Honours Project Report

Can machine learning algorithms be used to improve video game matchmaking?

Taylor Lee 2019

A report submitted as part of the requirements for the degree of BSc (Hons) in Computer Science

At

Oxford Brookes University

Contents

Abstract

Matchmaking is a crucial part of competitive video games. It allows players of the game to be placed in matches that give them the best experience possible. Unfortunately, matchmaking systems are not always perfect and can place players in unfair matches. This in turn makes the game far less enjoyable for either side as there is no challenge involved for the winning team and the losing team faces constant defeat. Match data from Heroes of the Storm a Massive Online Battle Arena (MOBA) developed by Blizzard Entertainment and machine learning techniques, in the case neural networks, were used to find the features of a "fair" match in this case a match with a close to 50%-win rate for either side. The neural network was trained to guess the winner of a heroes of the storm match giving win rates for both teams as the output. The final neural network achieved an accuracy of 61.73% in guessing the winner of a match and 61.49% accuracy on a second set of data. This trained model was then applied to an in-theory matchmaking system that used the trained model and matchmaking principles to generate "fair" matches for players.

1. Introduction

The competitive video games industry is constantly growing with the esports sector having a value of $620m in 2017 and with current estimates a value of $1.6bn by 2022 (PwC, 2018). Even with the industry's thriving climate a frequent issue is commonly brought up in the community, the issue being the quality of matchmaking systems within video games. As an example, the most viewed post in the Heroes of the Storm competitive forum is about the lack of balance in the competitive matchmaking (Heroes of the Storm Forums, 2018). A similar story can be said for the Overwatch competitive forum with the most viewed post being an appeal to Blizzard Entertainment (The developers of the game), asking them to improve the current matchmaking system (Overwatch Forums, 2018). As you can see the community isn't afraid to voice their opinion about the current state of matchmaking with a solution needed to generate 'fair' matches for players of competitive online video games.

'Fair' is defined as "Treating people equally without favouritism or discrimination." (Oxford Dictionaries, 2019). Using this definition in the concept of competitive video games, ‘fair’ matchmaking can be defined as “matching teams of players so that both teams have an equal chance of winning the given match”. If the matchmaking system isn’t created with balancing teams in mind it can lead to stagnation in the player base, with high skill players become bored as they are not faced with a challenge and lower skill players becoming frustrated due to not being able to win against the higher skilled players they are matched with. Attempts to provide ‘fair’ games to players dates back as far as 2006 when Microsoft proposed the TrueSkill matchmaking system, which assigns players a skill rating based on their past performance in matches (Herbrich and Graepel, 2006). An update to this system was proposed in 2018 called TrueSkill 2, this system considers more variables compared to its predecessor such as a players experiences, if the player is in a party with other players, the number of kills the player has scored, the tendency for the player to quit and the skill of the player in other game modes (Minka, Cleven and Zaykov, 2018). Although matchmaking system have come a long way since 2006, machine learning using neural networks could be an interesting way to improve these systems further by learning and adapting to the player base over the games lifespan, which is the main basis for the research question of this paper ‘Can machine learning algorithms be used to improve video game matchmaking?’.

I will be using match data from Heroes of the Storm a MOBA (Multiplayer online battle arena) developed by Blizzard Entertainment to conduct my research. The match data API will be acquired from hotslogs.com which is a site dedicated to storing data for Heroes of the Storm matches. This data will then be used in tandem with TensorFlow (a python machine learning library) with a neural network being the machine learning method used. Once the neural network has learnt what a “fair” match is (which is determined by multiple variables in the dataset such as the players levels, hero levels and the current player wide win rate of a certain hero) the network will then be tested by matching a dataset of actual players and matching them up accordingly. These generated matches will then be compared to actual games made by the current matchmaking algorithm with any improvements being noted. The target audience for this paper are game developers as using the research found to improve their matchmaking systems, the games developed by them will be able to keep a stable player base throughout its lifespan as there will be far less frustration in the community, due to the increases in ‘fair’ matches being generate. This in turn could increase the developers profits as with a wider market they have more of an opportunity to sell in game purchases and adverts to the players of the game and their reputation as game developers due to having more fluid matchmaking.

2. Literature Review

With the ever-growing popularity of competitive online video games, a need for skill-based matchmaking systems were needed as beforehand players were matched randomly via playlists or found a match themselves via the use of server browsers. The industry standard for video game matchmaking is based on the Elo system (Newell, 2018) which was first coined in the 1960s by Arpad Elo and then first published in detail in 1978 in his book *The rating of chessplayers, past and present* (Elo, 2008). This system was first primarily used in Chess to calculate the skill levels of players, a players’ Elo is determined by the outcome of games they play, so if a player wins a match their rating rises while if they lose their rating falls. As an example, if a high-skilled player wins against a low-skilled player their rating will only rise a few points and the low-skilled player will only lose a few points, while on the other hand if the low-skilled player wins the match they will gain a large amount of points while the high-skilled player loses a large amount. The Elo system has since become widely used in video games though most developers tend to use derivatives of the Elo system such as Overwatch using its own system (Overwatch, 2018). The Glicko and Glicko-2 rating system was invented as an improvement to the Elo system (Glickman, 2000) and is commonly used in games such as Guild Wars 2 (O'Dell, 2014). Which unlike the classic Elo system considers the inactivity of a player as the skill rating of a player who returns to the game after a period of time may not be a true reflection of their actual skill. Another commonly used system as an alternative to Elo is Microsoft TrueSkill system used in games such as Halo, (Herbrich and

Graepel, 2006) which has since had an update in the form of TrueSkill 2 (Minka, Cleven and Zaykov, 2018). This system unlike Elo is designed to support games with more than 2 players and can infer the skill of individual players from the team's results. As TrueSkill use multiple variables not just the player's win loss ratio so it can provide a much more accurate result of that player's 'true' skill. These variables include but aren't limited to a player's experience, if the player is in a party with other players, the number of kills the player has scored, the tendency for the player to quit and the skill of the player in other game modes.

As the classic skill rating system (portrayed in the above examples) typically uses single numbers to represent the individual players rank within games that use it. This system is getting vastly outdated due to modern games become progressively complex with extra statistics to consider such as roles within a team (the classic roles include healer, tank, damage) and the composition of the team (how many of each role each team has). Machine learning algorithms (in this case a neural network) are a good alternative to the classic skill rating system as they can take the extra statistics modern games provide into account when developed, which can lead to a much better understanding of the given game (in this case Heroes of the Storm) and what intitles a 'fair' match within the game. The results from the machine learning process can then be applied to the matchmaking system to improve its quality.

Research into using machine learning algorithms as means to improve video game matchmaking has happened multiple times in the past, for instance Delalleau (Delalleau et al., 2012) trained a neural network using match data from Ghost Recon Online to improve matchmaking within the game. They concluded that just using a skill value isn't enough to evaluate the balance of a match, with more to be gained by using a richer player profile that contains multiple statistics collected within the game. They also argued that fun is more important than balance and showed that it could be used as the main principle in a matchmaking system. This paper heavily relates to my research as they used a trained neural network to help find balanced matches from a pool of players which I will also be using. One of the problems with this paper was the number of matches used to train this model which was 3937 compared to the 1,028,706 which will be used in my neural network. Ghost Recon Online was also in an early beta-test when the neural network was implemented meaning most players were still learning the game and weren't playing to their full potential, also players were not matched by skill in the beta but randomly, both of these points led to the majority of matches being unbalanced and therefore the neural network wasn't trained on a fair data set with matches tending to be one sided. Due to Heroes of the Storm being three years old at the time of writing with an established player base and a larger set of match data to work with, I feel that the neural network will be able to be trained more efficiently.

Wang (Wang, Yang and Sun, 2015) investigated the idea of fun further by identifying players roles within the game of League of Legends and how teams with certain roles had more fun matches via the use of a neural network. The roles will split into four categories called GL (Global Liberal - Players that assisted others), LL (Local Liberal - Players that focused on killing the enemy), GC (Global Conservative- Players that focused on destroying buildings) and LC (Local Conservative- Players that focused on farming). From the results they found a team full of GL players was the most likely composition to win a match due to the team helping one another and a team full of LC players was the less likely composition to win a match as everyone just focuses on farming gold leading to inactive playstyle. From this they concluded a team of one or more GL players is the community standard for a 'fun' match as GL players tend to have good observation skills, help the team and have an active style of play. They also performed a survey with players of the game after finding most matches tended to end at the 25-minute and 26-minute marks due to the surrender mechanics within the game. 90% of players asked stated matches that are less than 26 minutes were not enjoyable as they tended to be one sided which lessens the enjoyment for both teams. Though they later stated that other kinds of game statistics rather than just match duration would be a better measure for enjoyment levels, especially as other games may not have a close relation to match duration as League of Legends. Overall this paper accomplished what it aimed to do by placing players in groups based on their play styles, though in the conclusion they stated that the research carried out could possibly be applied to matchmaking systems in the future to improve them. This will be considered in my project though it will not be the primary focus as to accomplish this it involves focusing on each player individually and learning their play style, rather than looking at the matchmaking system which is the focus of my project.

Claypool (Claypool et al., 2015) further investigated the balance and enjoyment of matchmaking using League of Legends matches. Concluding that games that were balanced by the ranks of players were found unenjoyable by the losing team and found enjoyable by the winning team even more so than balanced games, overall enjoyment is heavily correlated to winning. This led to them proposing that the matchmaking systems should place the player on an unbalanced team in their favour, when appropriate (such as the player having lost multiple games in a row). Although a good idea to increase the enjoyment of the game it has ethical complications due to forcing players into games that they will win most of the time and games the opposing time will more than often lose. This can lead to players overestimating or underestimating their “true” skill within the game due to being placed against weaker opponents or on the other hand stronger opponents. I will not be taking this approach to my matchmaking system due to its unethical nature though I will consider more statistics rather than just player ranking alone as the paper suggested to improve match balance and enjoyment. Also, the paper suggested looking at games other than League of Legends with Heroes of the Storm being one of them, which is the game being used to conduct research in this paper.

Tanuar (Tanuar et al., 2018) investigated team compositions (The number of each role on the team) within the MOBA game Mobile Legend: Bang Bang using neural networks. Concluding that only 24.6% of team combinations of heroes have more than 50% chance of winning a game. This puts in perspective how important the composition of a team factors into that teams win rate, especially in the case of MOBAs where each player plays a certain role be it healer, tank or damage. As Heroes of the Storm is a MOBA team compositions play a big part in deciding the outcome of the match with a bad composition generally leading to a loss. Due to this team compositions will be considered in the machine learning algorithm with the best and worst compositions being learnt, so the matchmaking algorithm can aim to match players in the best compositions possible and avoid the worst compositions. Also, as my dataset (1,028,706 matches) is far large than the one used in this paper (26 matches) it will provide a much more accurate result on what are truly the worst and best team compositions within the game.

Concepts from these papers will used in this project such as the importance of team compositions MOBAs (Tanuar et al., 2018), measuring enjoyment through the use of multiple game statistics rather than just player ranking (Wang, Yang and Sun, 2015) (Delalleau et al., 2012) (Claypool et al., 2015), taking player's play styles into account (Wang, Yang and Sun, 2015) and expanding on the research by Delalleau and Tanuar by using a game with an established player base (Delalleau et al., 2012) (Tanuar et al., 2018). Though the research preformed in this paper differs from the ones above as they mainly focus on judging the enjoyment of players (Delalleau et al., 2012) (Claypool et al., 2015) (Wang, Yang and Sun, 2015) rather than aiming to place players into as "fair" matches as possible, which this paper focuses on.

3. Methodology

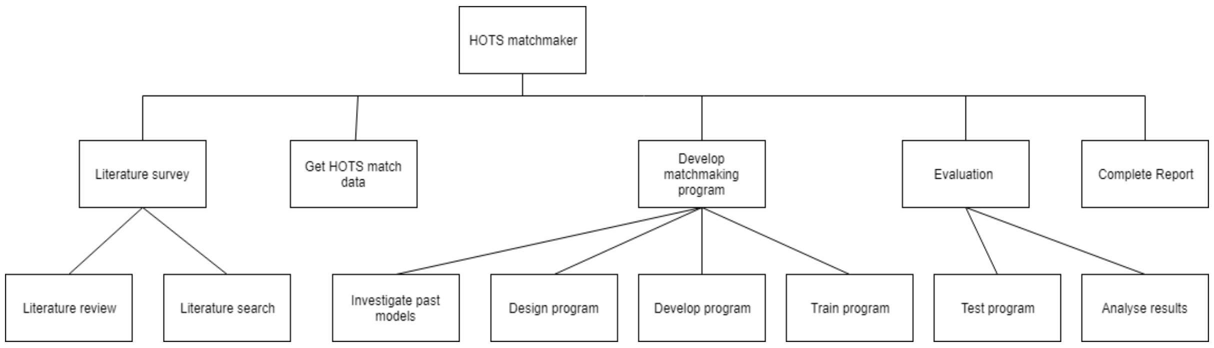


Figure 1: The original plan for this project

The original plan for this project is shown above in figure 1 it was mostly followed exactly with a few extra steps done when I realised, they were needed such as deciding on a machine learning application and learning the machine learning application. This added to time spent on the project as I had to learn the application from scratch which put me under more pressure but otherwise it was followed with good time management and due to this finished on time. I also took a different approach to analysing the results as it was difficult simulating matchmaking in a realistic environment.

Before collecting data to train my machine learning algorithm I needed to decide on an application to perform the machine learning. I debated on multiple applications including PyTorch and MATLAB but settled on TensorFlow due to the abundance of learning material and ease of use compared to other applications such as MATLAB. TensorFlow was also chosen as it can used with python, which is well known for being an excellent language for data science (being just behind a R in a 2016 poll) (KDnuggets, 2016) and I have good experience with python myself. I also used TensorFlow in tandem with Jupyter Notebook as it allows you to run the python code section by section rather than all at once, which is especially helpful for programs involving machine learning due to the processing time taking hours or possibly days. Along with this Anaconda was used (due to its ease of use) to install packages, which there were many of due to the TensorFlow's requirements.

The libraries that were used include tqdm (to show a progress meter when I normalised data), pandas (to help with data analysis), re (allowing the use of regular expressions (helps with filtering data via the use of lambdas), os (to help locate the data files in the operating system), numpy (to help calculate values such as means and losses and general arithmetic), sklearn (to help perform the machine learning, such as using train_test_split for splitting data into the holdout and training data and using the shuffle split function to train the model), matplotlib (to help generate the graphics displayed in the paper) and finally most importantly keras (provides the layers for the neural network and in all is the life blood of the neural network).

After deciding on the machine learning application, I had to collect data to be processed. Due to Heroes of the Storm having no first party API the data had to be acquired via a third party, this was accomplished via the hotslogs.com website (Hotslogs.com, 2018) which provides a 30-day long dataset of Heroes of the Storm matches. The dataset used within this project was from the 28th of November 2018 to the 28th of December 2018 with a total of 1,028,706 matches in the dataset. This data was then normalised to be used with the machine learning algorithms, this was a tricky process due to the data being split into 3 different csv and having to be combined into one this process explained in detail in the *data* section of the paper.

After normalising the dataset to be used for machine learning a decision had to be made on what algorithm to use. I experimented with two models one being a naive Bayes classifier and the other a neural network with the output being a percentage for each team based on their chance to win a match. The naive Bayes classifier was a great way to benchmark and quickly determine if a normalised feature from the dataset helped determine the win rate for a match. The neural network on the other hand while taking longer to train it eventually surpassed the naive Bayes classifier in terms of accuracy after having its layers tweaked, with the naive Bayes classifier being 56.83% correct in guessing the outcome of a game and the neural network guessing correctly 61.7% of the time on the same validation set with my final layers. The final decision for my neural network was a network with a dense layer of 1000 nodes, a PReLU layer to give inactive nodes a small gradient and an Adam optimiser using a mini-batch of 2000 to help improve optimisation. Overall it seems if you have a large dataset to train on a neural network is much more efficient than a naive Bayes classifier but if you have a smaller dataset the naive Bayes classifier outperforms the neural network, due to the dataset size the neural network was my machine learning algorithm choice. Another reason why I choose a neural network is due to its ability to take in large amounts of features and learn the correlation between them, which is especially helpful in a case like this where the final dataset will have a large number of features to process. This process is explained in detail in the *model* section of the paper.

After generating a model, it needed to be tested in a matchmaking environment. This is hard to replicate realistically without having access to

Blizzard's servers originally, I attended to replicate it but found it to be to much of a challenge. So as a compromise I suggested an in-theory model with graphics to help explain it. This is explained in detail in the *results* section of the paper.

4. Data

As stated previously the data was pulled from hotslogs.com's 30-day log (Hotslogs.com, 2018a) which contained 1,028,706 total matches between the dates of the 28th of November 2018 and the 28th of December 2018. More data could be included in this project, but it would entitle pulling data from other game patches which would be far harder to account for in the algorithm due to the way new heroes get added and the power of certain heroes being adjusted. This can lead to skewed results if the data is across multiple patches so to make it as simple as possible the model will focus only on the patch of the given data. The data appears in three different .csv files named:

- ***HeroIDAndMapID.csv*** which contains information about all the heroes and maps within the game
- ***Replay.csv*** which contains all the matches within the dataset with the features being the game mode ID (3=Quick Match 4=Hero League 5=Team League 6=Unranked Draft), the map ID, the time the match took and the date of the match
- ***ReplayCharacters.csv*** which contains all the players featured in the Replay.csv matches with the features being the Hero ID of the player, their hero level, if they won or not, their hotslog.com MMR (Matchmaking Rating) before the match started and multiple end game statistics such as their final in game level, takedowns, healing done, assists and deaths.

From these files a decision needed to be made on what relevant data to carry over to the final dataset.

4.1 Game Modes

In the dataset there are four modes *quick match* which is when players pick a hero prior to being matched with other players via the blizzard matchmaking system. With the other three modes: *hero league*, *team league* and *unranked draft* all featuring the same draft system, this involves each player taking turn picking their hero and banning heroes so they can't be picked. The difference between the modes are *hero league* allows solo players only, *team league* allows teams and solo players and *unranked draft* when lessens stress on players by not assigning them a rank and lack any team restrictions (Heroes of the Storm, 2019). Each of these game modes also have their own separate Matchmaking Rating (MMR) (an ELO equivalent) for each player as stated by Travis McGeathy a Blizzard employee (McGeathy, 2018). This is how the matchmaking system matches players in 'fair' matches although the actual Blizzard MMR for players is hidden from the public. As an alternative the MMR from hotslog.com will be used instead, this is further expanded on in the *players skill* section of the paper.

Out of the 1,028,706 games in the dataset they're split across the modes as follows:

- Quick Match: 589,652 (57.32%)
- Hero League: 89,405 (8.69%)
- Team League: 242,984 (23.62%)
- Unranked Draft: 106,665 (10.37%)

As you can see more than half the games are played in quick match with about a quarter in team league and the rest between hero league and unranked draft. Although some modes are more popular than others all four will be considered in the machine learning process as means to help determine the win rates for individual teams, due to certain combinations of heroes possibly being stronger in certain modes.

4.2 Maps

Another important part of the dataset are the maps the matches take place on, the HeroesMapID.csv list 21 maps though only 14 of them are actually used in the dataset. This is due to certain maps such as Sliver City, Checkpoint Hanamura, Pull Party, Industrial District, Escape from Braxis and Lost Cavern being reserved for heroes brawl, a game mode not include in the dataset. The last map, Haunted Mines is not included due to it being removed from all game modes except custom games in the 16th of October 2018 patch (Heroes of the Storm, 2018a). Below you can see the amount of times each of the 14 remaining maps appears within the dataset.

Map	**Dataset Frequency**
Battlefield of Eternity	84,912 (8.25%)
Blackheart's Bay	51,103 (4.97%)
Cursed Hollow	89,341 (8.68%)
Dragon Shire	85,174 (8.28%)
Garden of Terror	51,632 (5.02%)
Infernal Shrines	82,514 (8.02%)
Sky Temple	88,199 (8.57%)
Tomb of the Spider Queen	80,622 (7.84%)
Towers of Doom	86,700 (8.43%)
Braxis Holdout	85,606 (8.32%)
Warhead Junction	40,385 (3.93%)
Hanamura Temple	52,333 (5.09%)
Volskaya Foundry	75,242 (7.31%)
Alterac Pass	74,943 (7.29%)
Total	1,028,706

Table 1: The frequency that maps appear in the dataset

The reasoning why Blackheart’s Bay, Garden of Terror, Warhead Junction and Hanamura Temple all have far lower appearance rates in the data set is because they are only playable within quick match and none of the three other modes. The rest of the maps on the other hand have been in ranked play rotation since the start of season 3 on the 10th of July 2018 (Heroes of the Storm, 2018b), meaning they are playable in all 4 modes featured in the dataset. This could create a problem with the finished model as the algorithm won’t be able to truly process ranked matches on the corresponding 4 maps due to not being trained on any ranked matches featuring these maps. To get around this Blackheart’s Bay, Garden of Terror, Warhead Junction and Hanamura Temple will only be tested in quick matches and no other modes.

I feel using the map as a feature in the machine learning model will be another great way to improve the accuracy of win rates for both teams due to the possibility of certain heroes preforming better on specific maps.

4.3 Heroes

Probably the most important part of the dataset besides player skill are the heroes that are picked by the players. In the HeroIDMapID.csv there are 85 heroes and in the dataset itself there are 84, this is because the last hero in the HeroIDMapID.csv is Imperius who wasn't released until January which is after this dataset takes place. So, for the purpose of this project only the first 84 heroes in the game will be consider due to the lack of data on Imperius.

The hero that players pick can play a factor in their teams win rate with the highest win rate hero being Gazlowe at 54.8% and the lowest being Tassadar at 42.3% at the time of the dataset (Hotslogs.com, 2018b). As you can see from the 12.5% range of win percentages the heroes picked alone will play a big factor in deciding a team's win rate.

Composition	**Number of Games**	**Win Percent**
T,B,H,BD,SD	20,892	50.1%
T,B.H,SD,SD	10,978	49.2%
T,H,BD,SD,SD	10,395	50.3%
T,H,BD,SD,S	6,677	52.4%
B,B,H,BD,SD	5,996	52.1%
T,H,A,BD,SD	5,494	50.4%
T,B,H,SD,S	5,369	51.4%
T,T,H,BD,SD	4,741	46.8%
T,H,SD,SD,SD	4,726	49.7%
T,H,SD,SD,S	4,452	52.9%

Table 2: The 10 most popular team compositions and their respective win rates. Done by subclasses T=Tank B=Bruiser H=Healer BD=Burst Damage SD= Sustained Damage S=Siege A=Ambusher (Utility and Support aren't in the top 10) (Hotslogs.com, 2019a).

Heroes are also assigned classes within the game either being a Specialist, Warrior, Support or Assassin based on their role. Although the base classes are a good way to categorise heroes, hotslog.com breaks them down into a further 9 subclasses titled: Utility, Tank, Bruiser, Siege, Healer, Sustained Damage, Burst Damage, Ambusher and Support. Table 2 features the 10 most popular compositions of these subclasses and their win rates, ranging from 46.8% to 52.9% showing that having the right heroes has an impact on the team winning the match. Using these subclasses to the original 4 classes due to heroes such as Varian who is a listed as an assassin assigned with the bruiser role which is different from every other assassin who are either sustained damage, burst damage or ambushers with the bruiser role being more typical of warriors.

Assigning these roles helps because if heroes such as Varian were listed in the original 4 classes their true playstyle would not be fully considered, with Varian's being more similar to a Warriors. It also helps paint a more specific playstyle for each hero due to their being more than double the classes than the original 4, because of this the 9 subclasses generated by hotslog.com will be the features carried over to the model and not the original 4 classes.

4.4 Players Skill

Each individual player's skill is probably the best indicator of a teams win rate. In the case of this dataset the MMR for each player is included in the ReplayCharacters.csv, as stated before this is not an MMR produced by Blizzard due to it being hidden from the public and instead produced by hotslogs.com with the owner using a system similar to Microsoft's TrueSkill system (Hotslogs.com, 2019b). Although the accuracy may not be as good as Blizzard's true MMR due to not having access to every player's full match history, I still feel it will be a good feature to include in the model as it still shows each player's relevant skill.

The highest MMR in the dataset is 4997 and the lowest is -1019 with no recognised cap on the highest and lowest MMR can be, for this reason a range will need to be set to normalise the data between. Due to only 83 players actually being above 4500 MMR and 473 players being below 400 MMR I decided to make 4500 MMR to 400 MMR the range the data will be normalised between, due to lack of data below and above this range. Also, there are

26519 players without an MMR at all so matches featuring these players will also not be included in the data set due to the range set above.

Another good indicator of a player's skill is there level with the hero they have selected. In the case of Heroes of the Storm there is a levelling system that allows players to gain rewards by levelling up heroes as they play matches with them, with a level range from 1 to 20. For instance two players with the same MMR playing the same hero would be seen as evenly matched without hero levels taken into account but if hero levels were included comparing a player with level 1 in the selected hero to a player with level 20 in the selected hero, the level 20 would be seen as more skilled due to having invested more time in the selected hero while the level 1 has little experience with the selected hero.

Multiple other variables in the dataset could also be a good indicator of a player's skill with the variables being in game level, takedowns, killing blows, assists, deaths, highest kill streak, hero damage, siege damage, healing, damage taken, experience contribution, time spent dead and merc camp captures. Although these variables would give more of a clear indicator of skill, they are hard to implement into the model due to the dataset not including player's IDs so there's no way to track individual lifetime statistics and as this algorithm will be applied to a matchmaking system these variables will not be known beforehand due to them being know after the match has finished.

Though overall, I feel MMR along with the selected hero and their Level will be a good indicator of an individual player's skill even without implementing the additional statistics listed above into the model.

4.5 Normalisation

In an ideal world there would be 32 features to describe a game, the features being:

- The MMR, Hero Selected and Hero Level of each player with 10 players in a game which is 30 features
- The Map
- The game mode

The problem lies in how the data is stored with the maps, heroes and game modes all being IDs. This does not translate well over to machine learning algorithms due to their reliance on storing the data in weights which cannot easily happen to IDs as they are either in the given line of data or not. A solution to this is using one hot encoding.

One hot encoding works well with categorical values like the maps, heroes and game modes. As it allows us “perform “binarization” of the category and include it as a feature to train the model” (Vasudev, 2017). In other words, each of the heroes, maps and game modes are all features in the dataset and are set to either 1 if they are in that given line of data or 0 if they are not.

With one hot encoding we can end up with 84 features to describe a team as there are 84 heroes in the data set and they cannot be picked twice per team. As I also want to include the level and MMR features for each player to get an idea of their skill this increase to 168 features to describe a team, with the level and MMR columns being filled in if that hero is on the team and if the hero is not the two columns are left as a 0. The hero level and MMR unlike the heroes themselves can be easily normalised, so 1 to 20 for hero levels and 400 to 4500 for MMR are converted to weights between 0 and 1 for the model to read.

To allow the model to learn the importance of team composition I included the subgroups of the heroes in the team. This is done by creating a feature for each team of the individual 9 subclasses in the dataset which are then set a value between 0 and 3 (due to only 3 heroes of one type allowed on a team at once) and then normalised to a weight between 0 and 1. This in turn gives us 177 features to describe a team.

After some minimal testing using Naive Bayes Gaussian (stated in model section of paper) I decided to add two more features per team member with the features being the MMR and Hero level for that team member. The reason why I did this was due to how spread out between features the original hero MMR and level were, meaning the model finds it hard to learn how to represent player skill. So, to improve the model’s learning I set two additional features that are coherent throughout each match, although they contain the same data as their respective hero MMR and hero level, they allow the model to easily consider MMR and hero levels due to the information being more accessible as it is not spread out throughout the dataset. It also allows the model to consider MMR and hero level as a feature across the board no matter the heroes that are picked. This gives us 10 more features per team to describe a match bringing the total to 187.

This leaves us with 374 features to describe both teams (each team is randomly assigned as ‘Team A or ‘Team B’), one hot encoding the maps gives us 14 additional features and one hot encoding the game modes gives us 4 additional features. This leaves the final normalised dataset with 392 features in total to describe a full match of heroes of the storm.

After deciding on the features, the final dataset had to then be produced. This was done via a python program I created with the step by step process explained. The first step was to produce headings for all the features of the dataset and the label itself. The first heading was for the label which is called “team_a_won” which is set to 0 if team A lost the match or 1 if they won. The next four headings are for the four modes a match could possibly take place in, if a match is in a certain mode that column is set to 1 otherwise it is set as 0. The next headings were for the maps themselves, a heading was created for each map from HeroesIDandMapID.csv and similar to the mode columns are set to 1 if the match takes place on a certain map or 0 if the map is not used. After these headings had been made the HeroesIDandMapID.csv was used to create headings for each team with the first ten headings for each team being each team member’s MMR and Hero level. The next headings for the team included the MMR and Hero level for each of the heroes from the HeroesIDandMapID.csv, which are only filled in if that hero is present on the team (They contain the same values as their respective team member MMR and hero columns). The final headings for each team are the headings for each subgroup, with nine in total for each team.

The necessary data to fill in these columns is then pulled from the Replay.csv and ReplayCharacters.csv files by using the replayID for the given match, each team from the match is then randomly assigned as “Team A” or “Team

B” with the “team_a_won” set to 1 or 0 depending on the winner. The points of the dataset such as the mode and map are one hot encoded with help from the HeroesIDAndMapID.csv, meaning they are set to 1 or 0 depending if they are present in the match or not. The rest of the data such as MMR, Hero levels and subgroups are all set as weights between 0 and 1, converted from the respective values: In the case of MMR, it is between 400 and 4500, with Hero levels it is between 1 and 20 and with subgroups its set on a scale of 0 to 3 as only a maximum of three heroes from a certain subgroup can be on a team at once. This process then generates a full normalised data set once its looped though every match listed in Replays.csv.

This normalised dataset contains a total of 957,924 matches out of the 1,028,706 that were contained in the original dataset, this is because of players that have MMR out of the given range of 400 to 4500 and games including these players were removed from the final dataset. Although a loss of 6.88% of the total data I still feel it’s a large enough dataset to gain knowledge about the features of “fair” matches in Heroes of the Storm.

At this point the data needed to be filtered due to some columns being completely “empty” (full of zeroes). This was because of the way the headings were made, the process looped though every ID in HeroesIDandMapID.csv and therefore carried over some of the redundant IDs, including maps and heroes that weren’t present in the dataset. These “empty” columns were therefore removed from the normalised dataset before processing began.

5. Models

After normalising the data, it had to be applied to a model to so it could learn. I initial started with a Naive Bayes Gaussian to learn if adding more features to dataset would improve accuracy. I started with a Naive Bayes Gaussian due to it being far faster than a neural network, taking seconds compared to multiple minutes for the neural network would take. Initial I didn’t have the team member MMR and hero level, which is in the final dataset, but this was added after I saw a slight improvement in the Naive Bayes Gaussians accuracy from 56.68% to 56.83% but it was quickly scrapped afterwards due to not being effective. This is because Naive Bayes Gaussian is more for determining if adding features to a network would improve its accuracy and unlike neural networks cannot learn relationships between features (Dataaspirant, 2017). After I scrapped the Naive Bayes Gaussian to gain these features, I moved on to making a neural network instead to hopefully gain higher accuracy by making the model learn the relationship between features.

5.1 Initial Neural Network

My initial neural network was a basic one with one dense linear layer that took in took in 500 neurons, a sigmoid layer to give percentage win rates for both teams as output, an Adam optimiser with an initial learn rate of 0.001 and beta_1 of .9 which are the default setting for Adam and a binary_crossentropy loss function due to the output being one of binary, the label for the model is if 'team a' won. The reason why I choose Adam as the optimiser was due to its ease of implementation, computationally efficient nature and most importantly its ability to deal with dataset with a large number of parameters and sparse gradients, which is true in the case of this dataset (Kingma and Ba, 2015). During testing each of my neural networks were ran for 100 epochs or until the validation loss stopped improving. Out of the 957,924 matches in the data set 100,000 matches were used for validation during training and another 100,000 matches were a holdout set to be used during final testing of the model (to prevent overfitting by using the same data twice) this gave 757,924 matches to train the model on. The accuracy for this initial model was 59.609% to predict the winner of a match and increased to 59.881% after running for six splits using the ShuffleSplit function. ShuffleSplit splits the test data a different way multiple times based on the number of splits it is given (in this case six), to see if there are any improvements in the model's accuracy by training it on different sets of data.

5.2 Neural Network Testing

After making my initial neural network I decided on tweaking the layers to see if any improvements could be made. The first change I settled on was adding a dropout layer to the neural network, which is a regularisation technique patented by Google in 2013 by the name of "System and method for addressing overfitting in a neural network" (Google, 2013). Dropout layers work by randomly disconnecting neurons within the network resulting in a "thinned network". The number of neurons disconnected is decided based on a probability which is given, in the case of my network I set the probability to 0.5 which is the default and recommend for dropout layers meaning during train time each neuron has a 50% chance of being disconnected from the network (Srivastava et al., 2014). Adding a dropout layer to the neural network helps prevent overfitting, overfitted models are a problem due to relying on past data and as a result struggle to predict outcomes from unseen data. The accuracy of this model was 59.593%, even though the accuracy is slightly lower than the initial model I still felt the dropout layer would help better in the long run due to the model being able to predict the outcome of matches containing unseen data with more certainty.

After adding the dropout layer, I researched into neural network layers more and after some testing I decided to add a PReLU layer to the neural network. PReLU (Parametric Rectified Linear Unit) layers work by allowing leakage into a parameter that is learned alongside the other parameters, which in turn improves model fitting with nearly zero computational cost and little overfitting risk (He et al., 2015). Adding a PReLU layer to my previous dropout neural network increased its accuracy to 61.172% which is a 1.579% increase, due to this increase I decided to keep the PReLU layer in my final model.

After deciding on the layers within my neural network I tweaked the values of them. First off, I edited the number of neurons in the dense layer, initially it was 500 which I decided to change to 1000 neurons, this change gave the neural network an increased accuracy of 61.370%. After seeing this improvement, I decided to add more neurons to the network making the number 1500, this actually decreased the accuracy of the network to 61.346%. Due to this decreased in accuracy I decided to settle on 1000 as the number of neurons in my dense layer.

Next, I decided to edit the values for my Adam optimiser. First, I lowered the initial learning rate to 0.00001. This resulted in the training taking the full 100 epochs limit I set and having an accuracy of 60.938%, due to the length of time the training took I decided to tweak the learning rate to a higher value settling on 0.00005 to allow it to train faster. The accuracy of this test was 61.552%, due to this being an improvement over the model with a 0.0001 learning rate I decided to set the learning rate to 0.00005.

I then decided to tweak the beta_1 level. First, I set it to 0.5 this gave me an accuracy of 61.303%, this was lower than my previous models accuracy so I the value was tweaked again. This time I set the value to 0.7 this returned an accuracy of 61.675% which was better than the model had beta_1 set to 0.9, meaning a beta_1 level of 0.7 is what was assigned to the final model. *All test cases are featured in the appendix for reference with most staying between 61-61.5%*

5.3 Final Neural Network

My final neural network consisted of a linear dense layer of 1000 neurons, a PReLU layer, a dropout layer with a value of 0.5, a sigmoid dense layer for output and an Adam optimiser with a learn rate of 0.00005 and beta_1 of 0.7. After being trained through 6 splits using ShuffleSplit the model returned a final validation accuracy of **61.738%**.

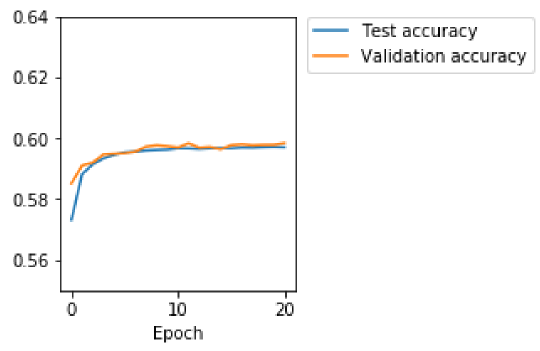


Figure 3: The Initial Neural Network

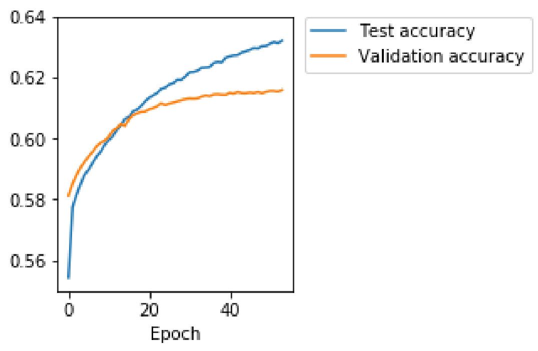


Figure 4: The Final Neural Network

In figure 3 you can see the test accuracy and validation accuracy for the initial neural network. After the first epoch there was little improvement in the accuracy of the model staying around 59.5% in validation accuracy and test accuracy, after 21 epochs the peak of 59.609% validation accuracy was reached. Figure 4 on the other hand features the same values but this time for the final neural network. In this graph the test accuracy and validation accuracy start off initially lower but soon rises and surpasses the initial model. Validation accuracy and test accuracy also have more of a curve which improves regularly over time reaching the peak of 61.738% for validation accuracy and 63.120% for test accuracy, after 54 epochs.

Game Modes	**Sample Predicted**
Quick Match	33,525 of 54,353 (61.68%)
Hero League	5,553 of 9,158 (60.64%)
Team League	15,394 of 25,228 (61.02%)
Unranked Draft	7,019 of 11,261 (62.33%)
Total	61,491 of 100,000 (61.49%)

Table 3: Number of matches correctly predicted on holdout data

In table 3 you can see the number of matches that were correctly predicted from the holdout data. Although the overall percentage is slightly lower than the validation percentage done in training (61.738%) at just over 0.2%, it’s to be expected due to how much the data can vary between the two. This also means my neural network is unlikely to be overfitted due to having a similar accuracy on a second set of 100,000 matches. Overall my final neural network is best at predicting unranked drafts and quick matches with the league game modes being harder to predict. This is due to league games tending to be more balanced with teams playing to their full capability and picking heroes that won’t place them at a disadvantage, making it harder to determine a winner from their team compositions alone.

Win Percentage	**Percentage of Matches**
50%	100%
55%	75.92%
60%	53.14%
65%	34.03%
70%	19.06%
75%	9.17%
80%	3.44%

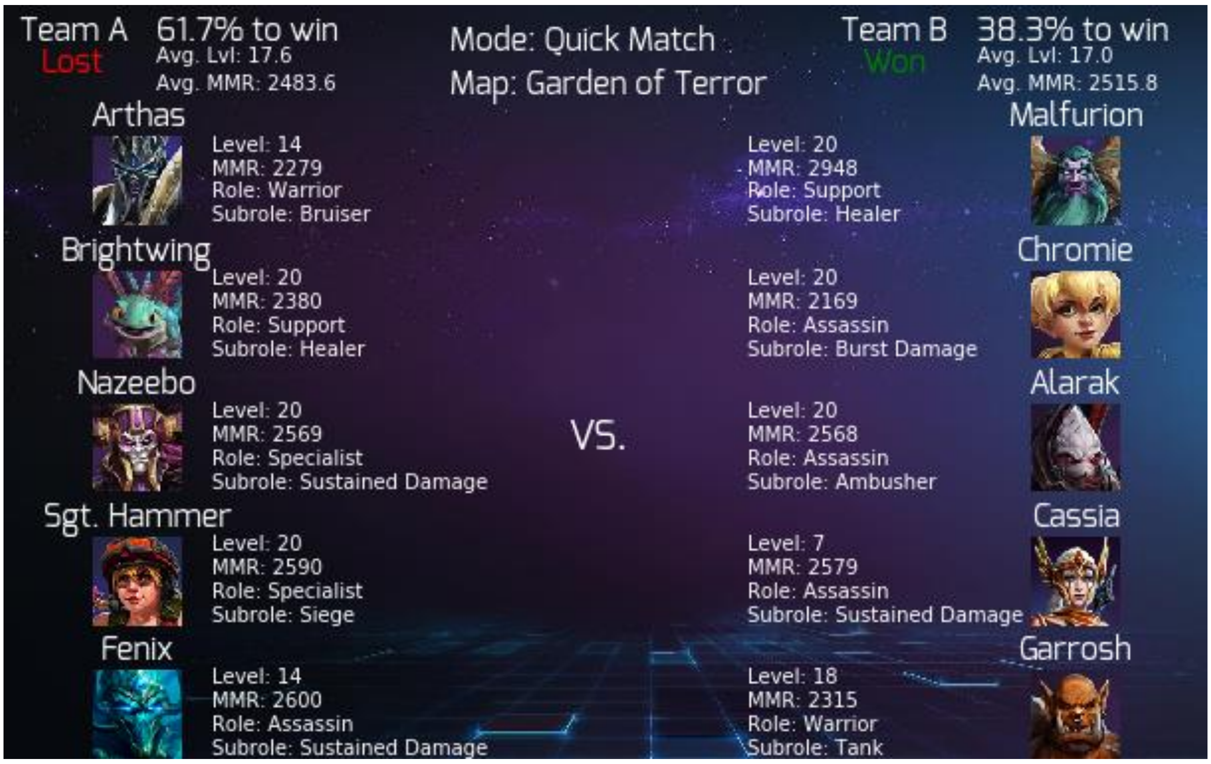
85%	0.86%
90%	0.12%
95%	0.01%
100%	0.00%

Table 4: Percentage of matches in the holdout data categorised into their win percentages (Win percentage shown is for the team that is mostly likely to win)

Table 4 shows the percentage of matches that fall into a certain win percentage. As you can see just under 50% of the matches fall in the 50-60% win percentage range, meaning around half the matches in the dataset are pretty evenly matched and the model finds it hard to predict a true winner. This can further be seen in the table as just under 10% of matches have a 75% or higher win percentage for the supposed 'winner', showing that there are very few matches where the model is certain on a winner for a match. There are also no matches with a 100%-win percentage due to both teams always having the possibility of winning the match. Overall this means that matches in Heroes of the Storm are mostly balanced with very few of them being one-sided, this is expected due the matchmaking system placing players in matches with other players that have similar MMR and therefore similar skill. Although the matches in the current matchmaking system are fairly balanced, my model hopes to improve the balance further by making the win percentages for the majority of matches as close to 50% as possible.

6. Results

After choosing my final model I needed to apply it in the context of a matchmaking system. To help with this I created a visual output of a match using matplotlib, examples of this can be seen in figures 5-9 on 5 random matches from the holdout dataset. The output shows both teams with, the hero for each team member, the level for each hero, the MMR for each team member, the role of each hero, the sub role of each hero, the mode the match was played in, the map the match took place on, the win rate for each team and what team actually won the match. I also included the average MMR and average level for each team to better see if there's a correlation between having a higher average and winning.


Team A Lost 61.7% to win
Avg. Lvl: 17.6
Avg. MMR: 2483.6

Mode: Quick Match
Map: Garden of Terror

Team B Won 38.3% to win
Avg. Lvl: 17.0
Avg. MMR: 2515.8

Arthas
Level: 14
MMR: 2279
Role: Warrior
Subrole: Bruiser

Brightwing
Level: 20
MMR: 2380
Role: Support
Subrole: Healer

Nazeebo
Level: 20
MMR: 2569
Role: Specialist
Subrole: Sustained Damage

Sgt. Hammer
Level: 20
MMR: 2590
Role: Specialist
Subrole: Siege

Fenix
Level: 14
MMR: 2600
Role: Assassin
Subrole: Sustained Damage

VS.

Malfurion
Level: 20
MMR: 2948
Role: Support
Subrole: Healer

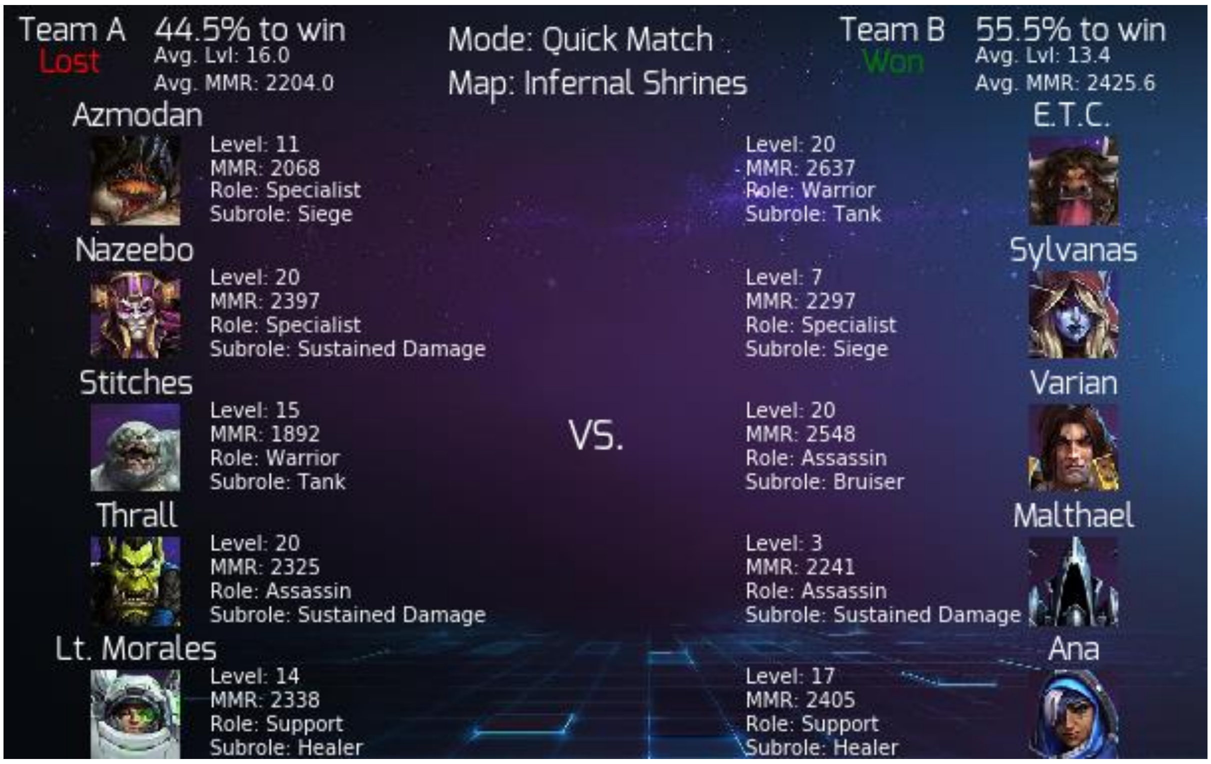
Chromie
Level: 20
MMR: 2169
Role: Assassin
Subrole: Burst Damage

Alarak
Level: 20
MMR: 2568
Role: Assassin
Subrole: Ambusher

Cassia
Level: 7
MMR: 2579
Role: Assassin
Subrole: Sustained Damage

Garrosh
Level: 18
MMR: 2315
Role: Warrior
Subrole: Tank

Figure 5: Match 1 from holdout data


Team A Lost 44.5% to win
Avg. Lvl: 16.0
Avg. MMR: 2204.0

Mode: Quick Match
Map: Infernal Shrines

Team B Won 55.5% to win
Avg. Lvl: 13.4
Avg. MMR: 2425.6

Azmodan
Level: 11
MMR: 2068
Role: Specialist
Subrole: Siege

Nazeebo
Level: 20
MMR: 2397
Role: Specialist
Subrole: Sustained Damage

Stitches
Level: 15
MMR: 1892
Role: Warrior
Subrole: Tank

Thrall
Level: 20
MMR: 2325
Role: Assassin
Subrole: Sustained Damage

Lt. Morales
Level: 14
MMR: 2338
Role: Support
Subrole: Healer

VS.

E.T.C.
Level: 20
MMR: 2637
Role: Warrior
Subrole: Tank

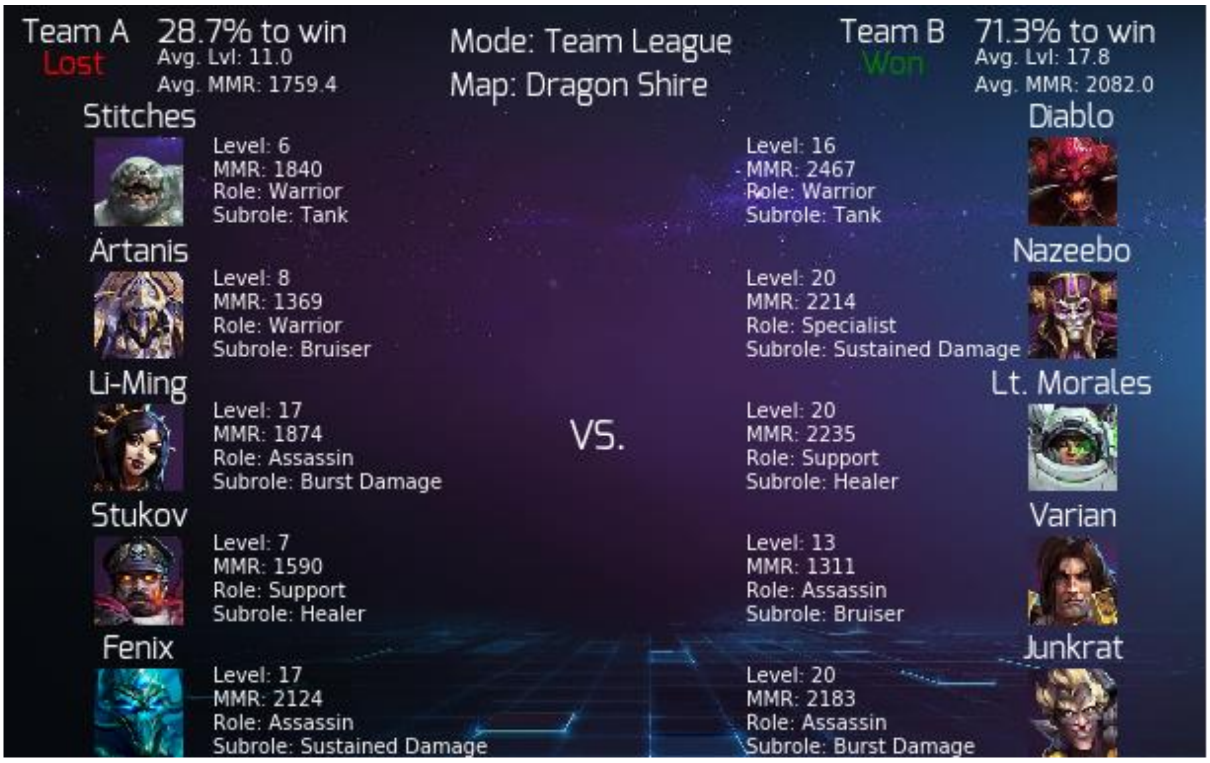
Sylvanas
Level: 7
MMR: 2297
Role: Specialist
Subrole: Siege

Varian
Level: 20
MMR: 2548
Role: Assassin
Subrole: Bruiser

Malthael
Level: 3
MMR: 2241
Role: Assassin
Subrole: Sustained Damage

Ana
Level: 17
MMR: 2405
Role: Support
Subrole: Healer

Figure 6: Match 2 from holdout data



Team A Lost 28.7% to win
Avg. Lvl: 11.0
Avg. MMR: 1759.4

Mode: Team League
Map: Dragon Shire

Team B Won 71.3% to win
Avg. Lvl: 17.8
Avg. MMR: 2082.0

Stitches
Level: 6
MMR: 1840
Role: Warrior
Subrole: Tank

Artanis
Level: 8
MMR: 1369
Role: Warrior
Subrole: Bruiser

Li-Ming
Level: 17
MMR: 1874
Role: Assassin
Subrole: Burst Damage

Stukov
Level: 7
MMR: 1590
Role: Support
Subrole: Healer

Fenix
Level: 17
MMR: 2124
Role: Assassin
Subrole: Sustained Damage

VS.

Diablo
Level: 16
MMR: 2467
Role: Warrior
Subrole: Tank

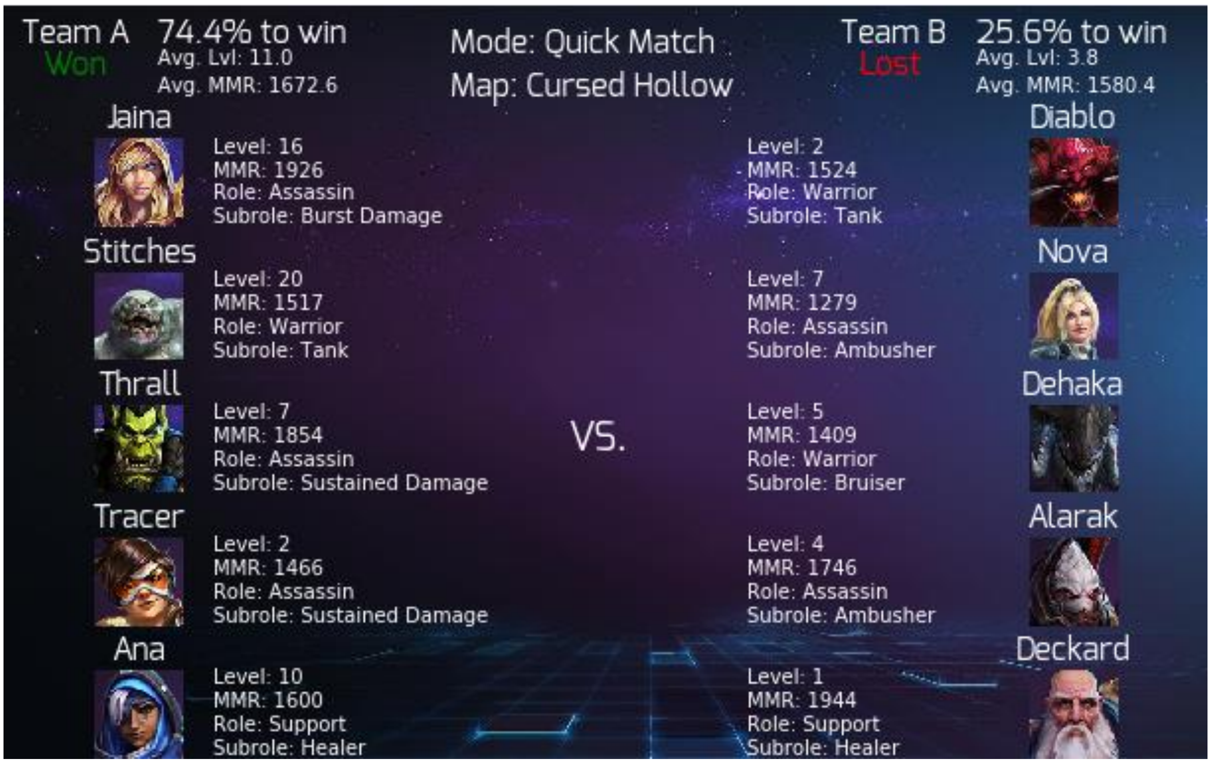
Nazeebo
Level: 20
MMR: 2214
Role: Specialist
Subrole: Sustained Damage

Lt. Morales
Level: 20
MMR: 2235
Role: Support
Subrole: Healer

Varian
Level: 13
MMR: 1311
Role: Assassin
Subrole: Bruiser

Junkrat
Level: 20
MMR: 2183
Role: Assassin
Subrole: Burst Damage

Figure 7: Match 3 from holdout data



Team A Won 74.4% to win
Avg. Lvl: 11.0
Avg. MMR: 1672.6

Mode: Quick Match
Map: Cursed Hollow

Team B Lost 25.6% to win
Avg. Lvl: 3.8
Avg. MMR: 1580.4

Jaina
Level: 16
MMR: 1926
Role: Assassin
Subrole: Burst Damage

Stitches
Level: 20
MMR: 1517
Role: Warrior
Subrole: Tank

Thrall
Level: 7
MMR: 1854
Role: Assassin
Subrole: Sustained Damage

Tracer
Level: 2
MMR: 1466
Role: Assassin
Subrole: Sustained Damage

Ana
Level: 10
MMR: 1600
Role: Support
Subrole: Healer

VS.

Diablo
Level: 2
MMR: 1524
Role: Warrior
Subrole: Tank

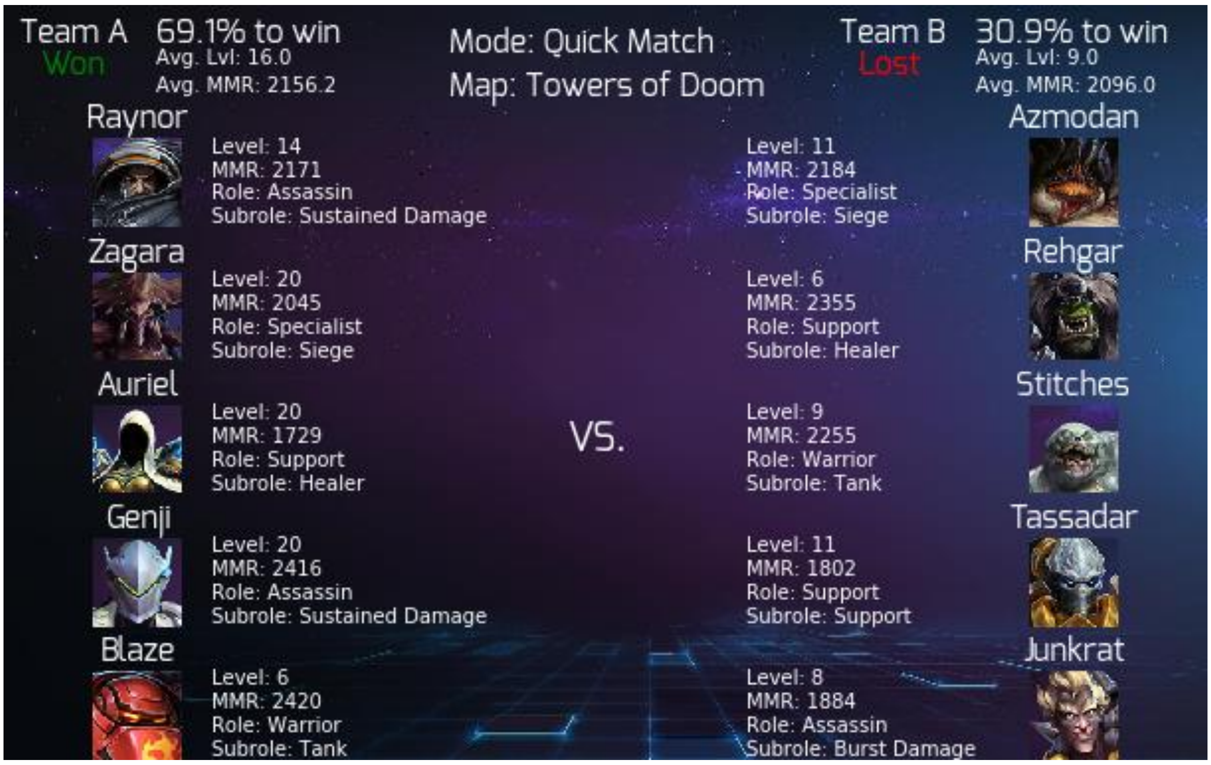
Nova
Level: 7
MMR: 1279
Role: Assassin
Subrole: Ambusher

Dehaka
Level: 5
MMR: 1409
Role: Warrior
Subrole: Bruiser

Alarak
Level: 4
MMR: 1746
Role: Assassin
Subrole: Ambusher

Deckard
Level: 1
MMR: 1944
Role: Support
Subrole: Healer

Figure 8: Match 4 from holdout data


Team A Won 69.1% to win
Avg. Lvl: 16.0
Avg. MMR: 2156.2

Mode: Quick Match
Map: Towers of Doom

Team B Lost 30.9% to win
Avg. Lvl: 9.0
Avg. MMR: 2096.0

Raynor
Level: 14
MMR: 2171
Role: Assassin
Subrole: Sustained Damage

Zagara
Level: 20
MMR: 2045
Role: Specialist
Subrole: Siege

Auriel
Level: 20
MMR: 1729
Role: Support
Subrole: Healer

Genji
Level: 20
MMR: 2416
Role: Assassin
Subrole: Sustained Damage

Blaze
Level: 6
MMR: 2420
Role: Warrior
Subrole: Tank

VS.

Azmodan
Level: 11
MMR: 2184
Role: Specialist
Subrole: Siege

Rehgar
Level: 6
MMR: 2355
Role: Support
Subrole: Healer

Stitches
Level: 9
MMR: 2255
Role: Warrior
Subrole: Tank

Tassadar
Level: 11
MMR: 1802
Role: Support
Subrole: Support

Junkrat
Level: 8
MMR: 1884
Role: Assassin
Subrole: Burst Damage

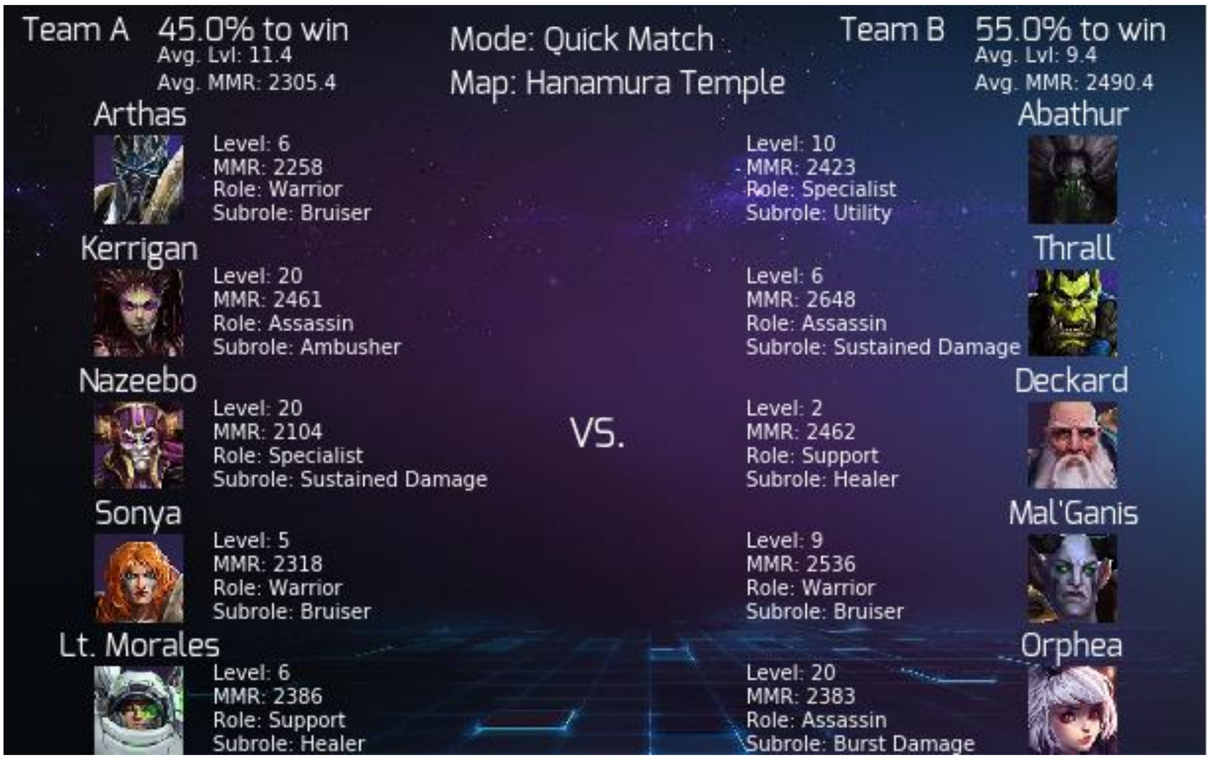
Figure 9: Match 5 from holdout data

The model achieved an accuracy of 61.49% on the holdout data as stated before. 4 out of the 5 random holdout games had correct predictions, which is generally expected due to the model's accuracy. The prediction that was wrong in figure 5 gave team B a lower than 50% chance of winning the match at 38.3% even though they won. A reason why the model may have predicted team A to win could be because its composition has a higher win rate at 53.0% compared to team B's 50.4% (Hotslogs.com, 2019a). Because of this I still feel the model is well tuned enough to give a good verdict on what a balanced match as the model was assuming the most 'likely' team to win and never said that team will win all the time.

6.1 Matchmaking

Due to heroes of the storm being owned by blizzard and therefore its source code private, the neural network model cannot be applied to the actual matchmaking servers themselves. As an alternative I will explain in theory, how the model would be applied to the matchmaking system.

Firstly, a sampling system would be put into place as suggested by Delalleau (Delalleau et al., 2012). It would work by pulling teams from the matchmaking queue and will see what's the most favourable matchup for the majority of teams by comparing multiple scenarios.


Team A 45.0% to win
Avg. Lvl: 11.4
Avg. MMR: 2305.4

Mode: Quick Match
Map: Hanamura Temple

Team B 55.0% to win
Avg. Lvl: 9.4
Avg. MMR: 2490.4

Arthas
Level: 6
MMR: 2258
Role: Warrior
Subrole: Bruiser

Kerrigan
Level: 20
MMR: 2461
Role: Assassin
Subrole: Ambusher

Nazeebo
Level: 20
MMR: 2104
Role: Specialist
Subrole: Sustained Damage

Sonya
Level: 5
MMR: 2318
Role: Warrior
Subrole: Bruiser

Lt. Morales
Level: 6
MMR: 2386
Role: Support
Subrole: Healer

VS.

Abathur
Level: 10
MMR: 2423
Role: Specialist
Subrole: Utility

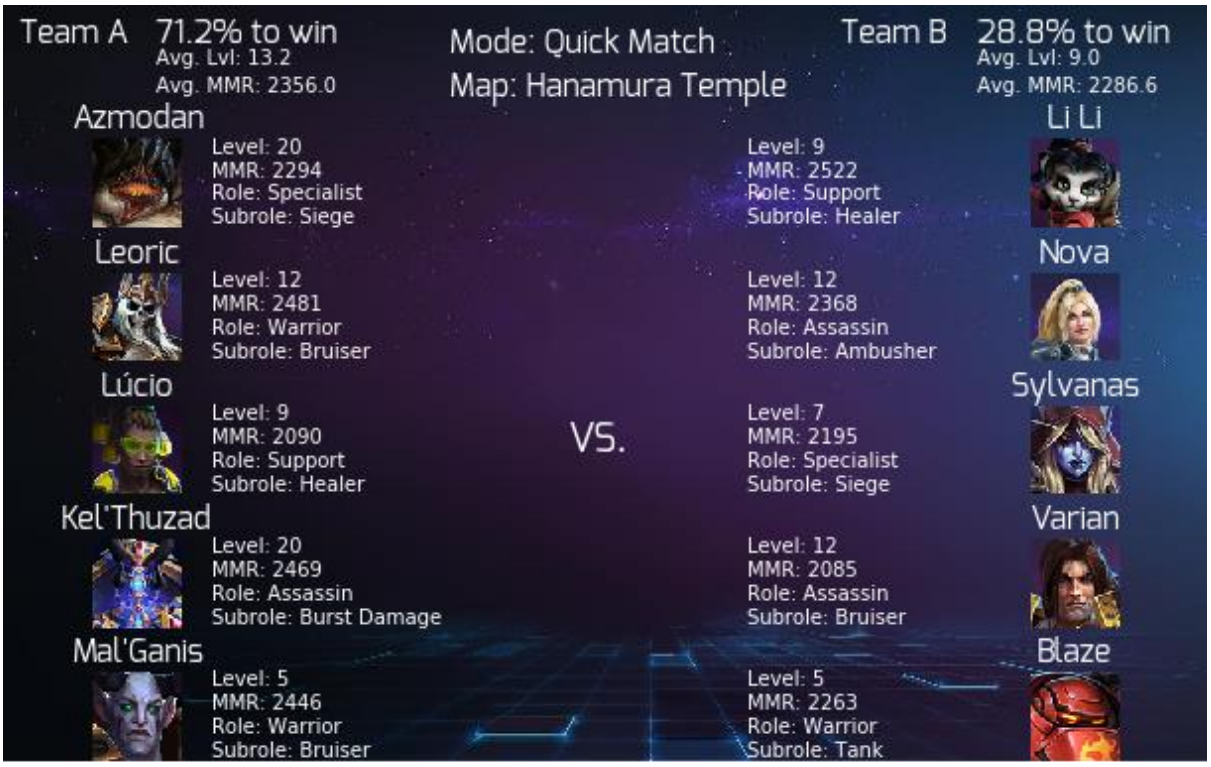
Thrall
Level: 6
MMR: 2648
Role: Assassin
Subrole: Sustained Damage

Deckard
Level: 2
MMR: 2462
Role: Support
Subrole: Healer

Mal'Ganis
Level: 9
MMR: 2536
Role: Warrior
Subrole: Bruiser

Orphea
Level: 20
MMR: 2383
Role: Assassin
Subrole: Burst Damage

Figure 10: Matchup 1: Match 1


Team A 71.2% to win
Avg. Lvl: 13.2
Avg. MMR: 2356.0

Mode: Quick Match
Map: Hanamura Temple

Team B 28.8% to win
Avg. Lvl: 9.0
Avg. MMR: 2286.6

Azmodan
Level: 20
MMR: 2294
Role: Specialist
Subrole: Siege

Leoric
Level: 12
MMR: 2481
Role: Warrior
Subrole: Bruiser

Lúcio
Level: 9
MMR: 2090
Role: Support
Subrole: Healer

Kel'Thuzad
Level: 20
MMR: 2469
Role: Assassin
Subrole: Burst Damage

Mal'Ganis
Level: 5
MMR: 2446
Role: Warrior
Subrole: Bruiser

VS.

Li Li
Level: 9
MMR: 2522
Role: Support
Subrole: Healer

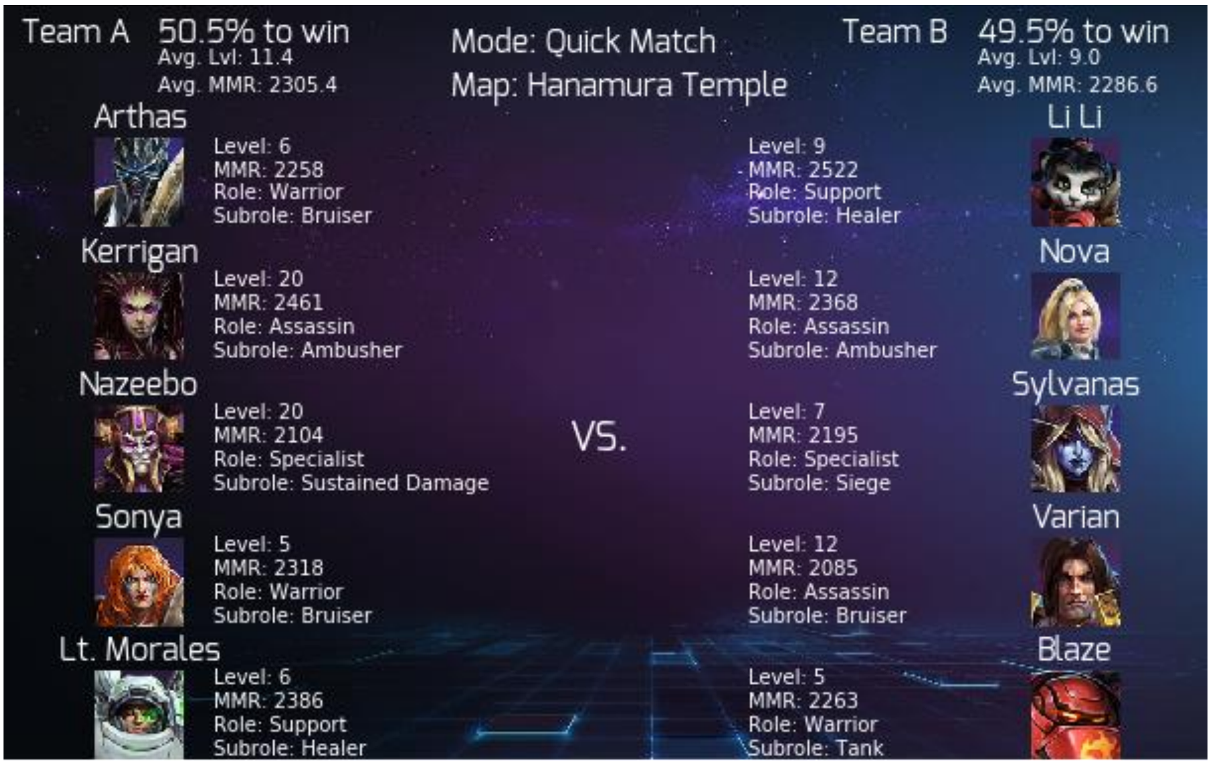
Nova
Level: 12
MMR: 2368
Role: Assassin
Subrole: Ambusher

Sylvanas
Level: 7
MMR: 2195
Role: Specialist
Subrole: Siege

Varian
Level: 12
MMR: 2085
Role: Assassin
Subrole: Bruiser

Blaze
Level: 5
MMR: 2263
Role: Warrior
Subrole: Tank

Figure 11: Matchup 1: Match 2



Team A 50.5% to win
Avg. Lvl: 11.4
Avg. MMR: 2305.4

Mode: Quick Match
Map: Hanamura Temple

Team B 49.5% to win
Avg. Lvl: 9.0
Avg. MMR: 2286.6

Arthas
Level: 6
MMR: 2258
Role: Warrior
Subrole: Bruiser

Kerrigan
Level: 20
MMR: 2461
Role: Assassin
Subrole: Ambusher

Nazeebo
Level: 20
MMR: 2104
Role: Specialist
Subrole: Sustained Damage

Sonya
Level: 5
MMR: 2318
Role: Warrior
Subrole: Bruiser

Lt. Morales
Level: 6
MMR: 2386
Role: Support
Subrole: Healer

VS.

Li Li
Level: 9
MMR: 2522
Role: Support
Subrole: Healer

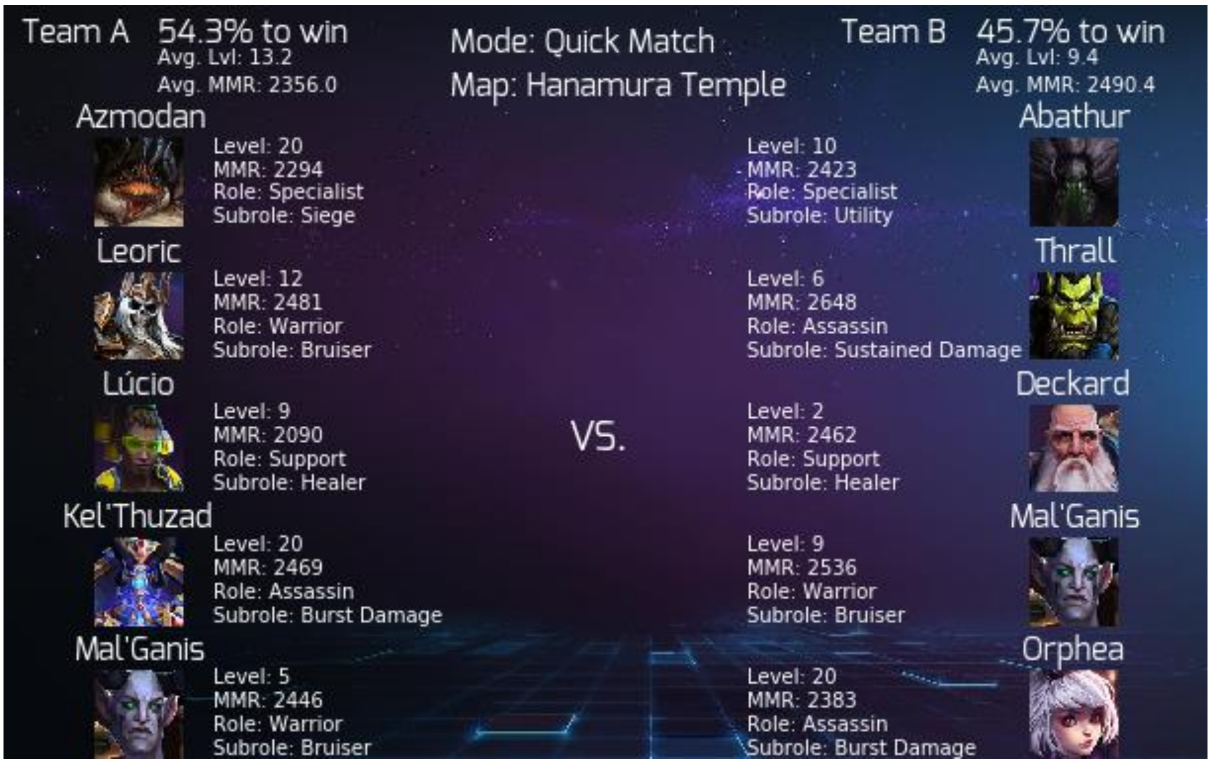
Nova
Level: 12
MMR: 2368
Role: Assassin
Subrole: Ambusher

Sylvanas
Level: 7
MMR: 2195
Role: Specialist
Subrole: Siege

Varian
Level: 12
MMR: 2085
Role: Assassin
Subrole: Bruiser

Blaze
Level: 5
MMR: 2263
Role: Warrior
Subrole: Tank

Figure 12: Matchup 2: Match 1



Team A 54.3% to win
Avg. Lvl: 13.2
Avg. MMR: 2356.0

Mode: Quick Match
Map: Hanamura Temple

Team B 45.7% to win
Avg. Lvl: 9.4
Avg. MMR: 2490.4

Azmodan
Level: 20
MMR: 2294
Role: Specialist
Subrole: Siege

Leoric
Level: 12
MMR: 2481
Role: Warrior
Subrole: Bruiser

Lúcio
Level: 9
MMR: 2090
Role: Support
Subrole: Healer

Kel'Thuzad
Level: 20
MMR: 2469
Role: Assassin
Subrole: Burst Damage

Mal'Ganis
Level: 5
MMR: 2446
Role: Warrior
Subrole: Bruiser

VS.

Abathur
Level: 10
MMR: 2423
Role: Specialist
Subrole: Utility

Thrall
Level: 6
MMR: 2648
Role: Assassin
Subrole: Sustained Damage

Deckard
Level: 2
MMR: 2462
Role: Support
Subrole: Healer

Mal'Ganis
Level: 9
MMR: 2536
Role: Warrior
Subrole: Bruiser

Orphea
Level: 20
MMR: 2383
Role: Assassin
Subrole: Burst Damage

Figure 13: Matchup 2: Match 2

Figures 10-13 feature four teams playing on the same map and mode with every possible matchup for the four teams (The won and lost text has also been removed due to there being no way of knowing the final outcome of these matchups). The matchmaking system would pull these two teams from the queue (players are matched into teams prior) and then decide on the two matches to assign to the teams, in the case of this figures 12 and 13 are the optimal matches. This is due to Team A having a win percentage of 50.5% and Team B having a win percentage of 49.5% in figure 12, while in figure 13 Team A has win percentage of 54.3% and Team B has a win percentage of 45.7%. Although Figure 10 has a good matchup with 45% for Team A and 55% for Team B, the matchmaking system would have to place the other two teams into the match featured in figure 11 which is far more uneven with Team A expected to win 71.2% of the time. Overall this makes figure 12 and figure 13 the optimal two matches for the four teams. As a side note the first two matches were ones created by the blizzard matchmaking system itself and the last two were fictional matches. The last two matches are better match ups but in reality, it would be impossible to always make matches like them due to factors out of the matchmaking system's control, such as the optimal players required for the matchup not being online at that time meaning the system will have to make compromises.

Although the sampling system will be the basis for the matchmaking system multiple other factors will need to be considered and compromises made to make the gameplay experience as smooth as possible for players. One of the factors is an increase in wait time for players if an optimal match cannot be found for them. A solution to this problem is giving priority to these players, for instance if a team is not matched up during the sampling process they will be placed to the top of the queue and will be the first team to be sampled with other new teams that join the queue. This hopefully will find each team a 'fair' match and if no optimal matches can be found they will be automatically matched to 'fairest' match possible at the time, after a certain time threshold is past to stop players getting bored. The only exception to this is league-based modes which will have a longer time threshold due to players being more competitive and in turn would want the 'fairest' match possible.

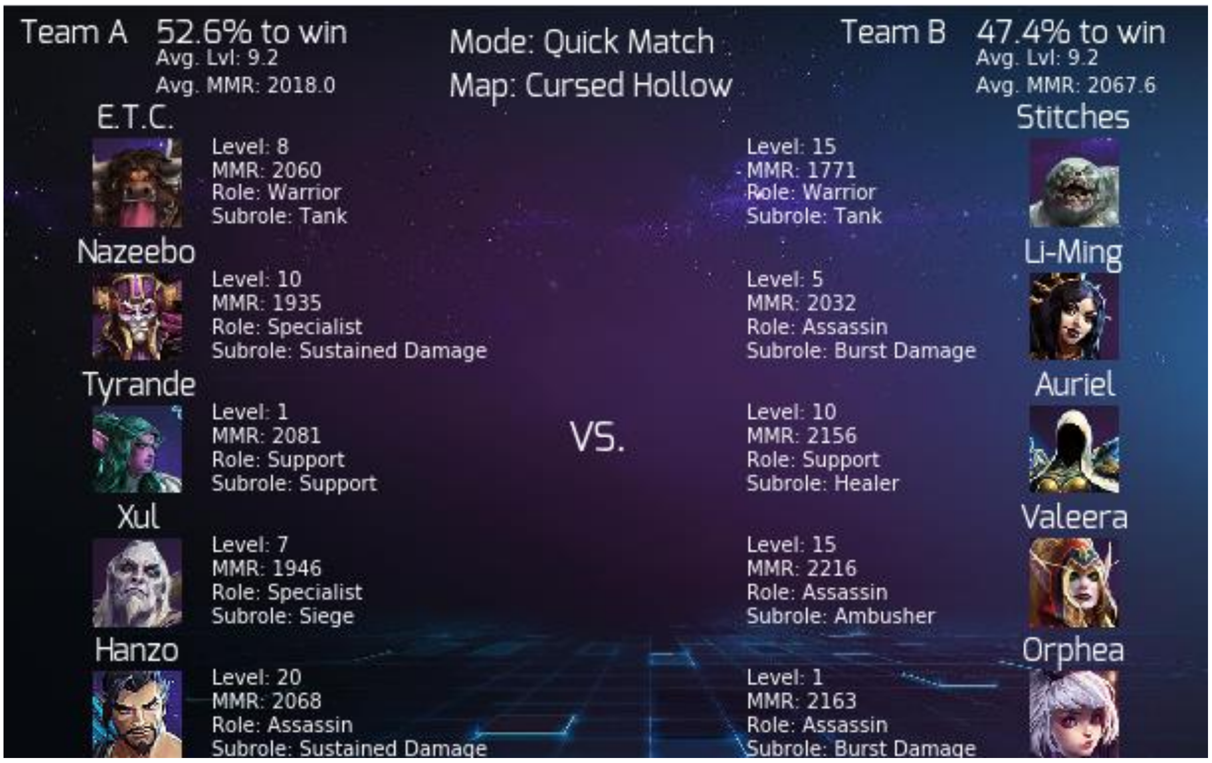
Another factor is the size of the player base, if the player base is large the sampling system would need a computing power to process all the match ups in due time. In the case of heroes of the storm, Blizzard has access to a large amount of processing power on their servers due to their history with world of warcraft (Blizzard, 2009). If there isn't enough power to process match ups for the player base, players will be forced to wait longer which will lead to frustration and boredom.

The location of players within in the world also needs to be considered. Even if a matchup is a 'fair' match but the players are from different sides of the world, the match itself has a high chance of having latency issues. These issues will lead to instances of lag in the game and in turn make the game not as fun for the players. Because of this each player will be matched up based on location with aiming to match players that are close to one another, this will work in tangent with the sampling system to provide players with the best experience possible.

Overall, I feel this proposed matchmaking system will improve the quality of heroes of the storm matches using the model while still taking fun into account in the form of low wait times and minimal latency.

7. Evaluation

Overall, I feel my work was a good success in helping understand the features of a 'fair' game, though I did come across one problem. If you switch the teams in a match (switch team A with team B and vice versa) it will give different win percentages for each team as seen in figures 14-17.


Team A 52.6% to win
Avg. Lvl: 9.2
Avg. MMR: 2018.0

E.T.C.
Level: 8
MMR: 2060
Role: Warrior
Subrole: Tank

Nazeebo
Level: 10
MMR: 1935
Role: Specialist
Subrole: Sustained Damage

Tyrande
Level: 1
MMR: 2081
Role: Support
Subrole: Support

Xul
Level: 7
MMR: 1946
Role: Specialist
Subrole: Siege

Hanzo
Level: 20
MMR: 2068
Role: Assassin
Subrole: Sustained Damage

Mode: Quick Match
Map: Cursed Hollow

VS.

Team B 47.4% to win
Avg. Lvl: 9.2
Avg. MMR: 2067.6

Stitches
Level: 15
MMR: 1771
Role: Warrior
Subrole: Tank

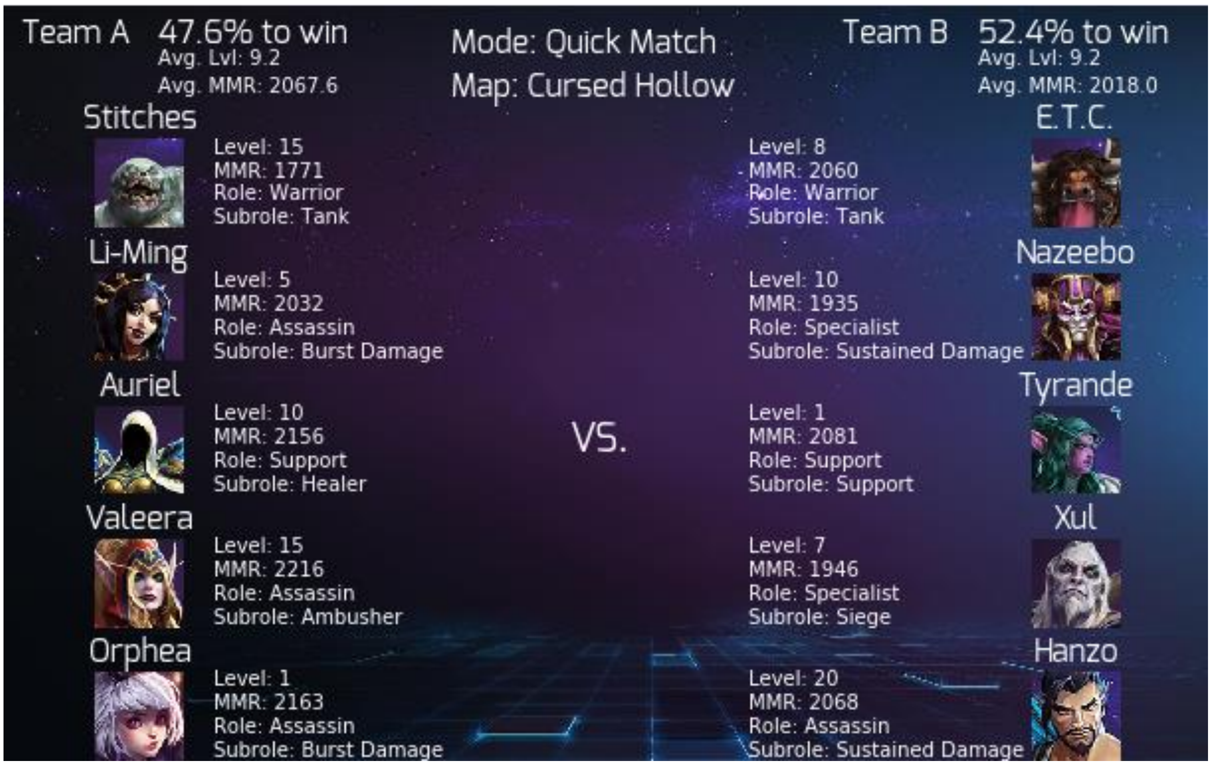
Li-Ming
Level: 5
MMR: 2032
Role: Assassin
Subrole: Burst Damage

Auriel
Level: 10
MMR: 2156
Role: Support
Subrole: Healer

Valeera
Level: 15
MMR: 2216
Role: Assassin
Subrole: Ambusher

Orphea
Level: 1
MMR: 2163
Role: Assassin
Subrole: Burst Damage

Figure 14: Match 1


Team A 47.6% to win
Avg. Lvl: 9.2
Avg. MMR: 2067.6

Stitches
Level: 15
MMR: 1771
Role: Warrior
Subrole: Tank

Li-Ming
Level: 5
MMR: 2032
Role: Assassin
Subrole: Burst Damage

Auriel
Level: 10
MMR: 2156
Role: Support
Subrole: Healer

Valeera
Level: 15
MMR: 2216
Role: Assassin
Subrole: Ambusher

Orphea
Level: 1
MMR: 2163
Role: Assassin
Subrole: Burst Damage

Mode: Quick Match
Map: Cursed Hollow

VS.

Team B 52.4% to win
Avg. Lvl: 9.2
Avg. MMR: 2018.0

E.T.C.
Level: 8
MMR: 2060
Role: Warrior
Subrole: Tank

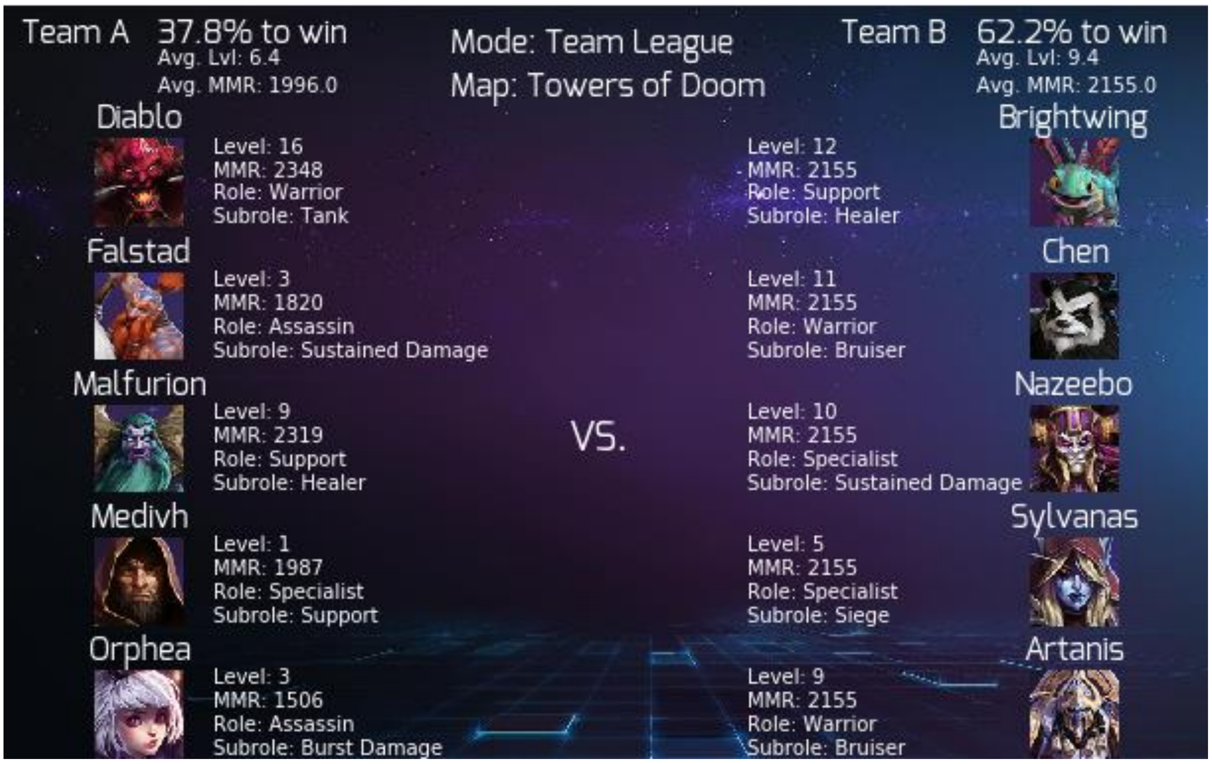
Nazeebo
Level: 10
MMR: 1935
Role: Specialist
Subrole: Sustained Damage

Tyrande
Level: 1
MMR: 2081
Role: Support
Subrole: Support

Xul
Level: 7
MMR: 1946
Role: Specialist
Subrole: Siege

Hanzo
Level: 20
MMR: 2068
Role: Assassin
Subrole: Sustained Damage

Figure 15: Match 1 with the teams switched


Team A 37.8% to win
Avg. Lvl: 6.4
Avg. MMR: 1996.0

Mode: Team League
Map: Towers of Doom

Team B 62.2% to win
Avg. Lvl: 9.4
Avg. MMR: 2155.0

Diablo
Level: 16
MMR: 2348
Role: Warrior
Subrole: Tank

Falstad
Level: 3
MMR: 1820
Role: Assassin
Subrole: Sustained Damage

Malfurion
Level: 9
MMR: 2319
Role: Support
Subrole: Healer

Medivh
Level: 1
MMR: 1987
Role: Specialist
Subrole: Support

Orphea
Level: 3
MMR: 1506
Role: Assassin
Subrole: Burst Damage

VS.

Brightwing
Level: 12
MMR: 2155
Role: Support
Subrole: Healer

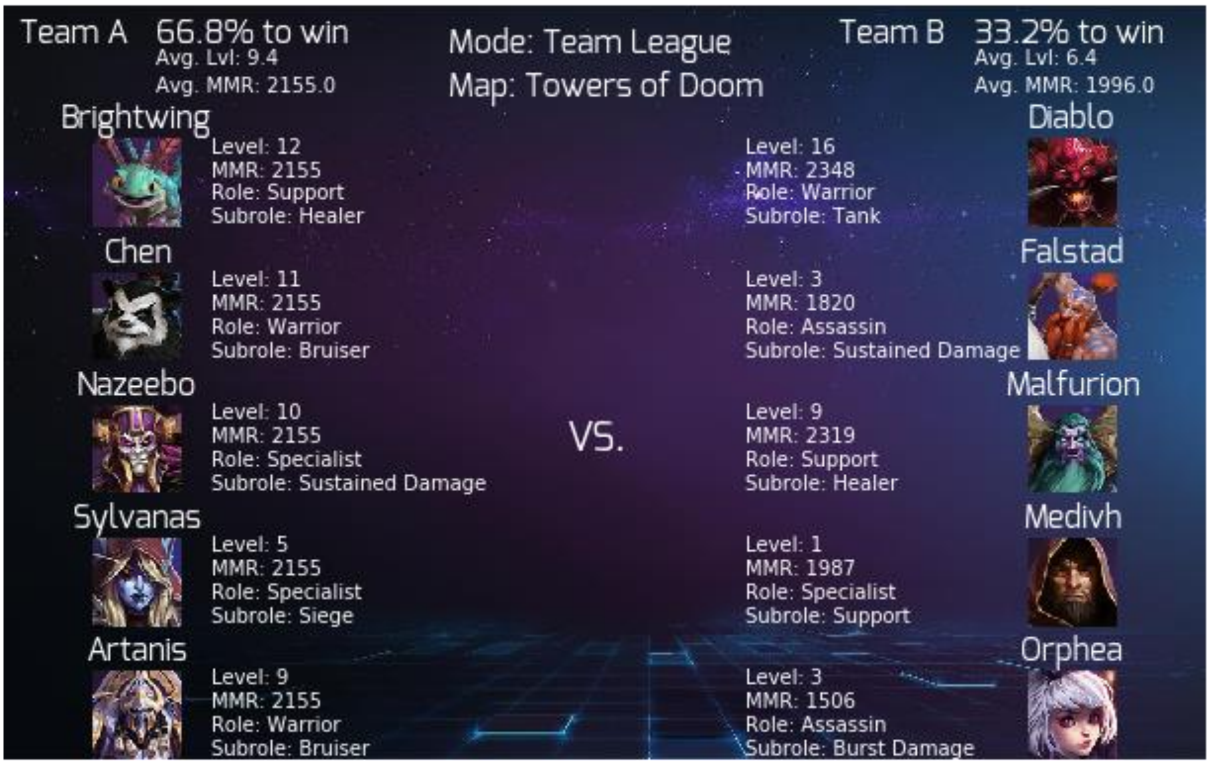
Chen
Level: 11
MMR: 2155
Role: Warrior
Subrole: Bruiser

Nazeebo
Level: 10
MMR: 2155
Role: Specialist
Subrole: Sustained Damage

Sylvanas
Level: 5
MMR: 2155
Role: Specialist
Subrole: Siege

Artanis
Level: 9
MMR: 2155
Role: Warrior
Subrole: Bruiser

Figure 16: Match 2


Team A 66.8% to win
Avg. Lvl: 9.4
Avg. MMR: 2155.0

Mode: Team League
Map: Towers of Doom

Team B 33.2% to win
Avg. Lvl: 6.4
Avg. MMR: 1996.0

Brightwing
Level: 12
MMR: 2155
Role: Support
Subrole: Healer

Chen
Level: 11
MMR: 2155
Role: Warrior
Subrole: Bruiser

Nazeebo
Level: 10
MMR: 2155
Role: Specialist
Subrole: Sustained Damage

Sylvanas
Level: 5
MMR: 2155
Role: Specialist
Subrole: Siege

Artanis
Level: 9
MMR: 2155
Role: Warrior
Subrole: Bruiser

VS.

Diablo
Level: 16
MMR: 2348
Role: Warrior
Subrole: Tank

Falstad
Level: 3
MMR: 1820
Role: Assassin
Subrole: Sustained Damage

Malfurion
Level: 9
MMR: 2319
Role: Support
Subrole: Healer

Medivh
Level: 1
MMR: 1987
Role: Specialist
Subrole: Support

Orphea
Level: 3
MMR: 1506
Role: Assassin
Subrole: Burst Damage

Figure 17: Match 2 with the teams switched

As you can see the team A in match one switches to 52.4% from 52.6% and in match 2 the team A switches from 37.8% to 33.2%. This is due to each team being trained on different set of data, meaning they give overall different predictions. A solution to this could be training the data on a match then train the data on the same match with the team flipped, I decided against this as you have a risk of overfitting the model due to using the same matches twice. The best solution would be to use more unique data to train the model instead, doing this will bring the percentages closer together without the risk of overfitting the model. Even though the model was trained on 757,924 matches it's a drop in the ocean to the number of unique matchups that can occur with trillions of possibilities, using combinatorics we can find out the total number of unique matchups.

$$\frac{84!}{(5!(84-5)!)} = 30{,}872{,}016 \quad \frac{(2+30{,}872{,}016-1)!}{(2!(30{,}872{,}016-1)!)} = 47{,}654{,}070{,}293{,}173{,}695$$

Figure 18: The first equation is how many unique combinations of the 84 heroes you can have on one team for quick play while the second equation is how many unique matchups there are

Figure 18 shows how many total possibilities for quick match there are at over 47 trillion match ups. With 757,924 matches being only 0.000000001590% of the total possibilities and that's even considering the 757,924 matchups as all unique, which they are not. Meaning a project like this requires large amounts of data to get the most accurate win rates, which will only become more apparent when new heroes are added to the game and therefore the number of unique matchups rises.

Though overall the win rates when the teams are flipped are still fairly close even though 757,924 matches have been used to train the model. I suggest even just doubling the number of matches placed into the model will pull the percentages near in line with one another.

7.1 Previous Work

Previous work on using neural networks to improve matchmaking is limited with Delalleau's paper being the main body of work (Delalleau et al., 2012). I feel I improved on the foundation laid by the paper with a point in the conclusion stating that using more data to train the neural network would be a good way to carry on research, which I have done in this project by having over 200 times more games in my dataset then they did. This paper was also performed on an FPS which has a different structure to MOBAs like heroes of the storm, proving that using a neural network to improve matchmaking is not limited to the genre of the video game as this project was done on a MOBA.

I also took 'fairness' into account aiming to get as even matches as possible while other papers focused more on fun (Delalleau et al., 2012) (Claypool et al., 2015) (Wang, Yang and Sun, 2015). Fun was still considered though although not the way originally attended (through game statistics). Instead fun was considered in the form of making sure wait times are not too long and making sure latency is as low as possible. Also, these papers focused on using more features than just playing ranking to understand matches better which was do in my project as well.

I considered the importance of team compositions in my project as stated by Tanuar (Tanuar et al., 2018). This can be seen by using the sub roles as features within the dataset and the table of team composition win rates in table 2. This overall helped the model gain a better understanding of a 'fair' match.

The only main idea from these papers that I didn't implement was player's play styles. (Wang, Yang and Sun, 2015) This is mainly due to the dataset being restricting and not providing player IDs meaning I could only focus on the statistics know before a match like MMR, other stats that would show playstyle such as damage done couldn't be considered due to it being a stat know after the match is finished.

7.2 Professional, ethical, social, security and legal issues

An ethical issue with matchmaking is that you have control of which players you match with other players, for instance in the Claypool paper it was proposed to matchmake players into games they would likely win if they were on a losing streak (Claypool et al., 2015). Activision even published a patent that suggest a system to match players that purchased in game items through microtransactions with players that didn't have those items, making the players who didn't purchase the items possible purchase them due to seeing them in action and feeling excluded (Activision Publishing, Inc., 2015). As my system aims to match players based on the 'fairness' of the match generated by the model and won't place players in matches that can be seen as unethical, such as the examples shown above.

Another ethical issue this time related to the model is that it could be used to solicitate gambling. Due to having a 61.73% accuracy on predicting the winner on the initial validation data and 61.49% on the hold data, people could possibly gamble on matches and will always win against the house due to the model guessing correctly more than 50% of the time. Due to the heroes of the storm esports scene shutting down in December 2018 gambling sights aren't as common (Blizzard, 2018). But the principle still stands and if models are trained on other games with a popular gambling scene, the model could be used unethically.

8. Conclusion

Overall a model was created that could predict the outcome of heroes of the storm matches 61.73% of the time and in turn learned the features of a 'fair' heroes of the storm match from a normalised dataset of 392 features made from data from hotslogs.com. This model was then applied to produce graphics showing the details of individual games such as the win percentage for each team, these graphics where then used to explain how in theory the model would be successfully applied to the matchmaking system.

The most significant part of this project was normalising the dataset it took a while to decide on what features I want to implement in my neural network from the original 3 .csv files and having to use one hot encoding due to most of the features being binary categories rather than weights. Though the majority of the time was spent running training on my neural networks as you expect with a project of this type, it was interesting to play with the layers and see how they affect the accuracy of the model. A decent amount of time was also spent on the final graphic for matches to make it appealing, it went through multiple iterations such as not having the map and mode on it until the final product was made.

Overall, I feel my project answers the question ‘Can machine learning algorithms be used to improve video game matchmaking?’. Although it cannot be truly proven until applied to an actual video game matchmaking system. Though using a neural network to learn how features of a match allows us to gain a deeper understanding to what entitles a ‘fair’ match than just MMR alone does, which could in theory help the matchmaking system if applied.

8.1 Further Work

Further work in this area could aim to increase the amount of data placed into the neural network and see if there’s any improvement in the accuracy of the model, depending on how larger the period of time the data has been spread across it would also have to take game patches into account due to heroes strength changing after patches. Research could also be done into other video games genres that aren’t MOBAs or other MOBAs such as DOTA 2 or league of legends and see if machine learning could help improve matchmaking in the respective game. PCA (Principal component analysis) could also be an area to look into due to it cutting down on the number of features implemented in the network, due to my lack of experience with machine learning I decided not to implement this. The final piece of further work that could be done is actually applying the model to a matchmaking system and seeing if it improves it, this would have to be done by Blizzard themselves or attempted by other developers using a model trained on a separate game.

Appendices

References

Activision Publishing, Inc. (2015). *System and method for driving microtransactions in multiplayer video games*. 9,789,406

Blizzard. (2009). *WoW's Back End: 10 Data Centers, 75,000 Cores*. [online] Available at: https://www.datacenterknowledge.com/archives/2009/11/25/wows-back-end-10-data-centers-75000-cores

Blizzard. (2018). *Heroes of the Storm News . . .*. [online] Available at: https://news.blizzard.com/en-us/blizzard/22833558/heroes-of-the-storm-news

Claypool, M., Decelle, J., Hall, G. and O'Donnell, L. (2015). Surrender at 20? Matchmaking in league of legends. *2015 IEEE Games Entertainment Media Conference (GEM)*

Dataaspirant. (2017). *How the Naive Bayes Classifier works in Machine Learning*. [online] Available at: http://dataaspirant.com/2017/02/06/naive-bayes-classifier-machine-learning/

Delalleau, O., Contal, E., Thibodeau-Laufer, E., Ferrari, R., Bengio, Y. and Zhang, F. (2012). Beyond Skill Rating: Advanced Matchmaking in Ghost Recon Online. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(3), pp.167-177

Elo, A. (2008). *The rating of chessplayers, past and present*. Bronx, N.Y.: Ishi Press International

Glickman, M. (2000). [online] Glicko.net. Available at: http://www.glicko.net/glicko/glicko2.pdf

Google (2013). *System and method for addressing overfitting in a neural network*. US9406017B2

He, K., Zhang, X., Ren, S. and Sun, J. (2015). *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. [online] arXiv.org. Available at: https://arxiv.org/abs/1502.01852

Herbrich, R. and Graepel, T. (2006). *TrueSkill(TM): A Bayesian Skill Rating System*. [online] Microsoft Research. Available at: https://www.microsoft.com/en-us/research/publication/trueskilltm-a-bayesian-skill-rating-system-2/

Heroes of the Storm. (2018a). *Heroes of the Storm*. [online] Available at: https://heroesofthestorm.com/en-us/blog/22548919/#battlegrounds

Heroes of the Storm. (2018b). *Heroes of the Storm*. [online] Available at: https://heroesofthestorm.com/en-us/blog/21940930/

Heroes of the Storm. (2019). *Heroes of the Storm*. [online] Available at: https://heroesofthestorm.com/en-gb/game/ranked-play/

Heroes of the Storm Forums. (2018). *Blizzard*. [online] Available at: https://us.forums.blizzard.com/en/heroes/t/horrible-experience-with-ranked-this-season/2633

Hotslogs.com. (2018a). *Hotslogs.com*. [online] Available at: https://www.hotslogs.com/Info/API

Hotslogs.com. (2018b). *Hotslogs.com*. [online] Available at: https://www.hotslogs.com/Sitewide/HeroAndMapStatistics

Hotslogs.com. (2019a). *Heroes of the Storm Popular Team Compositions* | *HOTS Logs*. [online] Available at: https://www.hotslogs.com/Sitewide/TeamCompositions

Hotslogs.com. (2019b). *Hotslogs.com*. [online] Available at: https://www.hotslogs.com/Info/MMRInformation

KDnuggets. (2016). *KDnuggets*. [online] Available at: https://www.kdnuggets.com/2016/06/r-python-top-analytics-data-mining-data-science-software.htmlhttps://www.kdnuggets.com/2016/06/r-python-top-analytics-data-mining-data-science-software.html

Kingma, D. and Ba, J. (2015). ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION. [online] Available at: https://arxiv.org/abs/1412.6980v9.

McGeathy, T. (2018). *2018 Gameplay Update Q&A – November 29, 2017 : heroesofthestorm*. [online] Reddit.com. Available at: https://www.reddit.com/r/heroesofthestorm/comments/7gfatp/2018_gameplay_update_qa_november_29_2017/dqiyw9f/

Minka, T., Cleven, R. and Zaykov, Y. (2018). *TrueSkill 2: An improved Bayesian skill rating system*. [online] Microsoft Research. Available at: https://www.microsoft.com/en-us/research/publication/trueskill-2-improved-bayesian-skill-rating-system/

Newell, A. (2018). *What is Elo? An explanation for competitive gaming's hidden rating system*. [online] Dot Esports. Available at: https://dotesports.com/general/news/elo-ratings-explained-20565

O'Dell, J. (2014). *Finding the perfect match – GuildWars2.com*. [online] GuildWars2.com. Available at: https://www.guildwars2.com/en/news/finding-the-perfect-match/

Overwatch. (2018). *Overwatch*. [online] Available at: https://playoverwatch.com/en-us/news/21363037

Overwatch Forums. (2018). *Blizzard*. [online] Available at: https://us.forums.blizzard.com/en/overwatch/t/why-handicapping-mmr-is-wrong-for-competitive-play/646

Oxford Dictionaries. (2019). *fair | Definition of fair in English by Oxford Dictionaries*. [online] Available at: https://en.oxforddictionaries.com/definition/fair

PwC. (2018). *Outlook segment findings*. [online] Available at: https://www.pwc.com/gx/en/industries/tmt/media/outlook/segment-findings.html

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. (2014). *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. [online] Cs.toronto.edu. Available at: https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

Tanuar, E., Abbas, B., Trisetyarso, A., Kang, C., Gaol, F. and Suparta, W. (2018). Back propagation neural network experiment on team matchmaking MOBA game. *2018 International Conference on Information and Communications Technology (ICOIACT)*

Vasudev, R. (2017). *What is One Hot Encoding? Why And When do you have to use it?*. [online] Hacker Noon. Available at: https://hackernoon.com/what-is-one-hot-encoding-why-and-when-do-you-have-to-use-it-e3c6186d008f

Wang, H., Yang, H. and Sun, C. (2015). Thinking Style and Team Competition Game Performance and Enjoyment. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(3), pp.243-254

Project Proposal

Taylor Lee – Project Proposal

1.1 Research Question

My research question is “Can machine learning algorithms be used to improve video game matchmaking?” Which will use match data from heroes of the storm (hots) to find the features of a balanced game and with this trained algorithm “dummy” matches will be set up

1.2 Keywords

Machine Learning; Matchmaking; Video Games; Game Development;

1.3 Project Title

My project title is “Competitive matchmaking in video games using machine learning algorithms”

1.4 Client, Audience and Motivation

The main group that would be interested in these findings are game developers, they would benefit from improved matchmaking systems due to it improving the satisfaction of the players of their games who in turn will stay playing the developers game for longer, generating more income for the developer overtime and increasing the positivity of the game’s community.

1.5 Project Plan

1. Gather data from heroes of the storm games
2. Create a machine learning program that learns features of bad games and good games e.g. bad games are one sides while good games are not
3. Train the program on the heroes of the storm data
4. Test the program on a select group of test data
5. Analyse the test results
6. Use the trained program to sort “dummy” players into games
7. Analyse these results
8. Write up report

2.1 Abstract

Matchmaking is a crucial part of competitive video games. It allows players of the game to be placed in matches that give them the best experience possible. Unfortunately, matchmaking systems are not always perfect and can place players in unfair matches. This in turn makes the game far less enjoyable for either side as there is no challenge involved for the winning team and the losing team faces constant defeat. Match data from *Heroes of the Storm* a Massive Online Battle Arena (MOBA) developed by *Blizzard Entertainment* and machine learning techniques, in the case neural networks, were used to find the features of a "fair" match in this case a 45%-55% win rate for either side. This trained algorithm was then applied to a matchmaking system that created "fair" matches using a list of players, these "fair" matches were then compared to actual matches set up by the in-game matchmaker with changes in how the matchmaking performed being noted.

2.2 Literature Review

Multiple people have proposed ways to use machine learning algorithms to improve video game matchmaking in the past, for instance Delalleau (Delalleau et al., 2012) trained a neural network using match data from Ghost Recon Online to improve matchmaking within the game. They concluded that just using a skill value isn't enough to evaluate the balance of a match, with more to be gained by using a richer player profile that contains multiple statistics collected within the game. They also argued that fun is more important than balance and showed that it could be used as the main principle in a matchmaking system. This paper heavily relates to my research as they used the trained neural network to help find balanced matches from a pool of players which I will be doing to.

One of the problems with this paper was the number of matches used to train this model which was 3937 compared to the 100,000+ used in my neural network. Ghost Recon Online was in an early beta-test when the neural network was implemented meaning most players were still learning the game and weren't playing to their full potential also players were not matched by skill in the beta but randomly, both of these points have led to the majority of matches being unbalanced and therefore the neural network wasn't trained to its full potential on the given match data. Due to Heroes of the Storm being 3 years old at the time of writing with an established player base and with my larger set of match data I feel that the neural network will be used more effectively.

Wang (Wang, Yang and Sun, 2015) investigated the idea of fun further by identifying players roles within the game of League of Legends and how teams with certain roles had more fun matches using a neural network. They also performed a survey with players of the game that stated that matches that were short, were more enjoyable. Though they later stated that other kinds of game statistics then match duration, would be a better measure for enjoyment levels, especially as other games may not have a close relation to match duration as League of Legends.

Claypool (Claypool et al., 2015) further investigated the balance and enjoyment of matchmaking using League of Legends matches. Concluding that games that were balanced by the ranks of players were found unbalanced by the losing team and games that were unbalanced were found enjoyable by the winning team. Which lead to them proposing that the matchmaking system should place the player on an unbalanced team in their favour when appropriate, to increase enjoyment.

Tanuar (Tanuar et al., 2018) investigated team compositions within the MOBA game Mobile Legend: Bang Bang using neural networks. Concluding that only 24.6% of team combinations of heroes have more than 50% chance of winning a game. This in turn puts in perspective how important the composition of a team factors into that teams win rate.

The research preformed in this paper differs from the ones above as they mainly focus on judging the enjoyment of players rather than aiming to always place players into balanced matches, which this paper focuses on. Though some idea from these papers will be used such as the idea of using in game statistics to improve matchmaking from Delalleau's work and the idea of team compositions being extremely important in MOBAs from Tanuar's work.

2.3 Ethics

Matchmaking can be seen as a way of artificially sorting people, as you're taking away the randomness and fairness of matchmaking, especially in the case of Claypool's proposal who suggested weighing the win rate in favor of one player for the sake of fun. To solve this my project will aim to make all the matches fair as possible for both teams.

Activision created a patent in 2015 that would matchmake players based on their in-game purchases to drive purchases of microtransaction. (Activision Publishing, Inc., 2015) For instance, placing a player without a character skin in a match with a player that has purchased the skin to encourage the first player to purchase the skin. This can be seen as an ethical issue due to in game purchases being forced upon the player. As my matchmaking solely focuses on balance of matches this will not be a problem.

2.4 Bibliography

Activision Publishing, Inc. (2015). *System and method for driving microtransactions in multiplayer video games*. 9,789,406.

Claypool, M., Decelle, J., Hall, G. and O'Donnell, L. (2015). Surrender at 20? Matchmaking in league of legends. *2015 IEEE Games Entertainment Media Conference (GEM)*.

Delalleau, O., Contal, E., Thibodeau-Laufer, E., Ferrari, R., Bengio, Y. and Zhang, F. (2012). Beyond Skill Rating: Advanced Matchmaking in Ghost Recon Online. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(3), pp.167-177.

Tanuar, E., Abbas, B., Trisetyarso, A., Kang, C., Gaol, F. and Suparta, W. (2018). Back propagation neural network experiment on team matchmaking MOBA game. *2018 International Conference on Information and Communications Technology (ICOIACT)*.

Wang, H., Yang, H. and Sun, C. (2015). Thinking Style and Team Competition Game Performance and Enjoyment. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(3), pp.243-254.

Project Process

Aims

Complete a literature review and search of machine learning used in a video game matchmaking environment

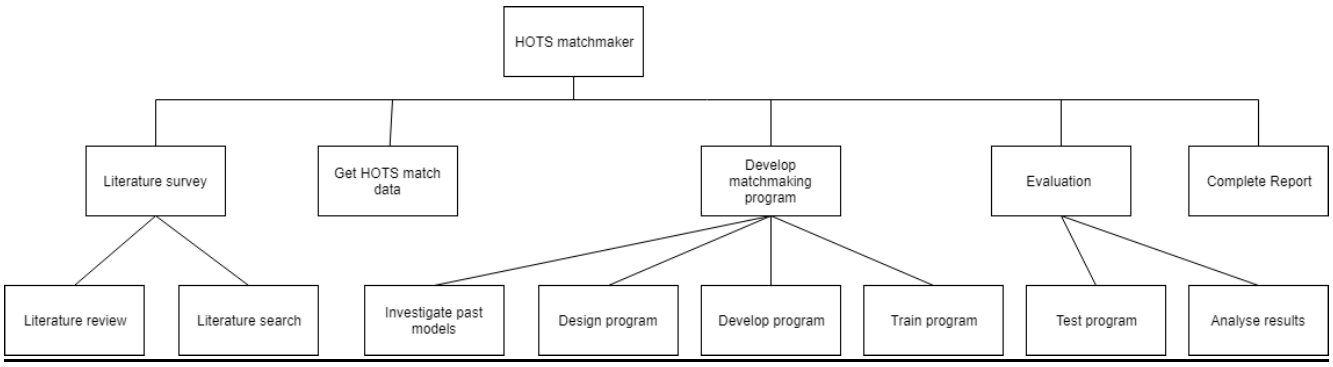
Collect data from hotslogs.com for use in the Tensorflow machine learning program

Develop the machine learning program in Tensorflow

Test the machine learning program produces the desired output

Write up report

Work Breakdown



Time estimation

Literature search – 1 weeks

Literature review – 2 weeks

Gather data – 1 weeks

Investigate past models – 3 weeks

Design Program – 4 weeks

Develop program including training – 6 weeks

Testing – 2 weeks

Analyse results – 2 weeks

Report completion – 3 weeks

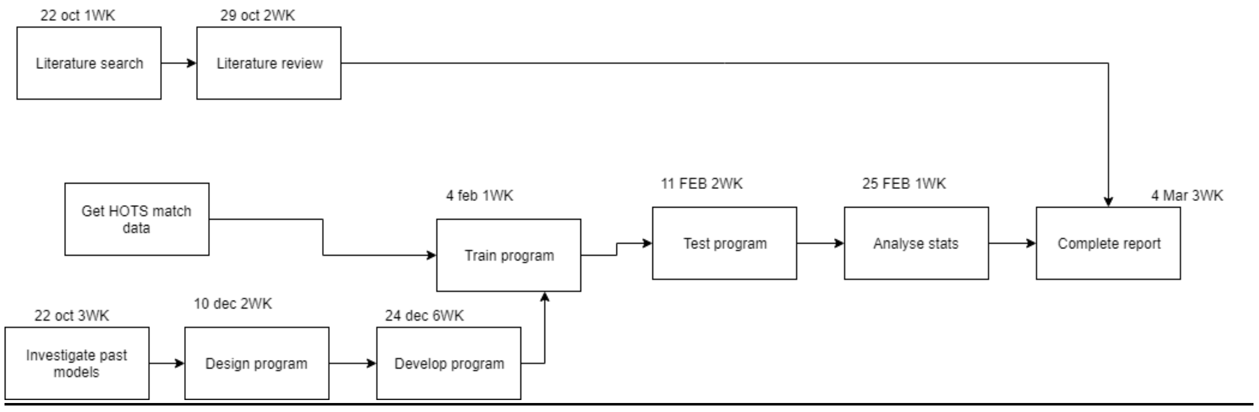
Total effort – 24 weeks

Milestones

Finish literature survey
Gather necessary data
Developed program
Trained program
Results analysed

Report finished

Activity network

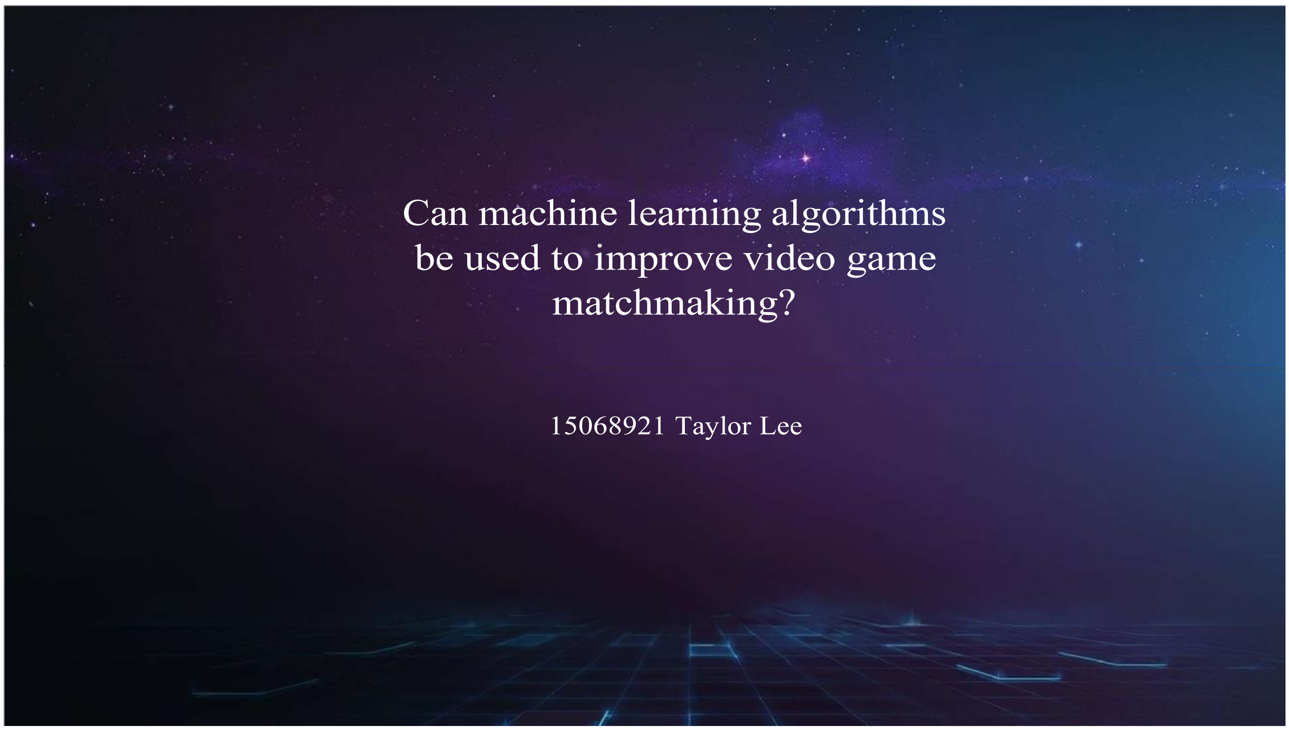


Risks

Illness – 3
Data loss – 5
Time underestimation – 6
Learning TensorFlow – 8

Most risks avoided by sticking to schedule

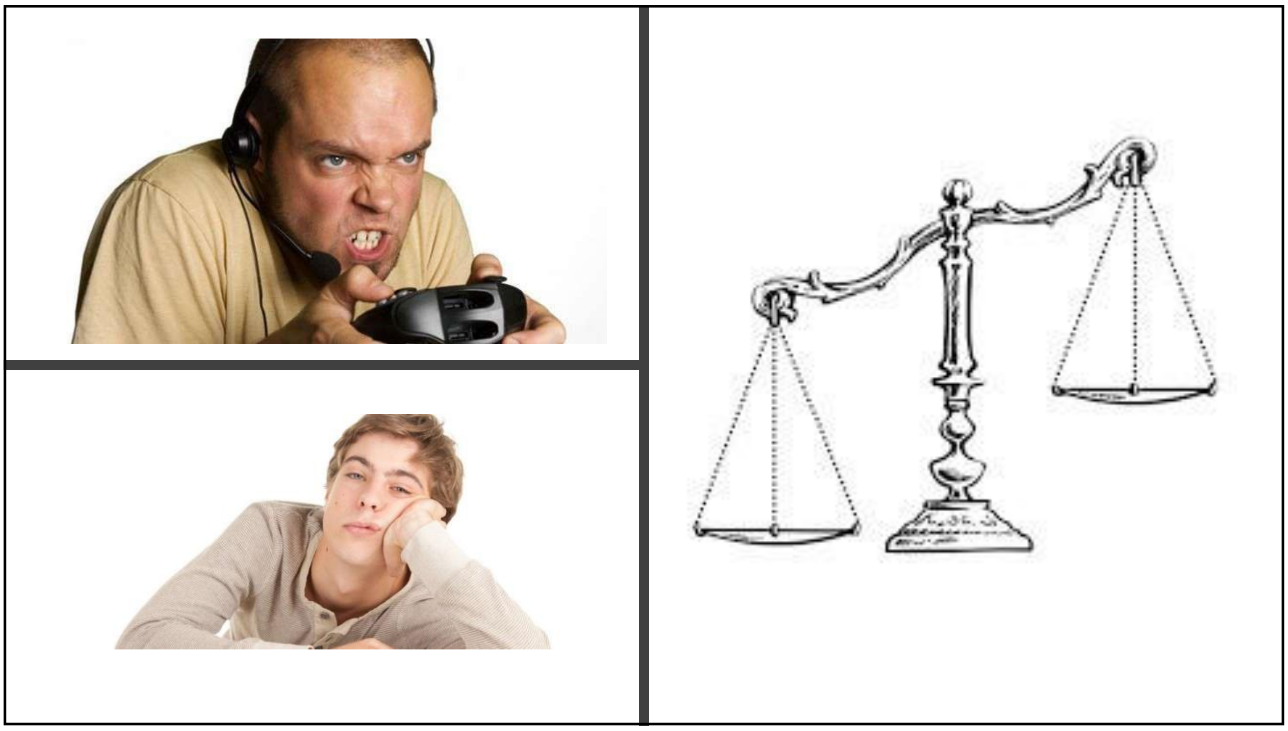
Presentation



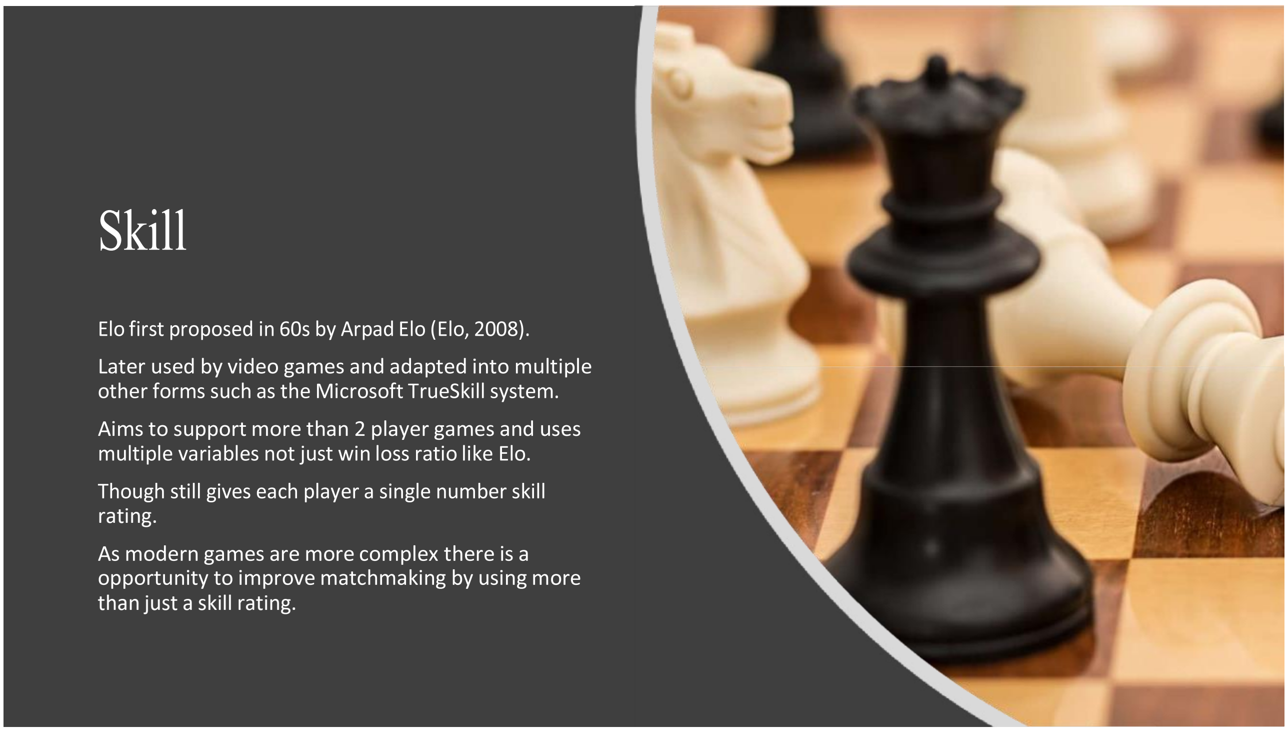
Can machine learning algorithms be used to improve video game matchmaking?

15068921 Taylor Lee

1



2



Skill

Elo first proposed in 60s by Arpad Elo (Elo, 2008).

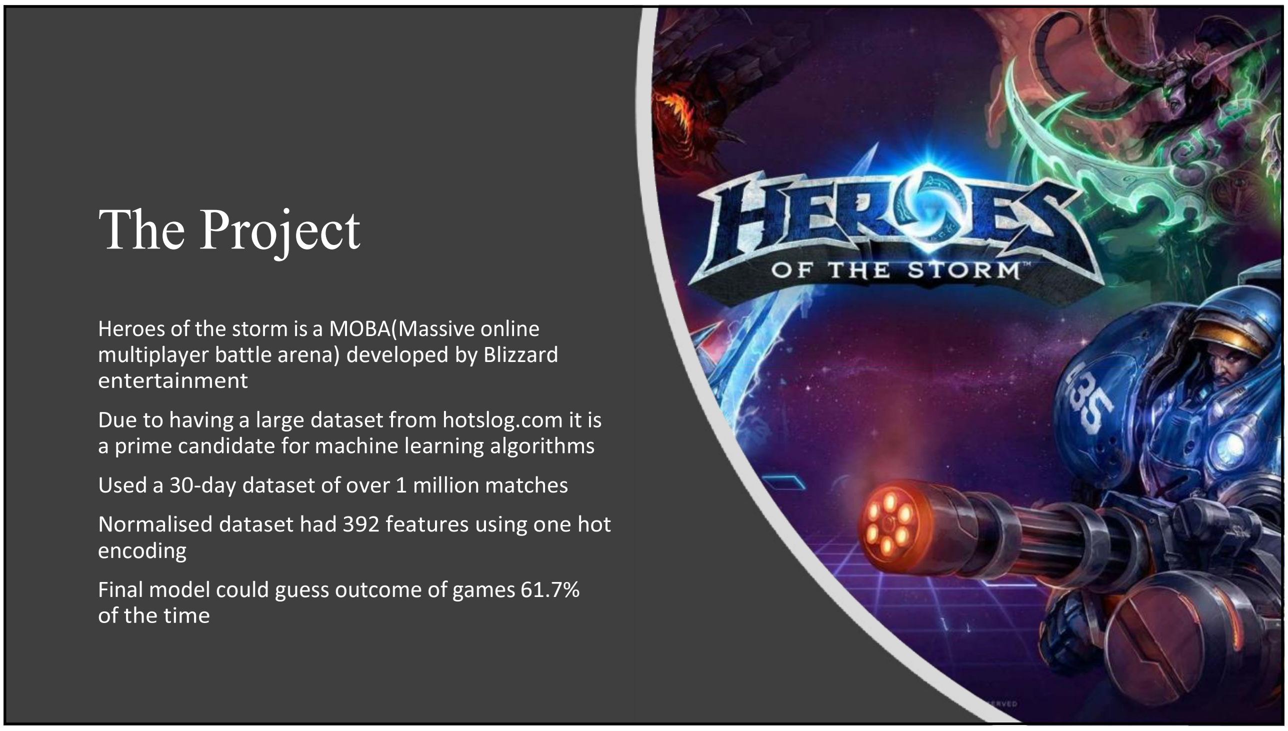
Later used by video games and adapted into multiple other forms such as the Microsoft TrueSkill system.

Aims to support more than 2 player games and uses multiple variables not just win loss ratio like Elo.

Though still gives each player a single number skill rating.

As modern games are more complex there is a opportunity to improve matchmaking by using more than just a skill rating.

3


HEROES
OF THE STORM™

The Project

Heroes of the storm is a MOBA(Massive online multiplayer battle arena) developed by Blizzard entertainment

Due to having a large dataset from hotslog.com it is a prime candidate for machine learning algorithms

Used a 30-day dataset of over 1 million matches

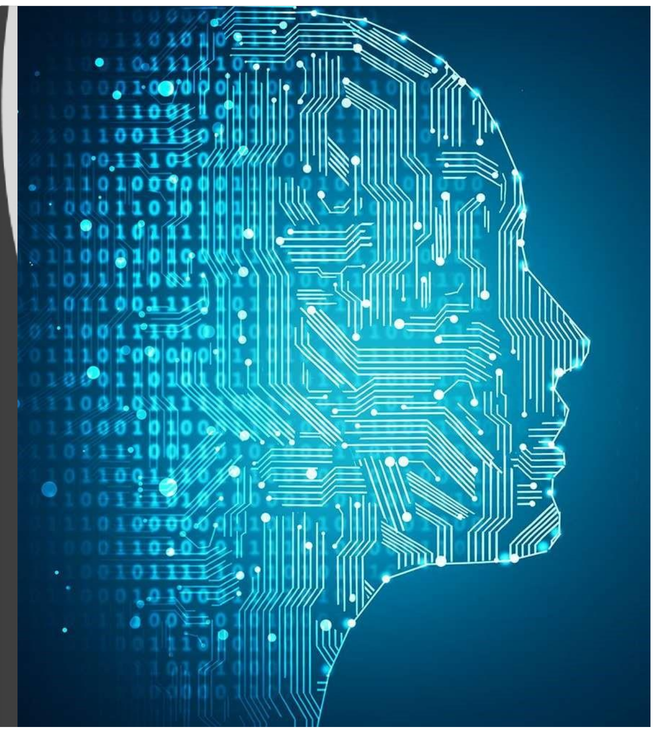
Normalised dataset had 392 features using one hot encoding

Final model could guess outcome of games 61.7% of the time

4

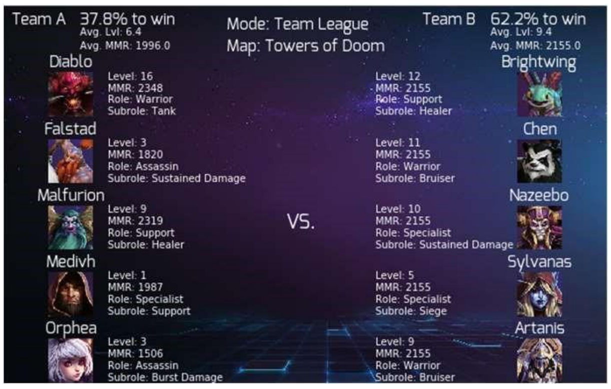
The neural network

- Linear dense layer with a 1000 neurons
- PReLU layer
- Dropout layer
- Sigmoid dense layer for output
- Adam optimizer



5

Problem



Team A 37.8% to win
Avg. Lvl: 6.4
Avg. MMR: 1996.0

Mode: Team League
Map: Towers of Doom

Team B 62.2% to win
Avg. Lvl: 9.4
Avg. MMR: 2155.0

Diablo
Level: 16
MMR: 2348
Role: Warrior
Subrole: Tank

Falstad
Level: 3
MMR: 1820
Role: Assassin
Subrole: Sustained Damage

Malfurion
Level: 9
MMR: 2319
Role: Support
Subrole: Healer

Medivh
Level: 1
MMR: 1987
Role: Specialist
Subrole: Support

Orphea
Level: 3
MMR: 1506
Role: Assassin
Subrole: Burst Damage

VS.

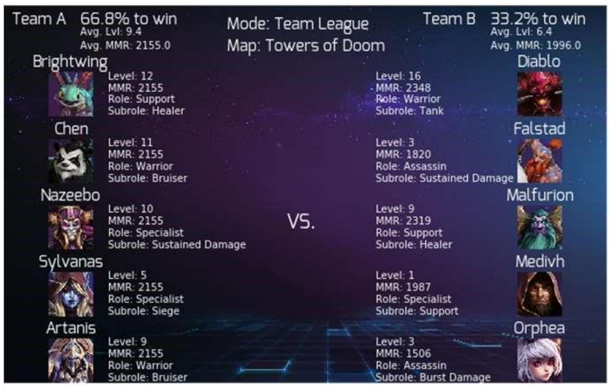
Brightwing
Level: 12
MMR: 2155
Role: Support
Subrole: Healer

Chen
Level: 11
MMR: 2155
Role: Warrior
Subrole: Bruiser

Nazeebo
Level: 10
MMR: 2155
Role: Specialist
Subrole: Sustained Damage

Sylvanas
Level: 5
MMR: 2155
Role: Specialist
Subrole: Siege

Artanis
Level: 9
MMR: 2155
Role: Warrior
Subrole: Bruiser



Team A 66.8% to win
Avg. Lvl: 9.4
Avg. MMR: 2155.0

Mode: Team League
Map: Towers of Doom

Team B 33.2% to win
Avg. Lvl: 6.4
Avg. MMR: 1996.0

Brightwing
Level: 12
MMR: 2155
Role: Support
Subrole: Healer

Chen
Level: 11
MMR: 2155
Role: Warrior
Subrole: Bruiser

Nazeebo
Level: 10
MMR: 2155
Role: Specialist
Subrole: Sustained Damage

Sylvanas
Level: 5
MMR: 2155
Role: Specialist
Subrole: Siege

Artanis
Level: 9
MMR: 2155
Role: Warrior
Subrole: Bruiser

VS.

Diablo
Level: 16
MMR: 2348
Role: Warrior
Subrole: Tank

Falstad
Level: 3
MMR: 1820
Role: Assassin
Subrole: Sustained Damage

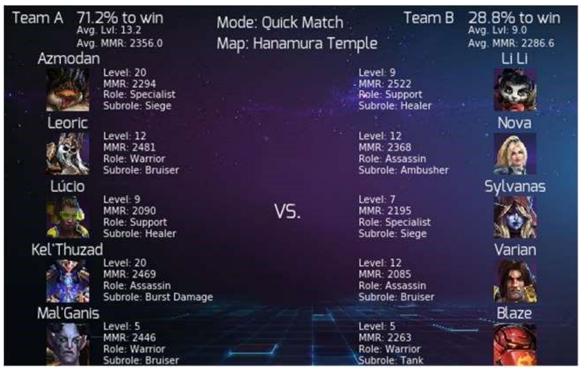
Malfurion
Level: 9
MMR: 2319
Role: Support
Subrole: Healer

Medivh
Level: 1
MMR: 1987
Role: Specialist
Subrole: Support

Orphea
Level: 3
MMR: 1506
Role: Assassin
Subrole: Burst Damage

6

Matchmaking


Team A 71.2% to win
Avg. Lvl: 13.2
Avg. MMR: 2356.0

Mode: Quick Match
Map: Hanamura Temple

Team B 28.8% to win
Avg. Lvl: 9.0
Avg. MMR: 2286.6

Azmodan
Level: 20
MMR: 2294
Role: Specialist
Subrole: Siege

Leoric
Level: 12
MMR: 2481
Role: Warrior
Subrole: Bruiser

Lúcio
Level: 9
MMR: 2090
Role: Support
Subrole: Healer

Kel'Thuzad
Level: 20
MMR: 2469
Role: Assassin
Subrole: Burst Damage

Mal'Ganis
Level: 5
MMR: 2446
Role: Warrior
Subrole: Bruiser

VS.

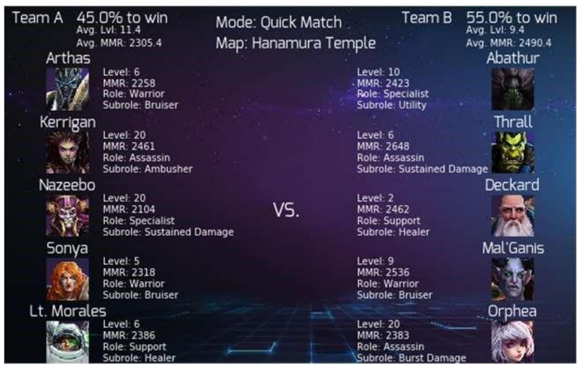
Li Li
Level: 9
MMR: 2522
Role: Support
Subrole: Healer

Nova
Level: 12
MMR: 2368
Role: Assassin
Subrole: Ambusher

Sylvanas
Level: 7
MMR: 2195
Role: Specialist
Subrole: Siege

Varian
Level: 12
MMR: 2085
Role: Assassin
Subrole: Bruiser

Blaze
Level: 5
MMR: 2263
Role: Warrior
Subrole: Tank


Team A 45.0% to win
Avg. Lvl: 11.4
Avg. MMR: 2305.4

Mode: Quick Match
Map: Hanamura Temple

Team B 55.0% to win
Avg. Lvl: 9.4
Avg. MMR: 2490.4

Arthas
Level: 6
MMR: 2258
Role: Warrior
Subrole: Bruiser

Kerrigan
Level: 20
MMR: 2461
Role: Assassin
Subrole: Ambusher

Nazeebo
Level: 20
MMR: 2104
Role: Specialist
Subrole: Sustained Damage

Sonya
Level: 5
MMR: 2318
Role: Warrior
Subrole: Bruiser

Lt. Morales
Level: 6
MMR: 2386
Role: Support
Subrole: Healer

VS.

Abathur
Level: 10
MMR: 2423
Role: Specialist
Subrole: Utility

Thrall
Level: 6
MMR: 2648
Role: Assassin
Subrole: Sustained Damage

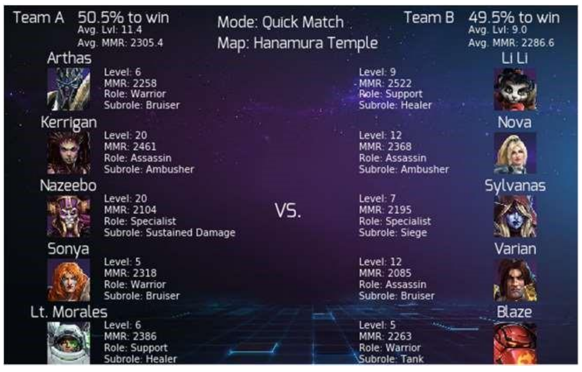
Deckard
Level: 2
MMR: 2462
Role: Support
Subrole: Healer

Mal'Ganis
Level: 9
MMR: 2536
Role: Warrior
Subrole: Bruiser

Orphea
Level: 20
MMR: 2383
Role: Assassin
Subrole: Burst Damage

7

Matchmaking


Team A 50.5% to win
Avg. Lvl: 11.4
Avg. MMR: 2305.4

Mode: Quick Match
Map: Hanamura Temple

Team B 49.5% to win
Avg. Lvl: 9.0
Avg. MMR: 2286.6

Arthas
Level: 6
MMR: 2258
Role: Warrior
Subrole: Bruiser

Kerrigan
Level: 20
MMR: 2461
Role: Assassin
Subrole: Ambusher

Nazeebo
Level: 20
MMR: 2104
Role: Specialist
Subrole: Sustained Damage

Sonya
Level: 5
MMR: 2318
Role: Warrior
Subrole: Bruiser

Lt. Morales
Level: 6
MMR: 2386
Role: Support
Subrole: Healer

VS.

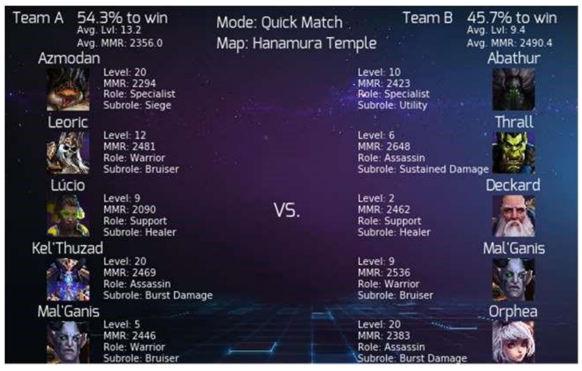
Li Li
Level: 9
MMR: 2522
Role: Support
Subrole: Healer

Nova
Level: 12
MMR: 2368
Role: Assassin
Subrole: Ambusher

Sylvanas
Level: 7
MMR: 2195
Role: Specialist
Subrole: Siege

Varian
Level: 12
MMR: 2085
Role: Assassin
Subrole: Bruiser

Blaze
Level: 5
MMR: 2263
Role: Warrior
Subrole: Tank


Team A 54.3% to win
Avg. Lvl: 13.2
Avg. MMR: 2356.0

Mode: Quick Match
Map: Hanamura Temple

Team B 45.7% to win
Avg. Lvl: 9.4
Avg. MMR: 2490.4

Azmodan
Level: 20
MMR: 2294
Role: Specialist
Subrole: Siege

Leoric
Level: 12
MMR: 2481
Role: Warrior
Subrole: Bruiser

Lúcio
Level: 9
MMR: 2090
Role: Support
Subrole: Healer

Kel'Thuzad
Level: 20
MMR: 2469
Role: Assassin
Subrole: Burst Damage

Mal'Ganis
Level: 5
MMR: 2446
Role: Warrior
Subrole: Bruiser

VS.

Abathur
Level: 10
MMR: 2423
Role: Specialist
Subrole: Utility

Thrall
Level: 6
MMR: 2648
Role: Assassin
Subrole: Sustained Damage

Deckard
Level: 2
MMR: 2462
Role: Support
Subrole: Healer

Mal'Ganis
Level: 9
MMR: 2536
Role: Warrior
Subrole: Bruiser

Orphea
Level: 20
MMR: 2383
Role: Assassin
Subrole: Burst Damage

8

Future work

- Larger dataset
- Different genres
- Possible use PCA to cut down on features
- Apply to a real-life matchmaking system itself

9

Project Log

DATE	TIME SPENT	SUMMARY
18/10/2018	12HR	Submitted project proposal with mini lit review *TOTAL TIME DUE TO NOT TRACKING*
10/11/2018	5HR	Looked at some TensorFlow tutorial to get to grips with it used a cat and dogs dataset to understand how it works
02/01/2019	2HR	Gathered data from HOTS.log to be used within the project contains large amounts of information about player matches
05/01/2019	3HR	Started of introduction to the project describing the need for an improvement to competitive matchmaking collected some references
08/01/2019	2HR	Carried on introduction describing past systems and explaining the use of the word "fair" in the project collected some references
10/01/2019	2HR	Finished off introduction explaining how the project will be carried out
15/01/2019	3HR	Transferred previous literature review and started appending it added new section talking about the history of ELO and its use within video games also talked about the TrueSkill system
17/01/2019	2HR	Appended lit review on Delalleau paper and talked about the complexity of modern gaming
18/01/2019	2HR	Appended lit review on Delalleau paper and talked about the complexity of modern gaming
22/01/2019	4HR	Appended lit review on Claypool and Tanuar paper also added to the conclusion of the lit review finishing the lit review and introduction. Handed in the introduction and lit review.
01/02/2019	4HR	Started work on normalising the dataset and listing possibly good features to carry over to be inputted into the model
02/02/2019	4HR	Finished the first normalisation of the dataset will be test on a classifier to see if improvements can be made

03/02/2019	6HR	Looked into using Naive Baysen classifiers to quickly understand if certain features improve the accuracy of the program. Was then applied to a set of normalised data
06/02/2019	5HR	Made final decision on how to normalise the dataset. Having 393 features +2 hours to normalise the data into one large csv
09/02/2019	6HR	Started work on neural network and experiment with layers tweaking them as I went. Was tried on a 100,000 dataset size for quick learning
12/02/2019	5HR	Looked into the use of Adam optimisers and PreLU layers was found to help improve the neural network
16/02/2019	4HR	Trained on my final neural network with a 61.5% accuracy in finding the winner of a match
18/02/2019	3HR	Started write up for methodology will be finished when final product is done
20/02/2019	3HR	Started write up on data section reaching the end of the game mode section
21/02/2019	3HR	Carried on with data section finishing the Maps and Heroes sections
22/02/2019	6HR	Finished Player section of the report and retrained final neural network gaining 61.7% accuracy
23/02/2019	10HR	Did a large amount of training on multiple different neural networks as I had no records of prior test cases
25/02/2019	4HR	Finished data section of report
26/02/2019	2HR	Started on model section of the report
27/02/2019	5HR	Trained a few more neural networks for more test cases also found neural network had accuracy of 61.49% on holdout dataset of 100,000 showing little chance of overfitting
01/03/2019	3HR	Finished the model section of my report
02/03/2019	5HR	Started worked on final output of project making a graphic using matplotlib and using thumbnail images of heroes
03/03/2019	1HR	Made a final.csv that has 4 matches and applied it to the model to show how the machine learning model would work in the matchmaking system
07/03/2019	10HR	Finished output graphic for the project theming it on heroes of the storm and also started on the results part of the report using the graphic to explain how the matchmaking system would be applied
08/03/2019	6HR	Started results part of the report and finished it also started on evaluation part of report
09/03/2019	10HR	Finished evaluation part of the report and started and finished conclusion as well also added title pages and started on the appendices
10/03/2019	10HR	Copied in source code and adding comments where needed also added test cases to the project and finished presentation

Ethics Form

OXFORD BROOKES UNIVERSITY

TDE Form E1

Faculty of Technology, Design and Environment

Ethics Review Form E1

This form should be completed by the Principal Investigator / Supervisor / Student undertaking a research project which involves human participants. The form will identify whether a more detailed E2 form needs to be submitted to the Faculty Research Ethics Officer.

Before completing this form, please refer to the University **Code of Practice for the Ethical Standards for Research involving Human Participants**, available at http://www.brookes.ac.uk/Research/Research-ethics/, and to any guidelines provided by relevant academic or professional associations.

It is the Principal Investigator / Supervisor who is responsible for exercising appropriate professional judgement in this review. Note that all necessary forms should be fully completed and signed before fieldwork commences.

Project Title: Can machine learning algorithms be used to improve video game matchmaking?

Principal Investigator / Supervisor: Mark Green

Student Investigator: Taylor Lee

		Yes	No
1.	Does the study involve participants who are unable to give informed consent? (e.g. children, people with learning disabilities, unconscious patients)	☐	☒
2.	If the study will involve participants who are unable to give informed consent (e.g. children under the age of 16, people with learning disabilities), will you be unable to obtain permission from their parents or guardians (as appropriate)?	☐	☒
3.	Will the study require the cooperation of a gatekeeper for initial access to groups or individuals to be recruited? (e.g. students, members of a self-help group, employees of a company, residents of a nursing home)	☐	☒

4.	Are there any problems with the participants' right to remain anonymous, or to have the information they give not identifiable as theirs?	☐	☒
5.	Will it be necessary for the participants to take part in the study without their knowledge/consent at the time? (eg, covert observation of people in non-public places?)	☐	☒
6.	Will the study involve discussion of or responses to questions the participants might find sensitive? (e.g. own drug use, own traumatic experiences)	☐	☒
7.	Are drugs, placebos or other substances (e.g. food substances, vitamins) to be administered to the study participants?	☐	☒
8.	Will blood or tissue samples be obtained from participants?	☐	☒
9.	Is pain or more than mild discomfort likely to result from the study?	☐	☒
10.	Could the study induce psychological stress or anxiety?	☐	☒
11.	Will the study involve prolonged or repetitive testing of participants?	☐	☒
12.	Will financial inducements (other than reasonable expenses and compensation for time) be offered to participants?	☐	☒
13.	Will deception of participants be necessary during the study?	☐	☒
14.	Will the study involve NHS patients, staff, carers or premises?	☐	☒

If you have answered 'no' to all the above questions, send the completed form to your Module Leader and keep the original in case you need to submit it with your work.

If you have answered 'yes' to any of the above questions, you should complete the Form E2 available at http://www.brookes.ac.uk/Research/Research-ethics/Ethics-review-forms/
and, together with this E1 Form, email it to the Faculty Research Ethics Officer, whose name can be found at
http://www.brookes.ac.uk/Research/Research-ethics/Research-ethics-officers/

If you answered 'yes' to any of questions 1-13 and 'yes' to question 14, an application must be submitted to the appropriate NHS research ethics committee.

Signed:	Mark Green	Principal Investigator /Supervisor
Signed:	Taylor Lee	Student Investigator
Date:	27/09/2019	

Source Code & Testing

Normalise.py code

```
#This python file normalised the data frm https://www.hotslogs.com/Info/API into one file to be trained on a neural network
import pandas as pd
import os
import re
import numpy as np
import csv
from tqdm import tqdm


#Loads data from the HeroIDandMapID.csv
heroandmapfile = os.path.join("data", "HeroIDAndMapID.csv")
heroesandmaps = pd.read_csv(heroandmapfile, index_col = 0)
#Loads the heroes into a pandas dataframe then gets lists for the subgroup and group as there are 84 heroes
heroes = heroesandmaps.loc[1:84]
```

```
herogroups = list(heroes.groupby('Group').groups.keys())
herosubgroups = list(heroes.groupby('SubGroup').groups.keys())
#Loads the maps into a panda dataframe as their IDs start at 1001 and end at 1022
maps = heroesandmaps.loc[1001:1022]
#removes group and subgroup as they aren't part of map
maps = maps.drop(columns = ['Group', 'SubGroup'])

#Loads data from the Replays.csv
replaysfile = os.path.join("data", "Replays.csv")
replays = pd.read_csv(replaysfile)
#Allows us to make the index done by replayID making it far easier to search for
#rather than just doing it in by a sequintial index
replays.index = replays["ReplayID"]
#Removes the game length and timestamp columns
replays = replays.drop(columns = ['Replay Length', 'Timestamp (UTC)'])
# gives new headings to the columns as gamemode is listed as:
# GameMode(3=Quick Match 4=Hero League 5=Team League 6=Unranked Draft) in the
.csv
replays.columns = ["replay_id", "gamemode", "map_id"]

#loads the ReplayCharacters.csv
replaycharactersfile = os.path.join("data", "ReplayCharacters.csv")
replaycharacters = pd.read_csv(replaycharactersfile, header=0)
#Gets a list of all the unique replayIDs
replayIDs = list(replaycharacters['ReplayID'].unique())
#Groups the players from replaycharacters into matches
matches = replaycharacters.groupby(['ReplayID', 'Is Winner'])

#List of the game modes to help make the labels
modes = ('Quick Match','Hero League','Team League', 'Unranked Draft')

#Makes the label headings for the modes and maps
modelabels = ["mode_" + modes[i] for i in range (0,len(modes))]
maplabels = ["map_" + maps.iloc[i].Name for i in range (0, len(maps.index))]

#Function to make label headings for a team 'team' is either the 'a' or 'b'
def teamlabels(team):
    labels = list()
    #Makes labels for players 1 to 5
    for i in range(1,6):
        labels.append(team+ "_playermmr_" + str(i))
        labels.append(team+ "_playerlevel_" + str(i))
    #Makes labels for all 84 heroes for that team
    for i in range(1,len(heroes) + 1):
        labels.append(team + "_herommr_" + heroes.loc[i].Name)
        labels.append(team + "_herolevel_" + heroes.loc[i].Name)
    #Makes labels for all the subgroups
    for subgroup in herosubgroups:
        labels.append(team + "_subgroup_" + subgroup)
    return labels

#Creates the full list of labels to describe a match and filters characters that may tamper
with reading
#the labels such as spaces and full stops
```

```
matchlabels = ["team_a_won"]+ modelabels + maplabels + teamlabels("a") +
teamlabels("b")
list(map(lambda x: re.sub("[^a-zA-Z0-9_]","", x) , matchlabels))

#Gets the team, the match array and the index the team starts at
def teamdetails(team, match, startindex):
    #Checks there's 5 members on the team if not raising an valueError and passes over
the game
    if len(team) != 5:
        raise ValueError
    #Checks the heroID, mmr and level all fall in criteria such as the mmr being between
400 and 4500
    for i, (heroID, mmr, level) in enumerate((team[["HeroID", "MMR Before", "Hero
Level"]].values)):
        if (heroID < 1 or heroID > 84 or  mmr < 400 or mmr > 4500 or level < 1 or mmr !=
mmr or level != level or heroID != heroID):
            raise ValueError
        #Assigns two columns for each player and hero (for their MMR and Level)
        playercolumns = i * 2
        herocolumns = 10 + int(heroID - 1) * 2
        #places the mmr and level into weights between 0 and 1
        weightedmmr = mmr/4500
        weightedlevel = (level + 5) / 25
        #Places the weightedmmr and weightedlevel in their respective columns
        match[playercolumns+startindex] = weightedmmr
        match[playercolumns+startindex + 1] = weightedlevel
        match[herocolumns+startindex] = weightedmmr
        match[herocolumns+startindex + 1] = weightedlevel
        #Adds the weight of the given hero's subgroup to the respective subgroup column
        subgroupcolumns = 10 + 84 * 2 +
herosubgroups.index(heroes.loc[heroID].SubGroup)
        match[subgroupcolumns+startindex] += .33

def matchdetails(ID):
    #Creates list to store the detail of a match, is the length of the matchlabels
    match = [0.0] * len(matchlabels)

    #Gets the replay for the match and gets the winning and losing team for the match
    replay = replays.loc[ID]
    winningteam = matches.get_group((ID, 1))
    losingteam = matches.get_group((ID, 0))

    #Gets the gamemode and map for the match and gives it a value of one (one hot
encoding)
    match[replay.gamemode-3 + 1] = 1
    match[replay.map_id-1001 + 5] = 1

    #gets length of teamlabels
    teamlableslen = len(teamlabels("a"))
    #Randomly assign a match to either team 'a' or 'b' and places them in the match list
    if np.random.randint(0, 2) == 0:
        #Winning team comes first in the .csv
        match[0] == 1
        teamdetails(winningteam, match, 5 + len(maplabels))
        teamdetails(losingteam, match, 5 + len(maplabels) + teamlableslen)
        return match
    else:
        #Losing team comes first in the .csv
        teamdetails(losingteam, match, 5 + len(maplabels))
        teamdetails(winningteam, match, 5 + len(maplabels) + teamlableslen)
        return match

#Writes the data from the .csv files to a normalised .csv
with (open(os.path.join('data', 'training_data.csv'),'w')) as file:
    writer=csv.writer(file)
    #Writes the first row as the matchlabels headings
    writer.writerow(matchlabels)
    #Goes through each replayID though pass if it doesn't meet validation such as
containing a player with less than
    #400 mmr in game
    #Regards to tqdm for allowing the progress bar https://github.com/tqdm/tqdm
    for replayID in tqdm(replayIDs, unit="matches", mininterval=1):
    #
        try:
            writer.writerow(matchdetails(replayID))
        except ValueError:
            pass

    print("Data has been normalised")
```

neuralnet.py code https://keras.io/ was very helpful in creating this class

```
#Python code to create the neural network and get trained models
import pandas as pd
import os
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.model_selection import ShuffleSplit
from keras.models import Sequential
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras.layers import Dropout, Dense, PReLU
from keras.optimizers import Adam


#Reads the normalised .csv into a panda dataframe
trainingdata = pd.read_csv(os.path.join('data', 'training_data.csv'), dtype=np.float32)

#for loop that goes through the columns in the dataset and places
#columns that aren't used (have a total sum of 0) in a list
unplayedmapsheroes = list()
for c in list(trainingdata.columns):
    if np.sum(trainingdata[c]) == 0:
        unplayedmapsheroes.append(c)
print(unplayedmapsheroes)
#removes the unplayed columns from trainingdata e.g maps like pullparty
trainingdata = trainingdata.drop(unplayedmapsheroes, 1)

#splits the data into holdoutdata and trainingdata the holdout data contains
#100000 matches
trainingdata, holdoutdata = train_test_split(trainingdata, test_size=100000)
```

```
#indexs holdoutdata and trainingdata by sequential indexes instead of ID
holdoutdata.index = np.arange(0, holdoutdata.shape[0])
trainingdata.index = np.arange(0, trainingdata.shape[0])

#Function to split the data into the outcomes (if team a wins or not)
#and features like heroes picked and the match mode
outcomes = trainingdata[trainingdata.columns[0]].astype(bool)
features = trainingdata.drop([trainingdata.columns[0]],1)

#The layers for the neural network takes in 1000 neurons, a .5 dropout layer, an adam
learn rate of .00005 and a beta_1 of .7
def networklayers(n=1000, d=.5, lr=.00005, beta_1=.7):
    #Sets up a sequential model which is the main neural network model for keras
    model = Sequential()
    #The input layer for the data takes in the 392 features from the dataset and places them
across the set neurons
    model.add(Dense(n, input_dim = 392, activation='linear'))
    #Adds a PReLU layer to the model to help with accuracy https://github.com/keras-
team/keras/blob/master/keras/layers/advanced_activations.py#L59
    model.add(PReLU())
    #Adds a dropout layer with .5 dropout to the model to prevent overfitting
https://github.com/keras-team/keras/blob/master/keras/layers/core.py#L81
    model.add(Dropout(d))
    #Adds the output layer to the model giving a weight between 0 and 1 for the likelyhood
team 'a' is going to win
    model.add(Dense(1, activation='sigmoid'))
    #Compiles the model and uses the adam optimiser to help with accuracy and using
binary_crossentropy because there’s two outcomes 0 and 1 (if team a wins or losses the
match)
    model.compile(loss='binary_crossentropy', optimizer=Adam(lr=lr, beta_1=beta_1),
metrics=['accuracy'])
    return model

#Arrays to stored the mean accuracy as losses of the models and
#an array to store the models themselves
accs = []
losses = []
models = []
#Runs the neural network for 6 splits with 100000 test size to generate
#6 different models
ss = ShuffleSplit(n_splits=6, test_size=100000)
for i, (train, test) in enumerate(ss.split(features, outcomes)):
    #prints heading for the split
    print('{} - Split {}/{}'.format("model", i+1, 6))
    #gets a location to store the .hdf5 of the model
    #best way to store the models as they're meant for large data
    modelname = os.path.join('models', "{}-{}.hdf5".format("model", i+1))
    #sets the model up using the networklayers class
    model = networklayers()
    #Set up the features of the model and the outcomes of the model as well as the
validation data sets up epochs as well and the 2000 batch size for adam also sets up
callbacks such as modelcheckpoint(writing to a file the best version of the current model)
and stopping early if the model doesn't improve (earlystopping)
    #keras callbacks was espcially helpful in this https://github.com/keras-
team/keras/blob/master/keras/callbacks.py#L275
```

```
    mademodel = model.fit(features.loc[train].values, outcomes.loc[train],
validation_data=(features.loc[test].values, outcomes.loc[test]), epochs=100,
batch_size=2000, callbacks=[ModelCheckpoint(filepath=filename, verbose=0,
save_best_only=True),EarlyStopping(monitor='val_loss', min_delta=.000002, patience=4,
verbose=0)], verbose=0)
    #Adds current model the the models array
    models.append(mademodel)
    #Loads the model into a hdf5 file
    model.load_weights(modelname)
    #Gives percentages and scores for the accuracy of the model
    score = model.evaluate(features.loc[test].values, outcomes[test], verbose=0)
    losses.append(score[0])
    accs.append(100*score[1])
    print("Test loss: {:.5f}; acc: {:.3f}%".format(score[0], 100*score[1]))
print("Model accuracy: {:.2f}% +/- {:.2f}%".format(np.mean(accs), np.std(accs)))
print("Model test_loss: {:.5f} +/- {:.5f}".format(np.mean(losses), np.std(losses)))
```

output.py

```
#produces the graphical outputs used in the report
import pandas as pd
import os
import numpy as np
import re
import matplotlib.pyplot as plt
import matplotlib.font_manager as fm
import neuralnet as nn

#Loads in final.csv the data with the 4 matches used to explain the matchmaking system
validationdata = pd.read_csv(os.path.join('final.csv'), dtype=np.float32)

#Splits the validation data up
outcomes = validationdata[validationdata.columns[0]].astype(bool)
features = validationdata.drop([validationdata.columns[0]],1)

#Loads the model that you want
model = nn.networklayers()
model.load_weights(os.path.join('models', 'model-1.hdf5'))

#Loads the heroIDandMapID.csv and makes lists for each of the heroes names,groups
and subgroups
heromap = pd.read_csv(os.path.join("data", "HeroIDAndMapID.csv"), index_col = 0)
heroes = heromap.loc[1:84]
heronames = list(heroes.groupby('Name', sort=False).groups.keys())
herogroups = list(heroes.groupby('Group').groups.keys())
herosubgroups = list(heroes.groupby('SubGroup').groups.keys())

#Using lambda filters the data into lists for herolevels,herommrs,modes and maps
herolevels = list(filter(lambda c: re.match("^[ab]_herolevel",c), validationdata.columns))
herommrs = list(filter(lambda c: re.match("^[ab]_herommr",c), validationdata.columns))
modes = list(filter(lambda c:
re.match("^mode_[QuickMatchHeroLeagueTeamLeagueUnrankedDraft]",c),
validation_data.columns))
maps = list(filter(lambda c:
re.match("^map_[BattlefieldofEternityBlackheartsBayCursedHollowDragonShireGardenof
TerrorInfernalShrinesSkyTempleTomboftheSpiderQueenTowersofDoomBraxisHoldoutWar
headJunctionHanamuraTempleVolskayaFoundryAlteracPass]",c),
validation_data.columns))


#Function to prepare data for a game to be printed in the print_game function
def game_data(row):
    #Dictionaries for the two teams and lists for their heroIDs
    teama = {}
    teamb = {}
    teamaID = []
    teambID = []
    #Finds the mode the match is being played and makes sure the 'mode' string
    #is well formatted
    for m in modes:
        if row[m] != 0:
            mode = m.replace("mode_","")
            mode = re.sub(r"(\w)([A-Z])", r"\1 \2", mode)

    #Finds the map the match is taking place in and makes sure the 'map' string
    #is well formatted
    for m in maps:
        if row[m] != 0:
            map = m.replace("map_","")
            map = re.sub(r"(\w)([A-Z])", r"\1 \2", map)
            map = map.replace("of", " of")
            map = map.replace("the", " the")

    #Gets the herolevel for each hero from the dataset and converts it from a weight to
    #an actually level. Also the IDs of the heroes are added to 'teamaID' and 'teambID'
    #lists to be used later to give well formatted hero strings because when the data was
    #normalised special characters like '-' were lost on certain heroes
    for h in herolevels:
        if row[h] != 0:
            hero = h.split('_')[2]
            level =int(round(row[h] * 25) - 5)
            if h[0] is 'a':
                teamaID.append(herolevels.index(h))
                teama.setdefault(hero, {})['level'] = level
            else:
                teambID.append(herolevels.index(h)-84)
                teamb.setdefault(hero, {})['level'] = level

    #Similar to the above for loop but this time for mmrs and doesn't add hero IDs.
    #It converts the weight back into an actual MMR value
    for h in herommrs:
        if row[h] != 0:
            hero = h.split('_')[2]
            mmr = int(round(row[h] * 4500))
            if h[0] is 'a':
                teama.setdefault(hero, {})['mmr'] = mmr
            else:
                teamb.setdefault(hero, {})['mmr'] = mmr

    return (teama, teamb, teamaID, teambID, mode, map)
```

```
#Recieves data from the game_data function and the model to be placed into a graphic
def print_game(teama, teamb, teamaID, teambID, mode, map, teama_won, prediction):
    #Sets up a background image which is a heroes of the storm themed background
    img = plt.imread(os.path.join('background.jpg'))
    fig = plt.figure(figsize=(12, 7))
    plt.imshow(img)
    plt.axis('off')
    #Sets the default text colour to white
    plt.rcParams['text.color'] = 'white'
    #Reads in the Exo font which is similar to font used in the actual game
    prop = fm.FontProperties(fname= 'C:\Windows\Fonts\Exo-Regular.ttf')
    #Makes a header for team A, states their chance of winning
    ax = fig.add_subplot(6, 3, 1)
    ax.axis('off', label=str(1))
    ax.text(.2,.6,"Team A", fontproperties=prop, size=16)
    ax.text(.55,.6,'{:.1f}% to win'.format(prediction*100), fontproperties=prop, size=16)

    #Makes a header for team B, states their chance of winning
    ax2 = fig.add_subplot(6, 3, 3)
    ax2.axis('off', label=str(3))
    ax2.text(-.1,.6,"Team B", fontproperties=prop, size=16)
    ax2.text(.25,.6,'{:.1f}% to win'.format((1-prediction)*100), fontproperties=prop, size=16)

    #Adds a VS. symbol between the two teams
    # and lists the Mode and Map the match takes place in
    ax3 = fig.add_subplot(6, 3, 3*2+5)
    ax3.axis('off', label=str(3*2+5))
    ax3.text(.4,.5,"VS.", fontproperties=prop, size=20)
    ax3.text(0.1,5,'Mode: '+mode, fontproperties=prop, size=16)
    ax3.text(0.1,4.5,'Map: '+map, fontproperties=prop, size=16)

    #Values to store total MMR and level for each team
    LevelTotala = 0
    LevelTotalb = 0
    MMRTotala = 0
    MMRTotalb = 0
    #Loops 5 times for the two teams doing a member for each in sequence
    for i, (a,b) in enumerate(zip(teama, teamb)):
        #Adds the level and MMR from the player from team a and from team b to the total
        LevelTotala = LevelTotala + teama[a]['level']
        LevelTotalb = LevelTotalb + teamb[b]['level']
        MMRTotala = MMRTotala + teama[a]['mmr']
        MMRTotalb = MMRTotalb + teamb[b]['mmr']
        #Lists the name,level,mmr,role,subrole and image of a hero on team a
        ax4 = fig.add_subplot(6, 3, i*3+4)
        ax4.axis('off', label=str(i*3+4))
        plt.title(heronames[heronames.index(heroes.loc[teamaID[i] + 1].Name)],
fontproperties=prop, size=16)
        img = plt.imread(os.path.join('portraits', '{}.png'.format(a)))
        ax4.imshow(img)
        ax4.text(100,10, "Level: {}".format(teama[a]['level']))
        ax4.text(100,30, "MMR: {}".format(teama[a]['mmr']))
        ax4.text(100,50, "Role: " + herogroups[herogroups.index(heroes.loc[teamaID[i] +
1].Group)])
        ax4.text(100,70, "Subrole: " +
herosubgroups[herosubgroups.index(heroes.loc[teamaID[i]+ 1].SubGroup)])
        #Lists the name,level,mmr,role,subrole and image of a hero on team b
        ax5 = fig.add_subplot(6, 3, i*3+6)
        ax5.axis('off', label=str(i*3+6))
        plt.title(heronames[heronames.index(heroes.loc[teambID[i] + 1].Name)],
fontproperties=prop, size=16)
        img = plt.imread(os.path.join('portraits', '{}.png'.format(b)))
        ax5.imshow(img)
        ax5.text(-240,10, "Level: {}".format(teamb[b]['level']))
        ax5.text(-240,30, "MMR: {}".format(teamb[b]['mmr']))
        ax5.text(-240,50, "Role: " + herogroups[herogroups.index(heroes.loc[teambID[i] +
1].Group)])
        ax5.text(-240,70, "Subrole: " +
herosubgroups[herosubgroups.index(heroes.loc[teambID[i]+ 1].SubGroup)])
    #calculates the average level and average MMR for a team
    ax.text(.55,.35, "Avg. Lvl: " + str(LevelTotala/5), size=10)
    ax.text(.55,.05, "Avg. MMR: "  +  str(MMRTotala/5), size=10)
    ax2.text(.25,.35, "Avg. Lvl: " + str(LevelTotalb/5), size=10)
    ax2.text(.25,.05, "Avg. MMR: " + str(MMRTotalb/5), size=10)
    #Adjusts the subplots so the graphic fits nicely
    plt.subplots_adjust(hspace=.5)
    plt.show()

#returns 4 matchups from the final.csv (the matchmaking one)
val_features, val_winner = split_features(validation_data.head(4))
#Loops for each of the matches
for i, row in val_features.iterrows():
    if i > 0:
        print("\n\n")
    #produces the data for the match
    teama, teamb, teamaID, teambID, mode, map = game_data(row)
    #produces the prediction for the match by the model
    prediction = model.predict(row.values.reshape(1,392))[0][0]
    #prints the graphic for the game
    print_game(teama, teamb, teamaID, teambID, mode, map, val_winner[i], prediction)
```

Initial model test with 59.609% accuracy (NOTE: Only every 3 epochs are shown after the first epoch, saves on space as some models reach 100 epochs)

```
initialmodel
Epoch 1(31s) - train_loss: 0.67566; train_acc: 57.435%; val_loss: 0.6
7052; val_acc: 58.523%
Epoch 4(34s) - train_loss: 0.66580; train_acc: 59.381%; val_loss: 0.6
6503; val_acc: 59.260%
Epoch 7(34s) - train_loss: 0.66407; train_acc: 59.587%; val_loss: 0.6
6366; val_acc: 59.430%
Epoch 10(36s) - train_loss: 0.66355; train_acc: 59.613%; val_loss: 0.
66309; val_acc: 59.649%
Epoch 13(32s) - train_loss: 0.66341; train_acc: 59.697%; val_loss: 0.
66307; val_acc: 59.695%
Epoch 16(37s) - train_loss: 0.66325; train_acc: 59.714%; val_loss: 0.
66314; val_acc: 59.575%
```

```
Epoch 19(38s) - train_loss: 0.66324; train_acc: 59.731%; val_loss: 0.
66285; val_acc: 59.609%
Epoch 22(37s) - train_loss: 0.66326; train_acc: 59.694%; val_loss: 0.
66311; val_acc: 59.626%
Training round took 13:30
Test loss: 0.66285; acc: 59.609%
Model accuracy: 59.61% +/- 0.00%
Model test_loss: 0.66285 +/- 0.00000
```

Initial model test with splits notice 59.881% in bold (split 2)

```
Split 1/6
Epoch 1(15s) - train_loss: 0.67444; train_acc: 57.718%; val_loss: 0.6
6892; val_acc: 58.940%
Epoch 4(15s) - train_loss: 0.66566; train_acc: 59.362%; val_loss: 0.6
6425; val_acc: 59.618%
Epoch 7(14s) - train_loss: 0.66405; train_acc: 59.600%; val_loss: 0.6
6376; val_acc: 59.629%
Epoch 10(14s) - train_loss: 0.66367; train_acc: 59.669%; val_loss: 0.
66283; val_acc: 59.760%
Epoch 13(14s) - train_loss: 0.66348; train_acc: 59.613%; val_loss: 0.
66285; val_acc: 59.786%
Training round took 3:37
Test loss: 0.66259; acc: 59.774%
Split 2/6
Epoch 1(14s) - train_loss: 0.67654; train_acc: 57.322%; val_loss: 0.6
7038; val_acc: 58.685%
Epoch 4(14s) - train_loss: 0.66596; train_acc: 59.341%; val_loss: 0.6
6480; val_acc: 59.426%
Epoch 7(14s) - train_loss: 0.66408; train_acc: 59.551%; val_loss: 0.6
6344; val_acc: 59.765%
Epoch 10(14s) - train_loss: 0.66362; train_acc: 59.611%; val_loss: 0.
66308; val_acc: 59.731%
Epoch 13(14s) - train_loss: 0.66349; train_acc: 59.679%; val_loss: 0.
66285; val_acc: 59.937%
Epoch 16(14s) - train_loss: 0.66329; train_acc: 59.664%; val_loss: 0.
66274; val_acc: 59.952%
Epoch 19(14s) - train_loss: 0.66328; train_acc: 59.674%; val_loss: 0.
66294; val_acc: 59.833%
Training round took 4:29
Test loss: 0.66265; acc: 59.881%
Split 3/6
Epoch 1(14s) - train_loss: 0.67492; train_acc: 57.585%; val_loss: 0.6
7048; val_acc: 58.493%
Epoch 4(14s) - train_loss: 0.66566; train_acc: 59.399%; val_loss: 0.6
6531; val_acc: 59.304%
Epoch 7(14s) - train_loss: 0.66397; train_acc: 59.591%; val_loss: 0.6
6390; val_acc: 59.603%
Epoch 10(14s) - train_loss: 0.66349; train_acc: 59.666%; val_loss: 0.
66406; val_acc: 59.533%
Epoch 13(14s) - train_loss: 0.66341; train_acc: 59.679%; val_loss: 0.
66355; val_acc: 59.640%
Epoch 16(14s) - train_loss: 0.66323; train_acc: 59.708%; val_loss: 0.
66338; val_acc: 59.625%
Epoch 19(14s) - train_loss: 0.66313; train_acc: 59.709%; val_loss: 0.
66341; val_acc: 59.663%
```

```
Training round took 4:56
Test loss: 0.66331; acc: 59.704%
Split 4/6
Epoch 1(14s) - train_loss: 0.67453; train_acc: 57.701%; val_loss: 0.6
7049; val_acc: 58.503%
Epoch 4(14s) - train_loss: 0.66547; train_acc: 59.364%; val_loss: 0.6
6600; val_acc: 59.374%
Epoch 7(14s) - train_loss: 0.66395; train_acc: 59.545%; val_loss: 0.6
6494; val_acc: 59.249%
Epoch 10(14s) - train_loss: 0.66346; train_acc: 59.672%; val_loss: 0.
66424; val_acc: 59.523%
Epoch 13(14s) - train_loss: 0.66332; train_acc: 59.718%; val_loss: 0.
66414; val_acc: 59.464%
Epoch 16(14s) - train_loss: 0.66312; train_acc: 59.725%; val_loss: 0.
66462; val_acc: 59.441%
Epoch 19(14s) - train_loss: 0.66307; train_acc: 59.706%; val_loss: 0.
66457; val_acc: 59.599%
Training round took 4:28
Test loss: 0.66413; acc: 59.506%
Split 5/6
Epoch 1(14s) - train_loss: 0.67469; train_acc: 57.706%; val_loss: 0.6
7029; val_acc: 58.707%
Epoch 4(14s) - train_loss: 0.66554; train_acc: 59.353%; val_loss: 0.6
6560; val_acc: 59.356%
Epoch 7(14s) - train_loss: 0.66389; train_acc: 59.608%; val_loss: 0.6
6462; val_acc: 59.621%
Epoch 10(14s) - train_loss: 0.66341; train_acc: 59.692%; val_loss: 0.
66393; val_acc: 59.749%
Epoch 13(14s) - train_loss: 0.66328; train_acc: 59.709%; val_loss: 0.
66409; val_acc: 59.677%
Training round took 3:31
Test loss: 0.66386; acc: 59.744%
Split 6/6
Epoch 1(14s) - train_loss: 0.67602; train_acc: 57.408%; val_loss: 0.6
7059; val_acc: 58.615%
Epoch 4(14s) - train_loss: 0.66589; train_acc: 59.302%; val_loss: 0.6
6522; val_acc: 59.453%
Epoch 7(14s) - train_loss: 0.66404; train_acc: 59.590%; val_loss: 0.6
6380; val_acc: 59.773%
Epoch 10(14s) - train_loss: 0.66355; train_acc: 59.646%; val_loss: 0.
66349; val_acc: 59.766%
Epoch 13(14s) - train_loss: 0.66341; train_acc: 59.687%; val_loss: 0.
66374; val_acc: 59.691%
Epoch 16(14s) - train_loss: 0.66330; train_acc: 59.669%; val_loss: 0.
66316; val_acc: 59.777%
Epoch 19(14s) - train_loss: 0.66327; train_acc: 59.660%; val_loss: 0.
66315; val_acc: 59.831%
Epoch 22(14s) - train_loss: 0.66320; train_acc: 59.678%; val_loss: 0.
66338; val_acc: 59.766%
Epoch 25(14s) - train_loss: 0.66318; train_acc: 59.692%; val_loss: 0.
66341; val_acc: 59.699%
Training round took 5:51
Test loss: 0.66314; acc: 59.738%
Model accuracy: 59.72% +/- 0.11%
Model test_loss: 0.66328 +/- 0.00057
```

Initial test for final neural network

```
Epoch 1(39s) - train_loss: 0.68366; train_acc: 55.394%; val_loss:
0.67392; val_acc: 58.201%
Epoch 4(37s) - train_loss: 0.67051; train_acc: 58.555%; val_loss:
0.66824; val_acc: 59.019%
Epoch 7(37s) - train_loss: 0.66674; train_acc: 59.206%; val_loss:
0.66535; val_acc: 59.486%
Epoch 10(37s) - train_loss: 0.66353; train_acc: 59.748%; val_loss:
0.66276; val_acc: 59.976%
Epoch 13(37s) - train_loss: 0.66019; train_acc: 60.300%; val_loss:
0.66044; val_acc: 60.277%
Epoch 16(37s) - train_loss: 0.65742; train_acc: 60.715%; val_loss:
0.65839; val_acc: 60.576%
Epoch 19(37s) - train_loss: 0.65495; train_acc: 61.103%; val_loss:
0.65677; val_acc: 60.846%
Epoch 22(37s) - train_loss: 0.65248; train_acc: 61.433%; val_loss:
0.65551; val_acc: 60.911%
Epoch 25(37s) - train_loss: 0.65068; train_acc: 61.658%; val_loss:
0.65444; val_acc: 61.173%
Epoch 28(37s) - train_loss: 0.64894; train_acc: 61.882%; val_loss:
0.65372; val_acc: 61.197%
Epoch 31(37s) - train_loss: 0.64735; train_acc: 62.146%; val_loss:
0.65298; val_acc: 61.264%
Epoch 34(37s) - train_loss: 0.64594; train_acc: 62.339%; val_loss:
0.65256; val_acc: 61.323%
Epoch 37(37s) - train_loss: 0.64463; train_acc: 62.481%; val_loss:
0.65230; val_acc: 61.443%
Epoch 40(37s) - train_loss: 0.64342; train_acc: 62.626%; val_loss:
0.65198; val_acc: 61.467%
Epoch 43(37s) - train_loss: 0.64249; train_acc: 62.762%; val_loss:
0.65194; val_acc: 61.520%
Epoch 46(37s) - train_loss: 0.64121; train_acc: 62.941%; val_loss:
0.65175; val_acc: 61.538%
Epoch 49(37s) - train_loss: 0.64022; train_acc: 62.998%; val_loss:
0.65156; val_acc: 61.620%
Epoch 52(37s) - train_loss: 0.63945; train_acc: 63.164%; val_loss:
0.65156; val_acc: 61.644%
Epoch 55(37s) - train_loss: 0.63838; train_acc: 63.241%; val_loss:
0.65153; val_acc: 61.648%
Epoch 58(37s) - train_loss: 0.63774; train_acc: 63.315%; val_loss:
0.65143; val_acc: 61.667%
Epoch 61(37s) - train_loss: 0.63675; train_acc: 63.484%; val_loss:
0.65149; val_acc: 61.701%
Epoch 64(37s) - train_loss: 0.63646; train_acc: 63.495%; val_loss:
0.65158; val_acc: 61.707%
Training round took 39:47
Test loss: 0.65141; acc: 61.675%
Model accuracy: 61.68% +/- 0.00%
Model test_loss: 0.65141 +/- 0.00000
```

Final neural network final testing best split in bold (split 4). This split was used for the final model

```
Split 1/6
Epoch 1(38s) - train_loss: 0.68311; train_acc: 55.541%; val_loss: 0.6
7401; val_acc: 58.054%
Epoch 4(38s) - train_loss: 0.67049; train_acc: 58.549%; val_loss: 0.6
6886; val_acc: 58.700%
Epoch 7(37s) - train_loss: 0.66657; train_acc: 59.292%; val_loss: 0.6
6591; val_acc: 59.267%
Epoch 10(37s) - train_loss: 0.66321; train_acc: 59.807%; val_loss: 0.
66337; val_acc: 59.670%
Epoch 13(38s) - train_loss: 0.66028; train_acc: 60.273%; val_loss: 0.
66125; val_acc: 59.978%
Epoch 16(38s) - train_loss: 0.65732; train_acc: 60.774%; val_loss: 0.
65931; val_acc: 60.357%
Epoch 19(37s) - train_loss: 0.65479; train_acc: 61.128%; val_loss: 0.
65788; val_acc: 60.600%
Epoch 22(37s) - train_loss: 0.65245; train_acc: 61.449%; val_loss: 0.
65657; val_acc: 60.751%
Epoch 25(38s) - train_loss: 0.65044; train_acc: 61.710%; val_loss: 0.
65569; val_acc: 60.840%
Epoch 28(37s) - train_loss: 0.64858; train_acc: 62.044%; val_loss: 0.
65497; val_acc: 60.963%
Epoch 31(37s) - train_loss: 0.64735; train_acc: 62.170%; val_loss: 0.
65444; val_acc: 60.989%
Epoch 34(37s) - train_loss: 0.64590; train_acc: 62.306%; val_loss: 0.
65399; val_acc: 61.075%
Epoch 37(37s) - train_loss: 0.64445; train_acc: 62.520%; val_loss: 0.
65364; val_acc: 61.026%
Epoch 40(37s) - train_loss: 0.64311; train_acc: 62.664%; val_loss: 0.
65351; val_acc: 61.120%
Epoch 43(37s) - train_loss: 0.64216; train_acc: 62.759%; val_loss: 0.
65325; val_acc: 61.097%
Epoch 46(38s) - train_loss: 0.64116; train_acc: 62.987%; val_loss: 0.
65322; val_acc: 61.133%
Epoch 49(37s) - train_loss: 0.64002; train_acc: 63.073%; val_loss: 0.
65302; val_acc: 61.168%
Epoch 52(37s) - train_loss: 0.63943; train_acc: 63.139%; val_loss: 0.
65320; val_acc: 61.151%
Test loss: 0.65302; acc: 61.168%
Split 2/6
Epoch 1(37s) - train_loss: 0.68350; train_acc: 55.442%; val_loss: 0.6
7451; val_acc: 57.888%
Epoch 4(38s) - train_loss: 0.67052; train_acc: 58.552%; val_loss: 0.6
6910; val_acc: 58.815%
Epoch 7(38s) - train_loss: 0.66670; train_acc: 59.187%; val_loss: 0.6
6644; val_acc: 59.212%
Epoch 10(37s) - train_loss: 0.66334; train_acc: 59.760%; val_loss: 0.
66415; val_acc: 59.589%
Epoch 13(37s) - train_loss: 0.66020; train_acc: 60.282%; val_loss: 0.
66208; val_acc: 59.951%
Epoch 16(37s) - train_loss: 0.65753; train_acc: 60.706%; val_loss: 0.
66027; val_acc: 60.220%
Epoch 19(37s) - train_loss: 0.65496; train_acc: 61.137%; val_loss: 0.
65869; val_acc: 60.423%
```

```
Epoch 22(37s) - train_loss: 0.65260; train_acc: 61.419%; val_loss: 0.
65758; val_acc: 60.587%
Epoch 25(37s) - train_loss: 0.65062; train_acc: 61.698%; val_loss: 0.
65661; val_acc: 60.742%
Epoch 28(38s) - train_loss: 0.64885; train_acc: 61.985%; val_loss: 0.
65578; val_acc: 60.787%
Epoch 31(37s) - train_loss: 0.64725; train_acc: 62.147%; val_loss: 0.
65533; val_acc: 60.841%
Epoch 34(37s) - train_loss: 0.64589; train_acc: 62.357%; val_loss: 0.
65486; val_acc: 60.854%
Epoch 37(37s) - train_loss: 0.64473; train_acc: 62.486%; val_loss: 0.
65461; val_acc: 60.883%
Epoch 40(37s) - train_loss: 0.64351; train_acc: 62.659%; val_loss: 0.
65427; val_acc: 60.964%
Epoch 43(37s) - train_loss: 0.64215; train_acc: 62.759%; val_loss: 0.
65440; val_acc: 60.953%
Epoch 46(37s) - train_loss: 0.64122; train_acc: 62.962%; val_loss: 0.
65405; val_acc: 61.016%
Epoch 49(38s) - train_loss: 0.64007; train_acc: 63.068%; val_loss: 0.
65412; val_acc: 61.073%
Epoch 52(38s) - train_loss: 0.63928; train_acc: 63.190%; val_loss: 0.
65392; val_acc: 61.090%
Epoch 55(37s) - train_loss: 0.63863; train_acc: 63.238%; val_loss: 0.
65388; val_acc: 61.141%
Epoch 58(37s) - train_loss: 0.63778; train_acc: 63.323%; val_loss: 0.
65387; val_acc: 61.138%
Training round took 36:08
Test loss: 0.65386; acc: 61.108%
Split 3/6
Epoch 1(37s) - train_loss: 0.68386; train_acc: 55.348%; val_loss: 0.6
7434; val_acc: 58.087%
Epoch 4(37s) - train_loss: 0.67076; train_acc: 58.509%; val_loss: 0.6
6862; val_acc: 58.780%
Epoch 7(37s) - train_loss: 0.66682; train_acc: 59.218%; val_loss: 0.6
6565; val_acc: 59.330%
Epoch 10(37s) - train_loss: 0.66357; train_acc: 59.769%; val_loss: 0.
66308; val_acc: 59.823%
Epoch 13(37s) - train_loss: 0.66031; train_acc: 60.299%; val_loss: 0.
66069; val_acc: 60.265%
Epoch 16(37s) - train_loss: 0.65741; train_acc: 60.716%; val_loss: 0.
65857; val_acc: 60.544%
Epoch 19(37s) - train_loss: 0.65503; train_acc: 61.075%; val_loss: 0.
65687; val_acc: 60.838%
Epoch 22(37s) - train_loss: 0.65275; train_acc: 61.367%; val_loss: 0.
65549; val_acc: 61.028%
Epoch 25(37s) - train_loss: 0.65082; train_acc: 61.622%; val_loss: 0.
65450; val_acc: 61.169%
Epoch 28(37s) - train_loss: 0.64888; train_acc: 61.925%; val_loss: 0.
65372; val_acc: 61.304%
Epoch 31(38s) - train_loss: 0.64757; train_acc: 62.142%; val_loss: 0.
65309; val_acc: 61.434%
Epoch 34(37s) - train_loss: 0.64601; train_acc: 62.298%; val_loss: 0.
65248; val_acc: 61.478%
Epoch 37(37s) - train_loss: 0.64472; train_acc: 62.450%; val_loss: 0.
65217; val_acc: 61.479%
Epoch 40(37s) - train_loss: 0.64365; train_acc: 62.605%; val_loss: 0.
65191; val_acc: 61.545%
```

```
Epoch 43(37s) - train_loss: 0.64265; train_acc: 62.777%; val_loss: 0.
65164; val_acc: 61.640%
Epoch 46(37s) - train_loss: 0.64134; train_acc: 62.861%; val_loss: 0.
65140; val_acc: 61.623%
Epoch 49(37s) - train_loss: 0.64054; train_acc: 63.016%; val_loss: 0.
65143; val_acc: 61.579%
Epoch 52(37s) - train_loss: 0.63952; train_acc: 63.138%; val_loss: 0.
65116; val_acc: 61.609%
Epoch 55(37s) - train_loss: 0.63868; train_acc: 63.260%; val_loss: 0.
65127; val_acc: 61.559%
Training round took 34:36
Test loss: 0.65116; acc: 61.609%
Split 4/6
Epoch 1(38s) - train_loss: 0.68373; train_acc: 55.373%; val_loss: 0.6
7413; val_acc: 58.159%
Epoch 4(38s) - train_loss: 0.67081; train_acc: 58.455%; val_loss: 0.6
6846; val_acc: 58.945%
Epoch 7(38s) - train_loss: 0.66689; train_acc: 59.138%; val_loss: 0.6
6543; val_acc: 59.484%
Epoch 10(38s) - train_loss: 0.66356; train_acc: 59.714%; val_loss: 0.
66283; val_acc: 59.966%
Epoch 13(42s) - train_loss: 0.66046; train_acc: 60.216%; val_loss: 0.
66049; val_acc: 60.315%
Epoch 16(38s) - train_loss: 0.65778; train_acc: 60.610%; val_loss: 0.
65853; val_acc: 60.561%
Epoch 19(40s) - train_loss: 0.65512; train_acc: 61.044%; val_loss: 0.
65684; val_acc: 60.892%
Epoch 22(41s) - train_loss: 0.65275; train_acc: 61.402%; val_loss: 0.
65533; val_acc: 61.060%
Epoch 25(39s) - train_loss: 0.65067; train_acc: 61.682%; val_loss: 0.
65422; val_acc: 61.284%
Epoch 28(37s) - train_loss: 0.64919; train_acc: 61.897%; val_loss: 0.
65342; val_acc: 61.361%
Epoch 31(37s) - train_loss: 0.64753; train_acc: 62.045%; val_loss: 0.
65278; val_acc: 61.498%
Epoch 34(37s) - train_loss: 0.64600; train_acc: 62.286%; val_loss: 0.
65224; val_acc: 61.602%
Epoch 37(37s) - train_loss: 0.64474; train_acc: 62.432%; val_loss: 0.
65191; val_acc: 61.611%
Epoch 40(37s) - train_loss: 0.64341; train_acc: 62.623%; val_loss: 0.
65156; val_acc: 61.671%
Epoch 43(37s) - train_loss: 0.64252; train_acc: 62.757%; val_loss: 0.
65129; val_acc: 61.706%
Epoch 46(37s) - train_loss: 0.64132; train_acc: 62.898%; val_loss: 0.
65118; val_acc: 61.658%
Epoch 49(37s) - train_loss: 0.64048; train_acc: 62.984%; val_loss: 0.
65101; val_acc: 61.697%
Epoch 52(42s) - train_loss: 0.63960; train_acc: 63.120%; val_loss: 0.
65097; val_acc: 61.697%
Training round took 34:21
Test loss: 0.65093; acc: 61.738%
Split 5/6
Epoch 1(38s) - train_loss: 0.68415; train_acc: 55.241%; val_loss: 0.6
7491; val_acc: 58.000%
Epoch 4(40s) - train_loss: 0.67057; train_acc: 58.505%; val_loss: 0.6
6911; val_acc: 58.790%
Epoch 7(44s) - train_loss: 0.66667; train_acc: 59.186%; val_loss: 0.6
6616; val_acc: 59.324%
```

```
Epoch 10(43s) - train_loss: 0.66351; train_acc: 59.676%; val_loss: 0.
66356; val_acc: 59.781%
Epoch 13(39s) - train_loss: 0.66037; train_acc: 60.217%; val_loss: 0.
66133; val_acc: 60.205%
Epoch 16(41s) - train_loss: 0.65754; train_acc: 60.690%; val_loss: 0.
65923; val_acc: 60.618%
Epoch 19(40s) - train_loss: 0.65492; train_acc: 61.042%; val_loss: 0.
65755; val_acc: 60.767%
Epoch 22(40s) - train_loss: 0.65285; train_acc: 61.383%; val_loss: 0.
65622; val_acc: 61.018%
Epoch 25(38s) - train_loss: 0.65067; train_acc: 61.712%; val_loss: 0.
65513; val_acc: 61.141%
Epoch 28(41s) - train_loss: 0.64883; train_acc: 61.956%; val_loss: 0.
65431; val_acc: 61.233%
Epoch 31(39s) - train_loss: 0.64732; train_acc: 62.139%; val_loss: 0.
65382; val_acc: 61.275%
Epoch 34(42s) - train_loss: 0.64589; train_acc: 62.334%; val_loss: 0.
65331; val_acc: 61.365%
Epoch 37(38s) - train_loss: 0.64468; train_acc: 62.488%; val_loss: 0.
65294; val_acc: 61.380%
Epoch 40(37s) - train_loss: 0.64343; train_acc: 62.597%; val_loss: 0.
65256; val_acc: 61.402%
Epoch 43(37s) - train_loss: 0.64243; train_acc: 62.767%; val_loss: 0.
65236; val_acc: 61.485%
Epoch 46(37s) - train_loss: 0.64147; train_acc: 62.937%; val_loss: 0.
65223; val_acc: 61.432%
Epoch 49(37s) - train_loss: 0.64027; train_acc: 63.056%; val_loss: 0.
65204; val_acc: 61.447%
Epoch 52(37s) - train_loss: 0.63938; train_acc: 63.214%; val_loss: 0.
65205; val_acc: 61.438%
Epoch 55(37s) - train_loss: 0.63839; train_acc: 63.258%; val_loss: 0.
65198; val_acc: 61.467%
Epoch 58(37s) - train_loss: 0.63758; train_acc: 63.359%; val_loss: 0.
65193; val_acc: 61.430%
Epoch 61(37s) - train_loss: 0.63663; train_acc: 63.480%; val_loss: 0.
65196; val_acc: 61.454%
Training round took 40:20
Test loss: 0.65193; acc: 61.430%
Split 6/6
Epoch 1(38s) - train_loss: 0.68385; train_acc: 55.406%; val_loss: 0.6
7393; val_acc: 58.392%
Epoch 4(37s) - train_loss: 0.67070; train_acc: 58.503%; val_loss: 0.6
6834; val_acc: 59.183%
Epoch 7(37s) - train_loss: 0.66680; train_acc: 59.181%; val_loss: 0.6
6571; val_acc: 59.528%
Epoch 10(37s) - train_loss: 0.66340; train_acc: 59.800%; val_loss: 0.
66329; val_acc: 59.935%
Epoch 13(37s) - train_loss: 0.66048; train_acc: 60.235%; val_loss: 0.
66107; val_acc: 60.235%
Epoch 16(37s) - train_loss: 0.65762; train_acc: 60.600%; val_loss: 0.
65914; val_acc: 60.512%
Epoch 19(37s) - train_loss: 0.65487; train_acc: 61.112%; val_loss: 0.
65743; val_acc: 60.797%
Epoch 22(37s) - train_loss: 0.65274; train_acc: 61.441%; val_loss: 0.
65623; val_acc: 60.919%
Epoch 25(37s) - train_loss: 0.65063; train_acc: 61.662%; val_loss: 0.
65513; val_acc: 61.066%
```

```
Epoch 28(37s) - train_loss: 0.64907; train_acc: 61.915%; val_loss: 0.
65439; val_acc: 61.073%
Epoch 31(37s) - train_loss: 0.64729; train_acc: 62.115%; val_loss: 0.
65369; val_acc: 61.172%
Epoch 34(37s) - train_loss: 0.64590; train_acc: 62.343%; val_loss: 0.
65327; val_acc: 61.166%
Epoch 37(37s) - train_loss: 0.64452; train_acc: 62.538%; val_loss: 0.
65295; val_acc: 61.170%
Epoch 40(37s) - train_loss: 0.64329; train_acc: 62.647%; val_loss: 0.
65273; val_acc: 61.276%
Epoch 43(37s) - train_loss: 0.64219; train_acc: 62.790%; val_loss: 0.
65282; val_acc: 61.259%
Epoch 46(38s) - train_loss: 0.64131; train_acc: 62.904%; val_loss: 0.
65241; val_acc: 61.215%
Epoch 49(37s) - train_loss: 0.64015; train_acc: 63.045%; val_loss: 0.
65228; val_acc: 61.272%
Epoch 52(37s) - train_loss: 0.63953; train_acc: 63.119%; val_loss: 0.
65229; val_acc: 61.265%
Epoch 55(37s) - train_loss: 0.63844; train_acc: 63.304%; val_loss: 0.
65226; val_acc: 61.234%
Training round took 34:16
Test loss: 0.65226; acc: 61.234%
Model accuracy: 61.45% +/- 0.22%
Model test_loss: 0.65200 +/- 0.00095
```

.9 adam instead of the set .7

```
.9 adam
Epoch 1(38s) - train_loss: 0.68431; train_acc: 55.184%; val_loss: 0.6
7443; val_acc: 58.148%
Epoch 4(37s) - train_loss: 0.67091; train_acc: 58.455%; val_loss: 0.6
6857; val_acc: 58.916%
Epoch 7(37s) - train_loss: 0.66678; train_acc: 59.223%; val_loss: 0.6
6560; val_acc: 59.381%
Epoch 10(37s) - train_loss: 0.66354; train_acc: 59.774%; val_loss: 0.
66302; val_acc: 59.846%
Epoch 13(37s) - train_loss: 0.66039; train_acc: 60.247%; val_loss: 0.
66071; val_acc: 60.266%
Epoch 16(37s) - train_loss: 0.65751; train_acc: 60.665%; val_loss: 0.
65867; val_acc: 60.581%
Epoch 19(37s) - train_loss: 0.65503; train_acc: 61.073%; val_loss: 0.
65705; val_acc: 60.740%
Epoch 22(37s) - train_loss: 0.65267; train_acc: 61.443%; val_loss: 0.
65570; val_acc: 60.904%
Epoch 25(37s) - train_loss: 0.65073; train_acc: 61.640%; val_loss: 0.
65479; val_acc: 61.066%
Epoch 28(37s) - train_loss: 0.64900; train_acc: 61.861%; val_loss: 0.
65382; val_acc: 61.200%
Epoch 31(37s) - train_loss: 0.64744; train_acc: 62.122%; val_loss: 0.
65334; val_acc: 61.347%
Epoch 34(37s) - train_loss: 0.64579; train_acc: 62.296%; val_loss: 0.
65286; val_acc: 61.410%
Epoch 37(37s) - train_loss: 0.64471; train_acc: 62.447%; val_loss: 0.
65258; val_acc: 61.441%
Epoch 40(37s) - train_loss: 0.64331; train_acc: 62.611%; val_loss: 0.
65245; val_acc: 61.487%
```

```
Epoch 43(37s) - train_loss: 0.64220; train_acc: 62.797%; val_loss: 0.
65209; val_acc: 61.442%
Epoch 46(37s) - train_loss: 0.64137; train_acc: 62.938%; val_loss: 0.
65198; val_acc: 61.481%
Epoch 49(37s) - train_loss: 0.64019; train_acc: 63.067%; val_loss: 0.
65193; val_acc: 61.485%
Epoch 52(37s) - train_loss: 0.63914; train_acc: 63.227%; val_loss: 0.
65190; val_acc: 61.459%
Training round took 33:33
Test loss: 0.65184; acc: 61.552%
```

```
500 nodes test
Epoch 1(21s) - train_loss: 0.68793; train_acc: 54.143%; val_loss: 0.6
7615; val_acc: 58.044%
Epoch 4(22s) - train_loss: 0.67350; train_acc: 57.996%; val_loss: 0.6
6935; val_acc: 58.815%
Epoch 7(23s) - train_loss: 0.66998; train_acc: 58.683%; val_loss: 0.6
6687; val_acc: 59.305%
Epoch 10(22s) - train_loss: 0.66761; train_acc: 59.089%; val_loss: 0.
66496; val_acc: 59.580%
Epoch 13(22s) - train_loss: 0.66511; train_acc: 59.533%; val_loss: 0.
66313; val_acc: 59.849%
Epoch 16(23s) - train_loss: 0.66300; train_acc: 59.875%; val_loss: 0.
66139; val_acc: 60.171%
Epoch 19(22s) - train_loss: 0.66089; train_acc: 60.126%; val_loss: 0.
65974; val_acc: 60.402%
Epoch 22(22s) - train_loss: 0.65886; train_acc: 60.474%; val_loss: 0.
65832; val_acc: 60.532%
Epoch 25(22s) - train_loss: 0.65720; train_acc: 60.726%; val_loss: 0.
65711; val_acc: 60.762%
Epoch 28(24s) - train_loss: 0.65575; train_acc: 60.940%; val_loss: 0.
65593; val_acc: 60.911%
Epoch 31(23s) - train_loss: 0.65413; train_acc: 61.183%; val_loss: 0.
65502; val_acc: 60.990%
Epoch 34(21s) - train_loss: 0.65273; train_acc: 61.398%; val_loss: 0.
65429; val_acc: 61.146%
Epoch 37(21s) - train_loss: 0.65183; train_acc: 61.555%; val_loss: 0.
65368; val_acc: 61.192%
Epoch 40(23s) - train_loss: 0.65070; train_acc: 61.697%; val_loss: 0.
65312; val_acc: 61.306%
Epoch 43(20s) - train_loss: 0.64954; train_acc: 61.845%; val_loss: 0.
65260; val_acc: 61.329%
Epoch 46(20s) - train_loss: 0.64873; train_acc: 61.928%; val_loss: 0.
65221; val_acc: 61.314%
Epoch 49(20s) - train_loss: 0.64798; train_acc: 62.026%; val_loss: 0.
65191; val_acc: 61.374%
Epoch 52(21s) - train_loss: 0.64706; train_acc: 62.167%; val_loss: 0.
65169; val_acc: 61.364%
Epoch 55(21s) - train_loss: 0.64646; train_acc: 62.206%; val_loss: 0.
65139; val_acc: 61.441%
Epoch 58(20s) - train_loss: 0.64571; train_acc: 62.385%; val_loss: 0.
65140; val_acc: 61.385%
Epoch 61(20s) - train_loss: 0.64519; train_acc: 62.441%; val_loss: 0.
65101; val_acc: 61.461%
Epoch 64(20s) - train_loss: 0.64447; train_acc: 62.555%; val_loss: 0.
65094; val_acc: 61.446%
```

```
Epoch 67(20s) - train_loss: 0.64390; train_acc: 62.620%; val_loss: 0.
65084; val_acc: 61.457%
Epoch 70(23s) - train_loss: 0.64345; train_acc: 62.621%; val_loss: 0.
65077; val_acc: 61.454%
Epoch 73(23s) - train_loss: 0.64293; train_acc: 62.760%; val_loss: 0.
65066; val_acc: 61.504%
Epoch 76(23s) - train_loss: 0.64247; train_acc: 62.801%; val_loss: 0.
65056; val_acc: 61.470%
Epoch 79(23s) - train_loss: 0.64167; train_acc: 62.918%; val_loss: 0.
65052; val_acc: 61.488%
Epoch 82(22s) - train_loss: 0.64112; train_acc: 62.920%; val_loss: 0.
65038; val_acc: 61.527%
Epoch 85(23s) - train_loss: 0.64071; train_acc: 62.965%; val_loss: 0.
65043; val_acc: 61.527%
Epoch 88(23s) - train_loss: 0.64024; train_acc: 63.035%; val_loss: 0.
65031; val_acc: 61.525%
Epoch 91(23s) - train_loss: 0.64014; train_acc: 63.052%; val_loss: 0.
65027; val_acc: 61.552%
Epoch 94(23s) - train_loss: 0.63934; train_acc: 63.176%; val_loss: 0.
65029; val_acc: 61.529%
Training round took 34:32
Test loss: 0.65027; acc: 61.552%
Model accuracy: 61.55% +/- 0.00%
Model test_loss: 0.65027 +/- 0.00000
```

```
1500 nodes test
Epoch 1(56s) - train_loss: 0.68116; train_acc: 56.116%; val_loss: 0.6
7264; val_acc: 58.442%
Epoch 4(55s) - train_loss: 0.66917; train_acc: 58.817%; val_loss: 0.6
6710; val_acc: 59.230%
Epoch 7(58s) - train_loss: 0.66471; train_acc: 59.512%; val_loss: 0.6
6360; val_acc: 59.777%
Epoch 10(58s) - train_loss: 0.66089; train_acc: 60.172%; val_loss: 0.
66066; val_acc: 60.202%
Epoch 13(61s) - train_loss: 0.65722; train_acc: 60.743%; val_loss: 0.
65837; val_acc: 60.631%
Epoch 16(61s) - train_loss: 0.65407; train_acc: 61.227%; val_loss: 0.
65639; val_acc: 60.810%
Epoch 19(58s) - train_loss: 0.65131; train_acc: 61.600%; val_loss: 0.
65491; val_acc: 61.070%
Epoch 22(54s) - train_loss: 0.64880; train_acc: 61.973%; val_loss: 0.
65390; val_acc: 61.188%
Epoch 25(56s) - train_loss: 0.64682; train_acc: 62.189%; val_loss: 0.
65326; val_acc: 61.298%
Epoch 28(56s) - train_loss: 0.64489; train_acc: 62.439%; val_loss: 0.
65263; val_acc: 61.300%
Epoch 31(60s) - train_loss: 0.64328; train_acc: 62.690%; val_loss: 0.
65230; val_acc: 61.371%
Epoch 34(54s) - train_loss: 0.64164; train_acc: 62.898%; val_loss: 0.
65205; val_acc: 61.361%
Epoch 37(54s) - train_loss: 0.64041; train_acc: 63.019%; val_loss: 0.
65209; val_acc: 61.437%
Training round took 37:10
Test loss: 0.65203; acc: 61.423%
Model accuracy: 61.42% +/- 0.00%
Model test_loss: 0.65203 +/- 0.00000
```

```
no PReLU 1000 test
Epoch 1(32s) - train_loss: 0.68187; train_acc: 56.087%; val_loss: 0.6
7127; val_acc: 58.538%
Epoch 4(30s) - train_loss: 0.67137; train_acc: 58.406%; val_loss: 0.6
6712; val_acc: 59.275%
Epoch 7(31s) - train_loss: 0.66864; train_acc: 58.843%; val_loss: 0.6
6545; val_acc: 59.504%
Epoch 10(32s) - train_loss: 0.66725; train_acc: 59.088%; val_loss: 0.
66441; val_acc: 59.643%
Epoch 13(30s) - train_loss: 0.66647; train_acc: 59.261%; val_loss: 0.
66390; val_acc: 59.630%
Epoch 16(31s) - train_loss: 0.66590; train_acc: 59.285%; val_loss: 0.
66356; val_acc: 59.666%
Epoch 19(31s) - train_loss: 0.66532; train_acc: 59.409%; val_loss: 0.
66343; val_acc: 59.730%
Epoch 22(31s) - train_loss: 0.66529; train_acc: 59.407%; val_loss: 0.
66324; val_acc: 59.736%
Epoch 25(30s) - train_loss: 0.66501; train_acc: 59.432%; val_loss: 0.
66344; val_acc: 59.702%
Epoch 28(30s) - train_loss: 0.66479; train_acc: 59.444%; val_loss: 0.
66308; val_acc: 59.768%
Epoch 31(31s) - train_loss: 0.66462; train_acc: 59.484%; val_loss: 0.
66314; val_acc: 59.746%
Training round took 16:23
Test loss: 0.66308; acc: 59.768%
Model accuracy: 59.77% +/- 0.00%
Model test_loss: 0.66308 +/- 0.00000
```

```
no PReLU 500 test
Epoch 1(17s) - train_loss: 0.68884; train_acc: 54.715%; val_loss: 0.6
7416; val_acc: 57.877%
Epoch 4(17s) - train_loss: 0.67413; train_acc: 57.939%; val_loss: 0.6
6977; val_acc: 58.639%
Epoch 7(17s) - train_loss: 0.67109; train_acc: 58.490%; val_loss: 0.6
6797; val_acc: 58.834%
Epoch 10(17s) - train_loss: 0.66923; train_acc: 58.777%; val_loss: 0.
66668; val_acc: 59.081%
Epoch 13(17s) - train_loss: 0.66800; train_acc: 59.029%; val_loss: 0.
66582; val_acc: 59.240%
Epoch 16(17s) - train_loss: 0.66727; train_acc: 59.090%; val_loss: 0.
66526; val_acc: 59.311%
Epoch 19(17s) - train_loss: 0.66667; train_acc: 59.149%; val_loss: 0.
66488; val_acc: 59.376%
Epoch 22(17s) - train_loss: 0.66617; train_acc: 59.282%; val_loss: 0.
66466; val_acc: 59.332%
Epoch 25(17s) - train_loss: 0.66585; train_acc: 59.340%; val_loss: 0.
66443; val_acc: 59.357%
Epoch 28(17s) - train_loss: 0.66554; train_acc: 59.396%; val_loss: 0.
66424; val_acc: 59.401%
Epoch 31(17s) - train_loss: 0.66540; train_acc: 59.403%; val_loss: 0.
66416; val_acc: 59.436%
Epoch 34(17s) - train_loss: 0.66504; train_acc: 59.476%; val_loss: 0.
66406; val_acc: 59.413%
Epoch 37(17s) - train_loss: 0.66491; train_acc: 59.456%; val_loss: 0.
66400; val_acc: 59.419%
Epoch 40(17s) - train_loss: 0.66472; train_acc: 59.520%; val_loss: 0.
66408; val_acc: 59.416%
```

```
Epoch 43(17s) - train_loss: 0.66454; train_acc: 59.526%; val_loss: 0.
66389; val_acc: 59.442%
Epoch 46(17s) - train_loss: 0.66453; train_acc: 59.568%; val_loss: 0.
66406; val_acc: 59.383%
Epoch 49(17s) - train_loss: 0.66430; train_acc: 59.592%; val_loss: 0.
66380; val_acc: 59.472%
Epoch 52(17s) - train_loss: 0.66430; train_acc: 59.561%; val_loss: 0.
66380; val_acc: 59.513%
Epoch 55(17s) - train_loss: 0.66416; train_acc: 59.594%; val_loss: 0.
66374; val_acc: 59.545%
Epoch 58(17s) - train_loss: 0.66411; train_acc: 59.652%; val_loss: 0.
66373; val_acc: 59.497%
Training round took 16:07
Test loss: 0.66373; acc: 59.525%
Model accuracy: 59.52% +/- 0.00%
Model test_loss: 0.66373 +/- 0.00000
```

```
no PReLU 1500 test
Epoch 1(42s) - train_loss: 0.68094; train_acc: 56.197%; val_loss: 0.6
7148; val_acc: 58.295%
Epoch 4(42s) - train_loss: 0.67004; train_acc: 58.680%; val_loss: 0.6
6694; val_acc: 59.170%
Epoch 7(42s) - train_loss: 0.66749; train_acc: 59.068%; val_loss: 0.6
6533; val_acc: 59.256%
Epoch 10(42s) - train_loss: 0.66621; train_acc: 59.241%; val_loss: 0.
66440; val_acc: 59.587%
Epoch 13(42s) - train_loss: 0.66563; train_acc: 59.323%; val_loss: 0.
66407; val_acc: 59.599%
Epoch 16(42s) - train_loss: 0.66533; train_acc: 59.380%; val_loss: 0.
66401; val_acc: 59.576%
Epoch 19(42s) - train_loss: 0.66490; train_acc: 59.424%; val_loss: 0.
66375; val_acc: 59.654%
Epoch 22(42s) - train_loss: 0.66475; train_acc: 59.485%; val_loss: 0.
66366; val_acc: 59.673%
Training round took 15:19
Test loss: 0.66364; acc: 59.691%
Model accuracy: 59.69% +/- 0.00%
Model test_loss: 0.66364 +/- 0.00000
```

```
no dropout 500 test
Epoch 1(13s) - train_loss: 0.68037; train_acc: 56.314%; val_loss: 0.6
7281; val_acc: 58.085%
Epoch 4(12s) - train_loss: 0.66797; train_acc: 59.038%; val_loss: 0.6
6782; val_acc: 58.895%
Epoch 7(12s) - train_loss: 0.66546; train_acc: 59.422%; val_loss: 0.6
6584; val_acc: 59.364%
Epoch 10(12s) - train_loss: 0.66416; train_acc: 59.602%; val_loss: 0.
66486; val_acc: 59.418%
Epoch 13(12s) - train_loss: 0.66359; train_acc: 59.658%; val_loss: 0.
66435; val_acc: 59.484%
Epoch 16(12s) - train_loss: 0.66326; train_acc: 59.703%; val_loss: 0.
66413; val_acc: 59.594%
Epoch 19(12s) - train_loss: 0.66309; train_acc: 59.737%; val_loss: 0.
66395; val_acc: 59.492%
Epoch 22(12s) - train_loss: 0.66299; train_acc: 59.745%; val_loss: 0.
66380; val_acc: 59.538%
```

```
Epoch 25(12s) - train_loss: 0.66293; train_acc: 59.755%; val_loss: 0.
66390; val_acc: 59.532%
Epoch 28(12s) - train_loss: 0.66287; train_acc: 59.763%; val_loss: 0.
66376; val_acc: 59.502%
Training round took 5:46
Test loss: 0.66373; acc: 59.606%
Model accuracy: 59.61% +/- 0.00%
Model test_loss: 0.66373 +/- 0.00000
```

```
no dropout 1000 test
Epoch 1(21s) - train_loss: 0.67716; train_acc: 57.216%; val_loss: 0.6
7161; val_acc: 58.337%
Epoch 4(20s) - train_loss: 0.66688; train_acc: 59.179%; val_loss: 0.6
6611; val_acc: 59.362%
Epoch 7(20s) - train_loss: 0.66465; train_acc: 59.515%; val_loss: 0.6
6427; val_acc: 59.617%
Epoch 10(20s) - train_loss: 0.66380; train_acc: 59.596%; val_loss: 0.
66347; val_acc: 59.800%
Epoch 13(20s) - train_loss: 0.66351; train_acc: 59.661%; val_loss: 0.
66303; val_acc: 59.839%
Epoch 16(20s) - train_loss: 0.66332; train_acc: 59.649%; val_loss: 0.
66277; val_acc: 59.886%
Epoch 19(20s) - train_loss: 0.66316; train_acc: 59.736%; val_loss: 0.
66297; val_acc: 59.768%
Epoch 22(20s) - train_loss: 0.66318; train_acc: 59.699%; val_loss: 0.
66246; val_acc: 59.925%
Epoch 25(20s) - train_loss: 0.66311; train_acc: 59.701%; val_loss: 0.
66263; val_acc: 59.888%
Training round took 8:49
Test loss: 0.66246; acc: 59.925%
Model accuracy: 59.92% +/- 0.00%
Model test_loss: 0.66246 +/- 0.00000
```

```
no dropout 1500 test
Epoch 1(29s) - train_loss: 0.67451; train_acc: 57.835%; val_loss: 0.6
7045; val_acc: 58.512%
Epoch 4(28s) - train_loss: 0.66584; train_acc: 59.341%; val_loss: 0.6
6589; val_acc: 59.384%
Epoch 7(28s) - train_loss: 0.66403; train_acc: 59.577%; val_loss: 0.6
6462; val_acc: 59.432%
Epoch 10(28s) - train_loss: 0.66349; train_acc: 59.696%; val_loss: 0.
66407; val_acc: 59.454%
Epoch 13(28s) - train_loss: 0.66328; train_acc: 59.697%; val_loss: 0.
66390; val_acc: 59.642%
Epoch 16(28s) - train_loss: 0.66321; train_acc: 59.736%; val_loss: 0.
66382; val_acc: 59.533%
Epoch 19(28s) - train_loss: 0.66313; train_acc: 59.711%; val_loss: 0.
66375; val_acc: 59.518%
Training round took 8:51
Test loss: 0.66375; acc: 59.460%
Model accuracy: 59.46% +/- 0.00%
Model test_loss: 0.66375 +/- 0.00000
```

```
.5 adam beta
```

```
Epoch 1(38s) - train_loss: 0.68378; train_acc: 55.384%; val_loss: 0.6
7356; val_acc: 58.235%
Epoch 4(38s) - train_loss: 0.67064; train_acc: 58.525%; val_loss: 0.6
6791; val_acc: 58.983%
Epoch 7(41s) - train_loss: 0.66699; train_acc: 59.160%; val_loss: 0.6
6489; val_acc: 59.537%
Epoch 10(40s) - train_loss: 0.66354; train_acc: 59.694%; val_loss: 0.
66254; val_acc: 59.920%
Epoch 13(41s) - train_loss: 0.66058; train_acc: 60.209%; val_loss: 0.
66012; val_acc: 60.298%
Epoch 16(40s) - train_loss: 0.65776; train_acc: 60.658%; val_loss: 0.
65804; val_acc: 60.730%
Epoch 19(42s) - train_loss: 0.65526; train_acc: 60.995%; val_loss: 0.
65635; val_acc: 60.884%
Epoch 22(41s) - train_loss: 0.65270; train_acc: 61.339%; val_loss: 0.
65507; val_acc: 61.026%
Epoch 25(46s) - train_loss: 0.65088; train_acc: 61.673%; val_loss: 0.
65400; val_acc: 61.140%
Epoch 28(41s) - train_loss: 0.64915; train_acc: 61.911%; val_loss: 0.
65317; val_acc: 61.274%
Epoch 31(40s) - train_loss: 0.64723; train_acc: 62.215%; val_loss: 0.
65258; val_acc: 61.294%
Epoch 34(44s) - train_loss: 0.64630; train_acc: 62.272%; val_loss: 0.
65211; val_acc: 61.340%
Epoch 37(44s) - train_loss: 0.64491; train_acc: 62.484%; val_loss: 0.
65175; val_acc: 61.319%
Epoch 40(43s) - train_loss: 0.64349; train_acc: 62.618%; val_loss: 0.
65149; val_acc: 61.485%
Epoch 43(44s) - train_loss: 0.64245; train_acc: 62.823%; val_loss: 0.
65115; val_acc: 61.453%
Epoch 46(42s) - train_loss: 0.64150; train_acc: 62.913%; val_loss: 0.
65099; val_acc: 61.523%
Epoch 49(42s) - train_loss: 0.64057; train_acc: 63.018%; val_loss: 0.
65083; val_acc: 61.529%
Epoch 52(41s) - train_loss: 0.63963; train_acc: 63.086%; val_loss: 0.
65076; val_acc: 61.513%
Epoch 55(41s) - train_loss: 0.63877; train_acc: 63.234%; val_loss: 0.
65071; val_acc: 61.520%
Epoch 58(49s) - train_loss: 0.63790; train_acc: 63.344%; val_loss: 0.
65078; val_acc: 61.530%
Epoch 61(48s) - train_loss: 0.63712; train_acc: 63.474%; val_loss: 0.
65065; val_acc: 61.531%
Epoch 64(48s) - train_loss: 0.63644; train_acc: 63.524%; val_loss: 0.
65066; val_acc: 61.553%
Training round took 46:05
Test loss: 0.65065; acc: 61.531%
Model accuracy: 61.53% +/- 0.00%
Model test_loss: 0.65065 +/- 0.00000
```

```
.001 adam learn rate
Epoch 1(48s) - train_loss: 0.67197; train_acc: 58.156%; val_loss: 0.6
6432; val_acc: 59.471%
Epoch 4(48s) - train_loss: 0.65181; train_acc: 61.443%; val_loss: 0.6
5281; val_acc: 61.347%
Epoch 7(48s) - train_loss: 0.64391; train_acc: 62.594%; val_loss: 0.6
5283; val_acc: 61.202%
Training round took 7:10
```

```
Test loss: 0.65198; acc: 61.450%
Model accuracy: 61.45% +/- 0.00%
Model test_loss: 0.65198 +/- 0.00000
```

```
.0001 adam learn rate
Epoch 1(49s) - train_loss: 0.67934; train_acc: 56.600%; val_loss: 0.6
7128; val_acc: 58.556%
Epoch 4(48s) - train_loss: 0.66688; train_acc: 59.158%; val_loss: 0.6
6485; val_acc: 59.599%
Epoch 7(51s) - train_loss: 0.66061; train_acc: 60.225%; val_loss: 0.6
6003; val_acc: 60.364%
Epoch 10(49s) - train_loss: 0.65528; train_acc: 61.013%; val_loss: 0.
65650; val_acc: 60.859%
Epoch 13(41s) - train_loss: 0.65118; train_acc: 61.561%; val_loss: 0.
65430; val_acc: 61.171%
Epoch 16(41s) - train_loss: 0.64807; train_acc: 62.088%; val_loss: 0.
65293; val_acc: 61.388%
Epoch 19(42s) - train_loss: 0.64550; train_acc: 62.353%; val_loss: 0.
65275; val_acc: 61.441%
Epoch 22(41s) - train_loss: 0.64326; train_acc: 62.676%; val_loss: 0.
65200; val_acc: 61.339%
Epoch 25(42s) - train_loss: 0.64134; train_acc: 62.919%; val_loss: 0.
65175; val_acc: 61.480%
Epoch 28(40s) - train_loss: 0.63935; train_acc: 63.171%; val_loss: 0.
65172; val_acc: 61.407%
Epoch 31(43s) - train_loss: 0.63774; train_acc: 63.394%; val_loss: 0.
65156; val_acc: 61.510%
Training round took 24:19
Test loss: 0.65151; acc: 61.475%
Model accuracy: 61.48% +/- 0.00%
Model test_loss: 0.65151 +/- 0.00000
```

```
.00001 adam learn rate
Epoch 1(43s) - train_loss: 0.69355; train_acc: 52.173%; val_loss: 0.6
8561; val_acc: 55.740%
Epoch 4(42s) - train_loss: 0.67963; train_acc: 56.673%; val_loss: 0.6
7489; val_acc: 58.013%
Epoch 7(40s) - train_loss: 0.67569; train_acc: 57.553%; val_loss: 0.6
7171; val_acc: 58.442%
Epoch 10(39s) - train_loss: 0.67380; train_acc: 57.964%; val_loss: 0.
67030; val_acc: 58.632%
Epoch 13(39s) - train_loss: 0.67214; train_acc: 58.295%; val_loss: 0.
66928; val_acc: 58.734%
Epoch 16(40s) - train_loss: 0.67120; train_acc: 58.402%; val_loss: 0.
66849; val_acc: 58.867%
Epoch 19(41s) - train_loss: 0.67037; train_acc: 58.574%; val_loss: 0.
66783; val_acc: 58.918%
Epoch 22(41s) - train_loss: 0.66940; train_acc: 58.750%; val_loss: 0.
66718; val_acc: 59.077%
Epoch 25(43s) - train_loss: 0.66866; train_acc: 58.901%; val_loss: 0.
66661; val_acc: 59.153%
Epoch 28(44s) - train_loss: 0.66783; train_acc: 59.060%; val_loss: 0.
66603; val_acc: 59.203%
Epoch 31(42s) - train_loss: 0.66728; train_acc: 59.148%; val_loss: 0.
66547; val_acc: 59.311%
```

```
Epoch 34(38s) - train_loss: 0.66655; train_acc: 59.238%; val_loss: 0.
66495; val_acc: 59.417%
Epoch 37(38s) - train_loss: 0.66577; train_acc: 59.372%; val_loss: 0.
66442; val_acc: 59.459%
Epoch 40(38s) - train_loss: 0.66500; train_acc: 59.493%; val_loss: 0.
66393; val_acc: 59.504%
Epoch 43(38s) - train_loss: 0.66446; train_acc: 59.597%; val_loss: 0.
66344; val_acc: 59.615%
Epoch 46(38s) - train_loss: 0.66362; train_acc: 59.713%; val_loss: 0.
66296; val_acc: 59.759%
Epoch 49(38s) - train_loss: 0.66319; train_acc: 59.820%; val_loss: 0.
66251; val_acc: 59.831%
Epoch 52(38s) - train_loss: 0.66266; train_acc: 59.913%; val_loss: 0.
66204; val_acc: 59.896%
Epoch 55(38s) - train_loss: 0.66193; train_acc: 59.954%; val_loss: 0.
66160; val_acc: 59.974%
Epoch 58(38s) - train_loss: 0.66141; train_acc: 60.135%; val_loss: 0.
66115; val_acc: 60.025%
Epoch 61(44s) - train_loss: 0.66077; train_acc: 60.151%; val_loss: 0.
66074; val_acc: 60.112%
Epoch 64(42s) - train_loss: 0.66015; train_acc: 60.300%; val_loss: 0.
66032; val_acc: 60.181%
Epoch 67(41s) - train_loss: 0.65966; train_acc: 60.381%; val_loss: 0.
65992; val_acc: 60.244%
Epoch 70(41s) - train_loss: 0.65902; train_acc: 60.413%; val_loss: 0.
65956; val_acc: 60.331%
Epoch 73(41s) - train_loss: 0.65843; train_acc: 60.563%; val_loss: 0.
65916; val_acc: 60.378%
Epoch 76(40s) - train_loss: 0.65797; train_acc: 60.604%; val_loss: 0.
65881; val_acc: 60.437%
Epoch 79(40s) - train_loss: 0.65750; train_acc: 60.743%; val_loss: 0.
65842; val_acc: 60.513%
Epoch 82(39s) - train_loss: 0.65694; train_acc: 60.812%; val_loss: 0.
65807; val_acc: 60.580%
Epoch 85(43s) - train_loss: 0.65630; train_acc: 60.882%; val_loss: 0.
65774; val_acc: 60.594%
Epoch 88(39s) - train_loss: 0.65584; train_acc: 60.955%; val_loss: 0.
65742; val_acc: 60.693%
Epoch 91(42s) - train_loss: 0.65528; train_acc: 61.043%; val_loss: 0.
65709; val_acc: 60.727%
Epoch 94(42s) - train_loss: 0.65499; train_acc: 61.094%; val_loss: 0.
65680; val_acc: 60.800%
Epoch 97(39s) - train_loss: 0.65437; train_acc: 61.164%; val_loss: 0.
65652; val_acc: 60.818%
Epoch 100(41s) - train_loss: 0.65388; train_acc: 61.220%; val_loss: 0
.65623; val_acc: 60.862%
Training round took 67:13
Test loss: 0.65623; acc: 60.862%
Model accuracy: 60.86% +/- 0.00%
Model test_loss: 0.65623 +/- 0.00000
```

```
initialmodel with dropout
Epoch 1(38s) - train_loss: 0.68322; train_acc: 56.023%; val_loss: 0.6
7140; val_acc: 58.419%
Epoch 4(50s) - train_loss: 0.67081; train_acc: 58.533%; val_loss: 0.6
6741; val_acc: 59.121%
```

```
Epoch 7(45s) - train_loss: 0.66811; train_acc: 59.020%; val_loss: 0.6
6589; val_acc: 59.318%
Epoch 10(52s) - train_loss: 0.66680; train_acc: 59.204%; val_loss: 0.
66511; val_acc: 59.486%
Epoch 13(45s) - train_loss: 0.66617; train_acc: 59.271%; val_loss: 0.
66468; val_acc: 59.552%
Epoch 16(50s) - train_loss: 0.66541; train_acc: 59.399%; val_loss: 0.
66444; val_acc: 59.514%
Epoch 19(44s) - train_loss: 0.66501; train_acc: 59.490%; val_loss: 0.
66439; val_acc: 59.544%
Epoch 22(47s) - train_loss: 0.66464; train_acc: 59.523%; val_loss: 0.
66430; val_acc: 59.569%
Epoch 25(47s) - train_loss: 0.66450; train_acc: 59.513%; val_loss: 0.
66432; val_acc: 59.621%
Epoch 28(40s) - train_loss: 0.66437; train_acc: 59.522%; val_loss: 0.
66434; val_acc: 59.612%
Epoch 31(45s) - train_loss: 0.66412; train_acc: 59.556%; val_loss: 0.
66411; val_acc: 59.561%
Epoch 34(39s) - train_loss: 0.66408; train_acc: 59.590%; val_loss: 0.
66407; val_acc: 59.566%
Epoch 37(39s) - train_loss: 0.66393; train_acc: 59.607%; val_loss: 0.
66404; val_acc: 59.565%
Epoch 40(46s) - train_loss: 0.66385; train_acc: 59.619%; val_loss: 0.
66396; val_acc: 59.593%
Epoch 43(48s) - train_loss: 0.66368; train_acc: 59.610%; val_loss: 0.
66407; val_acc: 59.590%
Training round took 33:10
Test loss: 0.66396; acc: 59.593%
Model accuracy: 59.59% +/- 0.00%
Model test_loss: 0.66396 +/- 0.00000
```

```
initialmodel with dropout + PReLU
Epoch 1(44s) - train_loss: 0.68404; train_acc: 55.353%; val_loss: 0.6
7376; val_acc: 58.056%
Epoch 4(43s) - train_loss: 0.66984; train_acc: 58.685%; val_loss: 0.6
6803; val_acc: 59.012%
Epoch 7(43s) - train_loss: 0.66507; train_acc: 59.493%; val_loss: 0.6
6462; val_acc: 59.723%
Epoch 10(43s) - train_loss: 0.66088; train_acc: 60.161%; val_loss: 0.
66156; val_acc: 60.193%
Epoch 13(43s) - train_loss: 0.65714; train_acc: 60.728%; val_loss: 0.
65896; val_acc: 60.552%
Epoch 16(43s) - train_loss: 0.65392; train_acc: 61.183%; val_loss: 0.
65706; val_acc: 60.808%
Epoch 19(43s) - train_loss: 0.65167; train_acc: 61.521%; val_loss: 0.
65594; val_acc: 61.004%
Epoch 22(43s) - train_loss: 0.64946; train_acc: 61.875%; val_loss: 0.
65506; val_acc: 61.086%
Epoch 25(43s) - train_loss: 0.64775; train_acc: 62.061%; val_loss: 0.
65433; val_acc: 61.169%
Epoch 28(43s) - train_loss: 0.64629; train_acc: 62.263%; val_loss: 0.
65400; val_acc: 61.131%
Epoch 31(43s) - train_loss: 0.64511; train_acc: 62.399%; val_loss: 0.
65367; val_acc: 61.192%
Epoch 34(43s) - train_loss: 0.64388; train_acc: 62.536%; val_loss: 0.
65344; val_acc: 61.168%
```

```
Epoch 37(43s) - train_loss: 0.64290; train_acc: 62.720%; val_loss: 0.
65332; val_acc: 61.191%
Epoch 40(43s) - train_loss: 0.64179; train_acc: 62.861%; val_loss: 0.
65334; val_acc: 61.229%
Epoch 43(43s) - train_loss: 0.64082; train_acc: 62.961%; val_loss: 0.
65321; val_acc: 61.187%
Epoch 46(43s) - train_loss: 0.64002; train_acc: 63.056%; val_loss: 0.
65319; val_acc: 61.182%
Epoch 49(44s) - train_loss: 0.63909; train_acc: 63.209%; val_loss: 0.
65308; val_acc: 61.201%
Epoch 52(43s) - train_loss: 0.63889; train_acc: 63.198%; val_loss: 0.
65307; val_acc: 61.172%
Epoch 55(43s) - train_loss: 0.63758; train_acc: 63.402%; val_loss: 0.
65324; val_acc: 61.204%
Training round took 40:13
Test loss: 0.65307; acc: 61.172%
Model accuracy: 61.17% +/- 0.00%
Model test_loss: 0.65307 +/- 0.00000
```

```
initialmodel with dropout + PReLU + 1000 Nodes
Epoch 1(79s) - train_loss: 0.67938; train_acc: 56.525%; val_loss: 0.6
7114; val_acc: 58.566%
Epoch 4(78s) - train_loss: 0.66661; train_acc: 59.222%; val_loss: 0.6
6486; val_acc: 59.492%
Epoch 7(78s) - train_loss: 0.66054; train_acc: 60.269%; val_loss: 0.6
6036; val_acc: 60.212%
Epoch 10(78s) - train_loss: 0.65531; train_acc: 61.026%; val_loss: 0.
65677; val_acc: 60.736%
Epoch 13(78s) - train_loss: 0.65120; train_acc: 61.609%; val_loss: 0.
65454; val_acc: 60.955%
Epoch 16(78s) - train_loss: 0.64768; train_acc: 62.033%; val_loss: 0.
65322; val_acc: 61.039%
Epoch 19(78s) - train_loss: 0.64521; train_acc: 62.420%; val_loss: 0.
65250; val_acc: 61.247%
Epoch 22(78s) - train_loss: 0.64292; train_acc: 62.714%; val_loss: 0.
65216; val_acc: 61.219%
Epoch 25(78s) - train_loss: 0.64087; train_acc: 62.968%; val_loss: 0.
65182; val_acc: 61.351%
Epoch 28(78s) - train_loss: 0.63919; train_acc: 63.216%; val_loss: 0.
65204; val_acc: 61.179%
Epoch 31(78s) - train_loss: 0.63766; train_acc: 63.408%; val_loss: 0.
65176; val_acc: 61.358%
Training round took 40:22
Test loss: 0.65163; acc: 61.370%
Model accuracy: 61.37% +/- 0.00%
Model test_loss: 0.65163 +/- 0.00000
```

```
initialmodel with dropout + PReLU + 1500 Nodes
Epoch 1(113s) - train_loss: 0.67789; train_acc: 56.857%; val_loss: 0.
67033; val_acc: 58.620%
Epoch 4(156s) - train_loss: 0.66470; train_acc: 59.577%; val_loss: 0.
66328; val_acc: 59.827%
Epoch 7(152s) - train_loss: 0.65706; train_acc: 60.727%; val_loss: 0.
65831; val_acc: 60.510%
Epoch 10(147s) - train_loss: 0.65143; train_acc: 61.604%; val_loss: 0
.65537; val_acc: 60.926%
```

```
Epoch 13(138s) - train_loss: 0.64692; train_acc: 62.198%; val_loss: 0
.65374; val_acc: 61.125%
Epoch 16(135s) - train_loss: 0.64366; train_acc: 62.610%; val_loss: 0
.65302; val_acc: 61.317%
Epoch 19(134s) - train_loss: 0.64080; train_acc: 63.013%; val_loss: 0
.65260; val_acc: 61.301%
Epoch 22(130s) - train_loss: 0.63846; train_acc: 63.287%; val_loss: 0
.65268; val_acc: 61.278%
Epoch 25(159s) - train_loss: 0.63610; train_acc: 63.546%; val_loss: 0
.65262; val_acc: 61.365%
Training round took 58:44
Test loss: 0.65240; acc: 61.346%
Model accuracy: 61.35% +/- 0.00%
Model test_loss: 0.65240 +/- 0.00000
```

```
initialmodel with dropout + PReLU + 1000 Nodes + 0.00001
Epoch 1(87s) - train_loss: 0.69305; train_acc: 51.960%; val_loss: 0.6
8598; val_acc: 56.045%
Epoch 4(95s) - train_loss: 0.67917; train_acc: 56.807%; val_loss: 0.6
7542; val_acc: 57.987%
Epoch 7(94s) - train_loss: 0.67521; train_acc: 57.688%; val_loss: 0.6
7221; val_acc: 58.417%
Epoch 10(91s) - train_loss: 0.67340; train_acc: 57.971%; val_loss: 0.
67075; val_acc: 58.580%
Epoch 13(112s) - train_loss: 0.67211; train_acc: 58.221%; val_loss: 0
.66979; val_acc: 58.718%
Epoch 16(105s) - train_loss: 0.67096; train_acc: 58.469%; val_loss: 0
.66901; val_acc: 58.840%
Epoch 19(86s) - train_loss: 0.67006; train_acc: 58.647%; val_loss: 0.
66837; val_acc: 58.946%
Epoch 22(98s) - train_loss: 0.66923; train_acc: 58.797%; val_loss: 0.
66769; val_acc: 59.072%
Epoch 25(91s) - train_loss: 0.66844; train_acc: 58.961%; val_loss: 0.
66708; val_acc: 59.191%
Epoch 28(89s) - train_loss: 0.66774; train_acc: 59.059%; val_loss: 0.
66655; val_acc: 59.295%
Epoch 31(103s) - train_loss: 0.66699; train_acc: 59.190%; val_loss: 0
.66595; val_acc: 59.385%
Epoch 34(86s) - train_loss: 0.66640; train_acc: 59.290%; val_loss: 0.
66541; val_acc: 59.442%
Epoch 37(88s) - train_loss: 0.66571; train_acc: 59.352%; val_loss: 0.
66490; val_acc: 59.522%
Epoch 40(88s) - train_loss: 0.66499; train_acc: 59.491%; val_loss: 0.
66433; val_acc: 59.607%
Epoch 43(94s) - train_loss: 0.66437; train_acc: 59.613%; val_loss: 0.
66381; val_acc: 59.681%
Epoch 46(88s) - train_loss: 0.66372; train_acc: 59.736%; val_loss: 0.
66332; val_acc: 59.772%
Epoch 49(91s) - train_loss: 0.66304; train_acc: 59.826%; val_loss: 0.
66282; val_acc: 59.854%
Epoch 52(95s) - train_loss: 0.66245; train_acc: 59.887%; val_loss: 0.
66231; val_acc: 59.933%
Epoch 55(90s) - train_loss: 0.66197; train_acc: 59.966%; val_loss: 0.
66184; val_acc: 59.987%
Epoch 58(92s) - train_loss: 0.66135; train_acc: 60.092%; val_loss: 0.
66138; val_acc: 60.099%
```

```
Epoch 61(93s) - train_loss: 0.66082; train_acc: 60.230%; val_loss: 0.
66092; val_acc: 60.155%
Epoch 64(90s) - train_loss: 0.66001; train_acc: 60.317%; val_loss: 0.
66048; val_acc: 60.245%
Epoch 67(96s) - train_loss: 0.65957; train_acc: 60.375%; val_loss: 0.
66005; val_acc: 60.319%
Epoch 70(79s) - train_loss: 0.65907; train_acc: 60.455%; val_loss: 0.
65963; val_acc: 60.410%
Epoch 73(80s) - train_loss: 0.65853; train_acc: 60.515%; val_loss: 0.
65923; val_acc: 60.451%
Epoch 76(80s) - train_loss: 0.65782; train_acc: 60.645%; val_loss: 0.
65881; val_acc: 60.531%
Epoch 79(79s) - train_loss: 0.65742; train_acc: 60.704%; val_loss: 0.
65844; val_acc: 60.580%
Epoch 82(79s) - train_loss: 0.65690; train_acc: 60.773%; val_loss: 0.
65807; val_acc: 60.672%
Epoch 85(79s) - train_loss: 0.65645; train_acc: 60.825%; val_loss: 0.
65770; val_acc: 60.704%
Epoch 88(79s) - train_loss: 0.65580; train_acc: 60.923%; val_loss: 0.
65735; val_acc: 60.783%
Epoch 91(79s) - train_loss: 0.65531; train_acc: 61.058%; val_loss: 0.
65700; val_acc: 60.829%
Epoch 94(79s) - train_loss: 0.65486; train_acc: 61.088%; val_loss: 0.
65671; val_acc: 60.865%
Epoch 97(79s) - train_loss: 0.65451; train_acc: 61.137%; val_loss: 0.
65640; val_acc: 60.909%
Epoch 100(79s) - train_loss: 0.65417; train_acc: 61.166%; val_loss: 0
.65610; val_acc: 60.938%
Training round took 146:43
Test loss: 0.65610; acc: 60.938%
Model accuracy: 60.94% +/- 0.00%
Model test_loss: 0.65610 +/- 0.00000
```

```
initialmodel with dropout + PReLU + 1500 Nodes + 0.00005 + .5 beta_1
Epoch 1(80s) - train_loss: 0.68376; train_acc: 55.380%; val_loss: 0.6
7430; val_acc: 58.065%
Epoch 4(79s) - train_loss: 0.67059; train_acc: 58.547%; val_loss: 0.6
6863; val_acc: 59.072%
Epoch 7(79s) - train_loss: 0.66652; train_acc: 59.259%; val_loss: 0.6
6570; val_acc: 59.476%
Epoch 10(78s) - train_loss: 0.66332; train_acc: 59.724%; val_loss: 0.
66334; val_acc: 59.879%
Epoch 13(79s) - train_loss: 0.66027; train_acc: 60.257%; val_loss: 0.
66109; val_acc: 60.240%
Epoch 16(79s) - train_loss: 0.65730; train_acc: 60.711%; val_loss: 0.
65919; val_acc: 60.539%
Epoch 19(78s) - train_loss: 0.65485; train_acc: 61.148%; val_loss: 0.
65767; val_acc: 60.745%
Epoch 22(78s) - train_loss: 0.65248; train_acc: 61.441%; val_loss: 0.
65635; val_acc: 60.979%
Epoch 25(79s) - train_loss: 0.65051; train_acc: 61.713%; val_loss: 0.
65543; val_acc: 61.117%
Epoch 28(78s) - train_loss: 0.64879; train_acc: 61.960%; val_loss: 0.
65485; val_acc: 61.089%
Epoch 31(79s) - train_loss: 0.64712; train_acc: 62.165%; val_loss: 0.
65452; val_acc: 61.040%
```

```
Epoch 34(79s) - train_loss: 0.64585; train_acc: 62.283%; val_loss: 0.
65363; val_acc: 61.336%
Epoch 37(79s) - train_loss: 0.64444; train_acc: 62.540%; val_loss: 0.
65336; val_acc: 61.294%
Epoch 40(79s) - train_loss: 0.64345; train_acc: 62.674%; val_loss: 0.
65297; val_acc: 61.295%
Epoch 43(79s) - train_loss: 0.64201; train_acc: 62.792%; val_loss: 0.
65286; val_acc: 61.296%
Epoch 46(80s) - train_loss: 0.64087; train_acc: 62.956%; val_loss: 0.
65280; val_acc: 61.338%
Epoch 49(79s) - train_loss: 0.64007; train_acc: 63.062%; val_loss: 0.
65261; val_acc: 61.303%
Epoch 52(79s) - train_loss: 0.63935; train_acc: 63.168%; val_loss: 0.
65253; val_acc: 61.303%
Epoch 55(79s) - train_loss: 0.63848; train_acc: 63.265%; val_loss: 0.
65268; val_acc: 61.270%
Training round took 73:27
Test loss: 0.65253; acc: 61.303%
Model accuracy: 61.30% +/- 0.00%
Model test_loss: 0.65253 +/- 0.00000
```