

**UNITED STATES**

**MILITARY ACADEMY**

**WEST POINT, NEW YORK**

**BetaGo: Using Convolutional Neural Networks**

**to Train Go Playing Artificial Intelligence**

by

David Weidman

May 2016

Research Advisor: Major Natalie Vanatta

Second Reader:

**MA491 Project Report**

THIS PAGE INTENTIONALLY LEFT BLANK

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **REPORT DOCUMENTATION PAGE** | | | | | *Form Approved OMB No. 0704-0188* | |
| Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503. | | | | | | |
| **1. AGENCY USE ONLY *(Leave blank)*** | | **2. REPORT DATE**  05/16 | | **3. REPORT TYPE AND DATES COVERED**  Senior Thesis | | |
| **4. TITLE AND SUBTITLE**  Dynamic Network Analysis | | | | | **5. FUNDING NUMBERS** | |
| **6. AUTHOR** David Weidman | | | | |
| **7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**  United States Military Academy  West Point, NY 10996-1786 | | | | | **8. PERFORMING ORGANIZATION REPORT NUMBER** | |
| **9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES)**  N/A | | | | | **10. SPONSORING/MONITORING**  **AGENCY REPORT NUMBER** | |
| **11. SUPPLEMENTARY NOTES** The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. | | | | | | |
| **12a. DISTRIBUTION / AVAILABILITY STATEMENT**  Distribution Statement (mix case letters) | | | | | **12b. DISTRIBUTION CODE** | |
| **13. ABSTRACT (maximum 200 words)**  In recent years, the growing prospects of deep learning in the fields of human-versus-computer artificial intelligence programming has become apparent to mathematicians and computer scientists alike. The widespread publicity began in 1996 when the IBM computer Deep Blue did what no other machine had done before: beat the reigning world champion, Gary Kasparov, in chess. Since then, researchers have continued this progress. One game, Go, has, until recently, proven too difficult for the AI world to conquer. This East Asian game of strategy is uniquely difficult for computers to play due to both the rate at which its computations increase and its tendency towards play-by-feel rather than deterministic logic. In this paper, I will use a deep learning methodology within the TensorFlowTM machine-learning framework to replicate the success that other research teams have had in generating successful strategies for competing with a human being in Go. | | | | | | |
| **14. SUBJECT TERMS** | | | | | | **15. NUMBER OF PAGES** 42 |
| **16. PRICE CODE** |
| **17. SECURITY CLASSIFICATION OF REPORT**  Unclassified | **18. SECURITY CLASSIFICATION OF THIS PAGE**  Unclassified | | **19. SECURITY CLASSIFICATION OF ABSTRACT**  Unclassified | | | **20. LIMITATION OF ABSTRACT**  UL |

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)

Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

In recent years, the growing prospects of deep learning in the fields of human-versus-computer artificial intelligence programming has become apparent to mathematicians and computer scientists alike. Widespread publicity of machine versus human game playing began in 1996 when the IBM computer Deep Blue did what no other machine had done before: beat the reigning world champion, Gary Kasparov, in chess. This victory on the part of the Deep Blue team was the beginning of a new paradigm in the computing world. Since then, researchers have continued progress in game-play arena, most recently through the application of deep learning. One game, Go, has, until recently, proven too difficult for the AI world to conquer. This East Asian game of strategy is uniquely difficult for computers to play due to both the rate at which its computations increase and its tendency towards play-by-feel rather than deterministic logic. In this paper, I will use a deep learning methodology within the TensorFlowTM machine-learning framework to replicate the success that other research teams have had in generating successful strategies for competing with a human being in Go.

* KEY WORDS: Machine-Learning, Convolutional Neural Network, Deep Learning, Go, TensorFlowTM

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. HISTORY and Orientation 1

A. aRTIFICIAL INteLLIGENCE 1

B. Game Playing 1

C. MACHINE LEARNING 2

II. Go 4

A. Rules 4

B. Why is it Difficult 7

iII. MACHINE LEARNING TECHNIQUES 9

A. The Perceptron 9

B. Artificial Neural network 10

C. Convolutional Neural Network 11

D. Feature Extraction 11

iv. MY WORK 13

A. Differences with AlphaGo 13

1. Monte Carlo Search 13

2. Genetic Algorithm 14

3. AlphaGo Algorithm 14

B. Datasets 15

C. feature Extraction 15

3. Manhattan Distance to Previous Move 15

D. Results 16

V. APPENDIX 17

A. TensorFlowTM 17

B. Docker 17

C. Raw Code 17

LIST OF REFERENCES 27

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

Figure 1 (from Ref. Carnegie Mellon) 4

Figure 2 (from Ref. CMU) 5

Figure 3 (from Ref. CMU) 5

Figure 4 (from Ref. CMU) 5

Figure 5 (from Ref. CMU) 6

Figure 6 (from Ref. CMU) 7

Figure 7 (wikipedia.org/wiki/Perceptron) 9

Figure 8 (wikipedia.org/wiki/Multilayer\_Perceptron) 10

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Creating and Artificial Intelligence capable of effectively playing Go, an ancient Chinese board game, has challenged the computer science world for decades. Due to several unique characteristics, Go poses difficulties that other board games which have long been solved, do not. Chief amongst these characteristics is the sheer number of potential game states. The immensity of the Go state space renders previously successful game-playing techniques, such as tree searches, infeasible. Instead, artificial intelligence experts have looked towards the ever-growing field of machine learning to address the problem.

In early 2016, the artificial intelligence company Google DeepMind wrote a long-pursued program that was capable of consistently defeating the reigning Go world champion, Lee Sedol. In order to do so the team leveraged several machine-learning techniques: deep learning, Markov tree searches and genetic algorithms. With these techniques, the program, AlphaGo, performed better than any other Go playing machine had before. In order to explore some of the concepts which comprise the cutting edge of the machine-learning world, this paper looked to address a simplified version of the Go problem. In order to do so, it would forge an effective 9x9 Go player using solely deep learning concepts. Ultimately these endeavors found (little, moderate, high) levels of success.

In order to build on the work in this paper, future researchers should incorporate into their algorithm the two machine learning techniques that this paper did not address. In order to catch up to the status quo, these further strategies for move prediction are essential. They should furthermore expand the scope to 19x19 games. This would require the acquisition of a new dataset.

I was not able to draw significant conclusions from the paper. The end result was able to play Go to some degree however I was unable to measure exactly to what degree it was effective at doing so. I did however create a basis for myself, and potentially others, to explore machine learning in the future.

THIS PAGE INTENTIONALLY LEFT BLANK

# I. HISTORY and Orientation

## A. aRTIFICIAL INteLLIGENCE

The study of Artificial intelligence is the attempt to understand and construct intelligent entities.[[1]](#footnote-1) The idea of constructing a machine that is capable of thought has fascinated philosophers, authors and scientists alike for hundreds of years. Beginning on a merely theoretical platform, intellectuals first addressed the idea of thinking machines through works of fiction. Notable examples of this are the ancient Greeks’ allusions to Talos, a mechanical man crafted by the gods, and, far later, Mary Shelley’s Frankenstein. In these works, the seeds were sewn for future exploration in a field of science that had not yet been developed.

Alan Turing, the father of computer science, built upon the fancies of these early prognosticators by realizing many of the previously, solely theoretical concepts. In 1950, he devised what has since come to be known as the “Turing Test.”[[2]](#footnote-2) The Turing Test is an assessment of whether or not a third party human evaluator would be able to distinguish reliably between a machine and a human being who are communicating with each other solely through natural human language. A machine that is sufficiently able to fool this third party human will “pass the Turing Test.”

## B. Game Playing

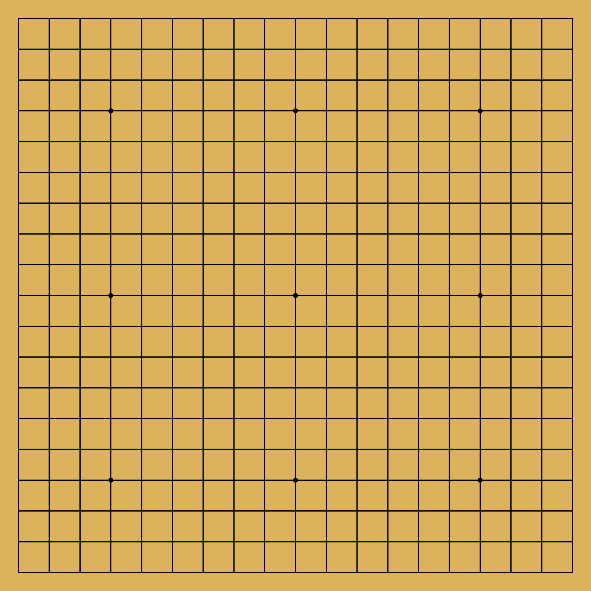
Parallel to the development of human language based assessments was the more functional development of logic-based assessment. One of the best ways to test a machine’s reasoning abilities is to pit it against a human in a strategic board game. All board games exist in one of three states.[[3]](#footnote-3) The first state is *solved.* A game is “solved” if the outcome of the game can be predicted from any position given that each player plays optimally. Tic Tac Toe and Connect Four were some of the first games that humans were able to solve computationally due to their relatively small domain of potential board configurations. The next state is *partially solved*. These games generally have a search tree that is too large for a computer to efficiently or possibly search. They must therefore resort to computational techniques outside of simple tree searches. Chess, Jeopardy and, as of recently, Go fall within this category. In each of these games, a piece of software can consistently beat the “best” human beings. The widely publicized Deep Blue-Kasparov match in 1997 and the AlphaGo-Lee match in early 2016 are examples where American corporations used these human achievements to present the technological progress their researchers had made to the public. The last category is *unsolved.* Unsolved games are those in which human beings still have an advantage in one-on-one contests. This ever-shrinking list of games includes some varieties of poker but as of the last five years, not much else.[[4]](#footnote-4)

## C. MACHINE LEARNING

We can attribute much of the progress in the world of human-machine gameplay between the Deep Blue and AlphaGo matches to machine learning. Machine Learning is the field of study that gives computers the ability to adapt to environments and detect patterns.[[5]](#footnote-5) Machine Learning combines elements from computational statistics, optimization, data mining and several other fields of mathematics and computer science to achieve this task. This paper will primarily focus on the application of a subfield of machine learning, artificial neural networks, to the problem of programming a machine to play Go. Artificial neural networks (ANNs) are the emulation of biological neurons through algebra and computation. The idea was first postulated in the 1940s.[[6]](#footnote-6) After years of honing its mathematical underpinnings and substantial advancement in computational efficiency, ANNs found themselves at the center of the machine learning world. The natural progression of Moore’s Law, which has allowed computation to progress at an exponential rate, eventually paved the road for an exhaustive deployment of ANNs in the form of convolutional neural networks (CNNs.) The concept of CNNs will act as the basis for this research.

# II. Go

## A. Rules

Go is an eastern Asian game of strategy. Originating in China 2,500 years ago[[7]](#footnote-7), the game spread throughout the world as a result of both its simplicity (in rule and board design) and its functionally infinite possible outcomes. Go is known as a game of encirclement wherein the ultimate goal is to capture more territory than your opponent does. In most official capacities, it is played on a gridded 19x19 board. Restricting the board size however does not alter the underlying rules of the game.

In Go, two players alternatively place black and white stones on the initially blank intersections of the lines on the board. Because the ultimate goal is to surround more territory than your opponent, the players must strike a balance between maximizing their own territory and minimizing that of his or her opponent. If all bordering intersection points (referred to as liberty) of a contiguous group of stones are occupied by the opponent’s stones, the encircled player’s stones will be removed from the board. These stones are considered *captured* and will eventually be counted towards the capturer’s final score. Each of the white stones on the figure 2 example board would be removed from the board.

Figure 1 (from Ref. Carnegie Mellon)

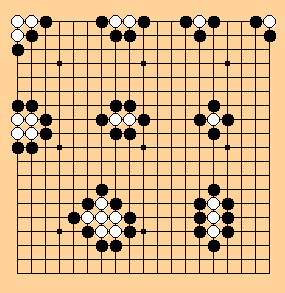
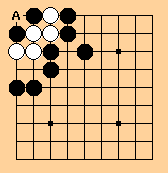
A player may not place a stone where it has no liberty except in the circumstance that it is a capturing move that will create additional liberty. This move is called suicide and is illustrated in figure 3. In most circumstances, white would not be allowed to place a stone at location A where it is surrounded by black stones. However, in this instance A is a capturing move and will therefore result in the two black stones being removed from the board. The move is therefore permitted.

Figure 2 (from Ref. CMU)

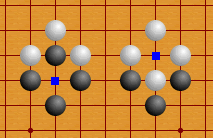
The next basic concept is called ko. Ko is an extension of the suicide rule wherein the stones are configured in such a way that the players can alternatively commit suicide in the same location. In these unique instances, the players are restricted to one consecutive move in the same location. The captured player must wait one move before he/she is able to recapture their lost stone. Figure 4 illustrates a circumstance where Ko could occur.

Figure 3 (from Ref. CMU)

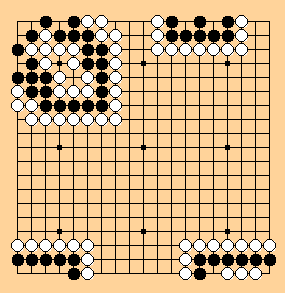
The last basic concept I will address is that of life and death. Ultimately, this definition concerns whether or not contiguous groups of a player’s stones are guaranteed to be captured by a perfectly logical opponent. Central to the idea of life is whether or not the group of stones has two *eyes*. An eye is an internal liberty. We can see this in the top two groups illustrated in figure 5 where white (on the left) and black (on the right) have two separate internal liberty locations. Even using a suicide move, these group’s opponents are unable to capture the group with a single move. Black in the bottom left is similarly alive. No matter what white plays, black can counter with by creating two eyes similar to the top right diagram. The bottom right is the last of the basic Go rules that I will address. The situation is called *seki.* Here, neither black nor white is specifically alive or dead. A move, by black, attacking the internal white stones will result in white placing a stone at its other liberty and capturing black. On the contrary, a move by white, attacking the black stones would similarly result in white’s capture. In this instance, *seki,* neither white nor black will count the territory toward their final score.

Figure 4 (from Ref. CMU)

Figure 5 (from Ref. CMU)

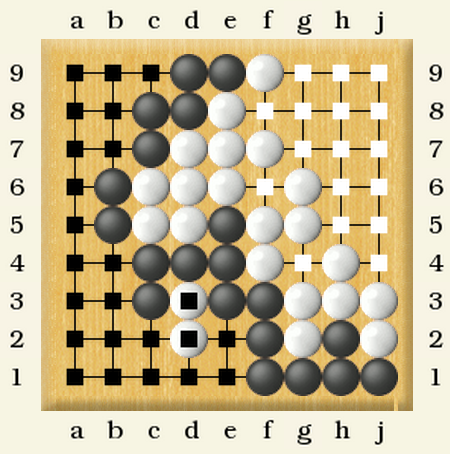
Ultimately, a player’s score is determined by the sum of two values: one, his or her total territory and two, the number of the opponent’s stones the player has captured or are mutually agreed to be dead. Figure 6 is a simple example on a 9x9 board. Each players’ territory is denoted by their respectively colored squares. Assuming neither player had previously captured any of their enemy’s stones white ends the game with 17 points and black with 26 points. Remember that we must count the two *dead* stones in black’s territories towards its ending score. Furthermore, all official games will include komi, or a handicap based on which color a player is playing with, in its scoring. Because black always plays first, it begins the game with a territorial advantage over its opponent. In a typical 19x19 game (under Japanese and Korean rules) komi is set as 6.5 points. This means that white will have 6.5 points added to its final score.

Figure 6 (from Ref. CMU)

## B. Why is it Difficult

Ultimately, the difficulty of partially solving Go is the result of several factors. The first is the size of the board. With a 19x19 board (symmetry notwithstanding), there are initially 361 possible moves. In contrast, chess is played on an 8x8 board and the number of potential moves is far fewer than the number of board locations. On a board where each additional move broadens the search tree by a factor of as much as 360, tree expansion happens at an enormous rate. This leads to a search space of positions.[[8]](#footnote-8) iterative deepening tree searches become unreasonable even when only looking a limited number of moves into the future.

Furthermore, it is more difficult to implement scoring heuristics in Go than in chess. While in chess, a computer scientist can apply analytical scoring functions with relative ease, Go’s tendency towards play by feel does not lend itself to the same analysis. Go requires a new generation of evaluation functions rather than a revision of old ones. A simple, yet effective evaluation function in chess, for example, would be attributing point values to each piece type and then finding the point sum for each player. The function would then place higher utility on situations where the point disparity is highest in the player’s favor. Evaluation functions for Go however are fundamentally more complex. Determining a player’s position strength requires analysis of, amongst other things, the life and death of their stone groups and the amount of influence they have on certain areas of the board.[[9]](#footnote-9) Even professional players must resort to experience and impression because there is no systematic or computationally efficient method for determining these things.

# iII. MACHINE LEARNING TECHNIQUES

The cutting edge of machine Go playing uses several common artificial intelligence techniques including deep learning, Markov Chain modeling and genetic algorithms. My research focuses on just one of these techniques, deep learning. The perceptron is the fundamental unit of a deep neural network.

## A. The Perceptron

A perceptron is a mathematical representation of a single, biological neuron. The perceptron algorithm is considered a binary classification algorithm.[[10]](#footnote-10) This means that given an input, the perceptron will determine to which of two arbitrary groups the input belongs. To illustrate the algorithm we will consider the input vector, . We will define the output as.

 = The **w** vector represents the scalar weighting of each of the inputs. Each of these weights determine how much each input will contribute to the output of the perceptron. *t* is an arbitrary time step or iteration number.

Because a perceptron’s pupose is to classify inputs, it best performs its function after it has been *trained*. The process of training a perceptron consists of feeding it an input with a desired output *d,*

Figure 7 (wikipedia.org/wiki/Perceptron)

The term represents the comparison between output computed by the current perceptron weights, and the desired output This difference is multiplied by the input value, and the learning rate, to yield a value that will be used to correct the previous weight according to the error present in this iteration. This value will then be *backpropagated*. Backpropagation is the process of updating the weights of the neural nodes by one layer at a time working backwardly from the output layer.[[11]](#footnote-13) Each additional training data point will alter the perceptron in such a way that it more accurately and holistically describes the training set of data.

## B. Artificial Neural network

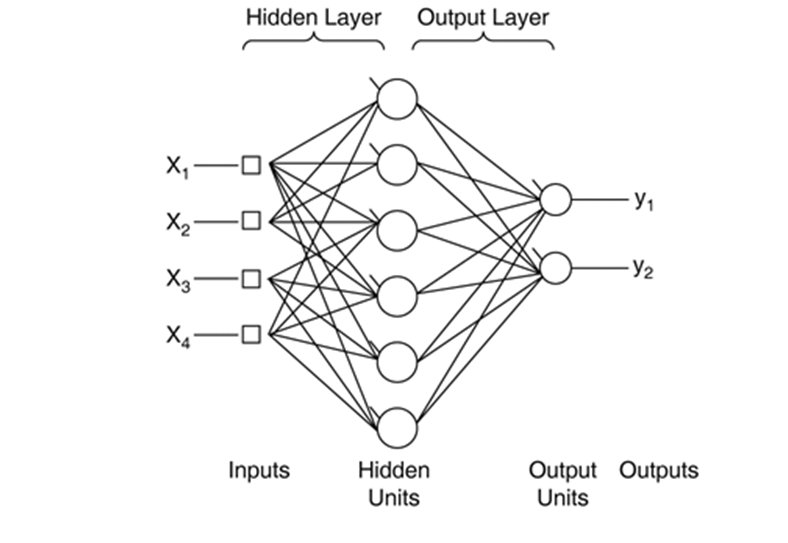
 A feed-forward artificial neural network is also called a multi-layer perceptron because, unlike the perceptron described in the previous section, an ANN has *hidden layers* in between the input and output layers. The hidden layer consists of weighted nodes and is governed by the backpropagation algorithm in the same way that the input layer is. Because it is *hidden,* however, its workings are not visible to a mathematician or program that merely feeds the network a set of training inputs and outputs. The advantage that the additional layer grants the user is an ability to distinguish non-linearly separable spaces.[[12]](#footnote-14) In other words, each additional layer defines an additional linear separator. This allows the ANN to define more complexly divided binary spaces. An example of an input into the system might, for example, be an encoding of a go board where the output is a binary win-loss value. In theory, once sufficiently trained, the network could take a board state as an input and determine who is currently winning the game as an output.

Figure 8 (wikipedia.org/wiki/Multilayer\_Perceptron)

## C. Convolutional Neural Network

Modeled off the animal visual cortex, CNNs have the several features that distinguish them from traditional neural networks. The most important is the principle of locality. CNNs are composed of many overlapping sub-regions of the domain. For example, in a 100x100 picture, a CNN will evaluate and map each 5x5, 6x6 and/or 7x7, etc. block of the picture with a specific evaluation function.[[13]](#footnote-15) The training algorithm then uses these additional mappings as a way to more accurately train the network to the data. This is helpful because this method for training is both more space efficient and more accurate per time-computational unit.[[14]](#footnote-16)

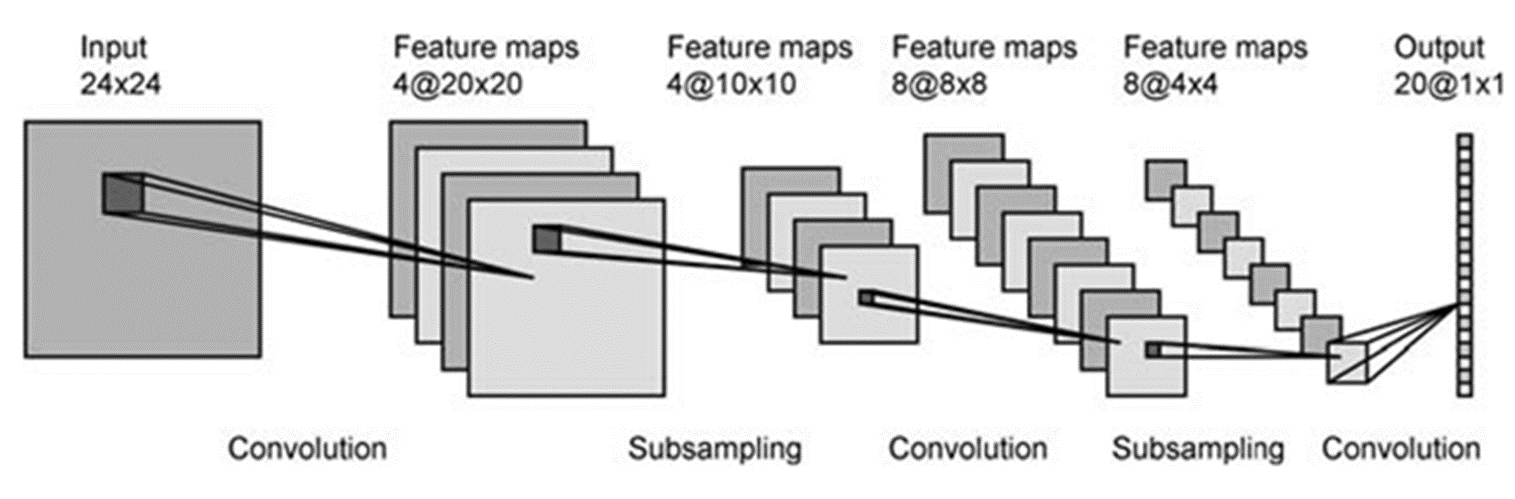


Figure 9 (wikipedia.org/wiki/Recurrent\_neural\_network)

## D. Feature Extraction

Semi-supervised machine learning is offering logical assistance into the learning algorithm through an external human entity. In the instance of feature extraction, the programmer would cue the algorithm to look at specific features.[[15]](#footnote-17) This allows for the program to, in simplified terms, take less time to look for patterns and instead determine what is significant about these features. An example of this could be a program intended to recognize whether a picture contains an alligator. In order to do this the programmer would map each pixel (or convolution) to a parallel array based on an evaluation function. In the alligator example, a relevant feature may be whether a pixel is green. Because alligators are, by definition, green, a higher number of green pixels in a certain area may indicate that an alligator would be in that area. As applied to Go, we can consider a distance-from-the-edge value as an additional determiner of the goodness of a particular game state. Go strategic theory generally considers the third, fourth and fifth rank as valuable early game assets This would therefore allow the feature mapping to bias these positions favorably towards defining a game as a win state. Likewise, non-strategic moves like moves placed directly on the edge will not be valued as highly.

# iv. MY WORK

## A. Differences with AlphaGo[[16]](#footnote-18)

The first major difference however, between my program and AlphaGo is that while AlphaGo was programmed to play on a 19x19 board, my program will instead play on a 9x9 board. Because Go’s rules are constant regardless of board size, this transition is mostly trivial. Games on this size board are popular and databases containing its games exist. 9x9 games have the benefit of allowing me to explore machine learning concepts without requiring the computing power that 19x19 games do. Furthermore, 19x19 games would require more complex feature extraction to achieve good results.

Another difference between my research and AlphaGo’s was the depth of the algorithm. AlphaGo used three major machine learning techniques to find the optimal move within a game. My research will primarily use one, deep learning. While three, genetic algorithm and Monte Carlo searches, would be good finalizing touches, the largest part of the programs functionality stems from the ability to train the program using the datasets.

My exclusion of the datasets from the final algorithm is a result mostly of my time restrictions. The additional complexity that would come with the implementation of these two algorithms is beyond what I would be able to implement in the time allotted.

### Monte Carlo Search

Monte Carlo searches are most commonly applied to game analysis.[[17]](#footnote-19) Whereas typical search trees exhaustively iterate through all possible future game states (or *playouts*) Monte Carlo differs in that it weights its different playouts based on their probability of success.[[18]](#footnote-20) This method is more computationally efficient because the algorithm is essentially *pruning* unfavorable branches of its search tree after it determines their chances of success are negligibly low. In light of the rate of search tree expansion in a typical Go game, this pruning is not only helpful, but necessary.

### Genetic Algorithm

The concept underlying genetic algorithms is based in evolutionary biology theory. In its biological context, individuals within a genetically diverse population who possess superior genes live long enough to pass their genes on to future generations. In that way the rules of the environment ensures survival of the fittest. Within the field of machine learning, computer scientists try to simulate that environmental fitness evaluation with genetic algorithms.[[19]](#footnote-21) Implementations of the concepts that underlie genetic algorithms can vary widely but all contain two basic steps.[[20]](#footnote-22) One is selection. The algorithm chooses which members of the population will breed based on a utility heuristic. The other is breeding. Produce an n+1 generation of the population based on traits from the set selected in step one. Insert an element of randomness between the two generations. The idea of randomness is essential to a functional genetic algorithm. Its purpose is to find potentially *fit* individuals that had not existed in the initial population set. This is important within the context of machine learning in Go because the neural network training is based off of past games. Infinite training of the neural network could therefore only make the machine as intelligent as the combination of all past recorded human Go players. In order for the machine to exceed the playing abilities of humans it needed an element of randomness to explore previously unplayed strategies.

### 3. AlphaGo Algorithm[[21]](#footnote-23)

In brief, the cutting edge in Go playing AI, AlphaGo, used all three of the above machine learning techniques. It began by extensively training a deep neural network using a vast historical record of professional Go matches in the manner described above. In order to widen its base of Go knowledge it used further elements of genetic randomness. These trained deep neural networks allowed AlphaGo to assign utility values to each possible position it could encounter. Once it had these utility values, AlphaGo was able to apply a Monte Carlo tree search. By looking and evaluating many moves into the future, AlphaGo was able to determine its best move at any given time.

## B. Datasets

My data file consists of two important pieces of information. Necessary for the training of the neural network is the input domain vector and the output labelling value. Because I used a 9x9 board, the domain consists of an array of 81 values representing each of the 81 board locations. Each location can either be occupied by a black stone, a white stone, or be empty. The data file represents these possibilities with a “#,” a “0,” and a “.” respectively.[[22]](#footnote-24) Each board configuration is paired with the eventual winner of the game. The pairing of these pieces of information alone will allow for a simplistic training of a neural network.

## C. feature Extraction

The following list contains the list of features BetaGo extracts from each position in order to assist in semi-supervised machine. Each of these features is something that even a beginning Go player will take into account when determining his or her next move.

1. **Capture (Number of Stones)**

Not only does piece capture count towards a player’s final score and directly count towards the capturers end game score but it also generally frees up territory for the capturer. This freed territory will generally positively affect the capturer’s strategic disposition in an indirect way. It is therefore natural that BetaGo should take into account whether or not pieces were captured on a given move.

1. **Distance to Border**

In Go, it is important that players strike a balance between security and territorial gains. The closer a player plays a piece to the edge the safer the play is generally considered. It is, for example, generally considered good practice to play on the fourth and fifth ranks at the beginning of the game in order to strike that balance.

### Manhattan Distance to Previous Move

This feature concerns locality of play. In general, players will play close to their previous. Although this may not have an explicit strategic purpose, it is an easily extractable feature that adds additional information about the playing board.

## D. Results

I was unfortunately never able to test my system against existing Go playing machines. I did however code a mechanism wherein a player could play against the machine I created. Given an input game-state, my Go playing algorithm will use the trained neural network to determine what the optimal next move is. What no absolute basis for comparison I can vaguely interpret my result as a machine that can make reasonable moves sometimes but will not beat a player with even a little bit of go playing experience.

# V. APPENDIX

## TensorFlowTM[[23]](#footnote-25)

In order to implement my program, I used the Google-produced open-source machine-learning framework, TensorFlowTM. Released in November 2015, TensorFlowTM was previously only available within the Google corporate structure. The purpose of TensorFlowTM is to create a more user friendly, more abstracted vehicle to train and use dataflow graphs. TensorFlowTM primarily runs using python but it is also distributed although not as well documented for c++.

## Docker[[24]](#footnote-26)

TensorFlowTM is distributed in Docker form. Docker is a deployment of applications along with all of their dependencies within a self-contained version of the Linux operating system. Docker physically runs on the same operating system kernel as its host however, it exists independently in a virtual sense. An installation of Docker will come with its own filesystem completely isolated from the machine you install it on allowing for a more simplistic installation process.

## Raw Code

* 1. DateRetriever.py

#Author: David Weidman

#https://github.com/djwide/Go9Board

#Read data from the 9x9 data sets available at

#http://dcook.org/compgo/9by9\_experiments.html

from \_\_future\_\_ import absolute\_import

from \_\_future\_\_ import division

from \_\_future\_\_ import print\_function

import os

import tensorflow.python.platform

import numpy

import csv

import pickle

import binascii

from six.moves import urllib

from six.moves import xrange # pylint: disable=redefined-builtin

import tensorflow as tf

def pullPositions(filename):

#Extract the images into a 4D uint8 numpy array [index, y, x, depth].

with open(filename, 'r') as f:

count=0

domain=[]

rang=[]

print('Extracting', filename)

while(count<850000):#f):

if(count%1000==0):

print(count)

count+=1

zobrist=f.read(8)

f.read(1)

board= []

sign=1

playerTurn= f.read(1)

if playerTurn=='W':

sign=-1

else:

sign = 1

for c in f.read(81):

if c== '.': board.append(0.0)

elif c== '#': board.append(1.0)

elif c== 'O': board.append(-1.0)

else: board.append(c)

zts= zobristToScore(zobrist)

if(zts!= None):

domain.append(board)

rang.append(sign\*int(binascii.hexlify(zts[0]),16)\*int(binascii.hexlify(zts[1]),16))

if(f.read(1)== hex(1)): f.read(8)

pickle.dump(domain,open("domain3.p", "wb"))

pickle.dump(rang,open("range3.p", "wb"))

#Retrieve serialized python objects

'''

domain= pickle.load(open("domain2.p", "rb"))

rang= pickle.load(open("range2.p", "rb"))

'''

M= max(rang)

m= min(rang)

dc=0

'''

for x in range(0,9):

print(domain[rang.index(M)][x:x+9])

print(m)

for x in range(0,9):

print(domain[rang.index(m)][x:x+9])

print()

for x in range(0,9):

print(domain[-1][x:x+9])

print(rang[-1])

for x in range(0,9):

print(domain[10][x:x+9])

#rang[1000]

'''

print("pullPositions")

domain= numpy.array(domain)

rang= numpy.array(rang)

return domain,rang

class DataSet(object):

def \_\_init\_\_(self, goBoards, labels, dtype=tf.float32):

"""Construct a DataSet.

one\_hot arg is used only if fake\_data is true. `dtype` can be either

`uint8` to leave the input as `[0, 255]`, or `float32` to rescale into

`[0, 1]`.

"""

assert goBoards.shape[0] == labels.shape[0], ('images.shape: %s labels.shape: %s' % (goBoards.shape,

labels.shape))

self.\_num\_examples = goBoards.shape[0]

# Convert shape from [num examples, rows, columns, depth]

# to [num examples, rows\*columns] (assuming depth == 1)

#assert goBoards.shape[3] == 1

goBoards = goBoards.reshape(goBoards.shape[0],81)

labels = labels.reshape(labels.shape[0],1)

if dtype == tf.float32:

goBoards = goBoards.astype(numpy.int8)

dtype = tf.as\_dtype(dtype).base\_dtype

self.\_goBoards = goBoards

self.\_labels = labels

self.\_epochs\_completed = 0

self.\_index\_in\_epoch = 0

self.\_num\_examples = 50000#images.shape[0]

@property

def goBoards(self):

return self.\_goBoards

@property

def labels(self):

return self.\_labels

@property

def num\_examples(self):

return self.\_num\_examples

@property

def epochs\_completed(self):

return self.\_epochs\_completed

def next\_batch(self, batch\_size, fake\_data=False):

"""Return the next `batch\_size` examples from this data set."""

start = self.\_index\_in\_epoch

self.\_index\_in\_epoch += batch\_size

if self.\_index\_in\_epoch > self.\_num\_examples:

# Finished epoch

self.\_epochs\_completed += 1

# Shuffle the data

perm = numpy.arange(self.\_num\_examples)

numpy.random.shuffle(perm)

self.\_goBoards = self.\_goBoards[perm]

self.\_labels = self.\_labels[perm]

# Start next epoch

start = 0

self.\_index\_in\_epoch = batch\_size

assert batch\_size <= self.\_num\_examples

end = self.\_index\_in\_epoch

return self.\_goBoards[start:end], self.\_labels[start:end]

def read\_data\_sets(dtype=tf.float32):#train\_dir, ):

class DataSets(object):

pass

data\_sets = DataSets()

pos\_file = os.path.join("9x9/", "input\_positions.dat")

zobrist\_file = os.path.join("9x9/", "zobrist\_board\_81.dat")

temp=pullPositions(pos\_file)

dom= temp[0]

ran= temp[1]

data\_sets.train = DataSet(dom[0:50000], ran[0:50000], dtype=dtype)

return data\_sets

#convert board position to fitness value

def zobristToScore(zob):#dictionary

with open(os.path.join("9x9/", "fuego\_chinese.dat"), 'r') as f:

for line in f:

if (zob in line and line.find(zob)+9< len(line)):#efficiency

if(line[line.find(zob)+9]=='W' or line[line.find(zob)+9]=='B') :

ind= line.find(zob)

return line[ind+10:ind+12]

#pos\_file = os.path.join("9x9/", "input\_positions.dat")

#pullPositions(pos\_file)

* 1. MachineLearning.py

#Author: David Weidman

#https://github.com/djwide/Go9Board

#Convolutional Network training .

#See https://www.tensorflow.org/versions/0.6.0/tutorials/mnist/pros/index.html

#for more information

import go9Data

import tensorflow as tf

import numpy as np

goData = go9Data.read\_data\_sets()

sess = tf.InteractiveSession()

#flattened array containing floats

x = tf.placeholder(tf.float32, [None, 81])

#matrices of given dim filled with zeroes

W = tf.Variable(tf.random\_normal([81, 1]))

b = tf.Variable(tf.random\_normal([1]))

y\_ = tf.placeholder(tf.float32, [None, 1])

#implement model w/ given equation

#y = tf.nn.softmax(tf.matmul(x, W) + b)

#backprop

#cross\_entropy = -tf.reduce\_sum(y\_\*tf.log(y))

#train

#train\_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross\_entropy)

def weight\_variable(shape):

initial = tf.truncated\_normal(shape, stddev=0.1)

return tf.Variable(initial)

def bias\_variable(shape):

initial = tf.constant(0.1, shape=shape)

return tf.Variable(initial)

def conv2d(x, W):

return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max\_pool\_2x2(x):

return tf.nn.max\_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')

#convolution definitions

W\_conv1 = weight\_variable([5, 5, 1, 32])

b\_conv1 = bias\_variable([32])

x\_board = tf.reshape(x, [-1,9,9,1])#USE THE IMPORTANT VALUES

h\_conv1 = tf.nn.relu(conv2d(x\_board, W\_conv1) + b\_conv1)

h\_pool1 = max\_pool\_2x2(h\_conv1)

W\_conv2 = weight\_variable([5, 5, 32, 64])

b\_conv2 = bias\_variable([64])

h\_conv2 = tf.nn.relu(conv2d(h\_pool1, W\_conv2) + b\_conv2)

h\_pool2 = max\_pool\_2x2(h\_conv2)

W\_fc1 = weight\_variable([3 \* 3 \* 64, 1024])

b\_fc1 = bias\_variable([1024])

h\_pool2\_flat = tf.reshape(h\_pool2, [-1, 3\*3\*64])

h\_fc1 = tf.nn.relu(tf.matmul(h\_pool2\_flat, W\_fc1) + b\_fc1)

keep\_prob = tf.placeholder("float")

h\_fc1\_drop = tf.nn.dropout(h\_fc1, keep\_prob)

W\_fc2 = weight\_variable([1024, 1])

b\_fc2 = bias\_variable([1])

y\_conv=tf.nn.softmax(tf.matmul(h\_fc1\_drop, W\_fc2) + b\_fc2)

cross\_entropy = -tf.reduce\_sum(y\_\*tf.log(y\_conv))

train\_step = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy)

correct\_prediction = tf.equal(tf.argmax(y\_conv,1), tf.argmax(y\_,1))

accuracy = tf.reduce\_mean(tf.cast(correct\_prediction, "float"))

sess.run(tf.initialize\_all\_variables())

for i in range(200):

batch = goData.train.next\_batch(50)

if i%100 == 0:

print i

train\_step.run(feed\_dict={x: batch[0], y\_: batch[1], keep\_prob: 0.5})

#Returns predicted move and board given the board layout

tensArr= W.eval(sess).transpose()

consArr= b.eval(sess)

boardArr= np.array([0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0,

0,0,0,0,0,0,0,0,0])#1= black

temp=[]

for i in range(0,len(boardArr)):

temp.append(np.float32(boardArr[i]))

boardArr= temp

highestOutcome= 0

maxMove= -1

for count in range(0,81):#

if(boardArr[count]==0):

boardArr[count]= np.float32(1.0) #change

x\_board = tf.reshape(boardArr, [-1,9,9,1])

h\_conv1 = tf.nn.relu(conv2d(x\_board, W\_conv1) + b\_conv1)

h\_pool1 = max\_pool\_2x2(h\_conv1)

h\_conv2 = tf.nn.relu(conv2d(h\_pool1, W\_conv2) + b\_conv2)

h\_pool2 = max\_pool\_2x2(h\_conv2)

h\_pool2\_flat = tf.reshape(h\_pool2, [-1, 3\*3\*64])

h\_fc1 = tf.nn.softmax(tf.matmul(h\_pool2\_flat, W\_fc1) + b\_fc1)

temp= (tf.nn.softmax(tf.matmul(h\_fc1, W\_fc2) + b\_fc2))

if(temp<highestOutcome):#change to less

highestOutcome= temp

maxMove= count

boardArr[count]= 0

print(maxMove)

print(highestOutcome)

boardArr[maxMove]= 1 #change

print(boardArr[0:9])

print(boardArr[9:18])

print(boardArr[18:27])

print(boardArr[27:36])

print(boardArr[36:45])

print(boardArr[45:54])

print(boardArr[54:63])

print(boardArr[63:72])

print(boardArr[72:81])

lowestOutcome= 0

minMove= -1

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

# LIST OF REFERENCES

Abadi, Martin, Agarwal, Ashish, Barham, Paul, et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. 2015. http://download.tensorflow.org/paper/whitepaper2015.pdf

Allis, Victor. *Searching for Solutions in Games and Artificial Intelligence*. 1994. Web.

Ball, Phillip. "Game Theorists Crack Poker." *Nature.com*. Nature Publishing Group, 08 Jan. 2015. Web. 01 Apr. 2016. http://www.nature.com/news/game-theorists-crack-poker-1.16683

Clark, Christopher, and Amos Storkey. "Teaching Deep Convolutional Neural Networks to Play Go." *ArXiv* (2015) Cornell University, 10 Dec. 2014. Web. http://arxiv.org/pdf/1412.3409v2.pdf

"Darren's Computer Go Pages: 9x9 Go Experiments: File Formats."

Lasker, Edward. *Go and Go-moku, the Oriental Board Games*. New York: Dover Publications, 1960. Print.

Maddison, Chris J., Aja Huang, Ilya Sutskever, and David Silver. "Move Evaluation in Go Using Deep Convolutional Neural Networks." 10 Apr. 2015. <http://arxiv.org/pdf/1412.6564v2.pdf>

Merkel, Dirk. Docker: Lig.htweight Linux Containers for Consistent Development and Deployment. Linux Journal. 2014. Print.

Russell, Stuart Jonathan., and Peter Norvig. *Artificial Intelligence a Modern Approach*. Harlow: Pearson Education Limited, 2014. Print.

Silver, David, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. "Mastering the Game of Go with Deep Neural Networks and Tree Search." *Nature* 529.7587 (2016): 484-89. Web. <http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html>

"The Japanese Rules of Go." *Article 1. The Game of Go*. Carnegie Mellon University. Web. 01 Apr. 2016

1. Russell, Stuart Jonathan., and Peter Norvig. Artificial Intelligence a Modern Approach. Harlow: Pearson Education Limited, 2014. Print. 1. [↑](#footnote-ref-1)
2. Russell, Stuart Jonathan., and Peter Norvig. Artificial Intelligence a Modern Approach. Harlow: Pearson Education Limited, 2014. Print. 2. [↑](#footnote-ref-2)
3. Allis, Victor. Searching for Solutions in Games and Artificial Intelligence. 1994. Web. [↑](#footnote-ref-3)
4. Ball, Phillip. "Game Theorists Crack Poker." Nature.com. Nature Publishing Group, 08 Jan. 2015. Web. 01 Apr. 2016. [↑](#footnote-ref-4)
5. Russell, Stuart Jonathan., and Peter Norvig. *Artificial Intelligence a Modern Approach*. Harlow: Pearson Education Limited, 2014. Print. 2. [↑](#footnote-ref-5)
6. Silver, David, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. "Mastering the Game of Go with Deep Neural Networks and Tree Search." Nature 529.7587 (2016): 484-89. Web. [↑](#footnote-ref-6)
7. Russell, Stuart Jonathan., and Peter Norvig. Artificial Intelligence a Modern Approach. Harlow: Pearson Education Limited, 2014. Print. xi. [↑](#footnote-ref-7)
8. Maddison, Chris J., Aja Huang, Ilya Sutskever, and David Silver. "Move Evaluation in Go Using Deep Convolutional Neural Networks." 10 Apr. 2015. [↑](#footnote-ref-8)
9. Lasker, Edward. Go and Go-moku, the Oriental Board Games. New York: Dover Publications, 1960. Print. 99. [↑](#footnote-ref-9)
10. Russell, Stuart Jonathan., and Peter Norvig. Artificial Intelligence a Modern Approach. Harlow: Pearson Education Limited, 2014. Print. 696. [↑](#footnote-ref-10)
11. Ibid. [↑](#footnote-ref-13)
12. Russell, Stuart Jonathan., and Peter Norvig. Artificial Intelligence a Modern Approach. Harlow: Pearson Education Limited, 2014. Print. 732. [↑](#footnote-ref-14)
13. Maddison, Chris J., Aja Huang, Ilya Sutskever, and David Silver. "Move Evaluation in Go Using Deep Convolutional Neural Networks." 10 Apr. 2015. [↑](#footnote-ref-15)
14. Ibid. [↑](#footnote-ref-16)
15. Russell, Stuart Jonathan., and Peter Norvig. Artificial Intelligence a Modern Approach. Harlow: Pearson Education Limited, 2014. Print. 929 [↑](#footnote-ref-17)
16. Silver, David, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, et al. "Mastering the Game of Go with Deep Neural Networks and Tree Search." Nature 529.7587 (2016): 484-89. Web. [↑](#footnote-ref-18)
17. Russell, Stuart Jonathan., and Peter Norvig. Artificial Intelligence a Modern Approach. Harlow: Pearson Education Limited, 2014. Print. 183 [↑](#footnote-ref-19)
18. Ibid. [↑](#footnote-ref-20)
19. Russell, Stuart Jonathan., and Peter Norvig. Artificial Intelligence a Modern Approach. Harlow: Pearson Education Limited, 2014. Print. 21. [↑](#footnote-ref-21)
20. Russell, Stuart Jonathan., and Peter Norvig. Artificial Intelligence a Modern Approach. Harlow: Pearson Education Limited, 2014. Print. 129. [↑](#footnote-ref-22)
21. Silver, David, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, et al. "Mastering the Game of Go with Deep Neural Networks and Tree Search." Nature 529.7587 (2016): 484-89. Web. [↑](#footnote-ref-23)
22. "Darren's Computer Go Pages: 9x9 Go Experiments: File Formats." [↑](#footnote-ref-24)
23. " Abadi, Martin, Agarwal, Ashish, Barham, Paul, et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. 2015. [↑](#footnote-ref-25)
24. Merkel, Dirk. Docker: Lig.htweight Linux Containers for Consistent Development and Deployment. Linux Journal. 2014. Print. [↑](#footnote-ref-26)