CS230

A Deep Learning Approach to Player Forecast Systems

Elliott Chartock*
Department of Computer Science
Stanford University
elboy@stanford.edu

1 Introduction

In this project we build a player forecast system that predicts season performance statistics for professional baseball players. Such forecast systems are pivotal for the success of the billion dollar fantasy sports industry. Additionally, there are practical applications to building highly accurate player prediction models. In the current Moneyball age of baseball, sabermetrics often guide management decisions to draft, trade, or drop players on their roster. Baseball managers around the world deserve better player forecast systems, be it a 14-year-old kid picking his first ever fantasy roster, or Theo Epstein devising his next World Series winning roster of star-studded, undervalued athletes.

Player predictions are made via a fully-connected neural network (FCNN) and a recurrent neural network (RNN). The input to the FCNN is a concatenation of previous player-season statistics. The input to the RNN is previous player-season statistics inputted as a time-series. For all models, we predict the number of home runs in the next season. Home runs is chosen as the prediction task because is is an easy-to-understand metric that has positive correlation with a team's winning percentage [1] and also has a high point value in fantasy sports. Making accurate predictions on the number of home runs in the next season is good proof of concept for being able to apply these models to make accurate player predictions over other statistics in future seasons.

2 Related work

Despite its data rich and statistics obsessed culture, baseball still has a large contingency that is adverse to statistical methods that aren't entirely intuitive or human-interpretable. For this reason, baseball is a field with exhaustive data records that remains heretofore absent of deep learning influence. Early data driven forecast systems attempted to map a player to one of 13 preset career paths, then applied an algorithm to "walk" the player down his given path, resulting in predictions for the upcoming season [2]. In 2006, Nate Silver introduced PECOTA, which is currently still the leading forecast system used by Baseball Prospectus to make fantasy sports predictions. PECOTA extends the idea of [2] to go from 13 preset career paths to an infinite number of career paths. The PECOTA algorithm makes player season predictions based on a weighted sum of all similar previous player seasons throughout baseball's 150 year history.

This project applies deep networks (fully connected network and recurrent neural network) by directly predicting the target stat in an attempt to outperform PECOTA's predictions. It should be noted that PECOTA projections are proprietary and the authors have not purchased access to such data. For this project all models will be compared to a common baseline. Comparison to industry projections will occur in future work.

*Thanks to Abhijeet Shanoi for guidance as project mentor.

3 Dataset and Features

3.1 The Lahman Database

In 1994, Sean Lahman began an initiative to make baseball statistics exhaustive and freely available to the public. The current state of the Lahman Database has over 20 tables containing statistics on hitting, batting, pitching, fielding, managers, and more. We use the **Batting** table for this project, which has seasonal hitting data for all Major League Baseball players between 1871 and 2016 [3]. Each row is a vector corresponding to a player's batting statistics for a given season, which can be identified by the key pair (*playerID*, *yearID*). The input features are comprised of common natural numbered baseball statistics, such as games played, at bats, and home runs. Figure 1 contains all 17 features used for training and shows an example of a row for a given player along with the label we are trying to predict.

	playerID	yearID	G	AB	R	H	2B	3B	HR	RBI	SB	CS	BB	SO	IBB	HBP	SH	SF	GIDP
Input	ortizda01	2015	146	528	73	144	37	0	37	108	0	1	77	95	16	0	0	9	16
Predict	ortizda01	2016							38										

Figure 1: Sample feature vector containing statistics for David Ortiz's 2015 season.

3.2 Preprocessing

The Lahman Database contains 102,816 distinct player-season pairs. First, we omit all inputs with missing data, leaving 66,175 player-seasons. We further filter by ignoring all player seasons in which the player does not reach a threshold number of at bats. [4] found that 100 at bats yields good results. The purpose of such a threshold is twofold. Firstly, a player with a greater number of at bats is more representative of the types of players we want to make predictions on: low usage players are not frequently drafted in fantasy baseball. Secondly, baseball statistics naturally have a large variance and with so few at bats, a player season would offer more noise than signal. After at bat filtering there are 21,981 player-seasons. Lastly, we omit all rows that do not contain a valid label, which corresponds to seasons in which the player did not have over 100 at bats in the next season (caused by injury or retirement, often). After all data cleaning and preprocessing, there are 17,130 valid player seasons. We employ a 80-10-10 split of the data into training, development, and test sets.

	Train	**Dev**	**Test**
No. Player-Seasons	13,704	1,713	1,713

Table 1: Train, development, and test set sizes.

4 Methods

4.1 Mean Last-K Baseline

The baseline for this model is a simplification of the Marcel the Monkey projection system [5]. To predict next year's player performance we simply average the previous K years numbers for that category. More precisely, for predicting target label 'HR' of player vector p in year t for $K = k$, we calculate:

$$p_t[HR] = \frac{1}{k} \sum_{i=t-k-1}^{t-1} p_i[HR]$$

For the Mean Last-K baseline, we test K = 1, 2, 3, 4, 5. While a very naive baseline, Marcel has been observed to outperform state-of-the-art projection systems in some seasons [6]. We hope to improve upon these results with deep learning models.

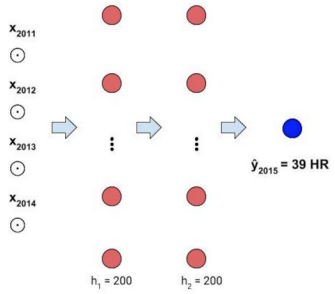

Figure 2: Fully Connected Last-K architecture for K = 4

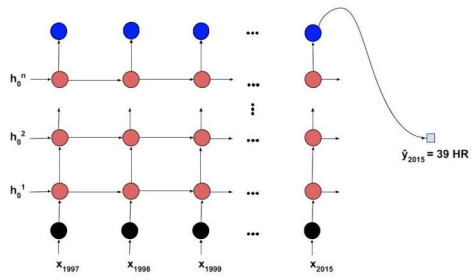

Figure 3: RNN architecture

4.2 Last-K Fully Connected Neural Network

In this model, we pass player data through a fully connected neural network to directly predict the number of home runs the player will hit in the following season. The input to Last-K FCNN is a concatenation of the statistics of the last K seasons for the given player. If the player did not play in K seasons yet, we interpolate the closest year's statistics as that season's statistics. K is a hyperparameter that we tune in Section 5.2. Figure 2 shows the basic architecture for this model, which is discussed in greater detail in Section 5.2.

4.3 Recurrent Neural Network

Given a sequence of vectors $x_1, ..., x_T$, a Vanilla RNN with L layers computes, at a single timestep t and layer l,

$$h_t^l = \tanh(W_{hh}^l h_{t-1}^l + W_{xh}^l x_t)$$
$$y_t^l = W_{hy}^l h_t^l$$

where h_t^l is a hidden vector of dimension H at the tth timestep and layer l, and W_{hh}, W_{xh} are parameters to the model. If we want to make a prediction on a player's statistics for year $T+1$, we the input to the model will be season feature vectors $x_1, ..., x_T$, where x_1 corresponds to the first year that player appeared in the MLB. Here all vectors y_t^L are ignored except for y_T^L.

The idea behind this method is that there are frequent temporal patterns that guide a players career trajectory. If we can learn these patterns we should be able to accurately project the next season of a player based on his performance in the previous few seasons.

4.4 Loss and Evaluation

The loss function for the models is the mean squared error, which measures the average of the squared difference between our predictions and the ground truth values.

$$MSE(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

We evaluate our models using the coefficient of determination R^2, a standard statistical measure of what proportion of the variance in our predictions can be explained by variance in our input features. With truth values y_i, predicted values $\hat{y}_i$, and mean of truth values $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$, this is calculated as

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}.$$

The best possible R^2 value is 1.0 (indicating perfect explanation of predictions by input features), and smaller or negative values indicate worse model performance.

5 Experiments and Results

5.1 Baseline Experiments

The Mean Last-K baseline algorithm is run for K = 1, 2, 3, 4, 5 on the the development set to determine the best value of K to use against the hidden test data. Table 2 shows that K = 2 yields the largest R^2.

K	1	2	3	4	5
$\boldsymbol{R^2}$	0.453	**0.526**	0.516	0.503	0.492

Table 2: Development set R^2 for baseline.

5.2 Last-K FCNN Experiments

For all experiments, batch normalization, dropout (p = 0.2), and ReLU non-linearity is applied to the data at each layer. Emperical results show that this combination of data manipulations yielded the best results. We perform parameter optimization using Adam optimizer and train all models for 50 epochs.

We perform architecture search and hyperparameter optimization to choose the best Last-K FCNN model. The following hyperparameters are tested: the number of layers in the network (L), the number of hidden units in each layer (h), the learning rate for Adam optimizer (lr), and the number of previous years' input vectors to concatenate and feed into our model (K). Experiments on weight decay for Adam optimizer had negligible effects on the model. For each experiment, we isolate the hyperparameter being tuned, keeping all other hyperparameters constant. It should be noted that for L = 0 we are simply running linear regression. Table 3 shows the R^2 on the development set for each hyperparameter experiment.

L	0	1	**2**	3	4	5	-
$\boldsymbol{R^2}$ **for L**	0.569	0.584	**0.591**	0.589	0.588	0.589	-
h	50	100	**200**	500	-	-	-
$\boldsymbol{R^2}$ **for h**	0.589	0.588	**0.592**	0.591	-	-	-
lr	5e-3	1e-3	5e-4	**1e-4**	5e-5	1e-5	5e-6
$\boldsymbol{R^2}$ **for lr**	0.589	0.591	0.588	**0.591**	0.589	0.566	0.294
K	1	2	3	4	**5**	-	-
$\boldsymbol{R^2}$ **for K**	0.566	0.588	0.591	0.584	**0.593**	-	-

Table 3: Development set R^2 for Last-K FCNN hyperparameters.

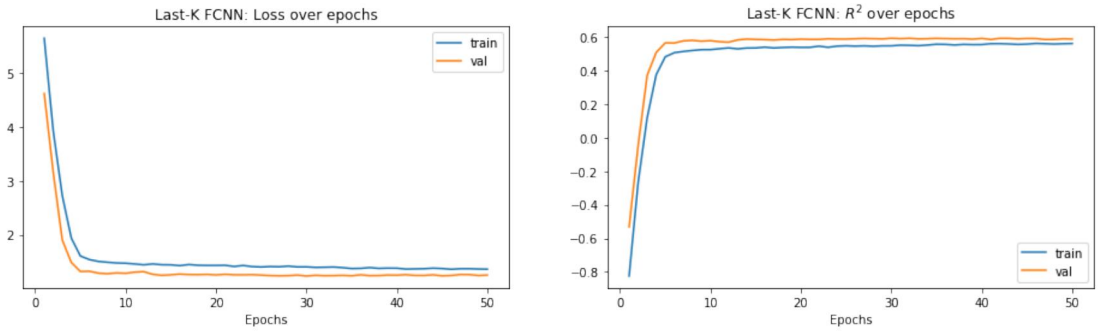

Figure 4: a) Loss over epochs for Last-K FCNN b) R^2 over epochs for Last-K FCNN

The best model concatenates the previous 5 years' statistics, uses Adam learning rate of 1e-4 and has 2 layers that contain 200 hidden units each. Figure 4 graphs loss and R^2 over epoch for training and validation data. The results of this model on the test set are in Section 5.4.

5.3 RNN Experiments

We use the vanilla RNN implementation as described in Section 4.3. The loss function MSE is only applied to the final timestep prediction. The following hyperparameters are tested: the number of layers in the network (L), the number of hidden units in each layer (h), the learning rate for Adam optimizer (lr). We also test two training implementations, one where all previous seasons are inputted as a time series, and another where a random subsequence of the most recent K years are inputted to the RNN. Table 4 shows the R^2 on the development set for each hyperparameter search.

L	1	**2**	3
$\boldsymbol{R^2}$ for **L**	0.562	**0.582**	0.575
h	20	50	**100**
$\boldsymbol{R^2}$ for **h**	0.520	0.575	**0.580**
lr	5e-4	**1e-4**	5e-5
$\boldsymbol{R^2}$ for **lr**	0.520	**0.562**	0.4533
Random subsequence	**True**	False	-
$\boldsymbol{R^2}$	**0.584**	0.579	-

Table 4: Development set R^2 for RNN hyperparameters.

The best model stacks two RNN layers with 100 hidden units at each timestep, uses learning rate of 1e-4 for Adam optimizer, and inputs a random subsequence of a players previous statistics during the training process.

5.4 Best Models Results and Discussion

Table 5 shows the results of the best models from Sections 5.1, 5.2, and 5.3 evaluated on the hidden test data.

	Train Loss	Train R^2	Dev Loss	Dev R^2	Test Loss	Test R^2
Baseline	-	0.584	-	0.526	-	0.489
Last-K FCNN	1.402	0.547	1.232	0.592	1.334	0.566
RNN	1.420	0.541	1.248	0.584	1.238	0.561

Table 5: Test set results for best baseline, FCNN, and RNN models.

We can see that the best Last-K FCNN slightly outperforms the best RNN model with a test R^2 of 0.566. Error analysis on the test set shows that for both implementations, the two worst predictions occur on breakout seasons. For George Foster's 52 home run season in 1977 (he had 29 home runs the year before) the Last-K FCNN predicts that he will hit 23 homeruns. Similarly, in Kirby Puckett's third year in the majors he jumped from 4 home runs to 31 home runs, while our model predicts that he will only hit 3 home runs. Both top models, outperform the baseline significantly, though struggle to predict outliers or identify breakout seasons. In Figure 5 it is clear that the model does not predict seasons with many home runs well, which lowers the R^2 score.

6 Conclusion

Our top deep implementations of a player projection system outperform a common baseball baseline by R^2 of 0.08. The Last-K FCNN and RNN models perform similarly, which shows that career projections are not easier to predict when considering the career of a player as a time-series versus looking at their past statistics holistically. While both neural network implementations perform well

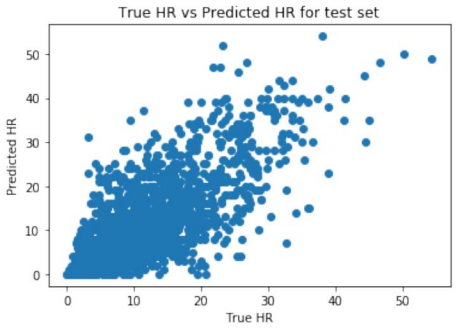

Figure 5: Home run ground truth vs. predictions for Last-K FCNN

on a hidden test set (R^2 = 0.566), the models do not perform well at predicting break out seasons, which would be a favorable trait of a good projection system.

6.1 Weighted Sum of RNN-Learned Player Embeddings

With six more months to work on this project, we would test a player projection system that combines our work with the methods in [2]. We would extend our RNN implementation by making predictions based on similarity of a player's learned embedding. Such a method is used heavily in NLP, in which it is common to apply a dimensionality reduction technique to an occurrence matrix and then compare vector similarity at a lower dimension [7]. In NLP these mapped embeddings have a learned semantic meaning that can be human-interpreted through applying a distance function, such as cosine similarity, to the embedding of a pair of words. In this setting, we would use h_T^L as a player embedding that might carry some sort of player semantics learned from the player's career stats. By using a distance function, the hope is that we would be able to better compare two players than PECOTA does by just comparing their raw statistics.

7 Contributions

Not applicable - I am the only contributor to this project.

References

[1] Julia Prusaczyk. Do more home runs mean more wins?, Sep 2016.

[2] 6-4-3: Reasonable person standard, Aug 2002.

[3] Nic. Baseball data | kaggle, Dec 2017.

[4] Brandon Liu and Bryan McLellan. Predicting season-long baseball statistics.

[5] Marcel the monkey forecasting system.

[6] Henry Druschel. 2015's projection systems in review, part two, Jan 2016.

[7] Hwee Tou Ng and John Zelle. Corpus-based approaches to semantic interpretation in nlp. *AI magazine*, 18(4):45, 1997.