Exploring Winner Predictions and Probabilities in Real-Time Strategy Games via Neural Networks

A thesis presented

by

James Chung and Andrew Jin

in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science

Advisor: Professor James Glenn

Yale University
May 2019
Abstract

Strategy games have served as critical platforms to assess the performance of artificial intelligence systems. As machine learning techniques have improved, researchers have shifted their attention from traditional turn-based strategy games to increasingly complicated real-time strategy games (RTS) such as StarCraft II, a computer game which has daunted the artificial intelligence community due to the game’s immense observation and action space. Much of the existing research endeavors in RTS games, however, have been concerned with the implementation of artificial intelligence learning agents to competitively play against human players. Very little work has been done to compute win-probabilities of participants in live real-time strategy games.

This project aims to bridge this gap by training neural networks via supervised learning in order to predict the win-probability of a given player during a competitive 1v1 RTS game. We begin this study by developing categorical and probabilistic neural networks for microRTS, a simplified Java implementation of a RTS game. Then, we build upon the framework of our microRTS models to construct similar, but more complicated, predictive classifiers for StarCraft II. We find that despite the simplifications to our microRTS and StarCraft II neural networks, our models manage to predict the winning player with relatively high accuracy and high calibration. Additionally, we demonstrate that the predictive capacities of our StarCraft II neural networks vary based upon the skill-disparity and the baseline skill-levels of players in the match. We conclude with several recommendations on what features can be introduced to our models to further improve prediction accuracies. While there is room for improvement in our neural networks, our models may serve as valuable preliminary designs for applications in electronic-sports betting and for visual tools in professional game-casting.
Table of Contents

1 Introduction 3

2 Background 4
   2.1 microRTS Game Mechanics ........................................... 4
   2.2 StarCraft II Game Mechanics ........................................ 4

3 microRTS 5
   3.1 Data Collection Procedure ........................................... 5
   3.2 Binary Classification Neural Networks ............................. 5
      3.2.1 Simple Neural Network ...................................... 5
      3.2.2 Army Composition Neural Network ............................ 6
   3.3 Probabilistic Neural Networks ..................................... 7
      3.3.1 Training Bias Inherent in Labeling Methods ................ 7
      3.3.2 Time-Based Probability Model ............................... 7
      3.3.3 History-Based Probability Model ............................ 8
   3.4 Evaluating Model Accuracies ..................................... 9

4 StarCraft II 11
   4.1 Data Collection Procedure ....................................... 11
   4.2 Bias within StarCraft II Replays ................................ 12
   4.3 StarCraft II Neural Networks .................................... 12
      4.3.1 Simple StarCraft II Neural Network (NN 1.0) ............ 12
      4.3.2 Game Loop, MMR, and Race Neural Network (NN 1.1) .......... 13
      4.3.3 NN 1.1 Performance on Large MMR Differential Data Set ...... 14
      4.3.4 NN 1.1 Performance on Data Set Controlled for Skil League .... 15
   4.4 Evaluating Model Accuracies ..................................... 16

5 Conclusion and Future Work 17

6 Acknowledgements 17
1 Introduction

Substantial artificial intelligence research has been conducted to train computer programs via supervised learning to replicate the decision-making abilities of humans.\textsuperscript{1−3} Specifically, machine learning techniques have been applied and developed in a multitude of strategy games, resulting in computer programs that have mastered several turn-based games such as Checkers, Chess, and Go.\textsuperscript{4−6}

For decades, games have served as critical platforms to assess the performance of artificial intelligence systems, and researchers have been working assiduously to tackle increasingly complex games. \textit{StarCraft II}, a real-time strategy (RTS) video game, has widely been considered by computer scientists as one of the most difficult challenges in the computational intelligence of games. Before 2017, however, even the best \textit{StarCraft} bots could be defeated by amateur players.\textsuperscript{7} And before AlphaStar, the best \textit{StarCraft II} AI agents were trained to beat other difficult AI agents, not professional human players.\textsuperscript{8}

According to DeepMind, “tackling full versions of RTS games has seemed daunting because of the rich input and output spaces as well as the very sparse reward structure.”\textsuperscript{9} As a result, much of the early \textit{StarCraft II} research revolved around specific mini games that reduced the large action space of the full game. Mini game agents such as Deepmind’s \textit{DefeatRoaches} showed that agents could learn to micro-manage marines to defeat a group of enemy roaches. Through DeepMind’s learning agent, AlphaStar, it was only until very recently (January 24, 2019) that computer scientists were able to successfully construct a program that competitively challenged professional \textit{StarCraft II} players.\textsuperscript{10}

Even AlphaStar, however, has its limitations. AlphaStar is restricted to playing one specific race match-up (Protoss vs. Protoss) and is unable to play the other eight possible race match-ups. Additionally, AlphaStar can only play on one specific map (“Catalyst LE”) on one patch of the game (version 4.6.2). Nevertheless, the creation of AlphaStar was a major breakthrough in artificial intelligence, and the research community is working to broaden AlphaStar’s capacity so that it can eventually play \textit{StarCraft II} competitively without restrictions.

While many \textit{StarCraft II} research endeavors have revolved around the implementation of learning agents to competitively play against human players, little work has been done on evaluating winning probabilities during live real-time strategy matches. Thus, this project aims to bridge this gap by training a neural network via supervised learning to predict the win probability of a given player during a competitive 1v1 real-time strategy match. We begin with a simplified approach of this problem by developing such a neural network for \textit{microRTS}, a simplified RTS game similar to \textit{StarCraft II}. Later, we extend the framework of our \textit{microRTS} models to construct similar but more complicated neural networks for \textit{StarCraft II}.

The e-sports industry is growing at an explosive rate, and with competitive professional matches comes pre- and live- game betting. The ability to predict the outcomes of matches between players has become increasingly important, and this project may shed light on how to quantify likelihoods of victory for e-sports participants. Additionally, by training an AI agent to predict win probability rather than to competitively play the RTS game, we eliminate some of the problems that negatively affect existing game-playing bots. In terms of the observation space, the learning agent can observe all features of the game and is not hindered by the “fog of war” game mechanic that is typical of RTS games. Furthermore, our agent does not need to generate in-game decisions in response to game-states; rather, our agent is tasked with synthesizing information in various states to estimate the extent of a player’s advantage.
2 Background

2.1 microRTS Game Mechanics

*microRTS* is a Java implementation of a simplified RTS game.

At the start of each match, both players begin with one base and five workers, units that can harvest resources that are located around the game map. The game map is a grid composed of 1x1 tiles in which players can place buildings and units.

There is only one type of resource in *microRTS*, and players can spend their resources to build barracks and additional workers. Barracks are structures that can build three types of army units once completed: heavy, light, and ranged units. The different units have varying levels of power, attack range, and speed. The objective of each game is simple: allocate resources efficiently to build an army to defeat the opponent’s base.

By building more workers during the early stages of the game, a player can boost their resource collection rate. However, this forgoes the opportunity to build offensive units, which leaves a player vulnerable to attacks from the enemy. The challenge for each player is to maximize their resource collection-rate while maintaining a sufficiently-sized army to ward off enemy attacks.

2.2 StarCraft II Game Mechanics

*StarCraft II* is the sequel to *StarCraft*, a timeless real-time strategy game that takes place in a science-fiction universe ruled by three races: Protoss, Terran, and Zerg. While *StarCraft II* can be played by more than two players, we limit our focus to competitive 1v1 games. The goal of each game is the same as in *microRTS*: allocate resources efficiently to build an army to destroy the opponent’s base.

*StarCraft II* has a built-in matchmaking system that attempts to pair players of similar skill levels. Every player has an MMR, or match-making ranking, a number that increases when players win and decreases when players lose. *StarCraft II*’s matchmaking system pairs players with similar MMRs to face off against one another. The extent to which a player’s MMR changes after a match depends on the MMR of their opponent. The MMRs of new players hover around 2000 while the MMRs of professional players frequently exceed 6000.

At the start of each game, both players start out with one base and twelve workers. In *StarCraft II*, there are two types of resources: minerals and vespene gas. Once harvested, resources can be spent on army units, buildings, additional workers, and upgrades. The types of structures, and hence units, that a player is allowed to build depends on their selected race.

*StarCraft II* also includes a game mechanic known as “fog of war,” which limits the vision of each player to a small radius around each of their units and buildings. As a result, each player has limited information about their opponent’s strategy, including their opponent’s allocation of resources, location of army, and barracks count. Consequently, players cannot simply rely on one strategy to achieve victory. Instead, players must routinely send some of their units to scout the performed actions of their opponents and adjust their own actions accordingly.

The keyboard and the mouse is the portal between players’ minds and units. Since the game is played in real time, speed and effective multitasking abilities are imperative. This is especially true late in the game, when players must control hundreds of units and dozens of buildings. Professional players average up to 300 to 400 actions per minute.
3 microRTS

3.1 Data Collection Procedure

In order to gather a sufficiently-sized data set of matches to train the microRTS neural networks, we acquired 7,700 replay files from the 2018 microRTS AI competition. All of these matches are between various computational AI learning agents and do not involve human-play. Of the 7,700 matches in the 2018 competition, 1,088 ended without a winner since the rules of the competition set a limit on the number of game cycles allowed per match. We eliminate these from our data set.

Each replay file contains game-state information approximately every 10 game cycles. There are 10 game cycles per second. At each time step, information is available regarding a player’s resources, unit composition and count, base count, and worker count. Game-state snapshots are recorded more frequently as the game progresses, so for each replay, more data is available for latter parts of the game.

The raw replay files are quite difficult to parse efficiently, so we implemented a parser to create readable lists of game-states for each replay. Code for the various parsers can be found in microrts_simple_data.py and microrts_unit_data.py.

Since longer-duration games contain more game cycles, there are more game-state data points for longer matches. To avoid over-training our models on longer games, for each neural network, we take a fixed number of random game-states from any given replay to add to our data set. Taking a set amount of data points from each replay prevents our model from learning less from shorter-duration games.

3.2 Binary Classification Neural Networks

3.2.1 Simple Neural Network

The simple neural network takes in ten features. We input five features of in-game attributes for each player into the neural network:

\[
[\text{resources}, \text{base count}, \text{barrack count}, \text{worker count}, \text{army count}]\]

Features are normalized to continuous values ranging from 0 to 1, inclusive. The labels of the neural network are either 0 or 1, indicating the winner of the match.

From each of replays from the 2018 microRTS AI competition, we collected three random game-states to generate 19,836 game-state data points. 80 percent of the data set was used for training a sequential neural network consisting of 100 nodes and one-hidden layer. The implementation was done using keras. The remaining 20 percent of the data set was used to test the network’s accuracy. Figure 1 shows the performance accuracy of this simple microRTS network.

---

1My project partner, Andrew Jin, designed and implemented the majority of sections 3.1-3.3. Please see his paper for more specific details regarding these sections of our microRTS models.

2More information about the microRTS AI competition and how replays are generated can be found at https://sites.google.com/site/micrortsaicompetition/rules

3The source code for all of the categorical neural networks for microRTS can be found in nnv1.py
Figure 1: **Accuracy of Simple microRTS Neural Network.** Prediction accuracy is plotted as a function of the time ratio ($t$). To generate prediction accuracies of the model, data points are bucketed into deciles based off of their time ratios. For each bucket, the plotted value of the regressor is the average of the bucket’s time ratio range, and the plotted value of the accuracy is the percentage of accurate predictions for observations that fall in the bucket’s time ratio range. During the first decile ($t \leq 0.1$), the network achieves an accuracy of 53.3 percent. By the sixth decile ($t \geq 0.6$), the network achieves greater than 87.5 percent accuracy.

The results from the simple neural network are consistent with our initial expectations. At the start of any given match, both players begin with the same amount of workers and bases. Hence, during the beginning stages of the game, no player has a distinct advantage. As a consequence, we should expect the predictions of the network to be essentially guessing the winner, with an expected accuracy of around 50 percent. We see that this is the case as the simple network achieves an accuracy of around 53 percent during the first time ratio decile.

As games progress, their action spaces increase exponentially, making it more likely for players to diverge in strategies and resource allocation. Thus, the additional information available in higher time-ratio game-states should lead to our model achieving better winner prediction accuracy as games develop. This seems to be the case as the prediction accuracy increases as a function of the time ratio until the accuracy plateaus at approximately 98 percent.

### 3.2.2 Army Composition Neural Network

We extend the initial framework provided by the simple neural network to incorporate army composition. Recall that in microRTS there are three different types of units: heavy, light, and ranged. Each of the unit types are associated with unique properties that could be advantageous against one type of unit but disadvantageous against another. Consequently, by accounting for the specific army composition in the neural network, we anticipate that the prediction accuracy of the network will improve during the middle stages of the game to account for which player has the more dominant makeup of army units.

The army composition neural network takes in fourteen features. Like the simple neural network, features of this network are normalized to continuous values ranging from 0 to 1, and the labels of the network are binary values to indicate the winner of the match. There are seven features of in-game attributes for each player in the network:

```
[ resources, bases, barracks, workers, light units, heavy units, ranged units ]
```

The methodology to generate game-state data points for the training and testing of the neural network was the same as in 3.2.1.
Figure 2: Adding Army Composition Improves Accuracy of microRTS Neural Network. Prediction accuracies are plotted as a function of the time ratio ($t$) for the army composition model (yellow) and the simple model (blue). Noticeable improvements can be seen in the army composition neural network when $0.2 \leq t \leq 0.6$ (an average improvement of 5.1 percent in prediction accuracy during this time ratio bound).

Figure 2 displays the prediction accuracy of both the army composition model and the simple model. As hypothesized, accounting for army composition significantly improves the accuracy of the model during the middle stages of the game. In the first decile, the army composition neural network is marginally more accurate than the simple network, achieving 57 percent accuracy. But, as players begin to create more army units, the network seems to be able to distinguish which player has the more advantageous composition of units: prediction accuracy noticeably improves when $0.2 \leq t \leq 0.6$.

In the last quarter of the game, the accuracy of both models converges around 98 percent. We presume that this convergence occurs because during the late stages of a match, the winning player has already destroyed a significant chunk of the opponent’s army and is simply in the process of destroying the opponent’s remaining structures. In that case, basic features such as army count is enough to accurately determine the winning player.

3.3 Probabilistic Neural Networks

3.3.1 Training Bias Inherent in Labeling Methods

One drawback with the categorical neural networks in 3.2.1 and 3.2.2 is that every game-state in our data set used to conduct supervised learning is labeled either 0 or 1. This method of supervised learning labels every game-state with the winning player, $w$, regardless of the real probability of $w$ achieving victory at that current game-state.

Consider for example a game in which player 1 is losing for the first half of the match, but makes a comeback to eventually beat player 0. An ideal model would learn to label the first half of the game-state data points as states in which player 0 is likely to win. However, the classification procedure used in 3.2.1 and 3.2.2 labels all of the game-states in our example with player 1 as the winner. This would interfere with the training of our neural network because our labeling methods suggest that player 1 was winning throughout the entire duration of the game.

The problem is that it is difficult to verify the “true probability” of winning at any particular game-state. While we can create a classification model to output probability predictions regarding which player will win, there is no method of verifying the accuracy of these probability predictions. In the following sections, we explore various techniques to estimate this probability in order to address the problems associated with our previous labeling methods.

3.3.2 Time-Based Probability Model

In order to address the issue of over-weighting beginning game-states, we change our labeling methods of game-state observations to incorporate the time-ratio of the observations. Specifically, instead of using binary classification techniques to label each observation as 0 or 1, we label each game-frame
using the following piece-wise function:

\[
    f(s, p) = \begin{cases} 
        0.5 - 0.5t_s & p = 0 \\
        0.5 + 0.5t_s & p = 1 
    \end{cases}
\]

where \( s \) is the game-state, \( p \) is the player who wins the match, and \( t_s \) is the time-ratio at game-state \( s \).

We then change the time-based probability neural network so that it is no longer a binary classifier. By setting the activation function of the model’s output layer to the sigmoid function, the time-based probability neural network outputs a continuous value between 0 and 1, inclusive. We then use the mean-squared error as our loss function and use the Adam optimizer. Similar to the binary classification neural networks, the time-based probability model consists of 100 nodes and one hidden layer and is implemented with keras.\(^iv\)

While the time-based probability model addresses the over-weighting of early game-states, the model assumes that the winning player’s probability of victory increases linearly throughout the course of the match. However, for many games, it is unlikely that the likelihood of victory increases exactly in a linear fashion. We only use the linear assumption to underweight beginning game-states since both players start out with an equal probability of winning the game. As a consequence, this method of pre-processing data labels still does not address games in which the winner makes a comeback.

3.3.3 History-Based Probability Model

To address the inadequacies of the previous models to account for games in which the winner makes a comeback, we construct a history-based probability neural network. For this network, we label a game-state \( g \) with a continuous probability value of player 1 winning \( p_1 \) such that:

\[
    p_1 = \frac{w_{p1}}{n}
\]

where \( n \) is the number of game-states that are comparable to \( g \), and \( w_{p1} \) is the number of games player 1 wins in the set of games that are comparable to \( g \).\(^v\)

After pre-processing game-state observations with historical probabilities, we train a probabilistic neural network with the same network architecture as the time-based probability model described in 3.3.2.\(^vi\) For both the time-based and history-based probabilistic neural networks, we round the probability outputs to the nearest integer to serve as the model’s categorical prediction of the winning player.

![Figure 3: Prediction Accuracies of Probabilistic Neural Networks for microRTS. Prediction accuracies are plotted as a function of the time ratio (t) for the army composition model (dark blue), the history-based probability model (yellow), and the time-based probability model (light blue).](image)

Figure 3 compares the prediction accuracies of the army composition model, history-based probability model, and time-based probability model. Surprisingly, the binary classification army composition

\(^{iv}\)The source code for the time-based probabilistic neural network can be found in nnv2.py

\(^{v}\)The specific procedure and metrics for categorizing buckets of comparable game-states are described in the final report of my project partner, Andrew Jin.

\(^{vi}\)Source code for the history-based probability model can be accessed at nnv3.py
model performed the best among the three models. The time-based probabilistic neural network lagged behind the other two models at the early-stages of games ($t \leq 0.4$). We theorize that the low-weighting of early-stage data points is under-weighting the importance of early battle engagements and “all-in” build-orders that rely on committing resources to building army units early in the game to surprise the opponent. The underperformance of the history-based probabilistic neural network may have been due to a lack of enough data to categorize enough game-states as comparable.

Note, however, that winner prediction accuracy is not the only metric to assess the performance of our neural networks by. We can also measure the performance of our \textit{microRTS} neural networks based on the mean-squared error (MSE) of their predictions to various proxy values for the “true probabilities” of game-states.

One such proxy for the true probability of game-states is the historical probability of comparable game-states. Another such proxy for this true probability is the time-probability. Figures 4 and 5 graph the MSEs of the \textit{microRTS} neural networks on both proxies for true probabilities. From the plots, it is obvious why the time-based probability model has the lowest MSE when compared to the actual time probabilities and why the history-based probability model has the lowest MSE when compared to the actual historical probabilities.

It is interesting to note, however, that for both proxy metrics of true probability, the MSE increases for all models as the time ratio of game-state observations approach 0.5. This occurrence implies that neural networks find it more difficult to accurately predict the probabilities in the middle of matches compared to the tail-ends of matches. This result makes sense — at the beginning of matches, both players start out with equal probabilities of winning; at the end of matches, one player is usually significantly dominating the other.

![Figure 4: Mean-Squared Error of \textit{microRTS} Models on Time-Adjusted Probabilities.](image)

We plot the mean-squared error (MSE) of the time-ratio adjusted probability and the probabilities predicted by the army composition model, history-based model, and time-based model.

![Figure 5: Mean-Squared Error of \textit{microRTS} Models on Historical Probabilities.](image)

We plot the mean-squared error (MSE) of the historical probability and the probabilities predicted by the army composition model, history-based model, and time-based model.

### 3.4 Evaluating Model Accuracies

Although we have examined the prediction accuracy of our \textit{microRTS} neural networks as a function of the time ratio of a game, we have yet to assess how accurate the value of the probability predictions of
our models are. Recall that for any given game-state observation, the classification models (the simple neural network and the army composition neural network) generate their own probabilities of player 1 winning the match. The models then use this probability to predict the winner. In this section, we attempt to evaluate the accuracy of the predicted probabilities of both the classification and probabilistic neural networks by measuring the calibration of both neural networks.

The predicted probability output of a well-calibrated model can be directly interpreted as a confidence level. For example, a well-calibrated binary classifier should classify game-state observations such that among the samples which it computed a predicted win probability of 20 percent for player 1, player 1 won 20 percent of the time. In order to quantify the percent correct classification of the microRTS models, we bucket network probability predictions into deciles. For each bucket, we calculate the actual win percentage of player 1 and plot this ratio as a function of the model’s predicted probability of player 1 winning. This plot is referred to as a reliability curve. The reliability curve of a perfectly calibrated model is the identity function. Hence, deviations from the identity function indicate a poorly-calibrated classifier.

Figure 6 displays the reliability curves of the microRTS neural networks. From the plot, the army composition neural network appears to have the best calibration among the three models. The MSEs of the history-based, time-based, and army-composition models compared to the identity function are 0.012826, 0.021781, and 0.003191, respectively.

Figure 6: Reliability Curves of microRTS Neural Networks. We plot the actual win percentage of player 1 (from the same game-state observations in the test set) as a function of the model-predicted probabilities of player 1 winning. We also plot the hypothetical performance of a perfectly accurate model as a comparison.

While the army composition neural network fashions a linear reliability curve, both probabilistic models cast sigmoid-like reliability curves, indicating underconfidence in these models compared to the categorical network. To illustrate, Figure 6 suggests that player 1 wins nearly all of the game-state observations that the time-based neural network predicts player 1 having an 80 percent chance of winning. In the same manner, player 0 wins nearly all of the game-state observations that the time-based model predicts player 0 having an 80 percent chance of winning.

In hindsight, the lower calibration of the probabilistic neural networks makes sense. In order to train the history-based and time-based models, we labeled our training data with our own approximations of the true probabilities of game-states. Based on the lower calibration of the probabilistic models, it is likely that the assumptions we made in order to estimate these true probabilities were flawed. Nonetheless, the symmetry of the calibration curves of both regression models makes us optimistic about our ability to recalibrate the probabilistic models.
4 StarCraft II

4.1 Data Collection Procedure

In order to create a neural network to predict the win probabilities of StarCraft II matches, it was necessary to acquire a sufficiently-sized data set of matches with which we could train our model. We collected 10,000 replays of 1v1 ladder matches on version 4.8.3 of StarCraft II using Blizzard’s API, sc2client-proto.12

Out of the 10,000 replays in our data set, 8,466 replays were matches that were longer than two minutes. The fastest attacks in StarCraft II, also known as “rushes,” typically take around two to three minutes to execute, so what accounts for these exceptionally short games? Occasionally, players will begin a ladder match and immediately surrender the game (presumably to lower their MMR so that they do not need to face off against more difficult opponents in the future). We assumed that the majority of replays under two-minutes were examples of such behavior and removed them from our data set. This seems to be a fair assumption since the average duration for the 1,534 removed games was only 12 seconds long.

To pre-process information for each replay, we modified the implementation of pysc2-replay,13 a framework built on top of DeepMind’s StarCraft II api, pysc2.14 All of the source code required for the pre-processing of information can be accessed in the submitted pysc2-replay-master directory. By construction, a StarCraft II replay file is composed of a list of player actions taken throughout the course of the game. A limitation that stems from this structural design is that replay observers cannot simply skip to certain segments of the match to instantly fetch the corresponding game state. Instead, the game state must be constructed action-by-action, synthesizing all of the prior game activity to reach the desired game frame.

pysc2-replay loads and plays a replay in the StarCraft II application in order to construct game states. Since pysc2-replay is dependent on the underlying StarCraft II application, it is necessary to ensure that the version of the StarCraft II application matches the version of the replay under analysis. For the purposes of this project, we limit our focus to version 4.8.3. Additionally, to load a replay on the StarCraft II application using pysc2, it is necessary to have the unique cache file that is associated with the specific game map. Cache files can only be generated manually by loading a replay on the actual StarCraft II application. Hence, to avoid the generation of multiple cache files (since there are countless maps), we limit our data set to 2019 Season 1 ladder games. All of these games took place on one of the seven approved ladder maps: Automaton, Cyber Forest, Kairos Junction, King’s Cove, New Repugnancy, Port Aleksander, and Year Zero.

Furthermore, by design, the pysc2 library limits the game controller to only accept a single observed player id, which prevents computer programs from observing the actions of both players in the same time step. Consequently, to scrape data from each replay, we iterate through each replay twice to gather information regarding each player at the same time step. For each replay, we record information about the game-state every 1000 time-steps, which is equivalent to about 45 seconds.

To speed up the process of data collection, we used the multiprocessing python package to run six separate pools to process replays in parallel.1vii However, a consequence of running multiple threads to process StarCraft II replays was that output files occasionally failed to record the time-step observation due to errors that stemmed from running multiple instances of the StarCraft II application. Due to time and resource constraints, it was impractical to iteratively process each replay. After removing corrupt and inconsistent output files, we were left with 4,304 output files that correctly captured the game-state information of both players in a match.

viiThe source code for data collection can be found at ScrapeSC2replays.py, transform_replay.py, ObserverAgent.py, combineReplayTxs.py, combineReplayTxs_all.py, merge_replays.py, and formsc2data.py
At each time step, we record a 43-element list of in-game attributes for each player. Some of the elements included in this list include: minerals, vespene gas, supply, supply cap, army supply, worker supply, idle worker count, total number of units, warp gate count, larva count, Blizzard score, idle production time, idle worker time, total value of units, total value of structures, killed value of units, killed value of structures, total collected minerals, total collected vespene gas, mineral collection rate, vespene gas collection rate, resources used on technology, and resources used on upgrades. Due to computational resource limitations, none of the elements in the list include the geographical positioning of units or army composition, which would be valuable inputs to add for extensions of this project.

4.2 Bias within StarCraft II Replays

There is a notable difference between the replays used to train the microRTS neural networks and the StarCraft II neural networks. Every microRTS replay in our data set involves matches between various computational AI agents. By design, these learning agents do not surrender in matches, which means every game must be played to completion. As such, for a player to win in microRTS, they must necessarily destroy all of their opponent’s buildings.

In StarCraft II, however, all of the replays in our data set are real games played by humans on Blizzard’s competition ladder. As a result, games are rarely ever played to completion. Rather, a player surrenders his match when he concludes that there is no chance for him to come back to beat his opponent. Therefore, there is an inherent bias within our StarCraft II replay data: in any given match, both players continue to play in a match only when they perceive they have a chance at winning the game. Accordingly, in higher-level matches, in which players have a more developed understanding of the game, players are often relatively even with one another for most of the match, until one large engagement dictates the winner. This should lead to less accurate predictions when StarCraft II neural networks try to evaluate games that involve higher-skilled players.

In general, since all StarCraft II players are likely to surrender before their opponents destroy all of their structures, a larger proportion of the early predictions computed by our StarCraft II models should hover around 50 percent. Towards the end of matches, there should also be a shorter plateau period of high prediction accuracy.

4.3 StarCraft II Neural Networks

4.3.1 Simple StarCraft II Neural Network (NN 1.0)

We begin with a simple neural network that takes in 86 features as input. We refer to this model as NN 1.0. 43 features are designated for each player and these features are the same elements that were collected in the 43-element list described in 4.1. Features were normalized to continuous values ranging from 0 to 1, inclusive. The labels of the neural network were either 0 or 1, indicating the winner of the match.

To generate a data set of game-state information, four time-step observations were randomly selected from each of the consistent replay output files. This resulted in a data set composed of 17,216 game-state data points. 80 percent of this data set was used for training a sequential neural network consisting of two-hidden layers and 800 nodes. The output layer of the network consisted of two nodes (one for each player), and the network’s output was determined by rounding the model’s predicted probability of player 1 winning the match. The remaining 20 percent of the data set was used to test the network’s accuracy. The implementation was done using keras.viii

---

viiiThe source code for all of the categorical neural networks for StarCraft II can be found in sc2nnv1.py
Figure 7: Accuracy of Simple StarCraft II Neural Network (NN 1.0). Prediction accuracy is plotted as a function of the time ratio ($t$). During the first decile ($t \leq 0.1$), NN 1.0 achieves an accuracy of 50.7 percent. By the seventh decile ($t \geq 0.7$), the network achieves greater than 74.0 percent accuracy.

Figure 7 plots the prediction accuracy of NN 1.0 as a function of the time ratio, $t$. The data is consistent with our initial hypotheses — the simple StarCraft II network performs worse in early stages of matches and lacks a plateau period of high prediction accuracy towards the end of matches.

The percentage accuracies for the first three deciles were 50.6, 49.2, and 54.1, respectively. As anticipated, the StarCraft II neural network does a worse job than the microRTS neural networks at predicting the winner during the early stages of games: for the first two quartiles, the StarCraft II model is essentially randomly guessing the winner, and the third quartile is barely better than random prediction.

The prediction accuracy of the model improves in a linear fashion for the next eight deciles ($0.2 \leq t \leq 0.9$), increasing by approximately 4.27 percent every decile. In the last decile, however, accuracy improves by a dramatic 9.31 percent, which is more than double the linear step found within the second to ninth deciles. This finding suggests that between the ninth and tenth decile, many games experience significant changes to some of the values in the neural network’s feature vector, which allows the model to significantly improve its prediction of the winning player. We surmise that these substantial changes in feature values stem from the aftermath of full-scale battle engagements that frequently result in one of the players surrendering the game because they have no more units.

4.3.2 Game Loop, MMR, and Race Neural Network (NN 1.1)

We extend the initial framework afforded by NN 1.0 to account for the game-state’s current time loop, player MMRs, and player races. We refer to this modified neural network as NN 1.1.

In StarCraft II, strategies are dependent on how much time has elapsed in a match. In early-stages of a game, having a few more army units than the opponent may be enough for a player to win. In late-stages of a game, the map may run out of resources, which makes a player’s existing resources and units more valuable. By accounting for the current game loop in each time-step, we expect that NN 1.1 will change weights of the other features based on how much time has elapsed in a game to account for such hypothetical scenarios.

Additionally, by adding the MMR of players to the features of our modified neural network, we anticipate that NN 1.1 will more accurately predict winners at the beginning of matches compared to NN 1.0. Finally, by accounting for the race of players, NN 1.1 may learn which factors are more important in certain race matchups, leading to higher overall prediction accuracy.
Figure 8: Adding Game Loop, MMR, and Race Leads to Marginal Accuracy Improvement In Random Supervised Learning. Contrary to initial expectations, accounting for MMR does not improve the initial prediction accuracy of NN 1.1. However, accounting for game loop and race leads to marginally improved prediction accuracy by an average of 4.63 percent for the third, fourth, and fifth deciles.

The prediction accuracies of NN 1.0 and NN 1.1 are presented in Figure 8. Contrary to our initial hypothesis, accounting for the game loop, the MMRs of players, and the races of players failed to significantly improve the accuracy of our StarCraft II neural network. There are a few reasons that could elucidate this unexpected finding. Adding the game loop and race may not have substantially improved the neural network because of the small size of our data set. Perhaps there were too few games in which strategies depended on the current game loop that there was no noticeable improvement in mid-game prediction accuracy ($t > 0.5$). Perhaps the model did not have enough data to learn the late-game nuanced relationships between the races. Finally, adding MMRs might not have improved initial accuracy because every match in our data set was played on the competition ladder, where players are paired together based on their MMRs. Consequently, the difference in MMRs may have been negligible in this data set to significantly impact prediction error. Nevertheless, the addition of these features did lead to minor improvements in the prediction accuracy of early-game deciles in our StarCraft II neural network.

4.3.3 NN 1.1 Performance on Large MMR Differential Data Set

To test NN 1.1 performance on matches with significant MMR differences, we filtered our data set for games in which the difference in the MMRs of players was greater than 300. In StarCraft II, there are multiple skill leagues, ranging from Bronze to Grandmaster. Each league except Grandmaster can be further broken down into three sub-tiers. Ranging from lowest-ranked to highest-ranked, the tiers are: III, II, and I. A difference of 300 MMR is approximately a two-tier skill difference. On the competition ladder, a StarCraft II player will rarely match with another player whose MMR differs by more than 300.

Figure 9: NN 1.1 Accuracy for Large MMR Differential Games. The yellow line represents prediction accuracy of games with MMR differences greater than 300. The blue line represents prediction accuracy of the general data set. In the first decile of games with significant MMR differences, NN 1.1 achieved 73.8 percent accuracy, a 26 percent improvement compared to its predictions on the first decile of the general data set. Prediction accuracy decreases during the second and third deciles, which indicates that the higher-skilled players are performing game-actions that are not being rewarded by the features in NN 1.1.
After filtering the data set, there were 426 games in which the difference in the MMRs of players was greater than 300. Figure 9 shows the predictive performance of NN 1.1 on the general replay data set and on the replays with significant MMR differences.

In the first decile of games with significant MMR differences, NN 1.1 realized an accuracy of 73.8 percent, a 26 percent improvement compared to its predictions on the first decile of the general data set. Since the in-game feature inputs such as resource count and army count are similar for both players in the first decile, NN 1.1 must be relying on the difference in MMRs to predict the winner. Specifically, NN 1.1 predicts that the player with the higher MMR will win the match. The higher prediction accuracy in the first decile indicates that the higher MMR player actually does win approximately 73.8 percent of the time.

However, prediction accuracy declines during the second and third deciles, which suggests that the better players (the players who actually win the match) are performing game-actions that are not being accounted for in NN 1.1. This implies that NN 1.1 is inadequately synthesizing the actions of higher-skilled players, which makes sense since the features of NN 1.1 are broad, macro-oriented game-state values. Future research endeavors may want to explore what specific features could be added to prevent this decrease in prediction accuracy.

4.3.4 NN 1.1 Performance on Data Set Controlled for Skill League

The analysis of NN 1.1 in section 4.3.3 poses interesting implications regarding the ability of higher-skilled players to perform game actions that are not measured by our network’s features. While section 4.3.3 investigated games between players with significant differences in MMR, we have yet to explore how the baseline skill levels of players affect the prediction accuracy of our model.

In this section, we partition our data set of replays into three different skill leagues: Bronze and Silver (BS), Gold and Platinum (GP), and Diamond and Master+ (DM). Replays were categorized into one of the three skill leagues based on the average MMR of both players (BS \( \leq 2640 \leq \text{GP} \leq 3440 \leq \text{DM} \)). There were 926 BS matches, 1,769 GP matches, and 1,611 DM matches.

We hypothesize that in lower leagues, the winner of a match is typically the player who sustains the better economy. Novice players tend to be unaware of efficient build-orders that maximize their resource collection rates and exhibit unpredictable play-styles. On the other hand, skilled players try to mimic a handful of consistent, well-known strategies used by professional players. As a result, we predict that a lack of a consistent play-style in lower leagues should allow our model to better predict matches in lower leagues.

Figure 10 depicts the performance of NN 1.1 across the various skill leagues. While prediction accuracies across the three leagues are similar for the first three deciles, disparities arise starting from the fourth decile. Between the fourth and eighth decile, the prediction accuracy for BS matches is 8.78 percent higher than that of DM matches. In the seventh and eighth deciles, the prediction accuracy for BS matches is 9.1 percent higher than that of GP matches.

While there does not seem to be a substantial improvement in predicting GP matches over DM matches, NN 1.1 more accurately predicts BS matches. This suggests that players in Bronze and Silver leagues frequently and promptly reach game-states in which one player has a substantial macroeconomic advantage over the other. One potential explanation for the disparity in players’ economies may be that lower-level players lack rigid build orders, which creates more potential for players to execute drastically different play-styles.
Figure 10: Accuracy of NN 1.1 Varies By Skill League. In the seventh and eighth deciles, the prediction accuracy for BS matches is 9.1 percent higher than that of GP matches. Between the fourth and eighth decile, the prediction accuracy for BS matches is 8.78 percent higher than that of DM matches. There does not seem to be a substantial improvement in predicting GP matches over DM matches. This suggests that by the time players have reached Gold league, predicting the winner of a match solely by focusing on macro game-state features becomes harder.

4.4 Evaluating Model Accuracies

Although we have examined the prediction accuracy of our StarCraft II neural networks as a function of the time ratio of a game, we have yet to assess how accurate the probability predictions of NN 1.0 and NN 1.1 are. In this section, we attempt to evaluate the accuracy of the predicted probabilities by measuring the percent correct classification of both neural networks. As in section 3.4, we bucket the network probability predictions of NN 1.0 and NN 1.1 into deciles. For each bucket, we calculate the actual win percentage of player 1 and plot this ratio as a function of the model’s predicted probability of player 1 winning.

Figure 11 displays the reliability curves of NN 1.0 and NN 1.1. The reliability curves of both StarCraft II neural networks barely deviate from the identity function. The mean-squared error of the probability predictions of NN 1.0 is 0.001508 and the mean-squared error of the probability predictions of NN 1.1 is 0.000353.

Both NN 1.0 and NN 1.1 demonstrate good calibration. However, it appears that the additional features of NN 1.1 help it achieve a higher percentage of correct classification than NN 1.0. As a result, while both models are well-calibrated, NN 1.1 appears to be the more accurate neural network based on two accuracy metrics: general prediction accuracy and percent correct classification.

Figure 11: Reliability Curves of StarCraft II Neural Networks. For each game-state observation in the test set, NN 1.0 and NN 1.1 generate their own probabilities of player 1 winning. We plot the actual win percentage of player 1 (from the same game-state observations in the test set) as a function of the model-predicted probabilities of player 1 winning. We also plot the hypothetical performance of a perfectly accurate model as a comparison.
5 Conclusion and Future Work

In this project, we explore methods to accurately predict winners in real-time strategy games using various neural networks. We begin this study using the simplified environment provided by microRTS. For microRTS, we develop both categorical and probabilistic neural networks that use features provided by single-frame game observations to predict a player’s likelihood of winning a match. We found that each model accurately predicts the winner in relatively early stages of games. Even the simplest categorical neural network achieves more than 80 percent accuracy at the midpoint of matches.

By adding specific army composition features to the simple categorical neural network, we were able to noticeably improve the model’s early-game prediction accuracy. We attribute the improved performance of the army composition model to the neural network’s ability to learn the relative dynamics between units.

We expand the framework of our microRTS models to construct similar, but more complicated, neural networks for StarCraft II. Because StarCraft II matches are rarely played to completion, we show that StarCraft II prediction models hover around 50 percent accuracy for a longer proportion of the early-game than microRTS models. Moreover, StarCraft II neural networks lack the high-accuracy plateau periods of microRTS models. Despite the more complex game mechanics of StarCraft II compared to microRTS, our simple StarCraft II neural network, NN 1.0, manages to attain close to 75 percent accuracy by the seventh decile of games.

We improve the accuracy and calibration of NN 1.0 by extending the feature vector to account for the current game loop, the MMRs of players, and the races of players. We also demonstrate that the prediction abilities of both NN 1.0 and NN 1.1 depend on the skill-disparity and the baseline skill-levels of players. Finally, we verify that the probability outputs of NN 1.0 and NN 1.1 are well-calibrated.

Future research endeavors regarding winner prediction in real-time strategy games may want to explore models that take into account of the hit-points remaining of units and building structures. The inclusion of this information as additional features may allow future models to reach higher prediction accuracy sooner than our existing models because future models may learn which player is winning a fight before units are actually killed. Furthermore, future prediction models may want to incorporate the geographical positioning of units using feature layers as network inputs. Adding the location of army units into the feature vectors of future prediction models could help neural networks determine which player has the advantage leading up to large engagements involving similar-sized armies. Finally, subsequent work in StarCraft II may want to use a larger data set than the one available to our research group to check if the introduction of more data allows neural networks to learn more nuanced trends.

Nevertheless, we believe our initial results are promising for the burgeoning field of e-sports. Our neural networks incorporate only broad in-game attributes of game-states and do not encode presumably valuable information such as the geographical positioning of units or army composition. Despite the simplifications to our StarCraft II neural networks, however, we manage to predict the winning player with relatively high accuracy. Hence, our initial models may serve as valuable preliminary designs for applications in e-sports betting and for visual tools in professional game-casting.

6 Acknowledgements

I would like to thank Professor James Glenn for being our senior project advisor, and for his guidance and support throughout the course of the semester. I would also like to thank my project partner Andrew Jin for helping me in the idea generation process.
References


