Using One Dimensional Convolutional Neural Networks to Predict Sports Outcomes

Logan Noonan

Fall Semester 2021

Abstract

Predicting the outcome of a sports match before it has even begun seems nearly impossible, yet millions of people every year make predictions like this, and with high stakes. This project aims to implement a 1D-Convolutional neural network (1D-CNN) to predict the outcome of NFL games. Match data from nearly every game spanning 10 seasons, 2009 through 2018, was collected to implement the deep learning model. After cleaning and pre-processing the data it was split into training, validation, and testing sets to verify it could generalize to new data. Our deep learning model was built using Tensorflow, a Python deep learning library. The model was able to achieve 60% accuracy in testing.

Contents

List of Figures

1 Introduction

The National Football League (NFL) is a well-known sports organization capable of attracting tens of thousands of fans to their games. Hundreds of thousands, even millions view the games remotely. For this reason the play by play details and outcomes have become a subject of interest for not only fans, but also for those looking to beat the bookie.

Convolutional neural networks (CNN) are one of the more popular deep learning networks and is often an entry point into the field. A CNN can process data of various dimensions, and is commonly used to handle grey-scale images (2D) as well as color images (3D). A one-dimensional (1D) CNN is less common but capable of processing sequences or signals. This type of CNN has been used to predict crop yields and air speed, among other things.

With the relatively small amount of research into this topic, this paper is intended to establish a 1D-CNN prediction model for predicting the outcomes of NFL matches by selecting telling features. If successful the algorithm produced could be useful in a variety of applications. These could include sports organizations using it as a means to predict future performance, to bookies and gamblers trying to make their next buck.

2 Related Work

Nowadays, with the wide spread development of analytical tools, predicting the outcome of sports competitions using artificial intelligence has become mainstream. Many machine learning methods have been used to predict the outcome of sports matches including logistic regression (Sblendorio), k-nearest neighbors, decision trees, and even linear models (Ukritw), among others. All of the mentioned models have one thing in common, that is that they are all machine learning models. Fewer models utilizing deep learning have been developed for the specific task of predicting sports outcomes.

3 Methodology

The framework for implementing the 1D CNN can be summarized in 8 steps, as illustrated in Figure 1. First, the game data of all 32 teams was collected and analyzed. Next, I performed feature selection as many of the data attributes were redundant or otherwise useless. Then, through feature engineering, I developed several supplementary features. The data was then normalized and split into training, testing, and

validation sets. I then built the model and proceeded to train and evaluate the model using the respective sub-sets of data. We can now discuss some of these steps in further detail.

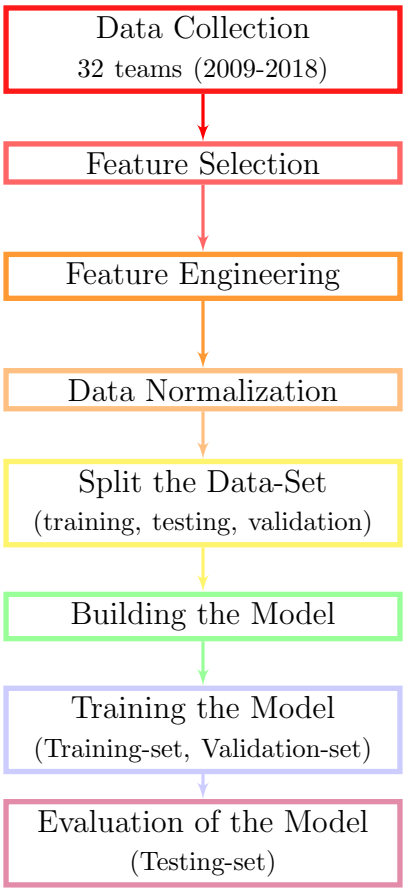

Figure 1: *Model Methodology*

3.1 Feature Selection and Engineering

There are two main reasons to perform feature selection. The first is to prevent over-fitting of the model. Over-fitting occurs when the model becomes exceedingly accurate in classifying its training data but then cannot generalize to test data. Feature selection is also performed to reduce the amount of data and complexity of the model, allowing quick training (although this is generally not an issue with a 1D-CNN model).

As previously mentioned the data-set contained many redundant attributes as well as others that simply did not offer any value to my model. There were 255 columns in the original data-set. Large chunks of these unnecessary columns we removed at once (if they were all adjacent on the index) otherwise the columns would be removed one at a time. An example of this process is shown in figure 2.

```
startIndex = dfTrimmed.columns.get_loc('shotgun')
stopIndex = dfTrimmed.columns.get_loc('total_home_score')
dfTrimmed.drop(dfTrimmed.iloc[:, startIndex:stopIndex], axis=1, inplace = True)
```

Figure 2: *Feature Selection Process*

The original data-set contained several entries per game i.e., it was built using play by play data. A more useful format for my model would be a single entry per game containing attributes for the two teams playing in the game. To implement this idea I indexed the initial data frame, extracted the relevant values, then stored them in lists which were then made into columns of a new data frame. An example of this process is shown in Figure 3. An example of creating an engineered feature is shown in Figure 4. The new column values are calculated via a function w, which produces a numeric value that is the ratio of the number of games the home team won in the previous year with respect to the number of games the away team won in that same year. The equation can be expressed as follows:

$$home\_win\_stat = \frac{w(ht, y-1)}{w(at, y-1)}$$

where ht = the current home team, at = the current away team, and y = the current year.

```
# The attribute for each of the following 6 columns of data (for the first
# instance of each unique game id) will be copied to the new data frame.
# Home and away scores are set to the maximum value of each column respectively.
ids = newdf.game_id
homeTeam = list() # Current home team
awayTeam = list() # Current away team
posessionTeam = list() # Current team in possesion
defensiveTeam = list() # Current defensive team
homeScore = list() # Final score of home team
awayScore = list() # Final score of away team

for i in ids:
    homeTeam.append(dfTrimmed[dfTrimmed.game_id == i].reset_index().
 ↪home_team[0])
    awayTeam.append(dfTrimmed[dfTrimmed.game_id == i].reset_index().
 ↪away_team[0])
    posessionTeam.append(dfTrimmed[dfTrimmed.game_id == i].reset_index().
 ↪posteam[0])
    defensiveTeam.append(dfTrimmed[dfTrimmed.game_id == i].reset_index().
 ↪defteam[0])
    homeScore.append(max(dfTrimmed[dfTrimmed.game_id == i].total_home_score))
    awayScore.append(max(dfTrimmed[dfTrimmed.game_id == i].total_away_score))

newdf.home_team = homeTeam
newdf.away_team = awayTeam
newdf.posteam = posessionTeam
newdf.defteam = defensiveTeam
newdf.total_home_score = homeScore
newdf.total_away_score = awayScore
```

Figure 3: *Reorganizing the Data*

```
# A list to track each teams wins by year.
total_wins_per_team_by_year = [dict((key, 0) for key in list(
                                        dic.keys())) for year in df.game_id.
 ↪unique()]
# Get the total wins per team by year, and fill in the list.
p = 0
for year in df.game_id.unique():
    temp_df = df[df.game_id==year]
    for i in temp_df.index:
        if temp_df.Home_win[i] == 1:
            total_wins_per_team_by_year[p][temp_df.home_team[i]] += 1
        else:
            total_wins_per_team_by_year[p][temp_df.away_team[i]] += 1
    p+=1
# Here I extract information to create the feature column
i = 0
p = -1
year = 2009
result = list()
while i in range(0, len(df.index)):
    if df.game_id[i] == 2009:
        result.append(None)
        i+=1
    elif df.game_id[i] == year:
        if total_wins_per_team_by_year[p][df.away_team[i]] != 0:
            result.append(total_wins_per_team_by_year[p][df.home_team[
                                    i]] / total_wins_per_team_by_year[p][df.
 ↪away_team[i]])
        else:
            result.append(total_wins_per_team_by_year[p][df.home_team[i]] * 1.5)
        i+=1
    else:
        p+=1
        year+=1
# "home_win_stat" is the feature column containing the data extracted above.
home_win_stat = pd.Series(data=result, dtype='float64', name='home_win_stat')
df = pd.concat([df, home_win_stat], axis=1)
```

Figure 4: *Feature Engineered Column "home_win_stat"*

3.2 Normalizing and Splitting the Data

The last steps before training the model are normalizing the data and sub-setting the data into training, validation, and testing sets. Normalization is performed to remove the relative variance between attributes, making them easier to compare. I normalized my data using the min-max method which maps all numeric variables to a new numeric value between 0 and 1 via the following equation:

$$X_{norm} = \frac{X_i - X_{min}}{X_{max} - X_{min}}, \quad i = 0, 1, 2, \cdots, \text{len(column)} - 1$$

where X_{min} and X_{max} are the minimum and maximum values of the current attribute.

It is vital to have some data set aside strictly for testing purposes. This data should never be seen by the model during the training phase. This is different than validation data which the model does see and use to update its trainable parameters. I accomplish this by passing portions of my data as input into three separate Tensorflow Data-sets as shown in Figure 5.

```
train_ds = tf.data.Dataset.from_tensor_slices((features[:1601], labels[:1601]))
test_ds = tf.data.Dataset.from_tensor_slices((features[1601:1847], labels[1601:
↪1847]))
validate_ds = tf.data.Dataset.from_tensor_slices((features[1847:], labels[1847:
↪]))
```

Figure 5: *Splitting Data into Tensorflow Data-sets*

3.3 Model Building

In just about any CNN there will be, atleast, the following three types of model layers. These are the Convolutional Layer, Pooling Layer, and Dense or Fully-connected Layer. Combine these layers in sequence and you will have a basic CNN model. Additionally, a drop-out percentage and activation function are important tune-able parameters for any CNN. Figure 6 shows my model summary.

	Layer (type)	Output Shape	Param #
0	InputLayer	[(64, 27)]	0
1	Reshape	(64, 27, 1)	0
2	Conv1D	(64, 27, 8)	32
3	MaxPooling1D	(64, 27, 8)	0
4	Conv1D	(64, 27, 16)	400
5	MaxPooling1D	(64, 27, 16)	0
6	Dropout	(64, 27, 16)	0
7	Flatten	(64, 432)	0
8	Dense	(64, 64)	27712
9	Dense	(64, 1)	65

Figure 6: *The Model Summary*

4 Visualizing the Data

With cleaned and processed data we can effectively visualize it. The correlation matrix in Figure 7 is useful in exploring how the predictor variables are related to the response variable. Figure 8 supports the assumption that the team who gains the most yards overall will generally win the game. Figure 9 suggests to us that if the away team doesn't have a recent win, then the home team is highly likely to win regardless of their number of recent wins. Similarly, if both the home and away teams have won all their recent games, the home team is still slightly more likely to win. This is what we would expect and define as home-field advantage. Figure 10 shows a somewhat unexpected trend. Third Down conversions are less important to a win when comparing home vs away, as the home teams have their greatest density of wins at a lower conversion rate than the away teams. Similarly, the away teams have their greatest density of wins at a lower conversion rate than the home team.

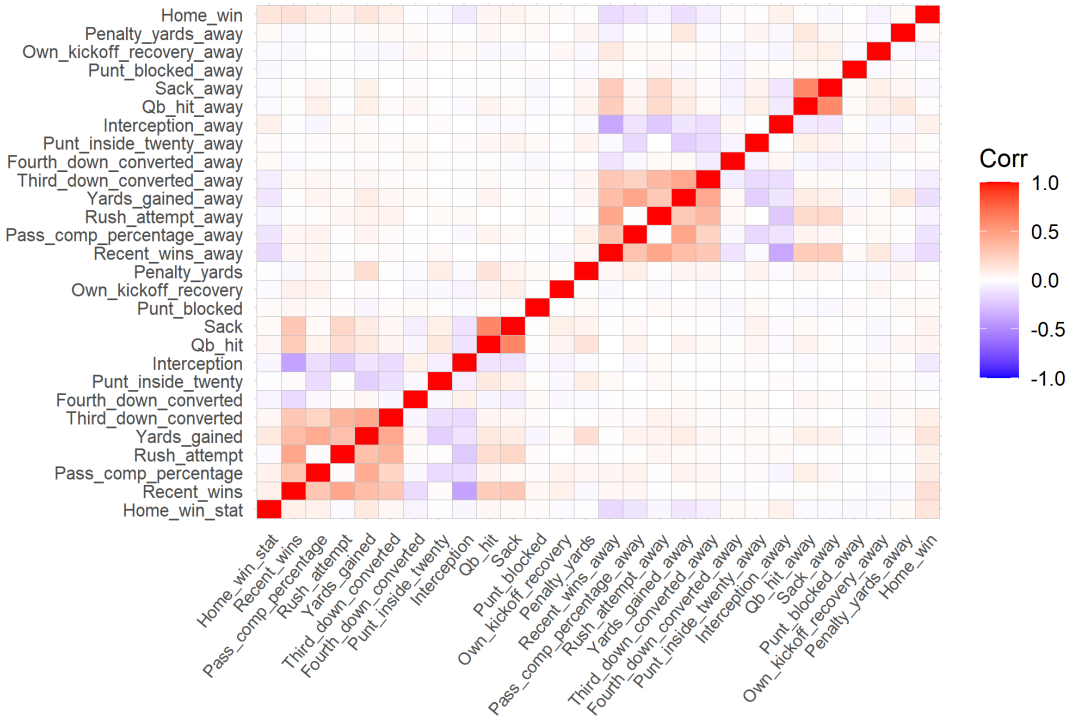

Figure 7: *Correlation Matrix of Model Attributes*

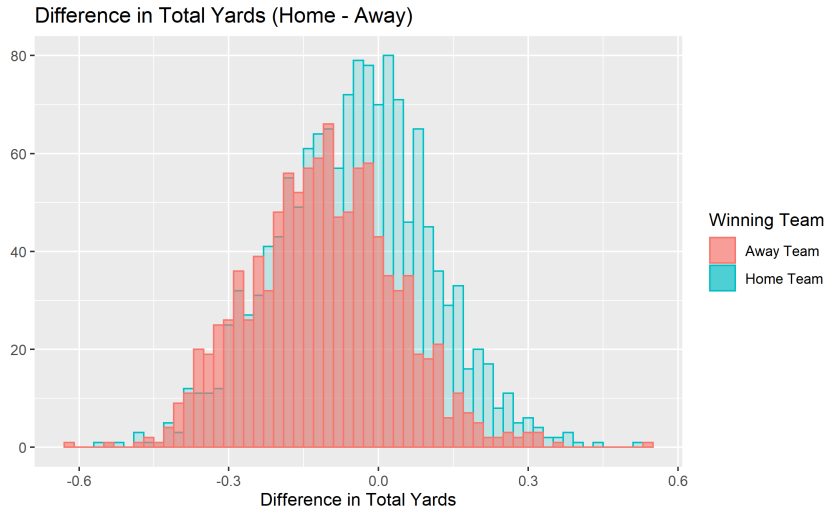

Figure 8: *Scaled Difference in Total Yards by Team*

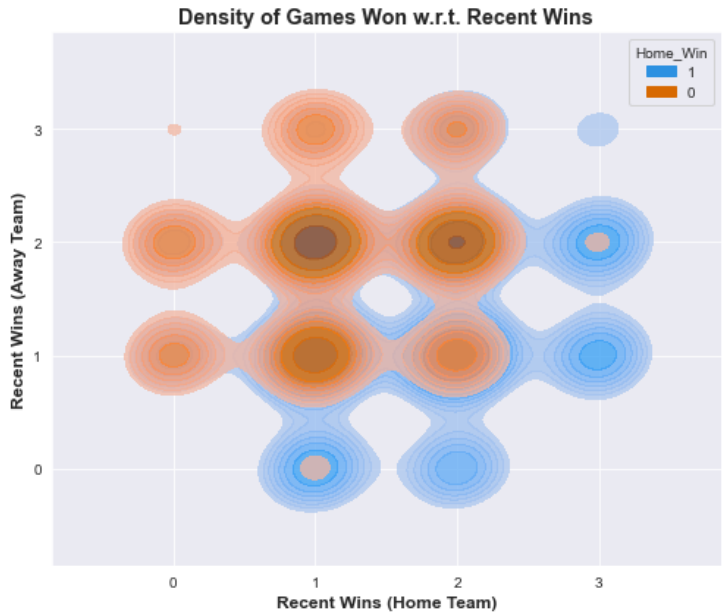

Figure 9: *A Comparison of the Relationship Between Recent Wins and Winning the Current Game*

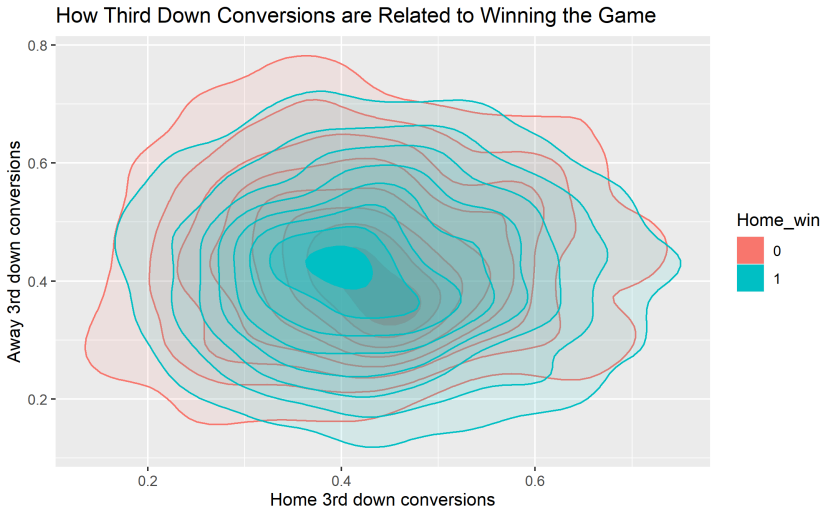

Figure 10: *Density of Away Team vs Home Team Recent 3rd-Down Conversions*

5 Experiment

The data-set applied in this paper, *NFL Play by Play 2009-2018 (v5)*, can be accessed at this link. Various model configurations were applied to the data-set. Parameters such as batch size and number of epochs were tested, also tested were varying input values for hyper-parameters including Convolution and Pooling kernel sizes, strides, and padding technique, Dropout rate, and Dense units.

The final version of the model was trained and validated over 1000 epochs in batches of size of 64. During training the model achieved an accuracy of 66.56%. The model was able to generalize well to the testing data achieving an accuracy of 60.07%. See figures 7 and 8.

	loss	acc
0	0.684607	0.600694

Figure 11: *Testing Loss and Accuracy*

	Real (1)	Real (0)
Predicted (1)	212.0	104.0
Predicted (0)	126.0	134.0

Figure 12: *Confusion Matrix Result From Model Testing*

6 Conclusion

The outcomes of NFL games were predicted by collecting 10 years worth of data on all 32 teams and using that data to train a 1D-CNN deep learning model. Various combinations of model parameters were tested and their resulting validation accuracy's compared. The final model's testing accuracy was just over 60%. According to ("Sports Betting Math - How To Win Money at Sports Betting") any sports betting winning record above 52.4% will earn a profit therefore, this model may be useful to sports enthusiast and team staffing alike. The amount of data used in this paper was relatively small so it would be reasonable to conclude that an even better testing accuracy's could be achieved with an a larger sample of data.

References

Sblendorio, Dante. "How to Predict NFL Winners with Python. Follow along the Steps." ActiveState, 3 Dec. 2020, https://www.activestate.com/blog/how-to-predict-nfl-winners-with-python/

ukritw. NFL Games Prediction Using Machine Learning. 2019. 2021. GitHub, https://github.com/ukritw/nflprediction/blob/e2fe0d27b8a2869259f3a9a4228adf2b3486e196/nflml.ipynb.

"Sports Betting Math - How To Win Money at Sports Betting." The Sports Geek, https://www.thesportsgeek.com/sports-betting/math/. Accessed 3 Dec. 2021.