Artificial Neural Network for Baseball Prediction

Artificial Intelligence - Prof. Luke Hunsberger

Robert Winkelmann & Jordan Gotbaum

May 17, 2017

1 Abstract

This report will review our implementation of an Artificial Neural Network (ANN) for use predicting the run differentials of baseball games. The ANN was implemented in Common Lisp, using a simple design with a single hidden layer. The ANN was trained and tested on public, game-level MLB data. Our ANN implementation shows that ANN can be used to some extent to improve predictions of baseball games, though more thorough analysis is needed.

2 Objective

Our objective was to create an ANN that could predict the outcomes of baseball games given input data with relatively reliable results. We wanted to have a viable ANN that we could train on various years worth of existing data, and ultimately use it to predict the 2016 baseball season's data.

3 Data

The data that we ran our ANN on was game logs from retrosheet.org. This data came in files with all of the games for one year and included information about who played, how many runs were scored, as well as the majority of the team stats for the game. We determined the best of these stats to extract from the file for our ANN to use as its inputs. We also extracted the runs that each team scored and used that to find the run differential for the game, which was our output value for training the ANN.

4 Design

We designed our ANN in lisp to look at the stats for the 2 teams playing as well as the 2 starting pitchers for the game. We had pitcher and team structures to keep these statistics stored. As we would iterate through the file game by game, we would first run the current statistics through the ANN and after we would update all of the structures. This was done in both our training and testing so there would be no impact from the current game in predicting its own outcome.

We utilized much of the code presented in class for the various vector functions and weight updating involved in training and using the ANN, written by Prof. Luke Hunsberger. We did not utilize a bias node for training the ANN.

The ANN took sixteen inputs - eight for each team, and solved for the expected runs of the visiting team less the expected runs of the home team. The inputs are as follows:

- RBIs - Runs Batted In
- Batting Average - Average Hits per At Bat
- Slugging Percentage - Average Bases Earned per At Bat
- Errors - Defensive Mistakes
- Average Run Differential for the Past 10 Games
- Batting Average Against Starting Pitcher
- Slugging Against Starting Pitcher
- Starting Pitcher Earned Run Average (ERA)

We normalized the various statistics such that they were within the range (-1, 1). In the process of propegation, the output variable was normalized via the sigmoid function. We then used a logit function (inverse sigmoid) to see the true predicted run differentials.

As the data for a given season of games is processed, statistics for each team and pitcher are stored and updated in a hashmap as new values are presented.

We determined success as leaning toward the winning team, that is having the predicted run differential have the same sign as the actual.

In order to calibrate our model, we developed a test suite to consider the performance of the ANN with many different configurations. Specifically, we considered having 4, 8, 10, 16, and 32 hidden nodes in our middle layer. We trained and tested those ANN configurations with alphas of 0.0001, 0.001, 0.01, 0.05, 0.1, 0.25, 0.5, and 1 and with 20, 50, 100, 250, and 500 training iterations on the sample data (2,400 games). For testing purposes, we trained the ANN on the 2005 season data and then tested it against the 2006 season data.

5 Conclusions

5.1 Implementation

Our biggest conclusion was that implementing an ANN for complicated data is a very arduous task. Especially given the fact that once the implementation is complete, you need to test and train it on the data in many different configurations before you can find an optimal predictor. This involved various combinations of the size of your hidden layer or layers, the extent of training, and the sensitivity parameter, alpha.

5.2 Data

There were a handful of problems with our data. Primarily, that our data input was limited to the statistics we could easily calculate and was on a game level granularity. This prevents us from attributing more advanced metrics based on specific players' performance at bat and specific pitchers. Furthermore, we could only consider data from the starting pitcher, who in almost all cases does not pitch the entire game and is generally followed by a mid-game pitcher or closer.

We had initially planned to train the ANN on multiple years of data, but ultimately could only train it on one year given the multitude of different configurations we had to consider. If we had more time, we could have trained the ANN much more extensively and on multiple years of data.

5.3 Results

We had lots of difficulties handling our output variable given that we were comparing it against the actual run differential, which was an integer. The existing code base normalized output variables, and rather than altering that

we were advised to invert the normalization of the output in our functions. For almost all of the calibrations we ran, the output variable was exclusively positive or negative. That being said, as the average value approached zero, the ANNs generally did a better job predicting outcomes.

This indicates that though our calibration was clearly imperfect, the ANN was able to improve predictions of game outcomes as its calibration and training lined up more appropriately.

Our best run of the calibration and test suite correctly predicted the winner 55.8% of the time, with 4 hidden nodes in our middle layer, $\alpha = 0.01$, and 1000 iterations of training on a season of games.

Unfortunately, we did not have time to more thoroughly test various calibrations and utilize multiple years of data.