This document is sort of a diary where I write down my thought processes because I find it interesting to look back at my thought processes sometimes and it allows me to pick up from last session on working on this project. This document is not a formal document of any sort and is just notes for my thoughts. Feel free to skip over this document.

Motivation:

To gain more experience with supervised learning with neural networks. This project is a personal exploration to see if neural networks can learn human bias. The goal of this project is to train a neural network that could rank player performance in a league of legends game in a team that is biased to our friend group’s rankings. There are currently many tools out there that ranks player performance in a game of league of legends already, but within my group of friends, we often found them to be not accurate to what we believe is the correct rankings, due to many reasons such as:

1) They are generalized over the whole population of players of different skill levels and may not accurately reflect our friend group’s skill level

2) We may have some bias within our friend group on deciding what we determine as impactful, (rankings out there are refined by professionals and are probably more accurate than any model we make, but in the end, the goal of the project is to get predict rankings based on our bias)

3) Perhaps there are some moments in the game where the result of the game is hugely affected by, but not reflected by the statistics of the players (thought is, could a neural network possibly learn these?)

Honestly, I don’t expect much to come out of building this neural network supervised model, but I believe it will be a good learning experience to use the tools and theory I learnt in CS540 Intro to Artificial Intelligence at UW Madison.

March 11th, 2023:

The idea comes from a joke, me and my friends are playing league of legends, and someone suggests that we should do pushups based on our in-game performance to encourage a better health lifestyle. After a lot of discussion, we found it hard to analyze a player’s performance by eye as there are many factors to take account of and each role in the game has different weights towards these statistics. So I decided maybe I could try modelling this to give a better guideline to ranking and perhaps I could make a model that could predict these rankings.

March 13th, 2023:

After doing some research, I find out that Riot (The company behind league of legends) has an API where I can access statistical data of a match per player so with the help of Jasper and Issac, we came up a list of features that we think would contribute to describing a player’s performance. The features are as follows: (these are per player, 31 features (I think?) per player) so we will have a 5x31 matrix of features. We could maybe take the ratios between each feature between each player to normalize the data as the data is based on the game time. Kind of stuck on how we can transform this 5x31 matrix into a feature vector x, would flatten mess up my data? Initial ideas on neural network -> last layer is a linear layer with 5 nodes, which will give us 5 values and we can base our rankings off that.

|  |
| --- |
| kills    int |
| assists    int |
| champExperience    int |
| damageDealtToBuildings    int |
| damageDealtToObjectives    int |
| deaths    int |
| doubleKills    int |
| dragonKills    int |
| firstBloodAssist    boolean |
| firstBloodKill    boolean |
| firstTowerAssist    boolean |
| firstTowerKill    boolean |
| goldEarned    int |
| inhibitorTakedowns int |
| nexusTakedowns    int |
| objectivesStolen    int |
| pentaKills    int |
| quadraKills    int |
| totalDamageDealtToChampions    int |
| totalDamageShieldedOnTeammates    int |
| totalDamageTaken    int |
| totalHeal    int |
| totalHealsOnTeammates    int |
| totalMinionsKilled    int |
| timeCCingOthers    int |
| totalTimeSpentDead    int |
| tripleKills    int |
| visionScore    int |
| win boolean |
| teamPosition string Basically the lane |
| gameDuration long game length in milliseconds |

Match 28th 2023:

Finally have a break from midterms to work on this project. Managed to go to office hours today to talk to Prof. Josiah Hanna to see if he could give me some guidance.

Advice he gave was very fruitful:

* He suggested that flattening would not mess up my data as this is something the neural network could learn.
* He suggested that it would not be necessary to take the ratios of the features between players, a neural network could learn that.
* The role of the player can be seen as a feature, and we convert it into a one hot vector so that each role is equidistance.
* Suggested that there could be existing models for ranking using neural networks, or I could try using regression and use MSE as the loss function.
* Suggested that I could use regularization (e.g. L2)

My plan here would be to try to use regression and MSE as the loss function first, then see as an extension I could compare it to an existing neural network models that are made for ranking. Currently worried about the lack of data, as of right now we have 29 data points only, compared to neural network training data sets such as MNIST.

April 4th, 2023:

Currently have 46 data points => start creating a program to retrieve the game data from the API. Also considered that maybe a player’s performance could be related to the player on the enemy team with the same role. However, this means the feature size would practically be double, and we would probably need a lot more data points to see it converge. I should go back to this when I have more data, would be interesting to look at. For now, I will stick with what we have previously and build a model with it.

April 6th, 2023:

Built the dataloader first following the tutorial, split the csv file into 2 parts, 80% trainset and 20% test set.

<https://pytorch.org/tutorials/beginner/basics/data_tutorial.html#creating-a-custom-dataset-for-your-files>

Built the model with a flatten layer, followed by2 hidden linear layers with relu activation and softmax for output linear layer. Used MSELoss for criterion (loss function) and SGD as optimizer. Learning rate was set to 0.0001.

Text

Description automatically generated

Results were not good. Accuracy stayed the same from the first episode. Perhaps the model is not complex enough? Have tried making the learning rate greater, but the accuracy still does not change.