**Using Artificial Neural Networks to Predict Men’s Professional Tennis Matches**

**Introduction**

As a long-time sports fan, I was extremely interested in applying the techniques we learned in this course to a sports dataset of some kind. In the past, I have participated in projects studying football, basketball, and soccer. However, for this project, I wanted to study a less explored sport analytically: tennis. In my experience, tennis usually utilizes advanced stats in television broadcasts in very specific ways; sometimes they point out key numbers that players need to hit to win, sometimes they tell the story of how a player develops his strategy throughout the match, and sometimes they just compare various data points for each player in each set. However, there is not too much academic work done on the sport, and I wanted to take a step to remedy that problem in my own project.

In order to complete my analysis, I utilized individual match data from all ATP-sponsored matches from 1991 to 2016. The ATP, or Association of Tennis Professionals, is the governing body of men’s professional tennis; they organize the tournaments, construct rankings, create competitive rules, and more. This dataset included 95359 matches, so it’s very comprehensive. Each observation details one match with winner and loser statistics such as aces, double faults, first serve stats, second serve stats, and more in each separate column. This dataset could be associated with separate datasets that gave more information on the players, date and time of match, location of the match, and other qualitative details that could be useful in the analysis. The main goal of my analysis was to classify each set of player statistics into a win or loss for each match to see how well a network could predict the results of a match based solely on one player’s statistics. To run the technique, I needed a set of inputs and an output variable that denoted the result for each row. Thus, I created a “Result” column and randomly assigned either a 1 or a 0 to that column for each row to denote a win or a loss respectively. Based on the value of that variable, I removed the data on either the winner or the loser; the final dataset had one player’s statistics and whether he won or lost the match. I had to do some data processing; there were a few matches with statistics that did not make sense, such as those that ended in retirements or withdrawals for certain players. After that, I split the data into training data and test data looking for the cleanest split I could get while keeping close to the 80-20 rule that we talked about. I ended up with 69000 matches in my training set and 18000 matches in my test set, which is about a 79-21 split.

**Results**

In this section, I’ll describe the process I underwent to construct my networks and some of the relevant findings I discovered during that process. I started with a single-layer network with 24 input features, or the number of statistics detailed for each match, and 2 output features for each class the network will attempt to distinguish, as well a rectified linear activation function. To start, I used 10 epochs, a batch size of 230, and a learning rate of .05. This did not give me good results at all; I found error rates of 50% for both training and testing, which is basically random assignment. This model was helped significantly by the removal of the ReLu activation function in favor of other specifications such as the inverse tangent and sigmoid functions as well as the addition of an optimizer. My best single-layer network consisted of the architecture above (24 input and 2 output features) with a sigmoid activation function and the Adam optimizer with a very low learning rate and weight decay. This model resulted in training error of about 4.5% and a test error of about 4.6% on average. There are a few things to note before proceeding into the more complex networks. First, it was very difficult from here on out to improve on the error metrics that the single-layer network gave; all my more complex structures ranged from 4.4%-5% error the vast majority of the time, so it is possible that is the best an artificial network can do. Second, I saw that the ReLu activation function did not do very well, so I shelved that one in favor of the other functions I mentioned moving forward. Lastly, I thought it was somewhat interesting to note that the other optimizers, stochastic gradient descent and RMSprop, functioned best with a high momentum value (.6-.9), which meant that they performed better with a higher number of gradients for this data.

My process for working through more complex networks with different techniques and a larger number of layers was essentially trial and error. I tried different combinations of hyperparameters, number of layers/parameters, optimization, regularization, and activation functions. I will not go into detail about all the models I tried, but I made a few notable discoveries during the process. For networks with a high number of layers that presented the largest variation in error with differing architectures, the addition of more non-linear activation functions drastically increased loss and error to about .69 and 48% respectively. The effect on less extensive networks was much less noticeable, but still present. This made me think that perhaps most of the relationships in my data are linear, which could work in favor of the single-layer and less complex networks. Also, I found that batch normalization was most effective technique for improving multiple-layer networks of any size; for large networks, it brought error down from about 5% regularly to about 4.5% on average and brought the loss down to .1-.2 consistently. Unfortunately, regularization did not have much of an impact, and neither did increasing or decreasing the number of input and output features in the linear layers. I began to progressively cut the number of layers in my network down from about 15, looking for the best combination of accuracy, simplicity, and computational efficiency.

**Discussion**

I settled on a three-layer network with a sigmoid activation function and batch normalization after each linear layer. On average, this network gave a training error of about 4.4% and a test error of about 4.5%. I used 15 epochs, a batch size of 345, and a low learning rate of .05, along with the Adam optimizer with learning rate of .002 and weight decay of .005. Comparing this to the single-layer network I used as my control, there does not seem to a significant improvement in error metrics by adding more layers to this model. Thus, we can tentatively conclude that this data may be better suited to a less complex model, such as logistic regression or another classification method. Overall, a misclassification rate of about five percent is quite good in my experience with sports analytics projects; I have tried similar classification projects with soccer, and I did not get anywhere close to that level of accuracy. So perhaps my project can serve as further evidence towards the utilization of artificial neural networks in sports analytics. For further work, I would attempt to run this project with some other methods and see if my hypothesis of linear relationships is true, or if the other methods can approach this level of accuracy with this set of ATP tennis data.

