Conflict Foresight: Teamfight Prediction in Dota 2 using Deep Learning

Alexander Bilchuk, Uri Moser

Introduction

With the rise of e-sports in recent years, vast amount of data is being generated every day and is waiting to be analysed. This creates significant value and opportunity in predicting in-game events using Machine Learning.

The Game

Valve's Dota 2 is one of the most popular games in the e-sports' scene. It belongs to the Multi-player Online Battle Arena (MOBA) genre, meaning, it is a 10 player game with two teams of five players, typically taking between 35-45 minutes. In comparison to games like League of Legends, Heroes of Newerth, and many others, Dota2 stands out for its high complexity and vast amount of in-game mechanics of both the micro and macro variety. In every game, each player selects one out of 121 unique heroes to play. Each hero has its own unique skills, abilities and attributes scaling sequence. 208 unique items can be purchased throughout each game with gold that is earned by killing hostile computer-controlled mobs and enemy heroes. There are 3 lanes through which each team must get to the enemy's Ancient (also known as Throne) and destroy it in order to win. That is where the name is coming from – Defense of The Ancients 2. There are multiple strategies and winning a game requires coordination within the team and the ability to react to the opposition's tactics and behavior.

Game map with a brief explanation

Items icons

Hero icons

Goals

So far, most of the research in the Data Science field was concentrated around prediction of the game's outcome. This field seems to be overpopulated. But micro-predictions, which seek to predict events inside a match, is still an unknown territory.

In this project we decided to focus on predicting an in-game event. In particular – prediction of a teamfight. A teamfight is a clash between teams, resulting in 3 or more heroes dead (sent to Tavern for a penalty-time). Being able to predict the participants and time of a teamfight can be useful for both e-sport commentators and audience, because they provide the ability to observe an event that might be otherwise missed. Another application is post-game analysis and similarly to football player's likelihood to score a goal, teamfight prediction can be a powerful tool in determining what tactics a player should apply to gain an advantage in an upcoming fight.

Data

For this project, we collected the data ourselves straight from the opendota.com API [1]. Meaning, we acquired binary-represented replays of over 2000 matches played on the Pro-scene from 20.8.2021 to 28.8.2021 and parsed them, using open-source java toolkit called Clarity [2]. Each replay file was originally around 50MB in size. To get the most out of them while keeping the size of the dataset manageable, we chose the sampling period to be 30 game ticks (10 seconds) thus created a Database representing each game in slices of 10 seconds (each match consists of +/- 310 rows). Since our main goal was to predict teamfights, we needed to build features that would be of most relevance to it. In the end we used 2762 features. Some of them are:

- Hero statistics. Info on hero's playing performance until now and status of goals achieved. Includes features like: Level, Kills, Deaths, Assists, ObserverWardsPlaced, SentryWardsPlaced, CreepsStacked, CampsStacked, RunePickups, TowerKills, RoshanKills, TotalEarnedGold, LastHitCount, TotalEarnedXP, Stuns etc.
- Hero state. General real-time data on status of hero's attributes like Agility, Intellect, Strength as well as MagicalResistanceValue, PhysicalArmorValue, Mana, Taunt Cooldown, BKBChargesUsed, AbilityPoints, PrimaryAttribute, MoveSpeed, Health, lifeState, TaggedAsVisibleByTeam etc.
- Ability. Includes the attributes of each ability, such as cooldown, level, mana cost.
- Items list. information on purchasing, toggle and activation times of items such as Blink Dagger, Black King Bar, Magic Wand, Quelling Blade, Power Treads, Hand of Midas, Hurricane Pike, Force Staff and many more.

Some features were derived from in order to improve the dataset. For example, instead of showing a simple coordinate of a player at a certain moment (which was pretty, useless) visibility history for each player of each player. If players saw an enemy last time a few seconds ago he can still figure where he is now. However, if the enemy is out of sight for a few minutes – it's almost impossible to figure out where he is. Therefore, we added:

- Proximities: Hero coordinates, 1-4 Ally proximity, 1-4 change Enemy proximity ,1-5 Enemy proximity, 1-5 change Closest, ally tower proximity, Closest ally tower proximity change, Closest enemy tower proximity, Closest enemy tower proximity change, enemy visibility history.

Even though the dataset is quite extensive, there is still important information missing which a human observer pays attention to. For instance, there is no information on projectiles or non-player characters (NPCs).

The classification labels are binary values representing whether the player will be participating in a teamfight within 60 seconds of a given instance. The 60 seconds range provides us with a relatively balanced data and a similar ratio of True and False labels. With this, we decided to use accuracy as the model's primary measure for success, as will be noted note in the 'Model Analysis' section.

Normalizing

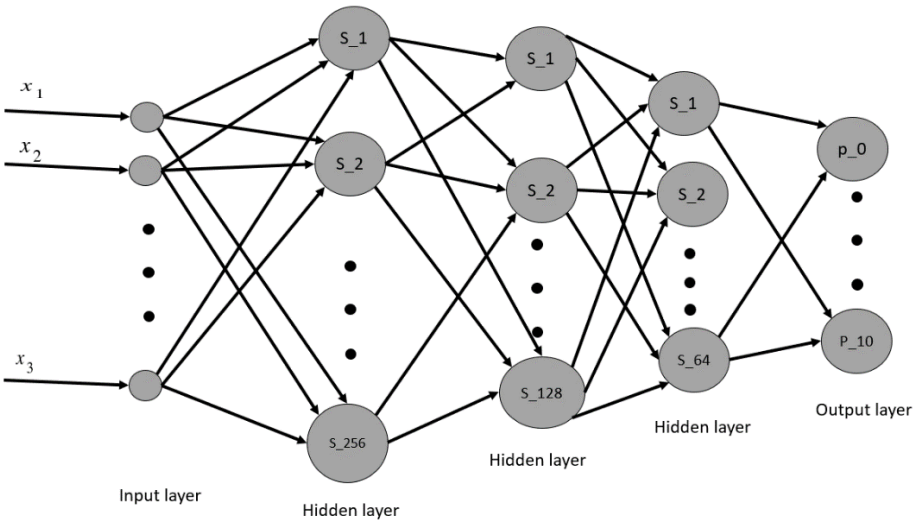
In order to normalize the data efficiently, we sampled 150 random games, and used the range of each feature's values within said games to normalize the entire dataset. Normalizing the features was done across all the players: For each feature, we calculated the maximum and minimum values that any player had in any match and used these values to scale the features to the range between zero and one. We also dropped redundant columns if they haven't changed its value throughout the entire dataset.

Baseline

For the baseline we used several basic algorithms, implemented with the sklearn library: Random Forest, Gradient Boosting, Decision Tree, and AdaBoost Regressors. Some algorithms weren't designed to work with more than 1 column as a prediction label, so we used sklearn's MultiOutputRegressor function, which handles the task by simply fitting one regressor per target.

Basic Models

For the basic part of the project, we decided to work on Feedforward Neural Networks using the Pytorch library in python. For each instance of a match, the models received an input of all the previously discussed features and returned a predicted probability for each player to be participating in a teamfight within 60 seconds. Of the several simple FFNN variations that we tried, the best results were achieved by using a Multilayer Perceptron Neural Network, consisting of 3 layers:



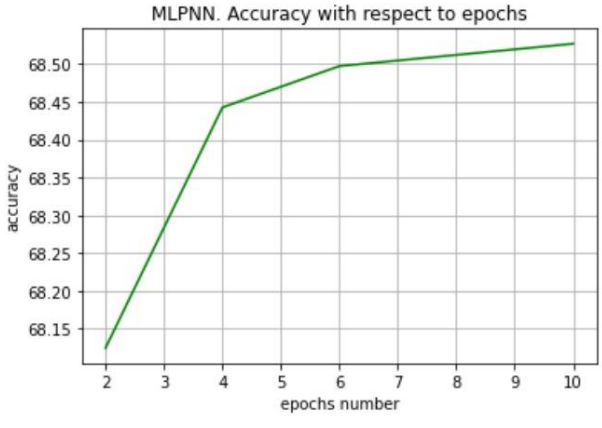
Less layers than that seems to lead to a 'vanishing gradients' problem and increasing the layers' size didn't lead to an increase of accuracy.

For the activation function we used the Rectified Linear activation function (ReLU) [3]. After we compared it to Hyperbolic Tangent function (TanH) function, ReLU seemed to perform just slightly better. One of the reasons could be a danger of falling into a 'fading gradient' problem.

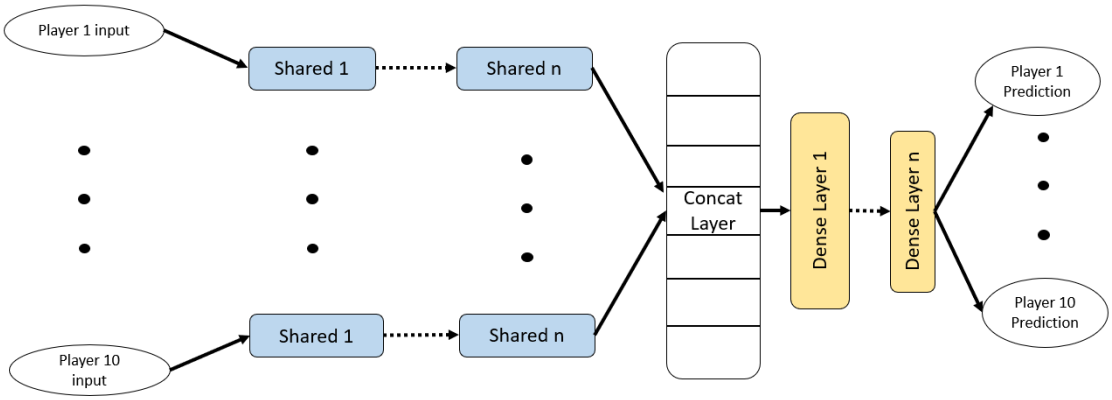
For the loss function – we used Binary Cross Entropy with Logits Loss [4].
Binary Cross Entropy fit the needs of the problem of multi-target binary classification, and the Logits Loss variation of the function provides additional numerical stability thanks to the log-sum-exp trick. The optimizer we chose for our model was Adaptive Moment estimation, also known as Adam [5]. .

Hyper-parameter Tuning

While trying to both figure out the best hyper-parameters and keep the running-time realistic, we conducted a series of experiments, in which we implemented a simplified version of a grid search. After narrowing down our options to a few values, found that the best accuracy was achieved with 10 epochs, batch size of 50, a hidden layer of size 256, and a learning rate of 10^{-4}.



Advanced Model



For the 'advanced' part of the project we attempted to enhance the performance of the FF network by reshaping the data and adding shared weights for each player to make it easier for network to learn the structure of the games' data. What we did was limit the algorithm to share weights between each player's features as if they are an inherently similar unit with minor differences. Thus, we came with a modified model that utilizes that fact:

Each player's features flow through the shared layers, which calculate the same representation for each player. The layers' output is then concatenated into a single vector that is sent to a standard fully connected network layer. As the final output the network returns the probabilities of each player to participate in a teamfight within the next 60 seconds. As shown in the 'Model Analysis' section, this model performs significantly better than the MLP and the other basic algorithms that were previously discussed.

Creative Part

Inspired by the NLP task of Part-of-Speech tagging, we tried to look at our data as one would look at a written text, each player's features being a word and each instance being a sentence. With that, we used a model that is used to predict PoS and applied it to our own problem.

We reshaped the featured similarly to how we did for the advanced model, with each instance being represented as a sequence of each player's features and gave the data to a 2-layer bi-directional LSTM model implemented in pytorch. We also converted the 10 binary target values into a single integer between 0 and 1023. That meant that the model's output was an integer that could be later decoded back into those values.

While the LSTM's performance would seem slightly worse than that of the MLP, it's important to note that the way we measured accuracy is not quite the same as it was for previous models. For all the other models, accuracy was calculated as the number of correctly predicted player labels divided by ten times the total amount of instances, meaning that an instance where most player predictions were correct would contribute to the total accuracy. In this model however, since we now have 1024 different possible labels, a similarly close prediction would be counted as any other wrong prediction. So overall we would consider this model to yield better results than the MLP and possibly even better than the advanced model.

Model Analysis

As a main metric for our evaluation, we chose to focus on *accuracy*. It is the ratio of number of correct predictions to the total number of input samples. Since we padded the teamfights occasions with a 60 second prior prediction, labels became less sparse, justifying our metric. The results amongst the models are as follows:

Algorithm name	Accuracy
Random Forest	**65.4788**
Gradient Boosting	**62.1475**
Decision Tree	**64.8753**
AdaBoost Regressors	**64.6219**
Multilayer Perceptron Neural Network	**68.6977**
Shared Weights Feedforward	**70.4610**
Long Short-Term Memory RNN	**67.1255**

While being a relatively old concept, MLP seems to be able to compete well with other Machine Learning algorithms.

It is also important to note that sometimes the algorithm predicted the teamfights that are more than 60 seconds away. To reduce the confusion in the calculations we decided to still count those just as we would any other misprediction.

Conclusion

We felt like this project was a sort of baptism of fire for us. We are thankful for the opportunity to dive into the world of practical Machine Learning.

Above we outline our research in the e-sports game Dota 2. With the help of a multilayered perceptron that utilizes shared weights, one can predict the occurrence of a near teamfight quite accurately. A big part of the work was dedicated to the pre-process and the features' engineering and extraction. All the work is replicable, the code will be published on github.com and the dataset to kaggle.com to wait for future experiments for the field of micro-predictions appears to be endless.

Bibliography

[1] The website providing Dota2 game replays (https://www.opendota.com/api-keys)

[2] Repository of the Clarity parser used (https://github.com/skadistats/clarity)

[3] Pytorch. ReLU documentation (https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html)

[4] Pytorch. 'BCE With Logits Loss' documentation (https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html)

[5] J. Brownlee 'Gentle Introduction to the Adam Optimization Algorithm for Deep Learning' (https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/), 2017