# FINDING COMPETITIVE MICROMANAGEMENT COMMANDS IN STARCRAFT USING NEURAL NETWORKS

THOMAS CUMMINGS

**ACM Reference Format:**
Thomas Cummings. 2019. FINDING COMPETITIVE MICROMANAGEMENT COMMANDS IN STARCRAFT USING NEURAL NETWORKS. 1, 1 (December 2019), 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 ABSTRACT

StarCraft is a fascinating area of research for machine learning. This research focuses on using neural networks to control armies in two specific combat scenarios that are commonly found in games of StarCraft.

## 2 BACKGROUND

StarCraft is a beloved classic in the genre of Real-Time-Strategy games. StarCraft was created by Blizzard Entertainment company in 1998 and later that year expanded with the StarCraft: Brood War expansion pack. In 2017, StarCraft: Remastered was released, which did not change the gameplay but instead updated the graphics and allows the game to be played in much higher resolution [1]. StarCraft is played by collecting two different types of resources, constructing buildings, and training combat units to attack the opponent. The game starts with each player having a home base and a few resource collecting units. As the game progresses, each player must create new buildings that can be used to train new combat units, such as soldiers and tanks, and building new bases for collecting resources when resources in the home base are depleted. The game can be played using many different strategies, but most strategies involve building an army of combat units and using that army to destroy the opponent's army and buildings. The game is won when the opponent no longer controls any buildings. A typical game lasts between 20 minutes and 30 minutes.

### 2.1 Strategy Basics

Strategy in StarCraft can be divided into two different categories, macromanagement and micromanagement. Macromanagement (or macro) deals with strategies about how many resources to gather, which buildings to create, which combat unit upgrades (technologies) to research, how many combat units to build vs how many resource collecting units to build, and what time should units be built. There are three main macro strategies utilized in in the early stages of the game, economy, tech, and rush.

Author's address: Thomas Cummings.

In an economy strategy, the player focuses on maximizing resource collection in the early game, and then using those resources to build a large army of combat units to attack the opponent in the middle game. In a tech strategy, the player focuses on constructing technology buildings which allow the player to build high-tech combat units, and then attack the opponent in the middle game with these high-tech units. In a rush strategy, the player focuses on training as many combat units as possible as early as possible and overwhelming the opponent in the early game with a superior army. This paper focuses on a common early game scenario where the defending player is using an economy or tech strategy (and thus having a smaller army in a defensive position), and the attacking player is using a rush strategy (and thus having a larger army moving out to attack). The second scenario this paper studies is a common middle game scenario where one player has used an economy strategy (and thus has a large army) and the other player has used a tech strategy (and thus has a smaller, but high-tech army).

Micromanagement (or micro) is the tactical aspect of controlling combat units in the most effective way possible. Since the game is played in Real-Time (as opposed to a turn-based game), the human player is greatly limited in time that can be used in order to micro combat units properly. Combat units can only be issued commands in groups of 12 at a time. This means that when larger amounts of units are involved, it requires many mouse-clicks and keyboard button presses to issue commands to all those units. Professional StarCraft players train to increase the speed that they can click the mouse and press buttons on the keyboard in order to micro combat units faster. Player's speed can be measured in terms of actions per minute, where each action is a command that has been issued in the game. Novice players will have speeds of less than 60 actions per minute and will see large improvements in win rate as they increase their speed up to 150 actions per minute. After 150 actions per minute, there are diminishing returns since not all the actions end up being effective. However, many professional players find improvements as the actions per minute increase well beyond 150. Many professional players can average sustained speeds of over 300 actions per minute and short bursts of over 500 actions per minute. These speeds can be absolutely overwhelming to witness for players new to the game, but these speeds are needed in order to macro and micro at the same time.

Even with this amazing speed, players are constantly having to make compromises in macro or micro because there is simply not enough time to do everything. And in StarCraft, if you must choose between compromising on macro or compromising on micro, you should compromise on micro. Because of this, there is a great deal of strategy improvement that can be done in micromanagement using machine learning because the computer speed means it doesn't need to compromise on micro.

Micro plays such a key role in successfully winning battles that strategies that distract the opponent, such as attacking in multiple places at once, can be extremely effective. These strategies can often outright win games against all but the most skilled opponents. Because micro requires such a large time investment for humans and there is such a limited amount of time in each game, this category of StarCraft strategy is the part where machine learning will find results most quickly.

This project focuses on the results of improving micromanagement in two common and pivotal engagements in StarCraft when good micro will be at its most important. The first is a combat scenario happens during the early game where a player uses an "economy" or "tech" strategy and must defend against an opponent who is using a "rush" strategy. The defending player has a smaller army than the attacking player and must defend their buildings from the attack. The second combat scenario is when an "economy" player is attacking a "tech" player during the mid-game. The economy player has a larger army, but the defending player has more effective high-tech combat units. Controlling units properly during these scenarios is important for developing strong AI opponents in StarCraft.

## 3 RELATED WORK

In this work, two combat scenarios are simulated, and the results measured. Another avenue is to analyze saved games of StarCraft which are called "replays". Research has been done to learn macromanagement from replays of professional StarCraft players [2]. This work on macromanagement is similar but not the same as micromanagement. However, combat scenarios from actual games would be more accurate than the combat scenarios used in the paper. Using the technique of analyzing replays would be a good source of future research on this topic.

Research is also being done using reinforcement learning [3]. This research focuses on learning micromanagement, and would has similar goals of controlling combat units effectively. A combination of the data collection technique from this project and reinforcement learning could be used for further research.

## 4 METHODOLOGY:

This project is made up of two components. The first generates a training dataset for training the neural network. A Java program was written to create this dataset by running two common combat engagements in StarCraft.

The second component trains the neural network on that dataset and uses the neural network's predictions about the outcome of a combat scenario to determine a suitable command for the combat units.

### 4.1 Setting up the combat scenario:

The battlefield in StarCraft is determined by the "map" (a file created by the StarCraft Map Editor program) and contains all the battlefield terrain (and in this case, the combat units needed for each combat scenarios). The map file is selected at the beginning of each game. A custom map is created for each of the combat scenarios. The contents of the first map are: 8 defending combat units (Zealots), and three defending buildings (Nexus, Pylon, and Photon Cannon), and 24 attacking combat units (Zerglings).

The map for the second combat scenario contains 24 attacking combat units (12 Zealots and 12 Dragoons) and 20 defending combat units (8 Siege Tanks and 12 Vultures).

### 4.2 Running the Combat Scenario:

The two combat scenarios require a different setup before they can be run. For this project, each combat unit is placed randomly within a specific area. Once the armies are in place, then one army receives the command to attack. Figures 1 and 2 show the two combat scenarios.

### 4.3 Evaluating the Result of a Combat Scenario

Each battle is assigned a score based on several factors, including the number of combat units that survived and how healthy they are. Each combat unit that is still alive is worth 50 points, each remaining hit point is worth one point, and each remaining shield point is worth 0.5 points.

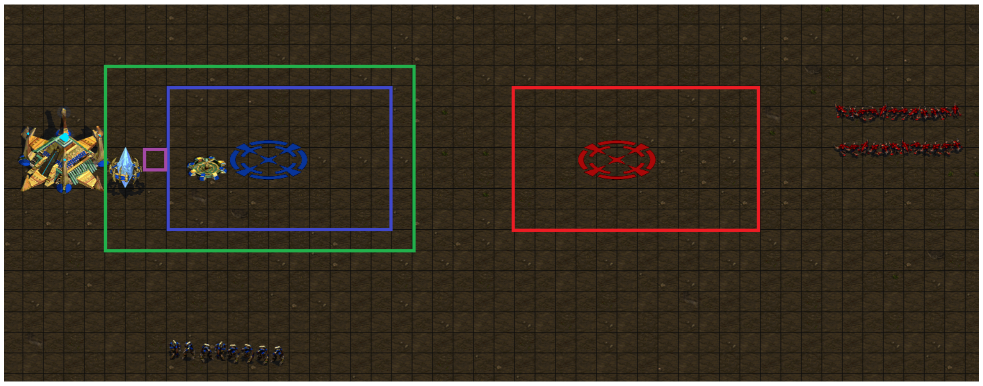

Fig. 1. Combat Scenario #1 Setup. The defending combat units (8 Zealots) are placed randomly in the blue grid and the attacking combat units (24 Zerglings) are placed randomly in the red grid. The attackers always attack towards the purple location. The defenders are commanded to move to a location within the green grid.

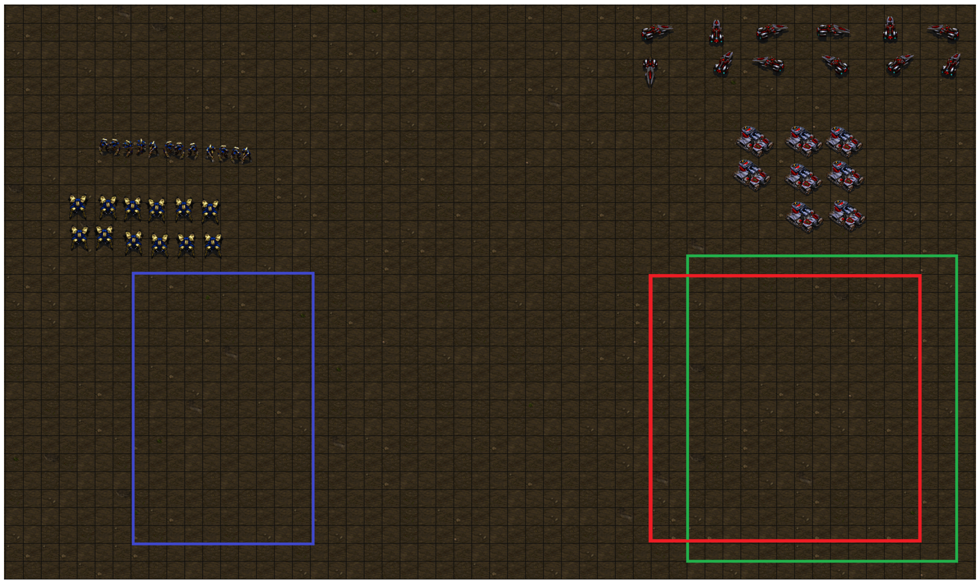

Fig. 2. Combat Scenario #2 Setup. The attacking combat units (12 Zealots and 12 Dragoons) are placed randomly in the blue grid and the defending combat units (8 Siege Tanks and 12 Vultures) are placed randomly in the red grid. In addition, the Siege Tanks will use their "Siege Mode" ability, and the 12 vultures will each use their "Spider Mine" ability to place a total of 12 "Spider Mines". The attackers are commanded to move towards a position in the green grid.

Combat scenario 1 focuses on defending a base from an opponent's attack. 100 points are awarded for keeping the Photon Cannon alive with additional points awarded for the remaining hit points and shields. Points are deducted if the base buildings are destroyed, with 300 points deducted for allowing the Pylon to be destroyed and 5000 points deducted for allowing the Nexus to be destroyed.

Combat scenario 2 focuses on destroying the opponent's army. Points are deducted for failing to kill the enemy combat units, with 25 points deducted for each enemy Vulture that survives and 50 points deducted for each enemy Siege Tank that survives. Additional points are deducted for the remaining hit points on any surviving units. No points are deducted for any remaining Spider Mines.

## 4.4 Interfacing with StarCraft to Collect the Training Dataset:

A Java program was created to move the combat units to their starting locations and conduct the combat scenarios. Figure 3 describes the method for interfacing with StarCraft using Broodwar API (BWAPI).

StarCraft Broodwar version 1.16.1 was used because it is compatible with BWAPI 4.12. BWAPI has a C++ interface which can be used by Java programs using BWMirror, a Java wrapper for the C++ interface.

The Java program only controls one player at a time. This presents a problem since armies from both players need to be positioned before each combat scenario starts. Creating map code (called "triggers") solved this problem. The triggers temporarily give control of all combat units to the player being controlled by the Java program. After positioning both armies, a second map trigger returns control of the combat units to their original player. There is room for improvement in this part of the project since we would have better control setting up the combat scenario by having two Java programs connect to the game using network play. While the triggers are useful, the precision of control is much higher when using a Java program using BWAPI.

The Java programs use a BWAPI feature allowing the combat scenarios to be run with greatly increased game speed. The first combat scenario ran 20,000 times and the second combat scenario ran 40,000 times. These datasets are available on GitHub at https://github.com/LegendaryTom/StarcraftMLFinalData.

## 4.5 Training the Model:

The "Keras" library is used to create the neural network. There are two networks created, one for each combat scenario. Table 1 and table 2 show that 66 inputs controlled the neural network for the first combat scenario and 114 inputs for the second scenario. The inputs are the coordinates of the combat units and the command that the combat units were sent to attack towards. The neural network's output is the estimated score of the battle. The "sklearn" library is used to normalize the inputs before training.

## 4.6 Neural Network and Hyperparameters:

A fully connected neural network is used for training on both datasets. The neural networks contain 4 hidden layers. Each hidden layer is the same size as the input layer. High learning rates often result in the weights becoming too large. So, low learning rates between 0.001 and 0.0001 are used for the first combat scenario and 0.01 and 0.001 for the second combat scenario. The Adam optimizer is used. It is important to normalize the training data before training the neural network. Unnormalized training data causes extremely large losses during training. There is room for future improvement

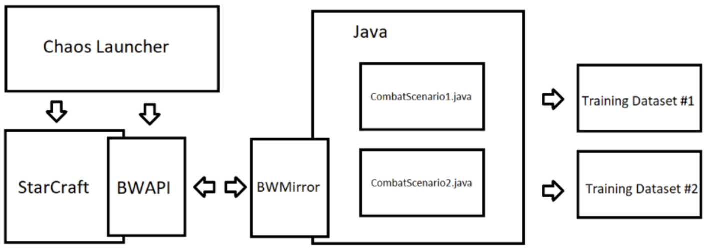

Fig. 3. Project Architecture Diagram. StarCraft is started with the Chaos Launcher tool and enables the BWAPI interface. The BWAPI tool exposes a C++ interface to interact with StarCraft. BWMirror is a Java wrapper for the C++ interface, allowing for two custom Java programs to control the two combat scenarios. The two Java programs each position combat units for the engagement, command the combat units to engage in battle, and then stores the result in a CSV to be used as training data.

in selection of hyperparameters because there was not enough time to search for optimal parameters.

When confronted with a scenario where the defenders and attacker's positions are known, we desire to use this neural network to predict the best move for the defenders/attackers. However, the neural network only predicts how successful a battle will be and does not directly output the optimal command for the combat units. Directly predicting the optimal best coordinate to attack to is useful for these two combat scenarios, but an estimate of the battle score is useful in a wide range of problems for creating StarCraft AI. Knowing the result of a future battle lets us answer important questions. For example, it's important to know if an attack will be successful so that a meaningful decision can be made about whether to attack, wait, or retreat. Additionally, this neural network will be able to estimate the battle result with differing starting positions for the combat units. This allows an AI to test different starting positions for the combat units, and position units before the fight to get a better result in the battle.

## 4.7 Using the Model to Select the Best Command

Generally, we will be dealing with a situation where the positions of all the combat units is known but we don't know what the best command is. This means that all the inputs except the last two ("Position to move Attackers X/Y coordinate") are known. To find good coordinates to send the attackers, we simply test all the possible input values for those last two inputs. This gives us a predict battle score for each possible command we can give. The coordinate pair with the highest predicted battle score becomes our selected command.

## 5 RESULTS

To see the results of the command coordinates picked by the neural network, 1000 trials were generated for each combat scenario. These trials were then run in StarCraft with the command coordinates that were predicted to do best by the neural network.

The results for the first combat scenario showed a large improvement in score over selecting a random command. The neural network focused on putting the zealots right in front of the cannon (between the attacking Zerglings). This provided a much higher average score. In addition, the results were consistently higher with zero battles resulting in buildings being destroyed.

For combat scenario 2, the neural network prioritized the corner and predicted that the bottom right corner of the green grid for every battle. Using this result doubled the average battle score as can be seen in Figure 5. Despite the improvement in average battle score, there are still many runs of the combat scenario that end in defeat. So there is a lot of variation in the results for this combat scenario. This scenario is much larger (nearly twice as many inputs to the neural network) and there's not a readily apparent solution (such that a human could point out easily what the solution would be). Because of this, there may be no solution that provides a winning battle score for every run as was found in the first scenario. However, a more consistent solution was looked for.

| First Combat Scenario | Inputs | Outputs |
|---|---|---|
| Zealot Position (X-coordinate) | 8 | |
| Zealot Position (Y-coordinate) | 8 | |
| Zergling Position (X-coordinate) | 12 | |
| Zergling Position (Y-coordinate) | 12 | |
| Position to move Defenders (X-coordinate) | 1 | |
| Position to move Defenders (Y-coordinate) | 1 | |
| Battle Score | | 1 |

Table 1: List of inputs and Outputs for the First Neural Network

| Second Combat Scenario | Inputs | Outputs |
|---|---|---|
| Zealot Position (X-coordinate) | 12 | |
| Zealot Position (Y-coordinate) | 12 | |
| Dragoon Position (X-coordinate) | 12 | |
| Dragoon Position (Y-coordinate) | 12 | |
| Vulture Position (X-coordinate) | 12 | |
| Vulture Position (Y-coordinate) | 12 | |
| Spider Mine Position (X-coordinate) | 12 | |
| Spider Mine Position (Y-coordinate) | 12 | |
| Siege Tank Position (X-coordinate) | 8 | |
| Siege Tank Position (Y-coordinate) | 8 | |
| Position to move Attackers (X-coordinate) | 1 | |
| Position to move Attackers (Y-coordinate) | 1 | |
| Battle Score | | 1 |

Table 2: List of inputs and Outputs for the Second Neural Network

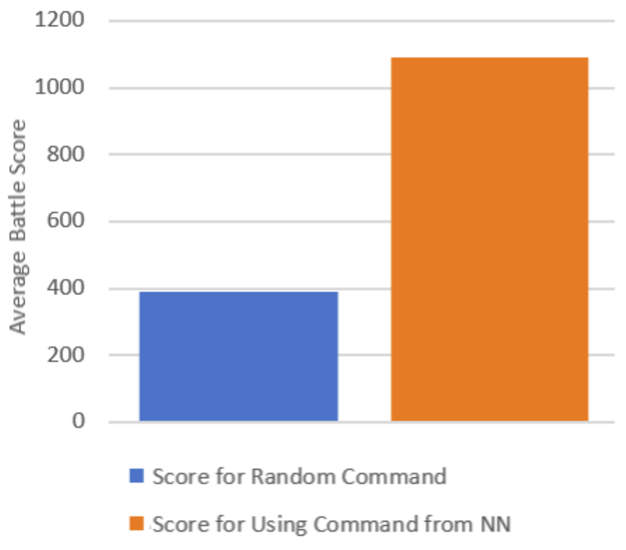

Fig. 4. Results for the first combat scenario. The average battle score from submitting a random command is significantly lower than the average battle score from using a command from the neural network.

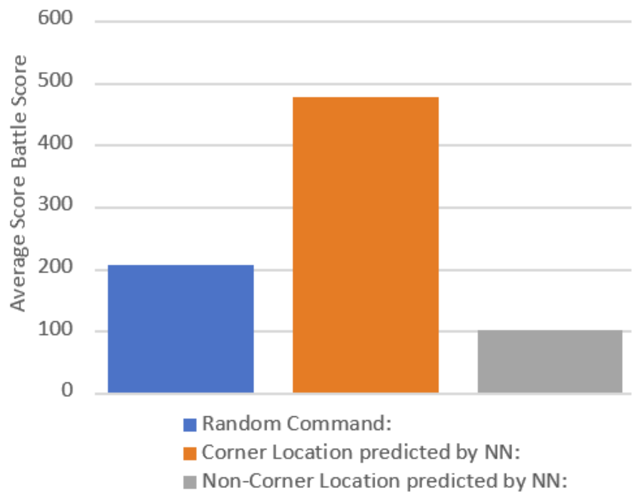

Fig. 5. Results for the second combat scenario. The neural network's prediction to command the units to the corner location resulted in a doubling of the average battle score when compared to a random command. Continued training caused overfitting and predicted non-corner locations that scored worse than a random command.

Continued training of the model was done, and the model began predicting positions that were not the corner position. When the neural network was trained with many epochs (5000) or more, the network stopped prioritizing the corner and would start predicting other locations as scoring higher than the corner. Increasing the number of epochs increased the percentage of predictions that didn't show the corner as the best. At 20000 epochs, 89% of the predictions were not a corner prediction. However, the predicted locations were not performing as well as the score for a random command. The conclusion is that these predictions were worse because of overfitting to the training data. For this scenario, the best that can be done is to always attack towards the bottom right corner of the green grid for every battle.

## 6 CONCLUSION

Neural networks can be used to successfully control the micromanagement of units in StarCraft. The results of the two combat scenarios above show a greatly improved average battle score when compared with a random command. In the first combat scenario, the neural network identified a command for a defensive position that many humans would point out as a good or best solution. In the second combat scenario, there wasn't such a readily apparent solution. However, the neural network identified an attack command that scored twice as well as a random attack command.

## 7 REFERENCES

(1) Matulef, Jeffrey. "StarCraft and Its Brood War Expansion Are Now Officially Free." Eurogamer.net, Eurogamer.net, 18 Apr. 2017, https://www.eurogamer.net/articles/2017-04-18-starcraft-and-its-brood-wars-expansion-are-now-officially-free.
(2) Justesen, Niels, and Sebastian Risi. "Learning Macromanagement in StarCraft from Replays Using Deep Learning." Learning Macromanagement in StarCraft from Replays Using Deep Learning - IEEE Conference Publication, 26 October 2017, https://ieeexplore.ieee.org/document/8080430.
(3) Wender, Stefan, and Ian Watson. "Applying Reinforcement Learning to Small Scale Combat in the Real-Time Strategy Game StarCraft:Broodwar." Applying Reinforcement Learning to Small Scale Combat in the Real-Time Strategy Game StarCraft:Broodwar - IEEE Conference Publication, 11 Sept. 2012, https://ieeexplore.ieee.org/document/6374183.