Deep Learning on Volleyball Data

Weixuan Liu

June 8, 2024

Abstract

Deep learning has been gaining public attention in the recent years. Neural networks, the main components of deep learning, are well known for their great capability in capturing underlying pattern of complicated data and make predictions. We have messy data from volleyball games parsed from videos. Utilizing this data, we explored several deep learning architectures to see whether or not the models we built can learn to predict certain game states based on the available information. We tried to predict the total number of winning attacks of both the home team and away team using information of the match as input. We used multi-layer perceptron(MLP) first, and then recurrent neural network(RNN), and finally a MLP with different inputs and outputs. Our models did learn something from the data, but further improvement and analysis are needed.

1. Introduction

Deep learning is a method in artificial intelligence (AI) that teaches computers to process data in a way that is inspired by the human brain. Neural networks, the underlying technology in deep learning, have been demonstrated to have great power in recognizing complex patterns in pictures, text, sounds, and other data in the recent years. Therefore, they have been widely use in many computational scientific fields to help people analyze a huge amount of data, figure our the patterns and make predictions. Given the fact that volleyball data are highly noisy and complex, it would be a promising and exciting path to see what artificial neural networks (ANNs) would get from them. The primary objective is to predict the number of winning attacks for both home and away teams, leveraging advanced neural network models to offer more insightful and actionable data for coaches.

2. Data Description

2.1. Dataset

The dataset, provided by the UW Women Volleyball Coach, includes detailed records of all Pac-12 matches from the 2023 season, totaling 629 games. The data is stored in DataVolley files(.dvw), which contain 86 features and time steps ranging from 2100 to 3000. Each .dvw file contains a wealth of detailed information about the corresponding volleyball match. It includes names of the participating teams and players, playing positions of each player, detailed records of every action performed during the match, result of each action etc. In addition, the data in a .dvw file is primarily event-based, logging specific actions and events as they occur during the match rather than recording data at a fixed frequency. This method captures the key moments and outcomes relevant to the analysis of the game.

2.2. Data Visualization

Visualizations were created to map the coordinates of volleyball start and end zones for attacks(Figure 2.1) helping to identify patterns and areas of high activity on the court. These visual insights aid in understanding the spatial dynamics of the game.

The left plot in Figure 2.1 shows the distribution of the start and end positions. Clearly, the attack of the home team always starts from the top left side in the bottom half of the court and lands on the the top left side in the top half of the court. The players from the away home team intentionally do so, because many team members in the away team are left handed. When the ball appears in their right-hand side, they are more likely to miss the ball.

The right plot in Figure 2.1 pairs the end position with its corresponding start position, exhibiting more detailed information for each home-team attack.

3. Methodology

3.1. Problem Formulation

In our research, we mainly focusing on predicting the number of winning attacks - the attacks that successfully make the team that perform them win a point - for both home team and away team given data of some selected features that may implicitly affect the outcome, like skill and skill_type, across the time. There were 37 chosen features in total, where the features in the file that neither apparently have nothing to do with the winning attacks nor explicit encode it.

Therefore, the input was of the shape the total number of time points(2100-3000) times the number of selected features(37). The output is 2 dimensional, meaning two numbers are outputted for each input.

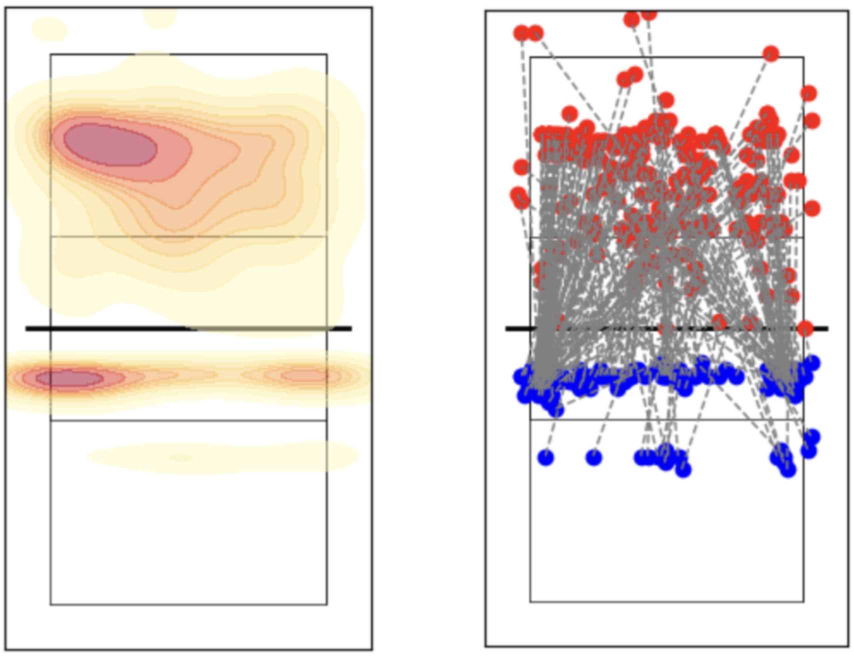

Figure 2.1: Heat Map(left) and link dots(right) about the start and end positions of the ball for each home team's attach. The top half of the volleyball court represents the positions of the away team, while the bottom half represents those of the home team.

3.2. Neural Network Models

3.2.1. Multilayer Perceptron (MLP)

There are many variants of neural networks, each of them designed for certain scenarios. MLP is the most simple and straightforward one. The fact that it is easy to implement and that is powerful to handle data makes it a good starting point.

The MLP analysis is a feed-forward machine learning method, which designs for predicting one or more output variables using input variables. The MLP model consists of three main components, i.e., (1) the input layer, (2) the hidden layer and (3) the output layer. Each layer is composed of neurons that are fully connected to the neurons in the two neighboring layers. In general, the MLP model could use interconnected layer of artificial neurons with the input of data to produce a set of output data. The loss is computed as a metrics to measure the difference between the output and the target(or the ground truth). Then, the MLP model could be trained through a back-propagation process, during which gradient descent technique is used to update the parameters so that the loss could decrease.

The equation of the hidden layer is described as follows:

$$Z_j = H(\sum_{i=1} W_{i,j}^G \times x_i + W_{0,j}^G) \tag{3.1}$$

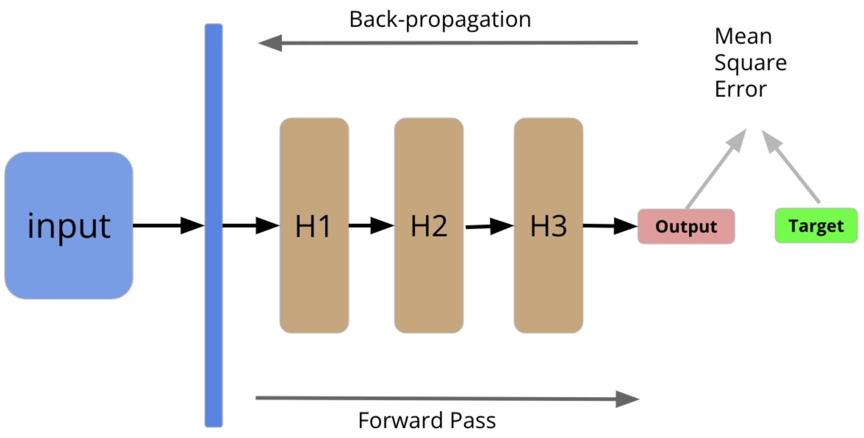

Figure 3.1: Our multi-layer perceptron(MLP) model. It first flattened the input into a long vector, and then forward pass it through 3 hidden layers. After we get the output from the 3rd hidden layer, we compute the mean square error of the output and the target, and then feed the error back to MLP and update the model's parameters

where $W^G_{i,j}$ is the weight of the neuron between the input and hidden layer and $W^G_{0,j}$ is an activation constant for neuron j. The activation function H is often non-linear. To get the activation level of a neuron, the weighted sum of neurons from the previous layer is calculated and then passed through a nonlinear function.

The architecture of our model is shown in Figure 3.1. We used 3 hidden layers, each of which contains 256 neurons. Before we pass the input to the hidden layers, we first need to flatten it into a long vector, due to the architecture of MLP model where one input should be a vector instead of a matrix. Mean Squared Error is used to compute the loss, which calculates the average of the squares of the differences between the output values by the neural network and the actual target values.

The model is trained on the training data through 100 epochs. Then we test the model on the testing data. The test loss is 135.08. The learning performance of the model during the training process is shown in Figure 3.2. Despite achieving some level of accuracy, the MLP's performance was limited due to its inability to fully leverage the sequential nature of the data.

3.2.2. Recurrent Neural Network (RNN)

To better handle the sequential data, RNN was used which was designed specifically for sequential inputs.

The core of an RNN consists of a cell that performs the following operations at each time

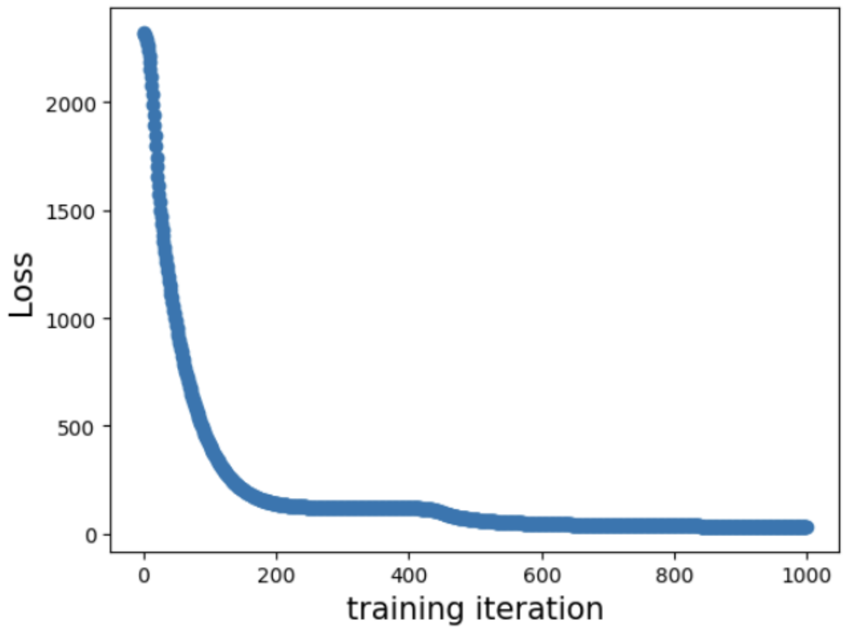

Figure 3.2: Learning performance of MLP. It shows how MLE loss changes over training epochs

step t:

$$\begin{aligned} h_t &= \sigma(W_h x_t + U_h h_{t-1} + b_h), \\ y_t &= W_y h_t + b_y \end{aligned} \tag{3.2}$$

where h_t is the hidden state at time step t, x_t is the input at time step t, h_{t-1} is the hidden state from the previous time step $t-1$, W_h, U_h are weight matrices for the input and hidden state, respectively, b_h is the bias term for the hidden state, σ is the activation function, y_t is the output at time step t,W_y is the weight matrix for the output, b_y is the bias term for the output. The recurrence nature of RNNs is encapsulated in the hidden state update equation $h_t = \sigma(W_h x_t + U_h h_{t-1} + b_h)$. This equation shows how the current hidden state h_t depends not only on the current input x_t but also on the previous hidden state h_{t-1}. By recursively applying this update across all time steps in the sequence, RNNs can capture temporal dependencies and long-range correlations within the data.

In this study, we built an RNN to predict the outcomes of volleyball attacks by modeling the sequential nature of the game events, demonstrating their capability to handle complex temporal dependencies in sports analytics. However, in order to adapt RNN to our problem where the output is not sequential, the output is calculated only at the last time step(Figure 3.3). We only used 1 hidden layer with the hidden size being 256. We still sticked with the MSE, but now we used Backprop Through Time, a technique that extends standard Backpropagation to handle the temporal dependencies in RNNs by unrolling the network through time.

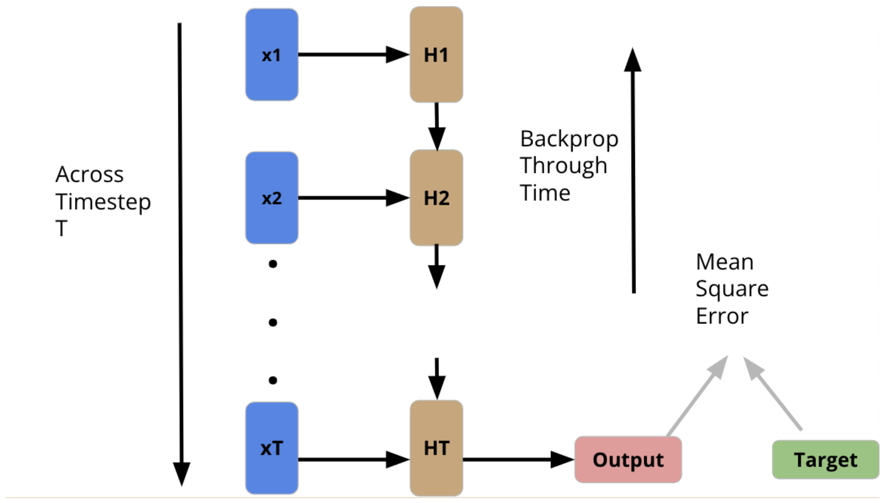

Figure 3.3: The plot shows our RNN architecture. The input is sequential while the output is not. We now use a Backprop Through Time instead of Backpropergation for the temporal dependencies in the RNN

We trained the model using training data. The learning performance is shown in Figure 3.4. Then we tested the model on testing data, where the loss is 143.23.

However, the model constantly gave us the same output regardless of the inputs. There are two potential reasons behind this learning failure. First, compared to the total number of data points we have (629), the sequence length (2100-3000) is too large. This discrepancy causes problems because the model must learn from a relatively small dataset, making it difficult to capture meaningful patterns across such long sequences. The limited number of data points means that the model has insufficient examples to generalize effectively, leading to poor performance.

Second, the long sequences increase the complexity of the model, exacerbating issues like vanishing gradients. The vanishing gradient problem occurs because during BPTT, gradients are propagated back through many time steps, and they tend to shrink exponentially due to repeated multiplication by weight matrices with small values. As a result, the gradients become extremely small, effectively becoming zero after a certain number of steps. This causes the network to stop learning from earlier time steps in the sequence. Consequently, the model fails to capture long-term dependencies and defaults to outputting constant values regardless of the input, as it cannot effectively update the weights associated with those long-range connections.

The first one is a common issue for all machine learning models, while the second one appears specially in the RNN due to its recurrence nature and in other variant of neural

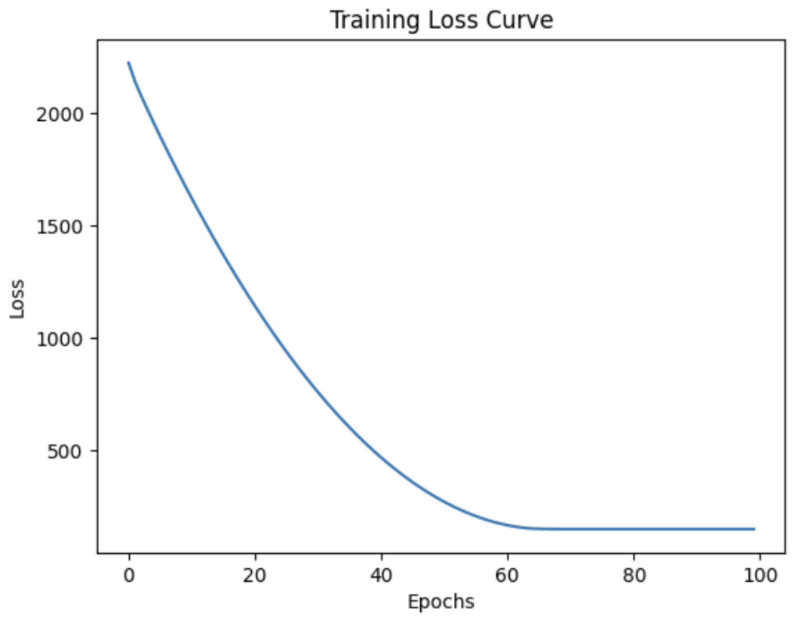

Figure 3.4: The plot is about the learning performance of our RNN model on training data. It shows how the loss changes over 100 epochs

network with lots of hidden layers. In this case, the first issue also happened in the MLP we built previously. To make the model having better performance, we managed to find a way to make the number of features less than the number of data we have.

3.3. New Approach

We noticed that the winning attacks occurs only when there's an attack happening. Therefore, we can only extract time points of attacks. Then we used them to predict whether each attack is a winning attack for a certain team, where 1 means "yes" and 0 means "no". Now the input is a vector having 37 entries, each corresponds to a feature. The output is a single number. We can extract 260 attack data points in a play, so the total number of data is 260, which is more than the number of features. The issue is solved

This method also converted the problem into a binary classification task, allowing us to use cross-entropy loss because the output is now a binary value (0 or 1), unlike the previous approach where the output was a continuous value necessitating the use of MSE. We chose cross-entropy loss because it is particularly powerful for binary classification tasks, as it effectively measures the performance of a classification model whose output is a probability value between 0 and 1.

Furthermore, since the input only contains attack information, its sequential property was broken. Therefore, we have to use MLP instead of RNN. The architecture of this MLP is

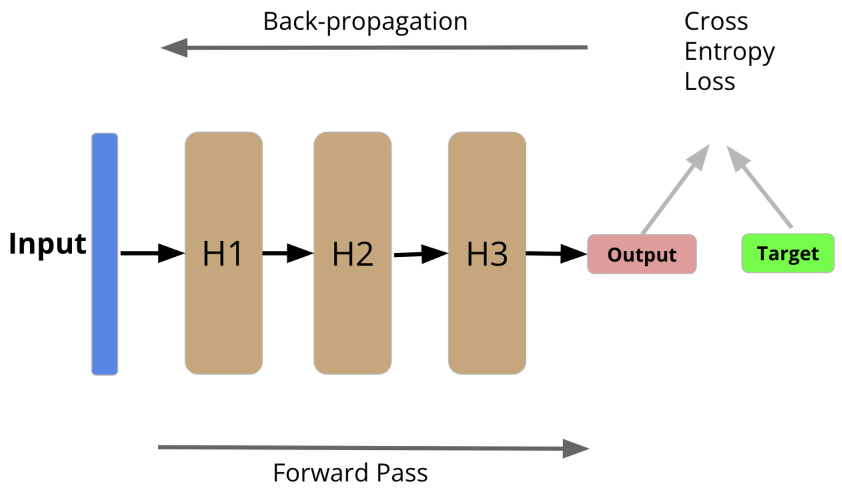

Figure 3.5: The plot shows the architecture of the new MLP. Flattening is excluded since the input is already a vector. The output is a binary number, so we replaced MSE with Cross Entropy Loss.

shown in Figure 3.5. We still used 3 hidden layers, each with hidden size being 256. The original input is already a vector, so we can directly feed it into the first hidden layer without flattening it.

We did train-test split on the data where 80% being training data and the remaining 20% being testing data. Then we standarized the data. We built up two MLP models. One model predicted whether the attack is a winning attack of the home team; the other one predicted whether the attack is a winning attack of the away team. We trained these two models using the training data. The learning performances of both models are shown in Figure 3.6. Then we tested them on the testing data. The test accuracy was 78.84% and 75%, respectively.

Then we gave all the attack data to the models. Two arrays for home team and away team respectively were outputted. The sum of entries of each array would be the predicted number of winning attacks for both teams. The MSE between the prediction and the ground truth was 36. Using this method, we successfully solved the previous issue and the model also achieved better performance.

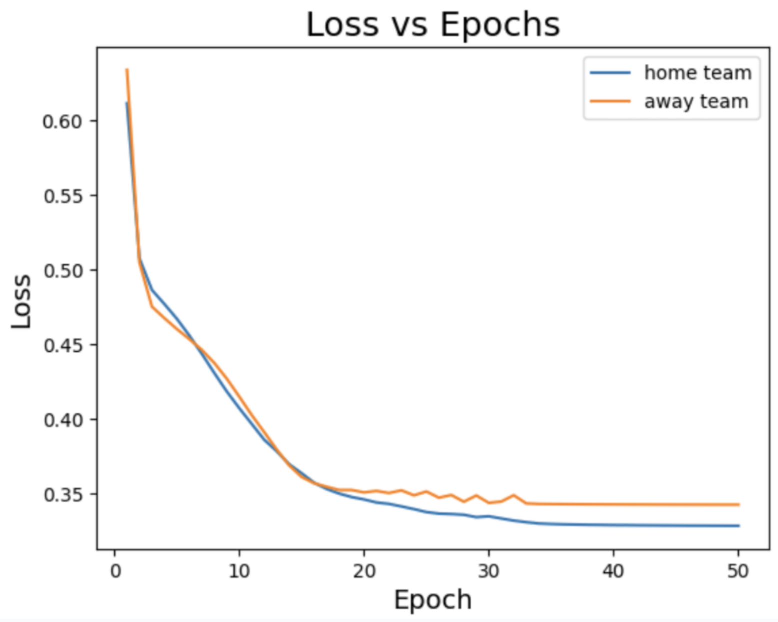

Figure 3.6: The plot shows the learning performance of two new MLP models trained on training data across 50 epochs. The final loss of both teams are very close while the loss of the home team is slightly lower than that of the away team

4. Discussion

4.1. Impact

The proposed models can be used by coaches to analyze match data and predict attack outcomes, helping them make informed strategic decisions. By processing post-game video data into .dvw files, coaches can extract sequences and predict expected winning attacks, comparing them to actual results to assess performance.

4.2. Future Work

Future research will focus on expanding the dataset beyond Pac-12 games, improving data preprocessing, and identifying key features for better coaching insights. Enhancing the computational power and refining the models will also be critical in advancing this study.

5. Conclusion

This study illustrates the potential of deep learning techniques in sports analytics, specifically for volleyball. By developing and refining neural network models, we have provided a foundation for more accurate predictions and actionable insights. Future work will continue to build on these findings, aiming to further enhance the applicability and accuracy of deep learning in sports.