

# Optimizing Small Scale Combat With Neural Networks

Peixian Wang  
May 3, 2016

## Abstract

A major challenge in real-time strategy games is individual unit control, where the player must issue separate orders and deploy units in different places in order to maximize the effectiveness of a single unit. Starcraft: Brood War (henceforth referred to as Starcraft), a real-time strategy game released in 1998 by Blizzard Entertainment, provides a ripe environment for optimization within combat deployment. We present a backpropagation neural network design that hooks into the Alpha-Beta combat algorithm, which then allows an AI to consistently position units before combat in a manner to beat the build-in computer AI.

## 1 Introduction

Micro-management, or the management of units on an individual scale, is an extremely integral part of modern real-time strategy (RTS) games today. Players are required to move units in order to gather resources, construct buildings, or engage in combat. Within combat alone, effective players must decide: 1. which units to engage with, 2. proper battlefield to engage on, and 3. where the engaging units should be placed. Oftentimes, this level of decision making has happened on the scale of seconds for human players, where quick decision making is oftentimes rewarded with a successful engagement. Due to the fast-paced nature of combat micro-management, decisions are oftentimes complex despite only having a small number of units.

Modern RTS's provide a limited amount of assistance within micro-management, commands can be issued en masse to a group of units. However, effective management of units on an individual scale oftentimes lead to a more favorable outcome, such

as the technique called Scourage Cloning<sup>1</sup>. While players are able to devise complex techniques in order to work around the limitations of Starcraft itself and selectively issue batch commands when necessary, AI playing Starcraft do not have access to the same nuances and quick decision making a player does. Consider the following scenario: a player has a heterogeneous distribution of units, with both units that can only attack within a very small melee range (melee units) and units that attack from a distance (ranged units). A good player would know to place melee units in front of the ranged units in order to maximize combat effectiveness when moving the group, as the ranged units can still fire upon the enemy when the melee units are attacking the enemy, but the melee units would be blocked behind the ranged units if unit positions were switched around. To an AI, however, this distribution of units looks homogeneous, there is no easy way for the AI to realize that ranged units would be more effective if placed at a distance behind the melee units.

This combat-oriented domain of micro-management is extremely well-suited for the applications of neural networks, as neural networks can learn the proper positioning of units based upon repeated trials and a large corpus of training data. In this paper we present an application of a neural network trained through backpropagation, over the course of 3000 games an AI properly learns the correct positioning of a small melee unit called the Zergling. In section 2 we discuss some background on Starcraft and SparCraft, a combat simulation program. In section 3 we discuss the technology stack used to enable this neural network and the AI itself, along with introducing our AI called Ultralisk. Section 4 recounts the technological approach to solving the problem, including the topology

---

<sup>1</sup>[http://wiki.teamliquid.net/starcraft/Scourage\\_Cloning](http://wiki.teamliquid.net/starcraft/Scourage_Cloning)



Figure 1: A *Zergling*, the basic *Zerg* infantry unit.



Figure 2: A *Zealot*, the basic *Protoss* infantry unit.

of the neural network, obtaining training data through SparCraft, and feeding the data into the Alpha-Beta combat algorithm. Section 5 and 6 detail the results of the simulation and experiment trials, along with concluding and discussing the interpretations of the results.

## 2 Background

### 2.1 Starcraft

Starcraft is an extremely complex game that has captured an entire country, becoming the first major e-sport. At a high level, Starcraft consists of 3 stages:

1. Use units to gather resources (called *Minerals* and *Vespene Gas*).



Figure 3: A *Marine*, the basic *Terran* infantry unit.

2. Spend resources to construct buildings, which enable gathering more resources and constructing more powerful units (called *expanding* and *moving up the tech tree* respectively).
3. Use the accumulated units to engage the enemy units, with the eventual goal of destroying all of the enemy buildings and units.

Starcraft is a robust game, consisting of 3 races (called the *Zerg*, the *Protoss*, and the *Terran*), each with their own unique buildings and units. In order to meet the time constraints, we focused the three basic infantry units in particular, each for a single race:

1. *Zerg* - *Zergling* (Figure 1) - A small, melee-ranged, fast-moving unit with 35 health points (HP)
2. *Protoss* - *Zealot* (Figure 2) - A medium-sized, melee-ranged, moderate-moving unit with 160 HP
3. *Terran* - *Marine* (Figure 3) - A small, ranged, moderate-moving unit with 50 HP

As these 3 units form the backbone of any army, we felt it was necessary to train the neural network on all 3 units. Due to time constraints, the AI we built (henceforth referred to as *Ultralisk*) is only able to utilize *Zerglings*, but can play against any race.

## 2.2 SparCraft

SparCraft<sup>2</sup> is an open source abstract combat simulator written by David Churchill, from the University of Alberta for combat simulations. While it is *not* a Starcraft combat engine, rather, it is able to accurately model unit size, damage, and movement. Because it is able to simulate combat extremely quickly, it was chosen to generate a corpus of training data for the neural network.

SparCraft runs on a chosen map size, the map size chosen for this experiment was 1280 x 740, which creates a grid of 1280 by 740 cells. Each unit occupies 1 grid area, and each unit has a speed per frame. The Zergling for example, can move 1.5 grid points in any direction (it can move across its own cell and half of the next cell), while both the Zealot and the Marine can only move 1 grid point. The Marine, however, can begin attacking a target 4 cells away, while both the Zergling and Zealot can only attack units in the cell directly next to them.

## 3 Technology Stack

### 3.1 Overview

Ultralisk<sup>3</sup> is composed of two parts: 1. Glaive, the neural network that reads training data, and 2. the Ultralisk AI itself. The technology stack is shown in figure 4. By scripting SparCraft to generate hundreds of combat scenarios and then filtering those combat scenarios for preferred results, a large training corpus is generated. This training data is then fed into Glaive, which uses backpropagation to train itself. After convergence, the neural network generates data for Ultralisk to follow within its combat algorithm, which then enables Ultralisk to assign optimal unit positions.

### 3.2 SparCraft Scripts

The SparCraft scripts are meant to generate a large training corpus for Glaive to train itself on. In order to accomplish this, the scripts create two zones,

<sup>2</sup><https://github.com/davechurchill/ualbertabot/wiki/SparCraft-Home>

<sup>3</sup><https://github.com/peixian/Ultralisk>

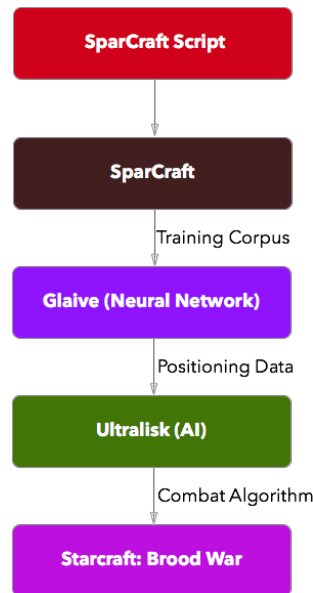


Figure 4: *Ultralisk technology stack, scripts run SparCraft combat simulations to generate a training corpus, which is then fed into Glaive. Glaive generates positioning data, which is then fed into Ultralisk, which then is carried out in the Starcraft game itself.*

shown in figure 5. The left, blue zone is where the ally can deploy its 3 Zerglings, while the right, red zone is where the enemy can deploy its choice of units between 3 Zerglings, 3 Zealots, or 3 Marines. The script then runs through 1000 scenarios, randomly generating locations for both the enemy and the ally, and then sets these locations into SparCraft to simulate the combat. If the ally is victorious after the simulation is complete, the script saves both the enemy positions and ally positions into the corpus to be trained on by Glaive later, with the enemy positions being the input (domain) and the ally positions being the target (range).

### 3.3 Glaive

Glaive (figure 6) is a neural network with 3 input nodes and 3 output nodes. While we used 100 hid-

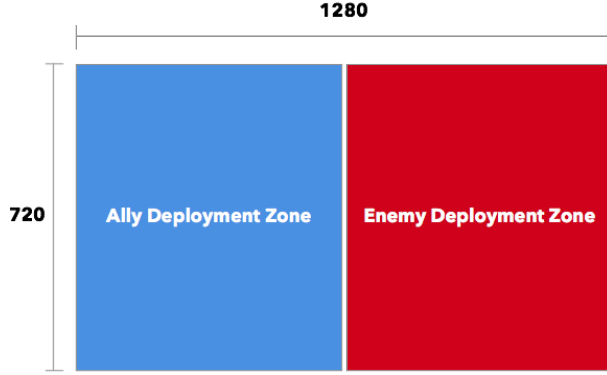


Figure 5: *Zones where the units can be deployed*

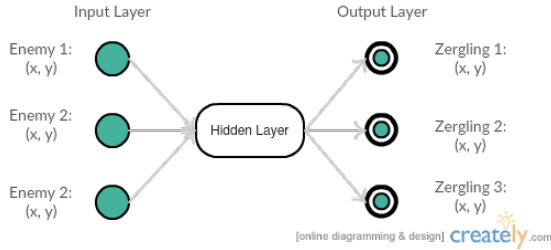


Figure 6: *Glaive has 3 input nodes and 3 output nodes. It reads the tuples of the enemy positions  $(x, y)$  for up to 3 enemies, and then generates 3 tuples with the locations of where the ally units should be placed.*

den nodes in 1 hidden layer along with bias nodes, Glaive itself can take any arbitrary amount of hidden nodes. Each input node takes in a tuple of values,  $(x_{enemy,i}, y_{enemy,i})$  for each enemy unit  $i$ , and then generates 3 tuples at the output nodes  $(x_{ally,j}, y_{ally,j})$  for each unit  $j$ . After Glaive is trained to convergence, the weights within are saved to be passed into Ultralisk.

### 3.4 Ultralisk AI

The Ultralisk AI is a C++ AI forked from UAlbertaBot<sup>4</sup>, which is a bot utilizing BWAPI<sup>5</sup>. Rather than implement our own total bot, we decided to fork the UAlbertaBot, since UAlbertaBot was already able to be modified to suit our needs. The Ultralisk AI employs a heuristic search method in order to achieve a list of goals, which in our case was to gather resources to continue spawning Zerglings indefinitely. It also employs a multi-unit path finding algorithm, where each cell in a map is separated into traversable or non-traversable, and multiple agents are created to determine the optimal path to a target, using flood-fill based implementation.

## 4 Technical Approach

## 5 Results

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

## 6 Conclusions

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy

<sup>4</sup><https://github.com/davechurchill/ualbertabot>

<sup>5</sup><https://github.com/bwapi/bwapi>

pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.