# DEEPBALL: MODELING EXPECTATION AND UNCERTAINTY IN BASEBALL WITH RECURRENT NEURAL NETWORKS

BY

DANIEL CALZADA

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2018

Urbana, Illinois

Adviser:

Professor Oluwasanmi Koyejo

## Abstract

Making reliable preseason batter projections for baseball players is an issue of utmost importance to both teams and fans who seek to infer a player's underlying talent or predict future performance. However, this has proven to be a difficult task due to the high-variance nature of baseball and the lack of abundant, clean data. For this reason, current leading models rely mostly upon expert knowledge. We propose DeepBall, which combines a recurrent neural network with novel regularization and ensemble aggregation. We compare this to Marcel, the industry-standard open-source baseline, and other traditional machine learning techniques, and DeepBall outperforms all. DeepBall is also easily extended to predict multiple years in the future. In addition to predicting expected performances, we apply standard machine learning techniques to extend DeepBall to model uncertainty in these predictions by estimating the maximum-likelihood distribution over potential outcomes for each player. Due to the positive results, we believe that in the future, DeepBall can be beneficial to both teams and fans in modeling expectation and uncertainty. Finally, we discuss potential extensions to the model and directions of future research.

*To my parents, Manuel and Linda, who have shown me unwavering support in every way during my time as a student.*

## Acknowledgments

I would like to thank Microsoft Azure for providing computational support for the development of and experiments on DeepBall, as well as the volunteers at Retrosheet who have spent countless hours capturing and recording baseball data from the last century. Without this work, this paper would not have been possible. My research would also not have been possible without the work of the numerous researchers and enthusiasts who have studied the numbers, statistics, and trends in the game of baseball, and whose work has paved the way for the baseball dimension of this research. Furthermore, many thanks goes to the academic researchers who have laid the machine learning groundwork for this work, and to Professor Svetlana Lazebnik, in whose class I discovered my passion for applied deep learning.

Additionally, I would like to extend my sincere thanks to my thesis adviser, Professor Oluwasanmi Koyejo, for his instruction, constructive criticism, assistance, mentorship, and encouragement in the development of this work. Our conversations together have refined my thought processes and shaped my research in untold ways. I also appreciated being a teacher's assistant under him, an enjoyable experience from which I derived much benefit, both personally and academically.

I would also like to thank my friends who have stuck with me through thick and thin during my graduate studies and for whose friendships I am deeply grateful. In addition, heartfelt thanks go to my family, Manuel and Linda, David and Carrie, and Kevin and Amy, for their continuous interest and encouragement along this journey.

Most importantly, I would like to thank my Lord and Savior Jesus Christ, who has faithfully sustained me during this time, supplying true joy and hope through all difficulties encountered and regardless of the outlook of this research. Through it all, He has continually reminded me what is truly valuable, and for that I am eternally grateful.

## Table of Contents

## List of Tables

# List of Figures

## List of Abbreviations

| | |
|---|---|
| MLB | Major League Baseball. |
| NL | National League. |
| AL | American League. |
| DH | Designated Hitter. |
| wOBA | Weighted On-Base Average. |
| WAR | Wins Above Replacement. |
| oWAR | Offensive Wins Above Replacement. |
| AVG | Batting Average. |
| SLG | Slugging Percentage. |
| K%, kpct | Strikeout Rate. |
| BB%, bbpct | Walk Rate. |
| OBP | On-Base Percentage. |
| OPS | On-Base Plus Slugging. |
| PA | Plate Appearances. |
| G | Games Played. |
| AB | At-Bats. |
| K | Strikeouts. |
| BB | Walks. |
| HBP | Times Hit-By-Pitch. |
| SF | Sacrifice Flies. |
| 1B | Single. |
| 2B | Double. |
| 3B | Triple. |
| HR | Home Run. |

# Chapter 1: Introduction

Since the National League was founded in 1876, baseball has captured the interest of Americans and, perhaps to a lesser extent, the international community. Because of this, people have sought to understand and describe it quantitatively using simple statistics. In the middle to late 20th century, the usefulness of a traditional statistic, batting average (hits divided by at-bats), was called into question as a way to evaluate a batter's true performance and contribution to his team because it weights all hits equally and does not account for other contributions, such as drawing walks.

Baseball statisticians, commonly known as sabermetricians, developed new offensive statistics such as OBP and OPS to account for this gap. Furthermore, statistics like WAR [Fangraphs, 2013b, Baseball-Reference, 2017] and wOBA [Fangraphs, 2013c] have emerged in an attempt to better capture and isolate a batter's true talent and contribution to his team. While these are useful descriptive statistics, they have limited predictive quality, in part since they do not account for the contact quality of a hit. For example, was a batter's single the result of a hard-hit ball, poor placement of the infielders, good luck, such as a well-placed pop-up that was simply out of reach of the fielders, or a combination of these? A hard-hit ball might indicate talent that might carry over into the player's future performance, whereas lucky outcomes should not be expected to continue in the future. Knowing this will improve our predictions, but with these statistics, all are treated equally.

We seek to explore the answers to two major questions: can a machine learning model isolate a batter's true talent from this noise, and can it use this talent to generate a projection for the next season. We claim that the second objective is sufficient; in other words, if a model can generate an accurate prediction for a batter's performance, it must have an internalized representation of the batter's true talent.

Clearly, answering this question is of utmost importance to many people and groups, including Major League Baseball team front offices, who seek to make informed trade and playing time decisions optimal for short-term and long-term team performance, and fantasy baseball players, who seek to build an optimal roster for their league. For this reason, we propose a new projection system, DeepBall, to generate these predictions.

## 1.1 Related Work

Given the importance of MLB teams having reliable projections for players, we suspect that most if not all teams have their own proprietary projection algorithms, most likely relying upon proprietary data. In recent years, independent individuals or groups have developed systems whose results, and to some extent methods, are available to the public. However, according to our research, this manuscript is the first academic paper on this topic.

One popular baseball prediction system is Marcel, proposed by Tom Tango [Tango, 2001], which is designed to serve as a baseline for player season projection tasks. In generating a prediction for one statistic (for example, home runs), it uses a 5/4/3 weighted combination of the player's totals in that area for the last 3 seasons, taking into account the league averages over that time. It then regresses towards the mean for the league and factors in the player's age. It projects rookies of the same age to perform at the exact same level, approximately league average. Since it is open-source and thus it can be retroactively evaluated, we will use this as the primary industry-standard baseline. Interestingly, more complex systems struggle to significantly improve upon Marcel despite its simple design, and this underscores the difficulty of the problem we are attempting to solve [Druschel, 2016b, Druschel, 2017].

The Player Empirical Comparison and Optimization Test Algorithm, more commonly known as PECOTA, is an algorithm created by Nate Silver, then with Baseball Prospectus. To create a projection for a player, it explicitly searches for players with a similar body type, age, and past career [Druschel, 2016a]. It then uses the subsequent seasons of these comparable players to predict how our player will perform. With simple alterations, this algorithm can be used to predict other useful things such as Breakout Rate and rough floor/ceiling performances for a player. While PECOTA is a popular projection system, its player projections are held behind a paywall, so we did not examine it in depth.

Steamer was initially developed as a high school project [Cross et al., 2008] and is now used as one of the primary public projection systems. The system resembles Marcel by using a linear combination of five previous years including a regression towards the league mean [Druschel, 2016a, Gloeckner, 2014]. The linear weights, however, are not fixed but are determined by a regression over past players. Beyond this, additional significant details have not been released.

Another system whose results are available to the public is ZiPS, which was developed by Dan Szymborski with Baseball Think Factory in 2003. In a sense, this system is a hybrid between Marcel and PECOTA. It functions by first computing a weighted average of the player's last three or four years, after which it finds comparable players in a similar fashion as PECOTA, using these comparisons to generate its predictions [Druschel, 2016a]. ZiPS uses minor league data [Davidson et al., 2010] and heavily relies upon expert knowledge when determining player similarity [Szymborski, 2008].

These systems share some similarities. First, each system uses some weighted combination of a player's previous years, and each system (with the possible exception of PECOTA) ignores all data more than five years old [Davidson et al., 2010]. Furthermore, in the cases of Marcel, Steamer, and ZiPS, the data is weighted by a linear combination. Second, we see that player age is important in making projections. This supplements existing research which makes the claim that typical hitters and pitchers peak between ages 28-32 [Bradbury, 2010]. For this reason, most systems include a variable for the player's age. Third, the more complex systems incorporate expert knowledge, especially ZiPS when measuring player similarity.

Though PECOTA, Steamer, and ZiPS provide additional perspective regarding industry-standard approaches for solving this problem, we are unable to comprehensively and fairly compare their results to DeepBall for two reasons: since they are kept closed, the algorithm and implementation details are not known to us, and we have no access to additional proprietary data used for training these systems.

## 1.2 Problem Formulation

We will formally express the batter season projection problem in the following way. Let $k \in \mathbb{Z}^+$, let $\mathbb{B}$ be the set of all batters with at least one plate appearance, and let $\mathbb{S}$ be the set of all seasons. Given a batter $B \in \mathbb{B}$ and season $S \in \mathbb{S}$, we want to predict $B$'s performance in some categories (batting average, strikeout rate, etc.) for season $S$ using only information known before the start of season $S - k + 1$. We can interpret the parameter $k$ as the *shift factor* for our predictions. Setting $k = 1$ creates a projection system that estimates $B$'s performance for the subsequent year, setting $k = 2$ creates a system that predicts $B$'s performance two seasons in the future, and so on. Each $B$ can be represented as a sequence of the seasons in which $B$ played, and each of those seasons can be considered a timestep. In this way, we can frame this as a time-series prediction problem.

By posing our problem in this way, we have generalized it relative to those solved by the systems described in Section 1.1. Specifically, each of those systems is only able to predict one year in advance ($k = 1$), whereas we can predict for any $k$. (Though it was only designed for single-year predictions, in this paper, we have extended Marcel to predict multiple years in advance by creating a feedback loop, using previous years' predictions as input.)

In this case, we will be trying to predict batter performance in eight categories. Plate appearances (PA) are the number of times a batter went to the plate during the season, at least 500 for a full-time player. Games (G) are the number of games in which a batter appeared. In this case, we will only count the number of games with at least one plate appearance. A team plays 162 games throughout the course of the season, and a full-time player might play in around 140 of those. Offensive Wins Above Replacement (oWAR) is a sabermetric statistic designed to isolate a player's overall offensive contribution to his team, measured in wins added to the team relative to some baseline. General Wins Above Replacement (WAR) has been described by [Baseball-Reference, 2017]. Finally, we are interested in predicting five rate statistics:

$$\text{AVG} = \frac{\text{1B} + \text{2B} + \text{3B} + \text{HR}}{\text{AB}}, \tag{1.1}$$

$$\text{SLG} = \frac{1 \times \text{1B} + 2 \times 2B + 3 \times 3B + 4 \times HR}{\text{AB}}, \tag{1.2}$$

$$\text{K\%} = \frac{\text{K}}{\text{PA}}, \tag{1.3}$$

$$\text{BB\%} = \frac{\text{BB}}{\text{PA}}, \text{ and} \tag{1.4}$$

$$\text{OBP} = \frac{\text{1B} + \text{2B} + \text{3B} + \text{HR} + \text{BB} + \text{HBP}}{\text{AB} + \text{BB} + \text{HBP} + \text{SF}}. \tag{1.5}$$

While on the surface this problem may not seem more difficult than any other regression task, there are certain subtleties that we must take into account. First, the data we have is limited. There have only been 7894 players who have debuted between 1958 and 2014 with at least one career plate appearance, and these players have combined to bat in only 43891 batter/seasons. Therefore, there are only 7894 independent training sequences and 43891 dependent examples. To make matters worse, most of these seasons contain very little information. When eliminating all batter seasons with less than 100 plate appearances, there are only 3305 batters and 20963 batter/seasons. This issue was noted by [Yakovenko, 2017] when suggesting that complex neural networks may not have success with limited data.

Second, the data we do have is low quality and incomplete. Even though the database we are using includes data since 1958, some important pieces of data, such as pitch sequences, are not available for many of these years. As such, we need to deal with the issue of missing data.

Third, league averages are fluid. Baseball, like any sport, evolves over time. From the lowering of the mound after the 1968 season to the addition of the designated hitter in the American League beginning in 1973 to the steroid era of the late 1990s and early 2000s, the game has been constantly changing, and with it, the league averages in certain statistics have changed as well. Therefore, a good projection system must be able to adjust to the era

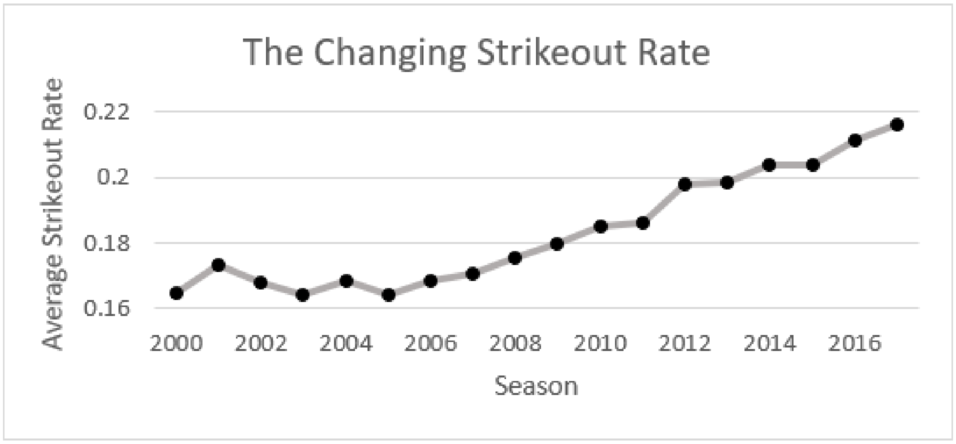


Figure 1.1: The league average strikeout rate has changed drastically since the turn of the century, and fluid league averages must be taken into account. Source data is from [Baseball-Reference, 2018].

for which it is projecting. For example, in recent years, the league average strikeout rate has been steadily climbing as seen in Figure 1.1. Because of this, a strikeout rate of 22% in 2000 would have been an outlier and most predictions would have been around 17%. In 2017, however, a strikeout rate of 22% was normal and projection systems would have predicted most players to be centered around this average.

# Chapter 2: Data Sources

All play-by-play and pitch data used for training and evaluation comes from Retrosheet [Retrosheet, 2017]. Since Retrosheet does not have complete play-by-play data from before 1957,[1] we only trained on players who began their career during or after 1958, only using their regular season statistics. The Sean Lahman database was used for player biological information [Lahman, 2016]. Finally, historical and up-to-date sabermetric data is freely available from Baseball Reference [Baseball-Reference, 2017].

To compare DeepBall's performance against Marcel, we used a custom implementation of the algorithm described by [Tango, 2004].

## 2.1 Park Factors

Baseball is unique in that each stadium is different. For this reason, it is imperative to allow our model to see the context in which a batter played or will play during a season. Therefore, when designing the model, an important consideration is park factors, which are measures of how hitter-friendly a stadium is in certain categories. For example, due to its high altitude, Coors Field in Denver is known as a hitter-friendly stadium, so its park factors in offensive categories are higher than the league averages.

There are many varieties of park factors. The most simple formula, popularized by ESPN, simply divides the performance of both teams in road games with those of home games [ESPN, 2018]. However, it has been noted that these factors are both biased and unreliable [Acharya et al., 2008]; therefore, we followed the method suggested in that paper, computing our park factors using least-squares estimates. Simply put, we operate under the assumption that the number of runs[2] scored in a game is dependent on four factors: the

[1]Technically, Retrosheet is missing some games between 1957 and 1973. We did not take this into account in our work. For a list of missing games, see `http://www.retrosheet.org/wanted/index.html`.

[2]This method was also used to compute park factor for other statistics besides runs scored, for example,

offensive team, the defensive team, the stadium, and whether a designated hitter[3] was used in that game. For each season, we create a sparse system of linear equations $A\mathbf{x} = \mathbf{b}$ where each equation is a game and each variable is a factor of either the offensive team, defensive team, stadium, or the designated hitter flag (thus, all entries in $A$ are 0 or 1, with three or four 1's in each row depending on the DH flag), and the elements of $\mathbf{b}$ contain the number of runs scored in that game adjusted for innings played.

Formally, let $b_i$ be the factor for team $T_i$'s offense, $d_i$ be the factor for team $T_i$'s defense and pitching, $p_i$ be the park factor for team $T_i$'s home field, and $f$ denote the designated hitter factor. Furthermore, assume $R_g^{(b)}$ is the total runs scored by the offensive team and $I_g^{(b)}$ is the number of innings in which the team batted. (This takes into account partial innings, so if the home team hit a walk off home run with two outs in the bottom of the 9th, $I_g^{(b)} = \frac{26}{3}$.) Then we have the over-determined linear system

$$\left[\begin{array}{c|c|c|c} B & D & P & F \end{array}\right] \mathbf{x} = \mathbf{b}, \mathbf{x} = \left[\begin{array}{c} \mathbf{b} \\ \hline \mathbf{d} \\ \hline \mathbf{p} \\ \hline f \end{array}\right] \tag{2.1}$$

where $B, D, P \in \mathbb{R}^{2G \times n}$, which represent the batting, defensive, and park factors for each team, and $F \in \mathbb{R}^{2G \times 1}$ which contains the DH flags for each game. As it is written here, $n$ is the total number of teams, currently 30, and $G$ is the total number of games played throughout the season between any two teams. We multiply the games by 2 because each game has two equations, one where a team is the offensive team and another where the same team is the defensive team. Each row of $B$, $D$, and $P$ will have exactly one 1 and the rest of the entries will be 0. The column index of the 1 corresponds to the team's index. $\mathbf{b} \in \mathbb{R}^{2G}$

home runs.

[3]The Designated Hitter rule was adopted in the American League in 1973, allowing another batter to bat in place of the pitcher while not playing any fielding position. This usually results in more offense since pitchers, generally poor hitters, are replaced with a more competent offensive player.

contains the number of runs scored normalized by the number of innings the team batted in, $\frac{R_g^{(b)}}{I_g^{(b)}}$, and $\mathbf{x} \in \mathbb{R}^{3n+1}$ contains the park factors we are interested in computing, in addition to the other factors that are not important for this task. We find a least squares solution for $\mathbf{x}$, from which we extract the park factors $\mathbf{p}$.

While more robust than the ESPN park factors, our park factors still have room for improvement. Most significantly, due to the asymmetric nature of many baseball fields, we believe creating separate park factors for left-handed and right-handed batters will result in more useful factors for prediction tasks [Heipp, 2005].

## 2.2 Missing Batted Ball Data

As discussed in Chapter 1, one important factor for making accurate predictions is determining the amount of randomness hidden within a batter's surface-level statistics. One such factor relates to batted ball statistics: how many ground balls, line drives, fly balls, or pop-ups a player hit, and the results of each of those plays. The underlying premise is that certain outcomes can be a result of good luck, whereas others are a result of bad luck, and simply examining the outcomes does not capture all the available information. Unfortunately, Retrosheet is missing approximately 14% of the batted ball data between 1958 and 2017. To account for this, we trained a Naïve Bayes classifier to estimate a distribution of batted ball types for each missing event. When the original data is missing, we impute these missing values with the expected number of each batted ball type. More details regarding this approach are outlined in Appendix A.

# Chapter 3: Approach

In this chapter, we will outline our general approach in creating DeepBall. We will discuss the inputs and outputs of the model at a high level, architecture, design choices, hyperparameters, regularization, and training settings for our network. In addition, we will outline our ensemble technique which reduces the variance of the models.

## 3.1 Variables

There are many variables used by our model as inputs, outputs, or both.

- *out_stats* contains the output rate statistics we are interested in predicting, listed in Equations 1.1-1.5. Because these formulas are divided by PA or some variant of it and thus may be unreliable for small sample sizes,[1] we used the square root of the batter's plate appearances for that season as sample weights. Furthermore, to account for the shifting league averages discussed in Section 1.2, it is useful to predict players' statistics relative to the league average as opposed to their absolute values [Tango, 2007]. As such, we normalized each of these statistics independently by dividing by the mean so that $\mu = 1$ for each season.[2] Furthermore, we standardize them to the same variance.

- *out_counts* contains the output counts we are interested in predicting: PA, G, and oWAR, as described in Section 1.2.

- *out_fielding* is the number of games started by this player in each season grouped by starting fielding position. We include the number of games started on the bench, when

[1]For example, a batter may have a 100% strikeout rate but only have one PA. In fact, the variance of a rate statistic is inversely proportional to the number of plate appearances. The theory behind this is discussed in further detail in Chapter 5.

[2]The task of converting the relative predictions to absolute predictions is simple. A season-to-season model of league averages is required, which can be created using a linear combination of previous years' averages as suggested by [Szymborski, 2008].

the player comes in as a substitute (position 0), each of the nine standard fielding positions 1-9, and the designated hitter is treated as position 10.

- *out_full_stats* contains the season offensive counting statistics from each of the hitter's seasons. These include statistics about playing time, total number of hits of each type and batted ball data.

- *out_saber* contains sabermetric stats from each of the hitter's seasons, including but not limited to WAR, oWAR, WAA, batting runs, and base-running runs.

- *out_running* consists of base-running counting statistics from each of the hitter's seasons, such as number of stolen bases, times caught stealing, and number of times the runner advanced an extra base when possible.

- *out_plate_discipline* contains plate discipline statistics from each of the hitter's seasons, when available, such as pitches seen, total balls and strikes, etc. These come from Retrosheet and are partially or completely missing for at least 40% of the batter/seasons.

- *out_park* stores the park factors for the fields in which the hitter played each season (including road games), weighted by playing time. For example, if a batter played 50% of his games in Cincinnati, 25% in Chicago, and 25% in Pittsburgh, the values of these park factors will be the park factors for those three stadiums combined linearly by their respective playing time ratios.

- *out_league* represents the total offensive numbers for all position players during the season. We used the same statistics as *out_stats*, in addition to average ground ball, line drive, and fly ball rates.

- *in_park* has the park factors for the home field in which the hitter will play in the upcoming year, which are determined by averaging the park factors for that stadium from the three previous years. When a player gets traded mid-season, these are the park factors for the team for which he first played that season.

- *in_team_flags* contain information regarding the first team for which the batter is playing in the upcoming season. In this case, the flags indicate certain rules that might affect the batter's statistics. One indicates whether a designated hitter was used in this batter's league, and the other indicates whether there was interleague play this season.[3]

- *in_position* is the player's projected fielding position for the upcoming season, taking on the same values as *out_fielding*. For a past season, we dynamically sample one starting position for one of this batter's games. For example, if a batter played 80% of his games at second base and 20% at shortstop, this field would contain second base 80% of the time. For upcoming seasons, we use the batter's projected fielding position, which can be obtained from any major baseball website. We use a one-hot encoding to represent this position.

- *in_age* is the player's age, truncated to an integer. Similar to *in_position*, we use a one-hot encoding to learn an embedding for each age. The studies done by [Bradbury, 2010, Weinberg, 2017] are a part of the significant body of research regarding player aging curves, and we elected to use a one-hot encoding because it allows the model to learn the most general aging functions.

- *in_bio* is the player's biological data, including height, weight, handedness, and whether or not they are a pitcher.

Each of these inputs can be represented as a tensor of size $(p, s, n)$, where $p$ is the number of sequences in the dataset (unique players), $s$ is the maximum number of timesteps in a

[3]Under the designated hitter rule, American League pitchers would be expected to have 0 PA. However, there can be exceptions, such as a pitcher who was traded to a National League team mid-season or during interleague play, where the rules of the home team's league are adopted and can force an American League pitcher to bat.

sequence (most number of seasons in which any player played), and $n$ is the number of values in that variable. These variables can be grouped into four main categories:

- *Output variables* are the results from a season $S_i$ that are not known before the start of the season. We may or may not be interested in predicting them, but we are not allowed to use these as inputs. Examples include all of the variables listed above with the prefix *out_*. In our model, we are only interested in predicting the values of *out_stats*, *out_counts*, and *out_fielding*.

- *Rolled results variables* are the output variables from previous seasons that are made available to the model when generating its future predictions. These tensors can be directly computed from original output tensors by shifting the output tensor $k$ units along its timestep axis and padding the first $k$ timesteps with zeros. By convention, we will refer to these rolled output variables by their original name prepended with *in_*, for example, *in_out_full_stats*. We did not use  and *in_out_counts* inputs because this information is already fully encoded into *in_out_full_stats* and *in_out_saber*.

- *Input variables* are known about a player's season $S_i$ before it starts, such as *in_fielding_position*, *in_park*, and *in_age*.

- Finally, *bio data* is known before a season and is considered fixed throughout a player's career. While it could be represented as a tensor of size $(p, n)$, it is convenient to repeat the same values across all timesteps to keep the tensors the same number of dimensions. The only bio data input is *in_bio*.

## 3.2 Architecture

The information flow and basic architecture of our model is outlined in Figure 3.1. First, all inputs are passed through a Batch Normalization layer [Ioffe and Szegedy, 2015], standardizing the values so $\mu = 0, \sigma^2 = 1$, optionally rescaling and shifting the values. Exceptions

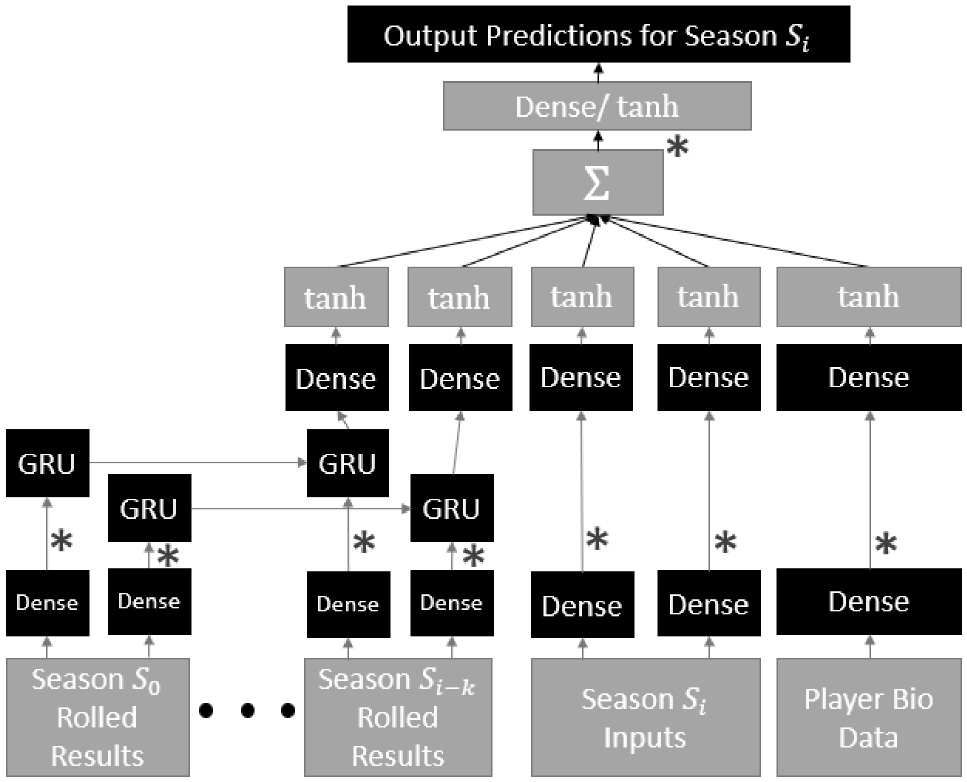


Figure 3.1: A basic overview of the recurrent neural network architecture

to this are *in_out_full_stats*, which does not adjust the $\beta$ and $\gamma$ terms, and *in_out_league*, which also does not adjust $\beta$ and $\gamma$ and adds Gaussian noise with $\sigma = 0.1$. Noise is added to ensure that the model sees different values; otherwise, it would only see about 60 distinct sets of values and may overfit on them.

After the preprocessing step, each input is individually passed through a fully-connected layer that doubles its dimensionality. The exceptions are *in_out_full_stats*, *in_out_plate_discipline*, and *in_out_running*, where the dimensionality is tripled, activated with tanh, and then restored to its original size by a linearly-activated dense layer. Other exceptions include *in_position* and *in_age*, whose output dense layer has dimensionality 8 to reduce overfitting on specific values.

For the player bio and each season input type, a single dense layer with tanh activation is applied. Each rolled output variable is passed through its own Gated Recurrent Unit cell [Cho et al., 2014]. An exception to this is *in_out_full_stats*, *in_out_saber*, *in_out_park*, and *in_out_league*, which are concatenated together and passed through a fully-connected tanh-activated layer with output size 48. We do this to reduce the model's parameter count, but we believe it carries additional significance since it contains most details about a player's general season performance adjusted to the park and era.

After each individual input has been fully processed, a tanh-activated fully-connected layer with output size 85 is applied to each and the results are summed, resulting in one 85-dimensional vector for each timestep. Here, addition worked better than concatenation, and we suspect it was because it requires fewer parameters in subsequent layers and it allows the gradient to flow through less-significant variables, such as the bio data, that might have otherwise lost all gradient flow. Furthermore, we used tanh activation because it worked better than ReLU in practice. Even though the possibility of a vanishing gradient exists, we did not see this in practice. We then apply a dense layer to this vector.

Finally, we can generate the player's season predictions for *out_stats*, *out_counts*, and *out_fielding*. For each of these outputs, we create a Maxout layer [Goodfellow et al., 2013] with 8 features and use mean squared error (MSE) loss. In the end, the model had 75,000 trainable parameters.

## 3.3 Training

Our implementation was based on Keras [Chollet et al., 2015] using the TensorFlow [Abadi et al., 2015] back-end. We used a batch size of 128, though in practice, changing this did not significantly impact the results. Furthermore, we used the Adam optimizer [Kingma and Ba, 2014] with a learning rate of $3\times10^{-4}$, though RMSProp [Hinton et al., 2012] also worked well.

For each season being predicted, different models were trained. For example, models designed to make 2016 predictions were trained using 1958-2015 data, whereas those making 2017 predictions were trained separately on 1958-2016. We did this so as to give each model as much training data as possible.

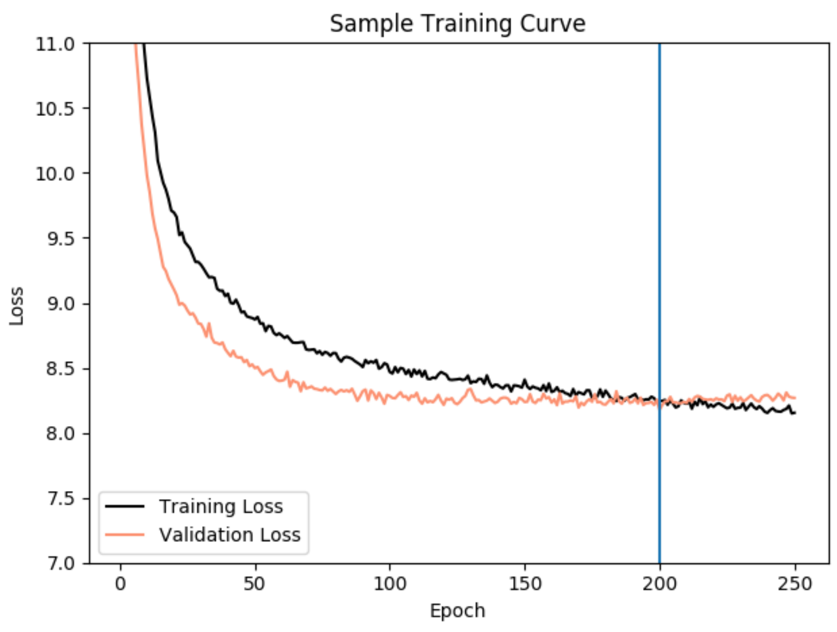


Figure 3.2: A sample training curve. The vertical line represents the epoch with the lowest validation loss.

Of this training data, approximately 80% of the examples were reserved for actual training and the rest were set aside for validation. Each epoch, the training examples were used to update the weights, and the model which performed best on the validation data was chosen. Finally, we implemented early stopping after 50 epochs of no improvement. Typically, this condition was met after about 200 epochs. A sample training curve is shown in Figure 3.2.

## 3.4 Regularization

Because neural networks are high-complexity low-bias models, the bias-variance tradeoff indicates that they will have a high variance. This issue will be magnified due to the small amount of training data available. As such, to have a successful model, we will need heavy regularization.

Our primary form of regularization is Dropout [Srivastava et al., 2014]. In feed-forward networks, dropout functions by applying a random mask from incoming neurons, setting the inputs to 0 with some probability $p$. Since the subsequent layer cannot count on having each individual neuron available, this encourages the previous layers to learn redundancy. This can be interpreted and analyzed as an ensemble method [Hara et al., 2016]. We apply dropout with $p = 0.3$ to the locations in Figure 3.1 marked with an asterisk.

In addition to dropout in the dense layers, we apply dropout inside the GRU cells. We achieve optimal results by applying the same dropout mask to the update gate of each timestep [Gal and Ghahramani, 2016, Semeniuta et al., 2016]. We applied this method to every GRU using $p = 0.5$.

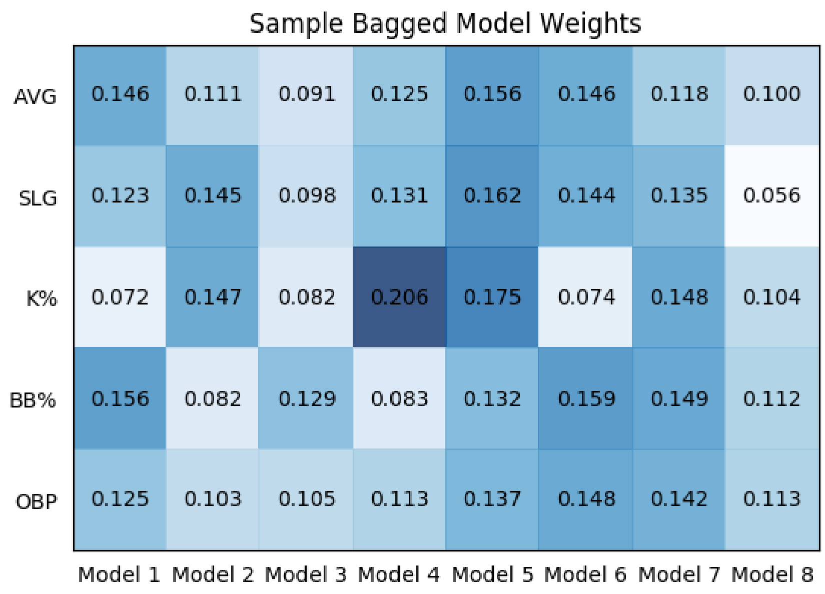
Finally, we used L1 regularization with $\lambda = 10^{-4}$ on the dense layers proceeding *in_out_full_stats*, *in_out_plate_discipline*, and *in_out_running* to encourage sparsity.

## 3.5 Bagging

Even with the regularization techniques discussed above, the model still has a very high variance. This variance is at least partially due to the initialization of the network: two networks trained on the same data can make very different predictions for the same player. [Perrone and Cooper, 1992] suggests applying an ensemble method to reduce the variance of a neural network. Specifically, the authors suggest forming an ensemble that learns a weighted mean of the networks' predictions. For each individual output statistic $s$, let us assume we have trained $n$ models which have predicted a player's performance in that statistic as $s_1 = m_1^{(s)}, \cdots, s_n = m_n^{(s)}$. We want to find weights $\alpha_i, 1 \leq i \leq n$ such that

$$\alpha_1^{(s)}, \cdots, \alpha_n^{(s)} = \underset{\alpha_1^{(s)}, \cdots, \alpha_n^{(s)}}{\operatorname{argmin}} \mathbb{E}\left[\left(\alpha_1^{(s)} m_1^{(s)} + \cdots + \alpha_n^{(s)} m_n^{(s)} - s\right)^2\right]. \tag{3.1}$$

In this problem, we approximated the expected MSE with the MSE on the validation data. This is the same validation set that was used for model selection as discussed in Section 3.3. Ideally, we would have a separate data set for this task, but due to the limited data available, we elected to reuse the same validation set. In the end, we independently trained $n = 8$ networks $m_1, \cdots, m_8$ on the same training data, and finding the model weights for



| | Model 1 | Model 2 | Model 3 | Model 4 | Model 5 | Model 6 | Model 7 | Model 8 |
|---|---|---|---|---|---|---|---|---|
| AVG | 0.146 | 0.111 | 0.091 | 0.125 | 0.156 | 0.146 | 0.118 | 0.100 |
| SLG | 0.123 | 0.145 | 0.098 | 0.131 | 0.162 | 0.144 | 0.135 | 0.056 |
| K% | 0.072 | 0.147 | 0.082 | 0.206 | 0.175 | 0.074 | 0.148 | 0.104 |
| BB% | 0.156 | 0.082 | 0.129 | 0.083 | 0.132 | 0.159 | 0.149 | 0.112 |
| OBP | 0.125 | 0.103 | 0.105 | 0.113 | 0.137 | 0.148 | 0.142 | 0.113 |

Figure 3.3: Values for $\alpha_i^{(s)}$, the weights of the 8 models being aggregated for *out_stats* ($k = 1$), trained using data from 1958-2014. Notice that each model has areas of strength and weakness and some models are overall stronger than others. Note also that the sums of the rows are approximately 1.

each statistic $s$ $\alpha_1^{(s)}, \cdots, \alpha_8^{(s)}$ that minimize the error on the validation set. We also added L2 regularization and a term to encourage (but not constrain) the $\alpha_i^{(s)}$ terms to sum to 1,

$$R = \beta^2 \left[1 - \sum_{i=1}^{n} \alpha_i^{(s)}\right]^2 + (\gamma n)^2 \sum_{i=1}^{n} \left[\alpha_i^{(s)}\right]^2, \beta = 100, \gamma = 2. \tag{3.2}$$

Some values for $\alpha_i^{(s)}$ have been visualized in Figure 3.3.

# Chapter 4: Experiments on Expectation

To evaluate DeepBall's performance in predicting the players' expected performances, we have constructed or obtained baseline systems to which it will be compared. The first baseline to which we will compare DeepBall is Marcel, which was discussed in detail in Section 1.1 and is commonly used in industry as a standard baseline.

Our second baseline is a Random Forest regression. We used the `sklearn` implementation, `RandomForestRegressor` [Pedregosa et al., 2011]. The inputs, outputs, and sample weights were the same as the ones for DeepBall, which were described in Section 3.1. We used 60 random trees, each of which uses a random one-third of the available features. Since our problem deals with sequential data and this cannot be represented in a traditional random forest, we only used data from the previous four timesteps and flattened it, replacing missing values with zeros.

The third and final baseline we will be using is a single recurrent neural network whose simple architecture is laid out in Figure 4.1. We concatenated all inputs at each timestep and passed that vector through two GRUs whose output dimensions are the same as the input dimensions, and each used $p = 0.5$ dropout in the update gate. The final layers and loss functions were the same as DeepBall, with each distinct output layer having its own Maxout layer attached to the last GRU. Similar to DeepBall, we used the validation set for model selection, and we trained 8 models and used the same validation set to compute the optimal weights for each model.

We used data from the 2015-2017 seasons as test data, selecting all position players (non-pitchers) who had at least one plate appearance during the season. The errors were weighted according to each example's sample weight. In total, there were 897 distinct players and 1913 distinct batter/seasons in the test set. We evaluated *out_stats* using mean squared error, mean absolute error, and the linear regression coefficients for each system, and we

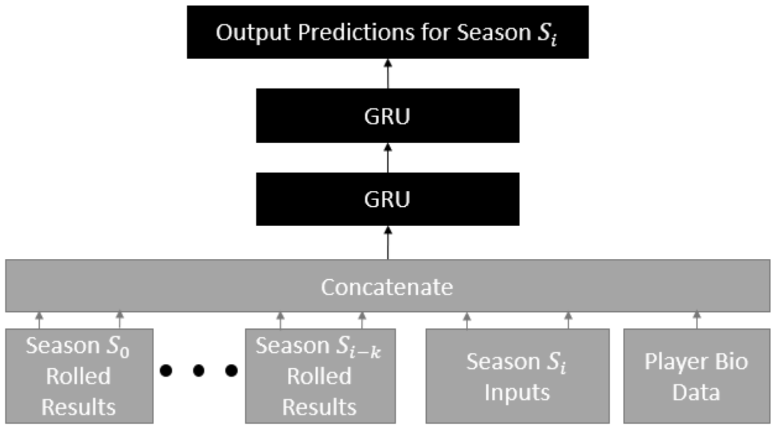

Output Predictions for Season $S_i$
GRU
GRU
Concatenate
Season $S_0$ Rolled Results
Season $S_{i-k}$ Rolled Results
Season $S_i$ Inputs
Player Bio Data

Figure 4.1: The architecture for the baseline RNN model

| **k = 1** | AVG | SLG | K% | BB% | OBP |
|---|---|---|---|---|---|
| Marcel | 0.03268 | 0.04738 | 0.07604 | 0.12824 | 0.02446 |
| RF | 2.49% | 1.97% | -1.44% | -5.93% | 5.06% |
| RNN | 9.24% | 8.78% | 10.81% | 3.83% | 9.73% |
| DeepBall | **9.91%** | **9.04%** | **13.61%** | **5.68%** | **10.97%** |
| **k = 2** | AVG | SLG | K% | BB% | OBP |
| Marcel | 0.03379 | 0.04926 | 0.08791 | 0.14022 | 0.02528 |
| RF | 2.12% | 3.47% | 5.32% | -0.97% | 5.58% |
| RNN | 10.15% | 9.47% | 12.25% | 6.32% | 11.06% |
| DeepBall | **10.58%** | **10.03%** | **13.90%** | **7.56%** | **11.86%** |

Table 4.1: Marcel MSE performance (first row), and MSE relative improvement vs. Marcel (subsequent rows) for $k \in \{1, 2\}$. We observe that DeepBall outperforms Marcel and all other baselines.

evaluated *out_counts* using mean absolute error.

## 4.1 Weighted MSE

The first metric we will use to evaluate our results is weighted mean squared error relative to each system's league average (as described in Section 3.1) for both $k = 1$ and $k = 2$. This metric is useful because it corresponds to the loss function we used to train our model and is the standard loss function for regression analysis. The results are shown in Table 4.1, where the absolute error for Marcel is given and each system's performance is measured as a relative percentage improvement upon Marcel's error.

As we would expect, the error for $k = 2$ is universally larger than that for $k = 1$. We can observe a few more interesting trends from these results:

- *DeepBall outperforms all baselines in all categories.* DeepBall makes an impressive showing, outperforming Marcel by at least 5% in all categories.
- *DeepBall's performance relative to Marcel is nearly identical for $k = 1$ and $k = 2$.* This speaks to the robustness of Marcel, that simply treating one year's predictions as the ground truth and forecasting an additional season is able to keep pace with much more complex models. More importantly, it demonstrates that without any modifications to the network architecture, DeepBall can be used to obtain reasonable, reliable, and generally accurate predictions for multiple seasons in the future.
- *The Random Forest baseline struggles to perform well.* We suspect this is because Random Forests have an inductive bias that similar data can be grouped together using axis-aligned rectangles. However, very little in this problem can be viewed this way. For example, many statistics used in baseball, including all values in *out_stats*, involve division, which cannot be expressed in this way.[1]
- *The rankings are relatively uniform for each statistic and k, with neural models performing the best.* Interestingly, the RNN baseline significantly outperforms Marcel in all categories. One could understand this because the RNN has at its disposal more information and variables than Marcel, especially for rookies. Additionally, in most cases it is within one percent of DeepBall's error, and this also highlights the strength of the neural models which have the ability to properly combine high-dimensional input data to create high-quality predictions. These results also suggest that DeepBall is not sensitive to changes in the architecture of the network, since a network carefully designed using domain knowledge only slightly outperforms a network with a simple

[1]Neural networks also cannot exactly express the division operation, but their flexibility allows them to approximate this.

| **k = 1** | AVG | SLG | K% | BB% | OBP |
|---|---|---|---|---|---|
| Marcel | 0.1207 | 0.1519 | 0.1794 | 0.2581 | 0.1045 |
| RF | -1.54% | -1.07% | -4.99% | -3.15% | -0.86% |
| RNN | **4.65%** | **4.75%** | 5.96% | 2.31% | 4.54% |
| DeepBall | 4.51% | 4.69% | **6.61%** | **3.23%** | **5.00%** |

| **k = 2** | AVG | SLG | K% | BB% | OBP |
|---|---|---|---|---|---|
| Marcel | 0.1243 | 0.1561 | 0.2013 | 0.2746 | 0.1071 |
| RF | -0.55% | -0.15% | -0.55% | -0.86% | 0.36% |
| RNN | **5.70%** | 4.55% | 7.07% | **4.53%** | 6.28% |
| DeepBall | **5.70%** | **4.96%** | **7.53%** | 4.42% | **6.64%** |

Table 4.2: Marcel MAE performance (first row), and MAE relative improvement vs. Marcel (subsequent rows) for $k \in \{1, 2\}$. Once again, DeepBall has the lowest error relative to the other baselines, being slightly outperformed by the RNN in a few statistics.

architecture.

## 4.2 Weighted MAE

The second metric we will use is weighted mean absolute error relative to league average, again for $k = 1$ and $k = 2$. We chose to include this because other analyses in the baseball industry, such as [Druschel, 2016b, Druschel, 2017, Silver, 2007], use MAE as opposed to MSE. Once again, the results are measured relative to Marcel and are shown in Table 4.2.

As we might expect, the errors with MAE are much closer together. In addition, the rankings with MAE are mostly the same as with MSE, with DeepBall being the best in most cases, followed by the RNN, Marcel, and the Random Forest.

## 4.3 Linear Regression Coefficients

For our third experiment, we will perform a least-squares linear regression for each statistic and $k$ with the three baselines and DeepBall and examine the resulting coefficients. This approach has been suggested by [Silver, 2007] as a way to measure a combination of accuracy and uniqueness. The differences between our evaluation and Silver's are that we train our regression using more than one year of data, and we first normalize all systems to the same

| $\mathbf{k=1}$ | AVG | SLG | K% | BB% | OBP |
|---|---|---|---|---|---|
| *Blend* | *10.21%* | *9.67%* | *16.35%* | *6.08%* | *11.10%* |
| Marcel | 0.0004 | 0.0529 | -0.1390 | 0.1265 | -0.0289 |
| RF | 0.0809 | 0.0337 | 0.1551 | 0.0137 | 0.0617 |
| RNN | 0.4248 | **0.5829** | 0.0617 | 0.3390 | 0.4594 |
| DeepBall | **0.4797** | 0.3144 | **0.9419** | **0.5167** | **0.4966** |

| $\mathbf{k=2}$ | AVG | SLG | K% | BB% | OBP |
|---|---|---|---|---|---|
| *Blend* | *11.16%* | *11.50%* | *16.34%* | *7.91%* | *12.51%* |
| Marcel | 0.0313 | 0.0466 | 0.0133 | 0.0109 | 0.0267 |
| RF | 0.0510 | 0.1080 | 0.0944 | 0.0117 | 0.0577 |
| RNN | 0.4474 | 0.3951 | 0.0207 | 0.2766 | **0.5345** |
| DeepBall | **0.4530** | **0.4310** | **0.8985** | **0.6939** | 0.3668 |

Table 4.3: Blend relative MSE improvement vs. Marcel for $k \in \{1, 2\}$ (first row). Blend is implemented as a weighted linear regression of baselines. The regression coefficients are also shown in subsequent rows. DeepBall has the highest coefficient in all but two cases and the Blend system performs better than DeepBall (compare to Table 4.1).

league average, the importance of which was discussed by [Tango, 2007]. The coefficients were obtained by training using the position players in the validation data from 2000-2014. In the best case, we would have a separate dataset for this task, but due to the overall shortage of data in this problem, we reused this validation data despite the fact that DeepBall and the RNN models were selected, bagged, and combined using this same validation set as discussed in Section 3.5. As such, the errors for DeepBall and the RNN on the validation set will be slightly optimistic and the linear weights will be slightly inflated.

In addition to measuring the coefficients, we include the weighted mean squared error relative to Marcel of a new system, Blend, which is a linear blend of the four systems using the given coefficients. The coefficients and Blend relative errors are shown in Table 4.3.

We observe that DeepBall receives the highest weights in all but two categories, and in both cases, the RNN model has the highest weight. We can make additional observations from this experiment.

- *Marcel's weights are small or negligible for all statistics for both $k = 1$ and $k = 2$.* This is an excellent indication that our baselines and DeepBall are able to make robust predictions. Marcel is the simplest system discussed, having the highest bias and lowest variance, and it is very robust. If our systems had a high variance, we would expect the Marcel weights to be larger to reduce this variance. The results indicate the opposite.

- *DeepBall's coefficients are the largest for strikeout rate for both $k = 1$ and $k = 2$.* This is surprising, since the RNN baseline also performed well in this category (see Table 4.1). One possible explanation is that DeepBall and the RNN are learning very similar predictions of strikeout rate but the RNN's predictions contain slightly more noise. In other words, the RNN's predictions may be a redundant, noisy version of DeepBall's. As such, even though its error is still low, DeepBall's predictions are more reliable. Even more surprising, though they are small, the Random Forest baseline has the second-highest coefficients despite the RNN being the more accurate baseline. We interpret this as being the opposite of the case with the RNN: even though its accuracy is lower, it is capturing useful trends that the RNN and DeepBall are not.

- *The Blend system achieves lower MSE relative to Marcel than any individual system.* Since these results were measured on the test set, this indicates that blending the systems using these weights would yield a more robust system than any individual system. This is not surprising because each system has variance, and bagging the results of the systems will reduce this. In addition, as discussed above, different systems may capture different trends, so combining these varying perspectives will decrease the overall error.

## 4.4 MAE for Counts

Finally, we will consider the mean absolute error of *out_counts*. This experiment is simpler than the previous ones, since we did not use sample weights or normalize to the league average. The results, measured relative to Marcel with the exception of oWAR because it is not predicted by Marcel, are shown in Table 4.4.

| **k = 1** | PA | G | oWAR |
|---|---|---|---|
| Marcel | 143.73 | 34.26 | N/A |
| RF | 21.46% | 16.90% | 0.8604 |
| RNN | 27.31% | 20.97% | 0.8297 |
| DeepBall | **29.18%** | **22.37%** | **0.8076** |

| **k = 2** | PA | G | oWAR |
|---|---|---|---|
| Marcel | 167.76 | 38.74 | N/A |
| RF | 21.60% | 16.54% | 0.9267 |
| RNN | 30.77% | 23.51% | 0.8638 |
| DeepBall | **31.91%** | **24.35%** | **0.8531** |

Table 4.4: Marcel MAE performance for *out_counts* (first row), and relative MAE improvement vs. Marcel (subsequent rows). oWAR is not predicted by Marcel, so absolute errors are presented instead of relative improvement. Marcel's error significantly increased, but the systems' rankings remained the same.

From these results, we see that Marcel has significant struggles with predicting playing time as opposed to rate statistics, though an alternate hypothesis for the disparity between Marcel and the other systems is that our systems perform better as opposed to Marcel performing worse. The former is the most likely explanation, however, since Marcel's model of playing time is so simplistic: a linear combination of the previous two years' playing time and a constant. Once again, we generally see the same ranking of errors, with DeepBall performing the best.

In summary, from these experiments, we gather two significant insights. First, we have seen that neural models perform better than both a traditional baseball model, Marcel, and a traditional machine learning regression, the Random Forest. Second, we see that the RNN and DeepBall perform at a similar level, but DeepBall's relative errors are slightly lower than the RNN's. Since DeepBall was designed with domain knowledge and the RNN was not (with the exception of the selection of features, of which all models except Marcel share), we see that domain knowledge is useful in creating a reliable projection system, though even without domain knowledge we can still build a good system.

# Chapter 5: Modeling Uncertainty

While the analyses in Chapter 4 and in the industry at large often seem to make the assumption that predictions are homoscedastic, this assumption does not hold in theory or in practice. In reality, heteroscedasticity is apparent and abundant in many aspects of the problem.

The first cause of heteroscedasticity is that rate statistics with smaller sample sizes will generally have higher error rates. To see this, let us use K% as a representative of all rate statistics without loss of generality. In addition, let us make the assumption that each plate appearance within a season is independent. We can then model the number of strikeouts in a season as a binomial distribution,

$$\text{K} \sim B(n = \text{PA}, p = \text{K\%}), \tag{5.1}$$

where K% is the player's expected strikeout rate and PA is the number of plate appearances for that season. Assume for the sake of argument that PA is known, in which case

$$\text{Var}\left[\text{K}\right] = (\text{PA})(\text{K\%})(1 - \text{K\%}), \text{ and} \tag{5.2}$$

$$\text{StdDev}\left[\hat{\text{K\%}} = \frac{\text{K}}{\text{PA}}\right] \propto \frac{1}{\sqrt{\text{PA}}}, \tag{5.3}$$

where $\hat{\text{K\%}}$ is the observed strikeout rate for that season. Using this reasoning, we expect players with less playing time to have a higher variance and therefore higher error in any rate statistic. This effect has been visualized in Figure 5.1.

A second possible cause for heteroscedasticity is limited information regarding rookies or uncertainty about aging players. For example, we would expect rookies to have a higher variance in possible outcomes, since they have not established themselves at the major league level, one way or the other. Uncertainty would be expected in aging players, too,

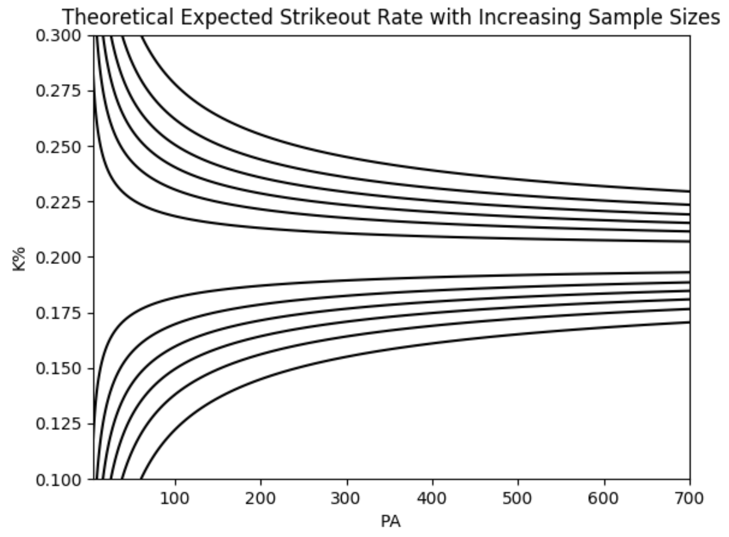


Figure 5.1: If a player's talent level dictates that they will have K% $= 0.2$, we will most likely not observe this exact value. By assuming plate appearances as independent and modeling the result of each as a Bernoulli process, this contour plot of probabilities marginalized over PA shows that estimated variances vary proportional to $\frac{1}{\text{PA}}$. (This plot was generated using a Gaussian approximation to the binomial distribution, which exhibits the desired qualitative features for approximately PA $> 30$.)

partly because there are fewer training examples for older players, and partly because their role on their team may become more uncertain. For these reasons, ZiPS only uses three years of data for very old or very young batters [Druschel, 2016a].

Third, unexpected results from previous seasons may cause uncertainty in future predictions. If a system observes a normally good player have a bad season or vice versa, it may predict either a recovery or another below-average (or above-average) season. Both are justified, but now may be less certain than we were in previous years than we would be when projecting a normally stable player.

Finally, we acknowledge some statistics and therefore errors are correlated, and understanding these correlations is useful. Notably, AVG and SLG are correlated, since according to Equations 1.1 and 1.2, SLG is simply a weighted version of AVG. This correlation is visualized in Figure 5.2. However, these are not the only correlated statistics: OBP and BB% are very correlated, as are PA and oWAR. We also expect rate statistics to be correlated

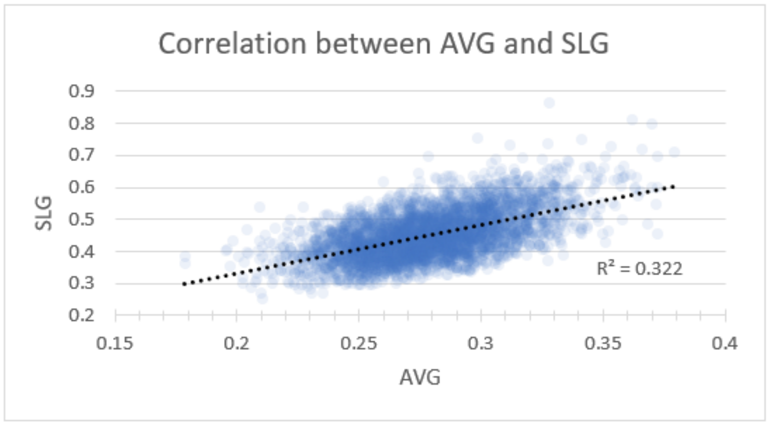


Figure 5.2: The batting average (AVG) and slugging percentage (SLG) for batter seasons since 1990 with at least 500 PA. It is apparent that these two statistics are correlated, so an error in one may suggest an error in the other.

with playing time because better players often receive more playing time. Furthermore, we may observe heteroscedasticity because these correlations may change as a player progresses throughout his career. (This effect is observed and discussed in Section 6.3.)

For all these reasons, we expect the distribution of possible outcomes to be heteroscedastic with respect to the player, season, and statistic. In other words, some errors may be expected, but some errors will be unexpected. With this in mind, it is logical to seek to model not only the expected outcomes, but the expected error in these predictions. Doing so would have many uses. In the context of team front offices, knowing the uncertainty associated with a player's prediction would provide valuable insight if a team is investigating a possible trade: more competitive teams may be more likely to look for a player whose talent is well-established, whereas less competitive teams may be more willing to take a risk by trading for a player with a higher upside but a lower downside as well. For fans, understanding uncertainty will be useful when building a fantasy roster or tracking a player's career over time.

Because of the usefulness of estimating uncertainty, a projection system should not only be able to give an expected outcome but a measure of uncertainty as well. To this end, Marcel and Steamer give reliability scores, where a high value indicates high confidence or low uncertainty. Similarly, ZiPS, in addition to expected results, gives thresholds in many areas along with probabilities that a player will exceed these thresholds. Likewise, we would like DeepBall to predict uncertainty.

Unlike other systems, we would like DeepBall to predict a fully-defined probability distribution of a player's possible outcomes for a season. This would have many benefits. First, this is the most general approach, and any of the aforementioned uncertainty metrics could be derived from this distribution. This would allow DeepBall to be easily compared to the other systems. Second, it would yield the most interesting insights regarding a player's range of likely outcomes. For example, depending on our choice of distribution, we have the ability to represent a player with a bimodal distribution of expected outcomes. This distribution can then be visualized, allowing for baseball analysts to obtain a deeper understanding of what to expect from a player. Third, it yields a natural solution to the outlier detection problem: we can compute the likelihood of the actual results under the distribution of expected results. We can use this as a concrete metric of the unusualness of a batter's season. Fourth, it allows us to compare the predictions of batter seasons not simply by their expected outcome but by their variance and skewness as well.

## 5.1 Approach

In the previous chapters, we have seen how the neural network architecture used by DeepBall outperforms baselines by a wide margin. Furthermore, the model architecture in the final layers is flexible: it currently uses a fixed-length representation to generate first-order predictions for rate and count statistics. As such, we propose adding a new output layer dedicated to making second-order predictions as well. That is, for each output $s_i$, rather than simply outputting $\mathbb{E}\left[s_i\right]$, we will predict both $\mathbb{E}\left[s_i\right]$ and $\mathrm{Var}\left[s_i\right]$. This gives DeepBall the flexibility to reflect some of the heteroscedasticity discussed above. Furthermore, to account for the correlations among output statistics, we can also predict the covariance $\mathrm{Cov}\left[s_i, s_j\right]$ between any outputs $s_i$ and $s_j$.

## 5.2 Single Gaussian Loss Function

Based on these requirements, a natural first step would be to model the outputs as a multivariate Gaussian. Let

$$\mathbf{v} = [\text{AVG}, \text{SLG}, \text{K\%}, \text{BB\%}, \text{OBP}, \text{PA}, \text{G}, \text{oWAR}] \sim \mathcal{N}(\mu, \Sigma), \tag{5.4}$$

where the entries of $\mu$ and $\Sigma$ will be predicted by the model at each timestep. To train our model, we can use the negative log likelihood of this distribution as our loss,

$$L_G(\mu, \Sigma^{-1}; \mathbf{v}) = \frac{d}{2}\log 2\pi - \frac{1}{2}\log\det\Sigma^{-1} + \frac{1}{2}(\mathbf{v}-\mu)^\top\Sigma^{-1}(\mathbf{v}-\mu), \tag{5.5}$$

where $d = 8$ and $\mathbf{v}$ are the observed values. When formulated in this way, it becomes apparent that it is best to return the pair $(\mu, \Sigma^{-1})$ rather than $(\mu, \Sigma)$.

However, if we were to implement a network that directly predicted the entries of $(\mu, \Sigma^{-1})$, we will experience issues if $\Sigma$ is not symmetric and positive semi-definite, which would cause $\det\Sigma^{-1} < 0$. As a workaround, we introduce a new matrix $A$ such that

$$\Sigma^{-1} = A^\top A + \xi I_d. \tag{5.6}$$

It can be seen that regardless of the value of the entries of $A$, $\Sigma^{-1}$ is symmetric. Due to numerical errors, though, the matrix may not be positive semi-definite, so we add $\xi I_d$. The larger we let $\xi$ be, we are less likely to see invalid values of $\Sigma^{-1}$, with the trade-off being that we are enforcing minimum values along the diagonal of $\Sigma^{-1}$. In our work, we used $\xi = 1$ and we never observed an invalid $\Sigma^{-1}$ during training, though this is a hyperparameter and its value may significantly depend on the task.[1] Under this new expression, it is sufficient

[1]The most general value is $\xi = \max(0, -\lambda_d + \epsilon)$, where $\epsilon > 0$ and $\lambda_d$ is the smallest eigenvalue of $A^\top A$. However, evaluating this at every timestep had detrimental effects on the training time, so we elected to use a constant value for $\xi$.

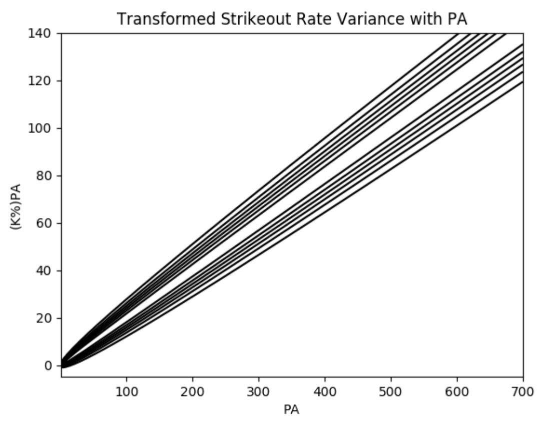
to directly predict the entries in $(\mu, A)$ since there are no constraints on the values of either.

## 5.3 Sample Weights and Reliability

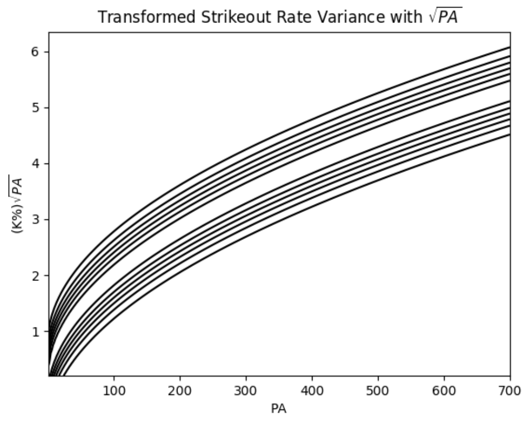
The simplified approach of modeling Equation 5.4 as a Gaussian may work for players with significant playing time, but as seen in Figure 5.1, the reliability of the rate statistics varies proportional to $\frac{1}{\sqrt{\text{PA}}}$. As a result, we will observe poorly-behaved values for the rate statistics in when PA is small. In Section 3.1 we solved this by applying sample weights to all rate statistics but not count statistics, but this only worked because *out_stats* and *out_counts* were treated as separate outputs with their own loss functions. However, it is not clear how to apply sample weights to only some components of a multivariate Gaussian distribution, and since we are also interested in measuring the covariance between playing time and rate statistics, we cannot separate these into two Gaussians.

At least three solutions to this problem exist. First, we could apply sample weights to all variables in the Gaussian. This would cause our model to be biased towards predicting higher playing time, since we would be using PA as a weight for its own predictions. Second, we could ignore this issue entirely. On its surface, this solution is more plausible. However, the regions of highest likelihood seen in Figure 5.1 are not conducive to being modeled with a single Gaussian distribution, especially for small values of PA. Because of this, outcomes such as one strikeout in one or two PA, while not inconceivable, will result in an extremely low likelihood score.

We propose a third solution: transform the data to make it more conducive to being modeled with a well-behaved distribution. We will design our transformation in such a way that similar outcomes will be grouped together, unlike what we see in Figure 5.1 where smaller sample sizes lead to similar outcomes being spaced further apart. To this end, our



(a) $w = f(\text{PA}) = \text{PA}$. Here, we can see that $\text{Var}\left[(\text{PA})(\text{K\%})\right] \to 0$ as $\text{PA} \to 0$.



(b) $w = f(\text{PA}) = \sqrt{\text{PA}}$. Here, $\text{Var}\left[(\text{PA})(\text{K\%})\right]$ is constant for all PA.

Figure 5.3: We compare two possible transformations to the rate statistics by multiplying them by $w = f(\text{PA})$. We assume that $\mathbb{E}\left[\text{K\%}\right] = 0.2$.

transformation will take the form of

$$\mathbf{s} = \left[(w)\text{AVG}, (w)\text{SLG}, (w)\text{K\%}, (w)\text{BB\%}, (w)\text{OBP}, \text{PA}, \text{G}, \text{oWAR}\right], \tag{5.7}$$

where $w = f(\text{PA})$ is a weight factor calculated as a monotonically-increasing function of playing time. We consider two possible values for $w$: $w = \text{PA}$ and $w = \sqrt{\text{PA}}$. Similar to Figure 5.1, the distribution of theoretical outcomes when accounting only for empirical noise is plotted in Figure 5.3.

We should choose our function for $w$ based on how easily the resulting distribution can be modeled with a Gaussian. In this regard, each option has benefits and drawbacks. For $w = \text{PA}$ (Figure 5.3a), we see that the mean of the distribution is linear with respect to the playing time, so any local Gaussian approximation will have the same correlation coefficient between PA and $w(\text{K\%})$. This can be useful to interpret the distributions, because a higher slope of the distribution corresponds to a higher value in that rate statistic. However, the distribution still exhibits heteroscedastic behavior since the variance increases linearly as PA increases. For $w = \sqrt{\text{PA}}$ (Figure 5.3b), the distribution is homoscedastic, but that comes at the expense of the mean not being linear with respect to PA.

Therefore, in their current forms, neither of these transformations yields a distribution that is able to be perfectly modeled with a single Gaussian distribution, and both options are justified. In our work, we used $w = \text{PA}$. Under this transformation, we see that $\text{Var}\left[(\text{PA})(\text{K\%})\right] \to 0$ rather than $\infty$ as $\text{PA} \to 0$. Additionally, this joint distribution is much easier to model with a single Gaussian than the original.

However, regardless of our choice of $w$, there is a side-effect of this transformation: the value of any rate statistic (the values we are interested in modeling), for example, K%, at any point is now the same as the slope of the line passing through that point and the origin. In other words, K% can be modeled as the quotient of two correlated, non-centered, normally-distributed variables. In the case of $w = \text{PA}$,

$$\text{K\%} \sim \frac{(\text{PA})(\text{K\%}) \sim \mathcal{N}(\cdot,\cdot)}{\text{PA} \sim \mathcal{N}(\cdot,\cdot)}. \tag{5.8}$$

This is a ratio distribution, and it is difficult to model analytically,[2] so we elected to study it via Monte Carlo sampling. When sampling from this distribution, it is important to weight each sample by $\sqrt{\text{PA}}$ like in the previous parts to maintain consistency when comparing these results with those obtained in previous sections. This step becomes significant when the values of the rate statistics are correlated with PA, as seen in Section 6.3.

## 5.4 Mixture of Gaussians

The transformation described above, however, does not adequately model the baseball prediction problem because single Gaussians are symmetric, which is not an appropriate assumption for baseball data, especially playing time. For example, DeepBall may predict a player's games played (G) to have $\mu = 130, \sigma = 15$. However, it is much more likely

[2]If we had chosen $w = \sqrt{\text{PA}}$, this would be even more complicated since it would require $\sqrt{\text{PA}}$ rather than PA in the denominator, and analytically modeling the square root of a normally-distributed variable would add additional difficulties.

that the player plays in $\mu - 3\sigma = 85$ games, perhaps because of an injury, than that he plays in $\mu + 3\sigma = 175$ games, because today's MLB seasons are only 162 games long. In the extreme case, suppose this player suffers a major injury at the beginning of the season and only plays in 25 games (unlikely but not impossible). He will be 7 standard deviations from his mean expected performance, a result for which nearly no probability mass has been reserved. Clearly, we need a new model that can account for such situations.

To resolve these issues, we direct our attention to predicting a mixture of $n$ Gaussians rather than a single Gaussian. In other words, we use Equation 5.5 and apply the loss function

$$\begin{aligned} L_{MOG}(\mu_1, \Sigma_1^{-1}, w_1, \cdots, \mu_n, \Sigma_n^{-1}, w_n; \mathbf{s}) &= -\log \sum_{i=1}^{n} w_i \exp\left(-L_G(\mu_i, \Sigma_i^{-1}; \mathbf{s})\right) \\ &= -\log \sum_{i=1}^{n} \exp\left(\log w_i - L_G(\mu_i, \Sigma_i^{-1}; \mathbf{s})\right) \end{aligned} \tag{5.9}$$

where $w_i \geq 0$ and $\sum_{i=1}^{n} w_i = 1$, which was ensured by first applying the softmax function to all $w_i$. Additionally, we applied the same transformations discussed in Section 5.2 so that we predict $A_i$ rather than $\Sigma_i^{-1}$.

In essence, what we have proposed is nothing new to the deep learning community: we are estimating the maximum-likelihood parameters of a probability distribution from training data. This consists of choosing a probability distribution and mapping the output of our network, $\mathbf{x} \in \mathbb{R}^n$, onto the space of valid parameters for this distribution using an activation function $\mathbf{y} = g(\mathbf{x})$. After this, we can use the negative log likelihood of the observed value under our distribution and current parameter estimation as the loss function. For binary classification tasks, we use the Bernoulli distribution along with the sigmoid activation $g(x) : \mathbb{R} \ni x \mapsto y \in [0, 1]$ and logarithmic loss. For multiclass problems, we model our output as a categorical distribution, requiring the softmax activation and categorical cross-entropy loss. Likewise, we can consider Equation 5.6 as our activation function and Equation 5.9 as our loss function.

## 5.5 Training

We created separate models for $n \in \{1, 2, 4, 8, 16\}$ using the same architecture described in Section 3.2, replacing the original loss functions with a mixture of Gaussian loss (Equation 5.9). These models used 1958-2014 as training data. When training the new models, we observed stability issues for small batch sizes, the effects of which worsened as $n$ increased. To combat this, we used a larger batch size by splitting our training dataset into 8 batches, each containing approximately 800 sequences. The rest of the training hyperparameters remained the same as in Section 3.3. Sample training curves for the mixture of Gaussian loss are given in Figure 5.4.

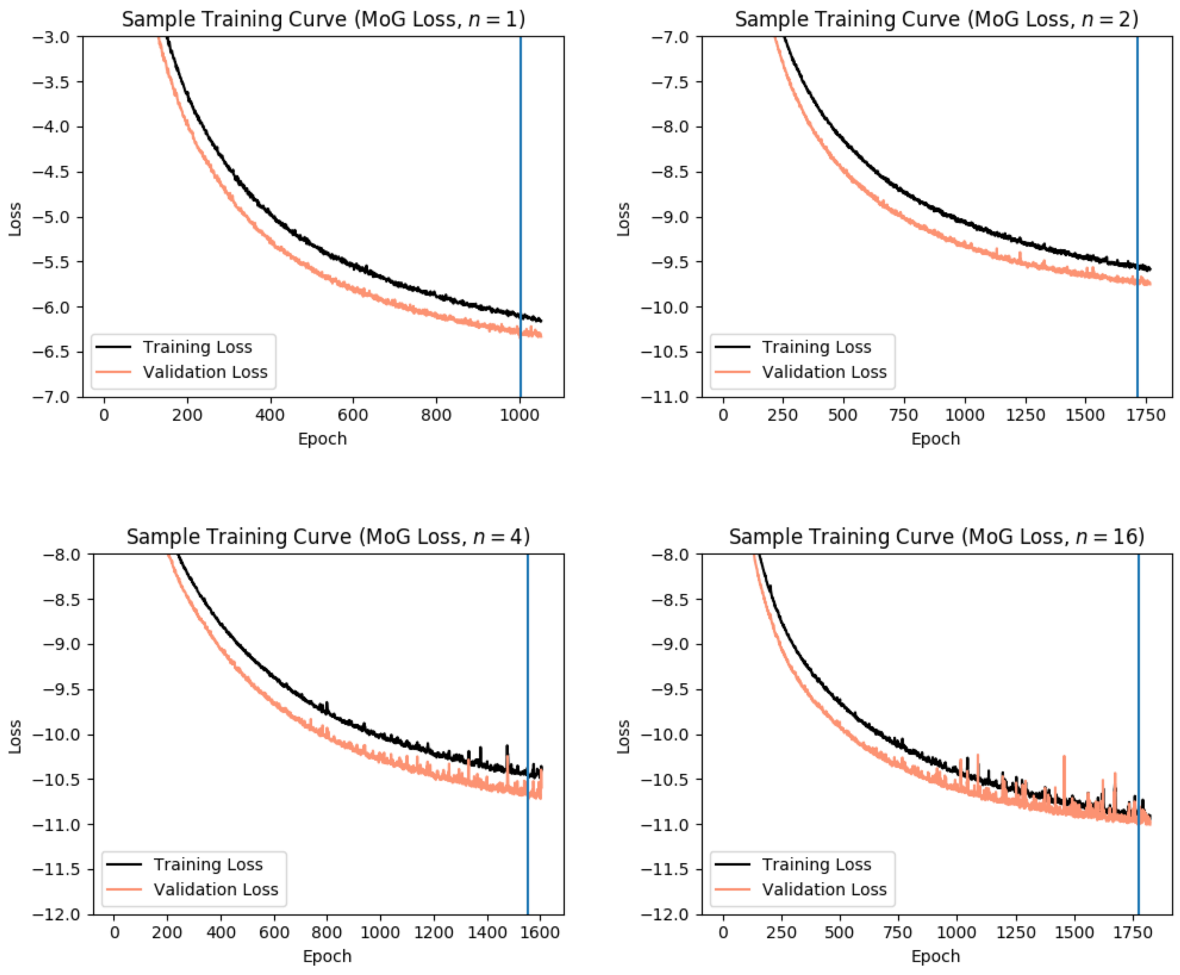


Figure 5.4: Four training curves for the mixture of Gaussian loss using $n \in \{1, 2, 4, 16\}$. The vertical line represents the epoch with the lowest validation loss. Note the shift in the loss axis between the figures, and that the loss axis is negative, indicating a positive likelihood. We observe stability issues as $n$ increases, but they were not detrimental to the training process.

# Chapter 6: Experiments on Uncertainty

In this chapter, we show four ways of analyzing the distribution predictions. Quantitatively, we evaluate the weighted mean squared error of the sampled means of the distributions similar to Section 4.1 and we evaluate the log likelihoods of the ground truths over the predicted distributions. We also evaluate the results qualitatively, gleaning information and insights from anecdotal player distributions predicted by the model. In doing so, we observe the effects of heteroscedasticity and see how DeepBall is able to account for this.

To evaluate our method, we evaluated our models with 2015-2017 data. For each value of $n$, we trained four models and selected the one with the lowest validation loss (highest average likelihood on the validation set). All analyses are done with $k = 1$, but similar to the original model, we can trivially extend this for $k > 1$ by further shifting our outputs and retraining the models. For brevity, we have not analyzed this functionality, though we expect the results similar to those in Chapter 4.

## 6.1 Mean Squared Error

First, we seek to quantitatively evaluate the results of our model. A starting point is to examine the means of the predicted distributions and compute the mean of the squared error between these means and the actual results. This comparison technique is the same as that described in Section 4.1.

To compare the mean rate statistic predictions from the mixture of Gaussian distributions with the results from Chapter 4, we must find the mean values in the original space instead of the transformed space. To do this, we obtained the means of our mixture of Gaussians model by sampling from Equation 5.8 and weighting the sampled outcomes by the square root of the sampled PA. The latter step is necessary because most rate statistics are correlated with playing time as hypothesized in Section 5 and seen in Section 6.3. In addition, similar to

| **k = 1** | AVG | SLG | K% | BB% | OBP |
|---|---|---|---|---|---|
| Marcel | 0.03268 | 0.04738 | 0.07604 | 0.12824 | 0.02446 |
| *DeepBall+* | *10.33%* | *11.00%* | *17.22%* | *6.48%* | *11.41%* |
| RF | 2.49% | 1.97% | -1.44% | -5.93% | 5.06% |
| RNN | 9.24% | 8.78% | 10.81% | 3.83% | 9.73% |
| DeepBall | **9.91%** | 9.04% | 13.61% | **5.68%** | 10.97% |
| MOG ($n = 1$) | 7.10% | 6.70% | 11.57% | 0.02% | 8.59% |
| MOG ($n = 2$) | 9.57% | 10.33% | 13.26% | 3.48% | **11.29%** |
| MOG ($n = 4$) | 9.28% | 10.33% | **14.94%** | 4.63% | 10.77% |
| MOG ($n = 8$) | 8.99% | 10.43% | 12.51% | 4.67% | 10.42% |
| MOG ($n = 16$) | 9.34% | **10.56%** | 14.07% | 5.02% | 11.09% |

Table 6.1: MSE relative improvement for our baselines, original DeepBall model, and the means of the mixture of Gaussian models for $n \in \{1, 2, 4, 8, 16\}$. For $n > 1$, the Gaussian means have approximately the same error as the model trained using MSE loss. DeepBall+ has been implemented as a linear combination of the systems listed, and it outperforms each individual model and the Blend model shown in Table 4.3.

Section 4.3, we have computed a linear regression between the original system, the RNN baseline, and the mixture of Gaussians systems for $n \in \{2, 4, 8, 16\}$. This regression was trained using all player/seasons in the validation set from 2000 to 2014. This is different from Blend in Table 4.3, which was trained only on position players because it included Marcel in the regression whose predictions are only available for position players. The results are shown in Table 6.1.

From these results, we observe that for most statistics, any $n > 1$ gives results comparable to DeepBall, and in some statistics, the sampled means from the mixture of Gaussians model outperform DeepBall. The $n = 1$ model is the exception, whose error rates are slightly higher than those of the RNN baseline in most cases. This supports our hypothesis that using $n = 1$ Gaussian is too simplistic and cannot accurately represent the data distribution.

Additionally, similar to Section 4.3, we see that a linear combination of these systems has the lowest error in all statistics compared to any single model. Furthermore, DeepBall+ has lower error than the Blend system for $k = 1$. We can interpret this in two ways. First, we can say the variance of each individual model is high, and though Blend reduces some of the variance, DeepBall+ reduces it further. Second, each system or method may have

| **All** | $n = 1$ | $n = 2$ | $n = 4$ | $n = 8$ | $n = 16$ |
|---|---|---|---|---|---|
| 2015 | 9.65 | 10.32 | 11.54 | 11.75 | 11.71 |
| 2016 | 9.80 | 10.49 | 11.77 | 11.96 | 11.93 |
| 2017 | 9.03 | 9.73 | 10.90 | 11.06 | 11.08 |
| *All* | 9.50 | 10.19 | 11.41 | 11.60 | 11.58 |
| **Pos.** | $n = 1$ | $n = 2$ | $n = 4$ | $n = 8$ | $n = 16$ |
| 2015 | 4.34 | 5.08 | 5.91 | 6.02 | 5.99 |
| 2016 | 4.38 | 5.18 | 6.09 | 6.07 | 6.10 |
| 2017 | 4.53 | 5.38 | 6.20 | 6.24 | 6.31 |
| *All* | 4.42 | 5.21 | 6.07 | 6.11 | 6.14 |

Table 6.2: Average log likelihood for all players (top) and position players (bottom) on the seasons reserved for testing using $k = 1$ and $n \in \{1, 2, 4, 8, 16\}$ Gaussians.

different strengths (for example, the mixture of Gaussian models tend to do much better than DeepBall in predicting SLG), and the results improve once we factor these into the linear regression. Either way, in terms of weighted mean squared error, it is clear that DeepBall+ is the most accurate, robust system discussed in this paper.

## 6.2 Average Log Likelihood

Using mean squared error of the means of the distributions is not sufficient since it discards the second-order predictions. Since second-order predictions are the main contribution from this approach, we would like a way to quantitatively evaluate results these as well. To this end, we will evaluate the average log likelihood of the ground truth under our predicted distribution on the test data for all players and position players. Unfortunately, unlike the experiments in Chapter 4, we do not have an industry-standard baseline to which our results can be compared. As such, we will compare the results for different values of $n$ for all players and again specifically for position players. The results are given in Table 6.2.

There are a few interpretations we can draw from the average log likelihood results. First, there is a large disparity in the overall log likelihoods when pitchers were removed. This is understandable, since pitchers in general are easier to predict in the transformed space.

For example, most pitchers will have less than 50 PA, and since the rest of the variables in the distribution vary proportionally to PA, the pitcher outcomes are clustered near the origin. Furthermore, pitchers are generally poor hitters with few exceptions, and this will make predictions appear to be more accurate.

Second, as hypothesized in Section 5.4, it is clear that using $n = 1$ is not sufficient to adequately capture the distribution of possible outcomes. We can see this because the average log likelihood for $n = 1$ is much lower than those for $n > 1$. The log likelihood increases until $n = 8$, at which point it appears to plateau. Because of this, it is unclear whether $n = 8$ or $n = 16$ yields the best results. Therefore, we chose to apply Occam's razor, using $n = 8$ for the remainder of this paper when evaluating our results.

## 6.3 Season Case Studies

Beyond the quantitative analysis described above, we seek to evaluate our predicted distributions qualitatively. More specifically, by examining interesting anecdotal predictions, we can gain further insights into our model's behavior in certain cases. To do so, we will examine contour plots of the players' 2018 predicted distributions. We selected a few representative players and visualized the probability density function for $n = 8$ Gaussians. Because it is impossible to visualize an 8-dimensional distribution, we have selected two pairs of representative statistics to visualize.

The first pair we chose is PA and oWAR. This is perhaps the most interesting choice available for multiple reasons. First, both stats will vary widely between different players, so it is a useful tool to compare them. Second, oWAR is a statistic that combines many aspects of a player's performance, so it carries more information regarding a player's talent than any individual rate statistic. Third, PA and the magnitude of oWAR are correlated since the more playing time a player has, the more opportunities he has to contribute value to his team. Because of this, for example, a prediction of 2 oWAR in 400 PA is much more significant than 2 oWAR in 700 PA. For this reason, it is useful to examine both oWAR and

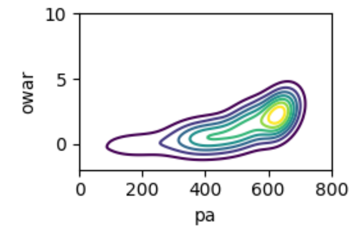
PA together.

The second pair we have selected is PA and (K%)(PA).[1] We have chosen this in part to demonstrate that the distribution of these two variables (if marginalized over PA) loosely follows the theoretical distribution in Figure 5.3a. As we would expect, it does not follow this distribution exactly, and this fact demonstrates some important qualities in the data. In order to grasp these differences further, we will examine the distribution of PA and (K%). Like the previous experiments, it is difficult to analyze this by directly obtaining a probability density function, so we use Monte Carlo sampling to obtain an approximation to this distribution. The visualizations for the 2018 predictions for Jason Heyward, Mike Trout, and Matt Olson are given in Figure 6.1.

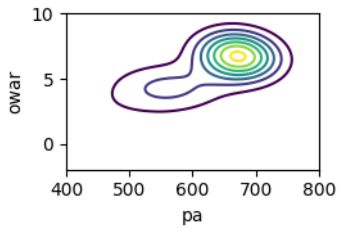
Each of these players has a unique past and an equally-unique 2018 outlook, and these outlooks manifest themselves in the distributions. Jason Heyward began his career as a rookie sensation but has increasingly struggled as his career has progressed. His struggles in 2016 and 2017 have been so notable that DeepBall projects him to receive only 489 PA in 2018, less than the full-time load of 500 PA. Interestingly, we see in Figure 6.1g that his strikeout rate and playing time are correlated negatively. Judging by these results, after his struggles, Jason Heyward needs to prove that he can return to his original form in order to receive more playing time. The lower his strikeout rate, the better his overall performance will likely be and the more playing time he will receive.

Mike Trout, on the other hand, has been heralded as one of the best players of his generation, winning the American League Rookie of the Year along with the AL MVP twice. In his last six seasons, he has never posted an oWAR below 7.3. As seen from the graphs, his oWAR predictions are very high and he is easily projected to be a full-time player. Furthermore, compared with Jason Heyward, Mike Trout's strikeout rate and plate appearances are not as correlated, since established players will generally be given

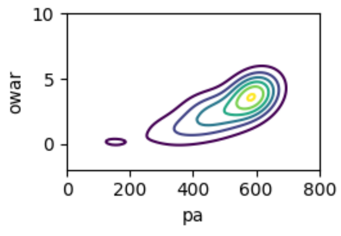
[1]Note that all references to K% in this section are relative to the league average as discussed in Section 3.1.



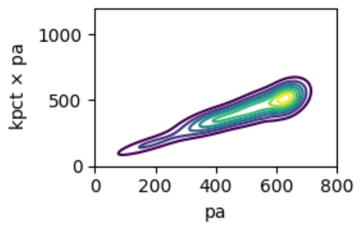
(a) Heyward, oWAR



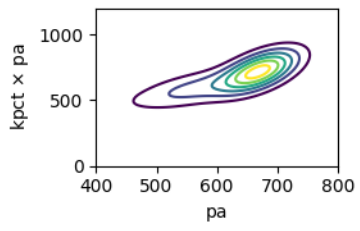
(b) Trout, oWAR



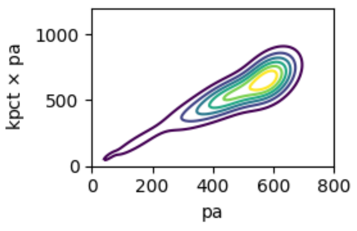
(c) Olson, oWAR



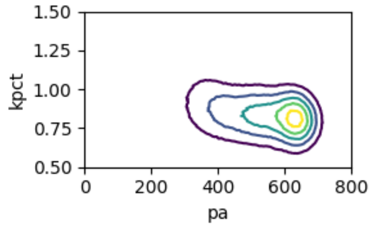
(d) Heyward, (K%)(PA)



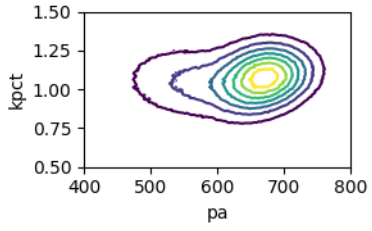
(e) Trout, (K%)(PA)



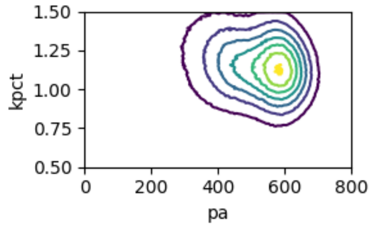
(f) Olson, (K%)(PA)



(g) Heyward, K%



(h) Trout, K%



(i) Olson, K%

Figure 6.1: Visualizations of 2018 prediction distributions for Jason Heyward, Mike Trout, and Matt Olson using $n = 8$ Gaussians. Each player has a different outlook, so the distributions vary both qualitatively and quantitatively. In addition, the plots of PA vs. (K%)(PA) are qualitatively similar to Figure 5.3a.

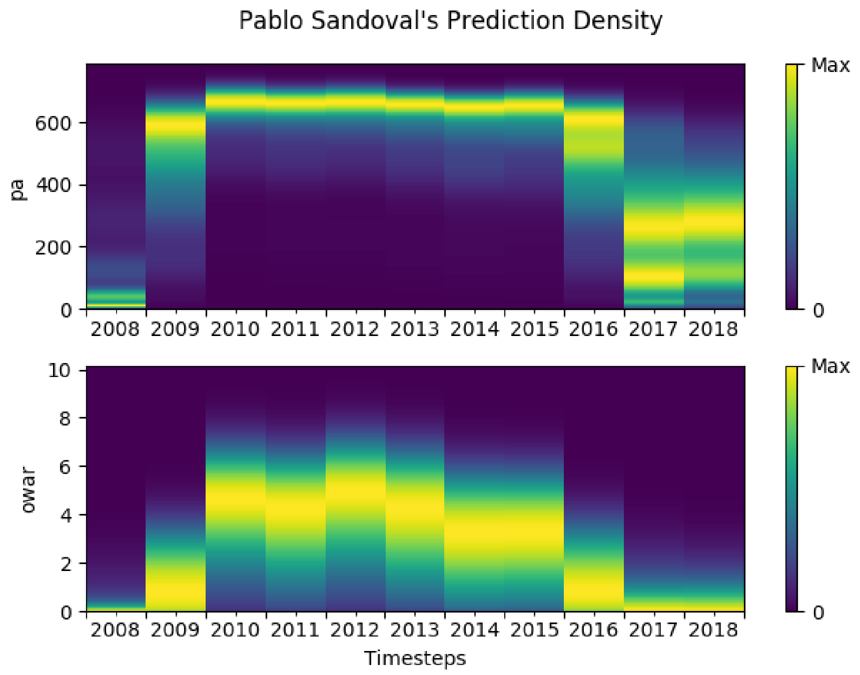
a full-time load regardless of their performance. Additionally, despite his high expected plate appearances, our model learns a skew distribution over PA, reserving some probability density for less playing time which could happen if an injury occurs.

Finally, Matt Olson, after reaching the Major Leagues and playing in only 11 games in 2016, played in 59 games in 2017, hitting a notable 24 home runs and contributing 2.1 oWAR in only 216 PA, finishing fourth in Rookie of the Year voting. Despite his youth and the associated uncertainty, DeepBall's projected distribution expects him to be nearly a full-time player and contribute 2.63 oWAR in 2018. However, it displays more uncertainty than Jason Heyward's projection because little is known about Matt Olson. Like for Heyward, Figure 6.1i suggests a slight negative correlation between playing time and strikeout rate, which we attribute to the same cause, that Matt Olson needs to prove himself at the Major League level.

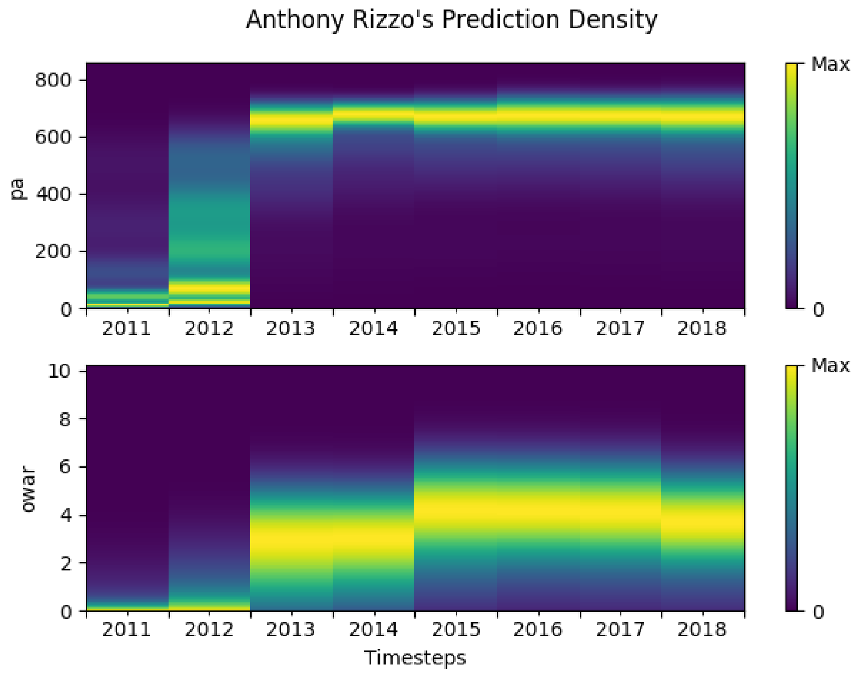
## 6.4 Career Case Studies

In addition to visualizing the relationship between predicted statistics within the same year, we can visualize the progression of player predictions throughout a player's entire career. For this, we examine the predictions for Pablo Sandoval and Anthony Rizzo, obtained using $n = 8$ Gaussians, which can be found in Figure 6.2. While both of these players' projections through 2014 were a part of the training dataset, we have nevertheless analyzed them because they demonstrate trends that appear in general throughout the system.

Looking at Pablo Sandoval's density plot over time (given in Figure 6.2a), we can segment his career into three sections: 2008-2009, 2010-2015, and 2016-present. At the beginning of his career, his projections are similar to any other rookie or young player, though DeepBall correctly predicted him to be a full-time player by his second year: though he only had 154 PA his rookie year, he was very successful and earned a full-time role by 2009, receiving 633 PA. His projections in these categories remained mostly the same between 2010 and 2015, with his expected value fluctuating between 3.5 and 5 oWAR. However, he struggled in 2015,



(a) Pablo Sandoval's prediction density for PA and oWAR



(b) Anthony Rizzo's prediction density for PA and oWAR

Figure 6.2: We have visualized the normalized preseason prediction distributions for Pablo Sandoval and Anthony Rizzo throughout their respective careers using $n = 8$ Gaussians. We can see the uncertainty in playing time and player value at the beginning of a career or, in the former case, after an injury.

posting career lows in OBP and SLG and a career high in K%. Even though he received more playing time in 2015 than he did in 2011 and 2012, DeepBall began to question his full-time status (note the long tail in the PA distribution) and dropped its expected oWAR prediction to near its 2009 level. To make matters worse, he suffered a season-ending injury at the beginning of the 2016 season, his offensive production did not recover in 2017 (as correctly predicted). Interestingly, DeepBall predicted a trimodal distribution for Sandoval's playing time in 2017, and his season of 279 PA fell into the top mode. Given his history, DeepBall yields pessimistic estimates on Pablo Sandoval's upcoming season.

Examining the same plot for Anthony Rizzo given in Figure 6.2b, we see a similar prediction for his rookie year but a much more pessimistic second-year prediction, where the mode of his PA prediction was under 100. This is not surprising, since Rizzo struggled in his 153 PA in 2011. In 2012, however, he had 368 PA with a good OBP and below-average K%. Given this information, DeepBall projected Rizzo to move to a full-time player in 2013, which he did. While he only posted 1.04 oWAR in 2013, DeepBall remained optimistic, maintaining this same projection for 2014. He lived up to this projection in 2014 with 4.6 oWAR, after which DeepBall shifted its projections up for 2015-2017 as he continued to perform well. While his 2017 season was still excellent, some of his statistics, such as SLG and oWAR, decreased, perhaps explaining the slight drop in oWAR predictions for 2018. (It should be noted, however, that judging by the error bounds, this drop is still within the margin of error.)

# Chapter 7: Applications

We have seen how DeepBall is able to tackle two critical problems in the baseball industry. Using surface-level, noisy statistics from the past, DeepBall generated reliable predictions for expected performance for any player for one or more future seasons. In this task, it outperformed both traditional machine learning algorithms and the industry-standard baseline, Marcel, in every category examined. Additionally, we have shown the flexibility of DeepBall by developing a new activation and loss function to estimate a complex probability distribution over possible outcomes for a player's future season.

It is not possible to enumerate all the applications for the methods which we have proposed. Nevertheless, we give some potential applications that we believe are interesting, useful, or both.

## 7.1 Park-Neutral Expectation Evaluation

The most natural application to what we have discussed is predicting the expected performance of players. However, we recognize that our network is simply a regression combining the results from the player's previous seasons and inputs for the upcoming season. By adjusting the inputs for the incoming season, we can simulate how the players will perform in the upcoming season under different circumstances given their past experience. Specifically, we can adjust the values for *in_park*, *in_team_flags*, *in_position*, *in_age*, or *in_bio* to generate new predictions. Using this method, we can study the anecdotal effects of any of these variables by changing them and comparing the resulting expectations or distributions.

Of particular interest is the *in_park* input, which if set to 0's will cause the model to make a prediction for the upcoming season in a park-neutral environment. We have done this for Charlie Blackmon, the center fielder for the Colorado Rockies. The Colorado Rockies play in Denver, where the high altitude causes the ball to carry further. For this reason,

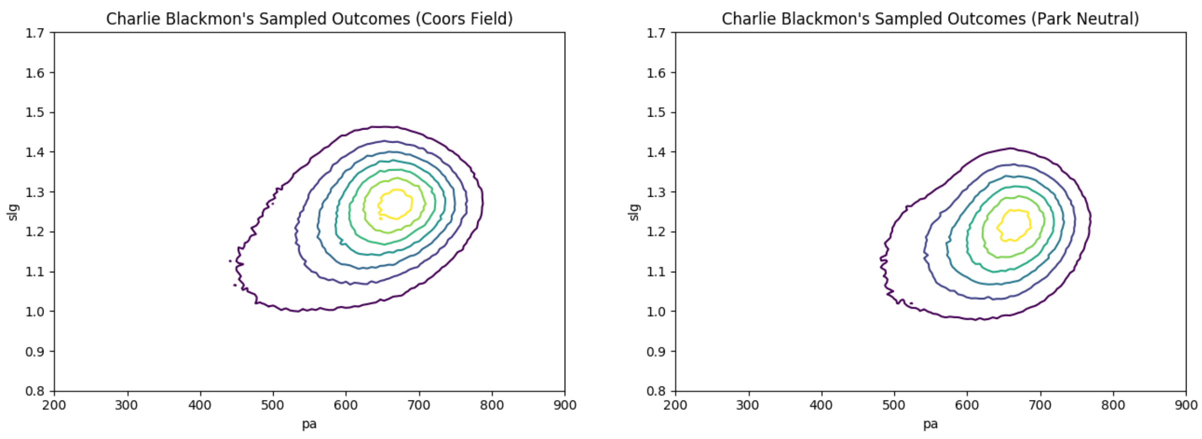


Figure 7.1: Charlie Blackmon's 2018 SLG predictions in Coors Field in Denver, compared to his predictions in a park-neutral environment. As expected, we see that Blackmon's park-neutral projections are slightly lower than his projections in Coors Field.

Coors Field is known as a hitter's paradise, and the often-inflated offensive statistics from Rockies players are normally taken with a grain of salt. We wish to examine how Blackmon would perform in a park-neutral environment. We have generated his SLG predictions both at Coors Field and in a neutral environment, and the results are given in Figure 7.1.

We observe that Blackmon's predictions in a park-neutral environment are lower than they are at Coors Field, and we claim that his park-neutral projections better reflect Blackmon's talent. Using this technique, we can attempt to isolate the components of a player's projection that are simply a result of his context and ignore those in comparing player outlooks. Furthermore, we can simulate how a player would perform in a different stadium, a useful feature for a team front office considering a potential trade.

## 7.2 Distribution Inspection

Another application of modeling probability distributions is that we can inspect players' distributions and infer meaning from them or compare them with other players' distributions. If our claims regarding heteroscedasticity hold, we would expect to observe two players with a similar mean prediction but different distributions of outcomes. We can test this by inspecting the distributions for two players. One such pair of players is Daniel Descalso

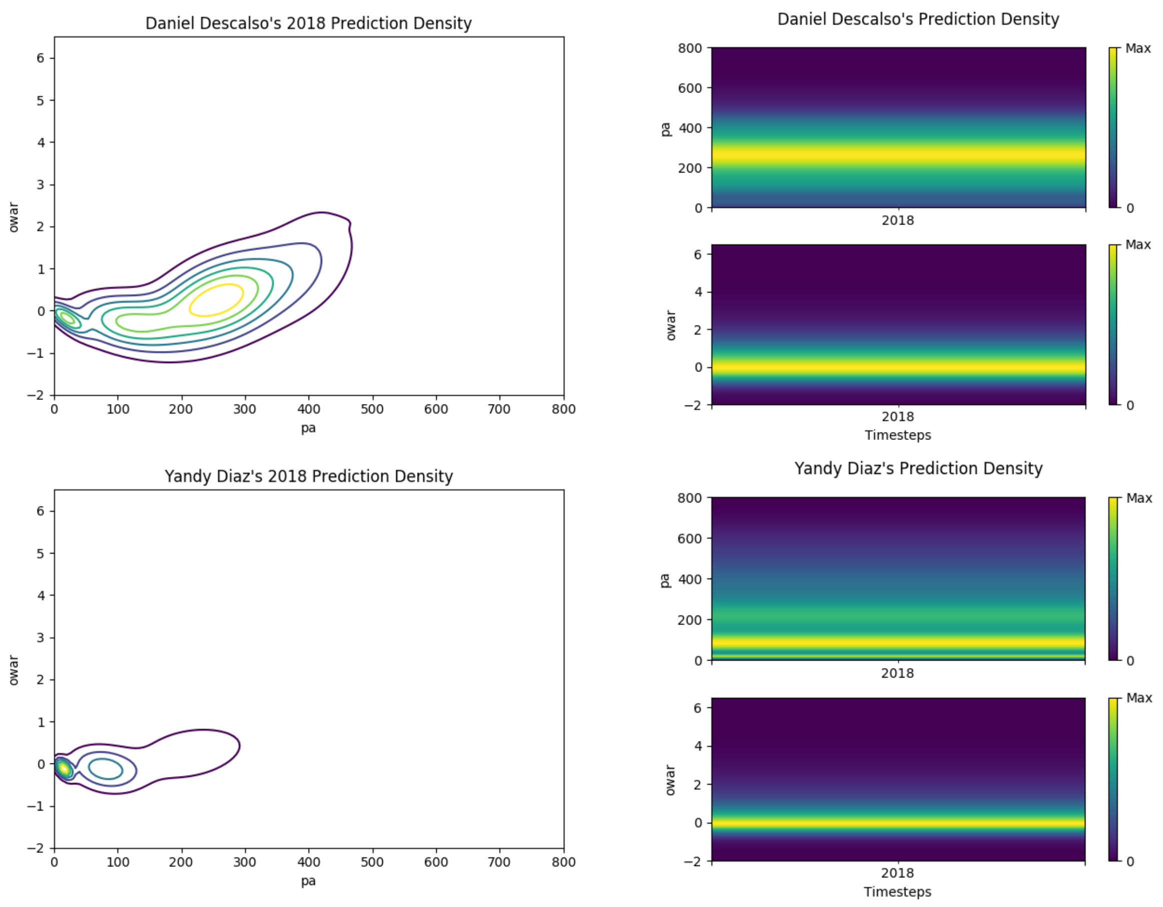


Figure 7.2: Even though the means of the 2018 projection distributions for Daniel Descalso and Yandy Diaz are almost identical, closer inspection reveals that they have divergent outlooks.

and Yandy Diaz, who have nearly the same playing time and oWAR mean predictions but unique distributions. Under $n = 8$ Gaussians, Daniel Descalso is projected to receive 254 PA and contribute 0.35 oWAR in a park-neutral environment in 2018, whereas Yandy Diaz is projected to receive 232 PA and contribute 0.42 oWAR. On the surface, this might seem to indicate that they may be of similar talent levels or have similar outlooks for the upcoming season. However, upon inspecting the distributions themselves given in Figure 7.2, we see that this is not the case.

We observe that the mode of Yandy Diaz's PA distribution is much lower than Daniel Descalso's but his PA ceiling is higher than Descalso's. In other words, Diaz's PA distribution is skewed positively whereas Descalso's is more symmetric. This is because Daniel Descalso is an established player, whereas Yandy Diaz only has 179 career PAs. As such, under this model, even though Descalso's oWAR mean is lower than Diaz's, Descalso has a slightly higher probability of achieving 1 oWAR ($\approx 0.215$, compared to $\approx 0.207$), though Diaz has a slightly higher probability of achieving 2 oWAR ($\approx 0.071$, compared to $\approx 0.058$). If we were building a team and were presented the option to choose either of these players, our choice should depend on our needs: if we wanted a player with a higher ceiling, we should choose Diaz, but if we want a player with less risk and a higher probability of being above offensive replacement level ($> 0$ oWAR), we should select Descalso.[1]

## 7.3 Divergence of Distributions

Generalizing the approach above, we wish to evaluate the dissimilarity of any two players' predictions. As seen above, evaluating the distance between the means is not sufficient, so we must compute the divergence of the expected distributions. To accomplish this, we use the Jensen-Shannon Divergence,

$$D_J(P_1 \parallel P_2) = \frac{1}{2} D\left( P_1 \,\middle\|\, \frac{P_1 + P_2}{2} \right) + \frac{1}{2} D\left( P_2 \,\middle\|\, \frac{P_1 + P_2}{2} \right) \in [0, \infty) \,, \tag{7.1}$$

where $D(P \parallel Q)$ is the Kullback-Leibler divergence. This quantity will be 0 if $P_1$ and $P_2$ are identical and is symmetric for any $P_1$ and $P_2$. Unfortunately, computing the KL divergence and therefore the Jensen-Shannon divergence between two mixtures of Gaussians is not analytically tractable [Hershey and Olsen, 2007]. As such, we use Monte Carlo sampling to approximate this quantity, using the property that

$$\begin{aligned} D(P \parallel Q) &= \mathbb{E}_{x \sim P} \left[ \log \left( \frac{P(x)}{Q(x)} \right) \right] \\ &\approx \frac{1}{n} \sum_{i=1}^{n} \left[ \log P(x_i) - \log Q(x_i) \right], x_i \sim P. \end{aligned} \tag{7.2}$$

[1]Of course, this analysis only considers the upcoming year and does not consider subsequent seasons or intangibles that would be valuable to a team front office, such as prior MLB experience and leadership traits. However, we are not concerned with DeepBall modeling these intangibles.

Using this approximation for KL divergence, we can compute an approximate Jensen-Shannon divergence, from which we can obtain a dissimilarity matrix $A \in \mathbb{R}^{n \times n}$ for all players, where $A_{ij} = D_J(P_i \parallel P_j)$. $A$ is symmetric, $A_{ij} \geq 0$, and $A_{ij} = 0 \leftrightarrow i = j$ (assuming no two players have identical distributions).

The matrix parameter $A$ has a few applications. First, it can be used to discover players with similar outlooks and not simply expected outcomes. We can interpret a low value of $A_{ij}$ to suggest that a team's outlook would be approximately the same by exchanging player $P_i$ for player $P_j$.[2] For example, Joey Gallo and Miguel Sano have very similar park-neutral distributions for 2018, so according to our model the Texas Rangers' and the Minnesota Twins' overall outlooks would not change significantly if these players were exchanged.

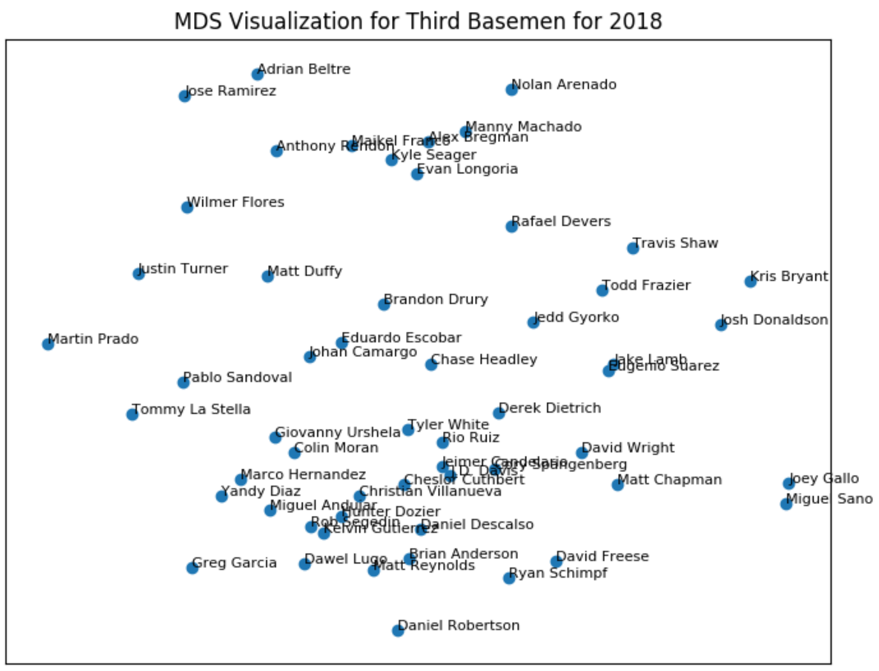
Second, we can apply nonmetric multidimensional scaling to $A$, obtaining a $d$-dimensional point for each player [De Leeuw, 1977]. We can then visualize or cluster these points, finding groups of similar players. We have applied this to all third basemen for the 2018 season. Using $d = 2$, we obtained the two-dimensional visualizations given in Figure 7.3. Two visualizations are shown, each generated using a different subset of the dimensions: the first computes the Jensen-Shannon divergence of the joint distributions or SLG, K%, OBP, PA, and oWAR, where the second only uses the joint distributions of PA and oWAR.

## 7.4 Outlier Detection

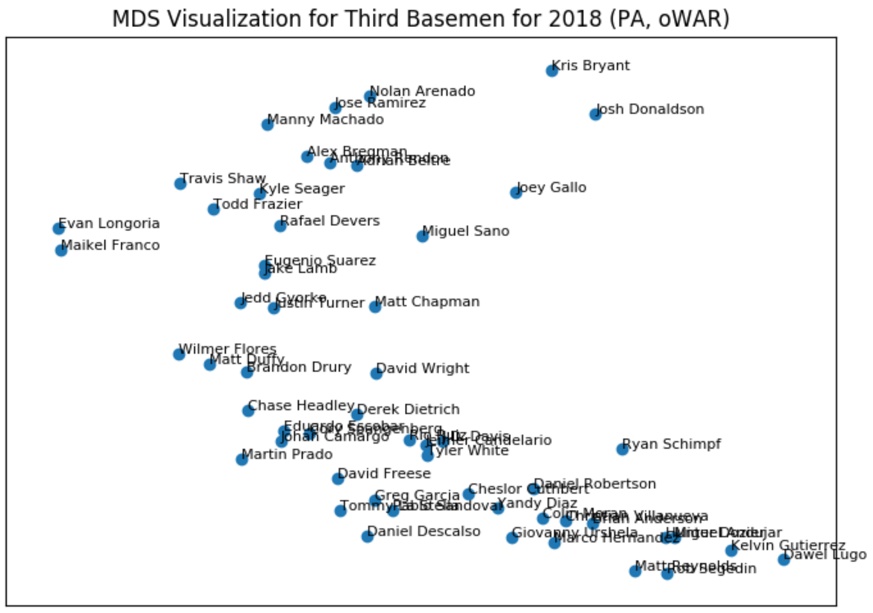
In addition to measuring and visualizing the dissimilarity between players or groups of players, we can use the predicted distributions to locate outliers or surprising seasons. We say a player's season ground truth was an outlier if the player's actual performance has a low likelihood under the predicted distribution for that season. In other words, a player's season was an outlier if the mixture of Gaussian loss specified in Equation 5.9 is large.

There are two possible causes for a player's season to be considered an outlier. First, the

[2]We have implicitly assumed that a player's individual season performance is independent of the performances of his teammates. This is most likely not entirely true, but it is convenient to assume. We leave modeling the correlations between teammates' performances as a topic for future research.



(a) MDS visualization over each player's joint distribution of SLG, K%, OBP, PA, and oWAR. The stress of this scaling is large (0.2077), but it reveals which players are most similar.



(b) MDS visualization over each player's joint distribution of PA and oWAR. The stress of this scaling is 0.0887, which is lower since the dissimilarities are only measured comparing two dimensions of the distributions.

Figure 7.3: We have used multidimensional scaling to visualize the dissimilarity matrix $A$ between player distributions, where $A$ was computed using the Jensen-Shannon divergence. We observe that similar players are grouped together.

| Player | Season | Log Likelihood |
|---|---|---|
| Brandon Guyer | 2015 | -20.5895 |
| Aaron Judge | 2017 | -17.7059 |
| Brandon Guyer | 2016 | -15.8169 |
| Bryce Harper | 2015 | -14.0331 |
| Tyler Flowers | 2017 | -13.9018 |
| Jung Ho Kang | 2015 | -13.5457 |
| Joc Pederson | 2017 | -12.4358 |
| Kris Bryant | 2016 | -10.8309 |

Table 7.1: We have selected player seasons between 2015 and 2017 with the lowest likelihood under our $n = 8$ Gaussians model. These data points can be interpreted as outliers.

player may have simply overperformed or underperformed according to DeepBall's prediction. We expect this to be especially common for rookies, because even though rookies have potential to have excellent seasons, DeepBall receives little information on these players and projects them all to perform approximately the same. Second, because DeepBall models the covariance between the statistics, the player may have performed at approximately the same level as expected, but the statistics may not have had the expected relationship. For example, a player may have had an unusually high AVG but an unusually low SLG (if most hits are singles), and though both values may be near the projected values, this method will detect this as an outlier. Eight players with the lowest likelihood have been listed in Table 7.1.[3]

We can group these player seasons into roughly three buckets. First, as discussed above, we see many rookies or young players who outperformed their projections. Examples of this are Aaron Judge, Kris Bryant, Jung Ho Kang. Each of these players were in either their first or second year and performed extremely well. Notably, Aaron Judge won the American League Rookie of the Year (RoY) award and broke the rookie record for home runs with 52, and Kris Bryant won the National League Most Valuable Player (MVP) award.

[3]Because of the transformation we applied in Section 5.3, the variance increases as PA, causing less probability mass to be reserved for any individual outcome as PA increases. For this reason, this method of outlier detection is biased towards samples with high PA. We could fix this by retraining using the transformation with $w = \sqrt{\text{PA}}$ given in Figure 5.3b, but we do not believe it would significantly change the results.

The second group consists of more established players who either underperformed or overperformed their expectation. Bryce Harper and Joc Pederson fit this criteria. Bryce Harper, in his fourth year in the MLB, significantly overperformed his projections, boosting his batting average by 0.060, his slugging percentage by 0.220, and his walk rate by 9% compared to his previous year, making him the unanimous 2015 National League MVP. This did prove to be an outlier, since he significantly regressed the subsequent year and his 2015 season remains his best to date. Joc Pederson's 2017 season, on the other hand, was an example of underperforming his projections. In his fourth year in the MLB, he dealt with injury issues, causing his playing time and performance to decrease. In addition, he had a low batting average on balls in play (BABIP) [Fangraphs, 2013a], which is commonly attributed to bad luck. These factors coupled together made his season an outlier.

The final group contains the players whose overall performances were roughly as expected but had unusual correlations between certain statistics: Brandon Guyer twice and Tyler Flowers. In each of these cases, the batter had an average or slightly above average batting average and walk rate but an unusually high on-base percentage. Upon closer inspection, it is clear what happened: both of these players were hit by pitches at extremely high rates during these seasons. The league average HBP per PA between 2015 and 2017 was 0.0091, but these three player seasons combined to have an average of 0.0682 HBP per PA, over 7 times the typical rate. Since OBP is the only statistic we are modeling that accounts for HBP (Equation 1.5), it is natural that OBP would be abnormally high. Consequently, since we are modeling the correlation between the statistics, it is expected that the likelihood of these outcomes will receive a low likelihood, causing them to be detected as outliers.

# Chapter 8: Conclusions and Future Work

There are a few possible areas for improvement upon this first iteration of DeepBall. Notably, we are missing at least two major data sources that will almost certainly be useful: injury data and minor league data. Currently, the only way DeepBall can detect an injury is by observing reduced playing time in a season. However, reduced playing time can have other causes, and DeepBall cannot distinguish among these. Moreover, certain injuries may affect a player's future performance more than others. In addition, Steamer and ZiPS use minor league data, and having this would improve DeepBall's projections for young players. Until minor league data is factored into DeepBall's estimations, it may be difficult for it to enter into mainstream use.

Second, as it is given here, DeepBall lacks breadth in the statistics it can predict. It is very reasonable for a fan to want to know an estimate of how many home runs a player will hit, but since this is not given as an output statistic in *out_counts*, it cannot be done without modifications. Furthermore, if we desire to add a new output statistic, we must retrain all models which is expensive and time-consuming. A possible solution is to approximate any new statistic by creating a regression using the current output statistics. For example, given a player's AVG, SLG, PA, and oWAR, we could likely arrive at a decent prediction for the player's home runs.

Furthermore, DeepBall as we have specified it here only predicts offensive statistics, whereas all major systems predict pitching statistics as well. This is a significant hole in the model, since a player's value and a team's performance depends heavily on both. We expect to be able to apply a similar recurrent neural network to predict pitcher performance, but the issue of limited data would become even more prevalent: there are only 4963 pitchers who have debuted between 1958 and 2014, and there are only 24561 seasons from those same pitchers through 2014. In solving this problem, we expect bagging to play an even more critical role in reducing variance.

This work also opens the door to many potential areas of future research. First, our network configuration allows for the possibility of generating latent representations of the data. For example, interpreting the 85-dimensional vector discussed in Section 3.2 as a latent representation of the season's prediction could allow it to be used for transfer learning to other tasks. As mentioned above, one such task could be predicting statistics that were not originally trained into DeepBall, such as home runs.

In addition, because the networks and ensembles are end-to-end differentiable, we can compute the gradients of the outputs with respect to the inputs, which may be useful for understanding DeepBall's projections. Because of this, DeepBall has the potential to launch into a new area of baseball research, discovering which features have the greatest predictive value and developing new statistics accordingly.

Finally, we can modify or add loss functions to predict various other statistics or uncertainty levels. One use for this could be predicting percentile thresholds or modes for players rather than means. The most general application of this was already done in Chapter 5, where we modeled a general probability distribution over the outcomes, from which either of these could be extracted. In summary, DeepBall has performed well on existing player projection tasks including modeling uncertainty, and we believe it shows potential to be extended into other areas of baseball prediction.

# Appendix A: Batted Ball Estimation

As mentioned in Section 2.2, we needed a way to estimate the batted ball type of any ball in play. To do so, we used Naïve Bayes with the following categorical features:

- A concatenation of the batter's handedness (left or right) and the position that fielded the ball, if both are available (otherwise, this feature is treated as missing),
- If there was an error, the fielding position of the player who made the first error on a play, along with the type of error (fielding, throwing, etc.),
- The fielding positions of the players who made the first two putouts and first assist on a play (if applicable),
- For each base-runner before the play, a concatenation of the base that they were at after the play and the play type (for example, the runner from first may have advanced to third on a single),
- The play made on the batter if the batter was out on the play, written in standard notation (for example, 63 represents an out recorded from the shortstop to the third baseman),
- The hit location as defined by [Retrosheet, 2003], and
- Flags for whether it was a bunt, sacrifice hit, or sacrifice fly.

We trained the model on data from 2003-2013, which has data for 99.97% of applicable plays, and evaluated it on 2014 data, on which it obtained approximately 78% accuracy.

## References

[Abadi et al., 2015] Abadi, M. et al. (2015). Tensorflow: Large-scale machine learning on heterogeneous systems.

[Acharya et al., 2008] Acharya, R. A. et al. (2008). Improving major league baseball park factor estimates. *Journal of Quantitative Analysis in Sports*, 4(2):4.

[Baseball-Reference, 2017] Baseball-Reference (2017). War explained. `https://www.baseball-reference.com/about/war_explained.shtml`. [Online; accessed 1-Feb-2018].

[Baseball-Reference, 2018] Baseball-Reference (2018). Major league baseball batting year-by-year averages. `http://www.baseball-reference.com/leagues/MLB/bat.shtml`. [Online; accessed 6-Feb-2018].

[Bradbury, 2010] Bradbury, J. (2010). How do baseball players age? `http://www.baseballprospectus.com/article.php?articleid=9933`. [Online; accessed 25-May-2017].

[Cho et al., 2014] Cho, K. et al. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP*.

[Chollet et al., 2015] Chollet, F. et al. (2015). Keras. `https://github.com/fchollet/keras`.

[Cross et al., 2008] Cross, J. et al. (2008). `http://steamerprojections.com/blog/about-2/`. [Online; accessed 24-May-2017].

[Davidson et al., 2010] Davidson, D. et al. (2010). `http://steamerprojections.com/blog/evaluation-of-2009-hitter-forecasts/`. [Online; accessed 25-May-2017].

[De Leeuw, 1977] De Leeuw, J. (1977). Applications of convex analysis to multidimensional scaling.

[Druschel, 2016a] Druschel, H. (2016a). A guide to the projection systems. `http://www.beyondtheboxscore.com/2016/2/22/11079186/projections-marcel-pecota-zips-steamer-explained-guide-math-is-fun`. [Online; accessed 24-May-2017].

[Druschel, 2016b] Druschel, H. (2016b). A million monkeys at a million spreadsheets: 2015's projection systems in review, part one. `http://www.beyondtheboxscore.com/2016/1/14/10733872/steamer-zips-pecota-marcel-projections-review`. [Online; accessed 30-May-2017].

[Druschel, 2017] Druschel, H. (2017). Grading the projections: 2016. http://www.beyondtheboxscore.com/2017/1/8/14189138/pecota-zips-steamer-marcel-projection-systems-graded. [Online; accessed 30-May-2017].

[ESPN, 2018] ESPN (2018). Park factors. http://www.espn.com/mlb/stats/parkfactor. [Online; accessed 1-Mar-2018].

[Fangraphs, 2013a] Fangraphs (2013a). Babip. https://www.fangraphs.com/library/pitching/babip/. [Online; accessed 30-Mar-2018].

[Fangraphs, 2013b] Fangraphs (2013b). What is war? http://www.fangraphs.com/library/misc/war/. [Online; accessed 15-Aug-2017].

[Fangraphs, 2013c] Fangraphs (2013c). woba. http://www.fangraphs.com/library/offense/woba/. [Online; accessed 15-Aug-2017].

[Gal and Ghahramani, 2016] Gal, Y. and Ghahramani, Z. (2016). A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, pages 1019–1027.

[Gloeckner, 2014] Gloeckner, L. (2014). Introduction to the baseball projections — 2014 version. http://mrcheatsheet.com/2014/01/22/introduction-to-the-baseball-projections-2014-version/. [Online; accessed 31-May-2017].

[Goodfellow et al., 2013] Goodfellow, I. et al. (2013). Maxout networks. *Proc. ICML.*

[Hara et al., 2016] Hara, K. et al. (2016). Analysis of dropout learning regarded as ensemble learning. pages 72–79.

[Heipp, 2005] Heipp, B. (2005). Park factors. http://gosu02.tripod.com/id103.html. [Online; accessed 1-Mar-2018].

[Hershey and Olsen, 2007] Hershey, J. R. and Olsen, P. A. (2007). Approximating the kullback leibler divergence between gaussian mixture models. In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, volume 4, pages IV–317. IEEE.

[Hinton et al., 2012] Hinton, G. et al. (2012). Neural networks for machine learning. *Coursera, video lectures*, 264.

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167.*

[Kingma and Ba, 2014] Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

[Lahman, 2016] Lahman, S. (2016). `http://www.seanlahman.com/baseball-archive/statistics/`.

[Pedregosa et al., 2011] Pedregosa, F. et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Perrone and Cooper, 1992] Perrone, M. P. and Cooper, L. N. (1992). When networks disagree: Ensemble methods for hybrid neural networks. *Published in R.J. Mammone, editor, Neural Networks for Speech and Image processing.*

[Retrosheet, 2003] Retrosheet (2003). Retrosheet hit location diagram. `http://www.retrosheet.org/location.htm`. [Online; accessed 8-Feb-2018].

[Retrosheet, 2017] Retrosheet (2017). The information used here was obtained free of charge from and is copyrighted by Retrosheet. Interested parties may contact Retrosheet at ”`http://www.retrosheet.org`”.

[Semeniuta et al., 2016] Semeniuta, S. et al. (2016). Recurrent dropout without memory loss. *arXiv preprint arXiv:1603.05118.*

[Silver, 2007] Silver, N. (2007). 2007 hitter projection roundup. `https://web.archive.org/web/20080111231423/http://www.baseballprospectus.com/unfiltered/?p=564`. [Online; accessed 8-June-2017].

[Srivastava et al., 2014] Srivastava, N. et al. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

[Szymborski, 2008] Szymborski, D. (2008). Zips q and a. `http://www.baseballthinkfactory.org/szymborski/zipsqa.rtf`. [Online; accessed 30-May-2017].

[Tango, 2001] Tango, T. (2001). `http://tangotiger.net/marcel/`. [Online; accessed 24-May-2017].

[Tango, 2004] Tango, T. (2004). `http://www.tangotiger.net/archives/stud0346.shtml`. [Online; accessed 24-May-2017].

[Tango, 2007] Tango, T. (2007). Forecast evaluations. `http://www.insidethebook.com/ee/index.php/site/comments/forecast_evaluations/`. [Online; accessed 30-May-2017].

[Weinberg, 2017] Weinberg, N. (2017). Aging curve. `http://www.fangraphs.com/library/principles/aging-curve/`. [Online; accessed 25-May-2017].

[Yakovenko, 2017] Yakovenko, N. (2017). Machine learning for baseball (my story). `https://medium.com/@Moscow25/machine-learning-for-baseball-my-story-84dbe075e4b1`. [Online; accessed 11-Feb-2018].