CS 699 Project Final Report
DeepSkill: Win Prediction and Matchmaking Framework for Elite Individuals and Teams

James Enouen
Viterbi School of Engineering
University of Southern California
Los Angeles, CA 90089
enouen@usc.edu

Alexander J. Bisberg
Viterbi School of Engineering
University of Southern California
Los Angeles, CA 90089
bisberg@usc.edu

Abstract

This is our class project report for CSCI 699 - Topics in Deep Learning. In this project, we applied modern deep learning techniques for win prediction on the popular multiplayer video game: League of Legends (LoL). We utilized methods which can easily be extended to win prediction in other e-sports and for some methods sports in general.

1 Background and Motivation

League of Legends (LoL) is a very popular online multiplayer game in the genre "MOBA = Multiplayer Online Battle Arena." In 2020 the game has over 100 million players globally and the Worlds Finals peaked at nearly 4 million concurrent viewers. Consequently, predicting the outcomes of professional games and balancing competitive matchmaking are two very interesting topics for the online game. In the following work, we blend state of the art methods for win prediction and skill rating with the newest techniques in deep learning. We addressed win prediction from the following two main perspectives.

1.1 Professional Team Based Prediction

By focusing on high skill or professional teams, we believe we can make some simplifying assumptions to our model. Over a season, we assume a professional player's skill is static, therefore removing the time sensitive aspect of traditional skill models. Moreover, by utilizing a season of competitive games amongst a certain organization as data, we intend to be able to build a model which is able to predict the outcomes of these games and reveals some of the complex interactions between teammates who have played together over many games.

1.2 Player Based Prediction

The average player can also play in a competitive environment by queuing for ranked matches. In this case, a player no longer has a fixed team, but rather is placed with other players according to a matchmaking system. In our experiments, we focus on high-ELO players in the top 2% of the ladder because we believe their better game understanding will lead to more consistent results. We draw inspiration from TrueSkill [3], a Bayesian model for skill (as a replacement for Elo) and hope that by conceptualizing our framework as a network with similar structure we will have a theoretical basis for training our model.

Lan et al. [2] uses detailed game data including 'ability', 'item', 'gold', and 'subtype' to make win predictions in DOTA2 (a similar MOBA). Using this real time data gets their prediction accuracy all

Preprint. Under review.

the way up to 87.9% prediction accuracy after a decent way into the game (20 full combats). Closer to the beginning of the game (1 full combat), their model only achieves 50.6% accuracy (most likely because this was not the focus of the work.) Regardless, we primarily take the perspective of "0 full combats" and make predictions after the important "draft phase" which is where all the players of the MOBA choose the champions that they will play in the upcoming game.

1.3 Organization of Report

First, we go over a somewhat brief introduction to modern methods to evaluate skill and predict winners (which is not a necessary read but gives good context for the project.) In section 3, we talk about the two task and the two associated datasets. In section 4, we talk about the preliminary results we achieved. In section 5, we talk about the deep methods and their success. Finally, we conclude and mention future work in sections 6 and 7.

2 Overview of Skill Recognition and Win Prediction

2.1 ELO System

One of the first serious statistical models attempting to capture the skill level of its players was the 1950 Elo rating system developed by Arpad Elo for chess matches. The premise of the system is that two players have a single number identifying their overall skill level. The difference in skill between two players corresponds exactly to the probability each player will win, the larger the gap between the two players, the more likely the higher-rated player will win. This can be calculated exactly according to the following formula where two players A and B have ratings R_A and R_B:

$$E_A := Prob(\text{A wins}) = \frac{1}{1 + 10^{(R_B - R_A)/400}} = 1 - E_B$$

In this case, a player difference of 100 points means there is a 64% chance to win; a player difference of 200 points means there is a 76% chance to win; etc. The idea behind this equation is that each player's skill hover around a Gaussian (technically logistic) and each game they randomly draw from their distribution. Then the game is decided by which player draws a higher number from their distribution. Evaluating this random expectation leads exactly to the above equation.

Additionally, each player's skill can be changing or rating can poorly reflect their actual skill. In either case, every time a match is played between two players, both of their scores are dynamically updated to better reflect the outcome which was actually seen after their match. The scores update according to this equation where S_A denotes the actual score of player A (0=lose, $\frac{1}{2}$=draw,1=win):

$$R'_A = R_A + K(S_A - E_A)$$

Here we see there is another variable K which is referred to as the "K-factor". This variable describes how malleable the player's scores are. In chess, the standard was $K = 16$ for the masters players and $K = 32$ for the weaker players.

This rating system has been extremely popular since its inception and has been implemented throughout countless board games and online games. Practically all current day rating systems are in some way derivative of this model, including Glicko and Trueskill.

2.2 Microsoft's TrueSkill

Microsoft saw the need for better matchmaking for their 2004 title Halo 2. One of the key differences they needed to overcome was to assign skill for multiple players and competing teams. They borrowed the distribution ideas from ELO and extended their model to a larger Guassian Bayesian statistical model.

To introduce their model, let us first consider the ELO model in the same context as TrueSkill. In the following figure, on the left, we can see 5 nodes for the ELO model, where the top two nodes are the players skill (the ratings R_A and R_B from earlier), the next two nodes are the Gaussian sampled values of the actual game, the bottom node represents the actual outcome of the game.

We know from our previous discussion that the connection between skill and performance for each player is a normal distribution/ Gaussian sample. We also know that the winner is then determined by

an ordinal comparison between the performances. This ideology is extended in TrueSkill's model shown below. Again, the true skill is a Gaussian dependent on skill and skill variance; performance is a Gaussian dependent on true skill with a fixed variance. Team performance is taken as the sum of the individual members' performances. Finally, the results are compared and a ranking of all the teams is compiled (notably their structure works for more than two teams.)

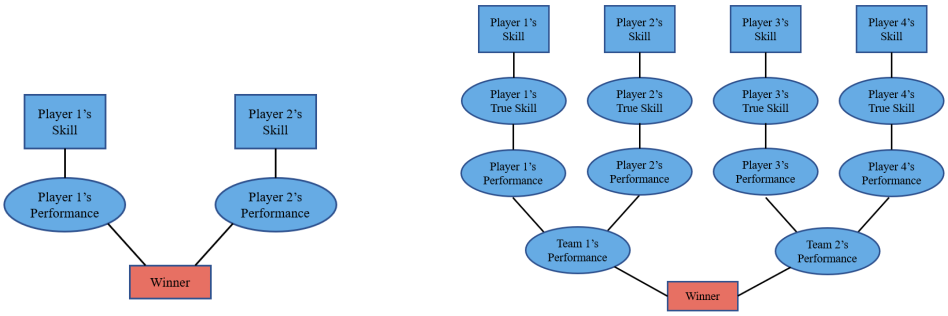

Figure 1: Bayesian Model for ELO rating (left) and TrueSkill rating (right)

This work was very important and has been built upon over the last two decades. In 2018 Microsoft published their second version of TrueSkill called TrueSkill2. Ultimately, many of the goals of the two algorithms are the same and the spirit is kept the same. The main difference is the additional game statistics which are included in the Bayesian modelling. In this way, a player's 'performance' in a given game can correspond more closely with their actual performance. The Bayesian model is depicted in the figure below.

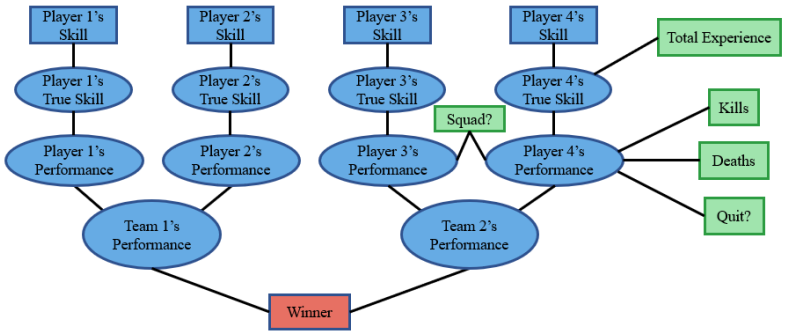

Figure 2: Bayesian Model for TrueSkill2 rating system

3 Dataset and Problem Formulation

3.1 Professional Teams

Our data source is a website that aggregates professional LoL games. The organizer, Tim Sevenhuysen, is a former professional analyst. They have data from 2014 to present.

For each game played in the professional leagues, this dataset contains the game version, year, teams, players, date, draft/ban phase, game length, winner, kills/deaths/assists, damage, wards, gold, etc. at the end of the game and also includes mid-game stats like gold and experience differentials at the 15 minute mark.

Year	Games
2020	5821
2019	6166
2018	6204
2017	5672
2016	4218
2015	1816
2014	918

Table 1: Number of games in each year of the dataset. Sources: Oracle's Elixir

3.2 Ranked Ladder

We also collected matches from the top 2% of the ranked ladder. We have two main hopes in this direction. First, to be able to transfer the knowledge we gain from high level games with professional players to their actual stage matches. Second, we want to be able to predict the outcomes of ranked matches before they conclude.

The dataset consists of 20,000 ranked matches occurring from August 5th to September 16th. The data was collected using the Riot Games API with a python wrapper [5, 6].

The data provided here was the draft information (player IDs, picked champion IDS, banned champion IDS); the overall winner; the team statistics (towers, inhibitors, dragons, rift heralds, and barons); and the individual player statistics (physical/magical/true damage taken/received, kills, deaths, assists, gold, creep score, crowd control score). In our formulation, we are trying to map from the draft information to the winner of the game. We utilized this extra information similarly to TrueSkill2 as we will further discuss later. The input to our models is depicted in the figure below.

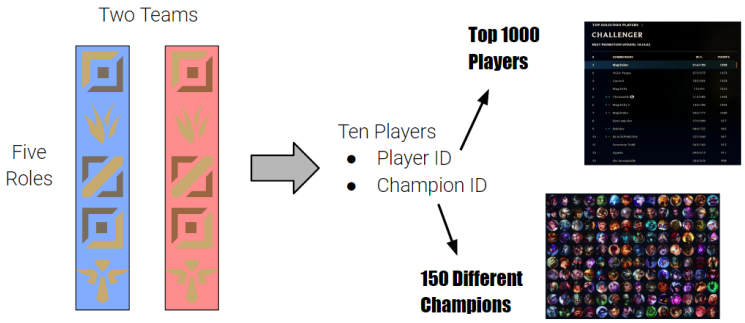

Figure 3: Represenation of draft information for ten players

4 Preliminary Results

4.1 Professional Teams

4.1.1 Feature Engineering

Each of theses datasets had data from each individual team member after each game, we extracted only team summary stats to start off with fewer features. After filtering and fitting for null values we ended up with around 30 features to describe a game. Next we determined that in order to predict a game at t_0, we couldn't use stats from that game, we would need to use data from the previous game (t_{-1}). Then again, there is likely more information than a team's performance in the last game alone. Likely there could be some added signal in the last N games a team plays. Therefore instead of looking only at the past game and predicting the next, we looked at the past 1, 3 and 5 games. However, It is unclear how these games should weighted. A random forest classifier does not have the capacity to account for time series data, so we just made a linear combination of features across

games with the most recent games waited stronger than the older games. Moreover, a team's *relative* performance may actually be more meaningful than their raw statistics in a game. Therefore, we compared using raw and relative features. In all cases we normalized features based on game time and then z-score normalized all features before training and testing.

4.1.2 Model Results

Lookback	1 game	3 games	5 games
Raw	0.563 ± 0.009	0.573 ± 0.010	0.572 ± 0.005
Relative	0.560 ± 0.009	0.569 ± 0.008	0.578 ± 0.012

Table 2: Win Prediction accuracy of random forest model

We used the vanilla Random Forest Classifier from the Sklearn Python library to produce these mdoels. In addition we performed 5 fold crossvalidation and pruned tree depth to prevent overfitting

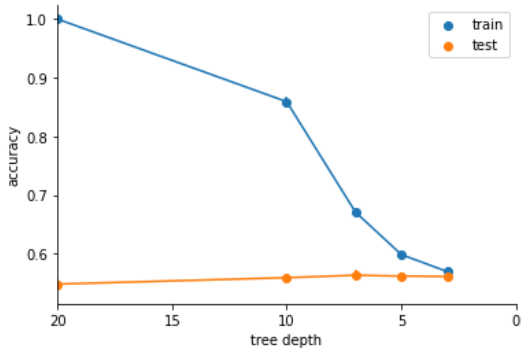

Figure 4: Tree depth for the best testing accuracy was usually between 5-7

Finally we observed the feature importance of the models using permutation feature importance to understand which were the important features for the model's prediction accuracy. Five games had higher prediction accuracy, but not by much. It does appear that smoothing out the past data from 5 games may have eliminated some of the more noisy features. For example, "double kills" are the third most important feature in the 1 game lookback model but these seem much less important than the other status surrounding them like "total gold" and "towers."

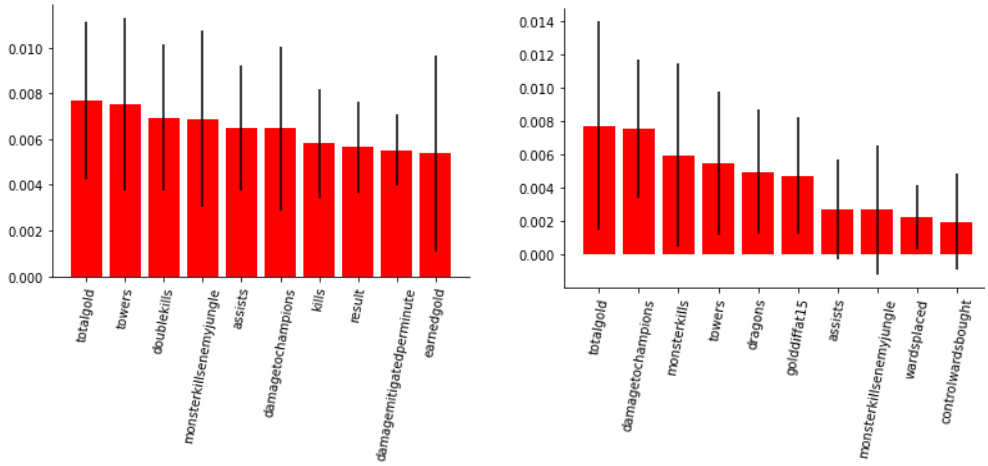

Figure 5: Permutation feature importance for 1 game (left) and 5 game (right) feature sets for the random forest model

4.2 Ranked Ladder

4.2.1 ELO Based-Gradient Logits

Using the ELO method of player skill as a one dimensional feature as discussed in the previous section. This approach has worked extremely well in chess for decades, however, in complex video games, it is clear that the myriad strengths and weaknesses of players can not be captured in one number. In our case, we trained the ELO values for the top 1000 players. Players outside this set were given a baseline zero rating. Training this method with gradient descent on the sampled matches ultimately yields around 51% validation accuracy.

4.2.2 Naive Bayes on Champions

Using only the champions selected at the very beginning of the game, this approach utilizes the historical win rates of either "synergies" (same-team pairs of champions) or "adversaries" (opposite-team pairs of champions). It then collects over all pairs in the game to estimate each team's percentage chance of winning.

In the figures below, it is very clear that this method is over-fitting to the training data. Even when normalizing by the thousands of games, the gap in performance between the training set and validation set remains quite large.

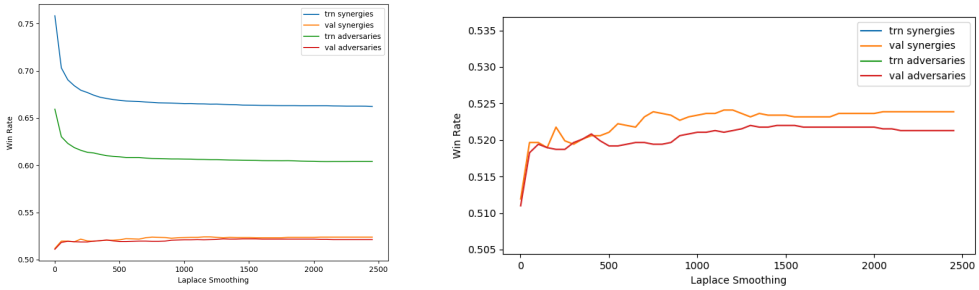

Figure 6: The two Naive Bayes methods on both the training set and validation set, while varying the normalization (left) full picture (right) zoomed

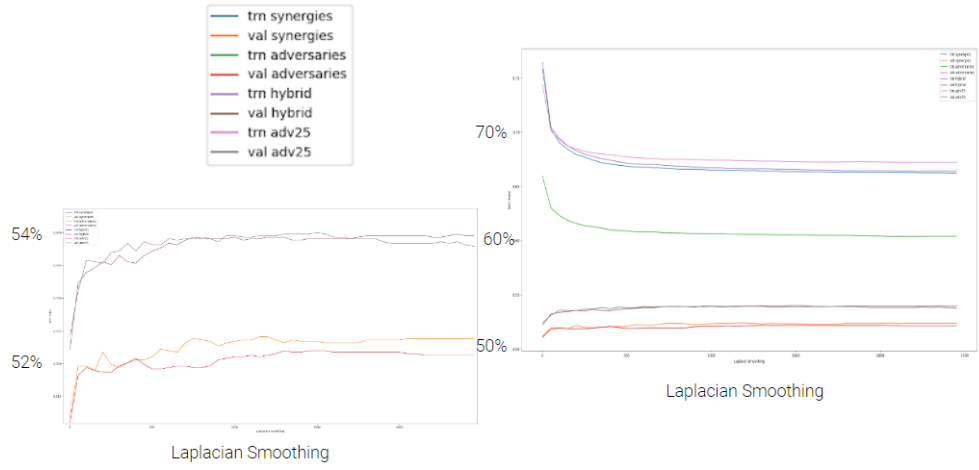

Figure 7: The two Naive Bayes methods on both the training set and validation set, while varying the normalization (right) full picture (left) zoomed

Using either synergies or adversaries alone yields around 52% validation performance, and using both simultaneously boosts performance up to 54% validation accuracy.

5 Deep Methods Results

5.1 Professional Teams

5.1.1 GCN-WP: Graph Convolutional Networks for Win Prediction

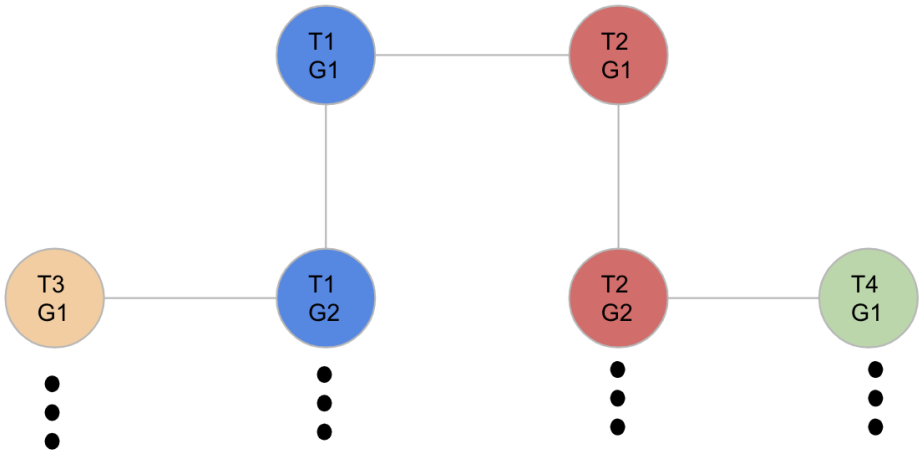

Figure 8: Part of a GCN-WP Graph. Each node represents a team-game (one instance of a team playing a single game) and is connected to that team's opponent and their previous game.

5.1.2 Constructing the Network

One important contribution of this project is formulating win prediction as a graphical neural network (GNN) representation. Kipf's semi supervised learning on graph convolutiuonal networks was an appropriate starting point to conceptualize this problem [1]. Given that most play at a professional level happens *within* a league, it is fitting to represent each league as a discrete graph. In addition, teams do change within the season, but most changes to the players happens *between* seasons. So our goal was to use graphs from a single season to learn a feature representations for win prediction and the apply that trained structure to the other leagues. Therefore this ended up being an inductive graph classification problem. Each of the nodes had the same features and labels that we used in the random forest setup. Although this may not work for every sport/esport this process is generalizable with some consideration about the league structure.

League	Games
LPL	725
LCK	473
VCS	304
PCS	264
LCS	264
LEC	241

Table 3: Number of games in 2020 for each league. Sources: Oracle's Elixir

For the sake of simplicity we focused on the 6 major regions. The Chinese league, LPL, had the most games played during the season by a factor of two, so we decided to fix this as our training network, validate on the LCK which had the second most games, and test on the rest of the four leagues. Using semi-supervised learning is beneficial because we don't have to throw out the last game each team plays – we don't know the result of their next game, because there isn't one. One modeling assumption of Kipf's GCN is that the network in homogeneous. Currently there is no

difference between self past game nodes and opponent nodes. We modified the code from the GitHub repo mentioned in [1] to build the networks tested in the following section.

5.1.3 Results

Model	Accuracy
Best RF	0.578 ± 0.012
Best GCN + delta	0.529 ± 0.009
Best GCN-cheby + no delta	0.533 ± 0.016
Best GCN-cheby + delta	$\mathbf{0.676 \pm 0.036}$

Table 4: Number of games in each year of the dataset. Sources: Oracle's Elixir

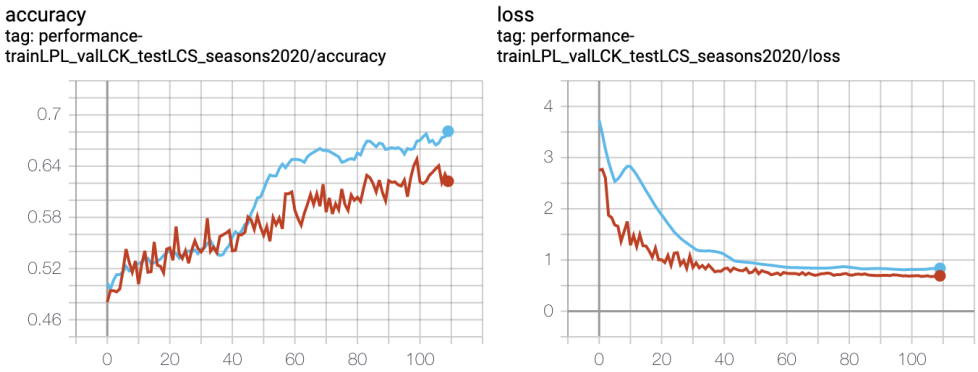

Figure 9: Best GCN-cheby training accuracy and loss. We did not have significant overfitting in this model

We performed several rounds of cross-validation to determine the optimal features and model parameters. We tested the number of nodes in the hidden layer(s) $(8, 16, 32)$ and the dropout rate $(0.25, 0.5, 0.75)$ after some initial explorations showed that the base values were optimal for our dataset. It was interesting to see that using Chebyshev polynomials, although increasing compute time, significantly increased model accuracy.

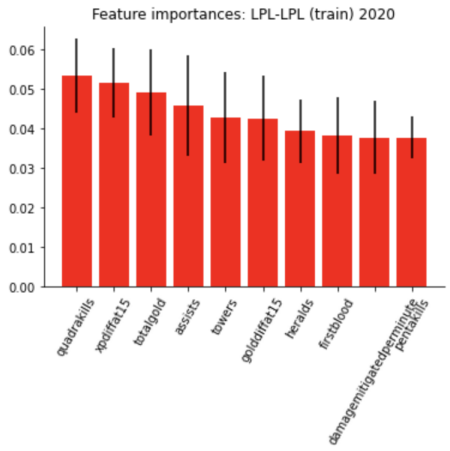

Figure 10: Best GCN-cheby training feature importance.

This was an interesting results especially since the original paper this method actually performed slightly worse. Ensuring they were set to 1 degree meant that our network shouldn't be integrating future game data. Given then feature representation of this data is directly Euclidean, as opposed to

word vector embeddings used in the paper, the technique of spectral convolutions likely has a more meaningful impact on the analysis of a node's neighbors. Given the results of permutation feature importance (Figure 10), we didn't feel the need to remove or reduce any of the features in the training set.

Some of the assumptions of the GCN framework limited our network structure in this problem context. For example, a graph convolution integrates data from the immediate neighborhood of a node. However, this graph is not directly encoded to be directional (although the adjacency matrix is asymmetrical). Therefore, if we have more than one graph convolution, we could effectively be allowing our network to look in to the future. To exemplify this, this Figure 9 shows that a two layer network has much higher prediction accuracy for a one-game look-ahead. A "fair" prediction for a two layer GCN would actually be predicting two games ahead. Looking further into the future further reduces the accuracy of this network, but is still better than random. This is why we held the number of hidden layers fixed to one for our experiments.

5.2 Ranked Ladder

5.2.1 Network Structure

We explored an abundance of different structures and techniques within deep learning to try to boost performance on this prediction problem, but ultimately our network structure fall into two main categories depicted in the following figure. One of them uses the full connection structure of a typical deep neural network; the second uses a special connection structure which only connects players with opposing roles. (implemented in Pytorch[4])

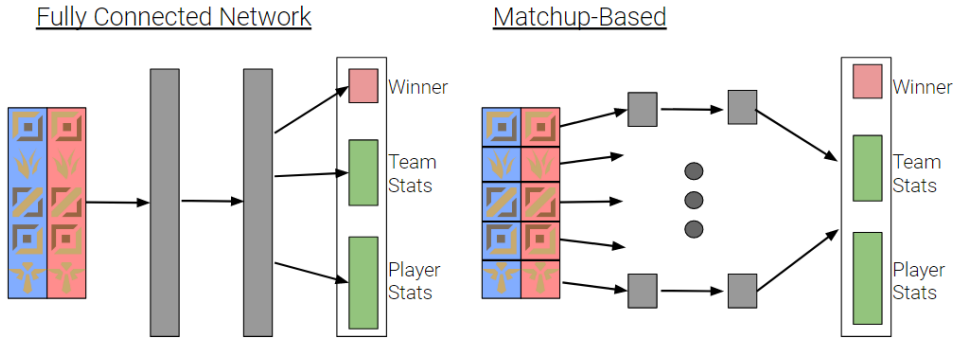

Figure 11: The two main network architectures used in our experiments (left) fully connected (right) partially connected by role

5.2.2 Results

Below we can see an example of a training curve for a deep neural network we used in this project. Below that we can see the table of final results. For the three architectures we use, we report both the single-model accuracy as well as the ensemble model test accuracy. We managed to get all the way to 60% accuracy which is a very exciting result. Although in the context of deep learning, it may seem like a low result, but we should consider this accuracy is in the context of the win prediction. The matchmaker is trying to make even games which hover around 50% win chance for each side, so the win should be inherently random which we discuss this more in the next section. Even the TrueSkill2 paper only acheived an accuracy of 68% (again not very impressive for deep learning.) Overall, we are satisfied with this result and believe it is the highest reported accuracy on MOBA win prediction which only utilizes draft phase (0 full combats.)

5.2.3 Challenges

The primary issue we faced in this section was the overfitting problem. This issue is prominent in deep learning and has many de facto solutions including early stopping, dropout, soft labels, and ensembling. Unfortunately, none of these (besides early stopping) had very significant impact

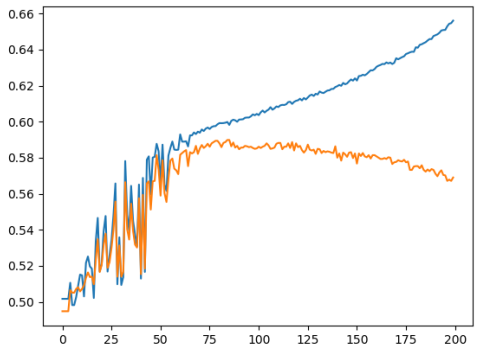

Figure 12: Example of the Training Curve for a Neural Network (blue-train orange-val)

	Fully Connected	Matchup Connected	Matchup and Pair Connected
Validation Accuracy	59.0%	59.0%	59.0%
Ensemble Accuracy	59.0%	59.0%	59.0%

Table 5: Number of games in each year of the dataset. Sources: Oracle's Elixir

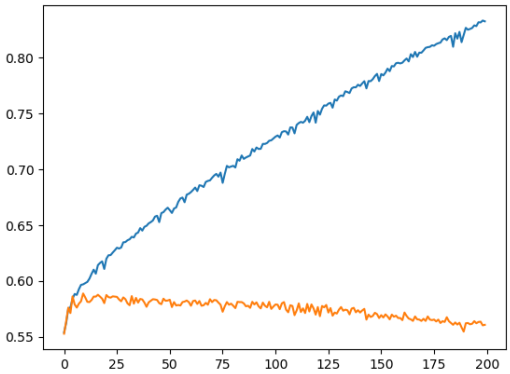

Figure 13: Overfitting Training Curves (blue-train orange-val)

on the performance of the model. One way in which our issue is slightly different from classical deep learning is due to our problem formulation. Since we only give the model access to the draft phase, it is unable to learn from the drastically large amounts of data it would have available to it. Consequently, it needs to focus on abstracting many different game scenarios to ultimately come up with a final probability. Finding a good probability instead of estimating a binary prediction is a huge problem in deep learning referred to as the 'calibration problem' Addressing these issues in the context of win prediction is a clear direction of future work, moreover, incorporating statistical models like those in TrueSkill2 have high potential of solving this problem. Additionally, solving this problem in win predictions, if it can be extended, has myriad applications throughout machine learning where confidence or calibration is desirable.

6 Conclusions

This work is a breakthrough in win prediction modeling. For team modeling, we do not explicitly model a single team over time, but instead learn a representation of a league structure and use that representation to predict success of teams in other leagues. Although this use case is particularly well suited to League of Legends, this has applicability to other esports, like Overwatch, Counter Strike: Global Offensive, or even sports like Baseball or Basketball which have international leagues. For player modeling, we achieve the highest reported (to our knowledge) accuracy only utilizing

draft phase. This has many applications throughout League of Legends esports, it could be useful for players who want to climb by understanding their chances of winning beforehand or by coaches who want to understand the merits of particularly champion compositions.

7 Future Work

For team data, we looked at win prediction agnostic of following a particular team over time, but future work could use TrueSkill rating as a score feature to boost the model's performance. In addition, we plan to expand the expressiveness of this model by using a heterogeneous (and directed) GCN model to represent the data [7]. This will enable us to use more convolutional layers without the risk of leakage. For player data, one of the main directions of future work is to further analyze the type of overfitting which occurs in the model. This can be explored by measuring how one-sided certain games are and comparing the model's confidence to the actual outcome. Creating a model which is 100% accurate 60% of the time and unsure 40% of the time would be an insane improvement over 60% accurate 100% of the time. This type of model is the ultimate goal for this type of win prediction. Moreover, a further extension of this model would be to integrate the model into a champion selector which could actually give mid-draft advice on which champions to ban and which champions to choose. Overall, there are many extremely interesting directions to go with win prediction and hopefully this beginning research inspires future work in this direction.

References

[1] Thomas N. Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks". In: *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings* (2017), pp. 1–14. arXiv: 1609.02907.

[2] Xuan Lan et al. "A player behavior model for predicting win-loss outcome in MOBA games". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 11323 LNAI.2016 (2018), pp. 474–488. ISSN: 16113349. DOI: 10.1007/978-3-030-05090-0_41.

[3] Tom Minka, Ryan Cleven, and Yordan Zaykov. "TrueSkill 2: An improved Bayesian skill rating system". In: (2018).

[4] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[5] *Riot Games API*. URL: https://developer.riotgames.com/.

[6] *Riot-Watcher (Riot API Wrapper)*. URL: riot-watcher.readthedocs.io/en/latest/.

[7] Yaming Yang et al. "Interpretable and Efficient Heterogeneous Graph Convolutional Network". In: (2020), pp. 1–14. arXiv: arXiv:2005.13183v2.