 4.7 in a six-game match by a score of $4\frac{1}{2} - 1\frac{1}{2}$. The computer’s victory in game four is the first defeat of a human master in a tournament.
- **1996**: Deep Blue is defeated by Garry Kasparov.
- **1997**: Deep Blue defeats Garry Kasparov.
- **2006**: The undisputed world champion, Vladimir Kramnik, is defeated 4–2 by Deep Fritz.

In 1950, Shannon's "Programming a Computer for playing Chess" pointed out that methods in which the chess computers of the future will play chess can be divided into two categories:

- **Type A**: a brute-force search looking at every variation upto a given depth.
- **Type B**: a selective search looking at "important" branches only.

Shannon and early programmers started out with "Type B" kind of programs which stayed popular until the 1970's after which the "Type A" programs had enough processing power and more efficient brute force algorithms to become stronger.
Today most of the programs are closer to "Type A", but have some characteristics of "Type B" programs called selectivity.

We aim to play chess as more of a pattern recognition task, as opposed to the conventional approach of accomplishing it as a predominantly search problem.

Aspects of expert human chess playing

We aim to play chess as more of a pattern recognition task, as opposed to the conventional approach of accomplishing it as a predominantly search problem.

- **Adrian de Groot (1996)**
  - He studied chess players' transcripts of verbal utterances and their eye gaze movement and also interviewed a number of beginner and master level players.
  - He concluded that although all players essentially examine 40-50 positions before playing a move, master level players develop pattern recognition skills from experience which helps them examine a fewer lines to a greater depth.

We aim to play chess as more of a pattern recognition task, as opposed to the conventional approach of accomplishing it as a predominantly search problem.

- **Adrian de Groot (1996)**
- **Chase and Simon (1973)**
  - Master level players are capable of recognizing familiar patterns and specific arrangements of the board.
  - They also have a capability to learn from the weaknesses of the opponent or from their own mistakes.

We aim to play chess as more of a pattern recognition task, as opposed to the conventional approach of accomplishing it as a predominantly search problem.

- **Adrian de Groot (1996)**
- **Chase and Simon (1973)**
- **Gobet and Clarkson (2004)**
  - Nor are the grandmasters known to have an exceptional IQ (Bilalić et al. 2007), nor are they known to examine the game tree more rapidly (de Groot, 1996).
  - The difference however is that the mental representation of game states is in terms of larger chunks, so that positions and possible actions are encoded more efficiently.
  - This also means that the space no longer of distinct boards, but is a space consisting of chunks where the pattern matching is much more efficient.

Aspects of expert human chess playing

We aim to play chess as more of a pattern recognition task, as opposed to the conventional approach of accomplishing it as a predominantly search problem.

- **Adrian de Groot (1996)**
- **Chase and Simon (1973)**
- **Gobet and Clarkson (2004)**
- **Ericsson et al. (2007)**
  - In his article “How to become an expert?”, he says “Experts are made, never born”.
  - He studied expertise in various domains and concluded that the amount and quality of practice were key factors in the level of performance achieved.
  - He alo mentioned that the training needs to be deliberate, meaning that it consists of considerable and sustained efforts to learn something that you can’t already do well.
  - In the match where Garry Kasparov defeated Deep Blue in 1996, he lost the first game but came back strongly to win the match by 4-2. The reason Kasparov could win was that he recognized Deep Blue’s weaknesses and captilized on them, while Deep Blue relied on human programmers to tweak strategies and cover weaknesses.

The capability to learn chess can be accomplished if a machine can:

- Learn to play legal moves
- Rank the possible moves without any explicit guidance on relative importance of material or position.
- Evaluate board positions

To accomplish this task using minimal knowledge prior, we use Convolutional neural networks to learn the game directly from board positions, moves and outcomes.

Related Work

Convolutional Neural Networks for playing Go

- Sutskever and Nair (2008) attempted to use Convolutional Neural Networks to learn Go in 2008 with a small network and achieved modest success.

- Sutskever and Nair (2008) attempted to use Convolutional Neural Networks to learn Go in 2008 with a small network and achieved modest success.
- Maddison et al. (2014) used deep CNNs to predict moves in Go.

- Sutskever and Nair (2008) attempted to use Convolutional Neural Networks to learn Go in 2008 with a small network and achieved modest success.
- Maddison et al. (2014) used deep CNNs to predict moves in Go.
  - The model could predict 55% of the moves correctly.

- Sutskever and Nair (2008) attempted to use Convolutional Neural Networks to learn Go in 2008 with a small network and achieved modest success.
- Maddison et al. (2014) used deep CNNs to predict moves in Go.
  - The model could predict 55% of the moves correctly.
  - It could beat GnuGo in 97% of the games without any search.

- Sutskever and Nair (2008) attempted to use Convolutional Neural Networks to learn Go in 2008 with a small network and achieved modest success.
- Maddison et al. (2014) used deep CNNs to predict moves in Go.
  - The model could predict 55% of the moves correctly.
  - It could beat GnuGo in 97% of the games without any search.
  - It was also able to reach the performance level of a state-of-the-art Monte-Carlo tree search algorithm which simulated 1M moves per second.

- Sutskever and Nair (2008) attempted to use Convolutional Neural Networks to learn Go in 2008 with a small network and achieved modest success.
- Maddison et al. (2014) used deep CNNs to predict moves in Go.
  - The model could predict 55% of the moves correctly.
  - It could beat GnuGo in 97% of the games without any search.
  - It was also able to reach the performance level of a state-of-the-art Monte-Carlo tree search algorithm which simulated 1M moves per second.
- **Go vs Chess**:

- Sutskever and Nair (2008) attempted to use Convolutional Neural Networks to learn Go in 2008 with a small network and achieved modest success.
- Maddison et al. (2014) used deep CNNs to predict moves in Go.
  - The model could predict 55% of the moves correctly.
  - It could beat GnuGo in 97% of the games without any search.
  - It was also able to reach the performance level of a state-of-the-art Monte-Carlo tree search algorithm which simulated 1M moves per second.
- **Go vs Chess**:
  - The existing state of the art systems are weaker compared to the best human players, unlike Chess where the best computers nowadays are far above human performance.

- Sutskever and Nair (2008) attempted to use Convolutional Neural Networks to learn Go in 2008 with a small network and achieved modest success.
- Maddison et al. (2014) used deep CNNs to predict moves in Go.
  - The model could predict 55% of the moves correctly.
  - It could beat GnuGo in 97% of the games without any search.
  - It was also able to reach the performance level of a state-of-the-art Monte-Carlo tree search algorithm which simulated 1M moves per second.
- **Go vs Chess**:
  - The existing state of the art systems are weaker compared to the best human players, unlike Chess where the best computers nowadays are far above human performance.
  - The moves in Go are much smoother as they add just 1 pixel to the game board image and the random chance of getting a move correct is $\frac{1}{361}$ as compared to chess which has $\frac{1}{4096}$.

- Sutskever and Nair (2008) attempted to use Convolutional Neural Networks to learn Go in 2008 with a small network and achieved modest success.
- Maddison et al. (2014) used deep CNNs to predict moves in Go.
  - The model could predict 55% of the moves correctly.
  - It could beat GnuGo in 97% of the games without any search.
  - It was also able to reach the performance level of a state-of-the-art Monte-Carlo tree search algorithm which simulated 1M moves per second.
- **Go vs Chess**:
  - The existing state of the art systems are weaker compared to the best human players, unlike Chess where the best computers nowadays are far above human performance.
  - The moves in Go are much smoother as they add just 1 pixel to the game board image and the random chance of getting a move correct is $\frac{1}{361}$ as compared to chess which has $\frac{1}{4096}$.
  - Chess requires a stronger domain knowledge as the rules and tactics are characterized by strict logical forms.

- Beal and Smith (1997) came up with methods to *learn* evaluation functions using handcrafted features like piece values, piece-square values and mobility.

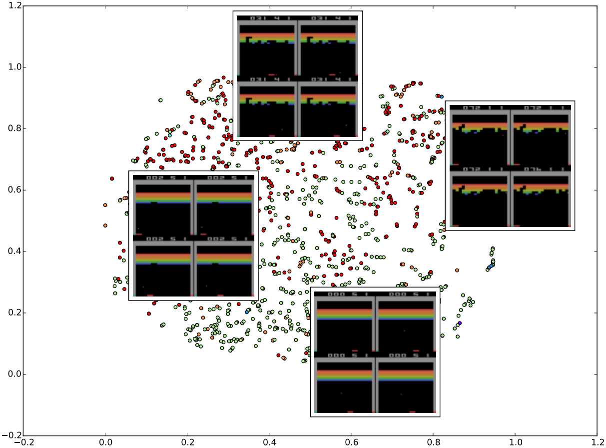
- Beal and Smith (1997) came up with methods to *learn* evaluation functions using handcrafted features like piece values, piece-square values and mobility.
- Hyatt (1999) aimed to learn opening book moves specifically.

- Beal and Smith (1997) came up with methods to *learn* evaluation functions using handcrafted features like piece values, piece-square values and mobility.
- Hyatt (1999) aimed to learn opening book moves specifically.
- Another class of chess programs that use Genetic programming learn by optimizing parameters using evolutionary methods (Bośković and Brest, 2011 and Vázquez-Fernández et al., 2012).

- Beal and Smith (1997) came up with methods to *learn* evaluation functions using handcrafted features like piece values, piece-square values and mobility.
- Hyatt (1999) aimed to learn opening book moves specifically.
- Another class of chess programs that use Genetic programming learn by optimizing parameters using evolutionary methods (Bośković and Brest, 2011 and Vázquez-Fernández et al., 2012).
- Baxter (1999) developed Knightcap and Tdleaf, which combined temporal difference learning with game-tree search to optimize the evaluation function starting from an initial evaluation function.

- Beal and Smith (1997) came up with methods to *learn* evaluation functions using handcrafted features like piece values, piece-square values and mobility.
- Hyatt (1999) aimed to learn opening book moves specifically.
- Another class of chess programs that use Genetic programming learn by optimizing parameters using evolutionary methods (Bośković and Brest, 2011 and Vázquez-Fernández et al., 2012).
- Baxter (1999) developed Knightcap and Tdleaf, which combined temporal difference learning with game-tree search to optimize the evaluation function starting from an initial evaluation function.
- Thrun (1995) developed NeuroChess which used an artificial neural network to learn the evaluation of a board as an output to certain handcrafted features input into the network. It integrates inductive neural network learning, temporal differencing, and a variant of explanation-based learning.

The main task is to directly map the visual input to a Q function which chooses action based on the states. The task is often referred to as deep Q-learning because it employs a CNN to learn the represenation of the visual space, which is further mapped to a set of actions.



Shannon described the following ideal evaluation function.

Shannon described the following ideal evaluation function.

- Assign all the positions with no further play possible as:

$$f(position) = \begin{cases} 1 \text{ , if White has won} \\ 0 \text{ , if it is a draw} \\ -1 \text{ , if Black has won} \end{cases}$$

  Use the recursive rule up the game tree:

$$f(p) = \max_{p \to p'}(-f(p'))$$

  where p' is a position reachable in one move from position p.

Shannon described the following ideal evaluation function.

- Assign all the positions with no further play possible as:

$$f(position) = \begin{cases} 1 \text{ , if White has won} \\ 0 \text{ , if it is a draw} \\ -1 \text{ , if Black has won} \end{cases}$$

  Use the recursive rule up the game tree:

$$f(p) = \max_{p \to p'}(-f(p'))$$

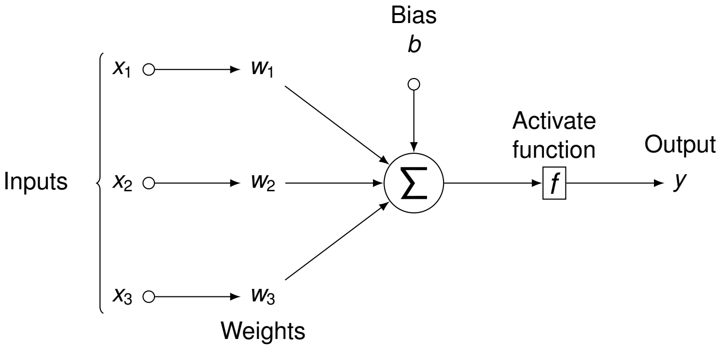
  where p' is a position reachable in one move from position p.

- Since, it is infeasible to compute such an evaluation function, we rely on approximations of f to evaluate a board position. Most of such evaluation functions have a strong component of material weights in it.

The fundamental implementation details of chess-playing computer system include:

The fundamental implementation details of chess-playing computer system include:

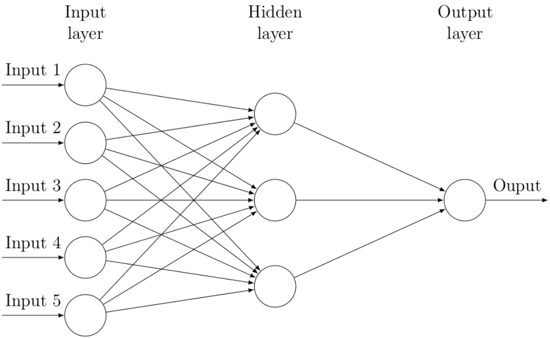
- **Board Representation** – how a board position is represented in a data structure. The performance of move generation and piece evaluation depends on the data structure used to represent the board position.
- **Search Techniques** – how to identify and select the moves for further evaluation. Some of the most widely used search techniques are– Minmax, Negamax, Negascout, Iterative deepening depth-first search etc.
- **Leaf Evaluation** – how to evaluate the position of the board if no further evaluation needs to be done.



An artificial neuron as a mathematical model

An artificial neuron inspire by a biological neuron is the basic component of an artificial neural network.
It computes $f(\sum_i w_i x_i + b)$, where f is the activation function, w_i is the weight given to the input from one of its dendrites.

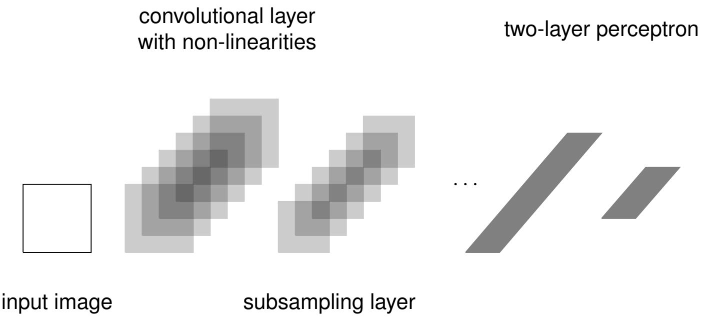
A combination of such neurons makes up an artificial neural network.



A two layer Artificial Neural network

- Hornick (1989) proved that Neural networks with at least one hidden layer are universal approximators.
- This means that given any continuous function $f(x)$ and some $\epsilon > 0$, a Neural network with one hidden layer containing a sufficient number of hidden layer neurons and a suitable choice of non-linearity, say represented by $g(x)$, exists such that $\forall x, |f(x) - g(x)| < \epsilon$.
- In other words, we can approximate any given real valued continuous function with a two layer Neural network upto a certain accuracy.
- The fact that two layer Neural networks are universal approximators is a pretty useless property in the case of machine learning. Neither does it tell the number of hidden units required to represent a given function upto the desired precision, nor does it promise that it represents a generalized function that fits the unseen data.
- The generalized function is expected to be smooth, while the overly precise two-layer network may overfit for the input data and not learn a promising representation.

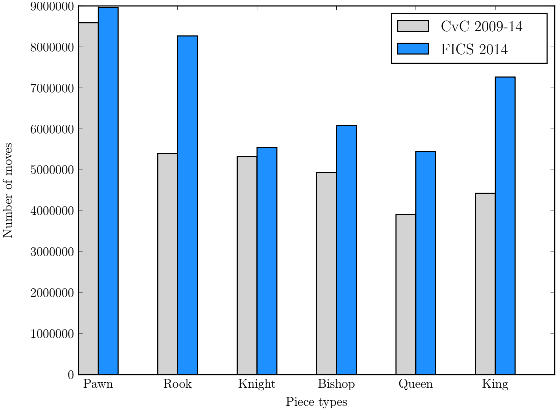
Convolutional neural networks are nothing but neural networks which use convolution instead of full matrix multiplications in atleast one of the layers (Bengio, 2015).



A typical Convolutional Neural Network architecture

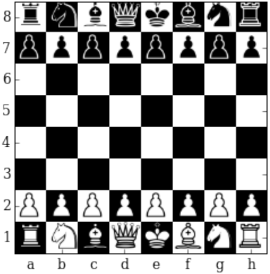
It is the most successful approach for almost all recognition and detection tasks in computer vision (Krizhevsky et al., 2012; Tompson et al., 2014; Taigman et al., 2014) and even approaches human performance on some tasks (Ciresan et al., 2012)

Dataset Name	Number of Games	Total number of boards
CvC 2009-14	370, 480	74, 162, 875
FICS 2014 (avg ELO$>$2000)	522, 356	32, 598, 059
FICS 2014 (all ELO, subset)	$\sim 1M$	$\sim 125M$



Datset

Representation



- 6 layer representation– Each piece type has one layer.
- Friendly pieces are represented as 1.
- Opponent pieces are represented as -1.

Similarly we have a 12-layer representation, where each piece type has 2 layers each. The positions with pieces contain 1s. The alternate layers are used for the pieces of each of the players.

0	0	0	0	0	0	0	0
-1	-1	-1	-1	-1	-1	-1	-1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0

-1	0	0	0	0	0	0	-1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1

0	-1	0	0	0	0	-1	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	0

0	0	-1	0	0	-1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	1	0	0	1	0	0

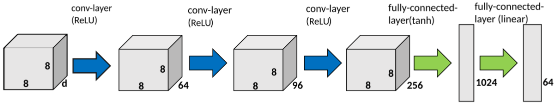
0	0	0	-1	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	-1	0	0	0	0

0	0	0	0	-1	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0

- There is more information we can provide to our model. We choose to add additional bias channels to the 6 or 12 layer data representation.
- **Piece Layer**: While making predictions for the MOVE|PIECE network, we add a channel with a 1 in the position of the predicted piece. This makes the representation consistent with $P(move|board) = P(from|board) \times P(to|from, board)$.
- **Outcome Layer**: We can provide additional information about the outcome of the game, so that each move is learned keeping in consideration the outcome of the game. The layer adds an addition $1 \times 8 \times 8$ channel to the data. It contains all 1s for the players who won, all 0s for the players who lost, 0.5 for the players who drew. While predicting, we use all 1s in the outcome layer.
- **ELO layer**: For each move, we add an additional layer with values equal to $\frac{X - MINELO}{MAXELO - MINELO}$, where X is the ELO rating of the player who played the move in the datset, MAXELO and MINELO are the maximum and minimum ELO ratings of the players in the dataset.

Piece and Move|Piece Networks

- We make use of multiple CNNs to predict the correct move when presented with a board– PIECE predictor and MOVE|PIECE predictors.
- $P(move|board) = P(from|board) \times P(to|from, board)$
- PIECE network predicts the piece. PAWN, ROOK, KNIGHT, BISHOP, QUEEN and KING predict the Move|Piece where the piece has been predicted by the PIECE predictor.

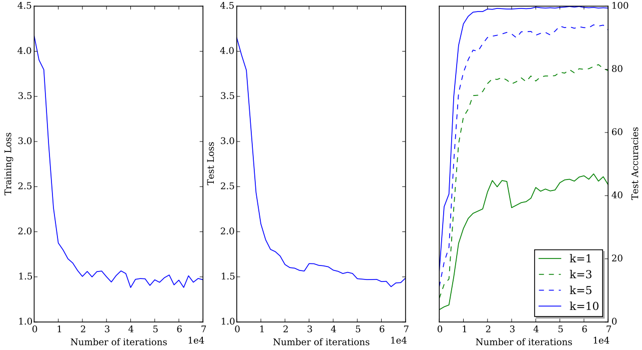


- Architecture:
  - Convolutional Layer 1: 64 filters of size 3×3 followed by ReLU
  - Convolutional Layer 2: 96 filters of size 3×3 followed by ReLU
  - Convolutional Layer 3: 256 filters of size 3×3 followed by ReLU
  - Fully Connected Layer 1: 1024 dimensions
  - Softmax layer: Outputs a 64-dimensional probability distribution

- Softmax layer outputs the probability of each of the 64 possibilities as: $p_k = \frac{e^{o_k}}{\sum_i e^{o_i}}$, o_k is the activation of the last fully connected layer in the network
- Loss is computed using: $L = -\sum_j y_j log p_j$
- Gradients for the last layer:

$$
\begin{aligned}
\frac{\partial L}{\partial o_i} &= -\sum_k y_k \frac{\partial log p_k}{\partial o_i} \\
&= -\sum_k y_k \frac{1}{p_k} \frac{\partial p_k}{\partial o_i} \\
&= -y_i(1 - p_i) - \sum_{k \neq i} y_k \frac{1}{p_k}(-p_k p_i) \\
&= p_i \left(\sum_k y_k \right) - y_i \\
&= p_i - y_i
\end{aligned}
$$

where y_i is the actual outcome for i^{th} label while p_i is the predicted probability.



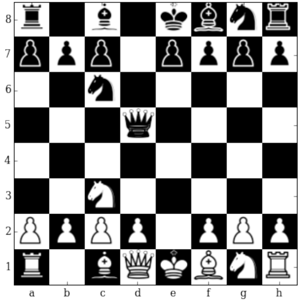
(a) Softmax-loss on training dataset (b) Softmax-loss on testing dataset; (c) Accuracies at k=1,3,5,10 on testing dataset.

Accuracies of the *piece* and *move|piece* predictors

Model Name	Accuracy at k			
	k=1	k=3	k=5	k=10
Piece	56.0	87.9	95.8	99.5
Pawn	94.8	100.0	100.0	100.0
Rook	58.0	85.2	93.5	99.6
Knight	77.3	96.6	99.3	100.0
Queen	54.2	81.3	90.4	98.2
King	71.7	95.6	99.6	100.0

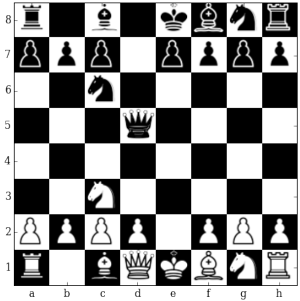
Test set performance without masking

- Correct prediction in the top-10 predictions almost everytime
- The dataset already has multiple labels for even a single board.
- We need to analyze more– Full move predictions, gameplay....



- Masking is done by zeroing out the probabilities of the illegal predictions.
- For the PIECE predictor, the opponent piece positions and the empty positions are zeroed out.
- For the MOVE|PIECE predictor, the illegal final positions are zeroed out for the respective piece.

Masking



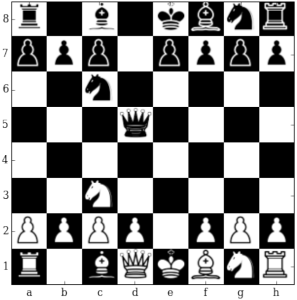
- Masking is done by zeroing out the probabilities of the illegal predictions.
- For the PIECE predictor, the opponent piece positions and the empty positions are zeroed out.
- For the MOVE|PIECE predictor, the illegal final positions are zeroed out for the respective piece.

0.0000000	0.0000001	0.0000000	0.0000042	0.0000001	0.0000000	0.0000000	0.0000000
0.0000000	0.0000004	0.0000000	0.0000037	0.0000000	0.0000016	0.0000010	0.0000002
0.0000001	0.0000000	0.0000000	0.0000001	0.0000001	0.0000003	0.0000006	0.0000005
0.0000002	0.0000000	0.0000000	0.0000000	0.0000000	0.0000001	0.0000000	0.0000001
0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
0.0000000	0.0000000	0.6692031	0.0000003	0.0000000	0.0000000	0.0000000	0.0000000
0.0017913	0.0001493	0.0000070	0.1242154	0.0000000	0.0097696	0.0081490	0.0008657
0.0001785	0.0000001	0.0000146	0.0031704	0.0000972	0.0206326	0.1617075	0.0000350

Unmasked Probability distribution

Masking

Introduction
Aspects of human chess playing
Related Work
Background
Computer Chess
Deep Learning
Convolutional Neural Networks
Dataset
Move Predictor
Description
Training
Performance
Case studies
Evaluation Function
Examples
Game Trajectories
Gameplay
Conclusions



- Masking is done by zeroing out the probabilities of the illegal predictions.
- For the PIECE predictor, the opponent piece positions and the empty positions are zeroed out.
- For the MOVE|PIECE predictor, the illegal final positions are zeroed out for the respective piece.

0.0000000	0.0000000	0.0000000	0.0000000	0.0000001	0.0000000	0.0000000	0.0000000
0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
0.0000000	0.0000000	0.6692101	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
0.0017914	0.0001493	0.0000070	0.1242167	0.0000000	0.0097697	0.0081490	0.0008658
0.0001785	0.0000000	0.0000146	0.0031704	0.0000972	0.0206328	0.1617092	0.0000350

Masked Probability distribution

Let p and q be the distributions before and after masking. We compute two distances between both.

$$D_{sq\ euc}(p, q) = ||p - q||^2$$

$$D_{chebyshev}(p, q) = \max_i |p_i - q_i|$$

	$\|\|p - q\|\|^2$	$\max_i \|p_i - q_i\|$	Illegal move	Avg Rank of actual move	
Model			%age	unmasked	masked
PIECE	3.09×10^{-8}	4.28×10^{-5}	0.0	2.06342060113	2.06341424668
PAWN	1.72×10^{-4}	5.82×10^{-4}	0.0045	1.08001395504	1.07996844947
ROOK	3.74×10^{-3}	1.37×10^{-2}	0.72	2.31513493742	2.28972185557
KNIGHT	1.74×10^{-5}	4.8×10^{-4}	0.0	1.44417866616	1.44410761652
BISHOP	3.94×10^{-3}	1.15×10^{-2}	0.47	1.83962121376	1.82638963959
QUEEN	5.49×10^{-3}	1.89×10^{-2}	1.23	2.52614094756	2.47512457486
KING	3.35×10^{-3}	3.82×10^{-3}	0.33	1.59097552371	1.5873805164

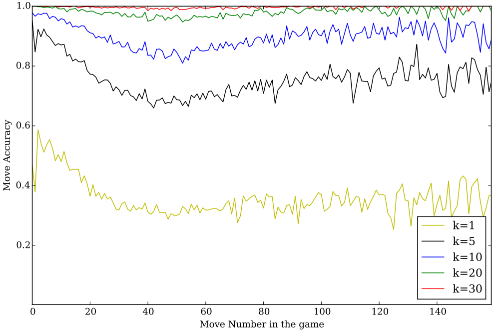
- Observe that there is not much difference between the probability distributions before and after masking.
- Piece selector model doesn't predict any illegal piece positions.
- There are no incorrect moves predicted for knights.

Performance after Masking

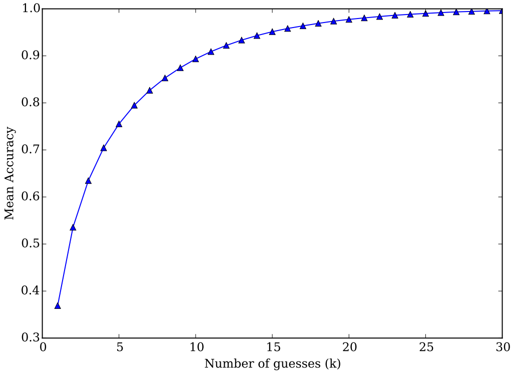
In the table, the accuracies of each of the models is compared before and after masking for the set of 314, 740 board positions.

Model	Percentage Accuracy	
	before masking	after masking
Piece	56.11	56.11
Pawn	53.60	53.60
Rook	50.98	51.26
Knight	72.77	72.77
Bishop	59.89	60.13
Queen	47.99	48.42
King	64.49	64.76

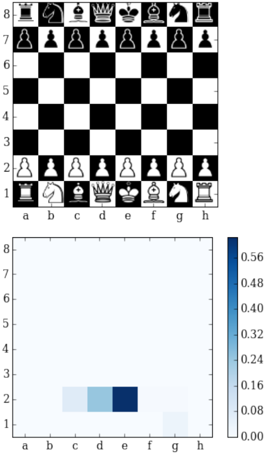
Accuracies before and after masking.



Average accuracies at different move numbers in test dataset games for top k predictions (k=1,5,10,30).



Plot of the variation of mean accuracy of the actual move lying in the top-k predictions with k (the number of guesses).



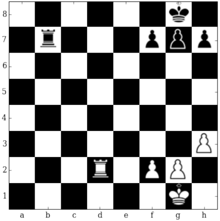
$P_{\mathrm{PIECE}}(B)$

Predictions for initial board position

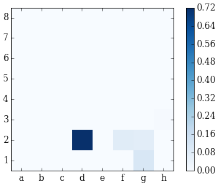
Move	P_{PIECE}	No. of times played	%age times played
All	1.0000000	4970725	100.00%
e2e4	0.6088475	2221439	44.7%
d2d4	0.2467637	1618291	32.6%
c2c4	0.0725896	286295	5.8%
g1f3	0.0334573	334741	6.7%
e2e3	0.0171652	62744	1.3%
f2f4	0.0059381	67248	1.4%
g2g3	0.0052637	93072	1.9%
b1c3	0.0021147	98306	2%
b2b3	0.0013837	64692	1.3%
c2c3	0.0013727	17449	0.4%
d2d3	0.0011852	43647	0.9%
g2g4	0.0008549	11245	0.2%
h2h3	0.0007792	5814	0.1%
b2b4	0.0007009	13761	0.3%
a2a3	0.0005240	5843	0.1%
g1h3	0.0002781	3875	0.1%
b1a3	0.0002716	1013	0%
h2h4	0.0002087	7216	0.1%
a2a4	0.0002071	4967	0.1%
f2f3	0.0000310	9067	0.2%

The percentage of times a certain opening was played according the the opening database on `http://www.ficsgames.org/openings.html`.

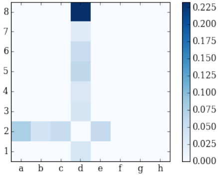
Case studies
Checkmate in 1 and Detecting a check



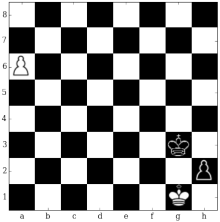
There is a check and mate possible in one move of the white.



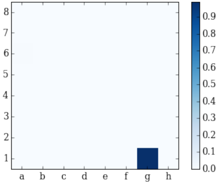
$P_{\text{PIECE}}(B)$



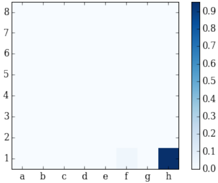
$P_{\text{ROOK}}(B, d2)$



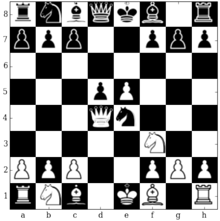
The white king is under check



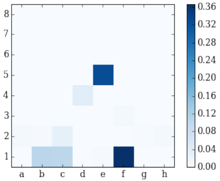
$P_{\text{PIECE}}(B)$



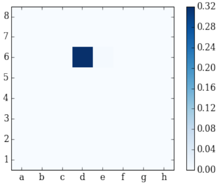
$P_{\text{KING}}(B, g1)$



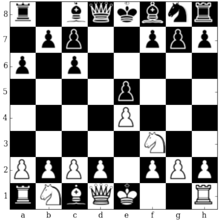
The d-pawn just moved 2 steps



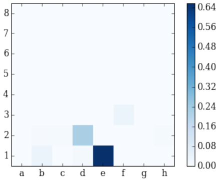
$P_{\text{PIECE}}(B)$



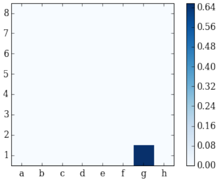
$P_{\text{PAWN}}(B, e5)$



Board position. Castling is one of the favorable moves available.



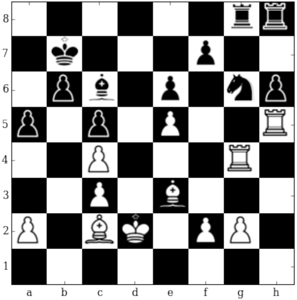
$P_{\text{PIECE}}(B)$



$P_{\text{KING}}(B, e1)$

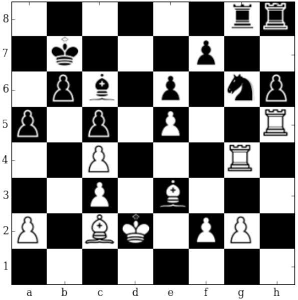
Case Study

Carlsen vs Anand, Game 6, 2014 World Championship



Black to move. 26th move.

Case Study
Carlsen vs Anand, Game 6, 2014 World Championship

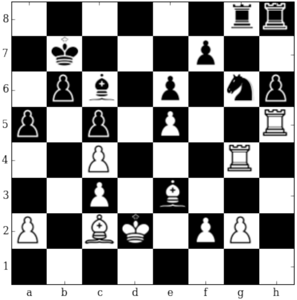


Black to move. 26th move.

Move	Predicted Probability
g8d8	0.252880722284
g8g7	0.148237019777
g6e5	0.13111974299
b7c7	0.0660261586308
h8h7	0.0471959412098
c6g2	0.0403394699097
c6e8	0.0358173549175
g8c8	0.0323895104229
b7c8	0.0302288047969
c6d7	0.0250529013574
a5a4	0.0231475103647
g6e7	0.0229441132396
b7a6	0.0208601523191
g8a8	0.0198916308582
b7b8	0.0190593209118
c6a4	0.0153007712215
g6f8	0.0150824002922
g8f8	0.0115283448249
g8e8	0.00727259740233
b7a7	0.00666899653152

Possible moves after the mistake by Carlsen in Move 26

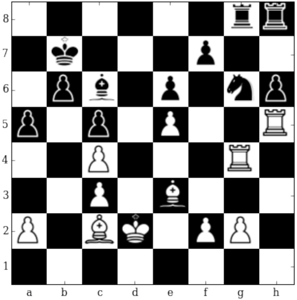
Case Study
Carlsen vs Anand, Game 6, 2014 World Championship



Black to move. 26th move.

Move	Predicted Probability	
g8d8	0.252880722284	
g8g7	0.148237019777	
g6e5	0.13111974299	Expected Move
b7c7	0.0660261586308	
h8h7	0.0471959412098	
c6g2	0.0403394699097	
c6e8	0.0358173549175	
g8c8	0.0323895104229	
b7c8	0.0302288047969	
c6d7	0.0250529013574	
a5a4	0.0231475103647	Anand's actual move
g6e7	0.0229441132396	
b7a6	0.0208601523191	
g8a8	0.0198916308582	
b7b8	0.0190593209118	
c6a4	0.0153007712215	
g6f8	0.0150824002922	
g8f8	0.0115283448249	
g8e8	0.00727259740233	
b7a7	0.00666899653152	

Possible moves after the mistake by Carlsen in Move 26



Black to move. 26th move.

Move	Predicted Probability	
g8d8	0.252880722284	
g8g7	0.148237019777	
g6e5	0.13111974299	Expected Move
b7c7	0.0660261586308	
h8h7	0.0471959412098	
c6g2	0.0403394699097	
c6e8	0.0358173549175	
g8c8	0.0323895104229	
b7c8	0.0302288047969	
c6d7	0.0250529013574	
a5a4	0.0231475103647	Anand's actual move
g6e7	0.0229441132396	
b7a6	0.0208601523191	
g8a8	0.0198916308582	
b7b8	0.0190593209118	
c6a4	0.0153007712215	
g6f8	0.0150824002922	
g8f8	0.0115283448249	
g8e8	0.00727259740233	
b7a7	0.00666899653152	

Possible moves after the mistake by Carlsen in Move 26

A tweet said– *"Wow 26...Ne5! and black is winning....Anand plays 26...a4? Whats going on :) #CarlsenAnand"*

Evaluation function

- Given a game in the dataset, consider the following assignment:

$$V(b_{t_{final}}) = \begin{cases} 1 \text{ , if White has won} \\ 0 \text{ , if it is a draw} \\ -1 \text{ , if Black has won} \end{cases}$$

- According to the recursive rule the discounted reward for a board at time t into the game is:

$$V(t) = \gamma V(t+1), \forall t < t_{final}$$

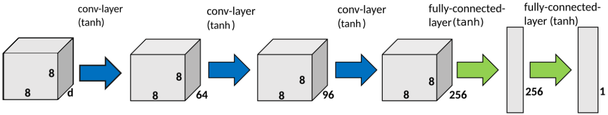
- Use the following rule moving up into the game:

$$V(b_{t_{final}-i}) = \begin{cases} \gamma^i \text{ , if white won eventually} \\ -\gamma^i \text{ , if black won eventually} \\ 0 \text{ , if the game was a draw} \end{cases}$$

$$V(b'_{t_{final}-i}) = \begin{cases} -\gamma^i \text{ , if white won eventually} \\ \gamma^i \text{ , if black won eventually} \\ 0 \text{ , if the game was a draw} \end{cases}$$

where $b_{t_{final}-i}$ is the board (as it appears to the white player) i steps away from the finish, while $b'_{t_{final}-i}$ is the rotated board with flipped colors i steps away from the finish.

- In this way, each board in the dataset is assigned an evaluation.

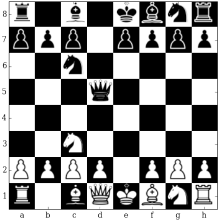


- Modeled as a regression problem with the evaluation value as the dependent variable and the board images as the regressors.
- Loss function:

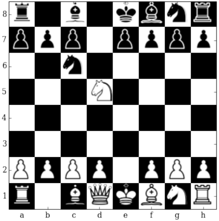
$$L = \frac{1}{2}(f(x) - y)^2$$

where x is the value at the last fully-connected layer in the network and f is the activation function (tanh in our case).
- Simply the gradient looks like:

$$\frac{\partial L}{\partial x} = (f(x) - y)\frac{\partial f(x)}{\partial x}$$

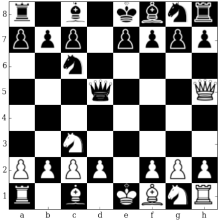


Board position (White to move)



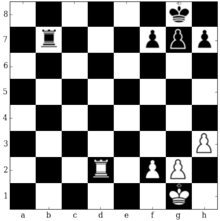
Move: c3d5

$V_{\gamma=0.7} = 0.0545$

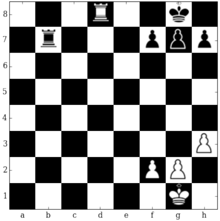


Move: d1h5

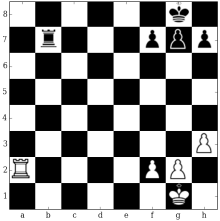
$V_{\gamma=0.7} = 0.0144$



Board position (White to move)

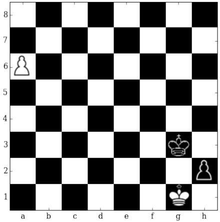


Move: d2a8
$V_{\gamma=0.7} = 0.0543$

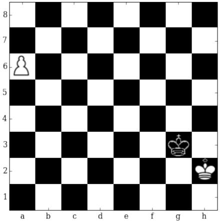


Move: d2a2
$V_{\gamma=0.7} = 0.0177$

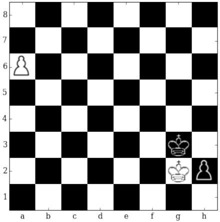
Evaluation function
Examples



Board position (White to move)



Move: g1h2
$V_{\gamma=0.7} = 0.0216$



Move: g1g2
$V_{\gamma=0.7} = 0.0084$

- We compare the learned evaluation function with the material heuristic values of the boards in our dataset.
  $V_{\text{MATERIAL}}(\mathit{board}) =$
  $1 \times (P - P') + 3 \times (N - N') + 3 \times (B - B') + 5 \times (R - R') + 9 \times (Q - Q')$
  where the values of P, N, B, R and Q come from:

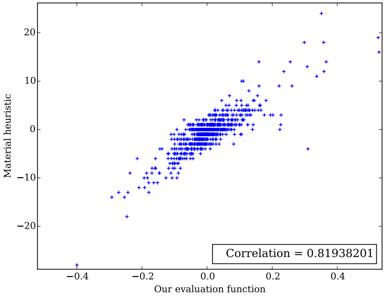
Piece	Pawn	Rook	Knight	Bishop	Queen
Value	1	5	3	3	9

- We compute the correlation of the two evaluation functions– $V_{\gamma=0.7}$ (learned by our model) and V_{MATERIAL}.

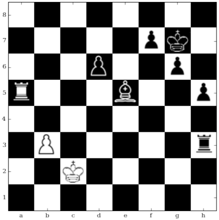
- We compare the learned evaluation function with the material heuristic values of the boards in our dataset.
  $V_{\text{MATERIAL}}(board) =$
  $1 \times (P - P') + 3 \times (N - N') + 3 \times (B - B') + 5 \times (R - R') + 9 \times (Q - Q')$
  where the values of P, N, B, R and Q come from:

Piece	Pawn	Rook	Knight	Bishop	Queen
Value	1	5	3	3	9

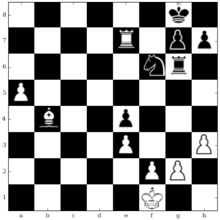
- We compute the correlation of the two evaluation functions– $V_{\gamma=0.7}$ (learned by our model) and V_{MATERIAL}.



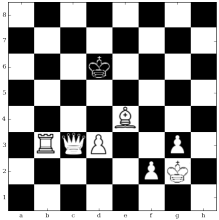
$V_{\gamma=0.7} = 0.0015796$
$V_{\text{MATERIAL}} = 0.0$



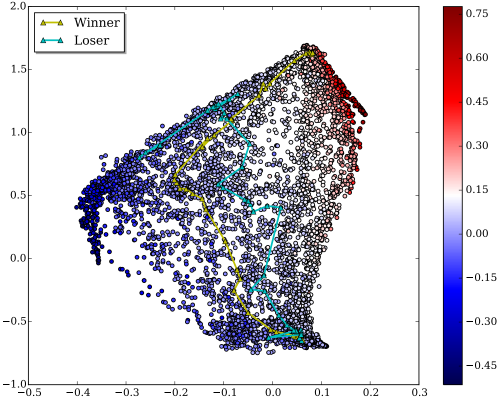
$V_{\gamma=0.7} = -0.0748723$
$V_{\text{MATERIAL}} = -6.0$



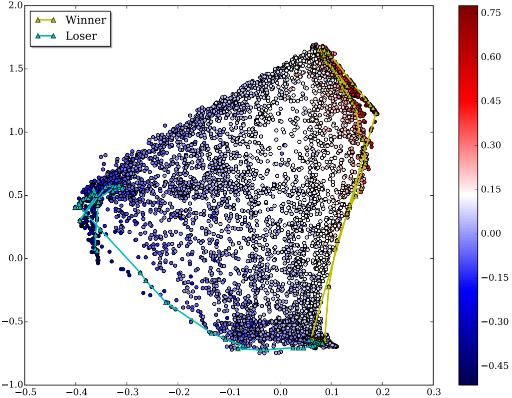
$V_{\gamma=0.7} = 0.0902274$
$V_{\text{MATERIAL}} = 2.0$



$V_{\gamma=0.7} = 0.302926$
$V_{\text{MATERIAL}} = 20.0$



t-SNE embedding of the activations at the last fully connected layer of the board evaluation CNN. The red end is the set of boards close to winning, the blue end is the set of boards close to losing a game. We plot a game on the embedding. The winner(yellow) ends on the red side of the embedding while the loser(cyan) ends on the blue side of the embedding



t-SNE embedding of the activations at the last fully connected layer of the board evaluation CNN. The red end is the set of boards close to winning, the blue end is the set of boards close to losing a game. We plot a game on the embedding. The winner(yellow) ends on the red side of the embedding while the loser(cyan) ends on the blue side of the embedding

Gameplay

- Use the predictions and evaluations made by the models discussed to play a game of chess
- Choosing a move:
  - Top Move Method: Choose the piece with the highest predicted probability, and then choose the position to move the piece to using the *move*|*piece* model

Gameplay

- Use the predictions and evaluations made by the models discussed to play a game of chess
- Choosing a move:
  - Top Move Method: Choose the piece with the highest predicted probability, and then choose the position to move the piece to using the $move|piece$ model

Algorithm 1 Top-move method

1: initial_pos = argmax($P_{piece}(board)$)
2: piece_type = getType(initial_pos)
3: final_pos = $argmax(P_{move,piece\_type}(board))$
4: **return** chess_coords(initial_pos)+chess_coords(final_pos)

- Use the predictions and evaluations made by the models discussed to play a game of chess
- Choosing a move:
  - Top Move Method: Choose the piece with the highest predicted probability, and then choose the position to move the piece to using the $move|piece$ model
  - Top Prob Method: Choose the move with the maximum joint probability–
    $P(move|board) = P(piece|board) \times P(final\_position|piece, board)$

- Use the predictions and evaluations made by the models discussed to play a game of chess
- Choosing a move:
  - Top Move Method: Choose the piece with the highest predicted probability, and then choose the position to move the piece to using the *move*|*piece* model
  - Top Prob Method: Choose the move with the maximum joint probability– $P(move|board) = P(piece|board) \times P(final\_position|piece, board)$

Algorithm 2 Top-prob method

piece_dist = $P_{piece}(board)$
cumulative_dist = *zeros*(64,64)
for $0 \leq i < 64$ **do**
    if $board[i/8, i\%8] \neq 0$ **then**
        piece_type=*getType*(i)
        move_distr = $P_{move,piece\_type}(board)$ * piece_distr[i]
        cumulative_distr[i] = move_distr
    end if
end for
initial_pos, final_pos = *argmax*(cumulative_distr)
return *chess_coords*(initial_pos)+*chess_coords* (final_pos)

Gameplay
Negamax

- A modified version of minimax algorithm
- It utilizes the same subroutine for the Min player and the Max player at each step, passing on the negated score following the rule:

$$max(a, b) = -min(-a, -b)$$

Algorithm 3 The basic Negamax algorithm for Chess

```
procedure NEGAMAX(()depth)
    if depth==0 then
        return evaluate()
    end if
    max = −∞
    generateMoves(...)
    while m = getNextMove() do
        makeMove(m)
        score = -Negamax(depth -1)
        unmakeMove(m)
        if score > max then
            max = score
        end if
    end while
    return max
end procedure
```

- Played against Sunfish (Ahle, 2015) implements MTD-f for search
- Pruned the search space to k=15 moves for every state
-

Method Used	Games Played	Won	Drawn	Lost	Details
Top-Prob	73	7	20	46	$10 \leq maxn \leq 1000$
TopProb-Negamax	19	3	4	12	$10 \leq maxn \leq 100$
Evaluation function ($V_{\gamma=0.7}$) with Negamax	21	6	4	11	2 <Negamax depth <5
Evaluation function ($V_{\gamma=0.7}$) with Negamax	25	16	2	7	Negamax depth=4

The table shows the result statistics for evaluation of gameplay against sunfish. In most of our experiments we limit the number of nodes explored by Sunfish between 10 to 1000 (chosen randomly on a log scale). For other deviations, the details are mentioned in the details column

Conclusion

- We need to evaluate the game play to get an ELO rating for our system.
- A possible improvement could be to use a bigram or a trigram input to train the networks. This will help the networks learn the long term tactics.
- A faster implementation could be provided by not relying on another chess playing engine to generate search trees.

Thank you

Questions?