Deep Learning for League of Legends Match Prediction

Kenneth T. Hall
Department of Engineering Science and Mechanics
The Pennsylvania State University
`kih5283@psu.edu`

Abstract

Deep neural networks exhibit state-of-the-art performance on classification tasks ranging from face and object recognition, to EEG decoding, to stock market prediction. They are especially successful on tasks with an abundance of data to use for training.

We employ neural networks to predict which team will win a League of Legends match based upon the historical game play statistics of match participants. The performance of three neural network architectures is compared.

The most successful network achieves greater than 80% win prediction accuracy on the testing data. This network has multiple hidden layers and uses dropout, batch-normalization, and hand-engineered features in its input data. It outperforms a single-layer network and a medium sized network, even when the dropout and batch-normalization parameters of those networks are carefully tuned to achieve the best testing loss.

1 Introduction

Neural networks are a modern derivative of the perceptron networks originated by McCulloch and Pitts (Palm [1986]). Although neural network research has been active for decades, the most successful applications have emerged in the last several years. The increased computational power of GPUs is one factor behind the recent breakthroughs. The prevalence of massive data sets is another. In this work, we will apply neural networks to the problem of classifying the outcome of League of Legends matches before they are played based upon players' past performances.

Neural Networks A neural network is a graph of nodes and edges which transforms input data into a vector of output values. The networks are traditionally separated into layers, with each node in a layer connected by a weight variable to each node in adjacent layers. Each node, or "neuron", in the neural network graph is represented mathematically as adding a hyperplane to the state space of the input data. Then, the position of the hyperplanes is set through a training process called backpropagation.

In backpropagation, the output of the last layer of a neural network is compared to some desirable value, or "label". Some function of the difference between the network output and the label is calculated, called the "loss". Then, the gradient of all variables in the neural network is computed with respect to this loss value. All of the weight and bias variables are moved along the direction of their gradients. The size of this adjustment is controlled by a scaling factor called the "learning rate". Thus, each time a new input is fed to the neural network, all of the weights and biases are adjusted such that the output classification becomes more accurate. After repeating this procedure for many thousands of input data points, the network can learn features which allow it to perform a general classification task on inputs which it has not been "trained" on.

League of Legends In 2009, the company Riot Games released their first and only game. League of Legends, or LoL for short, quickly became the world's most popular video game (lol 2017). With over 100 million active monthly players, a vast and widely viewed competitive "e-sports" scene, and countless club and university teams around the world, Leage of Legends gained fame and players, eclipsing Blizzard Entertainment's Starcraft II as the premier online multiplayer battle game.

League matches pit two teams of five players against one another. They control a character known as a "champion", and cooperate with their allies to seize important objectives, collect gold, upgrade their champion by purchasing items, and kill the enemy team. To win the game, the team must advance across the map by destroying a sequence of defensive structures known as "turrets". Then, they must enter the enemy base, destroy one or more structures known as "inhibitors", and finally destroy the enemy "nexus".

Success in a League of Legends game requires skill, strategy, teamwork, and often a bit of luck. During the course of a match, each player takes thousands of actions which are recorded and compiled into a post-game report. Such reports are intended to allow players to compare their performance to that of the other players, as well as enable them to track their progress as their skill level advances. Riot Games stores hundreds of statistics about each player for every match that they play. These statistics are available to application developers through the Riot Developer API (Stats 2017).

2 Data Collection and Parsing

The most important factor in successfully training a neural network is whether a sufficient amount of labeled data can be compiled to use for training and testing. The League of Legends developer API provides access to player statistics from over one billion League of Legends games. The 1000 recently played games used in this study were accessed using a Python interface to the Riot API (RiotWatcher).

2.1 Match Data

Some of the game statistics available in match logs include:

- Kills - how many members of the enemy team a given player killed during the match.
- Deaths - how many times a player died during the match.
- Assists - how many times a player provided support or aided in killing an enemy, but did not deal the final blow.
- Total Damage - how much damage the player inflicted on enemy and neutral monsters, players, objectives, and structures.
- Total Damage Taken - how much damage the player received during the game.
- Damage to Champions - how much damage the player inflicted on members of the enemy team
- Physical Damage - how much of the damage done by the player was characterized as physical, or melee/auto attack damage (as opposed to magic damage or true damage).
- Magic Damage - how much of the damage done by the player was characterized as magical, or ability power damage, as opposed to physical damage or true damage.
- True Damage - how much of the damage done by the player was characterized as true damage. True damage is unique in that it is not affected by armor or resistance items the enemies may possess.
- Gold Earned - how much gold the player acquired.
- Gold Spent - how much gold the player spent.

Many dozens of additional statistics are available, and are employed by the neural networks trained in this paper. However, the above list gives a sufficient concept of the type of statistics which are available to use as inputs to the neural network.

2.2 Feature Engineering

One option when training a neural network is to provide all of the raw data available as input, and hope that the network is able to learn all of the features relevant to classification through backpropagation. Another option is to use expert knowledge of the problem domain to hand-engineer features that are likely to be useful indicators of data class. These pre-processed features are then provided as the input to a neural network for classification.

Both approaches are attempted in this work, and the results compared. A set of features were engineered by the author based upon his experience and knowledge of League of Legends. The procedure for collecting data for both approaches begins as follows:

1. For player X in match Y, submit an API call to request all of player X's past ranked games.
2. Collect all of the statistics from player X's ranked games, except for game Y. Store these in a matrix.
3. Repeat steps 1 and 2 for all other players in match Y. If the neural network is to be trained on raw data with no engineered features, stop here and concatenate all of the collected statistics into a single vector. Apply a label, "1" if the blue team won the game, "0" if the red team won the game.
4. For each player in match Y, compute the mean value of each game statistic over all of the ranked games he has played.
5. For each team in match Y, sum the averaged statistics of each player. Now, each team has a vector of values representing the past performance of its players.
6. Optionally, weight the statistics vectors for each team by eliminating those features which are not likely to be useful for win prediction (like damage to neutral monsters), and assigning a heavy weight to features that are likely to be highly useful (like kills).
7. Create weighted linear combinations of the features from step 6.
8. Concatenate all features from both teams into a single vector, and apply the label "1" if the blue team won the game, "0" if the red team won the game.

Note that the champion which a player selects for the match is *not* used to influence which data is collected from players' match histories. This is different than the approach taken in Huang et al. 2017, and in Lin 2016, where player win rate on the champion selected for the game is characterized as the most crucial feature for win prediction.

2.3 Data Used for Experiments

Included is a brief description of the data used for training the networks presented in this paper.

Data was collected for 1000 matches of League of Legends occurring between Nov. 12 and Nov. 13. These matches were separated into 800 training samples, and 200 testing samples. For each of these 1000 matches, statistics are collected for each participant's last 20 ranked games. It would be preferable to gather statistics from the entire ranked match history for each player, but Riot Games imposes a limit on the number of API calls that can be made per minute which makes this impractical given that many players have completed hundreds of ranked games.

Ranked matches are League of Legends games where players compete to move up a ladder which rates their performance against that of other players over the course of a "season", which generally lasts several months. Data is collected only from ranked games, as it is in this type of match that League of Legends players typically give their best effort to win the game. Unranked, "normal" games are more susceptible to player behaviors like disconnecting while the game is in progress, experimenting with unconventional strategies, and using champions that are unfamiliar. All of these factors might obfuscate features relevant to winning League of Legends matches when players are giving their maximum effort. Therefore, our analysis of match history is limited to ranked games.

3 Network Architectures

Three neural networks are compared for the binary classification win prediction task. Architecture 1 has a single layer with 50 neurons. Architecture 2 has three hidden layers with 20, 10, and 6 neurons. Architecture 3 is the largest network, with 200, 100, and 60 neurons in 3 hidden layers. The raw feature input has 2000 features, or 200 from each player. Hand engineered input vectors have only 9 features. The full, raw feature input vectors are used for the purposes of the network notation that follows. The style of notation is that of Xu [2017].

Network Parameters These parameters apply to network architectures 1, 2, and 3. All networks include both dropout and batch normalization.

Parameter	Value
Learning Rate	.0001
Dropout Ratio	0.5
Batch Size	85
ϵ (batch normalization)	.001

3.1 Network 1

This network has a single hidden layer with 50 neurons.

First fully connected layer:

$$f^0 = \theta^0(x) \in \mathbb{R}^{50}$$

with $\theta^0(x) = W^0(x) + b^0$, $W^0 \in \mathbb{R}^{2000x50}$, $b^0 \in \mathbb{R}^{50}$

Second fully connected layer:

$$f^1 = \theta^1(g^1(f^0)) \in \mathbb{R}^2$$

with $g^1 : \mathbb{R}^{50} \to \mathbb{R}^{50}$ being the ReLU activation function and
$\theta^1(x) = W^1x + b^1$, $W^1 \in \mathbb{R}^{50x2}$, $b^1 \in \mathbb{R}^2$

3.2 Network 2

First fully connected layer:

$$f^0 = \theta^0(x) \in \mathbb{R}^{20}$$

with $\theta^0(x) = W^0(x) + b^0$, $W^0 \in \mathbb{R}^{2000x20}$, $b^0 \in \mathbb{R}^{20}$

Second fully connected layer:

$$f^1 = \theta^1(g^1(f^0)) \in \mathbb{R}^{10}$$

with $g^1 : \mathbb{R}^{20} \to \mathbb{R}^{20}$ being the ReLU activation function and
$\theta^1(x) = W^1x + b^1$, $W^1 \in \mathbb{R}^{20x10}$, $b^1 \in \mathbb{R}^{10}$

Third fully connected layer:

$$f^2 = \theta^2(g^2(f^1)) \in \mathbb{R}^6$$

with $g^2 : \mathbb{R}^{10} \to \mathbb{R}^{10}$ being the ReLU activation function and
$\theta^2(x) = W^2x + b^2$, $W^2 \in \mathbb{R}^{10x6}$, $b^1 \in \mathbb{R}^6$

Fourth fully connected layer:

$$f^3 = \theta^3(g^3(f^2)) \in \mathbb{R}^2$$

with $g^3 : \mathbb{R}^6 \to \mathbb{R}^6$ being the ReLU activation function and
$\theta^3(x) = W^3x + b^3$, $W^3 \in \mathbb{R}^{6x2}$, $b^1 \in \mathbb{R}^2$

3.3 Network 3

First fully connected layer:

$$f^0 = \theta^0(x) \in \mathbb{R}^{200}$$

with $\theta^0(x) = W^0(x) + b^0$, $W^0 \in \mathbb{R}^{2000x200}$, $b^0 \in \mathbb{R}^{200}$

Second fully connected layer:

$$f^1 = \theta^1(g^1(f^0)) \in \mathbb{R}^{100}$$

with $g^1 : \mathbb{R}^{200} \to \mathbb{R}^{200}$ being the ReLU activation function and
$\theta^1(x) = W^1x + b^1$, $W^1 \in \mathbb{R}^{200x100}$, $b^1 \in \mathbb{R}^{100}$

Third fully connected layer:

$$f^2 = \theta^2(g^2(f^1)) \in \mathbb{R}^{60}$$

with $g^2 : \mathbb{R}^{100} \to \mathbb{R}^{100}$ being the ReLU activation function and
$\theta^2(x) = W^2x + b^2$, $W^2 \in \mathbb{R}^{100x60}$, $b^1 \in \mathbb{R}^{60}$ Fourth fully connected layer:

$$f^3 = \theta^3(g^3(f^2)) \in \mathbb{R}^2$$

with $g^3 : \mathbb{R}^{60} \to \mathbb{R}^{60}$ being the ReLU activation function and
$\theta^3(x) = W^3x + b^3$, $W^3 \in \mathbb{R}^{60x2}$, $b^1 \in \mathbb{R}^2$

3.4 Dropout

When the number of parameters in a neural network is larger than the amount of input data, the network is highly susceptible to overfitting. This occurs when a neural network learns features specific to individual data points which may not generalize well to other instances of the class.

One method of reducing overfitting is using dropout. In a neural network with dropout, at each training step some fraction of the neurons are ignored during backpropagation; their weights and biases are not modified. Dropout is equivalent to training an ensemble of many neural networks, all of which have some shared weights. This helps prevent overfitting by reducing the repeated exposure of weight and bias variables to the same training data points.

3.5 Batch Normalization

After each layer of the neural network, the output is normalized. The parameters of the normalization are *learned*. They are subjected to backpropagation. This step ensures that the interesting part of the neuron activation function is used.

4 Results

Architecture 2 demonstrated the best performance, while architecture 3 was slightly better than architecture 1. The baseline accuracy (how well one might do by guessing one class for every input data point) on all testing and training data was near 50-51%. The precise values are included in a table at the end of this section.

For all networks, training and testing loss and accuracy will be presented. Training statistics are represented in blue, while testing statistics are represented in red.

4.1 Network 1

One hidden layer, with 50 neurons.

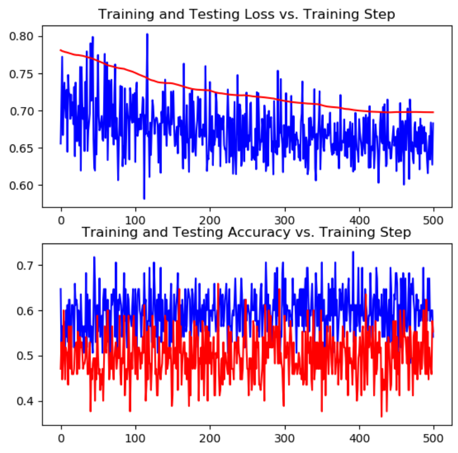

Figure 1: Architecture 1 Performance on Raw Input Data

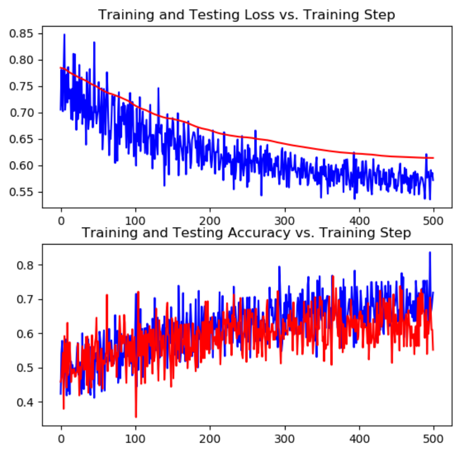

Figure 2: Architecture 1 Performance on Engineered Input Data

The performance of this network on the raw data is poor, with a mean testing accuracy of around 49.2% after convergence. On engineered features, the testing accuracy reaches a mean after convergence of 59.8%.

4.2 Network 2

Three hidden layers, with 20-10-6 neurons.

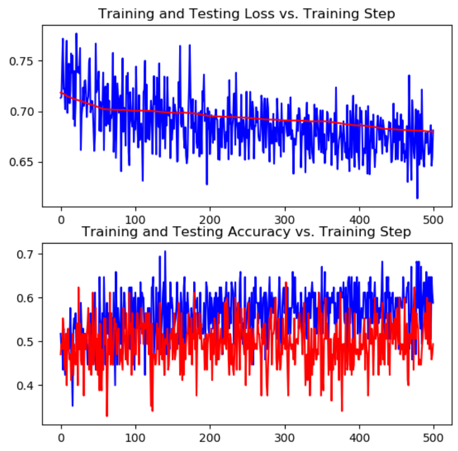

Figure 3: Architecture 2 Performance on Raw Input Data

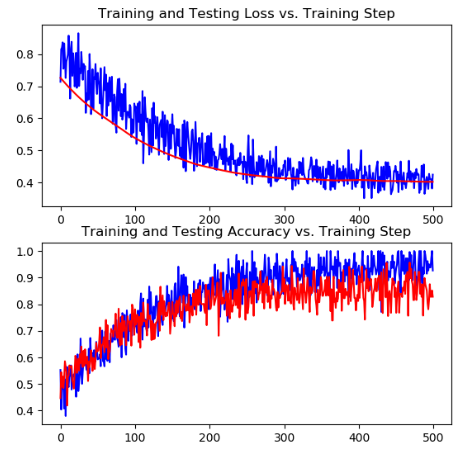

Figure 4: Architecture 2 Performance on Engineered Input Data

The performance of this network on the raw data is poor, with a mean testing accuracy of around 50.1% after convergence. On engineered features, the testing accuracy reaches an impressive mean of 84.4% after convergence. This value indicates that the network has learned features which provide an impressive ability to predict which team will win a match.

4.3 Network 3

Three hidden layers, with 200-100-60 neurons.

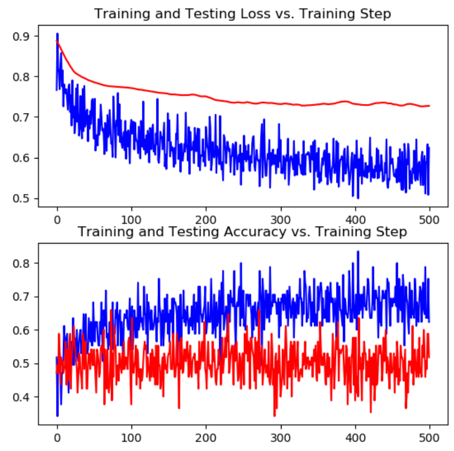

Figure 5: Architecture 3 Performance on Raw Input Data

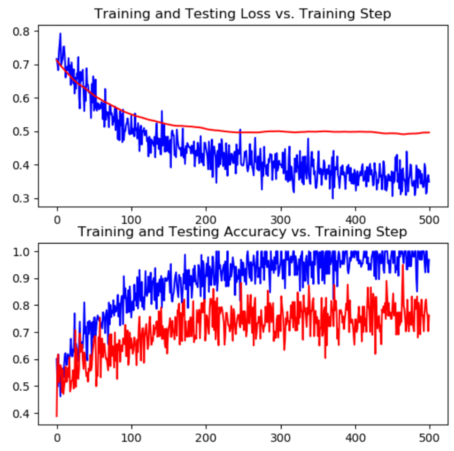

Figure 6: Architecture 3 Performance on Engineered Input Data

The performance of this network on the raw data is poor, with a mean testing accuracy of around 48.6% after convergence. On engineered features, the testing accuracy reaches a mean of 70.2% after convergence. The network did learn useful features for win prediction, but was hindered by severe overfitting. This is expected, as the number of free parameters in the network drastically exceeds the amount of training data available.

Table of Win Prediction Accuracies

Architecture	Raw Train (%)	Raw Test (%)	Engineered Train (%)	Engineered Test (%)
1	59.7	49.2	61.1	59.8
2	55.6	50.1	89.7	84.4
3	65.1	48.6	95.2	70.2
baseline	51.3	50.6	51.3	50.6

5 Discussion and Open Problems

Riot Games may need to adjust their matchmaking algorithm. The 84.4% accuracy on testing data for network architecture 2 is a very strong performance. League of Legends players depend on fair matchmaking which gives them and their opponents an equal chance of winning a game. The fact that a neural network can predict which team will win a match with such high accuracy should be a concern. The matchmaking system used by Riot Games has access to the same (or more) information used to train this neural network. While Huang et al. relied on knowledge of which champion a player had selected (information not available to Riot's matchmaking algorithm), the networks in this paper have no such limitation.

Feature engineering is very important to successful League of Legends win prediction. The accuracy of all three network architectures on raw input data was poor, never exceeding even the baseline accuracy of 50.6% which could be achieved by simply guessing that the blue team would win every match. It is a known problem that the blue team wins slightly more often than the red team. The reasons for this are unclear, but Riot Games has purportedly attempted to fix the problem by increasing the skill level of players on the red team compared to the blue team. Using engineered features introduces expert human knowledge about the game into the network. The increase in win prediction accuracy when engineered features are employed suggests that the author was able to craft highly useful high-level features from linear combinations of raw input features. This simplifies the network's learning task. Mathematically, the energy landscape of the loss function is altered such that a better minimum becomes accessible through stochastic gradient descent.

Unsolved Problems and Improvements This paper only used data from ranked games. The motivation for this exclusion was to eliminate "trolling" behaviors and non-competitive factors which affect win probability. However, it is likely that useful information can be extracted from unranked games if features are hand-crafted. Expanding the player history examination to include data from other game modes is one possible extension of this work.

After a neural network has been trained, certain input nodes gain larger influence over the final output than others. Examining which input nodes have become the most important might lead to identifying player characteristics which most contribute to win probability. Such a result would be of great interest to players of League of Legends who wish to have data-driven goals for improving their strategy and behavior.

Finally, acquiring additional training data might further enhance the performance of the neural networks in this report. Of the billions of League of Legends matches available to train on, only 1000 were used. This could be a point of criticism of this work. The rate limit imposed by Riot Games' API sets an upper limit on how rapidly additional data can be acquired. If Riot Games were willing, they could provide additional training data which might boost the win prediction accuracy presented in this paper further.

References

G. Palm. Warren mcculloch and walter pitts: A logical calculus of the ideas immanent in nervous activity. *Brain Theory*, page 229–230, 1986. doi: 10.1007/978-3-642-70911-1_14.

League of legends overview. URL `https://na.leagueoflegends.com/en/`.

LoL Stats. Stats straight from the source. URL `https://developer.riotgames.com/`.

RiotWatcher. Riotwatcher api. URL `http://riot-watcher.readthedocs.io/en/latest/`.

Thomas Huang, David Kim, and Gregory Leung. League of legends win prediction. URL `http://thomasythuang.github.io/League-Predictor/abstract.pdf`.

Lucas Lin. League of legends match outcome prediction. URL `http://cs229.stanford.edu/proj2016/report/Lin-LeagueOfLegendsMatchOutcomePrediction-report.pdf`.

Jinchao Xu. *Math 597 Notes - Deep Learning*. 2017.