DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING, SECOND CYCLE, 30 CREDITS

STOCKHOLM, SWEDEN 2017

Predicting expert moves in the game of Othello using fully convolutional neural networks

HLYNUR DAVÍÐ HLYNSSON

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF COMPUTER SCIENCE AND COMMUNICATION

Predicting expert moves in the game of Othello using fully convolutional neural networks

HLYNUR DAVÍÐ HLYNSSON

Master in Machine Learning
Date: July 8, 2017
Supervisor: Josephine Sullivan
Examiner: Danica Kragic
Swedish title: Förutsäga expertrörelser i Othello-spelet med fullständigt konvolutionella neuronala nätverk
School of Computer Science and Communication

Abstract

Careful feature engineering is an important factor of artificial intelligence for games. In this thesis I investigate the benefit of delegating the engineering efforts to the model rather than the features, using the board game Othello as a case study. Convolutional neural networks of varying depths are trained to play in a human-like manner by learning to predict actions from tournaments. My main result is that using a raw board state representation, a network can be trained to achieve 57.4% prediction accuracy on a test set, surpassing previous state-of-the-art in this task. The accuracy is increased to 58.3% by adding several common handcrafted features as input to the network but at the cost of more than half again as much the computation time.

Sammanfattning

Noggrann funktionsteknik är en viktig faktor för artificiell intelligens för spel. I denna avhandling undersöker jag fördelarna med att delegera teknikarbetet till modellen i stället för de funktioner, som använder brädspelet Othello som en fallstudie. Konvolutionella neurala nätverk av varierande djup är utbildade att spela på ett mänskligt sätt genom att lära sig att förutsäga handlingar från turneringar. Mitt främsta resultat är att ett nätverk kan utbildas för att uppnå $57,4\%$ prediktionsnoggrannhet på en testuppsättning, vilket överträffar tidigare toppmoderna i den här uppgiften. Noggrannheten ökar till 58.3% genom att lägga till flera vanliga handgjorda funktioner som inmatning till nätverket, till kostnaden för mer än hälften så mycket beräknatid.

Contents

Chapter 1

Introduction

In this thesis I apply recent advances in deep learning for the game of Othello. I use a class of functions called convolutional neural networks (ConvNets) to predict moves made by expert Othello players. I improve the highest reported result in this task by around 5.3%. Instead of building a whole game playing system I divert my effort on a particular module commonly used in board game programs.

The rest of this chapter presents a historical context of of the advances in Artificial Intelligence (AI) in board games. Taking note of some of the discoveries that led to recent advancement in other board games, this leads to the motivation for this thesis of creating a method that predicts expert moves in Othello with high accuracy. The rules of the game are described as well as its history and the advances for AI in Othello.

1.1 A short history of computers vs. humans in board games

Creating an agent that plays games at a similar or better level as humans is an important step towards a human-like AI as much of human interaction can be modeled as games [1]. In 1996, the general public took note as the world champion Garry Kasparov lost in a game of chess to the IBM computer Deep Blue [2]. The cognitive achievement of this feat has been downplayed later as most of the strength of Deep Blue lies in its "brute search" [3] rather than the machine recognizing complicated patterns. A year later, the reigning world champion in the game of Othello lost 6-0 in a bout against the AI Logistello written by Michael Buro [4].

Until 2016, the game of Go was the last bastion of human superiority to machines in board games as it has a much higher search space than chess [5]. This changed when the AI AlphaGo developed by Google Deepmind defeated the second highest ranking Go player in the world, Lee Sedol [6], an accomplishment considered to be 10 years away by AI experts [7]. This was done by having humans set up an architecture that approximates an electronic brain for the computer and having the computer analyze millions of games played by highly skilled Go players [8].

A little over a year later, in May 2017, AlphaGo defeated the highest ranked Go player in the world, Ke Jie, 3-0 [9]. At the time of writing Google DeepMind has not yet published a paper detailing the technical changes they have made to their player.

Instead of having an AI observe humans playing games and learn from our mistakes, advantage is usually taken of the computer's ability to simulate playthroughs of the game at a high speed. A program can then play itself or variants of itself and learn

from these mistakes in a framework called reinforcement learning. Such methods for many games have problems getting better if they are to start playing only taking random moves and playing against themselves. This might cause them to win games even if they make very bad moves, believing that those moves helped and reinforce their behavior of making those bad moves again. This is referred to as the credit assignment problem [10]. Initializing their AI to be an accurate expert move predictor before further learning from self-play showed great improvements, with small increases in the prediction accuracy leading to great improvements in player strength after further training [8].

Games also provide an interesting test bed for methods in AI. In a small environment with fixed rules it is good to gauge the strength of algorithms before transferring the algorithm to different problems. Skills that an AI learns in one domain can be transferred to other domains as well [11]. Similarly to how AlphaGo analyzed games played by humans, we can have agents in games or even the real life observe how humans behave and imitate them [12].

1.2 Research Question - Predicting good moves

The research question that this thesis aims to explore is: ***What are the benefits and costs of using a deep convolutional neural network for move prediction in the game of Othello, either using only exclusively implicitly learned features or explicitly handcrafted features in addition to implicitly learned feature, as measured by prediction accuracy and time needed for computation?***

1.3 Motivation - Why move prediction with ConvNets

Considering that state-of-the-art systems such as AlphaGo use ConvNets for the highest reported move prediction accuracy, I expect that the best move predictors for other similar games such as Othello will be based on ConvNets as well. The outcome of this work is indeed the highest reported prediction move accuracy for Othello so far.

Although machines can already outplay humans in Othello, it is still valuable to investigate the performance of different policy functions in the game. ConvNets haven't received much attention for this game so far so the application is novel. The board states are highly volatile as each new move can potentially change a large area of the board. Deep learning methods can be readily applied to this game since tournament data has been meticulously gathered over the past few decades.

In classical AI game playing strategies like minimax with alpha-beta pruning, we are reliant on domain knowledge to speed up the search [13]. Using deep convolutional neural networks, a policy can be trained to learn useful features of the game on its own without prior knowledge of the game being explicitly encoded [14]. Furthermore, understanding methods of automatic feature representation can improve general game playing in video games as well as board games [15].

Finding faster value functions can improve current minimax players as well and this is important for game apps in mobile devices as it causes the battery life to last longer [16].

The main strategy used by players such as AlphaGo, Monte Carlo Tree Search (MCTS), has been shown to benefit from being combined with a move evaluation function to guide the search towards more probable actions [17]. Combining ConvNets with MCTS, Go programs display a strong positive correlation between the playing strength of the

network and the networks accuracy in predicting expert moves. Having more accurate predictors can thus make stronger MCTS players [18] [8].

This was the key to have AlphaGo successfully predict its opponent's moves in the mid-game where move-databases are not of much help as it takes too long to search through them. By studying human play the AI was able to use this information to be ahead of the human player. This is the reason why move prediction is one of the most important elements of many game AIs

Having a human-like player is finally interesting in its own right as this AI could be used as a learning tool to help novice players see good alternatives to a move [19] or module in a real-world simulation to make the agents behavior more realistic or immersive.

1.4 Societal and ethical aspects

Automating board game players is a specific case of general automation facilitated by AI technologies. The work in this thesis is a part of a general trend of analyzing how humans perform certain tasks to match or even surpass their skills at those tasks. While creating programs that imitate human behavior we have to anticipate that it will at some point substitute other work for humans.

With a greater understanding of the problems people face every day and with greater tools for recognizing patterns, automation of tasks has become easier and easier. A great societal impact of AI will be the automation of jobs. Around an estimated 45% of workplace activities can already be automated, with an estimated 5% of jobs that can be fully automated using available technologies. [20]

As more and more jobs become automated, their absence will affect peoples' lives. Now obsolete professions such as elevator conductors have been described as creating a sense of community [21]. Instead of only aiming to optimize the explicit goal of a function, such as "win the game" for a video game AI, it can also be worthwhile to generate a semblance of humanness. An automated nurse in a nursing home might be the perfect medical caretaker for the elderly but might cause disappointment if it is too robotlike in some areas such as playing Othello with the residents.

Including a human-like Othello playing behaviour can thus decrease the sense of alienation while interacting with larger automated systems. On its own, however, there is little risk of automated Othello playing eliminating jobs as this is a game that's played for leisure and not profit in most cases. It will always be more exciting to watch non-artificial human behavior. It just makes it easier to practice it and to emulate the feeling so it encourages people to play the game instead of discouraging it.

1.5 Rules of Othello

The game is a two-player zero-sum, perfect information strategy board game where the players (black or white) have the goal of of having the most of their discs on the board at the end of the game [22]. The game has been modelled as a Markov decision problem in the literature [23].

The board is an 8×8 grid (figure 1.2) and players alternate placing a disc on an empty cell that's adjacent to an opponent's disc. The columns are denoted with the letters a-g and the rows with the numbers 1-8. When a disc is placed, all opponent discs that are

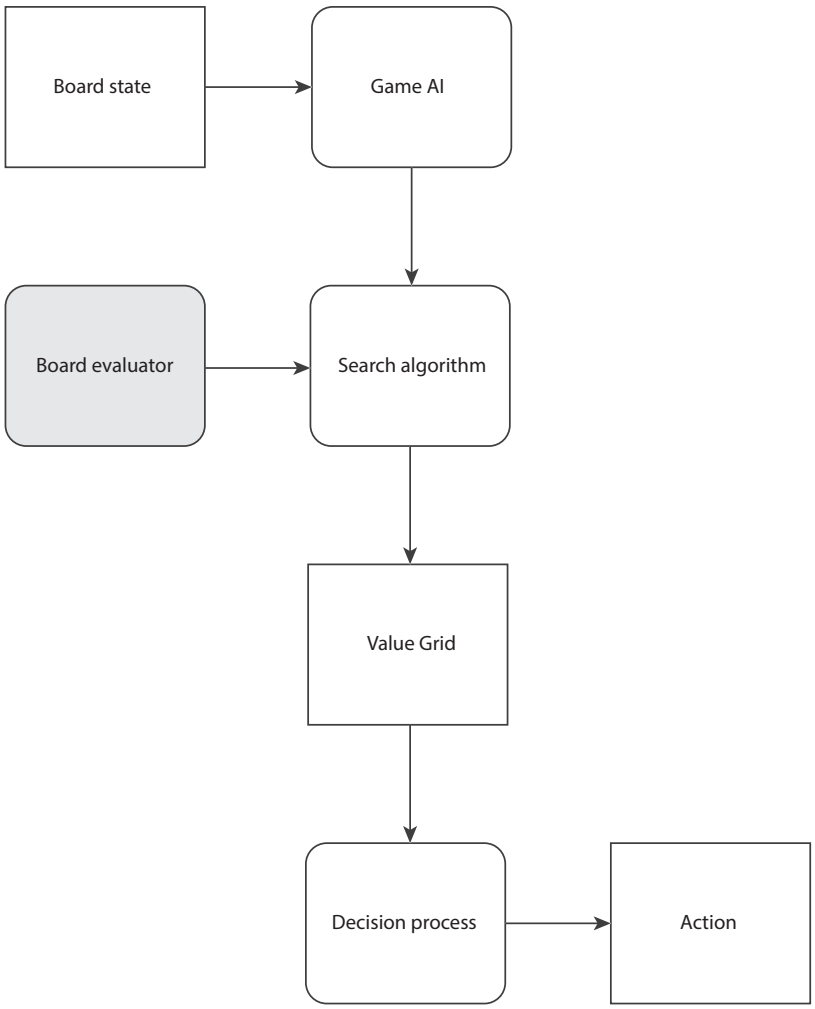
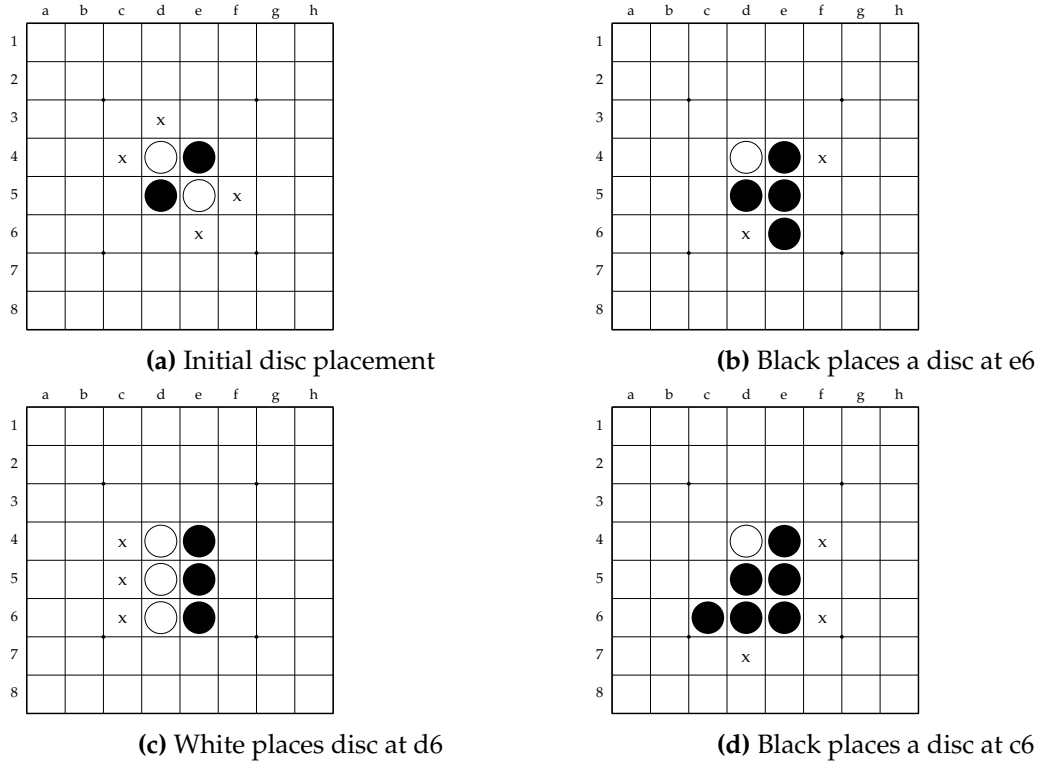


Figure 1.1: AI for board games at test time. A simplified version of the main components of an autonomous system to play board games. The AI accepts a representation of the current game state of the game being played. It uses this to search for the next move to take which leads to a winning conclusion. If there are too many states to consider to exhaustively consider every game trajectory, it may use a board evaluator or value function to approximate the likelihood of winning in the position. The work of this thesis focuses on the board evaluator. The search outputs a numeric evaluation of every possible action that can be taken. A decision is taken to act according to the result after the search, usually along the lines of choosing the move that maximizes the probability of winning, with an exception of exploring different lines of play to learn from them.

between the newly placed disc and at least one other disc owned by the acting player are flipped to the other color. Discs can also only be placed if they cause any of the opponent's discs to change color. If no cell satisfies this condition, the current player must pass their turn. The game ends when each cell contains a disc or if neither player can make a legal move.



(a) Initial disc placement

(b) Black places a disc at e6

(c) White places disc at d6

(d) Black places a disc at c6

Figure 1.2: Example Play In these first four states of a game we see how the board changes after the actions made by the players. **a.** The game begins with white discs on d4 and e5 and black discs on d5 and e4. Black is the first to move and can place discs at d3, d4, f4 or f6 as indicated by the crosses. **b.** Black places a disc at e6 and changes the white disc e5 to black as it is now being flanked by black discs. The possible counter-moves by white are indicated with the crosses again. **c.** White takes over the black disc on d5 by placing a disc on d6. **d.** Black takes over the discs on d5 and d6 by placing a disc on c6.

1.6 History of Othello

The game was invented in 1883 under the name Reversi by one of two Englishmen, Lewis Waterman or John W. Mollett, who both claimed ownership and called the other a fraud. The original version had the players place the first four pieces rather than starting as a default in the usual diagonal pattern. Both player also started with 32 discs and had to pass if they ran out.

The modern rules of the game were patented in 1971 by Goro Hasegawa under the name Othello [24]. The choice of the name alludes to a double dealing relationship between two parties in the play Othello by Shakespeare. In 1973 he established the *Japan Othello Game Association* and started selling the game internationally 1976 with a world championship the year after and every year since.

1.7 Computers playing Othello

The game has been solved for 4×4 and 6×6 boards and the second player is guaranteed to win under perfect play but the game remains unsolved for 8×8 boards and higher. [25]

The objective of the game is to end up with the most pieces but the total amount

each player has can change significantly between rounds. Heuristics like material count or counting pieces, which is useful for games such as chess, cannot be relied upon in Othello. This makes the design of an evaluation function non-trivial.

Many different and strong heuristics for the game have been developed however. For example a mobility metric which maximizes the number of moves available to the player and minimizing them for the opponent. This opens up more game-winning outcomes for the current player during a search and the opposite for the opponent. Counting the number of stable pieces, i.e. pieces that are guaranteed to not be flipped again during the game, is similar to material count in chess and has been utilized by early Othello programs [26].

Some heuristics exploit the fact that when a player places their piece in a corner, it is guaranteed to stay the same color as it cannot be flipped again for the remainder of the game. Similar knowledge about specific squares in the game is thus manually encoded, giving a positive value to good squares and a negative value to bad squares.

Another, more automatic, approach is to statistically evaluate patterns in the board. A pattern is any combination of squares on the board and a pattern configuration is any assignment of black, white or empty squares to the pattern. The value of the pattern assignment is then evaluated using statistical regression. This is the method used by Logistello when it defeated the world champion 6-0.

These heuristics have historically been combined using variants of minimax search such as alpha-beta pruning or negascout [27] [28]. See figure 1.1 for a graphic of a complete automatic Othello player. The search is made faster using for example move-ordering. Most strong Othello AIs today use a variant developed by Michael Buro called Multi Prob Cut [29] which has been shown to be highly effective in Othello and the game of Shogi [30].

The Prob Cut idea is also used for an endgame search. When there is a predetermined amount of empty discs left on the board, strong programs switch into a selective endgame search to find exact game outcomes. Specialized hardware has been prepared to make this search as efficient as possible [31].

As the branching factor early in the game is quite low, a strong method for AIs is to keep an opening book or database of common starting positions. This saves time as the program can instantly play along a known, strong line of play and avoid pitfalls [26].

Chapter 2

Related work

In this section I discuss the work that has been done on evaluation functions for Othello. The main types are pattern based heuristics and stability based heuristics. In this work the effect of explicitly giving ConvNets stability based heuristic inputs is examined. These are described to give a historical context for Othello in particular. A previous move prediction result for Othello is considered as well. Work done on artificial neural networks for Othello and the evolution of ConvNet design for Go is discussed.

In zero-sum games like Othello, where each game ends with one player winning and the other losing or a draw, a minimax search is a common method of minimizing the loss for the maximum loss scenario (fig. 1.1). Ideally, the search for each game state would result in the optimal sequence of moves needed to play a perfect game. This is not feasible for games with large search spaces however, so the search must be truncated. In this case the outcome of the game at the state where the search stopped under optimal play can be approximated with a value or heuristic function. [26].

Expert players can share their knowledge with programmers to fine tune such a function based on their intuition, or such a function can be learned in other ways automatically with a function approximator such as an artificial neural network.

Another search method, Monte Carlo Tree Search (MCTS), simulates potentially thousands of games and ranks the moves according to whether they are more likely to lead to game winning outcomes. In these simulations, the next move can be chosen randomly. Better results can be gained by guiding the search to begin with towards moves that are more probable to result in a win. In this work I focus on constructing a ConvNet that can be valuable in guiding the search of Othello players using MCTS. [8]

As no literature on ConvNet design was found for Othello, the design of the ConvNets in this thesis builds on the findings from the ConvNet development from Go. I describe the work done in the AlphaGo paper in detail in the last section. The inspiration for this thesis was the success of AlphaGo and the goal to investigate whether their methods can be useful for Othello as well.

2.1 Othello heuristics

The evaluation function that Michael Buro wrote for Logistello initially used a logistic regression to combine several features which were designed to maximize the number of moves, number of stable discs and finally the number of discs the player controls [22]. The function had different weights depending on which state the game is in

$$\text{game state} = \max\left(0, \lfloor \frac{\#\text{discs} - 13}{4} \rfloor\right) \tag{2.1}$$

i.e. a logistic regression is done for each of the 13 game states defined by this equation. [16]. When Logistello defeated the world champion the evaluation function used rather a sparse linear regression to fit around 1.2 million parameters for purely pattern-matching features, relying less on human game intuition [26].

An analysis of different Othello heuristics was performed by Sannidhanam et al. [32]. They the heuristics and examined what their contribution is to a program's playing strength. The heuristics were added individually and then an interplay between them to observe their correlations. They used mobility which counts the number of next moves a player has, stability which counts the number of discs which cannot change a color again. Additionally, they also investigated the value of total current amount of discs belonging to your color (they call this coin parity) and effect of capturing corners. In my work I use the stability heuristic as they found it to be the most valuable one, as well as the mobility heuristic but I did not include coin parity.

A thorough search of the literature for evaluation of Othello AIs on the basis of their ability to match human decisions along with their playing strength yielded a single result in the work of Runarsson et al [16]. Comparing a cornucopia of preference learning algorithms, they train and test the models on a set of a thousand games each from the French Othello Federation. In preference learning, a ranking of the possible moves is created. A method using N-Tuple features along with board inversion achieved the highest prediction accuracy of 53.0% and was their strongest Othello player overall. An N-Tuple feature is a random sample of N points from the board that can in the case of Othello have N^3 configurations.

2.2 Artificial neural networks for Othello

Binkley et al. investigated different artificial neural network (ANN) architectures for Othello using temporal difference learning. Some of their methods are similar to ConvNets as they embed weight sharing in the network but enforce symmetries and were found to outperform those who did not [33]. Another thing they found that it was advantageous to have multiple layers in the network rather than just one. It also gave worse results to encode the input from each square (black disc, white disc or empty) with a one dimensional vector rather than a three dimensional vector.

Chong et al. found that ANNs can outperform programs relying on simple handcrafted features such as player disc differences using co-evolutionary learning [34]. Using evolutionary strategies, Moriarty et al. trained ANNs to play against minimax players and found that their agent learned positional strategies and later a mobility based strategy (described above). [35] Makris et al. trained ANNs to play Othello at a master level using co-evolutionary learning. Networks that had 2 and 3 hidden layer and used weight sharing and exploited symmetries of the board gave the best performance. [36].

2.3 Modern ConvNet architectures

The most common building blocks of ConvNets are the convolutional layer along with ReLU, pooling, normalization and regular multi layer perceptron (fully connected) layers. See figure 2.1 for an example ConvNet architecture.

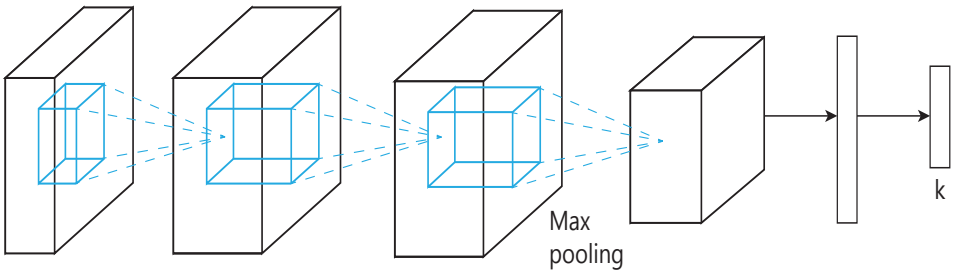


Figure 2.1: ConvNet. An abstract illustration of a ConvNet for image classification. The data passes from the input on the left to the output on the right. The black cubes represent the input to convolutional layers and the blue cubes represent the receptive field of the succeeding convolutional layer. An activation function such as ReLU is applied to the output of each box. A pooling layer, for instance max pooling, can be applied to the output of a convolutional layer, reducing the spatial dimension and total number of parameters. The last rectangle indicates that the image is classified into one of k classes while the preceding rectangle stands for a fully connected layer.

Max pooling (fig. 2.2) does a non-linear downsampling of the data. Commonly, they are 2×2 filters with a stride of 2, i.e. they keep only the largest value in non-intersecting 2×2 windows. This keeps the features with the strongest activations and preserves their relative topologies but discards 75% of less important data. In the example above, it's less important to know at what exact pixels the nose is if it's approximately in the middle of the face.

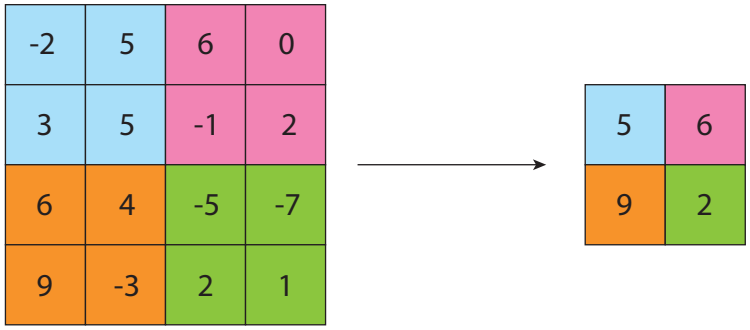


Figure 2.2: Max pooling. An example of the function of a 2×2 max pooling filter with a stride of 2. Such filters can bring about a great reduction of dimensionality and computation while preserving the most important information.

A common way to build ConvNets is to alternate convolutional and ReLU layers (explained in the Methods chapter) followed by pooling layers. The power of ConvNets grow with an increased number of layers up to a point where the net overfits. At this point they start memorizing the data they are trained on at the cost of generalizing well to unseen data.

Recent advancements have allowed more layers to be trained with the benefits of

higher accuracies. In addition to breakthrough algorithms, more data is becoming available which acts as a natural remedy to overfitting.

Dropout is a method popularized by Krizhevsky et al. [37]. During training, the output of each node is set to zero with some chance. This samples networks from the full arhcitecture during training and during test time an effective ensemble of these networks is run. They found that this reduced overfitting but doubled the training time.

Batch normalization layers address the problem of variable feature densities of the data being passed to each layer during training. Normalizing these inputs allows for faster training and more accurate ConvNets. The regularizing power of these layers have been found to eliminate the usefulness of dropout during training. [38]

Using batch normalization and a novel approach called skip layers, He et. al have trained powerful ConvNets that are hundreds of layers deep [39]. Using the notation from (3.2), a skip layer would instead output

$$\mathbf{s}^\ell = f\left(\mathbf{W}_\ell \mathbf{x}^{\ell-1} + \mathbf{b}_\ell\right) + \mathbf{x}^{\ell-1} \tag{2.2}$$

These methods along with a careful initialization of parameters allows them to reach a revolution of depth. They won the 2015 ImageNet classification challenge with a 152 layer network.

2.4 ConvNets for Go

To the best of my knowledge there hasn't been an analysis of deep ConvNets for Othello in the literature but this has been heavily researched for Go. The work of Sutskever et al. contributed an ensemble of shallow ConvNets using raw input as well as encoded expert knowledge that predicted expert moves in Go games with 36.9% accuracy [19]. Clark et al. achieved a 44.4% prediction accuracy using much deeper ConvNets and reported a performance boost by tying the weights of the convolutional filters to be invariant to reflections. They also report the first success of play using ConvNets in Go by consistently defeating the well known Go program GNU Go without using MCTS [40].

Maddison et al. trained a 12 layer convolutional neural network that predicted expert moves with 55.0% accuracy. They found that enforcing symmetries in filters helped shallow networks but not deeper networks. A notable contribution was their asynchronous MCTS which addressed the problem of using a computationally heavy ConvNet during the search [18].

2.5 AlphaGo

The research for AI in board games has culminated in the AlphaGo player developed by the Google DeepMind team. It is a creative combination of techniques in the literature that make the best Go player in the world. [8] I relate these to figure 1.1.

The search algorithm they use is called Monte Carlo Tree Search. It searches for the next move in a given game state by spawning a new independent game where it chooses the moves stochastically until the game has been played to completion. Instead of relying solely on value function, the search is guided as well by a function that gives states

that are more likely to be chosen by a human higher likelihood to be chosen. This is the board evaluator in the figure.

It's noted whether a win or loss was achieved in the end for each game state. Thousands of these simulations are performed per second. The method notes how often a state leads to a winning result, increasing the probability for those states to be chosen in future simulations. The states that were visited most often during the search are chosen for the action in the actual game. An array of these visitation values are the value grid in the figure and the decision process is to a greedy choice of highest visits.

To get the value function for the search, the AlphaGo team trains a ConvNet. To obtain the data for this they have similarly constructed convolutional neural policy networks play against itself, creating a synthetic database and learning from self-play. The learning is done with a method called policy gradient reinforcement learning and the networks were trained on 50 GPUs over a day.

To initialize the parameters for the reinforcement learning networks, they use supervised learning. The networks are trained on 30 million positions from the KGS Go server for three weeks. They opted out of enforcing weight symmetries as that hinders filters from identifying asymmetric features. They augment the data instead by adding all 8 reflections and rotations of a data point to the training.

The policy networks have 13 convolutional layers, each one with zero padding and a stride of one. The first layer has kernel size 5×5, layers two to twelve have kernel sizes 3×3 and the final layer has a kernel size of 1×1. Layers one to twelve apply a rectifier nonlinearity while the last layer applies a softmax nonlinearity. The input of the network is a $19 \times 19 \times 48$ tensor. As Go is played on a 19×19 board, these are 48 feature planes of the board state. They report a 57.0% prediction accuracy on a held out test set using a numerous expert features and 55.7% using raw board state as well as move history. This is the component of the AlphaGo work I aim to emulate for the game of Othello.

Chapter 3

Methods

Deep learning with ConvNets has achieved state-of-the-art for difficult tasks in many application domains. For tasks ranging from colorful image colorization [41] to generating convincingly realistic faces [42] to automatic game playing [43], ConvNets have indeed proven their strength in learning visual patterns.

In this thesis I want to investigate the efficacy of ConvNets for move prediction in Othello. To understand how ConvNets work I describe in this section the fundamentals of neural networks. This chapter contains a discussion of general machine learning classification and how it relates to move prediction in Othello. Gaining an understanding of the computational atoms of the full model constructed in the thesis is important.

The preparation of a neural network for a data-driven task and optimization are then considered. The methods in this thesis cannot work for a task without being guided or trained in some manner and I describe how this is done. The last section in this chapter describes the function of the convolutional layer used in this work which are the main building blocks of ConvNets. My work attempts to replicate the predictive success of the board evaluator module (fig. 1.1) of the AlphaGo system.

3.1 Classification

Generally, classification is correctly predicting a category for an input. If we have K potential categories of an input x, we can write this mathematically as a decision function D:

$$D : x \to \{0, \ldots, K\} \tag{3.1}$$

Examples are discerning whether a picture is of a dog or a cat or if a square in a board game is a good spot to place your next game piece or not. In machine learning, a function class such as a neural network is trained for such a task by being shown many images of things to recognize or classify. For example, a system can be trained on exclusively images of cats or dogs that are accompanied with the correct label "cat" or "dog", in the hope they can make correct guesses on unseen images without seeing the correct label. Move prediction for games played on spatial grids is similar to the task of semantic segmentation or pixelwise classification of images from computer vision.

The state of a board game can be processed as an image where each cell is a pixel and a classifier trained to output probabilities of a pixel belonging to one of two classes:

whether a move will be placed there or not. Normalizing the output mask of whether a move will be placed on each cell will give us a probability distribution of next moves which we can sample from during game playing, giving us a stochastic policy for the game. Making predictions for the next moves for the expert data is on the other hand deterministic, as we greedily choose the highest likelihood to be our guess.

3.2 Neural networks for classification

Artificial neural networks are a class of functions that have been shown to be powerful for prediction [44]. A simple type of a neural network is the single-layer perceptron which multiplies a weight w_i with each of the inputs x_i and adds a bias term b, optionally followed by a non-linear activation function f [45]:

$$f(\mathbf{w}^T\mathbf{x} + b) \tag{3.2}$$

The weights and inputs are stored in column vectors $\mathbf{w} = (w_1, \ldots, w_N)$ and $\mathbf{x} = (x_1, \ldots, x_N)$ respectively, where N is the dimension of the input.

Sigmoid functions such as $\tanh(x)$ or $(1 + e^{-x})^{-1}$ have historically been a popular choice for activation functions as differential non-linearities [46]. A default recommendation for deep learning purposes at time of writing [47] is using the rectified linear unit (relu) function $f(x) = \max(0, x)$ instead as they were shown to increase training speed by up to sixfold [37].

We can combine the outputs of several single-layer perceptrons (nodes) to build a multi-layer perceptron (MLP) [47] and write the output of layer $\ell = 1, \ldots, L-1$ as $\mathbf{x}^\ell$:

$$\mathbf{x}^\ell = f(\mathbf{W}_\ell \mathbf{x}^{\ell-1} + \mathbf{b}_\ell) \tag{3.3}$$

where $\mathbf{x}^0 = \mathbf{x}$, $\mathbf{W}_\ell = (\mathbf{w}_{\ell 1}^T, \ldots, \mathbf{w}_{\ell m_\ell}^T)^T$, $\mathbf{w}_{\ell j} = (w_{\ell j 1}, \ldots, w_{\ell j m_{\ell-1}}^T)^T$, $\mathbf{b}_\ell = (b_{\ell 1}, \ldots, b_{\ell m_\ell})^T$ and m_ℓ is the number of nodes in layer ℓ

The output of the final layer is **a**:

$$\mathbf{a} = g(\mathbf{W}_L \mathbf{x}^{L-1} + \mathbf{b}_L) \tag{3.4}$$

When $\mathbf{x}$ is to be put into one of $K > 2$ classes, a common choice is to pass the outputs of the final layer into a softmax activation function [48]

$$g(s_i^L) = \frac{\exp(s_i^L)}{\sum_{k=1}^K \exp(s_k^L)} \tag{3.5}$$

giving a probability vector for the classes which is useful for interpretability [49].

3.3 Optimization - training a neural network

Before we begin designing an MLP, a collection of input-label pairs $(\mathbf{x}, \mathbf{y})$ are gathered; our data. Let's write the label of $\mathbf{x}$ as a one-hot vector, i.e. if it's in class k then it's a vector of zeros except for 1 in the k-th position: $\mathbf{y} = (y_1, \ldots, y_k, \ldots, y_K) = (0, \ldots, 1, \ldots, 0)$.

The MLP outputs a corresponding vector of estimated probabilities $\mathbf{a} = (a_1, \ldots, a_K)$. The performance of the network is evaluated with a loss function $L(\mathbf{a}, \mathbf{y})$ which is to be minimized. With softmax classification the loss function is typically the cross-entropy loss [50]:

$$L(\mathbf{a}, \mathbf{y}) = -\sum_{k=1}^{K} y_k \log(a_k) \tag{3.6}$$

$$= -\log(\mathbf{y}^T \mathbf{a}) \tag{3.7}$$

Now we should explicitly write the loss L as functions of the weights and biases as well as inputs: $L(\mathbf{W}, \mathbf{b}, \mathbf{x}, \mathbf{y})$. We want to change the weights and biases such that the average value of the loss function on our data is at a minimum. In the backpropagation algorithm, we iterate through our data points and the gradients of the loss function are found with respect to the weights $\nabla_{\mathbf{W}} L$ and biases $\nabla_{\mathbf{b}} L$ using the chain rule [51].

These values can then be used in an optimization algorithm to find a local minima of the loss function. For example in a gradient descent algorithm, we update the weights along the direction with the most negative change $\mathbf{W} \leftarrow \mathbf{W} - \eta \nabla_{\mathbf{W}} L$ for a step size or learning rate η. We then calculate the gradients again and repeat the updates until the loss values are low enough [47].

Calculating these gradients for all of our data is time consuming so there are many extensions of this idea to speed up the convergence. For example in stochastic gradient descent (SGD) we do an update as above but with only a single data point or in mini-batch gradient descent we choose a subset of the data for efficient training on large data sets. [52].

A variant of SGD is the ADAM algorithm [53]. It applies the update step to each individual parameter θ with corresponding gradient element G_θ:

$$t \leftarrow t + 1 \tag{3.8}$$

$$\eta_t \leftarrow \eta \cdot \sqrt{1 - \beta_2^t}/(1 - \beta_1^t) \tag{3.9}$$

$$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot G_\theta \tag{3.10}$$

$$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot G_\theta^2 \tag{3.11}$$

$$\theta_t \leftarrow \theta_{t-1} - \eta_t \cdot m_t/(\sqrt{v_t} + \epsilon) \tag{3.12}$$

with suggested default hyper-parameter values $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 1e - 8$ and the initial values of the first moment m_0, second moment v_0 and times tep t are all 0.

3.4 ConvNets - sparse, translation invariant neural networks

A ConvNet is a neural network with one or more layers whose nodes perform together a discrete convolution on the input. I describe their defining feature, the convolutional layer. To simplify notation I restrict their definition to the types that I utilize in my work. Then I describe how they are used for image classification and semantic segmentation or equivalently move prediction.

3.4.1 Convolutional layer

We write our initial input **x** now as an $8 \times 8 \times 42$ tensor. Each layer has $3 \times 3 \times n_{\ell-1} \times 64$ weights **w** which we call filters and n_ℓ is the number of filters in the layer, except the last one, which has $1 \times 1 \times n_{\ell-1} \times 2$ weights. The outputs of the convolutional layers are $8 \times 8 \times n_\ell$ tensors. The output of convolutional layer ℓ can be formulated as

$$\mathbf{x}^\ell = f_\ell(\mathbf{s}^\ell) \tag{3.13}$$

where

$$s^\ell_{i,j,c} = \sum_{k=1}^{n_{\ell-1}} \sum_{r=i,i\pm1} \sum_{s=j,j\pm1} x^{\ell-1}_{r,s,l} w^\ell_{r,s,k,c} + b_c \tag{3.14}$$

for $\ell = 1 \dots L-1$ and

$$s^\ell_{i,j,c} = \sum_{k=1}^{n_{\ell-1}} x^{\ell-1}_{i,j,l} w^\ell_{i,j,k,c} + b_c \tag{3.15}$$

for $\ell = L$. We set f_ℓ to be the ReLU function for every layer except the last, where it is the softmax function as described above. In the summation, we set $x^{\ell-1}_{r,s,l} = 0$ for $r, s < 1$ and $r, s > 8$, preserving the spatial dimensions of the data. In other words, our filters have 3×3 or 1×1 receptive fields with a stride of one and a zero padding of one.

3.4.2 ConvNets for image classification

There are several attributes of ConvNets that help with image based classification tasks in particular. The spatial connectivity coupled with a deep architecture allows for the filters to respond to a small spatial patch of the input and then combine those in later layers. Convolutional layers such those described above are usually structured in such a way that the output of one layer is the input of the next and so on. This topology of deep ConvNets allows them to learn a hierarchy of features. For example in a simple face recognition ConvNet, the filters might learn basic features such as edges in the bottom layer. These edges are then arranged in various shapes such as noses, eyes and mouths in the higher layers. The top layer could finally decide if the pattern of "two eyes at the top, a nose in the middle between them with a mouth at the bottom" is matched.

Since the weights in a layer are shared, they allow for the networks to learn to respond to the same feature, wherever it may appear on the original input. For low level features, detecting the same types of edges is usually useful everywhere in the image. If we would like to predict if any given image is of a sport event, then it's likely that a ball could appear at any place in the image - increasing the odds that the image is indeed of a sports game. In other words, convolutional layers are translation invariant.

Image based classification networks often have a fully connected layer in their last layers instead of convolutional layers. These were added to combine the features calculated for the softmax output function for classification but at the cost of a high number of parameters [54]. They have been declining lately in popularity as adding more convolutional layers instead brings about a greater performance boost. [39]

3.4.3 ConvNets for semantic segmentation

Instead of predicting a class for a whole image, we can predict a class for each pixel in an image. The structure of ConvNets facilitates this because it expands the local context around each pixel. Considering the face example again, it's more likely that a pixel belongs to an eyeball if it is a part of a circular shape. This likelihood might then get increased when the context is expanded to include more features of the face.

This is equivalent to predicting the next move a player would make in a board game played on a grid. We have two classes here for each pixel or cell, whether the game piece is placed there or not. By adding more and more convolutional layers, the ConvNet eventually considers the context of the whole board in its calculations.

Networks that forego the fully connected layer in the end, so-called fully convolutional networks, have shown great success for semantic segmentation tasks [55]. I draw inspiration from the fully convolutional network described by Long et al. which achieved state-of-the-art segmentation on the PASCAL VOC dataset in 2015.

Chapter 4

Results

Here I describe what decisions went into the retrieval and processing on the data. My design choices of the ConvNets architecture and training are motivated. The experiments on those ConvNets are outlined as well as their results.

4.1 Software

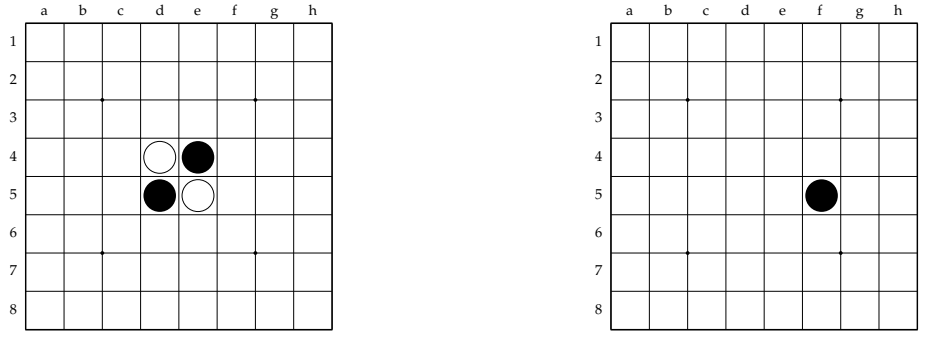
The ConvNets described below were written using the TensorFlow machine learning library for Python [56] as it provides a good framework for implementing ConvNets. Data processing and system experiments are written in Python 2.7 and Jupyter Notebook [57] was used during development to display fast visual outputs to various experiments throughout the working process. For the ConvNets that use explicitly handcrafted features, I made an API for the open-source Othello player Zebra to calculate the necessary features for the game state in a fast manner.

4.2 Dataset

The dataset was retrieved from the homepage of *Fédération Française d'Othello* (FFO) and consists of over 125 thousand Othello games from tournaments held between 1977 and 2016 [58]. I chose this as my dataset as it is very well organized and it is the largest database of Othello games available.

Each game consists of an ordered move list stored in a string, for example: "f5d6...a1", if the first d is placed in column F row 5, second disc in column D row 6 and so on until the last disc is placed in column A row 1, see figure 4.1. The prefix of the strings for games beginning as the example play in figure 1.2 would be "e6d6c6" and so on.

The strings are parsed to produce state - action pairs, i.e. what move was taken by a player for a given game state. The state is represented in its raw form as three 8×8 arrays or planes. This is a one-hot representation, so one plane contains a 1 in cells where there's a black disc but 0 in the other cells and similarly for white and empty spaces. Consider figure 4.1a: without calculating extra features, a raw one-hot representation of this state has three 8×8 planes/matrices. The first plane indicates position of white discs with 1 in d4 and e5 and 0 everywhere else, the second indicates black discs with 1 in d5 and e4 and 0 everywhere else and the third indicates which spaces are empty with 0 in d4, d5, e4 and e5 and 1 in all other non-occupied spaces. The associated action shown in



(a) State **(b)** Action

Figure 4.1: A sample state-action pair. A move history string "f5..." has been parsed to generate the raw initial state along with the first action taken. **a.** The initial state of the game, where white has discs on on d4 and e5 and black has discs on e4 and d5. **b** The action f5 associated with the state.

figure 4.1b will be parsed into a 8×8 plane that has 1 in f5 and 0 everywhere else. All actions similarly have 1 in one cell and 0 everywhere else.

Using this representation allows our network to learn non-linear functions of the input immediately in the first layer. Features of this type have been utilized for great success in Go [8].

Feature	# Planes	Description
Discs	3	Player, opponent discs or empty
History	4	Turns since move was played
Mobility	8	Legal moves for opponent after player disc is placed
Player stability gained	8	Stable discs gained by player after player disc is placed
Opponent stability gained	8	Stable discs gained by opponent after opponent disc is placed
Frontier	8	# Empty cells adjacent to cell
Legal moves	1	Cells where discs can be placed
Ones	1	Plane filled with constant ones
Player	1	Ones if current player is black

Table 4.1: Input features. The features into the ConvNets tested where input planes are 8×8 arrays of binary values. The features above the dashes line are the 3 features gotten from a one-hot representation of the raw board state. The dashed line splits the features into two groups of features for experiments: one set of architectures uses only 3 feature planes corresponding to raw disc placement but another set uses all 42 features.

4.3 Input features

The features are a mixture of the representation of the board state as well as a history of previous moves and derivative metrics from the board state. Each feature is one-hot encoded in 8x8 planes and is summarized in table 1. I chose this representation because it had been shown be the best encoding of the game state for Go as described in the re-

lated work. The features described were chosen because they can be passed as input to the ConvNets in a straightforward manner and have been shown to increase the playing strength of Othello programs [32].

The move history contains locations of the last 4 placed moves by either player. Features of this type are controversial for board games that can be modeled as Markov decision problems as the best move is game theoretically not dependent on the previous moves [40] although this is incorporated in the most successful Go player [8]. Incorporating this features can however increase the accuracy of a predictor [19]. This can be interpreted in such a way that players have a certain type of strategy in mind when they are executing play which follow some patterns.

One plane displays the cells where the current player can make a legal move. For each such cell the following values are calculated after the player's piece is placed there: opponent mobility, player stability gained and cell frontier. Mobility counts the number of legal actions the player has in a state. Stability is the number of stable discs a player has, i.e. the number of discs that cannot be flipped for the remainder of the game [59]. For each legal move by the player the opponent stability gained by his or her move in that cell is calculated as well. The frontier counts number of empty spaces adjacent to the position.

These features are estimated in other ways in modern Othello programs but are an easy way to give the ConvNet prior information about the game. The calculated values are represented by eight one-hot planes, representing the values $0, 1, \ldots, 6$ for the first seven planes and the eight representing plane representing values than 6. The final two input planes are a plane of ones which demarcate the legal "playing area" as each convolutional layer will pad its input with zeros, and a plane telling the ConvNet whose turn it is.

4.4 Architecture

For input with spatial dependencies such as our Othello data, ConvNets have been found to be the computationally and statistically most efficient types of neural networks [47].

The network outputs two planes with a softmax applied such that each cell can be interpreted as having a probability density of whether a disc is placed there in the next action. We take the plane with probabilities corresponding to whether a disc was placed there and zero out the cells where a disc cannot be placed legally. The cell with the highest value is chosen as our prediction. The final architecture used is summarized in table 4.2 or alternatively figure 4.2.

Nr.	Name	Width×Height	Filters	Filter size
0	Input	8×8	42	
1 - 10	Conv	8×8	64	3×3
	ReLU	8×8	64	
11	Conv	8×8	2	1×1
	Softmax	8×8	2	

Table 4.2: Convnet architecture A tabulation of the salient parameters of the best performing architecture, cf. figure 4.2

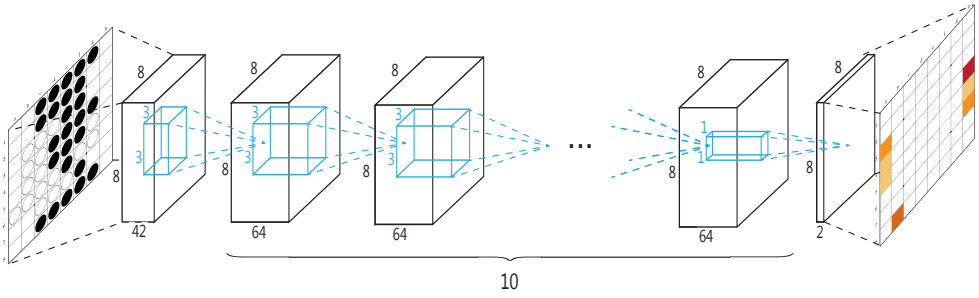


Figure 4.2: Convnet architecture. An illustration of the architecture that achieved the highest prediction accuracy. The board state is prepared outside the network and transformed into the 42 binary array or plane inputs. The network outputs a probability map of where the next disc would be placed by a human tournament attendee. The black boxes symbolize the dimensions of the convolutional layers. The spatial dimension 8×8 is kept constant throughout the network. Every convolutional layer has 64 filters and thus outputs the same amount of feature planes, except the last one which combines these features to two planes: whether a disc will be placed there or not. The blue boxes symbolize the area that each node processes in a convolutional layer. The filter dimensions are 3×3 for each layer except the last one where it is 1×1. The ReLU function is applied to the output of each conv layer except the last where the softmax function is applied.

We say that a 3×3 filter in the first layer has a receptive field 3×3 because that is the area it processes in the original image. If the layer above it has a 3×3 filter as well, it has an effective receptive field of 5×5 as it processes information from a 5×5 area of the image. A single 5×5 filter in the first layer would contain $5 \times 5 \times \#\text{input} \times \#\text{filters} + \#\text{filters}$ parameters while two 3×3 filters have $2 \times (3 \times 3 \times \#\text{input} \times \#\text{filters} + \#\text{filters})$ parameters. As stacking smaller filters results in the same receptive field and fewer parameters, experiments with filter sizes were not made even though this may prove useful.

As the prediction accuracy for expert move predictor ConvNets in Go rose, so did the number of filters per convolutional layer [19] [18] [8]. The number of filters chosen for AlphaGo were 192 so I chose a third of this at my ansatz of filters per player. Quick tests where networks were trained briefly indicated that performance on the validation set was worse for lower amounts of filters (32 or 48) or higher amounts (80 or 96) so I chose not to optimize this parameter further.

As the network architecture is entirely composed of convolutional layers we can say that it is fully convolutional and that it's computing a non-linear filter. I decided against having a fully connected layer in the end because architectures without them have been shown to be state-of-the-art for the similar task of semantic segmentation [55].

I also decided against having pooling layers which would decrease the spatial dimensions, e.g. making the output of the first convolutional layer $4 \times 4 \times 64$ instead of $8 \times 8 \times 64$. Max pooling layers are commonly used for pixelwise segmentation and output the maximum value of their input. This is usually done in networks that have inputs of a much higher spatial dimension such as 128×128 to decrease overfitting and reduce the number of parameters in the network. Although the spatial dimension can be recovered with an upsampling or convolutional transpose, our spatial dimensions are already too small to risk losing information.

An intuitive explanation of the effectiveness of max pooling is that the maximum values of features are more important than their exact location as long as their topology is

preserved. Considering the low spatial dimension of Othello and need for pixel-accuracy of the predictor, they were omitted for now.

Monitoring of the training process never indicated that the networks were in danger of overfitting (generalizing poorly to unseen data) so regularization techniques such as dropout or L_1 regularization were not added to the network. It's been observed for ConvNet design for Go that overfitting is not a concern and that dropout unnecessarily lengthens the training time [40]

4.5 Training

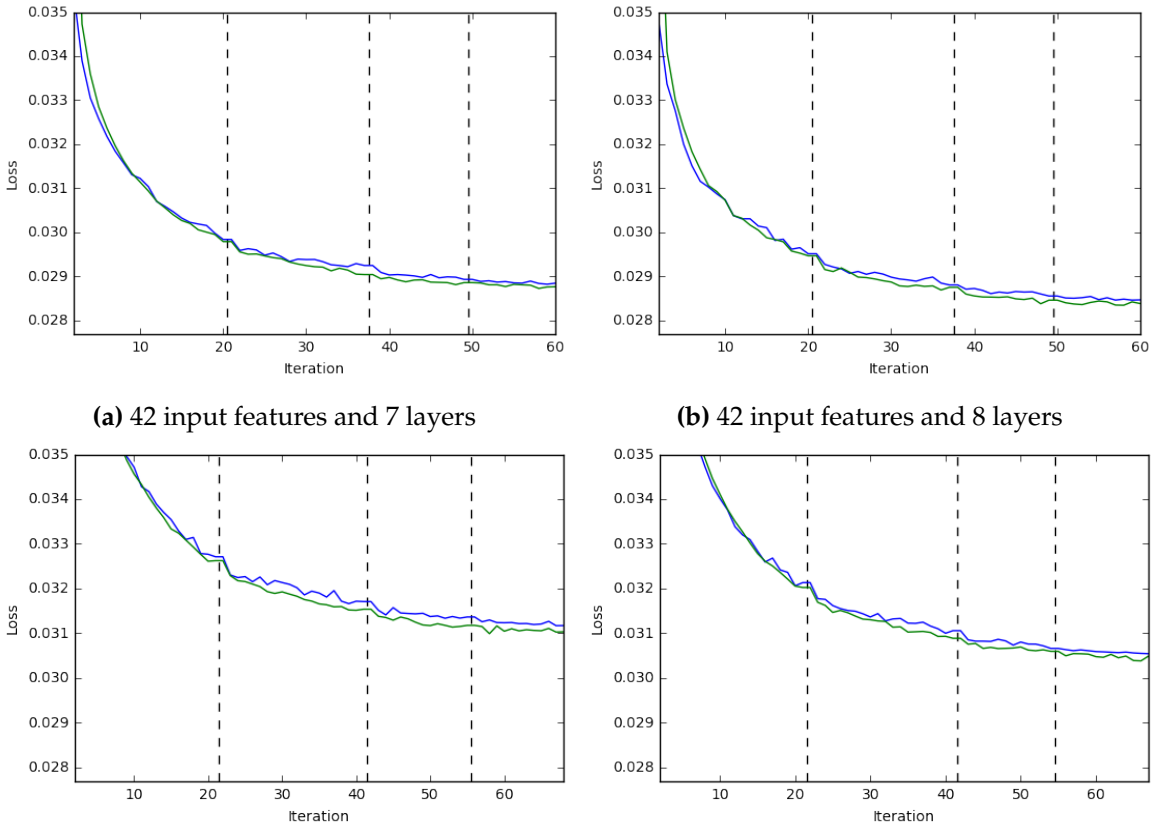
The networks were trained using the ADAM optimizer which is good for a large number of parameters [53]. All biases are initialized as constants 0.1 and all weights are drawn from a zero-mean truncated normal distribution with 0.1 standard deviation. This initialization was chosen to prevent the problem of "dead neurons", where some nodes might output 0 all the time and stop learning. The initial learning rate was $\eta =$ 1e-4 which was decreased step wise every 10-20 iterations (figure 3) to $\eta =$ 5-e5, $\eta =$ 2.5-e5 and finally $\eta =$ 1-e1. I chose the initial learning rates as it was the highest ones I found that steadily decreased the loss on the validation set. I decreased the learning rate for every model when the monitored loss values seemed to become less smooth. The batch sizes were 256 state-action pairs because this was big enough for the training to fast without taking too much space on shared GPUs.

Tournament games from 1977 to 2014 were used as the training set over which to minimize the softmax cross-entropy loss. Matches from 2015 were held out as the validation to help guide the parameter searching and newest matches from 2016 were held out as the test set to estimate final predictive power.

The average loss on the data in every training batch, before it is used in the learning update, is kept for every iteration. As it would take an approximate 11 hours to collect a loss value over the whole data set, this method of training loss estimation was preferred. These values for each iteration are then averaged again and plotted along with the average loss on the whole validation set. See figure 4.3 for the curves corresponding to the shallowest and deepest fully trained networks. The proximity of these curves indicate that the models do not overfit to the training set and should generalize well to unseen data.

The training data is ordered and each game was processed sequentially and in each iteration a state-action pair was added to the training with a 5% chance. This was done as a time-efficient method to decrease the correlation of the data in each mini-batch and the bias of the gradients to speed up convergence.

Symmetries of the game were exploited by adding transformations of each state-action pair using the dihedral group of four flips, one over each diagonal, to the batch. I chose not to add other operations such as a $90°$ rotation or a color inversion as those produce illegal game states immediately in the initial disc placement. No weight tying was enforced in the filter design to make them symmetric and allowing learning of potentially useful asymmetric filters. The most salient design choices are summarized in table 4.3



(a) 42 input features and 7 layers

(b) 42 input features and 8 layers

(c) 3 input features and 7 layers

(d) 3 input features and 8 layers

Figure 4.3: Loss plots The blue lines are the average loss values on the validation set and the green lines are the average loss values on the samples used in each training batch before the updates are applied. The dashed vertical lines indicate a change in learning rate. The learning rates are η_0 = 1e-4, η_1 = 5e-5, η_2 = 2.5e-5 and η_3 = 1e-5.

4.6 Evaluation

As the depth of ConvNets have been found to be the most important parameter for move prediction [18], I decided to test several varying depths. Relatively short trainings on a subset of the data indicated that 9 or 10 layers gave the best performance on the validation set so networks of depth 8, 9, 10 and 11 were tested.

Eight networks were trained simultaneously for a week on three Tesla 40K GPUs. All networks are the same as the one in table 4.2 except for the number of 3×3 convolutional layers and the input features. Predictions are made by choosing the maximum value of the output plane that corresponds to likelihood of a piece being placed there. The prediction accuracy on the validation set as a function of iteration can be seen in figure 4.4.

Table 4.4 gives an overview of the main results. The program is considered to have made a correct guess for symmetric board states if it produces any of the symmetric results. The 11 layer network which took handcrafted features as well as the raw board state as input achieved the highest overall accuracy, 58.2% on the validation set and 58.3% on the test set. This is an improvement by 5.3% over the highest reported rate of 53.0% from the literature [16] which compared multiple algorithms trained with preference learning on this task.

Design element	Choice	Motivation
Model	ConvNet	Performs well on data with grid-like topology
Number of layers	8-11	Result of parameter optimization
Hidden layer filter size	3×3	Large receptive field for relatively few parameters
Hidden layer filters	64	Result of parameter optimization
Hidden layer activation	ReLU	Faster training speed
Output filter size	1×1	Reduce dimensionality
Output filters	2	Output two-class probabilities
Output activation	Softmax	Interpretable probabilities
Regularization methods	None	Abundance of data and no apparent overfitting
Pooling layers	None	Simplicity of design
Initial learning rate	1e-4	Result of parameter optimization
Learning rate annealment	Step decay	Interpretable and easily controlled
Batch size	256	Highest batch size that would fit under constraints on GPU memory
Optimizer	ADAM	Good for a large number of parameters
Data augmentation	Diagonal flips	Assists learning symmetries in data

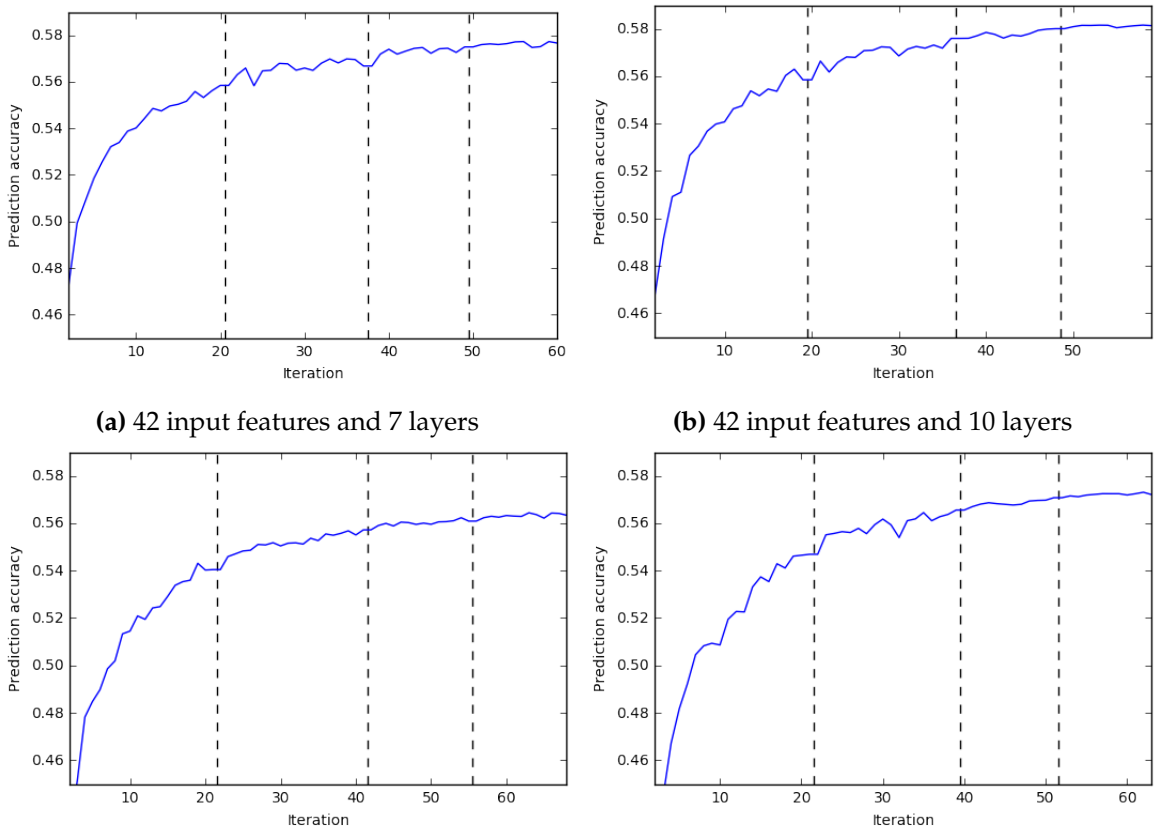
Table 4.3: Design choices. A table summarizing the most salient design choices in the architecture and training. Overall, the design choices were guided by AlphaGo. The parameters such as number of layers, number of filters and initial learning rate were initialized to be similar values that were used by successful predictor ConvNets for Go. A grid search of these parameters was conducted with training on a portion of the training data and validation data. The search ceased when only very small improvements were gained in tweaking these parameters.

The performance of the raw-input only networks seem to plateau at 10 layers, giving the same accuracy to a single decimal digit precision.

The forward passes of the ConvNets were calculated in TensorFlow. Calculations of the features were done by using custom API calls to Zebra which is to my knowledge the fastest open-source Othello program available. Even taking advantage of this speed, the 8 layer network that uses only the raw input is twice as fast as the one that takes all the features as input. Adding a layer with 64 filters of size 3×3 adds around 3 ms to the forward pass. The reward of utilizing these hand crafted features is an increase in accuracy of around 1%.

The networks have a relatively low amount of parameters. The best ConvNet only has around 360 thousand parameters, which is around a third of the amount of parameters used by Logistello. Fully connected layers commonly seen at the end of ConvNets are often responsible for the largest portion of the amount of parameters [37] so having a fully convolutional neural network can allow the network to be lighter.

Consider table 4.5 containing accuracies of each ConvNet per game state as defined by equation (2.1). The prediction accuracies display a strong negative correlation with the average branching factor (number of legal moves in a board state) of the respective game state. An interesting case is the third game state where every ConvNet displays a local



(a) 42 input features and 7 layers

(b) 42 input features and 10 layers

(c) 3 input features and 7 layers

(d) 3 input features and 10 layers

Figure 4.4: Validation accuracy plots. The average prediction accuracy on the 2015 matches was calculated every iteration. All networks were trained simultaneously and these values were monitored during the training. The learning rate was manually annealed for every network when these values showed erratic behaviour for any one of them. The dashed vertical lines indicate a decrease in learning rate similarly to figure 4.3.

maxima of prediction strength. Within the groups of inputs used, only all input features or using only the raw input, the values are all similar for a given game state but they tend to go higher for increasing depths.

Comparing the deepest ConvNets between these two groups, we find no difference in the first game state and raw input only are actually preferred in the second and third game state. This can be explained by players playing known opening games which the ConvNet can memorize more easily without the extra features, especially if they aren't a factor behind early play decisions.

Two of the handcrafted features in particular, the stability metrics, rarely come into play early in the game but are valuable in the middle and late game. No disc on the board is stable until some disc has entered a corner which rarely happens so early in tournaments. Using all the input features makes the predictions decisively better in the later stages of the game, making the difference as high as 3.7% in the last game state.

Marc Tastet of FFO has compiled a set of interesting positions, called the FFO Endgame Test Suite. I ran the ConvNet from figure 4.2 on the board state in fig. 4.5a. This position arose in the world championship finals between Graham Brightwell and Makoto Suekuni. The normalized output of the network is overlayed as probabilities. Perfect

Architecture		Evaluation			
Layers	Input	Validation Accuracy %	Test Accuracy %	Forward Pass (ms)	Parameters
8	42	57.8	57.8	34.5	246k
9	42	58.3	58.1	37.2	283k
10	42	57.8	58.0	40.1	320k
11	42	58.2	58.3	43.3	357k
8	3	57.0	56.8	17.6	223k
9	3	57.0	57.3	21.0	260k
10	3	57.2	57.4	24.7	297k
11	3	57.2	57.4	27.6	334k

Table 4.4: Performance of the fully trained networks. Results of the experiments conducted on different ConvNet depths as well as feature sets demarcated in table 1. Different amounts of identical convolutional layers were tested as well but otherwise having the same architecture as described in table 1. The forward pass times include preparing the features outside the networks. The times were measured on an Intel(R) Core(TM) i7-4500U CPU @ 1.80GHz.

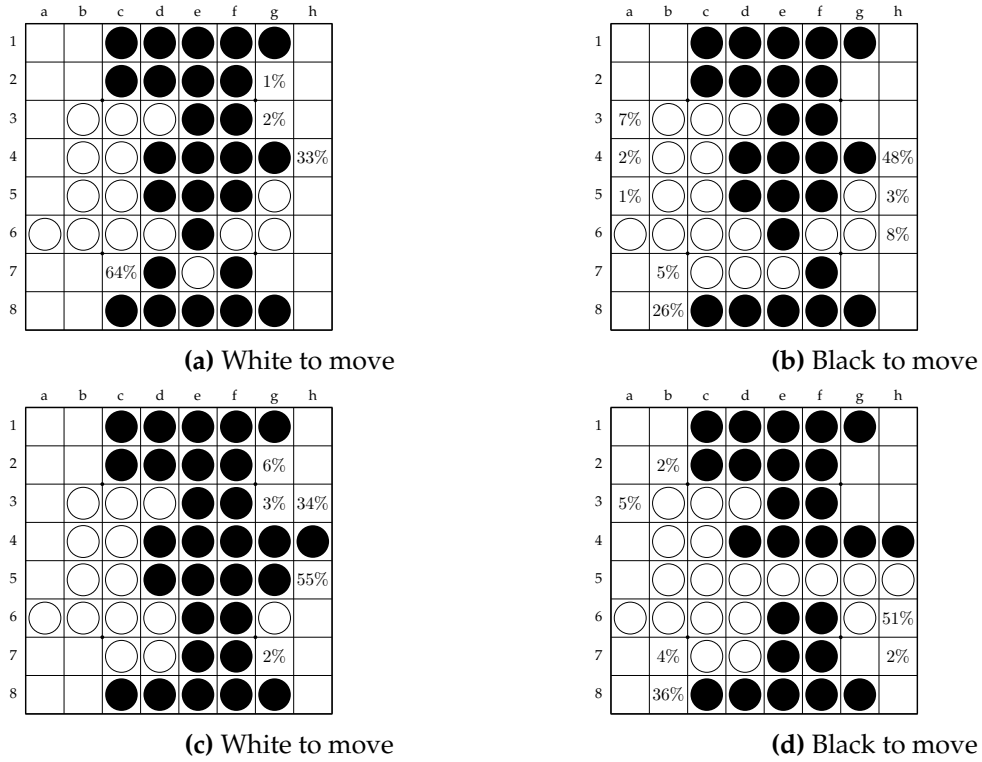
	Inputs: 42				Inputs: 3				
Moves	Layers: 8	9	10	11	Layers: 8	9	10	11	BF
1 - 12	74.6	74.9	74.5	74.9	74.8	75.0	75.2	74.9	7.1
13 - 16	40.7	40.6	40.5	41.2	41.4	41.9	41.2	42.1	11.0
17 - 20	49.3	50.0	50.0	50.0	50.1	50.2	50.7	50.9	11.5
21 - 24	46.8	47.4	46.5	47.4	47.7	48.7	48.1	47.2	11.9
25 - 28	45.7	46.6	46.1	46.1	45.4	46.5	46.4	46.9	11.7
29 - 32	46.9	47.3	47.2	48.1	46.4	46.3	47.6	47.4	11.3
33 - 36	49.0	48.6	48.5	48.9	47.2	48.1	48.2	48.1	10.6
37 - 40	49.1	50.0	49.5	49.8	48.1	49.6	49.0	49.3	9.6
41 - 44	52.7	52.3	52.8	53.3	50.7	50.8	51.2	51.5	8.4
45 - 48	55.1	54.6	55.3	55.4	53.1	53.2	53.2	53.2	7.1
49 - 52	59.1	59.8	59.9	60.0	56.1	56.7	56.6	57.0	5.5
53 - 56	65.5	65.7	65.6	65.6	61.5	62.2	62.3	62.1	4.0
57 - 60	83.3	83.4	83.7	83.5	79.1	79.6	79.8	79.8	2.5

Table 4.5: Accuracy per game state. The expanded accuracies of the convnets on the test set. Each row corresponds to a game state determined by equation 2.1. The greyed column highlights the highest overall accuracy. The last column displays the average branching factor of the rows game state.

play from both sides has been calculated by Zebra to produce the move sequence:

c7 h4 h5 b8 b7 a8 a7 a5 h3 a4 g3 h6 a3 h2 g7 a2

I ran the ConvNet on this scenario until it gave a non-optimal output. Using no lookahead search, the first three of these moves are correctly predicted and executed (figures 4.5b, c and d). The fourth move does not align with perfect play as it chooses h6 instead of b8.



(a) White to move **(b)** Black to move

(c) White to move **(d)** Black to move

Figure 4.5: States First four moves of FFO endgame test position #43, Brightwell vs. Suekuni in the World Championship finals of 1997. The optimal moves were found to be c7, h4, h5 and b8. The maximum value of the 11 layer convnet that uses all input features match these optimal moves for the first three states. In the fourth state it chooses rather h6 with a margin of 15%.

Chapter 5

Conclusion and discussion

Imitating the play of humans in tournaments is a step towards making convincingly human-like Othello players. In this work I showed that convolutional neural networks can be designed to perform well as expert move predictors for the game of Othello, exceeding the previous state-of-the-art for this task by around 5.3%. The best performing network was the deepest one presented, using handcrafted features as well as the raw board state as input. Only using the raw input only decreased the accuracy by 0.9% but sped up the calculations significantly. A complete player that incorporates such an evaluator in search can benefit greatly from the evaluator being much faster like this. Many variants of the architecture presented in this thesis are possible and the work here can be extended with novel AI techniques for more powerful Othello programs.

Similarly to the results that are found by different preference learning algorithms by Runarsson et al. [16], the accuracies per game state are almost a decreasing function of the branching factor of the game state.

Intuitively, it would seem that 4 layers would be good as it is the minimum amount for the receptive field to be at least the full scope of the board but combining features beyond this at deeper levels increased predictive power.

Only using the raw input, ConvNets are able to learn the game at a similar level as if they would get the handcrafted features as well. The speed gained by omitting calculating these custom features might be valuable if this technique is to be combined with a search for a complete Othello player.

Although I decided not to use pooling layers, it's conceivable they'll be incorporated in successful Othello ConvNets later. Narrowing down the types of building blocks in an experimental ConvNet also helps identifying the minimum necessary set of ingredients to reach a certain baseline [60]. The results in this work can potentially be improved by scaling up the networks or doing a more complete hyperparameter search.

A lot can be done to extend the work of this thesis in the future. Beyond experimenting with pooling layers, scaling the network up could be helpful. Redesigning the network as a ResNet will probably bring about a higher prediction accuracy but potentially at the cost of a longer running time, as ResNets architectures allow networks to be extremely deep [39].

Creating an ensemble of the networks could improve the prediction accuracy further but at a cost of greatly increasing the forward pass times. The analysis indicates that using the ConvNets that only use the raw board state as input during the first 28 guesses to make predictions and then switching a ConvNet that uses the full input features for

the rest would increase the prediction strength.

As the current top Othello programs have evaluation functions that change per game state, it would be interesting to fit different ConvNets per each game state. This allows the computation of each network to be solely for features that are beneficial for their respective states of the game. It might be the most fruitful for future research to concentrate efforts on finding good ConvNets for the middle game, as top Othello programs rely on an opening book and exhaustive search for the early and late games.

The ConvNets could receive as input a new feature plane describing what status of the game it is, which could effectively affect the weights that the features get as a function of the game state.

I tested either all the handcrafted features at once or none at all. With more time, a leave-one-out analysis could help determine which of the input features has the most effect on the prediction accuracy along with a description of the cost of adding this feature.

A more thorough analysis of the output of the ConvNets can be done. For example examining the outputs of the net for different board states that have easily interpretable good or bad moves. This could include placing a piece in a corner with no good alternative, or opting out of placing a piece in a corner to secure more stable pieces elsewhere or something similar.

As convolutions are translation-invariant, ConvNets taking larger dimensions can be trained on the same data but zero padding the input with the purpose of prediction or game play on larger Othello boards. Performance can be evaluated by pitting the resulting ConvNet against a program such as Daisy Reversi, which supports play for up to 24×24 boards.

Going beyond move prediction, the other aspects of the AlphaGo training pipeline can be tested for Othello. This includes training the ConvNets using policy gradient (PG) reinforcement learning after they have been trained on the expert data. A value network can then be trained by using self-play data from the PG ConvNets. The value and policy networks can then be combined in asynchronous MCTS algorithms.

I performed initial experiments with PG methods. The PG algorithm used was tested on small cases which indicated that the method was implemented correctly. However, when used on a fully supervised trained network they did not improve the playing strength. It did not help either to try from a network that was trained for half the supervised training time or after none at all. Considering the success of AlphaGo with this method, a more careful implementation should yield success for Othello as well.

With the decreasing costs and increasing speeds of parallel computation units, we might see such MCTS outperform variants of minimax players for Othello in the near future.

References

[1] Carl Shapiro. The theory of business strategy. *The Rand journal of economics*, 20(1): 125–137, 1989.

[2] Murray Campbell, A Joseph Hoane, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.

[3] Charles Krauthammer. Be afraid. `http://www.weeklystandard.com/be-afraid/article/9802`. *The Weekly Standard*. Retrieved: 5. May. 2017.

[4] Michael Buro. Takeshi murakami vs logistello. In *ICGA*, volume 20, pages 189–193, 1997.

[5] Bruno Bouzy and Tristan Cazenave. Computer go: an ai oriented survey. *Artificial Intelligence*, 132(1):39–103, 2001.

[6] Steven Borowiec. Alphago seals 4-1 victory over go grandmaster lee sedol. `https://www.theguardian.com/technology/2016/mar/15/googles-alphago-seals-4-1-victory-over-grandmaster-lee-sedol`, *The Guardian*, Retrieved: 5. May. 2017.

[7] William Hoffman. Elon musk says google deepmind's go victory is a 10-year jump for a.i. `https://www.inverse.com/article/12620-elon-musk-says-google-deepmind-s-go-victory-is-a-10-year-jump-for-a-i`. *Inverse*. Retrieved: 5. May. 2017.

[8] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[9] Sam Byford. Alphago retires from competitive go after defeating world number one 3-0. `https://www.theverge.com/2017/5/27/15704088/alphago-ke-jie-game-3-result-retires-future`, *The Verge*, Retrieved: 5. June. 2017.

[10] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[11] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 806–813, 2014.

[12] Gary. Eastwood. How video game ai is changing the world. http://www.cio.com/article/3160106/artificial-intelligence/how-video-game-ai-is-changing-the-world.html *CIO*, Retrieved: 5. May. 2017.

[13] Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai. In *AIIDE*, 2008.

[14] Yoshua Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529–533, 2015.

[16] Thomas Philip Runarsson and Simon M Lucas. Preference learning for move prediction and evaluation function approximation in othello. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(3):300–313, 2014.

[17] Rémi Coulom. Computing elo ratings of move patterns in the game of go. In *Computer games workshop*, 2007.

[18] Chris J Maddison, Aja Huang, Ilya Sutskever, and David Silver. Move evaluation in go using deep convolutional neural networks. *arXiv preprint arXiv:1412.6564*, 2014.

[19] Ilya Sutskever and Vinod Nair. Mimicking go experts with convolutional neural networks. *Artificial Neural Networks-ICANN 2008*, pages 101–110, 2008.

[20] James Manyika Michael Chui and Mehdi Miremadi. Four fundamentals of workplace automation. http://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/four-fundamentals-of-workplace-automation, *McKinsay Company*, Retrieved: 15. June. 2017.

[21] Michael M. Grynbau. The subway's elevator operators, a reassuring amenity of another era. http://www.nytimes.com/2011/04/29/nyregion/subway-elevator-operators-dwindle-in-new-york.html, *The New York Times*, Retrieved: 13. June. 2017.

[22] Michael Buro. An evaluation function for othello based on statistics. *Technical Report 31, NEC Research Institute*, 1997.

[23] Nees Jan van Eck and Michiel van Wezel. Reinforcement learning and its application to othello.

[24] Peter Michaelsen. More seriously, a brief history of othello. https://www.yumpu.com/en/document/view/33725044/marriage-of-the-eel-british-othello-federation/4 *Othello Museum*, Retrieved: 7. May. 2017.

[25] Joel Feinstein. Perfect play in 6x6 othello from two alternative starting positions, 1993.

[26] Michael Buro. The evolution of strong othello programs. In *Entertainment Computing*, pages 81–88. Springer, 2003.

[27] Aske Plaat. Mtd (f), a minimax algorithm faster than negascout. *arXiv preprint arXiv:1404.1511*, 2014.

[28] Alexander Reinefeld. *Spielbaum-Suchverfahren*, volume 200. Springer-Verlag, 2013.

[29] Michael Buro. Experiments with multi-probcut and a new high-quality evaluation function for othello. *Games in AI Research*, pages 77–96, 1997.

[30] K Shibahara, N Inui, and Y Kotani. Effect of probcut in shogi—by changing parameters according to position category. In *Proceedings of the 7th Game Programming Workshop*, 2002.

[31] CK Wong, KK Lo, and Philip Heng Wai Leong. An fpga-based othello endgame solver. In *Field-Programmable Technology, 2004. Proceedings. 2004 IEEE International Conference on*, pages 81–88. IEEE, 2004.

[32] Vaishnavi Sannidhanam and Muthukaruppan Annamalai. An analysis of heuristics in othello, 2015.

[33] Kevin J Binkley, Ken Seehart, and Masafumi Hagiwara. A study of artificial neural network architectures for othello evaluation functions. *Information and Media Technologies*, 2(4):1129–1139, 2007.

[34] Siang Y Chong, Mei K Tan, and Jonathon David White. Observing the evolution of neural networks learning to play the game of othello. *IEEE Transactions on Evolutionary Computation*, 9(3):240–251, 2005.

[35] David E Moriarty and Risto Miikkulainen. Discovering complex othello strategies through evolutionary neural networks. *Connection Science*, 7(3-1):195–210, 1995.

[36] Vassilis Makris and Dimitris Kalles. Evolving multi-layer neural networks for othello. In *Proceedings of the 9th Hellenic Conference on Artificial Intelligence*, page 26. ACM, 2016.

[37] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[38] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[39] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[40] Christopher Clark and Amos Storkey. Teaching deep convolutional neural networks to play go. *arXiv preprint arXiv:1412.3409*, 2014.

[41] Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *European Conference on Computer Vision*, pages 649–666. Springer, 2016.

[42] David Berthelot, Tom Schumm, and Luke Metz. Began: Boundary equilibrium generative adversarial networks. *arXiv preprint arXiv:1703.10717*, 2017.

[43] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

[45] Raúl Rojas. *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013.

[46] Jun Han and Claudio Moraga. The influence of the sigmoid function parameters on the speed of backpropagation learning. *From Natural to Artificial Neural Computation*, pages 195–201, 1995.

[47] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[48] Christopher M Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

[49] John S Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*, pages 227–236. Springer, 1990.

[50] Rob A Dunne and Norm A Campbell. On the pairing of the softmax activation and cross-entropy penalty functions and the derivation of the softmax activation function. In *Proc. 8th Aust. Conf. on the Neural Networks, Melbourne, 181*, volume 185, 1997.

[51] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[52] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[53] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[54] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[55] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[56] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané,

Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

[57] Fernando Pérez and Brian E. Granger. IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29, May 2007. ISSN 1521-9615. doi: 10.1109/MCSE.2007.53. URL `http://ipython.org`.

[58] Fédération Française d'Othello. La base wthor. `http://www.ffothello.org/informatique/la-base-wthor`. Retrieved: 24 Jan. 2017.

[59] Paul S Rosenbloom. A world-championship-level othello program. *Artificial Intelligence*, 19(3):279–320, 1982.

[60] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.