Predict Winning State Using Team Composition in Defense of the Ancient 2

Bowei Yu, Dahang Zhang

December 10th, 2016
Northeastern University
2016 Fall - EECE5644 17778 Machine Learning/Pattern Recognition

yu.bow@husky.neu.edu, zhang.dah@husky.neu.edu

Abstract. *In this paper, we try to use two machine learning algorithms to solve prediction challenge of the popular computer game Dota2 in two different use cases.We provide the detail analysis of the game and introduce some related work completed by other teams. Then we talk about how we utilize the K nearest neighbor to predict results in all-pick mode and use Long Short-Term Memory algorithm to predict results in ranking all-pick mode. We compare these two algorithms according to the predicting results and get some conclusions.Also we plan some features to be implement in the future.*

1. Introduction

With the emerging cost-effective computing hardware and the blooming of the Internet, machine learning algorithm starts to appear on every aspect of the application where complex data set exists. The most desirable feature of machine learning is to predict unknown input based on existing training data. Defense of the ancient 2, Dota2, is an online game which put two five-man teams on a standardized map for them to compete with each other. The team which destroy opponent's base will win the game. One aspect that makes Dota2 a popular online game is due to its enormous complexity. At the beginning stage of every game, each team member need to pick a hero, and every hero has its own unique abilities to help the team. Players could select from a hero pool which contains a total of 113 heroes (at the time of writing). This means that for a total of 10 players each game, the possible team composition is: $113 \times 112 \times ...104$ which is roughly 2.25×10^{20}. Compared to the game Go which has a 19-by-19 board, this number is about the total possibility of the first seven steps. In this paper, we will discuss using the data set from this stage to predict the outcome of the game. Furthermore, as the game progresses, different elements such as laning, item build, ability build sequence will influence the team performance and hence alter the game's final result. The enormous number of possibility makes Dota2 a complex and dynamic game as each decision made by player could have a significant impact on the outcome. That is why modern game AI are only based on human-crafted script to access the game situation and in many situations, in order for game AI to compete with human player, a cheating mechanism would even give the AI certain advantages for it to compete. We believe that with the right machine learning algorithm, the problem of this high-dimensional space would be solved. More specifically, in this project, we will develop a machine learning algorithm take team composition data as input and predict or classify the win or loss status as output.

2. Related work

Many preliminary works has been done by other groups since 2013. Semenov's literature review [5] covers most of the work and compares the results from many other works. From the table 1 in Semenov's review, key results for Dota 2 win predictions from drafts, we can see that during the past few years, the general accuracy of using the team composition to predict the outcome of the game falls between 57.00% to 74.10%. While the game is constantly updated, the dataset tendency could also changes its characteristics. Thus, the most recent research could carry more weight in comparison with our work and it is done by Kinkade and Nicholas's DOTA 2 Win Prediction [6]. In Kinkade's paper, it shows a 64.00% accuracy using optimized logistic regression algorithm. After adding extra features such as hero synergy and counter pick factor, the accuracy is increased to 73.00% on their dataset which contains 62000 game plays.

3. Methodology

In the following section, we will demonstrate our methodology of data set collecting and two machine learning algorithms, k nearest neighbor and long short-term memory recurrent neural networks (LSTM RNNs) in order to solve two use cases.

3.1. Dataset

The overall data processing architecture is shown on figure 1.

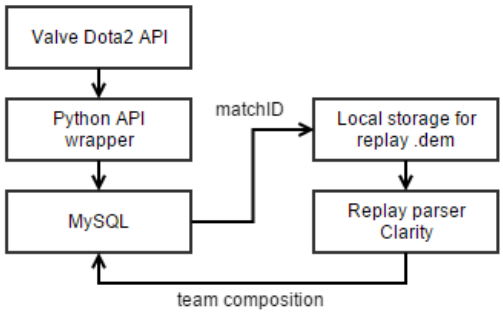

Figure 1. Data processing architecture

For every game that is played by any human player, a replay file will be generated. For games that are played using official online server, Valve, the game developer, provide an official API [1] for any other individual developer to access the game play data. The raw data scrapper is written using Python. Thus, an unofficial API wrapper called Dota 2 API [2] is utilized. The Python scrapper will send request to Valve server and gather the match ID. Match ID can be used to uniquely identify each game played on the public online server. Then, match ID will be stored in a MySQL server for easier data management. As the Valve official APi does not provide the sequence of the pick, in order to gather the sequence information, a separated Java parser, Clarity 2, is used. Clarity 2 parser has the ability to analyse the binary encoded replay file in dem format and extract any detail information about a particular game play. A Python script is utilized to download the dem replay file by using match ID from MySQL database and feed the dem replay file into the Clarity 2 parser. Then the parser will return the hero ID and the pick sequence of the given

game play. The result will be again stored in MySQL database in a separated table. A sample final result is shown on table 1 and the hero ID is a integer number corresponding to a specific hero [3].

player1		player2		player3		player4	
heroID	sequence	heroID	sequence	heroID	sequence	heroID	sequence
6	4	16	10	57	6	14	8
player5		player6		player7		player8	
heroID	sequence	heroID	sequence	heroID	sequence	heroID	sequence
73	2	84	1	71	5	98	9
player9		player10					
heroID	sequence	heroID	sequence				
74	3	67	7				

Table 1. Sample final result after Clarity 2 replay parser

The final dataset used in this report is based on games that played on Valve public server between October and December 2016. The game version covered in this time period is from 6.88e and prior to patch 7.00. As each dem has a file size between 15 MB to 50 MB depend on the actual game time, our team process a dem replay collection of the total size of 316.25 GB. The total number of collected game is 9170. In the following experiments, the training dataset is consists of 7531 number of match. The validation dataset used in training has 753 rows and the final test dataset has 886 rows.

Filter settings used when gathering the match ID through Valve Dota2 API are as following: the game skill bracket is set to 'very-high' to ensure the quality of the match; the lobby and game mode is limited to ranking all-pick which means that players are completing with each other to gain point and level up their ranking, all-pick mode means that most hero is available for pick except heroes that banned prior to hero selection phase. Up to five heroes could be banned.

3.2. K-Nearest Neighbors

K-nearest neighbors is a non-parametric method for classification and regression that predicts objects' class memberships based on the k-closest training examples in the feature space. We used a distance function and chose to utilize all training examples as "nearest neighbors." The feature vector and label we used is described in part 4.2.1 of this section. We chose to implement K-nearest neighbors in order to better model the relationships between heroes instead of simply taking into account wins when a hero is present.

3.2.1. Feature Vector

In Dota2 matches, one team is called the "radiant" and the other team is called the "dire." These terms are roughly analogous to "home" and "away", as they only determine the starting point of each team on the game world map, which is roughly symmetric. There are 113 heroes in Dota2 (as of writing), so for our algorithm we used a feature vector $x \in R^{224}$ such that:

$$x_i = \begin{cases} 1 & \text{if a radiant player played as the hero with id i} \\ 0 & \text{otherwise} \end{cases}$$

$$x_{i+112} = \begin{cases} 1 & \text{if a dire player played as the hero with id i} \\ 0 & \text{otherwise} \end{cases}$$

3.2.2. Algorithm

The training examples are vectors in a multidimensional feature space, each with a class label(either "win" or "not win"). The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples. In the classification phase, k is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the k training samples nearest to that query point. The distance metric for our implementation is Euclidean distance. Often, the classification accuracy of k-NN can be improved significantly if the distance metric is learned with specialized algorithms.

3.3. Long Short-Term Memory

Long Short-Term Memory, LSTM, is the state-of-art recurrent neural network (rnn) technique. Compared to the traditional convolution neural network (cnn), the rnn demonstrates good amount of advantages on the time sequence data set. LSTM as an modified form of the rnn, improves the performance and accuracy upon datasets that could have a vanishing gradient problem due to long-term dependence if training on the original rnn.

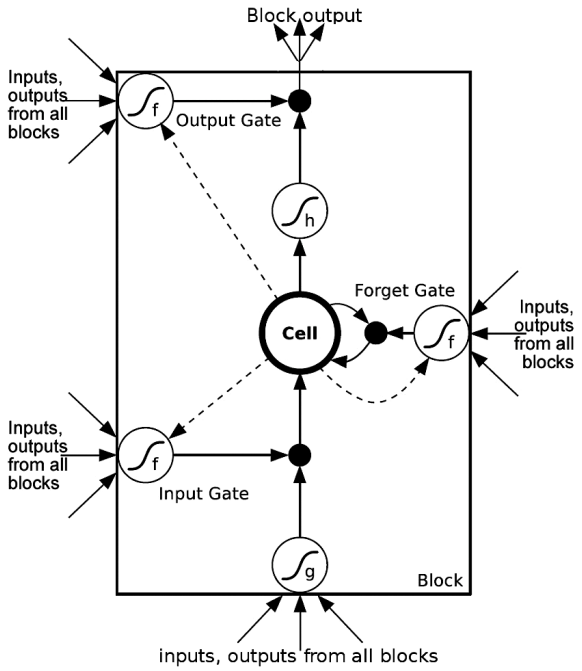

Figure 2. Basic block digram for LSTM from Morillot's paper [9]

The basic cell of rnn is usually constructed with a single layer of activation function, $tanh$. Thus, for a given time sequence data, the previous generated gradient will gradually be overwritten by new data inputs. The solution of this vanishing gradient problem could be solved by LSTM which has four interacting layers accroding to Olah's online journal [10] to counter this problem. The first layer is the cell state. It allows a LSTM cell to maintain

a certain state based on the input from previous time or other cells. Modifications could be made by three other built-in layers. The second layer is the forget gate layer which will determine whether to remove current cell state based on input.

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f)$$

This layer takes the output of previous cell h_{t-1} and current cell input x_t with a sigmoid activation, outputs a result between 0 and 1 to determine whether to throw away current cell state. The third layer input gate layer will determine which value to be used in the update and the fourth $tanh$ layer will create a vector contains candidate updated value that could be used to update the cell state.

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

$$C_t = tanh(W_C[h_{t-1}, x_t] + b_C)$$

After all four gated layers, the final $tanh$ activation function will determine the final output just like the traditional rnn method, but the cell state in the LSTM case, could contain extra information from different time. Thus, these procedures give LSTM the ability to analysis time sequence dataset that has long-term dependency.

The use of the LSTM rnn is for the purpose of analysing the ranking all-pick mode where the selection of the hero appears in a time sequence form. From table 1, we can construct a time sequence selection of hero ID using the sequence column under different players. The final sample input to the LSTM rnn is as follows:

$$[119, 84, 73, 74, 6, 71, 57, 67, 14, 98, 16]$$

The input list contains 11 items where the first item represents the picking side of the game, 118 means radiant side picks first and 119 means dire side picks first; The rest of the 10 items are the selection sequence of hero id. The matrix contains 0 or 1 where 1 represents radiant side wins the game and 0 otherwise. A function of zero padding is utilized in further analysis to convert above input into sub-sequence. A sample sub-sequence padding input is shown below:

$$[119, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

$$[119, 84, 0, 0, 0, 0, 0, 0, 0, 0, 0]$$

$$[119, 84, 73, 0, 0, 0, 0, 0, 0, 0, 0]$$

$$[...]$$

$$[119, 84, 73, 74, 6, 71, 57, 67, 14, 98, 16]$$

The overall structure of the LSTM rnn used in training the game data is shown on figure 4. The first layer is the embedding layer which convert the raw input integer into dense vectors of fixed size. This vector embedding is a crucial layer, without this vector transformation, the result will be terrible and in some cases, the rnn will not even converge to a reasonable accuracy. Following the embedding layer is two convolution neural network layers, the convolution layer and max-pooling layer. These two convolution layers are used to capture the spacial character of the input time sequence. The convolution

layer first filter out the embedded vector inputs with a set number of filters and then the max-pooling layer reduces the overfitting by summing certain number of filtered results together After the two convolution layers there are three layers of LSTM. At each LSTM layer, a dropout rate is set to reduce the overfitting problem of the rnn. And a final dense layer is added to produce the binary output of the LSTM network.

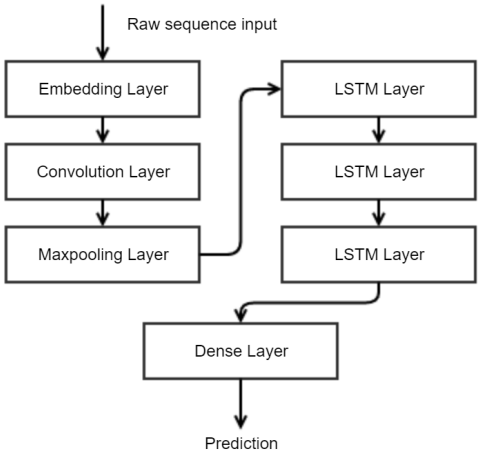

Figure 3. Overall LSTM based neural network structure

4. Results

4.1. KNN

In this part we utilize 10947 training samples and 1275 test samples to test our KNN algorithm. In order to analyse how the parameter k would affect the performance, We fix the sample's number and increase the k from 1 to 1275 as we can see in figure 5. The accuracy increase significantly when k reaches 200 at first and then converge to 0.644 gradually. So we suppose that our optimal accuracy is 0.646 and our optimal k would be the k of first local maximum which is 212. Then to discover how the number of training data would affect the accuracy, we fix k to k = 212 and the amount of test sample $N_{test} = 1275$, and increase the number of training sample from 500 to 10500 which gives us figure 6. In figure 6, the accuracy improve significantly from 500 to 5000 and become stable to around 0.646 after 5000 which means 5000 samples are sufficient to provide the optimal prediction for 1275 test samples at k=212.

4.2. LSTM

Figure 7 shows the number of data inputs without sub-sequence versus the training, validation and test accuracy. The accuracy for the training dataset fluctuates with the increase of the data input. As the matter of fact, the LSTM network will easily overfit the training dataset as the iteration number goes up. The more iteration the network being updated, the more accurate it will be in terms of the accuracy of the training dataset. However, in the mean time, the accuracy of the validation dataset drops significantly during this overfitting process and thus, the accuracy for the test dataset drops too. The final convergence of the validation and test accuracy will be between 50% to 55% which is the probability of tossing a coin and guess which side if will be after it settles. Although the

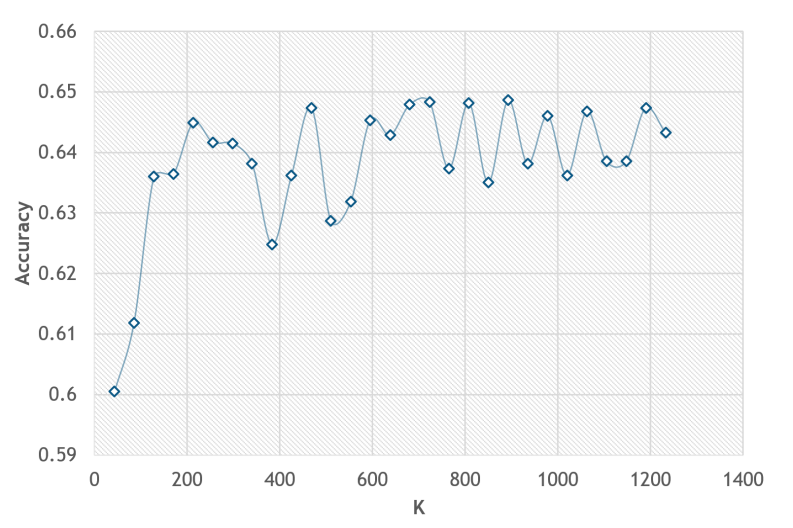

Figure 4. Accuracy for various k.($N_{training} = 10947, N_{test} = 1275$**)**

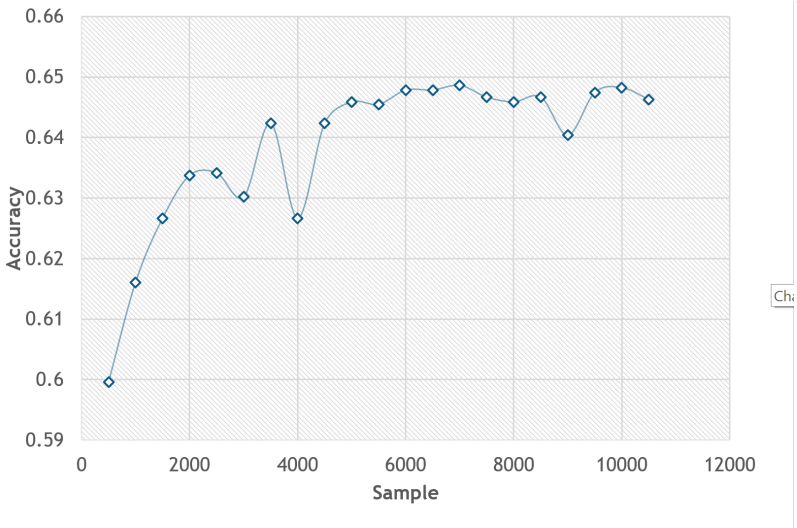

Figure 5. Accuracy trained by different $N_{training}$**.(**$k = 212, N_{test} = 1275$**)**

LSTM based neural network has two mechanism prevents this overfitting problem, the max-pooling layer for the convolution network and the dropout rate for the three LSTM layers, the overfitting still occurs after certain amount of iterations. However, the two overfitting prevention mechanisms do work, without them, the overfitting speed is much quicker. In the LSTM network without overfitting prevention mechanisms, the training dataset accuracy will start out in the 60% region and increases dramatically to 75% in many cases and the validation dataset accuracy is low at 50% region.

The best test dataset accuracy occurs when the training dataset is maximized. The test dataset accuracy has a trend of increasing as the training dataset increase. This could indicate that the test dataset accuracy could be further increased by feeding more data to the network. However, due to the constrain of our data processing architecture, it cannot be done in time for the completion of this report. Other researcher, Kinkade, completing a similar task achieves a test dataset accuracy of 73.00% with 6.5 times number of training

dataset. Thus, a further increase of the test dataset accuracy is expected if there are more training dataset available for the network.

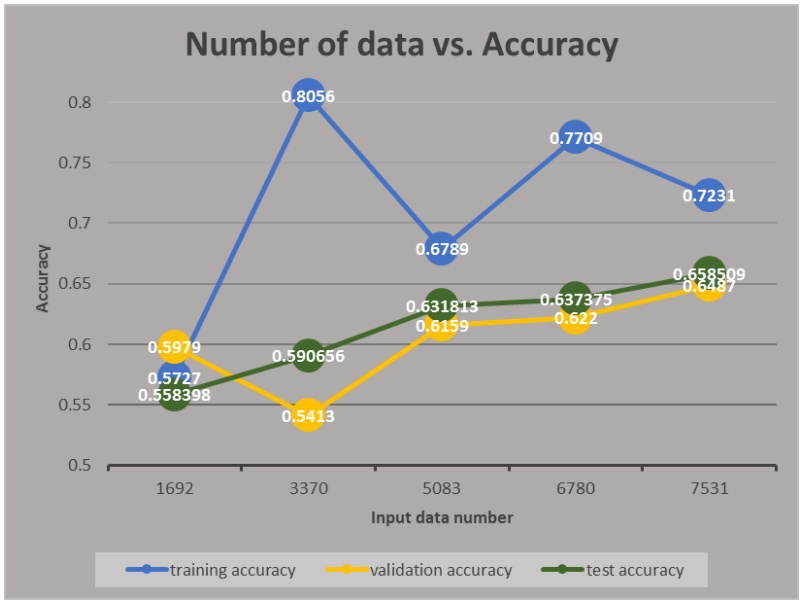

Figure 6. LSTM network accuracy trend with number of input

Figure 8 shows the LSTM based neural network performance using sub-sequence data input. Because each game data input is being processed to generate sub-sequences, the final number of the input is roughly increase 11 times the original. The cutoff of number occurs due to the fact that a new set validation data is generated using a factor of 0.1. As figure 8 demonstrates, the trend is similar to the non-sub-sequence case with a slightly decreases in all accuracy. This situation maybe caused by the early sub-sequence has fewer deterministic affect on the final game outcome. In other words, using first two hero selections to predict the final game outcome is less accurate than using all then hero selections. Thus, a drop of the accuracy could be expected. The potential solution for this problem is introducing a weight vector which increase for each sub-sequence so that the final ten hero selections will have the most impact in predicting the final outcome.

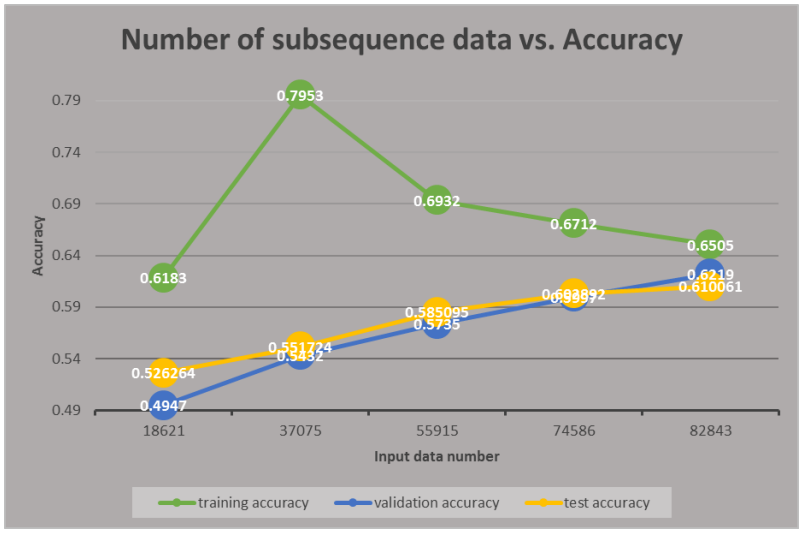

Figure 7. LSTM network accuracy trend with number of sub-sequence input

```
# 736 [2793440660, [115, 40, 62, 18, 6, 65, 110, 11, 17, 26, 79]]
[[ 119.   40.   62.    0.    0.    0.    0.    0.    0.    0.]] prediction:  0  true outcome:  1 raw prediction:  [[ 0.49038923]]
[[ 119.   40.   62.   18.    0.    0.    0.    0.    0.    0.]] prediction:  1  true outcome:  1 raw prediction:  [[ 0.51635402]]
[[ 119.   40.   62.   18.    6.    0.    0.    0.    0.    0.]] prediction:  1  true outcome:  1 raw prediction:  [[ 0.55919999]]
[[ 119.   40.   62.   18.    6.   65.    0.    0.    0.    0.]] prediction:  1  true outcome:  1 raw prediction:  [[ 0.61578637]]
[[ 119.   40.   62.   18.    6.   65.  110.    0.    0.    0.]] prediction:  1  true outcome:  1 raw prediction:  [[ 0.61158615]]
[[ 119.   40.   62.   18.    6.   65.  110.   11.    0.    0.]] prediction:  1  true outcome:  1 raw prediction:  [[ 0.64538366]]
[[ 119.   40.   62.   18.    6.   65.  110.   11.   17.    0.]] prediction:  1  true outcome:  1 raw prediction:  [[ 0.56468016]]
[[ 119.   40.   62.   18.    6.   65.  110.   11.   17.   26.]] prediction:  0  true outcome:  1 raw prediction:  [[ 0.49911928]]
[[ 119.   40.   62.   18.    6.   65.  110.   11.   17.   26.   79.]] prediction:  1  true outcome:  1 raw prediction:  [[ 0.51358181]]
```

Figure 8. Sample prediction output of the sub-sequence data input

4.3. Comparison

From the results above, we could see that our accuracy is pretty similar with other researchers' work. And LSTM is slightly better than KNN algorithm.In additional, LSTM sub-sequence is not optimized but could be used to analysis of the hero selection during the entire drafting process. Figure 10 shows all the optimal accuracies.

Algorithm	Accuracy
KNN	0.646274509803921
LSTM	0.658509
LSTM(subsequence)	0.610061

Figure 9. Comparison

5. Future Work

5.1. Further expansion of the LSTM input

As shown in the result section, the accuracy is relatively low and close to other authors' test accuracy. The suspect of this issue could be that the influence of the drafting phase has only a portion impact on the final outcome of the game. In other words, the team composition cannot definitively determine the outcome of the game. According to Kinkade's work, after factoring two game play phase feature in logistic regression algorithm, the test

accuracy could go up to around 74.10%. This indicates that the two game phases could be closely related and by combing both phase, the chosen machine learning algorithm could potentially predict the outcome more accurately. Because the easy expandability of the LSTM network, the input of the network could be expand just by increasing the list size. The LSTM network could also be expanded using the encode-decode technique. By using a encoder before the LSTM layer, the network will be capable of merging different input time sequence into a dense vector space and then produce the binary prediction result. Although further tuning of the LSTM network maybe needed, the more complex part of this input expansion would be the feature extraction of the raw dem replay file and how to organize the added features.

6. Conclusion

As stated in previous sections, both of our algorithms produce a relatively good accuracy for our test dataset. However, while many other works focus on using the end game result to predict the outcome, our LSTM algorithm could give the user the ability to analysis the progression of the prediction. More precisely, our LSTM algorithm could give the user suggestions and sights of how to decisions during the game. In the case of drafting phase, it could predict the winning rate with every single hero selection using our sub-sequence LSTM model which could greatly help the user.

7. References

[1] Valve Dota2 API https://wiki.teamfortress.com/wiki/WebAPI#Dota_2
[2] Dota 2 API https://dota2api.readthedocs.io/en/latest/
[3] Hero ID json https://github.com/kronusme/dota2-api/blob/348ab7846baa5d819ab31db4bfdf60355a936b2e/data/heroes.json
[4] Clarity 2 game replay parser https://github.com/skadistats/clarity
[5] Semenov, Aleksandr, et al. "Applications of Machine Learning in Dota 2: Literature Review and Practical Knowledge Sharing."
[6] Kinkade, Nicholas. "DOTA 2 Win Prediction."
[7] Johansson, Filip, and Jesper Wikström. "Result Prediction by Mining Replays in Dota 2." (2015).
[9] Morillot, Olivier, Laurence Likforman-Sulem, and Emmanuèle Grosicki. "New baseline correction algorithm for text-line recognition with bidirectional recurrent neural networks." Journal of Electronic Imaging 22.2 (2013): 023028-023028.
[10] Olah, Christopher. "Understanding LSTM Networks." Understanding LSTM Networks – Colah's Blog. N.p., n.d. Web.
[11] Tai, Kai Sheng, Richard Socher, and Christopher D. Manning. "Improved semantic representations from tree-structured long short-term memory networks." arXiv preprint arXiv:1503.00075 (2015).

8. Work distribution

Bowei Yu:

- Python data collection script
- Final Clarity 2 parser
- MySQL database management
- LSTM algorithm
- Final report and presentation slides

Dahang Zhang:

- Clarity 2 parser research
- Dota2 data source API research
- KNN algorithm
- Final report and presentation slides