# Reykjavík University Project Report, Thesis, and Dissertation Template

by

Sigurður Helgason

Thesis of 60 ECTS credits submitted to the School of Computer Science
at Reykjavík University in partial fulfillment
of the requirements for the degree of
**Master of Science (M.Sc.) in Computer Science**

December 2019

Examining Committee:

Yngvi Björnsson, Supervisor
Professor, Reykjavík University, Iceland

Tough E. Questions, Examiner
Associate Professor, Massachusetts Institute of Technology, USA

# Reykjavík University Project Report, Thesis, and Dissertation Template

Sigurður Helgason

December 2019

**Abstract**

this is my abstract written in the language of English I am testing thought

alas I knew

ok so now I'm trying something new

asdf

test

# Herkænsku mat á djúptauganetum

Sigurður Helgason

desember 2019

**Útdráttur**

this is my abstract written in the language of íslensku þæööasdf

# Important!!! Read the Instructions!!!

If you have not already done so, LaTeX the `instructions.tex` to learn how to setup your document and use some of the features. You can see a (somewhat recent) rendered PDF of the instructions included in this folder at `instructions-publish.pdf`. There is also more information on working with LaTeX at `http://samvinna.ru.is/project/htgaru/how-to-get-around-projects-publish.pdf`. This includes common problems and fixes.

This page will disappear in anything other than draft mode.

*I dedicate this thesis to my family who while not understanding most of what I do always support me. The friends that still want to talk to me even though my schedule has rendered me unmeetupwithable. Lastly but certainly not least,*
*Don't Panic.*

# Acknowledgements

So long, and thanks for all the fish.

Acknowledgements are optional; comment this chapter out if they are absent Note that it is important to acknowledge any funding that helped in the workThis work was funded by 2020 RANNIS grant "Survey of man-eating Minke whales" 1415550. Additional equipment was generously donated by the Icelandic Tourism Board.

# Preface

This dissertation is original work by the author, Sigurður Helgason.

The preface is an optional element explaining a little who performed what work. See `https://www.grad.ubc.ca/sites/default/files/materials/thesis_sample_prefaces.pdf` for suggestions.

List of publications as part of the preface is optional unless elements of the work have already been published. It should be a comprehensive list of all publications in which material in the thesis has appeared, preferably with references to sections as appropriate. This is also a good place to state contribution of student and contribution of others to the work represented in the thesis.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| MSc | Masters of Science |
| ML | Machine Learning |
| AI | Artificial Intelligence |
| ANN | Artificial Neural Network |
| DNN | Deep Neural Network |
| MCTS | Monte-Carlo Tree Search |
| CAV | Concept Activation Vector |

This part of the dissertation introduces the concepts and the field.

# Chapter 1

# Introduction

State the objectives of the exercise. Ask yourself: Why did I design/create the item? What did I aim to achieve? What is the problem I am trying to solve? How is my solution interesting or novel?

The world of deep learning (DL) is exciting, we have models that are able to examine images and very reliably be able to identify it's content. Deep learning models have beaten Opthalmologists in identifing diabetic retinopathy, they've identified cancer cells where others have not. Deep learning models have even predicted the likelyhood a person will die in the following year with good accuracy (CITE). These models have done these things extremely accurately, cheap, and fast.

DL in conjunction with a randomized State-space Search Algorithm Monte-Carlo Tree Search, was used to defeat the standing world champion Lee Sedol (CITE ALPHAGO ZERO) in the incredibly expansive game of Go. This was done in the year 2017, the researchers used reinforcement learning (RL), where the agent learned by only playing against itself. These result imply that given enough time to learn the computer agent is able to gain a deep insight on how the game should be played.

The field of examining deep learning models to gain a richer understanding of how they work, and what makes them so much better than humans at various tasks is still new. This is the field of Explainable Artificial Intelligence (XAI). If we wish to continue using artificial intelligence (AI) and machine learning (ML), to improve our lives we must examine how they work, this is not only to improve the models themselves but could also augment our ability in many cases. Furthermore, these explanations are a legal requirement now in many areas, namely, legal, and medical. Recently, XAI has generally been focused on Model Interpretability, in particular focus on gaining insights on the concepts learned by Deep Neural Networks.

In this we thesis takes a look at how we can examine an artificial neural network (ANN) to understand which higher level concepts (HLC) it deems important for a given state within games. These games can be simple like Tic-Tac-Toe or Breakthrough and extremely complicated like Chess or Go.

The method of examining HLC's within games has generally been done by examining the current state of the game by evaluating them with respect to some heuristic. A heuristic within games are evaluations of the end reward for a state given only the state, for example, the amount of pieces left within a game of chess. Intuitively, the amount of pieces left is a good estimate for a state in chess, this is a higher level concept we use to evaluate a chess position. The piece amount can be considered as a lower-level concept than other concepts we use. For instance, grand master chess players evaluate a position w.r.t. states where the king is safe from attack, or the structure of the pawn positions. This paper examines a neural networks evaluation of a state regarding those higher-level concepts, if human players evaluate the king safety of a state low but the neural network highly values it, and the neural network plays better than the player. There could be an avenue for us to improve our game by considering king safety more thoroughly.

We define the research questions

1. Given a NN that plays Breakthrough very well, can we evaluate if that NN recognizes the HLCs that human players use.

2. Does a NN that trains itself using self-play learn these HLCs over time, and does it start to emphasize the HLCs that are generally taught later to human players, more as it trains itself more.

3. Does a NN that plays better than some humans focus on HLCs that are not generally considered by those human players and it.

### 1.0.1 Summary of the thesis

In this thesis we take an example game of Breakthrough, train a neural network to play the game using only self-play. That is, the neural network is trained only by playing against itself with no outside input other than the rules of the game itself. And we examine the neural networks against popular Higher-level concepts (heuristics) that we use to play the game.

# Chapter 2

# Background

In this chapter we discuss traditional artificial intelligence in the context of search the algorithms we used, what some other similar methods exist as well as the field in general.

## 2.1 Environments

When doing research on Artificial Intelligence it is important to select an environment that is a suitable abstraction for the task at hand. Environments vary significantly, and can be identified by their characteristic. The characteristics that are generally used to describe environments can be seen in Table 2.1. CITE AIMABOOK

| **characteristic** | **Values** | **Description** |
|---|---|---|
| Observable | Fully, Partially | How much of the environment can your agent percieve. |
| Agents | Single, Multi | Are there multiple agents playing in the environment. |
| Deterministic | Deterministic, Stochastic | Do the actions your agent do deterministicly impact the environment |
| Episodic | Sequential, Episodic | Are actions episodic or sequential |
| Static | Static, Semi-Static, Dynamic | ehh |
| Discrete | Discrete, Continuous | Is your environment discrete w.r.t actions (does it end) |

Table 2.1: Characteristics of environments

Categorizing environments like this gives you the power to find an environment in which a method works and know it can be applied to different environments with the same characteristics. Talk about **agents** in the context of environments as the entities that act within the environment. A game player is an agent as well as the neural network model that exists within a car that is able to drive in the real world.

## 2.2 Game Environment

With classical Artificial Intelligence Game Environments are commonly used to validate a method, game environments can be games like Tic-Tac-Toe, Breakthrough, and driving simulators. Game environments are a suitable place to apply AI as they serve an important function as an abstraction of the real world, for instance, a self-driving car agent who is verified to avoid driving into walls in a simulation is possibly safer than one who is not.

## 2.3 Breakthrough

The game breakthrough is a simplified version of chess, the game is set up on an $MxN$ board with cells like in chess, and each player starts with two rows of pawns at either end. The objective of the game is to move one of the pawn pieces to the opposite end of the board. A player wins if either they have reached the opposite end of the board, or has captured all of his opponents pawns. The pawns differ from chess pawns in such a way that they can not move two squares on the first move and, they can move diagonally as well as forward, this leads to the game being impossible to draw as pieces are always able to move. An example of an initial board in breakthrough can be seen in Figure 2.1

Categorizing Breakthrough with the characteristics described in Section 2.1. We end up with the description shown in Table 2.2. These characteristics are identical to that of Tic-Tac-Toe, and Chess. This categorization is the most common in board games where two player compete.

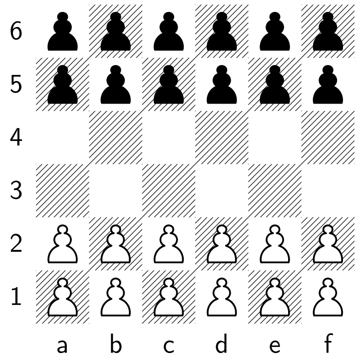


Figure 2.1: Example breakthrough board

| **Characteristic** | **Value** |
|---|---|
| Observable | Fully |
| Agents | Multi |
| Deterministic | Deterministic |
| Episodic | Sequential |
| Static | Static |
| Discrete | Discrete |

Table 2.2: Categorization of Breakthrough

### 2.3.1 Heuristics of Breakthrough

To evaluate the game of Breakthrough we can consider many heuristics (higher-level concepts) for instance a very simple heuristic would be the amount of pieces each player has left. This heuristics gives us some insight into how well the game is progressing, but obviously there are cases where this doesn't tell us much, as in cases when your opponent has a single piece left that is on the row immediately before the row needed for him to win. No matter how many pieces you have left, this state is bad for you if you're not able to capture that piece. An example of such a state can be seen in Figure 2.2.

A different heuristic would be the distance of your most advanced pawn minus your opponents most advanced pawn, this heuristic could give you insight into how close you are to winning the game or how close your opponent is. As a higher level concept we can call this concept your aggressiveness, as it closely resembles how aggressive you are for going in for the win. Generally in Breakthrough it is favorable to move your

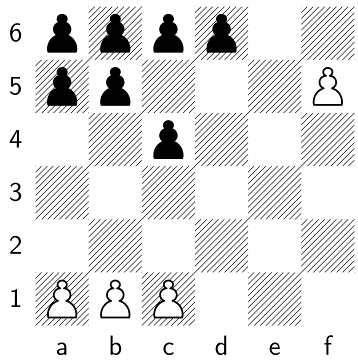


Figure 2.2: Breakthrough board where Heuristic 1 doesn't work well

whole team as a unit and play more defensively (CITE UNITY HEURISTIC), so this heuristic is probably not one that will lead to the most wins.

## 2.4 State-Space Search

Traditionally methods for playing games were searching through the environment using a heuristic to guide the search. A simple way of doing a heuristic based search would be to give all states a 0 as a score and give states where you win a positive score. We say that those search algorithm is not guided, and the algorithm will probably have to evaluate every state in the state-space. This method of searching is generally extremely inefficient as the state-spaces of game environments are often extremely large, and sometimes infinite. For instance an estimate of the state-space for the game breakthrough is $M * N * 2^{N*4}$ where $M$ is the height of the board $N$ is the width of the board. The $2^{N*4}$ represent each piece either being alive or captured. So for a small board $5x4$ the estimated state-space is 1310720 many states.

This is why a good heuristic is very valuable because we can disregard all states $S'$ that result from doing a move $a$ in state $S$ as they will only lead to worse outcomes.

The algorithms that are used in traditional state-space search are for instance Depth-first search (DFS), Breadth-first Search (BFS), Alpha-Beta Pruning Search (AB-Search), and Monte-Carlo Tree Search (MCTS).

More modernly these search methods have been amplified by Machine Learning, in such a way that we do not need to figure out a good heuristic for a given state, but rather, we apply a machine learning model to learn a function that takes in a state and returns an evaluation of that state. This can lead to a significant time reduction as we do not need to simulate a whole game from a state to receive its evaluation we rather receive the evaluation from the model.

### 2.4.1 Monte-Carlo Tree Search

In the algorithm Monte-Carlo Tree Search there is an agent exists within some environment. Where each node in the environment represents a state-action pair of the environment, by state-action pair what is meant it is some state and the action that brought the agent to that state. This pair should be unique within the environment. We will discuss these concepts within game-environments and therefore we might say player instead of agent and game instead of environment.

MCTS is a method of exploring an environment in a randomized manner (Monte Carlo is the term implying randomness). In MCTS there are four stages. Selection, Expansion, Simulation, and Back-propagation. They happen sequentially and repeatedly. MCTS is initialized with a tree consisting of the unexpanded initial state of the environment.

In MCTS there is a tree representing the game-environment. This tree consists of nodes $n_i$ where $i$ represents the point in time of the node, for example $N_0$ in chess is the initial position and $N_x$ is some position in the middle of the game and $N_e$ is one of the states representing a position where there is either a draw or one player has won the match. Each of the nodes has 4 values, $s$, $a$, $Q$, and $N$. These values represent these items, $s$ is state of the environment, $a$ is the action that brought the previous node $n_(i-1)$ to node $n_i$, $Q$ is the average value of the node (value meaning the outcome of the game), and $N$ the amount of times the MCTS algorithm has visited this node. The values $s$ and $a$ uniquely identify a position in the environment and are often called state-action pairs.

The MCTS algorithms four phases

1. Selection
2. Expansion
3. Simulation/Rollout
4. Backpropagation

$$\text{Child UCT value} = \frac{Q_{(s',a')}}{N_{(s',a')}} + c_{uct} * \frac{\sqrt{log(N_{(s,a)})}}{N_{(s',a')}} \tag{2.1}$$

## Selection

During the selection phase a node $(s, a)$ within the tree which has not yet been expanded is found. This process uses Upper Confidence Bound on Trees (UCT) to find that node $(s, a)$, the formula is described in Equation 2.1. For a parent node $(s, a)$ (initially the root of the tree) we select the child with the highest UCT value. Repeatedly until an unexpanded node is found. This process is done to balance the amount of exploration vs exploitation of nodes in the tree.

## Expansion

Then the expansion phase expands the node generating all of $(s, a)$'s children $(s', a')$ are generated and connected to $(s, a)$. This is done by applying all actions $a'$ to $(s, a)$. These children are the product of applying each action $a'$ to $s$ in the parent node.

## Rollout

Next during rollout actions from $(s, a)$ are randomly selected to move to $(s', a')$, then repeated to go to $(s'', a'')$, until a terminal node within the environment is reached. By terminal we mean a state in which the game is finished. A terminal node in MCTS can generally return any value, but in the context of this paper we only return (+1 white wins, -1 black wins, 0 draw).

### Backpropagation

The result from the terminal node is then propagated up through the path taken by selection $(s, a)$ up to the root of the tree, updating the $Q_{(s,a)}$ values of each node $(s, a)$.

When training a neural network the UCT formula is modified slightly to prefer selecting nodes that the neural network values highly by introducing a second scalar to the formula $f(s, a) = (p, v)$, the resulting formula is described in Equation 2.2, and is called PUCT. Secondly, the backpropagation process is modified to instead of doing rollout/simulation to recieve a reward the predicted value $v$ from the neural network is used instead.

$$\text{Child PUCT value} = \frac{Q_{(s',a')}}{N_{(s',a')}} + c_{uct} * P((s, a)) * \frac{\sqrt{log(N_{(s,a)})}}{N_{(s',a')}} \tag{2.2}$$

## 2.5 Machine Learning

Machine Learning (ML) is a research field in which machines apply statistical functions on data to achieve the output that is *correct*, by correct we mean the corresponding result which we expect. Generally this a repetitive process where we look at examples of the data, and the algorithm progressively gets closer to the underlying function of the data it is fed. This process is therefore similar to trial and error for humans. ML is a sub-field of Artificial Intelligence. ML algorithms come in the form of two classes *Classification*, where the algorithm should find a class representing the data its given, and *Regression* where the algorithm should find an underlying continuous numerical function and will return a numerical value representing the input.

Typically ML can be viewed in three different groups, Supervised Learning, Reinforcement Learning (RL), and Unsupervised Learning. Where in Supervised Learning, the algorithm is given data examples and their corresponding outcome. For example, a supervised learning algorithm could be provided with data regarding the weather and the corresponding temperature, the algorithm should then find a pattern within the weather data and find the continuous function represented by the data. This would then be an example of a regression task. Flipping thing example around, if the algorithm would just be provided the temperature and it should tell us whether it is sunny outside or not, that would be a classification task.

In RL the algorithm is given only some input, and then the algorithm tries some outcome generally actions in some environment. Then over an episode [1] once the episode is finished some reward is given. The algorithm will then learn whether the actions were good actions from the reward. Examples of this are agents playing a game like Flappy Bird, where the data they are given is the state of the game, and they try to either jump or not jump.

### 2.5.1 Supervised Learning

In supervised learning, the data and their corresponding outcomes are already available to us. Examples of the data we can use in supervised learning would be labeled images.

### 2.5.2 Reinforcement Learning

Reinforcement Learning is where a model learns from experience. That is, the notion of input/output values changes, the model uses itself to generate input values by acting in an environment. The environment then returns some result, that could be losing in a game, making a correct prediction, or any number of outcomes. The result is then propagated through the model allowing it to improve with this new information.

Many algorithms are popular in reinforcement learning, for instance Q-learning(CITE ME), CARLA (CITE ME), and many others.

### 2.5.3 Neural Networks

Neural networks (NN) are popular methods within a sub-field of ML which is called Deep Learning (DL). NN's are created to resemble how the human brain functions. In the brain we have neurons which when they get a signal they apply some function to them and if the resulting signal is high enough, they fire to the next neuron. This is

[1] a series of outcomes / a timespan

how it is done in the neural network model as well. There we have neurons that when they get some input, generally a vector of numbers. The neuron takes the sum of that vector, weights the sum by a constant, then applies a non-linear activation function to it. The result of doing this is then passed on to the next neuron. Until a final layer of neurons is reached. At that point we have a value that the neural network corresponds to the input value. This value can be a binary classification (cat or dog image), a regression value (the value of a property), or any number of outputs. It can then be said that a neural network is doing a function approximation of an input to some value. $f_n(w_n * f_{n-1}(w_{n-1} * \ldots f_0(w_0 * i))) = o$.

Neural networks are machine learning models that take in as input anything that is numerical and they apply differentiatable function to that input s.t. they end with some output. This output is then compared to the expected output the difference between these outputs is called a loss. A loss is backpropagated through the neural network, and each function is derived in order to find its slope. We can then modify the weights of the neural network in the direction of the correct output. Leading to a function approximation of this function $f_{neuralnetwork}(X_{input}) = O_{expectedoutput}$

## 2.6 Explainable Artificial Intelligence (XAI)

The field of XAI research is still very far behind it's counterpart AI research, within XAI two fields are concidered the largest that is Model Interpretability and Model Explainability.

### 2.6.1 Model Interpretability

#### 2.6.1.1 Saliency Maps

Within XAI many methods have been developed to try to evaluate ANN's, these methods are often referred as model interpretability methods. In the field image recognition there has been a lot of work examining which pixels of an image the model deems important. Arguably the most popular method for this is Saliency Maps (CITE ME).

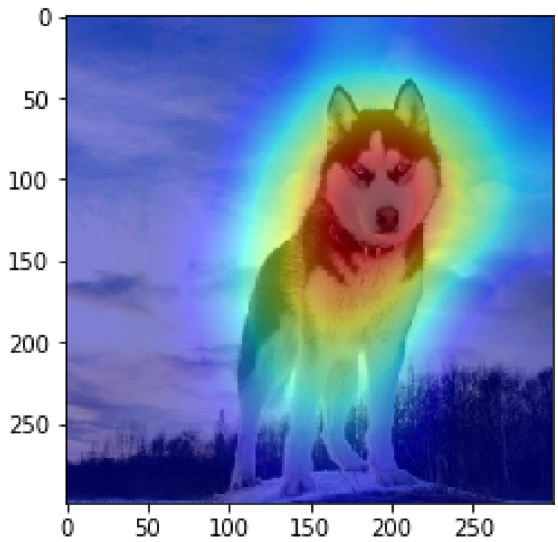


Figure 2.3: Example of a models saliency map for an image of a dog

There the pixel values the model deems important are colored in s.t. a human can examine the image and get a sense of what portions of the image are important to the model, an example of a classification of a dog can be seen in Figure 2.3.

While this method is understandable in the context of image recognition it lacks severely when your input is not an image.

#### 2.6.1.2 Shapley Values

Methods for explaining models that aren't image recognition models include shapley values. There the input is examined against it's output, then iteratively input values are selected to be fixed. Then the other input values are varied and an average change in prediction is calculated. With this the shapley value can be estimated for the fixed input value. This is done to examine which input values have the strongest link to the output value. Shapley values on a dataset can give insights on which input values the model deems important.

#### 2.6.1.3 Concept Activation Vectors

A recent paper by Been Kim Et. al (CITATION NEEDED), shows a method for examining a neural network giving a much more human insight into a prediction. Using Concept Activation Vectors (CAV) a directional derivative for a given input can be examined with respect to some HLC's. For example, when a human looks at an image of an animal and is supposed to decide whether the image is of a horse or a zebra, an intuitive approach would be to check whether the animal has stipes, or is both white and black. That method of determining if a horse is a zebra could then be called a higher-level concept, and if we're able to gather if a nerual network uses this strategy for prediction we have a deeper understanding of its underlying structure. Leading to an explanation of the result.

CAV's in neural networks are just binary classifiers of the same dataset used to train a neural network. The

### 2.6.2 Model Explainability

Model explainability within the context of neural networks isn't possible today. Model explainability referrs to firstly considering some input and output from a model. Then afterwards the model is examined to determine exactly what led to the predicted output. This concept is simple when we're working with Decision Trees. A decision tree is a tree whose nodes are representative of an input value and at every node a branch is selected based on the value of the input value. It is therefore easy to see how to examine the tree to explain the output. We just follow the branches in the tree. That being said the branches are created by algorithms like ID3 which construct branches based on the initial dataset used to construct the tree, but again ID3 follows simple statistics and can be explained properly.

When we talk about neural networks this process is much more difficult, the underlying nodes are generally in the thousands, the different layers of the neural network varies in the operations it applies to the input value and such while travelling through the neural network the modified value becomes far removed from the initial input value to the eyes of the reader. That being said, while the possibility of completely monitoring the training process and completely monitoring the evaluation process is truly possible it is not feasible. And secondly the process of seeing an input and it's corresponding output will not be of any value if one were to consider the process of prediction.

This part of the dissertation describes the work done.

# Chapter 3

# Methods

In this chapter we describe our implementation of the game Breakthrough and the methods for training a neural network to play the game Breakthrough, described in Chapter 2. The training process uses MCTS described in Section 2.4.1 to guide its training. We outline the architecture used in the neural network.

The training algorithm is a reinforcement learning self-play algorithm based on previous work by DeepMind (CITE ME).

## 3.1 Implementation

In this section we will take a brief look at the implementation of the game Breakthrough, how we model its state representation, and the pseudocode for training.

### 3.1.1 Breakthrough

In order for a neural network to be able to use input, that input needs to be numeric. We model the board in gamethrough with a single $NxM$ array with three values $\{-1, 0, 1\}$. Where each $x, y$ index on the board represents what value is on the board at the time, $-1$ represnts a white piece, $0$ represents an empty square, and $1$ represents a black piece.

A action vector is coupled with each state, the action vector is a 3-dimensional matrix where the lengths of the dimensions are $rowxcolumnsx6$ and each index of the vector $X, Y, Z$ represents an action moving from cell $x, y$ moving to the direction $z$. The directions are as follows 0 represents moving upwards and to the left diagonally on the board, 1 upwards, and 2 upwards and to the right diagonally, 4 downwards and to the left diagonally, 5 downwards, and lastly 6 downwards and to the right diagonally.

### 3.1.2 Selfplay

## 3.2 Neural Network Architecture

The neural network architecture we opted to use was a single convolutional layer with a ReLU activation function, followed by 5 residual layers each containing two layers of convolution, batch normalization and a ReLu activation. Lastly a split policy/value head, where the output of the last residual layer is split into two different outputs. The policy head are two layers: first a final convolutional layer, and last a fully connected layer with a *log softmax* layer. The value head, consists of three layers: first a convolutional layer, then a fully connected layer, and lastly a fully connected layer with a *tanh* activation function. The Figure 3.1. depicts the architecture.

### 3.2.1 Convolutional Layers

It is important to understand why we use convolutional layers when dealing with board games as convolution is often associated with image. This is because a convolutional layer is focused on merging multiple input parameters to a single neuron, making it an input parameter for the next layer representing the locality around the center point of the original input. Playing the game of Breakthrough, a piece in only able to capture pieces in its immediate vicinity. This lead to the selection of a convolutional kernel with the size of $3x3$ and a stride of 1.

The knowledge of each piece moves isn't enough to represent the whole gamestate within the neural network. This is why we use multiple residual layers with convolution. These residual layers stack convolutions on each other making the final convolution

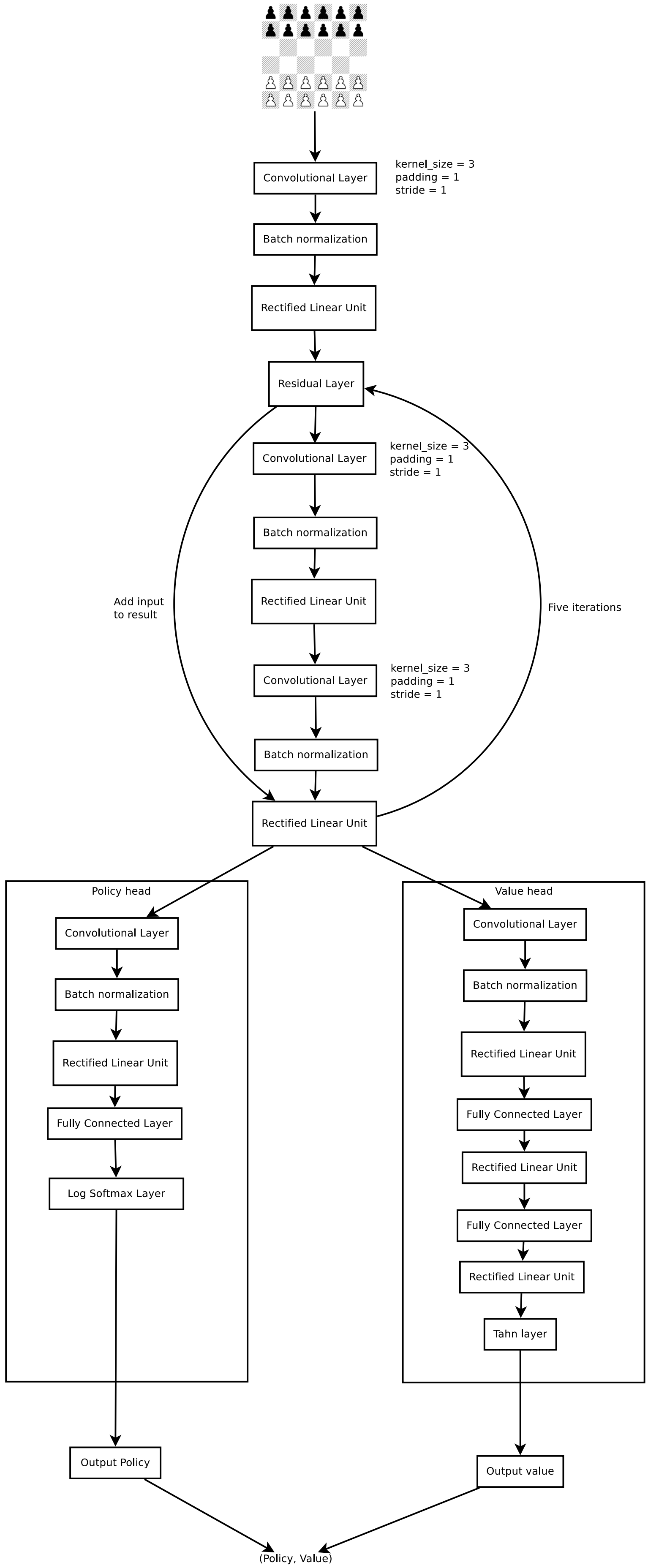


Figure 3.1: Neural Network Architecture

represent the locality of all the other localities, allowing the neural network to have a representation of the whole gamestate in its parameters.

The selection of parameters was done for these reasons as well as to most closely resemble the architecture described in AlphaZero.

### 3.2.2 Residual Layers

The residual layers as a concept remember the output of the previous layer and add the results of the residual layer and the previous layer together. On a higher level this leads to the internal representation of the neural network to maintain the whole game boards in its representation, s.t. two areas that are far away from each other maintain the same level of locality as two that are close.

Why we use a residual layer for applying a convolution layer multiple times is to gain locality of the whole game board. A residual layer applies a function like this $res(x) = x + l(x)$ where $l(x)$ is the function of the layer, commonly a convolutional layer.

### 3.2.3 Policy Head

The policy head of the neural network returns a vector of the size of the action space $w * h * 6$ where $w$ is the width of the board $h$ is the height of the board and 6 represents the six cardinal direction pawns can move (straight, and diagonally both left and right for the first player, and backwards for the second player). The vector is then masked s.t. the values representing moves that can not be played on the board are given the value of 0.

### 3.2.4 Value Head

The value head of the neural network returns a single value representing the predicted value of the neural network. This predicted value is trained to be the end value of the game after taking the predicted move.

### 3.2.5 Self-play

To train the neural network to play Breakthrough we initialize two neural networks $N_1$ and $N_2$ with random weights. Then we let $N_1$ play against itself using MCTS to direct its training. While playing against itself the neural network gathers data for it to train with.

The data collected is $(\pi, \tau, p, v)$, where $\pi$ is the action probabilities provided by MCTS for $N$ iterations, $\tau$ is the reward for the whole episode, 1 if white wins, $-1$ if black wins, $p$ is the policy vector from the neural network, and $v$ is the predicted reward of the game by the neural network.

This data collection trains the neural network in such a way that it is a function approximation for a single state to return the values that would have been returned from applying MCTS for $N$ iterations.

### 3.2.6 Loss Function

The data collected is backpropagated through the NN moving the weights to the direction of this loss function $l = (p * \pi) + (\tau - v)^2$ for each state the NN encountered during self-play.

## 3.3 Explainable state representations

In this section we describe the

### 3.3.1 Testing With Concept Activation Vectors

Once the neural network has learned to play the game of Breakthrough, we examine its internal state w.r.t HLC's that we understand. The first examined HLC is numbers advantage, that is, the number of pieces that a player has over his opponent. The higher level idea to a human player would be that they are in a better position since they have more pieces.

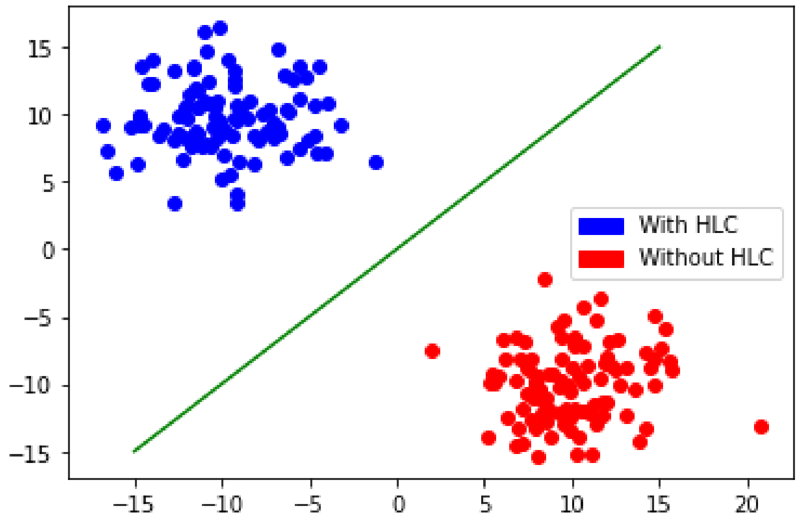


Figure 3.2: Trained linear classifier on a 2-dimensional space

To examine the higher level component we take a look at the internal state of the neural network itself. To do this we take a state, we run the state through the neural network, and while the state is propagating through the network we select a layer to split the network, we call that layer $l_{split}$. At that point in time the state is a mutated vector $l_{split}(l_{split-1}(...l_0(state)))$ of the initial state. We save that vector as a point in the N-dimensional space that it represents, and once we've gathered enough points in that N-dimensional space we're able to train a linear classifier using the HLC as a label. We train a Stochastic Gradient Descent classifier, from the SKLearn library, to construct a hyperplane that splits the space into two binary states, one that contains the HLC and another that doesn't. An example of how this would look with only two dimensions is shown in Figure 3.2 Once we've successfully trained this linear classifier we can run another state $s$ through the neural network. Once the prediction has finished, we view the selected action $a$ of $s$ from the policy vector $p$ from the neural network. We view the loss of that selected action $a$. Then we apply the backpropagation algorithm up to that same layer $l_{split}$ and evaluate the gradient there. If that gradient moves to the direction of the HLC we say that the state includes the HLC, and doesn't if it move away from the HLC. We show an example in Figure 3.3. where the arrow in the image represents the direction the gradient is moving, if it moves toward the HLC the state $s$ includes the HLC otherwise it does not. Using this method we evaluate the internal

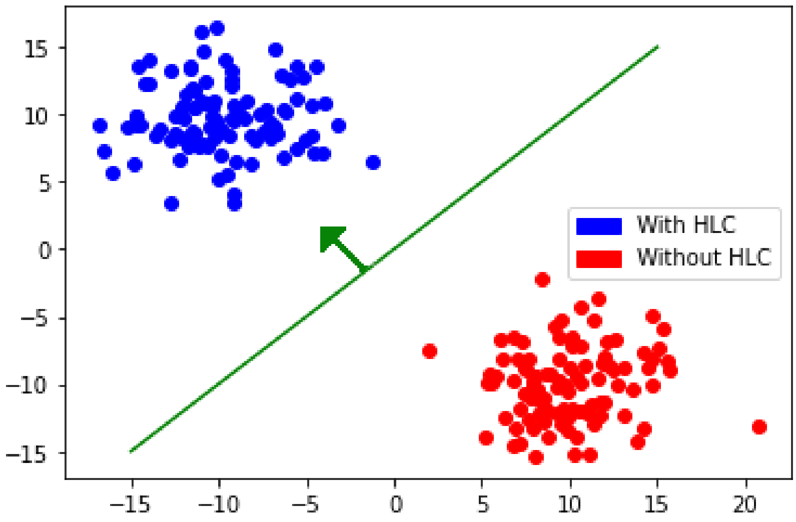


Figure 3.3: Trained linear classifier on a 2-dimensional space with arrow representing gradient

representation of the state within the neural network, and are able to see whether it recognizes the HLC.

# Chapter 4

# Results

In this section you discuss any issues that came up while developing the system. If you found something particularly interesting, difficult, or an important learning experience, put it here. This is also a good place to put additional figures and data.In this section we discuss the results of considering HLC within the game Breakthrough, using a Concept Activation vector for evaluating how the neural network recognizes the HLC's. We first take a look at the changes in ephasis of the neural network during training. The main point of interest there being whether the nerual network notices simple HLCs early then stops taking them into consideration as the network improves.

## 4.1 Evaluating the improvements of the neural network over generations

To test which HLC the neural network places its emphasis on during training we trained a neural network for 200 iterations, taking snapshots of the network every 10 generations. Then we had the neural network play against itself for 100 games collecting the states it encountered during play. These states were then examined by a concept activation vector representing these HLCs. Firstly there is the numbers HLC, where the number of pawn the player has minus the number of pawns the opponent had, the breakpoint we selected was 2 meaning that if you have 2 more pawns then your opponent, you're in a position where a HLC called numbers advantage is present.

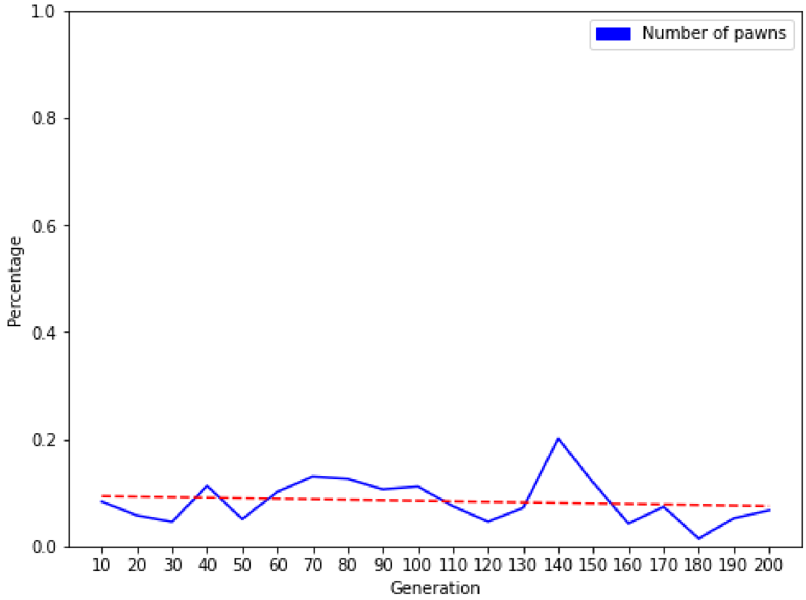


Figure 4.1: Percentage of selected states containing the HLC numbers advantage

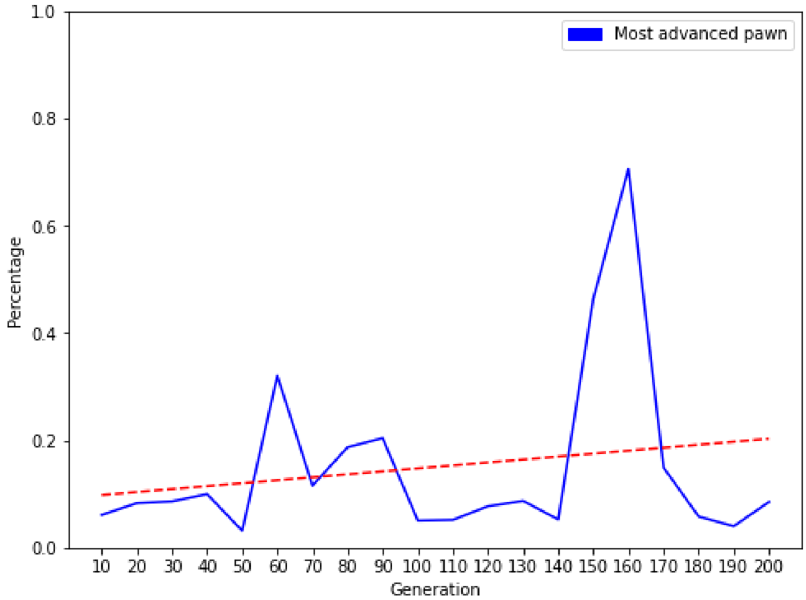


Figure 4.2: Percentage of selected states containing the HLC aggressiveness

The Figure 4.1. shows that over the course of training the numbers advantage HLC is only ever a slight factor in the selection of states and we can say that the neural network doesn't really consider number advantage in its selection process.

The second HLC we examined was aggressiveness, that is a state in which you most advanced pawn is 2 or more squares further than the opponents most advanced pawn.

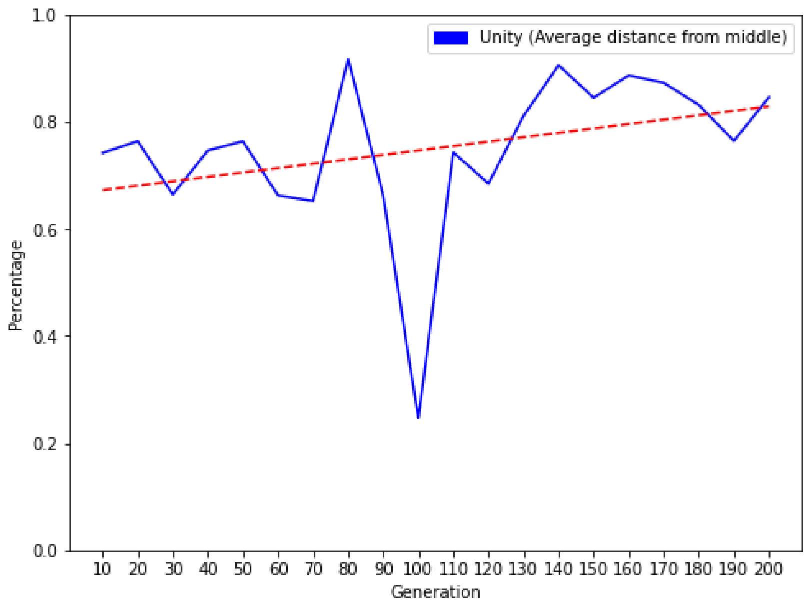


Figure 4.3: Percentage of selected states containing the HLC aggressiveness

From Figure 4.2. we can see that the aggressiveness HLC is a growing factor over as the nerual network is trained. Generally when your opponent is not skilled this strategy is considered a good one.

The last HLC that we examined was unity, the unity HLC represents the absolute average distance of your pawns from the center row of your pawns. This HLC is calculated as the row of your furthest pawn from the starting row $r_{far}$, the row of your nearest pawn from the starting row $r_{near}$. Finding the middle row is then $\frac{r_{far}+r_{near}}{2}$, we then take the absolute of the average distance from the pawns to that row. To find the point at which this value relates to a state with the HLC unity, we sampled a myriad of states and decided on the value of 0.35. The value of 0.35 generally allows your states to have two rows that have the majority of the pawns and one or two pawns one row away from the group.

The literature of breakthrough implies that being patient and waiting until your opponent makes a mistake is generally the preferred strategy for winning, and the concept activation vector for Unity is generally triggered, and increases with the amount of generations the neural network is trained for. These results imply that the neural network does infact recognize successful strategies, and focuses its training in the direction of the successful strategies.

This part of the dissertation talks about future work discussion concludes the work and

# Chapter 5

# Discussion

Here I will discuss the myriad of fields research like this can help for instance for health organizations, patients, taxpayers, and pharmaey

## 5.1 Summary

summarize the workey

## 5.2 Conclusion

conclude the work, discuss the significance of the workey