# Training a Neural Net to Analyze Dota 2 Team Composition

**Jason Teng**
Northeastern University, Boston, MA
Email: teng.ja@husky.neu.edu

## Abstract

The goal of this project was to develop a neural network that would be able to use information on hero picks to predict match outcomes in Dota 2. Match data from over 2000 professional Dota 2 matches from the last 4 months was collected and used to train a neural network. Evaluating the trained network on a new set of matches showed an accuracy of over 79% in the best case, showing that even with the vast amount of other variables in a Dota 2 match, the draft has a massive role in determining the end outcome.

## Introduction

Dota 2 is an immensely popular video game by Valve Corporation, boasting over ten million unique players worldwide each month. In addition to attracting a large audience casually, Dota 2 has a professional competitive scene, with dozens of teams consisting of players who make their livelihood by competing in tournaments. These tournaments have prize pools ranging from a few thousand dollars to tens of millions of dollars, the largest prize pools in all competitive online gaming.

Dota 2 is a rather complex game, where ten players, split into two teams of five, each control an individual "hero", with the ultimate objective of destroying the other team's base or forcing them to forfeit. In addition to the basic mechanics of the game, a huge amount of complexity is added by the fact that the ten heroes in any particular game of Dota 2 are chosen from a pool of 115 heroes, each with unique mechanics, and their own unique skillset. Since duplicates are not allowed, this means that there are over $7 * 10^{13}$ unique hero matchups in Dota 2.

With potentially millions of dollars riding on each game, many professional organizations spend significant resources attempting to extract every possible advantage they can. Almost every professional Dota 2 team has a dedicated coach, who does not play, but instead spends time analyzing other teams to develop counter strategies, as well as advise the players to help them perform better.

A more objective method of finding an advantage, however, can also potentially be found by analyzing statistics. One rather easily identifiable statistic is how successful certain "drafts" are in competitive matches.

In competitive matches of Dota 2, games are preceded by a drafting phase, where teams take turns selecting the heroes that each side will play during the game. As previously mentioned, with 115 heroes to choose from, and 10 total heroes selected for each game, there are $7 * 10^{13}$ possible drafts in Dota 2. If one only considers the 5 heroes selected for a single team, and disregards the picks for the other team, there are still over 150 million possible drafts for each side.

The question is thus: Can the draft be used to predict the outcome in a professional match of Dota 2? In this project, a neural network with two hidden layers was trained on data obtained from 2000 professional Dota 2 matches to determine if such a correlation exists.

## Background

Artificial neural networks, often shortened to neural networks, are machine learning systems whose design is inspired by real life animal brains. In neural network theory, the basic unit in a neural network is a perceptron, analogous to how the neuron is the basic building block of a brain. A perceptron takes a set of inputs and applies some weighted transformation on the inputs to produce an output. Again, this is analogous to how real life neurons can have multiple inputs, including physical and chemical inputs, or the action potentials from other neurons, and then decides whether to fire its own action potential or not.

At this point, the analogy breaks down, and artificial neural networks begin to differ greatly from real life nervous systems. For example, while

neurons' action potentials are binary (either a neuron fires or doesn't), a perceptron always sends out an output, whose value can be any real number (though typically it is bound to either between -1 and 1 or 0 and 1, depending on the function used).

The analogy further breaks down when one considers that in actual implementations of neural networks, the abstraction of perceptrons is stripped away, and the neural network is defined completely by the set of matrices and vectors describing its various weights and matrices, as well as the particular non-linear function used in the neural network.

Neural networks, like any other machine learning algorithm, can be trained using any of the three major learning paradigms: Supervised Learning, Unsupervised Learning, and Reinforcement Learning. Due to the nature of the data, with historical match data with known outcomes, supervised learning is the most logical choice for this project.

As previously stated, a neural network is defined by a set of weights and biases, along with an activation function. For this project, the sigmoid activation function was chosen, although the tanh function is another popular non-linear function for neural networks. In general, the output of a perceptron *i* in layer *j* is equal the weighted sum of all outputs of the perceptrons from the previous layer, plus some bias term, then run through the activation function. More specifically,

$$output = f\left(\left[\sum_{n} w_{n\,i\,j-1} x_{n\,j-1}\right] + b_{ij}\right)$$

where $w_{nij\text{-}1}$ refers to the weight value associated with the connection between perceptron *n* in layer *j*-1 to perceptron *i* in layer *j*, $x_{nj\text{-}1}$ refers to the output of perceptron *n* in layer *j*-1, and $b_{ij}$ is the bias value associated with perceptron *i* in layer *j*.

Once the output of the final output layer has been calculated, it can be compared to the desired output via an error function, and the weights and biases of the network can be updated via backpropagation.

## Related Work

Part of the inspiration for this project was the release of Dota Plus by Valve, a subscription-based service for Dota 2 which, among other things, offered something called the Plus Assistant. The Plus Assistant offers hero suggestions during the drafting phase and item and skill build suggestions in real-time during games. Additionally, when spectating a game in progress, the Plus Assistant will show a win prediction, showing which team is most likely to win based on the current game conditions.

While Valve has not released any details regarding the implementation, in the description of the Plus Assistant, it is stated that the suggestions are "generated from data gathered across millions of recent games at each skill bracket". Based on the sheer volume of data that needs to be processed, as well as the requirement for a real-time update, it would make sense that they use some sort of machine learning algorithm to dynamically learn various statistics to be displayed to the player.

While the Plus Assistant can deliver some useful statistics, it does not answer the primary question of this project for two reasons. First, either by design or some quirk of its implementation, the Dota Plus Assistant always predicts the outcome of match to be 50% in favor of either side at the beginning of the match. This is likely to dissuade players from becoming overly pessimistic if they were to see that they had a low chance of winning before the game even properly started, but it makes the Plus Assistant fundamentally useless in answering the question of how impactful the draft is on winning a game of Dota 2.

Second, the win percentage dynamically updates over the course of a game based on numerous factors that change during a match, including net worth and experience differential. Once again, the question posed in this project is much more focused in scope. After all, a team might have a large advantage fifteen minutes into a match because of an advantage gained in the draft. On the other hand, a team might have a draft advantage that does not manifest until much later into a match. Therefore, it is much more useful to consider the overall end result from a draft, rather than trying to predict the winner during the course of a game, which might not be representative of a draft's overall strength.

## Project Description

A neural network with two hidden layers was implemented for this project. The pseudocode is shown below in Figure 1:

```
1: initialize weights and biases to
random values

2: repeat (# epoch) times:
3:      for each data sample:
4:          forward propagate using
current weights and biases
5:          calculate training error
6:          accumulate training error
with previous errors
7:      back-propagate errors to update
weights and biases
```

Figure 1: Pseudocode for Neural Network Training

Overall, the algorithm for training the neural network itself is fairly straightforward. In fact, much of the difficulty in this project stems from data collection and formatting.

Luckily, Dota 2 professional match data is made publicly available. However, to limit the server bandwidth, Valve only allows third-parties to query around 200 matches at a time. Therefore, match data must be collected using a combination of automated scripts, as well as manual input to select the match ID ranges, in order to collect a sufficiently large sample size. Furthermore, due to the frequent updates to the game which often drastically change the balance of many heroes, it is not representative to use the data from matches from patches which are too far apart. Ultimately, the data from 2000 professional matches was collected, whose dates ranged from December 2017 to April 2018. This seemed like a reasonable range, as there was a major patch to Dota 2 in early December 2017, which would invalidate match data from earlier to that date.

The information in the data of a single match contains numerous fields, many of which are of no interest for this project, including the actual match ID, the timestamp, and player names.

In order to extract properly formatted inputs and labels for the neural network, extraneous features were removed, the data was grouped into individual matches, and then the data was passed through a formatter which generated an input list and a label list to be used to train the neural network.

## Experiments

Each experiment run was conducted by training the neural network on the training data, while recording the accumulated error on each epoch. In order to evaluate the generalizability of the neural network, a test set was generated using unique match data, and its performance was evaluated.

| Epoch | Average Error |
|---|---|
| 0 | 0.500135986 |
| 1000 | 0.5 |
| 2000 | 0.5 |
| 3000 | 0.5 |
| 4000 | 0.5 |
| 5000 | 0.5 |
| 6000 | 0.5 |
| 7000 | 0.5 |
| 8000 | 0.5 |
| 9000 | 0.5 |

Test Error: 0.5

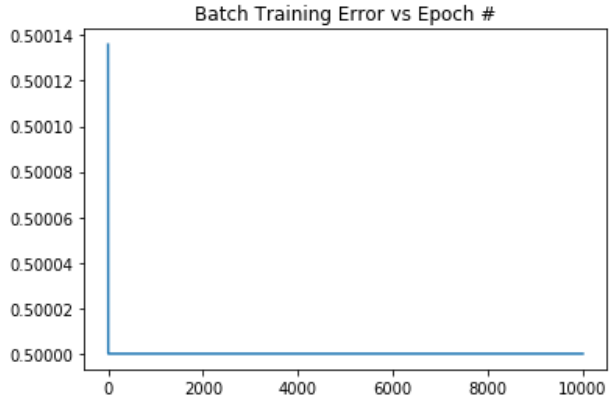


Figure 2: Training error vs epochs (α=0.1)

At first, a learning rate of 0.1 was used to train the network, but it was immediately clear that such a value led to ineffective learning. Changing the learning rate to 0.01 led to much more interesting results.

As can be seen in Figure 3, the performance of the neural network quite rapidly converges to its final value. Using a smaller epoch count of 10,000 allows for closer examination of the trend.

With the finer focus allotted by using 10,000 epochs, it is immediately apparent that there is considerable fluctuation in the error between around 1000-2000 epochs. Furthermore, there seems to be some evidence of over-fitting, as the test-data performance is better when using 10,000 epochs compared with 100,000 epochs.

| Epoch | Average Error |
|---|---|
| 0 | 0.500135986 |
| 10000 | 0.0123929 |
| 20000 | 0.010384332 |
| 30000 | 0.009354201 |
| 40000 | 0.008947196 |
| 50000 | 0.008711125 |
| 60000 | 0.008548733 |
| 70000 | 0.008427826 |
| 80000 | 0.008333386 |
| 90000 | 0.008258362 |

Test Error: 0.220432991441

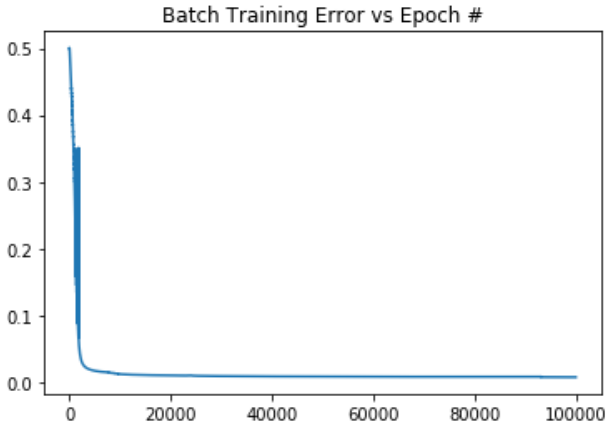


Figure 3: Training Error vs epochs (α=0.01)

In studying the implementation of neural networks, one topic that often appears is the choice between using batch training versus online training. Put simply, online training involves training the network by updating the weights after each sample is processed, while batch training updates the weights after a batch of samples has been processed, and the errors accumulated.

One source asserted that while both methods of training a neural network had their pros and cons, oftentimes online training led to less volatility during the learning process. In order to verify this claim, the training code was modified to use an online training process, as opposed to the full-batch training process which was used to obtain the previous results.

Of particular note is the fact that online training, or at least the implementation used in this project, runs exceptionally slowly compared to full-batch training.

| Epoch | Average Error |
|---|---|
| 0 | 0.500135986 |
| 1000 | 0.298929335 |
| 2000 | 0.047837974 |
| 3000 | 0.023876926 |
| 4000 | 0.019346975 |
| 5000 | 0.017334214 |
| 6000 | 0.016162342 |
| 7000 | 0.015390662 |
| 8000 | 0.014634876 |
| 9000 | 0.013372702 |

Test Error: 0.218421010529

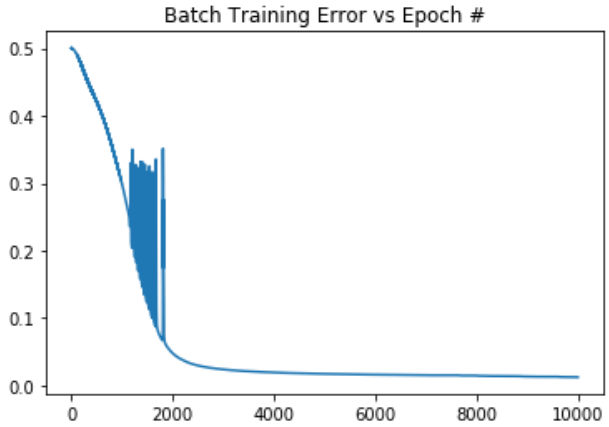


Figure 4: Training Error vs epochs (10,000 epochs)

For example, the 10,000-epoch full-batch training run took approximately 19 seconds to complete, while a mere 1,000-epoch online training run took over 150 seconds to complete. Epoch-for-epoch, online training runs about 80 times slower than full batch training.

We hypothesize that the major degradation in speed results from losing the ability to efficiently calculate matrix products, as is done in full-batch training. In full-batch training, the entire set of inputs can be updated as a full matrix, which allows for the use of numpy's various efficiency tricks, which with online training such methods are not possible.

Online learning makes up for the slowdown in epochs/second by greatly increasing the performance

of the resulting neural network and requiring fewer overall epochs. Even with only 1,000 epochs, the learning rate can be increased to 1.0, and though the runtime is significantly longer than full-batch training, the graph of average errors shows much less volatility, and the final performance on the test set is improved as well.

| Epoch | Average Error |
|---|---|
| 0 | 0.499440226 |
| 100 | 0.019243931 |
| 200 | 0.017896836 |
| 300 | 0.017462047 |
| 400 | 0.016933407 |
| 500 | 0.01626364 |
| 600 | 0.015866413 |
| 700 | 0.015729249 |
| 800 | 0.015153503 |
| 900 | 0.015009081 |

Test Error: 0.209446065804

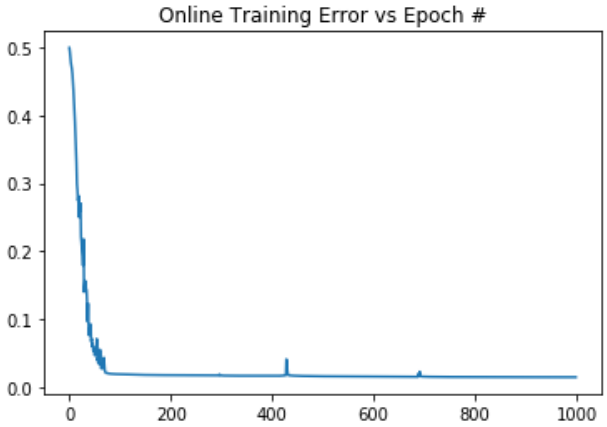


Figure 5: Online Training Performance

## Conclusion

Ultimately, the neural network trained in this project was able to correctly predict the outcomes of about 79% of matches in the test data set, based solely on which 5 heroes out of the 115 possible choices were drafted. Considering the massive amount of other potential factors involved in a match of Dota 2, including player skill, team communication, random game elements, etc., it is frankly astonishing that such a high correlation was found between merely the draft and a team's overall chance to win. This project was able to clearly show the pros and cons of online vs batch training for neural networks, with full-batch training being efficient, with decent performance, while online training is extremely slow, but more accurate. In fact, it is for this reason that many professional implementations of neural networks utilize a small-batch learning algorithm, which attempts to strike a balance between the two extremes.

One extension of the project which was attempted, but not included in this report due to poor performance, was to try to develop a neural network which could take as input four heroes, and output which hero would lead to the highest chance at victory. However, either due to poorly formed inputs or insufficient data, all attempts at learning such a network resulted in failure, with the network either randomly guessing or simply outputting zero for all heroes. Such a program would potentially allow not for simple retrospection on whether a 5-hero draft is likely to win or not, but to then extend into a predictive model that can help guide professional teams during a draft as they consider their remaining picks.