Tyler Gourley
16 April 2025

ECEN 471 Project 4
Sanguine Synergy: Modeling Team Chemistry in Soccer

1 Introduction and Problem Formulation

1.1 Background

In soccer, success on the field is not solely determined by the individual talent of players but also by their ability to function as a cohesive unit. The concept of **team chemistry**—the interplay of familiarity, experience, and tactical synchronization among players—remains an elusive yet critical factor in predicting team performance. While traditional analytics focus on player statistics, formations, and tactical strategies, the impact of team synergy has been largely unquantifiable… until now.

This paper seeks to bridge that gap by developing a machine learning model that captures the impact of team chemistry on match outcomes. Using a dataset of playing time distributions, we construct a framework to analyze how different lineups influence goal differential (a fundamental metric of success). We will aim to solve this **regression problem** by employing techniques such as linear regression, polynomial regression, and regularization. In doing so, we explore how team chemistry among players contributes to the ultimate goal of every soccer team, coach, manager and player: winning matches.

1.2 Impact

The impact of a highly accurate model describing the effect of team chemistry would alter the landscape of soccer forever, especially if that model indicated that strong team chemistry leads to an increased probability for success. Unfortunately, using the preliminary methods that are described in this paper, we will not be able to say for certain how much impact team chemistry truly has on winning games. The goal of this paper—however—is to create a model that can accurately predict goal differential based on a team's previous playing experience together.

Even so, soccer coaches could use the trained model created in this paper to get a preliminary estimate as to their team's expected performance (against any random opponent) based solely on their current team synergy.

1.3 Problem Structure

The inputs to the machine learning models are 55 feature vectors where each element is quantified by the number of games two players have played together in their professional careers. Given there are 11 starters on a soccer team and are concerned about two-player combinations, this size of the input feature vector can be verified easily: ${}_{11}C_2 = 55$.

The output of the model is goal differential—represented with a single integer. That is to be calculated as the difference of goals scored and goals allowed. For this paper, the metric for goal differential will only include goals that are scored during the 90-minute period (including extra time). It will NOT include goals scored from shoot-outs or any other tie-breaking procedure. So, a match that comes down to shoot-out will always have a goal differential of 0 regardless of if a team emerges victorious due to scoring more goals in the shoot-out.

I expect that teams with high team chemistry will be more likely to perform better. Because I believe there is a strong relationship between team chemistry and team success, I am confident a model can be made that drastically outperforms a simple baseline model.

1.4 Additional Depth

This project expands upon Projects 1, 2, and 3. In order to go above and beyond, I will be narrowing the dataset that I am training on to only include matches that were played in the top 5 leagues: Premier League (England), Bundesliga (Germany), La Liga (Spain), Serie A (Italy), and Ligue 1 (France). By narrowing my focus on only the top 5 leagues, the size of my dataset decreases to 5,863 datapoints (down from 55,780 that I used in Project 3).

I expect that neural networks will drastically outperform the models I have created for previous projects. However, I am substantially reducing the size of my dataset to exclusively analyze the soccer teams that play at the highest level. This reduced size of the training dataset may mean that there will not be enough training data to create a robust neural network.

Up to this point, the best model that has been the most accurate was a Decision Tree and kNN. I expect a neural network will produce a more accurate model because in theory, neural networks can accurately model an arbitrary function, so long as it is continuous. It is still my belief that there is a connection between team chemistry and performance, so a neural network should have good success modeling that relationship.

2 Methods and Experiments

2.1 Setup

2.1.1 Source of the Dataset

The data used in this project comes from an SQLite database that was assembled in Dr. Harrison's research lab that contains over 20 years of playing time data. Originally, the data was scraped from the ESPN website. In my responsibilities as a researcher, I was the one to compile the raw playing time data into the .csv files that will be used as the dataset for this project. I have included the dataset .csv files in the .zip file in my code submission (they can also be found at this link). We will be only analyzing matches played from 2008 and 2009 in the top 5 leagues in the world (for more information see Section 1.4), which contains 5,863 data points.

2.1.2 Test/Train Split

First, we separated the data into a training dataset and test dataset. The nature of the dataset is that each match has 2 inputs associated with it (one for each team). So, in order to avoid data leakage into the test set, the data was first grouped by match before performing a test/train split. We do this by using the GroupShuffleSplit() method splitting on match id (see documentation for sklearn.model_selection.GroupShuffleSplit). By stratifying the random split by match id, we are able to ensure that inputs from the same match are both found in the same set (either both in the test set or both in the training set). We used a 90/10 split—90% for training data and 10% for test data.

2.1.3 Estimating Enew

We will use the holdout validation method to estimate the error on data the model has never seen before (E_{new}). This decision was made to decrease the computation time of estimating E_{new}—holdout validation is less computationally expensive than cross-validation. Because we have a large dataset, we expect this method will provide accurate estimates for E_{new}.

We set the random seed to be 999 for all functions that use randomness.

2.1.4 Baseline Model

We will use Scikit-learn Dummy Regressor (from the sklearn.dummy.DummyRegressor package) as our baseline model to which we will compare the machine learning models we create. The strategy of the baseline model was set to always estimate the statistical mean of the training set. This was accomplished by setting the strategy parameter to 'mean' (strategy='mean'). All other parameters were left to the default values.

Using the holdout validation strategy on the baseline model, we get $E_{new} \approx 3.3030$ for MSE.

Finally, we trained the baseline model on the full dataset (comprising of both the training set and validation set, but not including the test set) to use as a reference to which we can compare the other models we create.

2.2 Train Model Without Dropout / Regularization

2.2.1 Set Up the Data

Before we begin setting up the model, we must allocate the training data into 2 groups: training and validation. When creating the holdout validation set, we stratified the data by match id in order to prevent data leakage (see Section 2.1.2). We decided to use 20% of the *original* dataset as the validation set. Because we had already split the data into train and test, we were sure to take that into account and split the data accordingly. We normalized the train/validation split to the 20% / 90% ≈ 0.22 (~22.2% validation and ~77.8% training). This left the final allocation of the data as 70% training data, 20% validation data, and 10% test data.

We chose our initial batch size to be 32, as it is the default for the keras neural network architecture. It should work well with our input data, and we will adjust it when we perform the hyperparameter sweep.

2.2.2 Set Up the Model

The first step is to create a simple neural network using keras. I chose to implement my model using keras-core, which more easily allowed me to implement JAX on the backend to accelerate the training process.

The nature of our output data (goal differential) is numerical. Thus, we will be using regression models. We will use mean squared error (MSE) as our loss function. We chose MSE because it is a relatively easy metric to understand the error of our model.

We used the Adam optimization algorithm. And set the learning rate to be 0.001. These choices were made because Adam is a well-optimized algorithm that helps loss to converge quickly when training neural networks. For clarity, the hyperparameters for the initial neural net are listed below (all other parameters were left to the defaults):

width=64 depth=3 activation="relu" learning_rate=1e-3

loss="mse" optimizer="adam"

2.2.3 Train and Compute Validation Loss

Now we can train the test model that we created in Section 2.2.2. We used the Jax backend to speed up the training process. The model was trained in 50 epochs, using a batch size of 32.

The figure below shows the loss from the training and validation sets throughout the training process. The validation loss seems to plateau after very few training epochs.

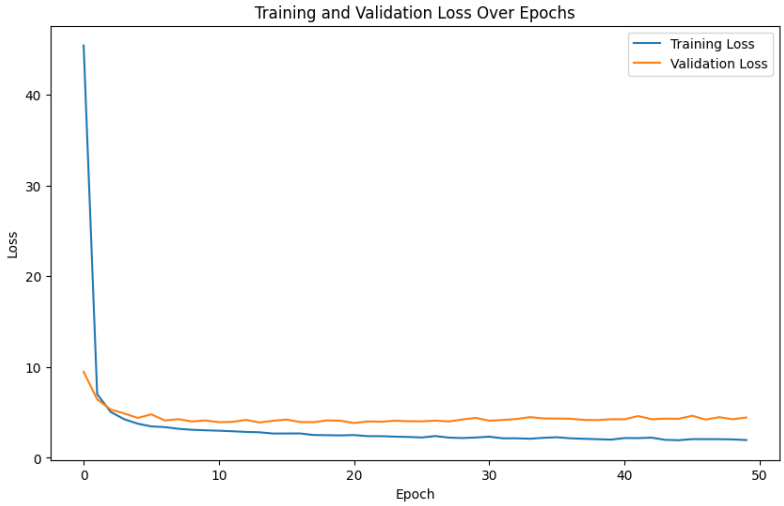

Figure 2.2.3A – Training and Validation loss (MSE) vs. Epochs

2.2.4 Hyperparameter Sweep and Final Model

Now we will sweep over many hyperparameters to determine the best model we can create (without dropout or regularization). Below are the values that we swept over for each hyperparameter:

depth: [2, 4, 8]

width: [64, 128]

batch_size: [32, 64]

epochs: [50, 100]

learning_rate: [1e-2, 1e-3]

activation: ["relu", "sigmoid"]

The optimizer and loss function were left constant in this hyperparameter sweep (Adam and MSE, respectively). This parameter grid had 96 combinations, each resulting in a trained neural network. The table below shows the parameters that produced the lowest estimated E_{new}.

Loss (MSE)	Depth	Width	Batch Size	Epochs	Learning Rate	Activation
3.231249	4	128	32	50	0.01	ReLU
3.232436	8	64	64	50	0.01	ReLU
3.250887	4	64	32	50	0.01	sigmoid
3.254878	4	64	32	100	0.01	sigmoid
3.256702	4	128	32	50	0.01	sigmoid

Table 2.2.4A – Top 5 best performing models from hyperparameter sweep

I found it interesting that all the highest performing models had—of the options tested—the lowest learning rate. Also, most of them were better with less training time (50 epochs instead of 100). That indicates that many of the models that were trained for 100 epochs probably must have overfit to the training data.

Finally, we saved the model that had the best performance. This will be used on the test set to evaluate its performance.

2.3 Train Model with Dropout + Regularization

2.3.1 Set Up the Data

We used the same train/validation split as before (for more details see Section 2.2.1). The final allocation of the data is 70% training data, 20% validation data, and 10% test data.

2.3.2 Set Up the Model

Now we will create a simple neural network that includes dropout and regularization. Again, we will use MSE for the loss function.

For this initial model, we will use the hyperparameters from the best performing model (from our hyperparameter sweep in Section 2.2.4). In addition to these 'best' parameters, we also set dropout to be 0.2 and implemented L2 regularization with a rate of 0.001. For convenience, the hyperparameters for the initial neural net are listed below (all other parameters were left to the defaults):

width=128 depth=4 activation="relu" learning_rate=1e-2 loss="mse"

optimizer="adam" dropout_rate=0.2 regularization_name="l2" reg_rate=0.001

2.3.3 Train and Compute Validation Loss

Now we will train the model (created directly above in Section 2.3.2) and compute its loss on the validation set. The neural net was trained in 50 epochs, using a batch size of 32.

The figure below shows the loss from the training and validation sets throughout the training process. Unfortunately, this graph does not give us too much valuable information about the training process, because it is so zoomed out. From this graph, it seems that our validation loss converges very early on in the course of training.

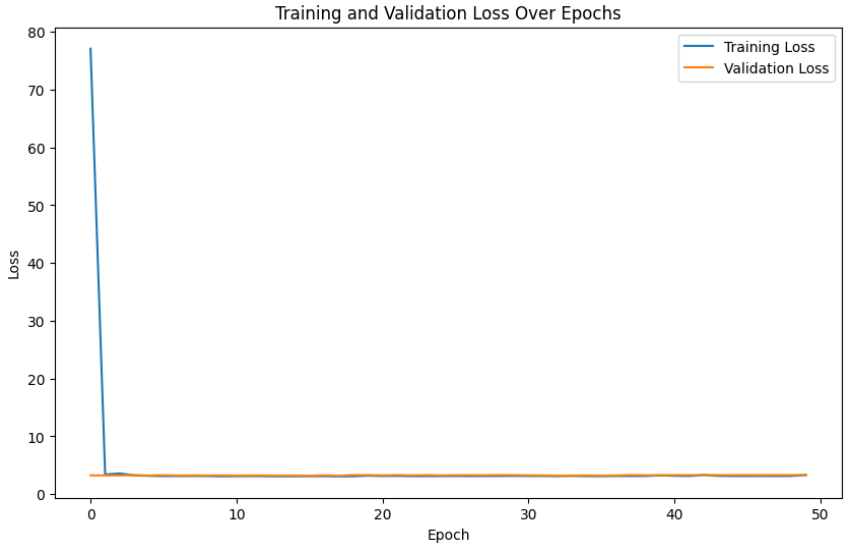

Figure 2.3.3A – Neural Network with Dropout and Regularization, Loss vs. Epochs

2.3.4 Hyperparameter Sweep and Final Model

Now we will perform a hyperparameter sweep to determine the best model we can create with dropout and regularization. In order to reduce training time while still producing a good model, we will use a heuristic of using the hyperparameters of the best model created so far (as shown in the first row of *Table 2.2.4A*). The optimizer and loss function were left constant in this hyperparameter sweep (Adam and MSE, respectively). Below are the values that we swept over for each hyperparameter:

dropout_rate: [0.0, 0.1, 0.2, 0.3, 0.4, 0.5]

regularization_type: ["l1", "l2"]

regularization_rate: [0.01, 0.001, 0.0001]

This parameter grid had 36 combinations, each resulting in a trained neural network. The table below shows the parameters that produced the lowest error on the validation data.

Loss (MSE)	Dropout Rate	Regularization Name	Regularization Rate
3.228904	0.1	L2	0.001
3.237939	0.1	L1	0.001
3.267942	0.3	L1	0.01
3.270755	0.0	L2	0.0001
3.272471	0.1	L2	0.01

Table 2.3.4A – Top 5 best performing models from hyperparameter sweep with Regularization

Finally, we saved the model that had the best performance. This will be used on the test set to evaluate its performance.

2.4 Final Testing and Comparison

2.4.1 Test all models on Test dataset

For the three final models we have trained, we used the test dataset to evaluate each model's overall performance. Here are the results:

Model	E_{test} (MSE)
Baseline Model (Dummy Regressor)	3.07521
Neural Network without Dropout/Regularization	3.0846
Neural Network with Dropout/Regularization	3.0719

Table 2.4.1 – Table of final performances (E_{test}) for each of the final models

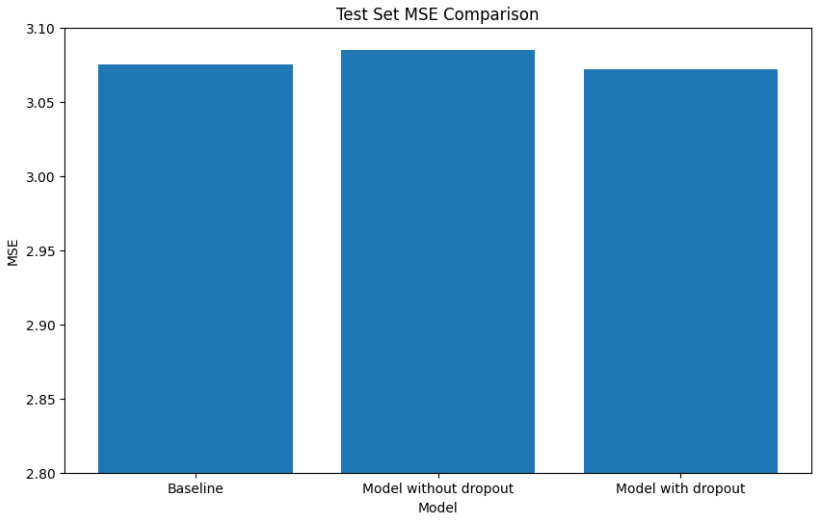

Figure 2.4.2 – Plot of E_{test} for each of the final models

3 Discussion and Conclusion

3.1 Analyze Performance Metrics

We can see that the neural network with dropout and regularization produced the best model (with the lowest error on the test dataset). However, we see that this model only slightly improved upon the Baseline model, and the model without dropout did worse than our baseline. Frankly, this is a disappointing result to see less than a 1% decrease in error compared to a model that always guesses the numerical mean of the dataset.

3.2 Discussion and Final Conclusion

Unfortunately, I do not have much confidence in any of the models that were created in this project. After four projects studying the effects of team chemistry, I believe it is safe to say that team chemistry is not a strong predictor of success in soccer. None of the machine learning models that I produced from in these projects were able to significantly outperform a “predict 0” model. Because of the limitations on the models, I would not feel confident using any of these models in the real world.

Neural networks performed similarly in comparison to the models that were created from Projects 1, 2, and 3: poorly. This is NOT a fair comparison, however, because I am using a much smaller dataset in this project (about 10% of the size I used for Project 3). This is because I chose to choose only to use match data from the top 5 leagues. But some sense, this result can be expected—or at least explained. At the highest level of soccer, there is more parity in player skill, and fewer lopsided final scores. We never expected team chemistry to be the most significant predictor of success in soccer—it is by all accounts a combination of player skill and coaching expertise. Also, the decreased sample size of the dataset makes our model even less accurate.

Still, I want to continue analyzing this dataset using even more advanced algorithms. In the future, I look forward to using Convolutional Neural Networks (CNNs) and Clustering Algorithms to experiment with more complex models. I would also consider changing the problem type from regression to classification, using WIN, LOSS, and TIE, as the output classes.