Rocket League Rotation Analysis

Abstract

The growing excitement around machine learning brings many new opportunities for performing mathematical and statistical analysis on a large variety of datasets. This study explores the potential of machine learning techniques and models in analyzing a massive positional data set obtained from the pro player FirstKiller's Rocket League games, renowned for his exceptional game awareness. The goal of this paper is to demonstrate that positions can be predicted on each independent axis (x, y, z), however, the results demonstrate that it is quite a complex problem. Three separate algorithms to attempt at getting an accurate solution to the training data; firstly, closed form Polynomial Regression, second is gradient descent on polynomial regression and finally wrote code for a neural network that has 4 Dense layers with ReLU activations.

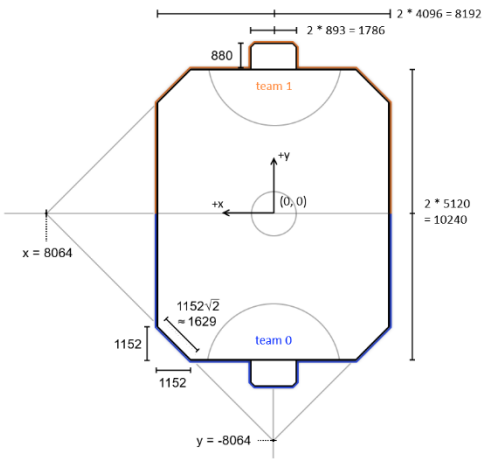
Introduction

To understand this project, Rocket League is a team game about playing soccer with cars. Competitive Rocket League consists of three players per team and one ball. Players drive around the pitch and try to hit the ball into the net much like normal soccer. There are many aspects such as boost collection, demolitions, bumping, and aerials (flying through the air) etc. Studying rotations and recommended positions is incredibly important in any type of sport, whether it be Football, Ice Hockey, Rugby, or online E-Sports such as Rocket League and many first-person shooters like CS-GO.

Upon research and investigation into this topic, it seems that this task has not been done, which makes for an interesting project. Everything will have to be original and thoroughly planned because there will be no reference material. However, there have been models employed to predict scoring chance, win chance during the game and others. Unfortunately, these previous projects did not assist with the development of this project. It should be noted that they do provide interesting insights and a great learning experience for players though.

Every machine learning project need data, so a dataset of ~64000 samples was gathered from replay data from professionally ranked games. RLCS is the highest competitive level of rocket league and because the data was gathered from these games the learned model should be trained at a "professional level" predicting desired positions. This data was gathered from the API available on ballchasing.com. The website developers provided excellent experience for gathering replays and data.

Some information on the data will also be needed to understand the errors and results. The diagram on the right is the measurements in uu (unreal units) which is what the data we are trying to predict is scaled in.



The research that was conducted includes various models that fit into two groups; polynomial regression which had two forms of analysis and a neural network. Each method tried to fit the original data, the normalized data, and the augmented features whose goal is to absorb the bias term and avoid calculating that too.

It should also be noted that because the positions on the field come in (x, y, z) tuples, the axis was split up for the sake of learning. This is doable because of their independence of one another, a relationship between X data is not necessary for predicting Y data.

Polynomial Regression

Closed Form

Closed form polynomial regression, otherwise known as least squares regression, fits a polynomial function to a set of data points to model the relationship between data. This form of data fitting can be computationally expensive and prone to overfitting which means that it generalizes poorly. However, it has great simplicity and is easy to implement.

Because the data is measured into thousands the mean squared error value is not necessarily relevant. This is because each error when squared could reach into the millions, more reasoning of this values lack of importance is mentioned in the gradient descent section. Instead, the focus will be shifted to the Root of the MSE and the Mean average error, which gives a better visualization on accuracy in terms of the rocket league pitch.

The following table it for the original input features

	X	Y	Z
RMSE	2341.85	2131.96	699.19
MAE	1887.92	1705.83	496.16

The following table it for the augmented input features

	X	Y	Z
RMSE	1124.39	1410.14	254.98
MAE	759.93	918.53	152.44

The data above was gathered over 10 trials and then averaged as the shuffling of the dataset prior to the closed form calculation provided unreasonably large errors that went over 7000, which in terms of x is about 2/3 of the pitch, this obviously was an unsuccessful trial but needed to be considered for this method regardless.

There is a clear increase in accuracy by augmenting the input features, this confirmed the intent for augmenting the input features. This is because augmenting can lead to reduced overfitting, and improved scalability for larger tasks.

Still, this error is quite high and would result in predictions with very poor accuracy from where the desired location is.

Gradient Descent

Gradient descent allows for fine tuning of parameters, it handles large datasets well. Another advantage is that because the values are quite large and given the huge dataset, the closed form solution may have overflow errors, causing incorrect calculations.

Gradient descent is also nice for noisy data, which is perfect for this instance because humans are never perfect, there are inconsistencies even if they are negligible for us, but it can cause the trained model to become unstable. Although the number of features per sample is only 3, the advantages of gradient descent may provide a better solution for this problem.

A note about the implementation; when doing the update with the negative gradient, the weights blew up to infinity and didn't converge, when updating with the positive gradient there was convergence and demonstrated similar values as the closed form solution.

The data below is the MAE for the test data.

The following table it for the original input features

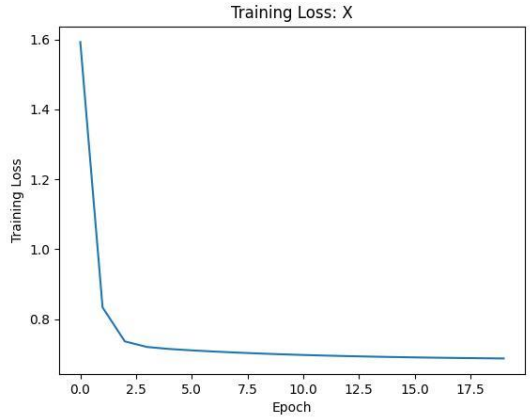
	X	Y	Z
MAE	750.25	972.1	140.98

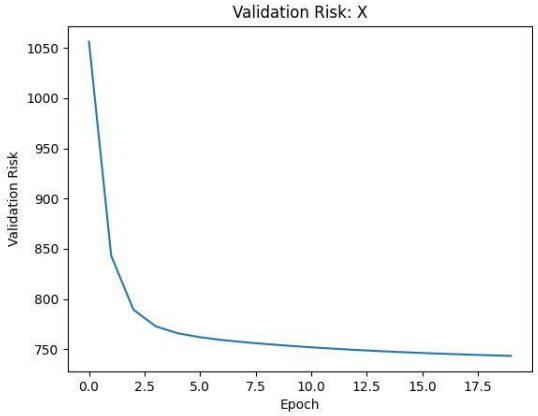
The following table it for the augmented input features

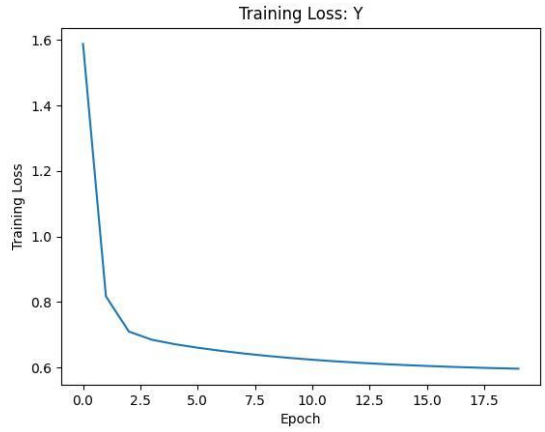
	X	Y	Z
MAE	750.25	972.1	140.98

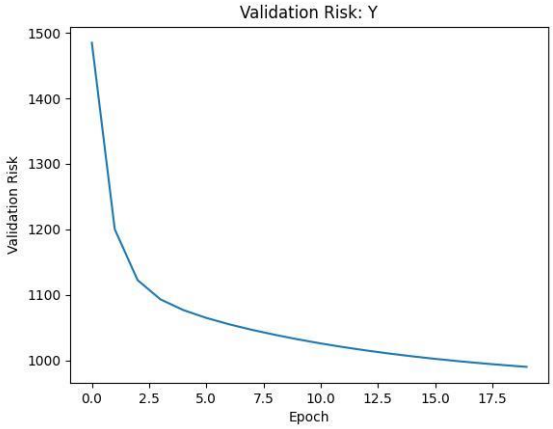
The results of this method were a bit strange, seeing that the results of the augmented data were identical to that of the original data. A possibility for this outcome may be due to the augmentation having no significance while doing the gradient weight updates with the gradient.

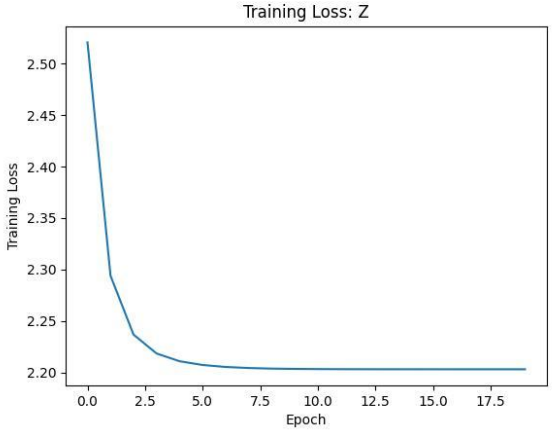
The following are the graphs of the progress over epochs during learning, on the left are the training losses, and on the right is the validation risk. The augmented data will be ignored as it had the same results.

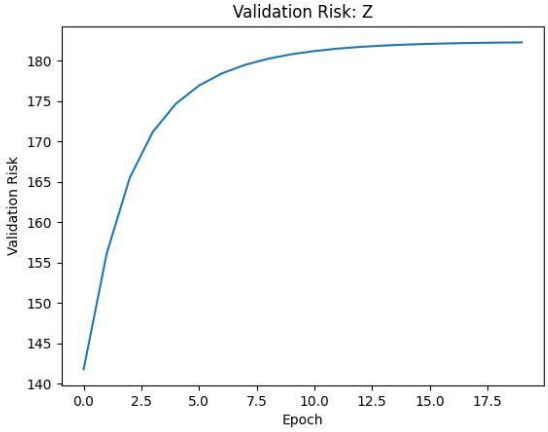












These results are what is expected from gradient descent, however, the validation risk for Z increased as the training progressed, while the training loss went down. This only happened some of the time when the experiment was run. An explanation for this phenomenon could not be determined as the validation risk for X and Y axis decreased similarly to the training loss.

It can also be noticed that the y axis for the training loss was very small, this is due to normalization of the data to curve wild convergence which was a problem that was encountered.

The normalization made the data much more workable as well as avoiding overflow errors. The validation was calculated with original untransformed data.

Multiple step size values, decay values and batch sizes were tested during training to find and improve on solutions. The results are using a step size of 0.001, such a low value was used due to the noisy data. Interestingly though, convergence was achieved by epoch 5 for the training loss, however the validation risk for Y and Z axis did not show convergence until much later.

Neural Network

The neural network was created using a layered network from tensorflow, there were 4 dense layers each having a ReLU activation unit, this activation was chosen because of its simplicity in large problems. Given that the data is quite noisy ReLU made a good choice. For the original input features the layers had 64, 16, 4, and 1 node respectively, and the augmented input features had 81, 27, 9 and 1 node respectively. These numbers were chosen because of the input shape of the data, although insignificant it seemed like an interesting choice. It should be noted that other amounts of nodes were also chosen including uniform ones across the board and little difference was noticed.

The following tables are the final RMSE losses after training

Original Input Features

	X	Y	Z
Training Loss	1113.24	1346.50	247.98
Validation Loss	1128.59	1315.46	245.71

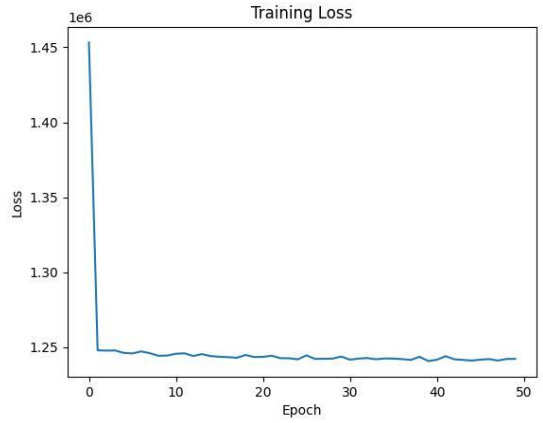
Augmented Input Features

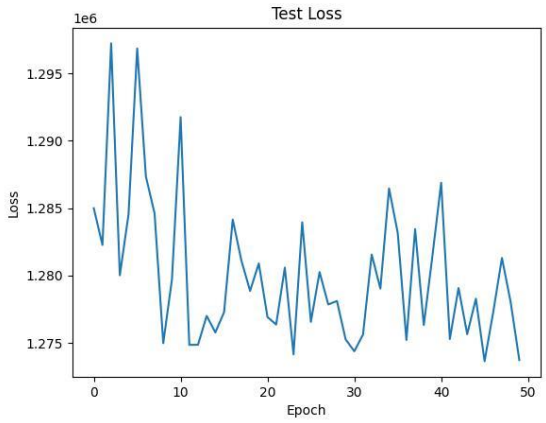
	X	Y	Z
Training Loss	1113.77	1347.42	247.99
Validation Loss	1129.90	1318.48	245.92

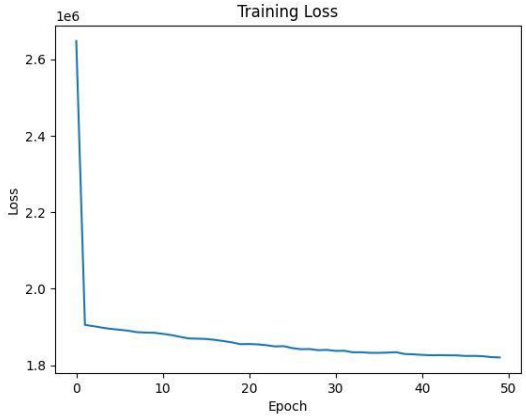
Again, the augmented data was incredibly like the original data. This can be due to it being insignificant for the type of neural network and learning method used. The gradient may have entirely ignored it.

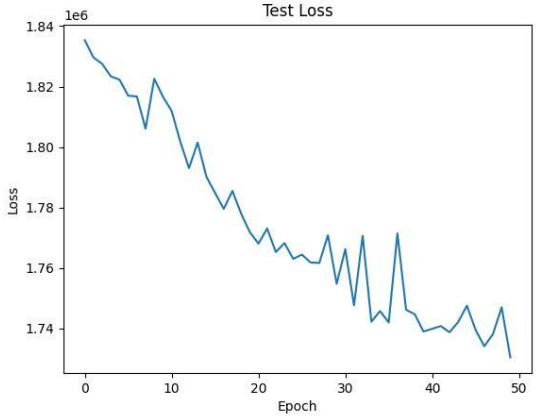
It can be noticed that the validation loss is almost equal to that of training loss which indicates that this solution generalized very well to new data.

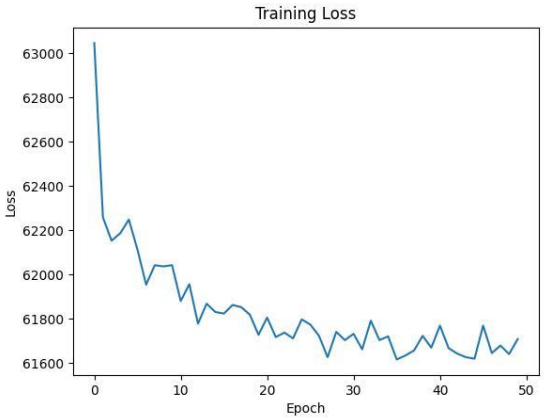
The following are the history graphs while training the model, on the left is the test loss and on the right is the training loss, in vertical order of the axis x, y, z. The graphs for the augmented data will again be ignored because of extreme similarities and lack of significance to this experiment.

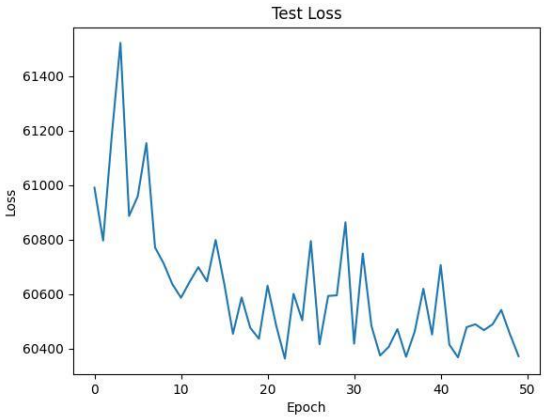












The model was not very stable as can be seen by the rapid increases and decreases of error during training. The rapidness may seem significant but upon inspection of the y axis the squared error only changes by about 0.005% of its value so it is not entirely significant. The training losses converged very quickly while the test losses did not converge so quickly, which is interesting.

Results

Concluding the final method, we can compare results for each of them. The closed form polynomial regression turned out to give us the best results and model, while gradient descent was not far behind, and the neural network finished last. This could be due to the relatively simple input features and the closed form solution being able to tackle these features well. This could also be due to lack of "best fitting" model parameters such as step size, batch size, epoch, decay, etc for the gradient descent and the neural network. Many attempts were made at finding the best parameters and the results above demonstrate the best ones that were found.

Unfortunately, though, the results were not quite as accurate as were planned to achieve for this dataset, this can be attributed to the human error during gameplay, as well as unique scenarios during the game that don't typically happen during other games. These inconsistencies create very noisy data and the only way to prune them from the dataset would be to watch all the replays of each game and remove the chunks of inconsistent gameplay. This, however, would create bias in the data and remove the widespread applicability.

Discussion

Overall, this experiment was a success for both my knowledge of the game of Rocket League and my knowledge of machine learning in general. It was incredibly interesting to be able to apply one of my passions to one of my favorite video games.

There are many expansions to this research including predicting positions for 2v2 and 3v3 games and even different game modes which have different playstyles themselves, this would bring on an entirely new set of challenges such as predicting 3 players positions all at the same time.

As stated in the results, it was unfortunate that accurate results were not obtained with the models, but it must be known that no on field rotation is down to a science and there is lots of randomness and stochasticity in ever game given the players movements, reactions, network delay and other unqiue gameplay scenarios.

There are also many other features to consider and predict during the game, one step up to this problem would be the direction vector, with a direction vector the player learning from this model would also be able to see the direction they would need to drive for the msot efficient gameplay. Including a direction vector a much more complicated model would consider boost levels, whether a boost pad is available on the field and playing for demolitions.

Conclusion

In conclusion, this research project explored the potential of machine learning techniques and models in analyzing a large dataset of positional data obtained from the professional player FirstKiller's Rocket League games. The objective was to predict positions on each independent axis (x, y, z) of the Rocket League pitch. Three separate algorithms, including closed-form polynomial regression, gradient descent on polynomial regression, and a neural network, were used to obtain accurate solutions to the training data. The results showed that while the prediction of positions is possible, it is a complex problem, and accuracy remains an issue. Augmenting the input features led to a clear increase in accuracy only for closed form polynomial regression. Future research could focus on improving the accuracy of the predictions and exploring expansions on the input features. Overall, this project contributes to the growing excitement around machine learning and its potential applications in the sports industry, particularly in the field of esports.

References

https://ballchasing.com/doc/api

https://github.com/SaltieRL/carball

https://alpscode.com/blog/rocket-league-and-sports-analytics/

https://joeydotcomputer.substack.com/p/neuralnextg-v010-analyzing-rocket