/ff @ 20:
Competitive Gaming Analysis in League of Legends

Benjamin M. Barnett
Department of Computer Science
Stanford University
ben.barnett@stanford.edu
https://github.com/benbarnett3/cs230-lol-predictor

Abstract

This study aims to investigate what minimum information is necessary to accurately predict the outcome of a competitive match of League of Legends (“LoL”). As esports and competitive gaming continues to grow internationally, the demand for prediction algorithms and “realtime” match analytics (before a game has finished) will likely soon mirror those of basketball, football, and other popular sports. Using an LSTM-RNN that was fed temporal structure information as well as team gold difference information for every minute in a match, the final model achieved approximately 80% accuracy in predicting the outcome of a game given 20 minute-encodings.

1 Introduction

LoL is an online, competitive, team-based game made by Riot Games with over 80 million monthly players from over 145 countries, in which each team of 5 players destroy the enemies' Nexus before the enemy can destroy theirs [1]. During these 25-40 minute games, players accumulate gold, kill enemy "champions," spend gold to make their champion more powerful, and destroy turrets (a.k.a. "objectives") that defend the enemy Nexus.

As mentioned above, esports and competitive gaming has continued to grow over the past decade across all continents, increasing the demand for prediction algorithms and “realtime” match analytics (before a game has finished). Furthermore, the results of this study could provide insight for professional coaches in terms of knowing what in-game objectives and priorities should be focused on most heavily, potentially altering strategies, player behavior, and the “meta” of League of Legends. For example, the trained model could indicate that certain in-game objectives are more likely predictors of success, so coaches would encourage their players to focus on those objectives.

In the final model, the input of the RNN at each time step is an encoding of the corresponding (single) minute during the same match. For example, the first time step input is an encoding of a game's first minute, the second time step corresponds to the game's second minute, and so on. Each minute-encoding contains information about (a) the gold difference between the two teams, and (b) which objectives on the map were destroyed during that minute, if any. The output at each time step is a binary value indicating which team is predicted to win the match.

2 Related work

Several other studies have attempted to predict the outcome of a game given information from before the game starts, such as the champions selected on each team, and various metrics on the players' abilities on each team (i.e. kill-death-assist ratio, win rate, etc.) using a variety of learning algorithms including decision trees, Naive Bayes, and multilayer perceptrons [5] [6] [7]. These project, however, often have small sample sizes (< 2000 matches of data), and even still get around 55-58% accuracy in correctly predicting the outcome of a game before it starts, which is barely better than randomly guessing. (In defense of these studies, Riot, the company that makes League of Legends, tries to keep the game fair and balanced, which means the outcome of the game should not be easily predictable before it starts, or else certain champions are too strong/weak, or players are not playing against opponents of equal skill level.)

Furthermore, other League related deep-learning studies take more holistic behavioral approaches such as "Ping to Win?..." which investigates the effects of non-verbal communication on the outcome of games of LoL [8], while still other studies like DeepLeague have leveraged computer vision and convolutional neural nets to predict the locations of players in game given images of the game's minimap [9].

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

However, as far as I am able to find, this project is the first of its kind in terms of making predictions about the outcome of game during the game. Although this project uses no apriori information about the game like the players on each team or the champions they are playing, it is likely that such information input to a more complex model that combines the final model discussed in this paper with models used in the aforementioned studies that incorporates this apriori information as well would result in better in-game prediction accuracy.

3 Dataset and Features

All data used was extracted from a Kaggle dataset published by Chuck Ephron, which contains comprehensive match information from approximately 7600 challenger-tier competitive matches from 2015 to 2018 [2]. Of these, approximately 7000 matches were used for training, while the remaining 600+ were split into a development for tuning each model, and a test set of 300 matches to test the model's final accuracy. The data-point of a single match contains information including the gold difference between teams at every minute mark until the game is completed, as well as the outcome of the game, times that each structure was destroyed, champions selected, kills at each minute, and more.

From the dataset above, I extracted the gold difference (a scalar value) between teams at 1 minute increments until 20 minutes, as well as the times that structures were destroyed rounded to the nearest minute mark (with the exception of the baseline model described in the following section). To encode temporal structure elimination, a 28-vector of boolean values was assigned to each minute mark of the game, wherein each index corresponds to a single structure (see Fig. 1), and a value of "true" means that structure was destroyed at approximately that minute mark. Due to the way the dataset was organized and separated into multiple files, preprocessing involved matching rows of data across several csv files by match id, which is unique to every match, and combining all information about a single match into a single data structure before input into the model.

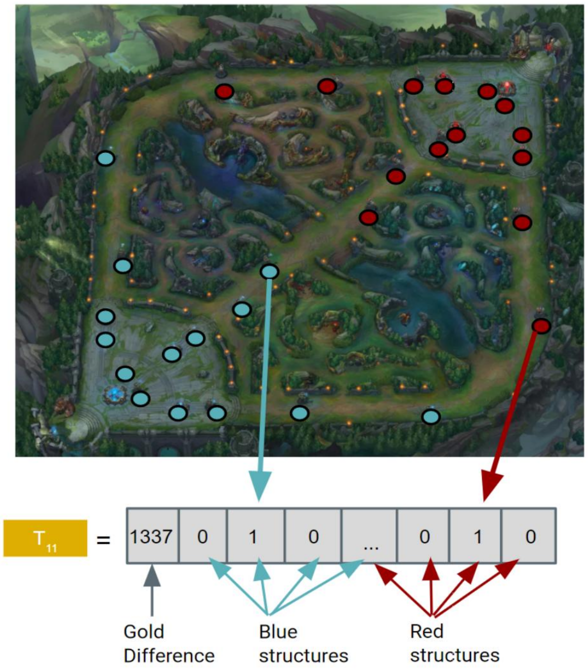


Figure 1: Map of Summoner's Rift with example minute encoding

4 Methods

Because this project's primary goal was to determine what minimum information is necessary to accurately predict the winning team in a game of LoL, I experimented with several models as I changed the information contained within the input passed to the models.

4.1 Model 1

The first "baseline" model tested was a simple feed-forward 3-layer NN with 50 hidden units in each layer. The model was fed an n-vector in which each value in the vector corresponds to one minute of game information, or more specifically, the gold difference between teams (an integer) at each minute mark. Note that although this value technically has no bounds, realistic values for this gold difference range from -30k to + 30k. In order to test the accuracy of predictions made at each minute mark from 1 minute to 20 minutes, technically 20 'mini baseline' models were tested as described above, with $n \in \{1, ..., 20\}$. Output of each previous layer was passed through a ReLU activation, with the exception of the final output $\hat{y_i}$, which was passed through a sigmoid activation. During training, this output was used to minimize the cross entropy loss function:

$$loss = -\sum_i y_i \log \hat{y_i}$$

Note however that at test time, values were rounded after being passed through the sigmoid function to make the final prediction, where 0 represented a win for one team, and 1 represented a win for the other.

4.2 Model 2

The second model tested was identical to the baseline model in every way except for the model's input. Instead of soley inputing the team gold differences at each minute mark, each minute-encoding—all of which were concatenated before being input to the model, as in the baseline model—contained an embedded representation of which structures/objectives in the game had been destroyed during that minute, if any (see Fig. 1).

4.3 Model 3

Unlike the first two models, the third and final model consists of a 1-layer RNN with LSTM cells in which, at each time step of the RNN, the sequential minute-encoding from the game is fed to the model (see Fig. 2). The LSTM cell has a hidden-unit size of 128 and uses a sigmoid activation function; more information on Tensorflow's implementation of the LSTM cell can be found at [4]. An RNN is a natural choice for this problem because there are literal time steps during a game of LoL which are to be used for predicting the outcome of the game given information up until a certain time step. Furthermore, an LSTM cell was chosen for its ability to preserve information from many previous steps—something that seemed extremely helpful for predicting the outcome of a game because it effectively allows the model to "remember" which structures in total have been taken, despite only receiving information about which structures were destroyed at the current time step and the current gold difference. In regards to the gold difference, this sort of memory preservation may also be helpful for telling if one team is losing their lead; for example, if the gold difference was 5k at 10 minutes, but is now -1k at 15 minutes, it is interpreted significantly differently than if the gold difference were -5k at 10 minutes. In other words, remembering the team gold differences and structures that have been destroyed at previous time steps contextualizes the current time step, making the model's prediction more accurate.

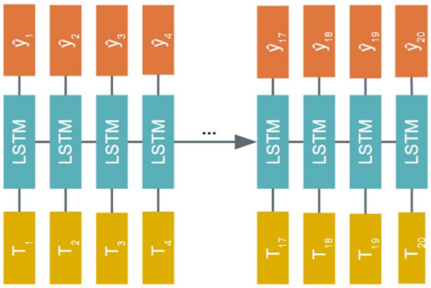
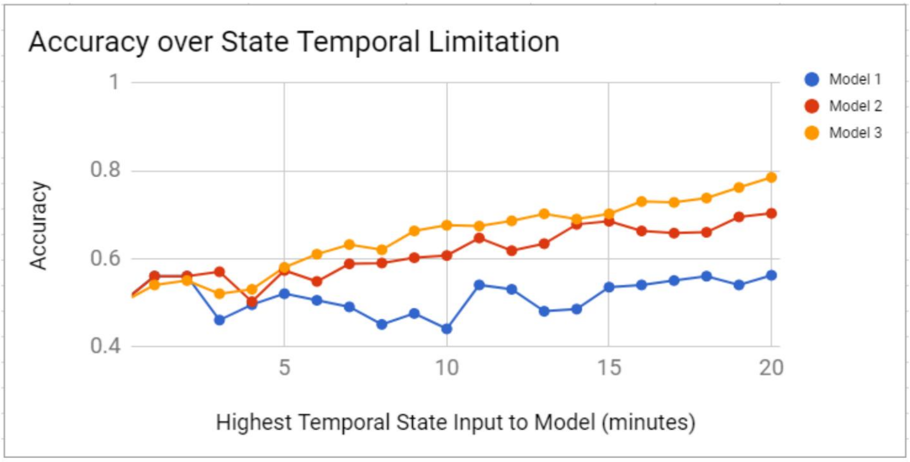


Figure 2: Visual representation of third and final model architecture, using 128-hidden-unit LSTM cells and cross-entropy loss.

5 Experiments/Results/Discussion

The following figure shows the accuracy of predictions on the test set made by each model as the number of minutes of information fed to the model increases. For example, the rightmost points show the accuracy of each model when input all 20 minutes of information. (Note that what the information is as well as how it is fed to the model is dependent on the specific model architecture.) For reference, Model 1 is the baseline NN with only team gold differences, Model 2 is the same as the baseline NN but given team gold differences as well as times structures were destroyed during the game, and Model 3 is the final model which uses an LSTM/RNN.



Time (minutes)	Model 1 Accuracy	Model 2 Accuracy	Model 3 Accuracy
0	0.5	0.5	0.5
1	0.56	0.56	0.541
2	0.56	0.56	0.552
3	0.46	0.57	0.52
4	0.495	0.502	0.53
5	0.52	0.573	0.581
6	0.505	0.548	0.61
7	0.49	0.588	0.632
8	0.45	0.59	0.62
9	0.475	0.602	0.663
10	0.44	0.607	0.676
11	0.54	0.647	0.674
12	0.53	0.618	0.686
13	0.48	0.634	0.702
14	0.485	0.678	0.69
15	0.535	0.685	0.702
16	0.54	0.663	0.73
17	0.55	0.658	0.728
18	0.56	0.66	0.738
19	0.54	0.695	0.762
20	0.562	0.703	0.795

Figure 3: Accuracy for different models as the highest temporal game state is limited from 1 to 20 minutes.

Due to the relatively small size of the training data set, stochastic gradient descent was not necessary; all examples were used in each epoch of gradient descent for all 3 models. Furthermore, all models used Tensorflow's implementation of an Adam optimizer, as it has been shown to be a high performing optimizer that uses both Root Mean Square Propagation (RMSProp) as well as the Adaptive Gradient Algorithm (AdaGrad), which specifically has shown to improve performance on problems with sparse gradients—a benefit that suits this problem well given the sparsity of each minute-encoding with likely 0-2 structures ever being destroyed during a single minute [10]. Each model's hyperparameters were chosen independently through quick iterations of various hyperparameters on the development set. However, due to the time constraint of the class and the final project, the hyperparameter search space was not as thoroughly investigated as would be ideal; it is possible that more searching would result in better performance across all models.

Model 1: $learning\_rate = 0.0001, num\_epochs = 3000$
Model 2: $learning\_rate = 0.0005, num\_epochs = 3000$
Model 3: $learning\_rate = 0.0005, num\_epochs = 2000$

Due to the relatively high performance of the third model, most analysis was on its performance. It is worth noting that although precision and recall are often helpful measures in analyzing classification problems (and were explicitly recommended for this final report!), it's important to recognize why they do not make sense in the context of this problem given the symmetric map and type of outcome; there is no distinction between true positives and false negatives, for example, because the outcome of the game is not true/false, but rather team 1 wins or team 2 wins. The assignment of 1 to team 1 (a.k.a. blue team) winning vs. 0 to team 2 (a.k.a. red team) winning was arbitrary. In contrast, the most helpful analysis besides the accuracy charts above are qualitative. Looking at individual cases where the model failed, it is often because of certain in-game events that occurred after the final minute mark of information the model receives; in other words, the model is able to tell when a team is winning, though it is unable to predict "throws" during the game, in which the winning team loses its lead. Although difficult to predict, these sorts of game-changers are often functions of the different champions chosen, as some perform better during different phases of the game; while a team of "early-game" champions may be winning up until the 15 or 20 minute mark, the other team's "late-game" team-composition may take the lead if the game continues long enough.

6 Conclusion/Future Work

The results of the final model using an LSTM-RNN imply that challenger-tier matches can be predicted with at least 79.5% accuracy by looking at the two teams' gold difference and times that objectives were taken up to the game's 20 minute mark. In other words, without even examining selected champions, kills, individual gold differences, or previous player history, an accuracy of around 80% was achieved. Furthermore, it is highly likely that an improved NN architecture or different hyperparameters could yield even better results. The simpler models may have also provided better results if some form of regularization were implemented, as overfitting is clear from the train/test accuracy discrepancy.

One other important fact to consider is that given how much variation in gameplay there could be in the following 5-25 minutes after the 20 minute mark, it may be impossible to make a prediction better than about 80% accuracy after 20 minutes; there is no definite outcome, because so much of the outcome of the game is based on the individual player behavior from that time until the game ends.

Given the success of this final model with such limited information, I would like to investigate the effects of adding other input features (apriori and temporal), such as champions selected (before the game starts), kills, individual players' gold differences, and the items players purchase during the game. I am also curious about predicting more than just the outcome of the game; for example, how accurately could a model predict the location and time of the next kill, or the next objective(s) to be taken? Lastly, the boasting accuracy of about 80% is only at the 20 minute mark, and while there should generally be no strong ability to predict the outcome of a game too early in the game, I am curious if other information would be able to increase the prediction accuracy at earlier time steps in the game.

7 Contributions

I completed all work independently.

References

[1]. "You'll Never Guess How Many People Play League of Legends." Unranked Smurfs. Oct. 4, 2017.
[2]. Ephron, Chuck. "League of Legends Competitive Matches, 2015 to 2018." Kaggle. Jan. 29, 2018.
[3] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. "TensorFlow: Large-scale machine learning on heterogeneous systems" (2015). Software available from tensorflow.org.
[4]. S. Hochreiter and J. Schmidhuber. "Long Short-Term Memory". Neural Computation, 9(8):1735-1780, 1997. http://www.bioinf.jku.at/publications/older/2604.pdf.
[5]. Thomas Huang, David Kim, Gregory Leung. "League of Legends Win Predictor". University of Northwestern, EECS 349-Spring 2015. http://thomasythuang.github.io/League-Predictor/
[6]. Yin, Jihan. "League of Legends: Predicting Wins in Champion Select With Machine Learning" (2018). UC Berkeley. https://hackernoon.com/league-of-legends-predicting-wins-in-champion-select-with-machine-learning-6496523a7ea7
[7]. Alexander Neumann, Joseph Clark, Alan Simon. "Developing a Model to Predict Match Outcomes in League of Legends" (2015). Arizona State University. https://repository.asu.edu/items/35623
[8]. Alex Leavitt, Brian C. Keegan, Joshua Clark. "Ping to Win? Non-Verbal Communication and Team Performance in Competitive Online Multiplayer Games" (2016). University of Southern California and Harvard University. http://delivery.acm.org/10.1145/2860000/2858132/p4337-leavitt.pdf
[9]. Majeed, Farzain. "DeepLeague" (2018). GitHub Repository: https://github.com/farzaa/DeepLeague
[10]. Brownlee, Jason. "Gentle Introduction to the Adam Optimization Algorithm for Deep Learning" (2017). https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/