Using Convolutional Neural Networks to Value Game Positions in Professional Basketball

A thesis

submitted in partial fulfilment

of the requirements for the Degree

of

Master of Science in Statistics

at

The University of Waikato

by

J.P. Gibb

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

2024

Abstract

Many professional sports are currently midway through their own "Data Revolutions", with advances in analytics being driven by the intense competition for even the most minor advantages, although typically being focused on offensive actions within a game, rather than statistics or metrics pertaining to a player's defensive abilities.

A significant portion of defensive duties is positioning one's self to prevent opponent actions from occurring at all, and often defensive actions such as tackles or steals are indicative of defensive mistakes rather than an example of good defending.

In other areas, Deep Learning models have shown an uncanny ability to identify trends and perform tasks typically considered too complex for software, and better left to humans. Various Deep Learning models have been designed and developed to estimate the value of positions in multiple sports, and shown to be very promising.

This thesis attempts to contribute to this by investigating the use of Convolutional Neural Networks to estimate the value of basketball possessions. The model predicts both the terminal actions and the expected value of a possession, and achieves good results relative to the closest published literature in similar tests.

Finally, we show how such a model may be utilised to produce metrics describing a player's positioning ability as a potential tool for player scouting or coaching.

Acknowledgements

I would like to express my deepest gratitude to my thesis supervisor, Dr. Jason Kurz, for his invaluable guidance throughout this project.

A special thanks to Dr. Bob Durrant for his early feedback and support of this research.

Lastly, I would like to acknowledge my family for their belief and moral support during this process.

Contents

1 Introduction **1**

1.1 Background 1

1.2 Thesis Aims 2

1.3 Thesis Structure 3

2 Background **4**

2.1 Problem Outline 4

2.2 Literature 5

2.2.1 Sabermetrics 6

2.2.2 AlphaZero 12

2.2.3 Football 21

2.2.4 Basketball 41

2.2.5 Deep Learning 52

3 Methodology **59**

3.1 Data 59

3.1.1 Description 60

3.1.2 Data Processing 61

3.2 Modelling 64

3.2.1 Player Embedding 69

3.2.2 Convolutional Block 70

3.2.3 Inverse CNN 71

3.2.4 Final Model 72

3.2.5 Data Augmentations 72

4 Results **75**

5 Conclusion **87**

5.1 Future Work 88

Appendices **97**

A **98**

List of Figures

2.1 Illustration of the “ko” rule in Go 15
2.2 AlphaGo “Residual Block” Diagram 20
2.3 Example of an xG Map (Sumpter and Andrzejewski [71]) . . . 23
2.4 Dangerousity Zone Values (Link, Lang, and Seidenschwarz [33]) 28
2.5 Dangerousity Pressure Zones (Link, Lang, and Seidenschwarz [33]) 29
2.6 Example of Dangerousity Shot Density (Link, Lang, and Seidenschwarz [33]) 31
2.7 Example of Dangerousity Pass Density (Link, Lang, and Seidenschwarz [33]) 31
2.8 Juego de posición zones example 33
2.9 Stacked LSTM Model Structure 51
2.10 Basic Neural Network Diagram 54
2.11 Convolution Example 57
2.12 Basic RNN Model Structure 57
2.13 LSTM Cell Structure Zhang et al. [76] 58

3.1 SportVU Data Structure 61
3.2 Base Model Network Architecture 68
3.3 Inverse Convolution Example 72

4.1 Base Model Training Results 76
4.2 Embeddeing Model Training Results 76
4.3 Conv. Block Model Training Results 77

4.4 Inv. CNN Model Training Results 77
4.5 Final Model Training Results 78
4.6 Final Model (Augmented) Training Results 78
4.7 Base Model and Embedded Model F1 Scores 79
4.8 Conv. Block and Inv. CNN Model F1 Scores 79
4.9 Final Model and Final Model (Augmented) F1 Scores 80
4.10 Final Model Prediction Confusion Matrix 81
4.11 Key Frames of Highlighted Play 83
4.12 Play Predicted Outcomes 84
4.13 Play Predicted Points 84
4.14 Heatmap of Expected Points 85

A.1 Quarter 1 Null Prediction Probability 98
A.2 Quarter 1 Shot Prediction Probability 98
A.3 Quarter 1 Foul Prediction Probability 99
A.4 Quarter 1 Turnover Prediction Probability 99
A.5 Quarter 2 Null Prediction Probability 99
A.6 Quarter 2 Shot Prediction Probability 100
A.7 Quarter 2 Foul Prediction Probability 100
A.8 Quarter 2 Turnover Prediction Probability 100
A.9 Quarter 3 Null Prediction Probability 101
A.10 Quarter 3 Shot Prediction Probability 101
A.11 Quarter 3 Foul Prediction Probability 101
A.12 Quarter 3 Turnover Prediction Probability 102
A.13 Quarter 4 Null Prediction Probability 102
A.14 Quarter 4 Null Prediction Probability 102
A.15 Quarter 4 Null Prediction Probability 103
A.16 Quarter 4 Null Prediction Probability 103
A.17 Quarter 1 Expected Points Prediction 103
A.18 Quarter 2 Expected Points Prediction 104
A.19 Quarter 3 Expected Points Prediction 104

A.20 Quarter 4 Expected Points Prediction 104

List of Tables

3.1 Input Channel Descriptions 67

4.1 Final Model Results on Test Set 81

4.2 Test Set F1 Scores by Class 81

Chapter 1

Introduction

> The idea is not to block every shot. The idea is to make your opponent believe that you might block every shot.
>
> *Bill Russell*

1.1 Background

In high-level competitive environments fine margins are often what separates the winners from losers; a single pawn in chess, a key tackle in football or making one unlikely shot in basketball can be the difference between champions and losers. This is the all-or-nothing nature of competitive sports, a 1 percent difference in performance, ability or luck is often rewarded with victory, while other times not. It is common knowledge that the best team doesn't always win, and in fact, this is the entire reason sports are so popular: the outcome is not pre-determined. Teams and players pursue any advantage they can however, as they understand they must maximise their chances of being victorious. The collection of data has also developed leaps and bounds to facilitate this constant struggle to find the next advantage, allowing for increasingly complex models to be developed for sports teams and athletes by data science teams, which are now found at nearly every top sports franchise in most major sports

[74]. Notably, data detailing the locations of every player in the field of play at split-second time intervals is now being collected and provided to teams, allowing for the increasingly granular analysis of player positioning and the effects this has on a team's performance.

Simultaneously, the recent advancement of Machine Learning and Deep Learning practices in particular has been difficult to ignore, and their ability to perform tasks and identify trends far beyond what was once considered possible by machines is becoming clear. Naturally, these techniques and models are gaining traction within a sporting context, with literature being published outlining applications to different problems within the sporting sphere.

1.2 Thesis Aims

While progress is significant and rapid, research is often focused on modelling offensive actions and abilities, meaning there exists a noticeable blind-spot, the effect of defensive positioning on the team's overall performance. There exists a multitude of ways to value a football player's shooting ability or a basketball player's offensive contributions through advanced stats, while defensive contributions often take the form of the absence of opponent actions, or even just reduced quality of opposition chances. In other words, the collection of measurable event statistics such as tackles and blocks are not capturing the complete picture of a player's offensive contributions.

Thus we aim to progress the analysis of players' defensive abilities, which we assume are largely un-quantifiable through traditional metrics or through the measurement of player actions. Therefore, in this thesis we propose a model that allows for the valuation of basketball possessions primarily using spatial data detailing the locations of the players on court. We will employ the use of Convolutional Neural Networks with the goal of capturing the spatial effects player positioning has on the expected value of a basketball possession. From this, it will be possible to determine the defensive positioning efficiency

of individual players, in both offense and defense, by comparing their observed position with the model optimum.

1.3 Thesis Structure

As stated, the structure of the network will be largely based on convolutional layers. We will also supplement a base model with additional layer types in an effort to extract further performance from our model.

Chapter 2 will provide an overview of the problem we wish to address, and build contextual understanding by briefly discussing relevant material on Artificial Neural Networks and studying the current literature on analysis techniques within sports.

Chapter 3 will outline our dataset and our modelling, including a description of our available data, data cleaning, and various model architectures tested.

In Chapter 4 we will present the results for our tested models, and further inspect more granular results for our best performing models. We will also present a potential method of quantifying the positioning of a player in this section.

Finally, our conclusion and potential future work will be presented in Chapter 5.

Chapter 2

Background

2.1 Problem Outline

Player positioning in basketball has historically been evaluated by humans heuristically using intuition gained from a history of experience within the game of basketball in some capacity, whether as a coach, player or video analyst. There are a number of issues with this method. Players are assessed individually by watching the player at a manageable speed, meaning this process is slow and limits the number of players that can be assessed. The assessors are also prone to bias. Human bias is not a new concept and its impact in sports is currently undergoing a significant amount of study. For example, black players in sports are more likely to be described as "powerful" vs "technical" for white players, or simply described negatively for similar traits, e.g. "sneaky" vs "clever" [47, 10, 40]. It's also difficult to separate a player's actions from the outcome of a play, a player can defend perfectly and still not succeed due to other factors such as random chance or the defensive efforts of the other players on the team. For example, forcing an opposition player into a difficult shot that they actually make, the defender succeeded by minimising the scoring probability despite the opposition still managing to make the shot. Extending this point, a player may be appropriately positioned, while not possessing the other necessary abilities to compete against their individual

opponent match-up. Similarly, we would ideally also take into account certain other player abilities. For example we would most likely want to defend a short player with high 3-point shot accuracy differently to a large player who tends to drive towards the hoop. Overall, it's difficult to compare the positioning of multiple players in a way that is not purely qualitative due to the above factors and the current method of evaluation.

The ideal solution would be an accurate model of the probability of a team scoring during a particular moment of the game that takes into account the effects of the offensive and defensive player positions, along with the particular abilities of each player. Using this we could then find the optimum position/s for each player and take some measurement of distance from the optimum as the metric with which we can describe a player's positioning during that moment.

Naturally a perfectly accurate model is currently unrealistic. However, in this study we will attempt to develop a model that will output a reasonable value estimate of a given basketball play. The most intuitive way to model this is by attempting to predict the expected number of points scored at any given point in a game within some time interval.

2.2 Literature

The aim of this thesis is to build a model to analyse the player positioning in basketball. We should begin by building a solid understanding of the current best practice of sports data analytics and the origin and thought-processes behind them. Due to the small number of teams and high level of competition, resources discussing analysis at elite basketball teams are essentially non-existent as teams are largely unwilling to share their work. Fortunately, due to the amount of analytics staff transferring between sports, the backgrounds of current analysts, and the practices across all elite sporting teams being fairly similar in terms of techniques and trajectory (which will become

evident by the end of this literature review), it is therefore useful to study analytics across sporting codes as an introduction to our problem. Additionally, we can make educated guesses about particular practices based on the limited information gained through articles, blogs and interviews with players or members of staff within a given sport or competitive environment.

2.2.1 Sabermetrics

> We can't do the same things the Yankees do. Given the economics, we'll lose.
>
> *Billy Beane*

It is difficult to discuss analytics within a sports context without at least mentioning Baseball. Baseball has historically had data collection deeply ingrained as part of its identity since the MLB's formation, collecting statistics such as hits, home runs, strikes etc. with the box score, statistics describing each players achievements during a game, being developed as early as 1859 [3]. Sabermetrics, coined in 1980 by Bill James, can in fact be traced back to this early box score. Although James' work was the first to experience widespread recognition and adoption within professional baseball, or any traditional major sport [68].

Due to the discretised nature of baseball, where a player is performing one task for the duration of a play and the play will end with some discrete outcome (out, strike, hit etc.), analytics is typically broken down into the three (major actions) a player can perform; batting, fielding and pitching.

The traditional, and most intuitive, method of measuring batting performance is through the calculation of the batting average, AVG. This is calculated using the formula:

$$AVG = \frac{H}{AB}$$

Where H is the number of hits a player achieves, and AB is the number of "At Bats" a player attempts. The closer this average is to 1 then the more likely a player is to make contact with the ball while batting, and the more likely a player is to either successfully progress through bases or give their teammates a chance to progress [1].

There are two main objections to this statistic: it does not account for all methods of getting on base, nor the impact of the method (number of bases earned). If we are interested in a players ability to advance through the bases, we should also incorporate other methods of progressing ("getting on base"), such as a hit-by-pitch or walk (where a pitcher pitches 4 balls outside of the strike zone while the batter does not swing), as these are just as valuable as getting on base via a hit. The On Base Percentage, OBP, can therefore be given as:

$$OBP = \frac{H + BB + HBP}{AB + BB + HBP + SF}$$

Where H is hits as with the previous formula for AVG, BB is "Base on Balls" which is more commonly known as a walk, HBP is "Hit-By-Pitch". SF is a "Sacrifice Fly" where a batter will deliberately hit a ball into the outfield which will likely be caught (leading to the batter to be called out), but giving other runners enough time to score runs. OBP therefore gives the percentage of plate appearances a given player has advanced bases [1].

The second objection is that batting average does not take into account the number of bases a hit advances the runners, e.g. a home run is intuitively more valuable than merely reaching first base, yet batting average treats each of these identically. The common metric to account for this is the "Slugging Percentage", SLG, which essentially computes the average number of bases a player reaches for each at bat. The formula for SLG is:

$$SLG = \frac{1B + 2 \times 2B + 3 \times 3B + 4 \times HR}{AB}.$$

The SLG takes the number of bases advanced (1, 2, 3 or home run) and

multiplies them by the number of times a player has reached this base from one at bat ($1B$, $2B$, $3B$ and HR) and then divides this by the total at bats a player has attempted [1].

OBP and SLG capture two important aspects of batting, the ability to get on base, and the ability to advance runners. Typically, these two metrics are combined to yield OPS, or "On-Base Plus Slugging", which as the name suggests is computed as such:

$$OPS = OBP + SLG.$$

Albert [1] (2010) found that OPS has a much higher correlation with total runs scored than any of the other previously discussed metrics, and has experienced mainstream acceptance among casual viewers due to the adoption of OPS by MLB and its inclusion in live coverage [1]. OPS is currently used as an overall measure of batting ability.

Pitching metrics have a slightly less straightforward history, traditionally pitchers were evaluated using a simple win percentage, where the idea is that better pitchers would win more games. Another slightly more advanced metric used to evaluate pitching ability is the "Earned Run Average", ERA, which is calculated as:

$$ERA = 9 \times \frac{EarnedRuns}{InningsPitched}$$

and scales the average runs a pitcher allows per inning to a per game basis, hence the multiplication by 9. The main issues with both of these metrics is that they are both heavily influenced by factors outside of the pitcher's control. Win percentage is obviously heavily influenced by the abilities of all other players, while ERA is often influenced by the pitcher's fielders and other external factors. For example a good pitch could still be hit for a home run by random chance or a fielder could make a mistake and give up a run. In 2001 Voros McCracken published his article "Pitching and Defense: How Much Control Do Hurlers Have?" which asserted that pitchers have very little

influence over the traditional metrics used to evaluate pitching ability, and justifies this claim by showing that the average ERA of the pitchers on a team is a better predictor of a player's ERA than the player's own ERA from the previous season. This indicates the team's defense is the largest factor in determining the runs allowed while the pitcher has very little influence [39]. The article goes on to define a new idea in baseball, independent of the defense, "Defense Independent Pitching Statistics", $DIPS$ which adjusted each pitcher's statistics to the league average for things outside of the pitcher's control, such as the number of hits (where a player advances by at least a base after hitting the ball into play). The main takeaway from McCracken's work is that the pitcher usually only has influence over three statistics, or outcomes, pertaining to a pitch: Home Runs, Walks, and Strikeouts, which should be the basis of the discussion when evaluating pitchers.

Using these Defence Independent Statistics, new metrics for evaluating a pitcher's ability can be developed. The most common is the Defence Independent Component ERA, $DICE$, also known as "Fielding Independent Pitching", FIP, which is computed using the formula:

$$DICE = 3 + \frac{13 \times HR + 3(BB + HBP) - 2 \times SO}{IP}$$

where SO is the number of Strikeouts a pitcher achieves, and IP is the total number of Innings Pitched. The formula coefficients are derived from computing the average value of each outcome, from the point of view of the batting team, relative to the average play, and multiplying this by the number of innings in a game (9), while the constant is derived from computing the average $DICE$ of the league and is used to scale the $DICE$ value to a similar range as the traditional ERA. Therefore, a lower $DICE$, as with ERA, indicates better pitching ability [36, 59].

For Fielding, our traditional measure is the "Fielding Percentage", $FLD\%$ or $FPCT$, which is merely a proportion of fielding actions a player attempts that have a positive impact on a teams defense, i.e. getting a player out or

assisting versus an error. It is computed using the formula:

$$FLD\% = \frac{PO + A}{PO + A + E}$$

where PO is the number of actions that complete an out (e.g. catching the ball or tagging a runner), A is the number of assists a player records (any touch of the ball before an opposing player is called out, even accidental) and E is the number of errors a player records. An error in this context is any time a player fails to complete an out that the official scorer considers an average fielder should complete [37, 1]. The main shortcoming of this metric is clearly the element of subjectivity in how an error is defined, and is evidenced by the distribution of $FLD\%$ by player positions. Shortstops and catchers record the largest number of errors due to the difficulty of the balls that players in those positions tend to deal with. A player may also perform an action better than an average fielder but may make a sub-optimal decision immediately after, leading to a less beneficial outcome for the team where an average fielder would have had a recorded detrimental outcome from the entire sequence.

There is also a need to determine how often a fielder is involved in fielding actions, a player who never makes mistakes but only makes a single fielding action per game is intuitively less valuable to a team than a player who makes some errors but is involved more in fielding. To account for this, the "Fielding Range Factor", RF, was also developed, which finds the average number of positive fielding actions a player records per game, this is computed using the formula:

$$RF = 9 \times \frac{PO + A}{IP}$$

This metric merely calculates the average number of positive fielding actions a player records per inning, and scales this to a per-game basis [1].

Recent advances in Sabermetrics include the usual tinkering done on previous metrics to improve their accuracy relative to observed results. For example, adjusting the weights for each of the terms in the OPS metrics based

on the observed value of each outcome (i.e. A Home Run is generally considered roughly 25% more valuable than reaching 3rd base) [61], or using the discussed statistics to create a metric called "Wins Above Replacement", or WAR, which aims to measure the number of additional games a player is expected to win for his team relative to an average baseball player [60, 38].

There has also been the recent introduction of *statcast*. This refers to statistics relating to player movement and athletics abilities, such as spin rate, release point and release velocity for pitching, exit velocity, distance and exit angle for hitting, as well as measurements of reaction times and speed for both baserunning and fielding. While some data is available in some capacity to the public [4], only the baseball clubs and MLB have full access to the data collected. While it is impossible to know how the clubs use this data, public comments by players and staff indicate that this new data has become the main focus of the analysis performed by teams, if not outright replacing the traditional metrics, with players stating they are being measured predominantly using statcast data [5, 8].

The overall takeaway from this section is the transition of baseball metrics from general statistics measuring occurrences of positive or negative events, to a more nuanced approach of isolating the contribution of individual players independent of their team. Analytics in baseball are now centred on the measurement and analysis of core aspects of play that are thought to maximise positive outcomes, such as pitch spin rate or player acceleration. Major league baseball has seen a trend of data becoming more and more focused on true predictors of player performance, while attempting to eliminate the noise generated by random chance and/or teammates.

2.2.2 AlphaZero

> Ridiculous! A machine will always remain a machine, that is to say a tool to help the player work and prepare. Never shall I be beaten by a machine!
>
> *Garry Kasparov*

The earliest form of an automated chess engine goes back to at least 1770, when the "Mechanical Turk" was built. Invented by Wolfgang von Kempelen in Vienna, the machine consisted of a chess board and the upper half of man wearing robes and a turban protruding from the top of large box which contained, according to the inventor, intricate machinery powering the man. The machine could solve difficult puzzles and play games at a Grandmaster level. This was largely due to the human Grandmaster von Kempelen would hide within the large box controlling the "machine", a fact only revealed after von Kempelen's death [27]. The story, while showing Man's ability to deceive, also shows how early we considered boardgames, specifically chess, as the showgrounds of AI.

The challenge of creating high-level chess engines has been the subject of study for many computer scientists for decades, including Alan Turing, who developed his own algorithm named "turochamp" in 1948. The algorithm used an evaluation function which valued each position heuristically along with every possible move to a depth of 2, and would use the minimax algorithm (where the positions after best possible player move and then the worst possible opponent move would be compared) to select each move. The algorithm would also evaluate positions after every forced move, so would evaluate to a depth further than 2 if one or both players are forced to move. The evaluation function takes point values for each piece, as well as taking into account piece mobility, pawn advancement, piece protection and possible captures or

checks/checkmates [18, 34].

The earliest form of these heuristic board evaluations is dated back to sometime in the 18^{th} century, with many notable players developing and refining their own systems by the 19^{th} and early 20^{th} centuries. Generally, pieces are given values ranging from 1 to 10 with values being normalised such that pawns are worth 1, and kings not being given a value. These are often supplemented by multipliers and adjustments based on piece position (e.g. advanced pawn), protection and connection to other pieces and even the stage of the game (e.g. two rooks are equal to a queen in the mid-game but become relatively more powerful in the end-game) [67, 32, 73].

Possibly the most noteworthy iteration of the idea of these evaluation functions is that of the "Deep Blue" chess engine. Developed by IBM in the 1990s, the engine famously beat the chess world champion at the time, Garry Kasparov, in 1997. It is important to note that the success of Deep Blue was in large part due to the specially designed hardware and parallel search algorithm in addition to the evaluation function. As this thesis is primarily concerned with the analysis and evaluation of game states, the first two aspects will not be looked at in detail, but nonetheless are important to acknowledge as significant parts of Deep Blue's performance [6].

The Evaluation function for Deep Blue consisted of a linear combination of around 8000 hard-wired and hand-tuned features (as in, features were incorporated into the hardware of the "chess chip", leading to reduced software requirements and improved speed, although making it very difficult to introduce new features). The features themselves did not necessarily need to be linear however. These features were divided between *fast* and *slow* evaluation features. *Fast* evaluation features consisted of quickly identifiable patterns of game-play such as particular pieces on particular squares, or whether a pawn can move forward, while *slow* evaluation features consisted of more difficult to identify patterns such as king safety or pawn structure. The fast evaluation was used as an estimation of the game value in search algorithms, and also

in the overall evaluation function where each of the fast and slow evaluations are weighted to give the final value used in more rigorous searching. These fast and slow evaluation features could also be further divided into static and dynamic weights. Static weighted features would be assigned at the beginning of a search and remain the same until the search is complete. Dynamic weights were also set at search initialisation, but they would be adjusted based on the context of the game board at evaluation time during the search.

The weights are stored in lookup tables and are generated at the beginning of a search by the "Evaluation Function Generator", which will take the current board position and generate the contextual values of each feature. These values can also be relative/dependent to/on each other, such as king safety being more important if more pieces are on the board. Positions are scanned one column at a time with values being assigned based on the pieces on that rank, with values assigned based on the features relevant to that piece [6].

The tuning process involved playing games between the current iteration of Deep Blue and one of the recruited Grandmasters, who would then work with the IBM computer scientists to identify and address weaknesses in Deep Blue's game-play [19]. Occasionally, due to the difficulty of adding features, these issues would need to be solved indirectly via estimation using the available features at the time.

Although Deep Blue has since been surpassed in playing strength, this approach has been the basis of most chess engines since. One can verify this as many top chess engines are developed as open-source software, meaning the codebase is available to view by anyone. The main gains made since Deep Blue have been largely due to improved hardware, search algorithms, board encodings and further fine tuning of features and their weights [46, 31].

In 2016, Google Deepmind, a subsidiary of Google focussing on AI research, organised a five game Go series between their program "AlphaGo" and Lee Sedol, who was ranked number one in the world from 2007 to 2012, and was was widely considered one of the best Go players ever [23].

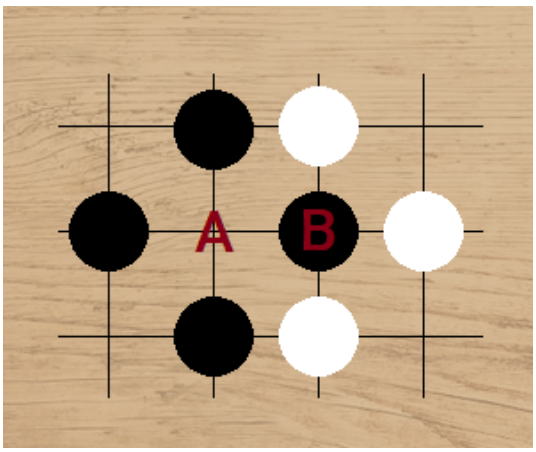


Figure 2.1: Illustration of the “ko” rule in Go

Go is a strategic board game invented in China more than 2500 years ago. The game is played by two players and consists of placing stones of the players assigned colour (black or white) on a 19x19 square board. There are two main rules of Go, the first is that every stone on the board must have, or be orthogonally connected to a group of the same colour with at least one orthogonally free space, called a “Liberty”; any piece or group that becomes fully encircled, or “captured” will be removed from the board and given to the capturing player as “prisoners”. The second rule, “Ko rule”, states that players cannot place stones that will return the board to a position identical to their previous turn. An example of a situation where this rule may be relevant is illustrated in Figure 2.1.

The white player may place a stone in position “A”, capturing the black stone in position “B” and removing it from the board. The Ko rule means that the black player is now not able to immediately place a piece back in position “B” capturing the white piece in position “A” and returning the board to its previous state. The black player can, however, play a piece somewhere else on the board before playing a piece back in position “B” in their next turn. There are a few other minor rules which essentially dictate how the main two rules will operate in particular circumstances, and these minor rules may vary by region. Players take turns placing stones or passing until both players pass consecutively, at which point the games ends and scores are tallied. Scoring is

done by removing any "dead" pieces (opposition pieces that would be captured trivially should the game continue) and adding these to the players' prisoners, then counting the number of "prisoners" each player has captured, and finally adding the number of fully encircled open spaces each player controls.

Go is largely considered more complex than chess, due to the nature of the game being less reliant on immediate calculation and more reliant on long term planning and strategy. Considering the static nature of the stones, poorly placed stones at the start of the game can lead to large disadvantages in the end game. Moreover, the size of the board often means there are multiple "sub-games" happening simultaneously on different areas of the board which eventually connect in the end game. The size of the board also leads to a significantly larger number of possible positions compared to chess, approximately 2×10^{170} different legal positions, compared to around 10^40 for chess [29, 25].

The historical methods used in chess engines did not yield a program capable of defeating a top professional Go player. Instead, Google DeepMind opted for two deep convolutional neural networks as their base for both evaluation and move generation [55]. The networks were the Value and Policy networks, the value network outputs a single number within (-1, 1) representing the value of the position, where negative values indicate the position is losing and positive is winning. The policy network outputs the logit probabilities of the next move being played in a given space and is essentially used to narrow the search space during move selection.

AlphaGo used a Monte Carlo Tree Search, *MCTS*, over the usual minimax algorithm for searching and move generation which combined the value and policy network outputs along with the number of times a position has been visited. Search algorithms are not the main focus of this study, so the MCTS algorithm used will not be analysed in depth, but for the sake of understanding how the network outputs are used will be discussed briefly here. The algorithm loops over four main steps, Selection, Expansion, Simulation

and Backpropogation (not to be confused with the backpropogation algorithm used in Neural Networks). The Selection stage will simulate games from the current board state, s_{root} along previously explored states, with actions, a, selected for each player from the state at time t, s_t. The action taken at time, t, is then described as:

$$a_t = argmax_a(Q(s_t, a) + u(s_t, a))$$

where $Q(s_t, a)$ is the action value of the state-action pair, (s_t, a), and is a weighted average of state values obtained via prior searches. The function u is a bonus to the action value

$$u(s_t, a) \propto \frac{P(s_t, a)}{1 + N(s_t, a)}$$

where $N(s_t, a)$ is the visit count for each position and $P(s_t, a)$ are the output probabilities of the policy network for each legal action, a. This action selection method therefore takes into account move value, move probability, and visit count to select a high value, high probability and under-explored a_t at each s_t.

Actions are selected this way until reaching a state that has no previously explored actions, at which point the Expansion stage begins. The Expansion stage randomly selects one or more actions using the policy network. The resulting states, s_L, called leaf nodes or leaf states.

The Simulation stage then evaluates these leaf states using the value network output at a given state, $v_o(s_L)$. This is then combined with the result of a fast rollout policy, z_L, simulated from state s_L until game termination. This fast rollout policy employs a linear weighted softmax of easily identifiable board patterns to determine likely moves, and essentially acts as a quick and easily computed estimate of the game outcome from state s_L. It is important to clarify here that s_L in this context is the game state at a leaf node, L, and is a previously unexplored *potential* state following some action on the deepest selected s_t. $v_o(s_L)$ and z_L are combined into a leaf state evaluation, $V(s_L)$ using some mixing factor, λ

$$V(s_L) = (1 - \lambda)v_o(s_L) + \lambda z_L$$

. Finally, the Backpropogation stage updates the counts of each position visitation, $N(s,a)$ and the action value of each state-action pair, $Q(s,a)$ for use in the next loop of the algorithm.

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{n} 1(s,a,i)V(s_L^i)$$

where $1(s,a,i)$ indicates whether an (s,a) was explored during the ith simulation. We can see now that $Q(s_t,a)$ is the weighted average values of the leaf states branching from state s_t. Once this algorithm loop has been executed a pre-defined number of times, the most visited move from the root position during the search is the final move chosen to be played.

The inputs for both networks were $19 \times 19 \times C$ where C is the number of feature planes, 48 for the policy network and an additional plane indicating which player's turn it is currently for the value network. The policy network utilised 13 hidden layers, the first padded each plane to size 23×23 and convolve k 5×5 filters with stride $= 1$ and applies a ReLU activation function. Layers 2 through 12 would pad each plane to size 21×21 and convolves k 3×3 filters, again with stride $= 1$ and activated using ReLU. The final layer applied 1 filter of size 1×1 with stride $= 1$ and individual biases for each position followed by a softmax function. The value network is identical from layers 2 through 11, and features an additional convolutional layer 12 before applying a single filter of size 1×1 with stride $= 1$ and ReLU activation (similarly to the final layer in the policy network, without individual biases). Layer 14 is a fully connected linear layer with 256 ReLU units with the final fully connected layer featuring just a single tanh unit. The number of filters for the convolutional layers, k, was varied and results recorded for values k = 128, 192, 256 and 384.

These networks were initially trained using data collected from real games using supervised learning, with a single distinct position selected from each game to ensure each position was independent. The networks were later trained using reinforcement learning, with newer networks playing against older versions of AlphaGo. The value network tended to overfit when using this method,

so they used a similar method to the supervised learning using positions from simulated games as the dataset rather than real positions. Both networks were trained using Stochastic Gradient Descent, SGD, minimising the Mean Squared Error, MSE. DeepMind found that the value network trained using positions from simulated games performed better than using positions from human games. Interestingly, the policy network tended to optimise for a single "best move" when trained on simulated games and training on human games instead gave a more diverse range of promising moves, something intuitively more desirable for the search algorithm. This version of AlphaGo defeated the European Go champion Fan Hui 5 - 0 in October 2015 [55].

A year later, DeepMind published an updated model architecture that combines the Value and Policy networks into a single network with value and policy "heads" with outputs similar to the previous networks. They found that this structure lead to significantly improved performance, winning 100 - 0 against the previous iteration of AlphaGo that defeated Lee Sedol, which was in turn stronger than the previously outlined version above. Unfortunately, it is difficult to determine the exact structure of the model that defeated Lee Sedol as the matches took place between the two publications, although it is likely that the elements of that model will be present in one or both of the published models [56].

The updated network took 17 19×19 planes. Eight of these planes consist of binary values indicating the location of each players stones for the past 8 moves (1 for the player's stone, 0 for empty and -1 for the opponents stone), another 8 planes of the corresponding information for the opponent player (essentially multiplying the previous 8 planes by -1), and the final plane a constant plane of 0s or 1s indicating which player is next to play.

The AlphaGo network consisted of a single 3x3 Convolutional layer with a ReLU activation, with 17 input channels and 256 output channels which would then feed directly into 19 or 39 consecutive ResNet blocks and finally split into the two heads.

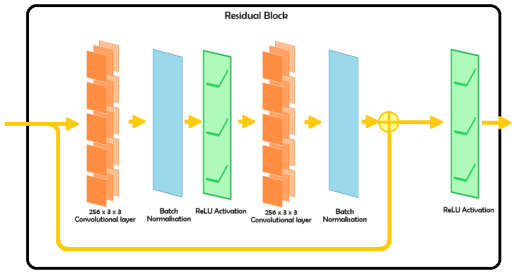


Figure 2.2: AlphaGo "Residual Block" Diagram

The foundation of the network is the Residual block, "ResNet block", consisting of 2 3×3 Convolutional layers each followed by a ReLU activation function and a batch normalisation layer, and finally a skip connection, where the input of the ResNet block is added back to the output before being passed into the next block. The ResNet block is illustrated in Figure 2.2.

The ResNet torso was capped off by two "heads", the value head for the position evaluation, and the policy head for move generation. The policy head consists of a 1×1 convolution with batch normalisation and a ReLU activation, taking inputs of 256 channels and outputting 2 channels, these are then passed through a fully connected layer and another ReLU activation outputting a $19 \times 19 + 1$ vector of logit probabilities representing the moves for all spaces and a pass move. The value head is similarly structured, with a 1×1 convolution with batch normalisation and a ReLU activation, again taking inputs of 256 channels but only outputting 1 channel. This is also followed by a fully connected layer of size 256 with another ReLU activation, followed by a final fully connected layer to a single tanh unit [56].

This network was fully trained using self-play and lead to another paper in which DeepMind applied the bulk of this model to other games, such as chess and shogi, after adjusting the inputs and outputs appropriately. The paper outlines that AlphaZero (no longer just a Go master) outperformed the leading chess and shogi engines even with significant time disadvantages, beating Stockfish, the world champion chess engine at the time, 29% of the time and losing 0.4% of the time as white, while as black winning 2% and

losing 0.8%, with draws making up the remaining results. AlphaZero also beat the champion shogi engine, Elmo, 84.2% of the time as white, and 98.2% of the time as black. Tweaks to the MCTS algorithm also lead to AlphaZero outperforming the previously outlined AlphaGo in matches between the two [54, 50].

Since then, a branch of Stockfish has introduced Neural Networks as part of its evaluation function and move selection, known as Stockfish-NNUE. Additionally, a new chess engine has been developed based on the AlphaZero structure called Leela Chess Zero, also known as "LCZero" or "lc0", and has placed in the top three in every computer chess tournament it has competed in since 2018.

Here, we would like to bring attention to the evolution of computer Chess and Go engine value functions, from the valuation of explicitly and carefully hard-coded features to the use of Neural Networks allowing a machine to identify and weight features independently through reinforcement learning. We can see that, especially in games more complex than chess, using this approach can perform better than the alternative of using human-identified features as a basis for valuation.

2.2.3 Football

> If I have to make a tackle, then I have already made a mistake.
>
> *Paolo Maldini*

While football suffers from much the same problems as baseball and basketball in terms of secrecy regarding the top level analytics advancements, the football analyst community is much more open to releasing and explaining past methods to the public. Barcelona FC in fact host a conference where analysts are able to present their ideas to their peers, and David Sumpter, a consultant for many professional European football clubs, has developed a set of material

designed to build the foundations of the skills required to pursue a career in football analytics. The material is free to access and features many current professionals working at elite clubs such as FC Barcelona and Liverpool FC. From this we can get an understanding of the recent state of analytics in football and the trajectory, leading to an educated guess of the current state of the art in the industry [69].

As with baseball, the basic statistics collected in football give an extremely limited understanding of the game. In 1950 Charles Reep was one of the first to annotate the events of a football match, and surely the first with the intention to gain a deeper understanding of the game. Reep's findings perfectly embody the risks and rewards of analytics in sports. The first noteworthy finding was that teams who managed to regain possession near the opponents goal were significantly more likely to score than after any other sequence, leading to the further development of the *pressing* game. "Pressing" describes the strategy of a team's forward players, usually the focal points of the attack, harrying the opponents defender's for the ball with the idea of pressuring the opponents' less-technically-skilled players for the ball in dangerous areas. Today, you will struggle to find a top level team that doesn't press the opponent in some capacity, showing the potency of this strategy.

Reep also found that roughly one in nine shots were scored and that over 90% of passing sequences never reach four consecutive passes. Reep concluded that teams should aim to get the ball into a viable shooting position as quickly, and with as few passes, as possible. This became known as *Route One* football. This is by no means an incorrect way to play, but many top teams today have found that playing simpler passes and keeping possession is in fact an effective strategy. Keeping possession allows a team to pick much higher quality shooting opportunities while limiting the opponents offensive capabilities as they have no opportunity to attack without the ball. Where Reep saw low pass completion and scoring probability as reason to pursue a "quantity over quality" approach to maximise goals scored, modern coaches may see these as

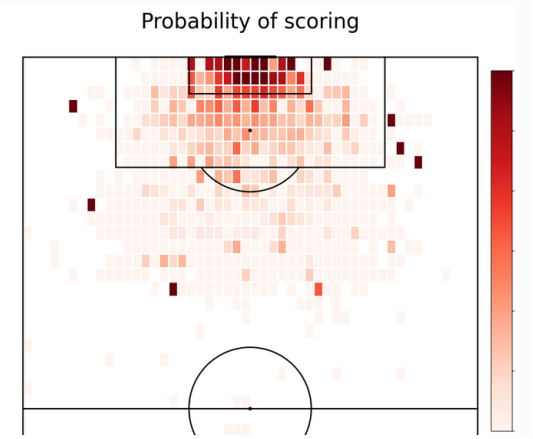


Figure 2.3: Example of an xG Map (Sumpter and Andrzejewski [71])

areas to improve to maximise goals scored while minimising goals conceded [2]. This example illustrates the difficulty in determining when the numbers are an indication of something true, or are being affected by other factors with human biases preventing the analyst from acknowledging this.

Above, we touched on the idea of higher quality shooting opportunities, the fact that the probability of scoring is inversely proportional to the distance from goal. This was quantified by Sam Green in April 2012, who developed the "xG" model computing the probability of scoring from each position on the pitch, seen in Figure 2.3 [66].

There have been previous models for the probability of a given shot being scored, taking into account things such as distance, angle to goal, and many other factors. Typically these perform well but have the requirement of data that may not be collected at every game. In contrast, the xG model divides the pitch into areas and finds the number of goals scored from each location divided by the number of shots from each location, yielding a useful metric of goal-scoring opportunities which does not require any additional data. We can take this model, which is essentially the distance and angle terms of previous models, and build on top any additional modelling required.

The natural extension of this idea is the question "Given a certain state of the game, how likely is the team in possession to score?". This question was first addressed publicly by Sarah Rudd in 2012 in her presentation "A

Framework for Tactical Analysis and Individual Offensive Production Assessment in Soccer Using Markov Chains" [49] where she describes how she uses Markov Chains to quantify the probabilities of scoring at the end of a possession sequence given the state of the game. Rudd is then able to quantify how the actions of individual players improve or damage their teams probability of scoring. She defines 39 game states: 2 absorbing states (goal or end of possession), 7 set piece states and 30 states defined by zone and defensive states. She then uses a 39x39 transition matrix, representing the probabilities of transitioning from each of the 39 states to any other state, including staying at the current state. She then takes advantage of the property of Markov Chains which allows us to multiply the transition matrix by itself until the probabilities converge, giving the final probabilities of scoring at the end of a sequence given a particular state of the game. Player actions can then be valued by taking the probabilities of scoring before and after an action and summing and/or averaging all of the players actions to give a measure of a player's contributions to their team.

Nowadays, this idea is usually called *Expected Threat* or xT, and can be divided into two approaches, Positional and Action based xT. Positional xT is similar to our xG map above, where for every position on the pitch, the probability of a goal being scored before the possession ends (before a turnover or the ball going out) is calculated and plotted. Following the methodology in Rudd's work, we can rank each player by their total xT contribution [58].

The initial implementation of xT divided the pitch into 12×16 zones and constructed a transition matrix from these using the probability of moving from each zone to each other zone and the probability of scoring from each particular zone. The matrix is then multiplied similarly to the earlier discussed Markov Chain implementation by Rudd. The difference between this implementation and Rudd's work is that this approach works backwards from the assumption that the possession ends in a shot as the possibility of losing the ball is not taken into account. We can consider this as the probability of scoring from a

position within n actions, where n is the number of multiplicative iterations performed. This approach purely uses evenly sized zones and scoring as the potential states. Compared to Rudd's 39 states, including 7 for set pieces, we can also think of this as an almost purely positional implementation of xT.

The second approach, the Action Based xT, groups chains of actions leading to some outcome together, e.g. all the passes and dribbles between regaining possession of the ball and some terminating outcome, like a shot or turnover, will be grouped with that outcome. A simple application of this thinking is xGChain, where all the players involved in a sequence leading to a shot, by passing, dribbling or shooting, will be assigned the xG value of the shot. This leads to the players most involved in a teams attacking sequences to gain the highest score. While this does tend to give reasonable results, it is tended to be weighted towards players at better teams as they are more likely to be involved in sequences leading to shots, as all sequences from the best teams are more likely to results in a shot. Another method is to take the start and end coordinates of each action in the chain, along with the result of the sequence and fit a machine learning model, typically xGBoost [70], to predict the outcome of the sequence. This, in theory, groups similar actions and estimates the value added by actions of this type. For example, if we take 10 hypothetical similar passes, and 1 of these ends in a goal, then passes like this are worth roughly 0.1xT [70].

Like baseball, football as a sport has trended towards increased data collection and analytics in recent years, and has introduced player tracking in an attempt to gain deeper understanding of how player positioning affects gameplay. Potentially the most well-known and widely-used is the "Off Ball Scoring Opportunity" or "OBSO" developed by William Spearman, now head analyst at Liverpool FC, in "Beyond Expected Goals" (2018) [62]. He attempts to address the significant shortcoming of previous models that they cannot account for actions that could have been taken to score but were not. The prime example Spearman leans on is a striker waiting at either goalpost for a cross

that never arrives. OBSO attempts to quantify the probability of scoring after the next pass, where the striker is given credit for taking up a good position rather than being punished for their teammates inability to deliver a pass. To do this, Spearman combines his prior models for spatial control and pass probability [64, 63] while incorporating scoring probability.

The control model assumes that a player's ability to control the ball depends on the time available, i.e. that the longer a player has to control the ball without defender interference, the more likely they are to be able to do this. They model this as a Poisson point process. The model seeks to quantify the control probability for each player for each position on the pitch, and is denoted by C. The arrival time of each player to a location using the average sustainable velocity and acceleration along with the player's current location and velocity produces a model with fairly large variance between the expected arrival time and the observed arrival times. To account for this they choose to model the distribution of the arrival time using the logistic function.

They then use a physics-based flight time for the time it takes for the ball to travel to a position from its current location, assuming the most advantageous pass for the attacker (as they will be the ones making the pass) such that the ball and player arrive at similar times. The final component is the control rate, λ, which is essentially a typical rate parameter for the Poisson distribution, although here they do scale this for defenders to account for the fact defenders likely do not need to control the ball as well as attackers (e.g. a clearance or header away from goal is easier than controlling and shooting).

The transition model seeks to quantify the probability of the next on-ball action being taken at a given location, denoted by T. They model this using a 2-dimensional normal distribution centered on the current location of the ball, as most subsequent actions tend to occur within a very close proximity to the current location (tackles, interceptions, passes, blocks etc.). They also incorporate the fact that actions won't be purely random, and that passes are likely to be directed towards spaces where they are unlikely to be intercepted

or blocked by opposition. They do this by superimposing the control model described above for all attacking players over the normal distribution here, weighting the probabilities towards areas the attacking team are likely to be able to control the ball.

The final model, the Scoring probability, denoted by S, essentially models the previously mentioned xG as a continuous function purely based on distance from goal, rather than a discrete function based on zones. They justify the simplification of the model by asserting that the other factors usually included in xG models will be modelled by proxy via the two prior models. It is worth noting the control and scoring models are not normalised in the spatial dimension. To further clarify, they quantify probabilities at each location, while the transition model quantifies the probability of the next on-ball action occurring at a given location and will integrate to 1 in the spatial dimension.

To finish, the total probability of scoring, $P(G)$, given a particular game state, D, can be obtained by combining the three models above:

$$P(G|D) = \sum_{r \in \mathbb{R} \times \mathbb{R}} P(G_r \cap C_r \cap T_r | D)$$

where r describes some point on the pitch and G_r, C_r, T_r indicate the outputs of these models at that point. The above equation can be expanded when considering the above models separately, with S_r being a simplification of G_r merely describing the probability of scoring from point r independent of the probability of the next on ball action occurring there. Thus,

$$P(G|D) = \sum_{r \in \mathbb{R} \times \mathbb{R}} P(S_r|C_r, T_r, D) P(C_r|T_r, D) P(T_r|D)$$

yields the probability of a player performing the next on-ball action, controlling the ball and scoring at each position on the pitch. This is encoded as a 3-Dimensional matrix where the third dimension are the individual players.

The use cases of this model are multi-faceted. One is the ability to integrate over the spatial dimension to obtain the total probability each player will score with the next on-ball action as a measure of player contribution. One can also

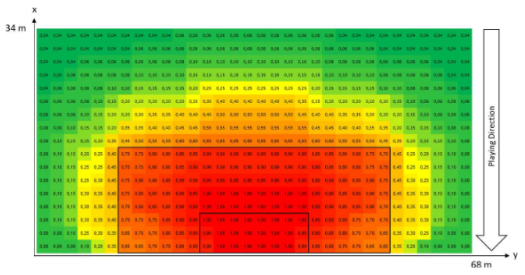


Figure 2.4: Dangerousity Zone Values (Link, Lang, and Seidenschwarz [33])

integrate over the player dimension to obtain the total probability a team will score at any given location on the pitch as a measure of identifying danger areas in coaching. Finally, we can integrate over both to obtain the probability a team will score with the next action as an overall measure of team performance.

Another well circulated approach among the professional community is the Dangerousity metric, derived by ex-professional footballers in an attempt to quantify the intuitive understanding gained from playing experience [33]. Dangerousity takes four key aspects; Zone, Control, Pressure and Density to determine the value of a given game state. The formula for determining the Dangerousity at a given time, $DA(t)$, is:

$$DA(t) = ZO(t) \times \left(1 - \frac{1 - CO(t) + PR(t) + DE(t)}{k_1}\right)$$

where $ZO(t), CO(t), PR(t), DE(t)$ represent the effects of Zone, Control, Pressure and Density at a given time, t, respectively. The constant k_1 is used to scale the reductive effect of Control, Pressure and Density on the Zone term, i.e. the Zone term can be thought of as the base value while the others act as contextual adjustments.

Zone is quantified similarly to the xG or xT maps, where 2×2 metre squares are assigned weights describing the value of the zone. The authors do not go into detail about how they derive these values, however they do seem to correlate roughly with xT with a slight bias towards areas closer to goal as can be seen in Figure 2.4.

Control is quantified using the average relative velocities of the player and

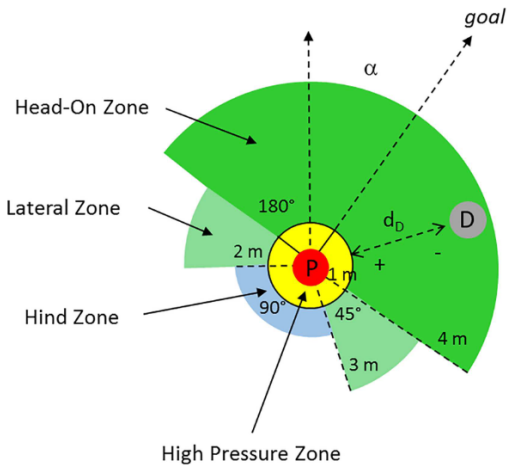


Figure 2.5: Dangerousity Pressure Zones (Link, Lang, and Seidenschwarz [33])

the ball in the last 0.5s prior to time t. They assume a quadratic relationship between relative speed and a player's ability to control the ball:

$$CO(v_{rel}) = 1 - (k_2 \times v_{rel}^2)$$

where v_{rel} is the average relative velocities of the attacking player and ball and k_2 is a constant which scales the model output such that if v_{rel} is greater than $25m/s$, then $CO(t) = 0$. In this equation, they take the max of 0 and $CO(v_{rel})$, and as v_{rel} can only be positive, $CO(v_{rel})$ only takes values between 0 and 1.

Pressure is assumed to be exerted by a defender nearby the attacking player. The space around the attacker is divided into three zones, the Head-on zone, Lateral zone and the Hind zone. The Head-on zone is the full 180^o facing the center of the goal, from the player's perspective and has a radius of 4m. The Hind zone is 90^o directly opposite the head-on zone and the goal with radius 2m, while the Lateral zones are the two 45^o regions between the Head-on and Hind zones, with radius of 3m. The direction the attacking or defending player's are facing does not matter for calculating pressure. A diagram of the pressure zones is given for clarity in Figure 2.5.

The pressure each defender exerts on the attacker is modelled by the equa-

tion:

$$PR_{Di}(d_{Di}, \alpha) = 1 - \frac{d_{Di}}{r_{ZO}(\alpha)}$$

The label α is the region the defender is located in, d_{Di} is the distance the defender is from the attacker, and $r_{ZO}(\alpha)$ is the radius of the zone the defender is located in. The overall pressure exerted on an attacker, PR, is given given by the equation:

$$PR(x) = 1 - \exp^{-k_3 x}$$

where x is the sum of pressure exerted by all defenders in pressure zones, and k_3 is another model constant.

Finally, the density is modelled by a weighted sum of the pass and shot density, where the weight is based on the centrality, C, of the attacker, so that if the attacker is perfectly central, then only the shot density is taken into account, whereas, if the attacker is near the sidelines then only the pass density is used:

$$DE(c) = C \times SD + (1 - C) \times PD$$

where C is the centrality of the attacker, SD is the density of defenders in the shooting zone, and PD is the density of the players in the passing zone. SD is the sum of individual density added by defenders in the shooting zone, where the shooting zone is a polygon formed by taking the points 2m either side of the goal and 2m either side of the attacker (perpendicular to the goal). The individual density is found using:

$$SD_D(d_D, d_{goal}) = 1 - (d_D)\frac{d_D}{d_{goal}}$$

where d_{goal} is the distance from the player to the centre of the goal, and d_D is the perpendicular distance between the defender and the centre line described by d_{goal}, visualised in Figure 2.6.

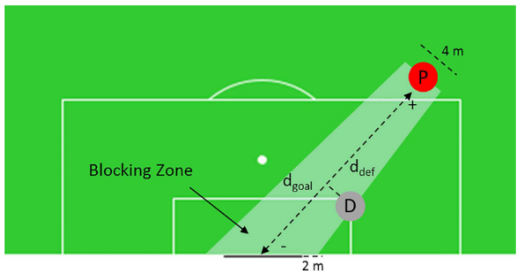


Figure 2.6: Example of Dangerousity Shot Density (Link, Lang, and Seidenschwarz [33])

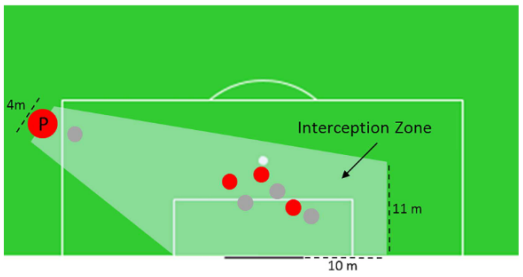


Figure 2.7: Example of Dangerousity Pass Density (Link, Lang, and Seidenschwarz [33])

The Passing density is found by taking the number of defenders vs attackers in the passing zone. The passing zone is the polygon formed by the points 2m either side of the passer (perpendicular to the centre of goal), the nearest intersection between the goal line and the 6-yard box, 10m to the opposite side of the goal and 11m vertical (relative to the pitch) from the previous point, seen in Figure 2.7.

The pass density is modelled by the equation:

$$PD(M) = 0.5 + \frac{\tan^{-1}(k_5 M)}{\pi}$$

where M is the number of defenders minus the number of attackers and k_5 is another model constant. The $\tan^{-1}$ function ensures that the effect of the players in the box is bounded by -1 and 1, so $PD(M)$ will take values between -0.5 and 1.5.

The value of player actions can be easily found once the Dangerousity

model has been established, where action value is simply the difference in Dangerousity of the game state before and after the action.

The evaluation of this model consisted of taking a number of real-life game states and asking professional coaches in Germany to rate them from 1 (not threatening) to 5 (extremely threatening) and comparing these to the output from their Dangerousity model. They found that in general, their model correlated strongly with the ratings provided. This result proves they have accomplished their goal of quantifying the intuition gained from professional experience. They have also compared their model to basic methods of game evaluation as predictors of win probability. They compared their model to Goals, Shots taken, Pass accuracy, Tackle rate and Ball possession, and found that their model was the best predictor for win probability of the factors analysed.

Neural Networks have also been applied to football, although not widely circulated by the professional community to the public. Academic researchers have shown that these techniques can yield positive results in this domain. Calvin C. K. Yeung, Tony Sit, and Keisuke Fujii developed a deep neural network based on the transformer framework to predict where, when and what the next event will be [75]. Their model is trained on a sequence of events, their locations, the time since the previous event and 5 features describing how the ball is progressing through the sequence, and outputs a prediction of the next event type, when it will occur and the zone of the pitch it will occur in. To do this they divided the pitch into the "Juego de posición zones" (fig. 2.8) (A method of dividing the pitch developed by coaches, described in more detail in "Juego de posición – A short explanation" RM [48]) and assigned each on a random ID. They then combined this with the time since last event and the event type ID along with the 5 additional positional features: distance to the previous events zone, change in x&y zone coordinates and distance and angles from the centre of the zone to the goal.

Therefore the input is a 2D tensor of size (seq. length, 8) and output is a

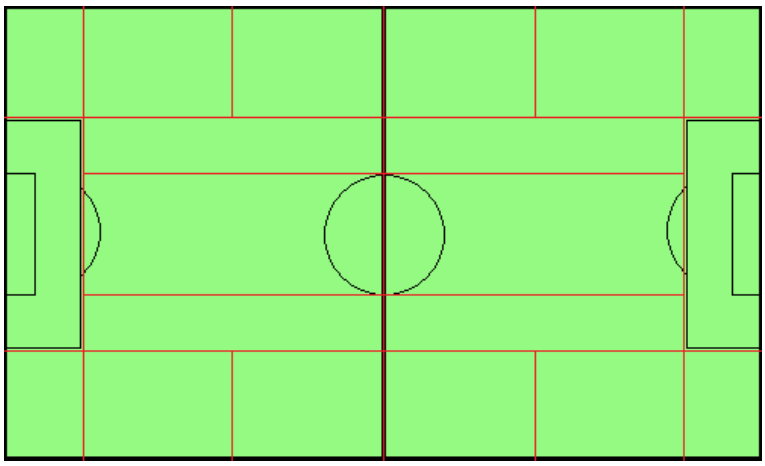


Figure 2.8: Juego de posición zones example

tensor of size (1, 1 + E + Z) where E is the number of event types and Z is the number of zones. 5 event types were observed (shot, pass, dribble, cross and end possession) and 20 pitch zones were used. The zone and event predictions were scaled to [0, 1] to provide probabilities for each zone and event, while the time until next event is left as a continuous variable.

The first stage is called the "Historical Encoding" stage and looks to extract the relevant features from the event sequence. The zone and the event type inputs are each passed through their own embedding layers, with output size of (seq. length, 5) for the event type and (seq. length, 20) for the zone. The other features are passed through a dense layer with output size (seq. length, 6). These three outputs are then concatenated and passed through another dense layer with output size of (seq. length, 31), followed by a transformer layer with an output with size (1, 1024), this is then followed by another dense layer with output size (1, 31).

Forecasting is done using three neural network blocks. The first consists of 2 dense layers and takes the 1x31 tensor from the historical encoding block, and outputs a single node representing the time until the next event. The second block is another 2 layer block and takes the output from the time block and combines it with the historical information, size (1, 31 + 1), and outputs a 1×20 vector of probabilities each representing the likelihood of the next

event occurring in a particular zone. The final block takes the outputs from the previous two blocks and the historical information, size (1, 31 + 1 + 20), and passes it through three fully connected layers, outputting a 1×5 vector of the probabilities of the next event being a particular event type.

The cost function used to train the model was a combination of Root Mean Squared Error, "RMSE", for the inter-event time and Cross Entropy Loss for both zone and event types. The model was compared to four baseline models, the first being an auto-regressive model of the events coupled with the transition probabilities between each zone, the other three baseline models took a pre-trained model called "Seq2Event" and added a final layer (one LSTM model, two Transformer models) before training again with the same cost function discussed above. One of the modified "Seq2Event" models was also further modified by increasing the size of the pre-trained model's transformer encoder layer from 8 to 2048. They found that apart from the auto-regressive model, performance was relatively similar across the remaining models, with the subject of the paper performing slightly better than the baselines in the $RMSE$ and CEL components, while training in less time than the nearest alternative.

CNNs have also been applied to this problem in football. Uwe Dick and Ulf Brefeld developed a network designed to value game states directly. They take "episodes" of uninterrupted possession by a team and label each with the outcome, in this case they use whether the in-possession team enters the attacking third of the pitch and the opposition penalty box, although this choice is arbitrary and the outcome labels could easily be shots on target or goals instead. Episodes consist of consecutive game states describing the x/y coordinates of each player on the pitch and the ball. These game states are recorded 25 times per second. They used a Reinforcement Learning approach, which is typically used to optimise controller behaviour, but is used here to measure the effectiveness of individual players [12]. The loss function is rather complex and is best understood in stages. The first equation to consider is the

estimated total value of the sequence up to state s_{t+n}, starting from state s_t.

$$G^n_{\hat{V}}(s_t) = \sum_{t'=1}^{n} \gamma^{t'-1} R(s_{t+t'-1}, s_{t+t'}) + \gamma^n \hat{V}(s_{t+n})$$

This essentially takes the returns for each state from time t to $t+n-1$, 1 for the penultimate state in a sequence that leads to success in the final state, and 0 otherwise (this does mean the majority of R values will be equal to 0). The last component is the model estimate of the value for state s_{t+n}, the model coefficient γ is equal to 0.95 in this study. Additionally, if $t+n$ is greater than the length of the episode, T, then the above equation just takes the value of the first sum term up to $n = T-1$.

$$G^n_{\hat{V}}(s_t) = \sum_{t'=1}^{T-1} \gamma^{t'-1} R(s_{t+t'-1}, s_{t+t'})$$

The second equation then takes this equation and applies it as such:

$$G^\lambda_{\hat{V}}(s_t) = (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G^n_{\hat{V}}(s_t) + \sum_{m=T-t}^{\infty} \lambda^m G^{T-t}_{\hat{V}}(s_t)$$

which is the weighted sum of the outputs from the above equations from n = 1 to the end of the episode ($G^n_{\hat{V}}(s_t)$ term), and the real return of the episode ($G^{T-t}_{\hat{V}}(s_t)$ term). Here, λ is another model coefficient, in this paper they use $\lambda = 0.7$.

The loss function, which the authors call the "lambda-loss function", is then the mean squared difference between $G^\lambda_{\hat{V}}(s_t)$ and the model estimate of the value of state s, $\hat{V}(s)$.

The model structure is comparably simple, it takes a 9 channel input with the same dimensions as the state image. The first three channels are zeros with the locations of the ball and the players from each team set to 1, the first channel represents the ball location, the second represents the team playing left to right, and the third channel is the team playing right to left. Channels 4 and 5 are the x and y speeds of the players from team 1, at the player locations, while channels 6, 7, 8 and 9 are the equivalents for team 2 and

the ball respectively. The model consists of 3 consecutive 6×6 convolutional layers with ReLU activations each outputting 32 channels. The final layer takes the flattened output from the final convolutional layer and combines it with a possession indicator (1 or -1) and passes this through a 256 node, fully connected layer with a ReLU activation. The output layer consists of a single node representing the predicted value of the game state.

The authors used AUC as the measure of model performance, where AUC is the Area Under the Receiver Operating Curve (ROC). They calculated the AUC for predictions made each second from 3 to 15 seconds, and found that while AUC does decrease the further from the end of the episode from around 0.85 at 3 seconds to 0.65 by 15 seconds, this still indicates better performance than uninformed guessing, 0.5. They also compared the AUC for predictions made a certain distance from goal; as expected, the closer to the opponents goal the ball is, the more accurate the predictions, with an AUC of just above 0.6 at the furthest point analysed to 0.75 at the closest starting point analysed, indicating that the model also performs better than uninformed guessing based on distance from goal.

A further extension to this published in 2021, using Graph Recurrent Neural Networks, $GRNN$, rather than Convolutional [13]. Each player and the ball is represented by a node in a graph with connections being represented by edges. This model uses a fully connected graph meaning each node, v^i, is connected to every other node, v^j where $j \neq i$, via edge e^{ij}. Like their previous model, a game is split into episodes of continuous possession made up of game states. The game state is passed into the network and is encoded into the node states via two fully connected layers with 128 and 256 units respectively. The graph state is then the sum of the encoded Graph Node values for a given layer and time step. Each of these are passed through a series of Graph Layers along with the node and graph states of the previous layer and the previous time step from the equivalent layer. It is important to note the initial game state is passed into each Graph Layer. This outputs the updated graph and

node states, both of which are passed to the next layer and the same layer for the next time step. Each time step will therefore output a node and graph state which can be used for prediction of the rest of the episode.

Each Graph layer takes the inputs described above and uses an attention mechanism from the corresponding nodes to compute each edge feature, $e^{ij}_{l,T}$, for each layer, l and time step, T. These are then aggregated into intermediate node features and passed through standard Gated Recurrent Units to produce the updated node features, $v^{i}_{l,T}$. The intermediate node features are also aggregated through another attention mechanism to generate the updated graph state, which comprises the Graph Layer output along with the updated node states.

The model uses the final node states to predict the movement of each of the players and the ball using a categorical mixture model, with K = 6. The mixture distribution is modelled using two fully connected layers, and is combined with a Gaussian model with means and variances also modelled using two fully connected layers. Each of the above layers contain 512 units with ReLU activations and softmax, linear and exponential output functions for the movement, Gaussian mean and Gaussian variance respectively. This prediction component is trained by minimising the log-likelihood of the observed movements. The model also predicts the outcome of the episode similarly to their previous paper, the graph state is passed through another two fully connected layers with 512 units. This component uses the lambda-loss function described above from their earlier paper for training. The results of this paper show that the model was effective at predicting the movement of players, and as one would expect, these predictions were closer to the observed movements the smaller the time period of prediction. The predictions were also better the more defensive the players were, predictions for defenders tended to be closer to the observed than midfielders, which were closer than wingers and forwards. The success prediction was able to be directly compared to their previous model, and they found that the GRNN model performed better (higher

AUC) at every starting distance than the CNN model, from an AUC of around 0.7 for the furthest starting point to 0.85 for the nearest, compared to 0.6 to 0.75 for the CNN model.

One application brought up in this paper is the ability to quantify a player's off-ball positioning. They state that due to both models including the player movement as well as their locations, merely adjusting the players location to find the optimal point for comparison would not be feasible as the movements may become impossible or unreasonable within context using this method. Instead, they use their player movement predictions essentially as a proxy for the average movement of a player in similar positions and using that to quantify relative off-ball positioning of a player.

Potentially, the best indication of the current state-of-the-art in football analytics is a paper by Javier Fernández (FC Barcelona), Luke Bornn (Simon Fraser University) and Daniel Cervone (Zelus Analytics) published in 2021 [17]. The authors use an ensemble of CNNs to estimate the probability of a pass, "ball drive" (often called a dribble), or a shot, along with the Expected Possession Value, EPV, of each of these. They also incorporate more than just the positional data into their model, such as the shape and distance of each defensive line, the number of players between the ball and each location etc. There are a number of these features, labelled "Contextual features" which have been developed using in-depth domain knowledge. Overall there are 21 "Spatial features" (player positions, angles and distances to goal etc.) and 14 "Contextual features" although each model within the ensemble may not use all possible features. The authors provide an appendix indicating which features are used by which models.

The ensemble consists of 7 models, for pass success probability, pass selection probability, pass EPV, shot success probability, ball-drive probability, ball drive EPV and finally action selection probability. The models for ball drive success probability, shot success probability and action selection probability are just shallow Neural Networks consisting of 2 fully connected layers each

with ReLU activations, the outputs (1 for the two action success networks and 3 for the selection network). The success networks use MSE loss functions while the selection network is trained using a cross-entropy loss function. The models are of course trained only on applicable data, i.e. the action selection model is trained on all points (as a decision is always being made), but the shot success and ball drive success models are only trained on data where these actions are attempted.

The remaining models are based on the same core architecture, using Convolutional layers on downscaled representations of the game state before re-upscaling the output and recombining the results to form an output of the probability of success, or value, at each point on the pitch. To be more specific, each model takes an input of $104 \times 68 \times K$, where 104 and 68 are the pitch length and width respectively, and K is the number of input features for the given model. The network essentially consists of upscaling blocks and convolutional blocks, linked by "fusion layers" and 2×2 max pool layers for downscaling. The convolutional block will take the input tensor, and pass it through a $5 \times 5 \times 32$, a $5 \times 5 \times 64$, and a $1 \times 1 \times 32$ convolutional layer with ReLU activations, followed by a linear $1 \times 1 \times 1$ convolution. An upscaling block will upsample the tensor by $2\times$, and pass it through a $3 \times 3 \times 32$ ReLU-activated convolutional layer followed by a linear $3 \times 3 \times 1$ convolutional layer. Finally, the fusion layers are merely a $1 \times 1 \times 1$ linear convolution of two input channels. Using these blocks the authors will take the input game state, pass it through a convolutional block, $1 \times \mathit{output}$, pass this through a max pooling layer to half the height and width dimensions, this is then passed through another convolutional block, $1/2 \times \mathit{output}$. This is repeated a third time to yield the $1/4 \times \mathit{output}$, which is then passed through an upscaling layer and combined with the $1/2 \times \mathit{output}$ via a fusion layer. This is then repeated once more, an upscaling layer and combined with the $1 \times \mathit{output}$ to give the final ouptut of size $104 \times 68 \times 1$.

The outputs of each model obviously describe different things and are thus

trained using the appropriate data, similarly to the shallow networks described previously. The pass success model outputs the map of probabilities of a pass to a given location being successful, and is trained by taking the output probability at the observed end location for a pass and computing the negative log-loss between the two. The expected value of passes is actually two models using the above architecture, one trained purely on successful passes, and the other on unsuccessful passes, with a Mean Squared Error loss function for both between the predicted value and the observed outcome, the expected value model for ball-drives and shots also uses this same process using the relevant ball-drive and shot data. The pass selection model outputs a probability density across the entire pitch using the softmax function on the output pixels, as the sum of probabilities for all possible passes given a pass will be attempted should be equal to 1. This model is trained using the log-loss of the prediction and 1 at the location the pass was observed (i.e. the loss for a pass is dependant on the predicted probability at the observed location with all other probabilities remaining unused).

A point of differentiation from other work of this type is the method of defining the successes and failures for the Estimated Value models. In this paper, they define a successful action as any action that occurred between the last stoppage and the next goal, where stoppages are defined as only goals, end of half or the end of the game. Notably, they exclude changes in possession or the ball leaving the field of play from this definition. Therefore, every action is assigned a value 1 if the team currently in possession scores before the end of the half/game, 0 if the half/game ends before either team scores, or -1 if the opposing team scores before the end of the half or game. Unintuitively, this means a pass in the first minute of the game would be assigned a value of 1 if the team in possession manages to score 30 minutes later, however, they are able to achieve remarkable results using this method. Although they do not compare the results of their model to a baseline, they do achieve low losses across their models and a low expected calibration error, which essentially

allows one to compute the average error on a per sample basis while also breaking the task into manageably sized bins [20].

This is perhaps the best indication of the current standard of professional analytics in football. In this section we also see some techniques applied to a continuous game rather than a game consisting of turns or plays. We will see there is significant overlap in the thought processes behind the techniques seen here and the ones to be covered when discussing analytics in Basketball, and as such we will be looking to take inspiration from materials developed in this sporting discipline due to the similarities of both sports.

2.2.4 Basketball

> I've always believed analytics is crap.
>
> *Charles Barkley*

The NBA has been slower to adopt analytics in general compared to most other sports, with teams only gaining convenient access to play-by-play data in 2009, and roughly half the teams even having what could be considered an analytics department at the time [51]. Uptake among athletes and media has been especially slow, with multiple active players expressing skepticism [44]. There have, of course, been organisations that commit fully to the idea of statistically-backed decision making. In 2015 ESPN published a comparison of the NBA analytics departments, as well as other American sports and highlighted Dallas, Houston, Philadelphia and San Antonio as the clear stand outs in the NBA [16]. As early as 2013 the Toronto Raptors had developed a system that analyses the defensive positioning of their players, although the accuracy and methodology were never disclosed [35]. The system employed the use of the then-relatively-new SportVU positional data; it would pair this with some representation of the players' abilities and compute the optimal defensive positioning and opposition match-ups for their players to take. Consequentially,

this requires some modelling of the expected value of the possession, which they compute multiple times per second. Given that this was revealed a decade ago, it is fair to assume that current methods are significantly more sophisticated, but are also along this same trajectory of utilising positional data to maximise the expected value of a team's possessions.

Analytics in the NBA are used for more than just squad building/player selection. In 2022 the Raptors also revealed a coaching tool utilising real-time ball tracking used to develop player shooting [11]. The tool tracks the flight of the ball of any shot a player takes in training and provides instant feedback on different aspects of the shot via 448 video boards courtside. We can conclude that it is likely that other analytics techniques and metrics are being used to aid coaches on what players are able to improve and how. A more well known example of this is the increased focus on close-range two, and three point shots. Statisticians assert that the increased probability of scoring if a player steps within the three point arc is not worth the sacrifice of the additional point. Conversely, shots from near the rim are so much more likely to score that shooting from further out is an inefficient decision.

Similarly to baseball, the NBA has developed a number of metrics to quantify a player's contribution, but unlike baseball these are able to be constructed as "all-in-one" metrics as the game is far less segmented. In basketball, positions are not clearly determined as fielders/pitchers/batters with defined roles within discrete plays. Rather, players are expected to contribute to the team effort in all aspects of play within a continuous game. In 2021, Bryan Kalbrosky conducted a fairly informal survey of 30 NBA staff and ex-staff of various teams, whereby each respondent was asked the degree to which they trust a specific metric and whether it was their preferred all-in-one metric [28]. Of those surveyed, a number were working as executives, team analysts and coaches, meaning we can conclude some of the following metrics are currently being used in some capacity at professional teams.

We can first look at a group of these metrics together, as they are based

on the same concept of quantifying the overall contribution of a player using individual box-score data collected from each game. The most basic of which is the Plus-Minus, which essentially tracks of the number of points a team concedes subtracted from the points a team scores while a player is on the court, with a positive value indicating the team scores more than they concede with the player on the court. The idea behind this metric is that the most effective players will cause their team to score more than their opponents more consistently. Other methods like "Box-Plus-Minus", BPM, utilise additional stats like rebounds, blocks etc. to develop a slightly more robust version of a plus-minus score, less susceptible to the influence of poor shooting luck or teammates.

Often, rotation units (bench players) will play against opposition rotation units, leading to positive plus-minus scores similar to those of the starters, despite being worse players. "Adjusted Plus-Minus", APM, developed by the Dallas Mavericks, attempts to address this by adjusting the Plus-Minus score based on the quality of the players on the court, and over time the metric is able to quantify the contributions of players to their team's point differentials.

Unfortunately, this method requires multiple seasons of data to overcome the amount of noise and roster changes present in the NBA. Real-Plus-Minus, RPM, was developed by ESPN, with the specifics of this metric being known only to those at ESPN, however the general process has been disclosed and can be discussed here. ESPN constructs an advanced APM by incorporating player tracking data and private statistics collected in-house by ESPN and splits this into offensive and defensive Plus-Minus scores. These are then regularized, meaning ESPN sets Bayesian priors of the strength of the players on the court using past data and more recent form, along with projected development or decline for young or older players. The regularisation step also helps in differentiating contributions of players who tend to play together often where one individual is most likely the main cause of the good/poor performance of the entire unit [15].

RPM allegedly incorporates player tracking to some extent in deriving additional statistics used to consolidate the traditional box-score statistics, and are generally used to determine offensive-defensive player matchups (i.e. which defenders are defending each player from the opposing team), so player *RPM* is further adjusted by the players in their vicinity for each play. The statistics-themed FiveThirtyEight further incorporates tracking data in their in-house metric, "Robust Algorithm (using) Player Tracking (and) On/Off Ratings" or *RAPTOR*, using the expected value of a shot based on the shot location (similar to Football's xG discussed previously) and computes similar values for other actions, such as rebounds or steals, based on the action locations and the game situation (player locations, matchups and preceding actions) [57].

The downsides of Plus-Minus metrics are largely due to the simplicity of their outputs. The effects of any metric are not additive, as teams with multiple high Plus-Minus players often don't achieve the success the metric predicts due to clashing play-styles or multiple players filling the same niche. Additionally, the value of each metric does not scale linearly, as it is far more beneficial to have a single player with a Plus-Minus of +8 than to have two players with Plus-Minuses of +4 due to the limited number of roster spots and available positions on-court.

Another disadvantage of Plus-Minus metrics is that while some can be used to project future player performance with some difficulty, the majority are built as backward-looking measures of contribution rather than forward-looking metrics predicting future performance. Addressing this issue is "Daily Adjusted and Regressed Kalman Optimized projections" or *DARKO*, which uses Gradient Boosted Decision Trees paired with Kalman filters to predict each player's individual box-score, and updates the model with every new game's information [41]. *DARKO* uses similar inputs to the previous metrics, including opponent strength, player tracking and previous box-scores, but also includes home/away effects, seasonality and the interactions between each model component. *DARKO* also uniquely models the effects of player

aging on an individual statistic basis, as the effects of aging are different on rebounds compared to field goal percentage for example.

DARKO works using an exponential decay model, where data from each game is weighted using β^t, where β is somewhere between 0.98 and 1, a t is the number of days since the game occurred. A modified Kalman filter is used to reduce the effects of noise, essentially accounting for the tendency of players to perform exceptionally well in certain statistics for a brief period of time, while also capturing actual underlying improvements in a player's game. The results of the decay model and Kalman filter are then combined using a gradient boosted decision tree. Beyond this, the exact implementation of *DARKO* is unknown.

A Kalman filter works by minimising the error of the modelled behaviour of a system, $\hat{y}$, compared to the observed behaviour of some measurable state of the system, y. This is done in order to more accurately estimate the behaviour of a hidden state of the system, $\hat{x}$ relative to x. In this context, *DARKO* uses its inputs discussed above to model how they reflect a player's abilities. We will assume the players' underlying abilities represent the un-measurable hidden state, and their manifestation in reality in the form of box-scores are the measurable behaviour of our system. As mentioned, the results of the Kalman filter are combined with the Decay Model via a gradient boosted decision tree. A gradient boosted decision tree is a large group of small decision trees, essentially an ensemble model of very simple decision makers. Gradient boosting refers to the method of sequentially adding these simple decision trees to minimise their collective loss function. *DARKO* was tested against similar box-score predictors and performed significantly better in predicting player box-scores than its competition [41]. It should also be noted that *DARKO* produces an all-in-one metric similar to *RPM* using a similar process described above to predict player offensive and defensive value.

Player tracking data is also being used to estimate the value of possessions in basketball, similarly to those in Football and Chess. The paper "A

Multiresolution Stochastic Process Model for Predicting Basketball Possession Outcomes" Cervone et al. [7] models the expected possession value, EPV, in basketball using a hierarchical stochastic model. For this paper, the authors consider any possession as a sequence of play where possession is held by a single team which ends in a shot or turnover, i.e. excluding fouls. They begin by considering a possession as a Markov chain, where the possession can transition between different states (e.g. Player 1 in possession, player 2 in possession, shot, rebound, shot made etc.) and describing four models from which they can then estimate the possession value at any given time based on a frame or snapshot of the possession at that time, where a frame is used as it has been previously: the locations of the 10 players on the court and the ball. The four models are:

1. The micro-transition model describing the infinitesimal movement of players assuming no major movement of the ball,

2. The macro-transition entry model describing the occurrence of a transition (shot, pass, etc.) within some time period,

3. The macro-transition exit model describing the outcome of the transition,

4. The Markov transition probability matrix describing the probability of transitioning to each state at a given time.

The first model splits player movement between offense and defensive movement. Offensive player movement is predicted using the equation:

$$x^l(t+\epsilon) = x^l(t) + \alpha_x^l[x^l(t) - x^l(t-\epsilon)] + \eta_x^l(t)$$

where x is the x coordinate of player l, t is the time since the beginning of the possession, ϵ is the time step, $\alpha_x^l[x^l(t) - x^l(t-\epsilon)]$ is the change in x coordinate over the time step, derived from a Taylor's Series expansion and essentially equating to the gradient at time t, and $\eta_x^l(t)$ represents the contributions of

higher order derivatives (accelerations) which are modelled via a spatial field of each individual player's movements with and without possession of the ball. This is done for the x and y coordinates for each player, meaning the spatial fields for the x and y directions can be combined into a field of vectors describing the average direction and magnitude of acceleration for each player at each location. Defensive movement must take into account the positioning of the attacking players and the position of a defensive player is modelled using a Normal distribution:

$$x^l(t+\epsilon) = N(\mu, (\tau_x^l)^2)$$

where μ is found using work by Franks, Miller, Bornn & Goldsberry (2015), which states that the defensive position, $z(t)$ of a basketball player is centred on a linear combination of the nearest attacker (being guarded, player k), the basket and the ball using: $m_{opt}^k(t) = 0.62z^k(t) + 0.11z_{bask} + 0.27z_{ball}(t)$. This is extended upon slightly such that the mean of the above Normal distribution is instead:

$$\mu = x^l(t) + a_x^l[x^l(t) - x^l(t-\epsilon)] + b_x^l[m_{opt,x}^k(t+\epsilon) - m_{opt,x}^k(t)] + c_x^l[x^l(t) - m_{opt,x}^k(t+\epsilon)]$$

which is then repeated for the y dimension. The equation can be broken down into three terms. $a_x^l[x^l(t) - x^l(t-\epsilon)]$, like the offensive model, is merely the current trajectory of player x. $b_x^l[m_{opt,x}^k(t+\epsilon) - m_{opt,x}^k(t)]$ is similar but describes the trajectory of the model position of the player instead. Finally, $c_x^l[x^l(t) - m_{opt,x}^k(t+\epsilon)]$ acts as a correction so the player movement is biased back towards the modelled position.

For the macro-transition entry model, they model the transition probabilities at time t, $\lambda_j^l(t)$, as a log-linear relationship, where l is the player currently in possession of the ball, and j is the transition type which will be one of six possibilities: Pass to players 1 through 4, shot and turnover. The equation

modelling this is:

$$\log(\lambda_j^l(t)) = [\boldsymbol{W}_j^l(t)]'\boldsymbol{\beta}_j^l + \xi_j^l(z^l(t))$$

where $\boldsymbol{\beta}_j^l$ is a vector of coefficients and $[\boldsymbol{W}_j^l(t)]'$ is a vector of time varying covariates, $z^l(t)$ are the coordinates of player l and ξ_j^l is a mapping of each location on the court to an additive effect on $\lambda_j^l(t)$. We can essentially think of this as a set of spatial fields representing the probability an individual player will choose to attempt each action type at each location on the court. For passing actions, there is another $\xi_j^l(z^l(t))$ term added for the receiving player, i.e. another set of spatial fields representing the probability player l will pass to player k given the latter's position. It is important to note that the coefficients, covariates and mapping terms are individual to each of the 461 players in the dataset as well as the 6 transition types. The output of this model is the probabilities the player in possession will decide to attempt each transition type.

The macro-transition exit model describes the outcome of the transitions modelled above. For turnovers, this is trivial as the possession ends with probability 1, and no positional modelling is required and the possession outcome is 0 points. For passes, they assume that the receiving player will not move a large amount while the ball is en route and the resulting state can be assumed to be identical to the state pre-transition, but with the pass recipient in possession of the ball. Lastly, the shot outcome can be modelled as a success probability as either the shot is successful or it is not; either way the current possession is considered over and a new possession has begun. The authors model the shot success probability using an equation similar to the macro-transition entry model:

$$\text{logit}(p^l(t)) = [\boldsymbol{W}_s^l(t)]'\boldsymbol{\beta}_s^l + \xi_s^l(z^l(t))$$

where the terms have the same interpretation as in the macro-transition entry model, $[\boldsymbol{W}_s^l(t)]'\boldsymbol{\beta}_s^l$ are vectors of coefficients and time-varying covariates and

$\xi_s^l(z^l(t))$ is an additive effect based on the location of the player shooting, all of which will vary for any of the 461 individual players in the NBA at the time.

The final model, the Markov transition probability matrix, would ideally be the observed transition probabilities, but unfortunately the occurrence of uncommon states mean this will degenerate at those points. Instead, the authors use the expectations from their two macro-transition models in place of the observed occurrences, which lead to much better behaviour in rare states.

To measure the accuracy of their model, the authors compared the results of a baseline model, where the transition probabilities are constant for all players and transition types, simplified models with only player coefficients included, player coefficients and spatial effects across players and a model with player coefficients remaining constant across players while spatial effects may change. Comparisons of each of these on a validation dataset found that including varying coefficients and spatial effects across players sometimes lead to poorer results than just using player coefficients alone, while using constant coefficients and varying spatial effects produced by far the best results. They identify that this is due to the previously discussed problem of the model degenerating due to rare states, so using constant coefficients therefore leads to better estimations in these cases as more data is being used to inform the model.

More recently, player tracking has also been used in conjunction with deep learning to produce an estimate of expected possession value. The paper "DeepHoops: Evaluating Micro-Actions in Basketball Using Deep Feature Representations of Spatio-Temporal Data" Sicilia, Pelechrinis, and Goldsberry [53] attempts to estimate the value of off-ball actions performed during NBA possessions, such as an offensive player setting a screen in order for the ball handler to gain space.

This paper uses SportVU data from 750 games in the 2016-17 NBA season, which captures the locations of all players and the ball on the court 25 times per second. They define a possession as a sequence of game states, which take

the form of a 24×1 vector detailing the x and y coordinates of each of the 10 players, the x, y and z location of the ball, and the reading on the shot clock at that moment. They use five outcome labels for their model, each describing one potential terminal action at the end of a play: Shot, Turnover, Shooting foul, Non-Shooting foul and Null. Each state is labelled with the outcome at the end of the possession if the state is within 5 seconds of the terminal action and null if the state is greater than 5 seconds from the terminal action. This is to account for the fact the multiple set-plays (team moves pre-planned by coaching staff) can be run during a single possession with play resetting if no shooting opportunity arises, therefore, it is intuitive to not reward actions that do not directly lead to any terminal action. The authors aim to build a deep learning model that will predict the probabilities of a possession ending in each terminal action, given some input sequence of game states, rather than predict the expected value of the possession directly. The proposed architecture of their model is a stacked LSTM model, where LSTM cells take outputs from the previous cell along with the next input in the sequence and pass outputs along the layer to the next cell, as is typical for LSTM models. In a stacked LSTM structure the outputs from a cell, the external state $h^{(t)}$, is also passed down a layer. We can conceptualise this as a grid of LSTM cells, where the output of cell, $C_{i,j}$, is passed to cells $C_{i+1,j}$ and $C_{i,j+1}$, where i and j are the row and column of the LSTM grid respectively and $i+1$ and $j+1$ are within the grid limits. Therefore, any cell in layers beyond the first will take the external state output from the cell above as the typical LSTM cell input, $x^{(t)}$. The stacked LSTM structure is illustrated in Figure 2.9.

For this model, the authors used a 32 cell wide, 3 layer deep grid of LSTM cells. The model also includes a player embedding layer, where a one-hot encoding of the players on court are mapped to an 8×1 vector representation which is learned during the training process. The final external state output from the LSTM grid is concatenated with the player embedding representations and passed through a final dense layer with a softmax output function, to

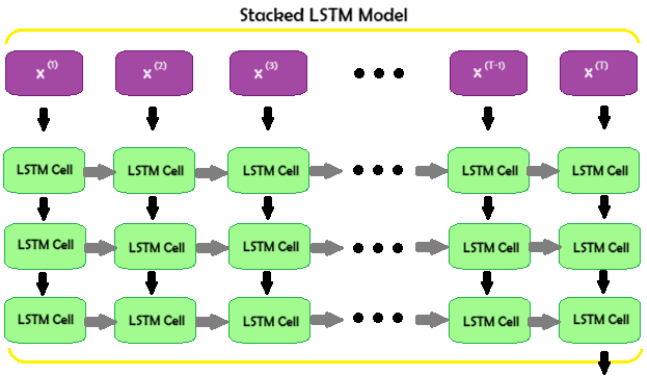


Figure 2.9: Stacked LSTM Model Structure

produce an output of a set of probabilities for each terminal action. The model is then trained with a negative log-likelihood loss function. One final point to note, is that the number of "null" labels in the dataset far outweighs that of any other label. Unbalanced datasets are known to cause issues when generating probabilities, therefore, the authors implemented a down-sampling scheme in order to ensure each terminal action class is equally represented in the training dataset.

To measure their models, the authors use a Brier score, which is equivalent to the Mean Squared Error of their model's predictions. They compare the model's Brier score, BS, to a "Climatology Model" Brier score, BS_{ref} whereby the predictions for each class in the testing set are constant and equal to the observed probabilities, i.e. if 70% of the testing set is labelled "null" then the outputs will all contain a prediction of 0.7 for the null class. They then compute a Brier skill score, BSS, which is given as:

$$BSS = 1 - \frac{BS}{BS_{ref}}$$

This essentially acts as a precise measure of how much better their model's predictions are compared to proportional random guessing. They found that their model consistently outperformed the climatology model at various levels of down-sampling. Finally, the proposed method of calculating the expected value from their model predictions involves computing the expected points for

each terminal action. For turnovers, this is 0 as there is no opportunity to score after turning the ball over, while for shots the expected value is equal to 1.25, which is the average points scored after a shot. Null has an expected points scored of 1.02, which is the average number of points scored per possession in the NBA, while shooting and non-shooting fouls yield expected values of around 1.7 and 1.1 respectively. These values are then multiplied by their respective probabilities given by the model output and summed to give the final Expected Possession Value of a game state.

We have seen that across multiple competitive sports and environments, there is a similar trend of increasing granularity of collected data, with player tracking being introduced and utilised across professional sports globally. We have also seen the newer trend of analysing this data via deep learning methods, especially among very popular and very "fluid" sports such as Basketball and football, as deep learning opens up the possibility of quantifying aspects of the game previously thought of as "intangible". We must remind ourselves once again that due to the nature of sports, the methodologies and techniques professional teams are willing to reveal are likely years behind the current state-of-the-art behind closed doors.

2.2.5 Deep Learning

> Artificial Intelligence, deep learning, machine learning - whatever you're doing if you don't understand it - learn it. Because otherwise you're going to be a dinosaur within 3 years.
>
> *Mark Cuban*

We must first begin by briefly discussing the evolution of the Deep Learning techniques. Machine Learning and Deep Learning algorithms are an area of

intense research and have a rich background of literature, such as [42, 76, 9, 22], and indeed the theory and application of these algorithms can, and do, fill multiple books on the topic. We will therefore only cover these algorithms in enough detail to understand their application in this thesis.

The basis of most statistical analysis is building a model that explains as much of the variation observed within the data as possible. If we look at a very simple example such as linear regression:

$$Y = \beta X + C + \epsilon$$

The regression equation coefficients have to have been inferred through optimising some function of the data, in this case the Likelihood Function. The likelihood function is the joint probability density function where the observed data, x is treated as fixed while the parameters, θ, are treated as variable. Therefore, if we express a typical probability density function as $p(x|\theta)$, we can express the likelihood function as $L(\theta|x)$. Finding the values of θ that maximise this function yields the regression equation parameters as these maximise the likelihood of observing our collected data. We can introduce non-linearity through polynomials, link functions (GLMs) or interaction terms, although even these are inadequate for very complex problems.

The downside of regression methods in our context, and many other problems, is that the large number of related data features leads to high complexity when looking to produce a prediction for points scored due to the non-linear effects of player positioning and the inter-connected nature of their positions.

If we extend this concept of finding the optimal point according to some function, we can understand Deep Learning as a method of optimising a large number of parameters in a non-linear model. Deep Learning algorithms fall under the larger Neural Network (NN) umbrella, and at a very basic level are made up of layers of "nodes", each consisting of a multiplicative "weight" and an additive "bias" and a non-linear "activation function", seen in Figure 2.10. A node input will be multiplied by the weight, the bias added and

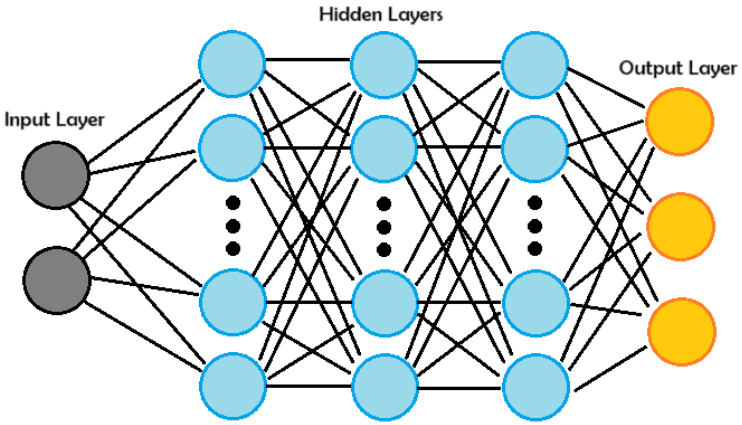


Figure 2.10: Basic Neural Network Diagram

then passed through the differentiable activation function, often as simple as a ReLU function or sigmoid [24, 43].

$$\text{ReLU}(x) = \begin{cases} x & \text{for } x \geq 0 \\ 0 & \text{for } x < 0 \end{cases}$$

$$\text{sigmoid}(x) = \frac{1}{1 + \exp\{-x\}}$$

A layer is then made up of some number of nodes, each with an individual weight and bias, although the activation function usually remains the same for every node in the layer. The output of each of these nodes is then combined in some way to form the overall network output. We can improve performance of a network by "stacking" layers, whereby the output from one layer is then used as the input for the subsequent layer, and so on. "Deep" learning then refers to stacking a large number of these layers, such that the network architecture looks "deep" rather than "wide" [21]. The final, or "output", layer will then transform the input according to problem specific information.

Here we measure the accuracy of our model outputs through some function, which we call the loss or cost function. In essence, these functions will all take the model output and find some difference between the outputs and the observed data, whether these are the squared difference between each node

output and the observed data, or something more complex. One important aspect of the loss function, like activation functions, is that it must be differentiable with respect to the output nodes (or we at least need to be able to find subgradients at non-differentiable points) [76]. For example we can consider the Mean Squared Error:

$$L(x, y) = MSE(x, y) = \frac{\sum_{i=0}^{N}(x_i - y_i)^2}{N}$$

Where x and y are vectors representing the model outputs and the observed outcomes respectively. This then has a differential with respect to x of:

$$\frac{\delta L(x, y)}{\delta x} = \frac{2}{N}\sum_{i=0}^{N}(x_i - y_i)$$

The "learning" of the algorithm is done using a process called back-propagation, where the gradient of the loss function is propagated back through the network layers and used to update the node weights and biases. If we take output layer x, where $x_k = \sigma(z_k^{(l)})$, and σ is the activation function applied to node z_k, where k is the index of the node and l is the index of the layer, which will be important to consider later. We then see that z_k^l will be the linear combination of the previous layer's nodes multiplied by their weights and the node bias. From this we can obtain:

$$\frac{\delta x_k}{\delta z_k^{(l)}}$$

As we should know the derivative of activation function σ, and:

$$\frac{\delta z_k^{(l)}}{\delta w_{kj}^{(l)}}, \frac{\delta z_k^{(l)}}{\delta b_k^{(l)}}, \frac{\delta z_k^{(l)}}{\delta a_k^{(l-1)}}$$

where w, b and a denote the weights, bias and the output from the previous layer respectively, and j represents the index of the node in the previous layer. As z_k is just given by:

$$z_k^{(l)} = \sum_{i=0}^{n} w_{kj}^{(l)} a_j^{(l-1)} + b_k^{(l)}$$

the gradient of $z_k^{(l)}$ with respect to any of these elements is trivial to find. The gradient of the loss taken with respect to a single output node x_j, $\frac{\delta L}{\delta x_j}$, can

be combined with the above gradients via the chain rule to find the gradient of the loss function with respect to any of these elements, these can then be used to update the current value of the weight or bias, or in the case of $\frac{\delta L}{\delta a_j^{(l-1)}}$ used as the "loss" for the previous layer allowing the algorithm to propagate the updates to the weights and biases of nodes in prior layers using the same process described here, hence the algorithm name. Here we can see the importance of easily differentiable activation functions such as ReLU, which give gradient values of 1 or 0.

This is the essence of a Neural Network, and while many variations exist, the process of training the model will not differ significantly from that described above. Two such variations which will be important for the understanding of this thesis are the Convolutional Neural Network, CNN, and the Long Short-Term Memory, $LSTM$, unit, as these have the most obvious applications to our problem.

A CNN is based on convolutional layers, where instead of all nodes in one layer being fully connected to all nodes in the following layer, the nodes are arranged in a grid structure, similarly to an image with pixels. This allows images to be read in directly with minimal changes to the formatting. A convolution essentially passes a "filter" of $n \times n$ weights over the node grid, $n \times n$ nodes at a time, and sums the results before using this as the value for a node in the following layer, after adding the bias term and passing the result through an activation function [14]. The filter is then shifted by some number of nodes, "step" size, and repeated until the entire grid of nodes has been convoluted and a new grid has been constructed. This technique obviously lends itself well to modelling images and problems with some spatial aspect. A visualisation of the convolution process is given in Figure 2.11.

Where CNNs are useful for the modelling of images, $LSTM$s are designed with sequences in mind. $LSTM$s are a type of Recurrent Neural Network, meaning that the outputs of nodes are used to influence the subsequent input of the same nodes, seen in Figure 2.12 [76].

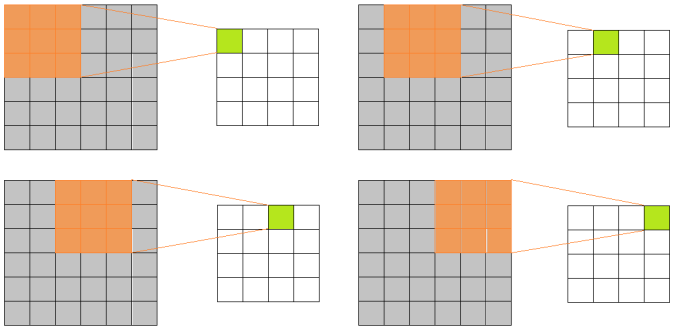


Figure 2.11: Convolution Example

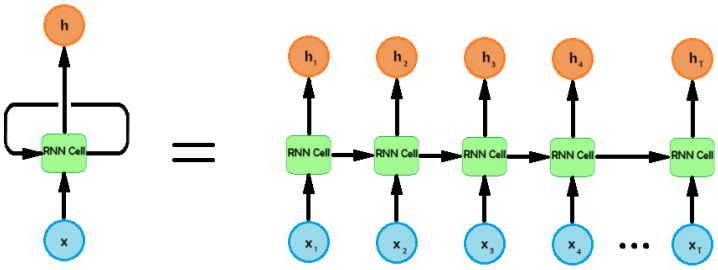


Figure 2.12: Basic RNN Model Structure

The $LSTM$ cell is most easily understood through a diagram of the cell architecture which can be seen in Figure 2.13.

The cell takes three inputs, the input vector, x at time t, the memory element from the previous cell, C_{t-1}, and the output, or hidden state, of the previous cell, h_{t-1}. The outputs C_t and h_t are then passed to the subsequent cell, or if t is the final time step in the sequence, h_t is used as the model output.

With the background of the methods used in this thesis now covered, we will proceed to analyse the current methods used in sports analytics. More details of Machine learning techniques will be covered as and when required.

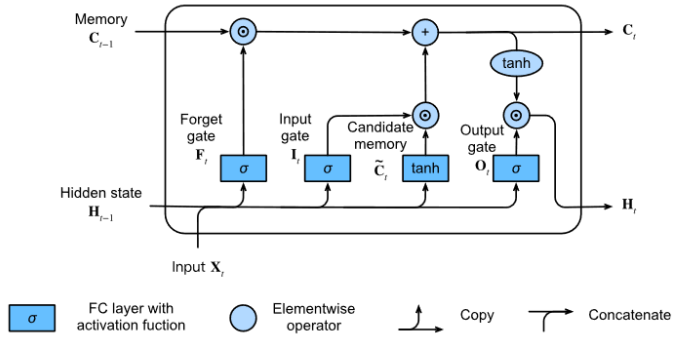


Figure 2.13: LSTM Cell Structure Zhang et al. [76]

Chapter 3

Methodology

As discussed earlier, the aim of this thesis is to propose a model that is capable of valuing basketball possessions primarily using the positioning of the players on court. We have obtained a dataset describing the coordinates of players on court for roughly half of the 2015/2016 NBA season. In this chapter we describe the data and the processing and cleaning required to transform the data into a usable format for our model. Following the discussing of our data, the base model structure and tested variations are described, along with the hyper-parameters used in the model.

3.1 Data

> It's one thing to have a ton of data. It is entirely another thing to know what to do with it.
>
> *Patrick Minton*

With the goal of the study in mind and an understanding of the typical processes followed by analysts in the field, we can begin developing our model. The first step is to inspect the data available, the format and any potential problems in our dataset. With the general structure our model will take and the formatting of the data required to facilitate this in mind, we then outline

the processing of the data to remove errors and restructure the data into a more convenient format.

3.1.1 Description

Our primary data source is a publicly available set of NBA game logs from the 2015/16 season collected by SportVU. The dataset contains roughly 630 NBA games played between the 27^{th} of October 2015 and the 23^{rd} of January 2016 [26]. This is the equivalent of about half of a full NBA season, which is comprised of $1,230$ games. Each game log contains the game ID that NBA uses to identify each game played, the date, and a series of 5 to 15 second "events". Each event contains an event ID that describes the type of event recorded, two dictionaries storing the data outlining the home and away team information and a list of "moments" making up the event. The "moments" are a list of snapshots of information at a given time, collected 25 times per second. A moment consists of the game quarter (1, 2, 3 or 4), the UNIX timestamp (universal running time in milliseconds), the current reading on the game clock and shot clock, a seemingly blank row containing no information, and finally, a 5 by 11 matrix of the team IDs, player IDs, x, y and z coordinates for the players and the ball in feet. The team and player ID columns for the ball are just listed as -1, while the z coordinates are not collected for the players, see Figure 3.1.

This nested structure makes working with the relevant data more laborious, and the constant re-recording of player and team information inflates the storage size of the files considerably. On closer inspection, we can also see that events frequently overlap with the events before and after, meaning some moments are recorded multiple times under different event labels.

To supplement our primary dataset, we employ the use of NBA play-by-play logs for the 2015/16 season, detailing every event and the time it occurred during a game for the entire season [45]. Each entry states the event type, event outcome, the game ID, home and away event descriptions, clock time of the

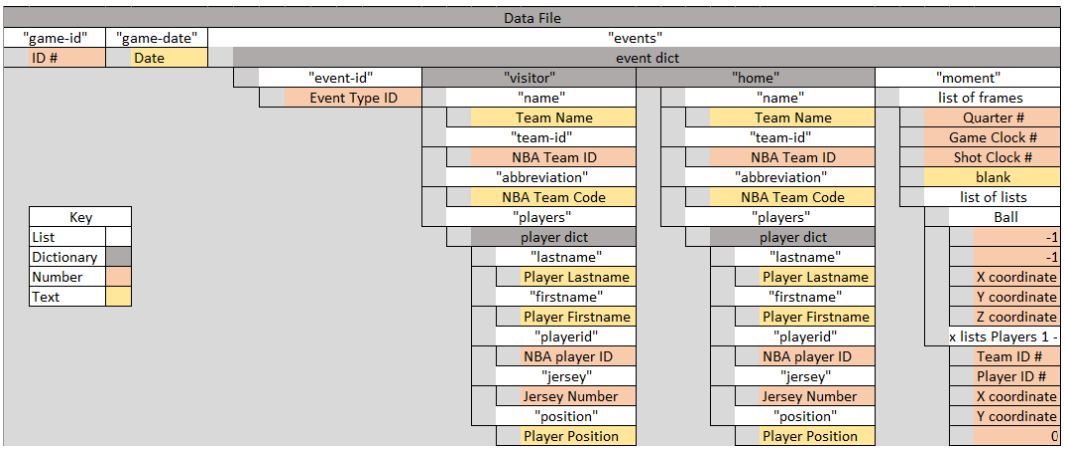


Figure 3.1: SportVU Data Structure

event, the game time of the event, the period/quarter of the event, player information for those involved in the event and the score and score margin at the time of the event [72].

One issue to note at this point is the temporal resolution of the play-by-play logs are to the nearest second, meaning there is the potential that while merging the two datasets, the moments may be mislabelled by a maximum of half a second, or roughly 12 moments, at the time of an event.

3.1.2 Data Processing

To address the issues discussed, we developed a method of processing the entire dataset to extract the required information while reconciling differences in moment labels.

Ultimately, the information we require from our data is the player IDs of the players on court, their x and y coordinates and the equivalent information for the ball. We also compute the velocity of the players and ball as this captures the orientation of a player and their previous locations which has been shown to be beneficial in literature. Velocities are easily calculated by finding the difference in coordinates between consecutive moments, and are therefore the distance travelled in 1/25 of a second in both dimensions. Lastly, we collect the information regarding the game state, i.e. the reading on the game and

shot clocks, as well as the current score margin and most recent points scored as an indicator of game momentum.

For clarity, the term "frame" in this study will describe the x and y co-ordinates and x and y speeds of the players on court at a given time, while *game state* will describe the additional information collected for each moment, discussed above.

Our labels are both the event type and the next point change within the event window, with values being integers within [-3,3], the logic behind this is a poor possession ending with the opposition counterattacking should be identified as worse than a regular possession ending with no points scored.

For the event window we have followed work done in "DeepHoops" [53] which uses a 5 second window for event labels, where all moments 5 seconds prior to the event up to the time of the event itself are labelled with the event ID. The idea behind this being that a team will run plays lasting not much longer than a few seconds before resetting positions if no shot is possible. Therefore, it makes little sense intuitively to label a moment more than 5 seconds prior to an event when it likely has very little correlation to the event in question. Our event labels are also inspired by this paper. We label moments with one of: shot, turnover, foul or null.

Our final stipulation for our data is that the attacking team should be playing in the same direction for all of our moments. There are a limited number of games available for training the model, so the direction of play has been removed from the data allowing our model to learn key features more efficiently.

To reconcile the event labels, the player tracking data has been aligned with NBA play-by-play logs temporally. To standardise the direction of play, there must be some way of indicating the current direction of play and the team in possession. Unfortunately there is no rule in the NBA dictating that the home or away team needs to be playing in a certain direction in the first quarter, however there are rules dictating that teams must switch at half-time, with

any overtime counting as an extension of the second half [52]. Therefore, if we know which team has possession and the direction of play in the first quarter, we can deduce the current direction of play. We keep track of each of these using binary indicators taking values 0 or 1. The first is easily tracked using the play-by-play logs as it records the participants of each event, so the home team committing an action would yield an indicator of 1 for every moment between the end of the previous action and the current action, and 0 for the away team. The second indicator is found by scanning the play-by-play logs for the first shot in the first quarter and then finding the location of the ball from the SportVU game log. If the home team is shooting in the left half of the court, or the away team is shooting on the right, then the indicator takes a value of 1, otherwise it takes 0, this then alternates at half-time. From this we determine whether to flip a frame using an XOR function;

$$I_f = \begin{cases} 0 & \text{if: } I_p = I_d \\ 1 & \text{otherwise} \end{cases}$$

Where I_p and I_d are the indicators for home possession and play direction respectively, and I_f is the indicator to flip a frame.

Then the new x and y coordinates can be computed using:

$$x' = I_f(94 - x) + (1 - I_f)x$$

$$y' = I_f(50 - y) + (1 - I_f)y$$

where the maximum court dimensions are 94×50, however this also conveniently correctly flips players who are located outside of these court dimensions. We then step through the events in the play-by-play logs and the SportVU logs simultaneously. We start with the first event, record the team in possession to I_p, then step through the SportVU logs, flip the frame if required, compute the player and ball velocities, then record the coordinates, velocities and IDs of the players and the ball, the game state is then recorded and the moments labelled. By default, moments are labelled as Null events, unless the game clock

in the SportVU logs is within 5 seconds of the play-by-play logs, in which case the moment is recorded as the relevant event type along with any point changes. Once the SportVU game clock passes the play-by-play clock, we take another step through the play-by-play logs. In the case of a turnover event, we also check whether the opposing team scores within 5 seconds immediately following the turnover in order to correctly label a potential negative point change.

With this process completed for all games in our dataset, we were able to proceed to the modelling stage. In terms of the identified issues prior to data processing, those being storage size, overlapping events and the awkward structure of the data, these problems have been significantly alleviated. Our storage size of the processed data is roughly half that of the unprocessed data (∼100MB per game to around 50-60MB per game), there are also no redundant recordings or double-ups of moments, while information is no longer recorded in a nested structure and far simpler to use while modelling.

3.2 Modelling

> Stats are killing the game of basketball. A lot of things happen that you can't measure in stats
>
> *Mark Gasol*

As discussed earlier, we have approached this task as a supervised learning problem. Unsupervised learning methods do not suit our problem as we aim to learn a function estimating class probabilities and expected value given some inputs, rather than trying to find hidden connections and similarities between different data points where the output is unclear. Reinforcement learning is potentially more applicable, but is infeasible for a study of this size. This would require an environment of adequate accuracy for agents to

learn in order for any outputs to be considered useful. This task could be likened to developing a sports video game, usually taking in the order of years to complete, and with constant struggles and complaints concerning realism. Aside from the practicality aspect, developing an agent which has learned to play basketball perfectly does not solve our problem. We are not able to control players on the court, so a reinforcement learning algorithm would essentially provide the optimal actions after the fact, rather than a measure of how valuable a possession is. For these reasons, it is reasonable to model this using a supervised learning approach that models the relationship between the value of a possession and player coordinates.

Sicilia, Pelechrinis, and Goldsberry [53] have shown that good estimates of the value of a possession can be obtained from the probabilities of terminal actions, while intuition would lead us to building a model with a direct value estimate as an output. There would, of course, be benefits to coaches and analysts to having the terminal action probabilities on top of the value estimate, and Google Deepmind have shown a model is able to produce two outputs successfully with AlphaGo; therefore, we model both the terminal action probabilities and the expected value of a possession directly.

Silver et al. [55] and Fernández, Bornn, and Cervone [17] have shown Convolutional Neural Networks can be used to model the effects of player/piece positioning successfully. Intuitively, CNNs should be well suited to extracting key information about the positions of players not just with respect to the court, but also their positions relative to other players, a single convolutional layer should in theory be capable of identifying off-ball screens for example. CNNs are also fairly versatile in their implementation, allowing for other types of modules to be added without sacrificing the structure of the model significantly. Therefore it is reasonable to use a CNN as a base of the model, with the potential to add other neural network layer types to supplement the model.

Data is selected for training via random selection. The training/validation data split is done at the initialisation of each new model run, with entire games

being divided into the training and validation set to ensure that all validation samples are independent from those in the training set. We also withhold 50 games prior to any split as the test set so that each model could be assessed on the same data while remaining independent of any training or validation data.

Once the training/validation split is complete, the data is then sampled from each game at an average of one moment for every five seconds of in-game time. This is done for both the training and validation sets, and performed every epoch to minimise the effects of selecting two moments within close proximity of each other. While sampling, the class label of each selected moment is recorded for the computation of weights used in the loss function, discussed later.

The sampled data is then passed into standard implementations of the PyTorch Dataset and Dataloader objects, with a batch size of 128 samples. The data transformation applied at this stage converts the set of coordinates into multiple input channels of size 50×94, the dimensions of the court in feet. The base model uses 9 channel inputs, the input coordinates are rounded down to give integer values which are used to reference the correct pixel in the transformed image. Values outside of the court dimensions (i.e. when a player momentarily leaves the court) are either rounded up to zero or down to the maximum value in the respective dimension, this essentially leads to out-of-bounds players being represented as close to the sidelines as possible. We also separate the game state from the frame at this stage and reserve the game state information for the later stages of the model. The 9 input channels used are given in Table 3.1,

where locations are represented as a 1 if the ball or a player is present at that pixel and zero otherwise, while speed channels are similar but give the speed in the respective dimension rather than 1.

As mentioned, we employ the use of a Convolutional Neural Network to estimate the function describing the relationship between the ball and player

Channel #	Team	Channel Type
Channel 1	Ball	Location
Channel 2	Ball	X Speed
Channel 3	Ball	Y Speed
Channel 4	Offensive team	Location
Channel 5	Offensive Team	X Speed
Channel 6	Offensive Team	Y Speed
Channel 7	Defensive Team	Location
Channel 8	Defensive Team	X Speed
Channel 9	Defensive Team	Y Speed

Table 3.1: Input Channel Descriptions

locations, and the value of the possession. The base model consists of three convolutional layers, the first consisting of 18 7×7 filters, the second consisting of 32 7×7 filters, each followed by a 2×2 Max Pooling layer. The final convolutional layer being 64 3×3 filters, all convolutional layers use a stride of one and a padding of 1 such that the 3×3 filters return a tensor with the same X and Y dimensions as the input. The result of the convolutional layers is then flattened and passed through two fully connected layers, with 1024 and 512 nodes respectively. This is then recombined with the game state information for the given moment, and passed through a final fully connected layer with five output nodes. The first four of the output nodes are left as un-normalised logits to be passed into the loss function and represent the values for the possession classification task, while the final node is passed through a hyperbolic tangent function and further transformed to fit within [-3, 3]. This represents the model's estimated value for the moment analysed (figure 3.2). We also include Dropout layers, where nodes and their connections are randomly dropped during training, between each of the convolutional and dense layers to prevent overfitting [65]. This is intended to prevent the network

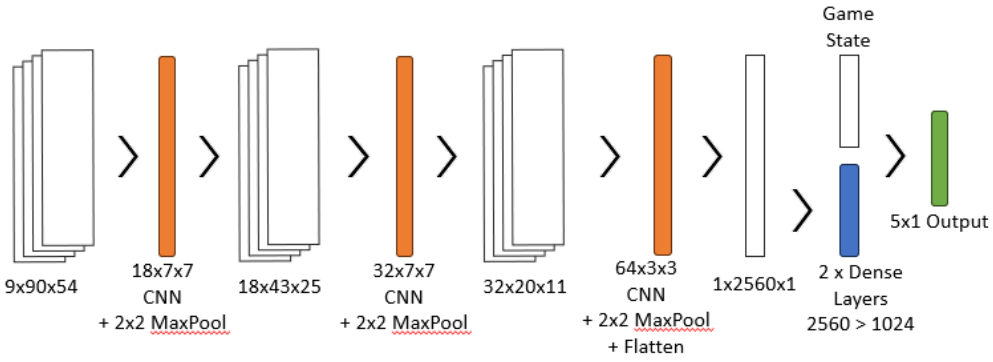


Figure 3.2: Base Model Network Architecture

from becoming over-reliant on a small number of nodes for prediction. For this thesis we use a dropout rate of 0.2 unless otherwise stated.

As our model outputs two types of prediction, a classification output and a regression output, we must consider the loss function carefully. Ideally we wish to weight the loss function for the class predictions to counteract the effects of our imbalanced dataset, so to accomplish this we will use a multi-objective loss function. The loss function used in this study is a combination of the Cross Entropy Loss of the classification predictions and the Mean Squared Error of the possession value. These are mixed with a simple mixing variable, $\alpha \in (0, 1)$. The loss function is given by:

$$Loss = \alpha CEL(x_{class}, y_{class}, w) + (1 - \alpha)MSE(x_{point}, y_{point})$$

Where x_{class} and x_{point} are the model outputs for the class and expected points predictions respectively, while y_{class} and y_{point} are the class and point labels for the moment analysed, w is a tensor of the class weights obtained using the number of occurrences for each class collected during the sampling stage for each epoch. The weights are then calculated using the equation

$$w_c = \frac{\sum_{c=1}^{C} n_c}{C \times n_c}$$

where w_c is the weight of the c^{th} class, C is the total number of classes, in this case $C = 4$ and n_c is the number of occurrences of the c^{th} class. This yields weights inversely proportional to the prominence of the class in the training

set, such that occurrences of very rare events such as fouls are weighted higher than more common occurrences of null events.

Our Cross Entropy Loss function for an individual output then takes the form:

$$CEL(x, y, w) = -\sum_{c=1}^{C} w_c log \frac{exp(x_c)}{\sum_{i=1}^{C} exp(x_i)} y_c$$

And the mean of these values for each individual moment over the entire batch is then taken as the loss function output.

Lastly, as we have only one output node for the estimated value of a possession, our Mean Squared Error for an individual prediction is:

$$MSE(x, y) = (x - y)^2$$

And similarly to the class predictions, the mean value across the batch is taken as the loss function value.

The above model was tested with ReLU and leaky ReLU activation functions [43] following all layers. Dropout was also tested between the fully connected layers, while the learning rate was optimal between 0.001 and 0.0005 while α did not significantly affect training or overall performance. Finally, Stochastic Gradient Descent optimisation was also tested against ADAM [30]. The tests concluded that the leaky ReLU function, dropout and ADAM optimiser performed significantly better in both model fit and training time than the alternatives. Furthermore, a learning rate of 0.005 was found to produce the most stable results. Therefore, only the results from models using these settings are presented in this thesis.

Each of the following models are additional to the base model, and were not implemented together unless stated.

3.2.1 Player Embedding

Similarly to Sicilia, Pelechrinis, and Goldsberry [53], player embedding was tested. All players in the dataset were encoded using one-hot encoding, a

vector of zeros with a one at the player-unique index. This vector is then passed through an embedding layer (equivalent to a fully connected layer in this case) with a 5×1 output representing the player. The idea behind this is that measuring the effects of every player at any moment is difficult given not every player finds themselves in every situation often enough to build an accurate idea around how they influence the possession. Player embedding allows us to group similar players in N-dimensional space, where $N = 5$ in our case, and each dimension or element of the vector hopefully captures some aspect of a player's play-style and ability. The embedding layer essentially acts as a mapping function between a player index or ID and their unique vector, which can then be learned by our network.

The player embedding is incorporated into our existing base model prior to the first convolutional layer in order to also capture a player's spatial influence. We therefore have a $50 \times 94 \times 5$ tensor after the player embedding has been merged with the positional information, this can be visualised as a three-dimensional "box" of zeros with columns of player representations at their X/Y coordinates. This is passed through a $5 \times 7 \times 7$ 3D convolutional layer, which is aimed at capturing the interactions between different aspects of nearby player's play-styles. The output of this 3D layer is a $1 \times 50 \times 94$ tensor which is then recombined with the 7 remaining ball location and velocity channels described in the base model, and passed through the rest of the existing model without any other alterations.

3.2.2 Convolutional Block

Schrittwieser et al. [50] employed the use of multiple "blocks" of 3×3 convolutional layers with residual elements to ensure the original input is still prominent in the latter blocks. These are intended to extract information while maintaining the size of the image, allowing for incredibly deep networks able to learn very nuanced patterns. We implement these convolutional blocks as a supplement to our model.

The convolutional block consists of two 3×3 convolutional layers each with 64 input and output channels followed by a batch normalisation layer before adding the input to the block back to the result of the block. Additionally, only the first of the convolutional layers uses an activation function, which is a leaky ReLU function in this case. This block is introduced at the beginning of the network, along with an additional 3×3 convolutional layer primarily used to resize the tensors to 64 input channels such that the convolutional block can be defined once and initialised multiple times within the model if necessary. The aim of this is to increase the capacity of the network in a modular way, where the capacity may be increased gradually until a satisfactory point is reached. The base model is minimally altered to accommodate the additional module, where the input and output channels of the base model layers are increased 128, and the first dense layer resized to account for the increased number of channels.

3.2.3 Inverse CNN

Similarly to our reasoning for the convolutional blocks, we aimed to increase the capacity of our model by introducing an inverse convolutional layer. Fernández, Bornn, and Cervone [17] shows that up-sampling can be a useful tool within convolutional neural networks in a sporting context, utilising a number of them throughout their models. As the name suggests, an inverse convolutional layer is an up-sampling method that operates similarly to a typical convolutional layer in the opposite direction. More specifically, if we consider a 3×3 convolutional filter, where the nine pixels captured by the filter are multiplied by their weights and summed before adding a bias, an inverse convolutional layer with the same sized filter will multiply the value of a single pixel by the nine filter weights. These results are added to the results of convolutions from neighbouring pixels where these convolutions overlap due to step size and stride length, see Figure 3.3.

The base model is altered by inserting a 3×3 inverse convolutional layer at

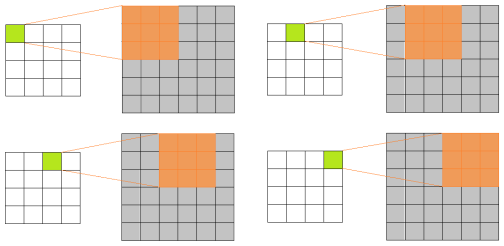


Figure 3.3: Inverse Convolution Example

the beginning of the model, such that the inverse layer takes the model input tensor directly, and is followed by an additional 5×5 convolutional layer. It is evident that introducing a single inverse convolutional layer in fact allows the opportunity to introduce two layers capable of feature extraction.

3.2.4 Final Model

Variations on the base model were also tested, such as using differing convolutional filter sizes as well as differing numbers of convolutional and fully connected layers. Summarised results for these tests are given in the next chapter, but the best performing of these will be inspected further. The final model adds an additional 3×3 convolutional layer and an additional fully connected layer. The input tensor is therefore passed through two 7×7 convolutional layers, two 3×3 convolutional layers, again, all with strides and paddings of 1, and finally three fully connected layers with 1024, 64 and 5 nodes in each layer respectively, where the game state is introduced in the final layer.

3.2.5 Data Augmentations

Generally, a model can be greatly improved by significantly increasing the amount of available data. In this context this would be the form of additional seasons of data and/or data from other leagues to account for any stylistic trends the current teams in the dataset may have followed for the single season

we have available. Teams are likely to be running similar game-plans and plays throughout the season due to the players and coaches remaining mostly static throughout the season, and the NBA is likely to have its own distinct style within the greater context of world basketball. Unfortunately we are unable to obtain the desired data due to being un-available or non-existent, so to synthesise additional data we opt to flip our existing data on the y axis, essentially doubling the size of our dataset. As we still wish for the attacking team to be playing left we will not flip on the x axis. We must acknowledge that this is far from ideal, as the aforementioned stylistic tendencies of teams and the league overall will still be prevalent meaning the synthesised data may not be significantly different after transformation. We also assume by doing this that teams do not already run mirrored set plays to a significant degree.

We flip the frames within the transformation function in the dataloader object, upon being passed a moment to process, we will randomly sample from a Bernoulli distribution, where our result is either 0 or 1. This is used to determine whether to flip a frame or not, where our transformed y coordinate can be given by:

$$y' = f(50 - y) + (1 - f)y$$

where f is our randomly sampled Bernoulli variable, y is our original y coordinate and y' is our transformed y coordinate. We must also flip our y velocities, which can easily be done using:

$$v'_y = -(2f - 1) \times v_y$$

where v'_y is our transformed x velocity and v_y is our original velocity. The term, $-(2f - 1)$, evaluates to -1 if our f variable is equal to 1 thereby flipping the y velocity appropriately, the term also evaluates to 1 if our f variable is equal to 0, leaving the y velocity unchanged.

A number of new input features are also introduced, namely the angle and distance to the hoop. Similar features are used in Fernández, Bornn, and Cer-

vone [17] and Link, Lang, and Seidenschwarz [33], and show potential benefits in this context. These are computed using the raw x and y coordinates, not the integer pixel coordinates, and are simply calculated using basic trigonometry. These are passed into the model as six new channels, two for the ball and each of the teams. In each of these channels the pixels at locations of the player or ball take the value of the feature, similarly to the velocity channels.

While not implemented in this study, other options for additional model features could include: distance to nearest defender/attacker, angle between movement direction and the hoop, locations of defensive blocks, density of defenders ahead of players etc. along with a number of other options presented in papers previously discussed.

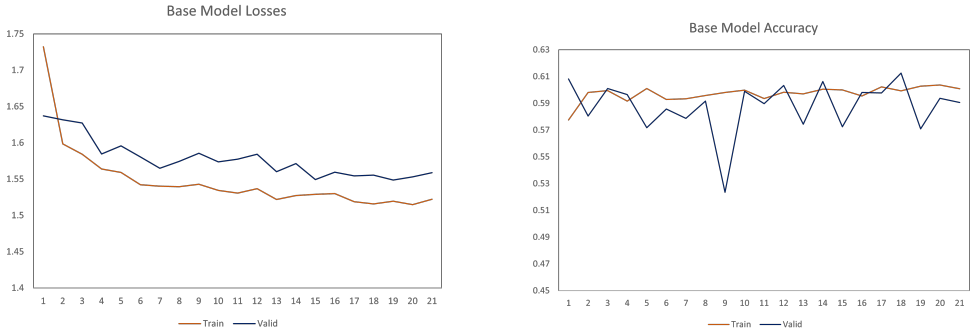
Chapter 4

Results

> Every action on a basketball court is influenced by nine other players, not to mention a coach. For this reason, there is no 'holy grail' in basketball equivalent to baseball's on-base percentage.
>
> *Chris Ballard*

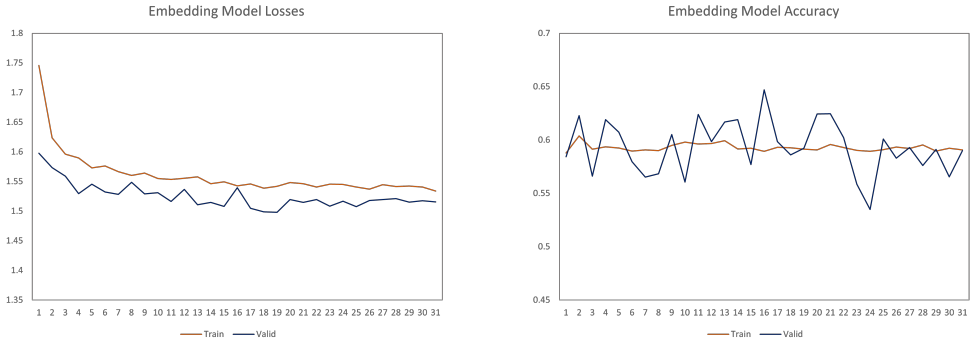
The average loss and accuracy of each epoch for the training and validation datasets are shown in Figures 4.1, 4.2, 4.3, 4.4, 4.5 and 4.6. The accuracy is calculated using only the class prediction portion of the model output, where the highest output probability is taken as the prediction and compared to the target outcome, this is then averaged over the entire epoch.

Generally the models reach roughly the same threshold for both the loss and accuracy, regardless of the complexity of the model. This indicates that we are reaching the upper end of the capability of models and datasets of this size using our method of problem encoding. It is still likely based on the improvement of deep learning models in other areas such as Large Language Models, that significantly increasing the amount of data available and the network size would lead to further improvements. In some cases the training loss is higher than the validation loss, this is primarily due to the combined



(a) Base Model Loss (b) Base Model Accuracy

Figure 4.1: Base Model Training Results

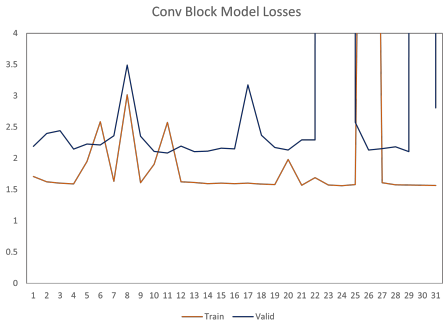


(a) Embedding Model Loss (b) Embedding Model Accuracy

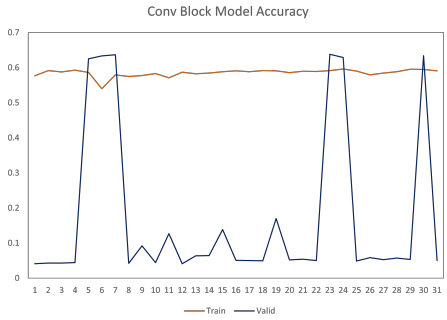
Figure 4.2: Embeddeing Model Training Results

effects of the dropout layers, the sampling methods for the validation set, and the weighting of the Cross Entropy Loss component. Overall, while this is not optimal behaviour, it does not pose a significant problem as long as the model predictions are adequate. It is also evident that the trend is not visible in the model accuracy, implying it is likely caused by some quirks of the loss calculation rather than anything concerning the model predictions.

The figures also show that while the training accuracy is generally stable for each model, the validation accuracy tends to be significantly more noisy, although this is attributable to the sampling method used to construct the validation set. Notably, our results for the convolutional block model are incredibly poor. The loss fluctuates wildly, especially on the validation set, while the accuracy on the training and validation sets indicate the model is

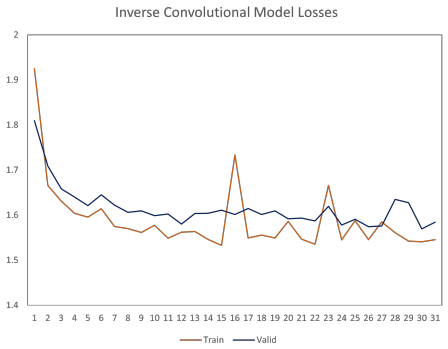


(a) Conv. Block Model Loss

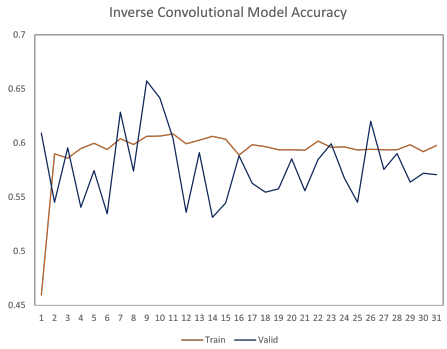


(b) Conv. Block Model Accuracy

Figure 4.3: Conv. Block Model Training Results



(a) Inverse CNN Model Loss

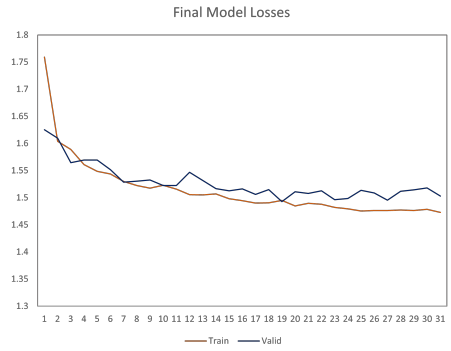


(b) Inverse CNN Model Accuracy

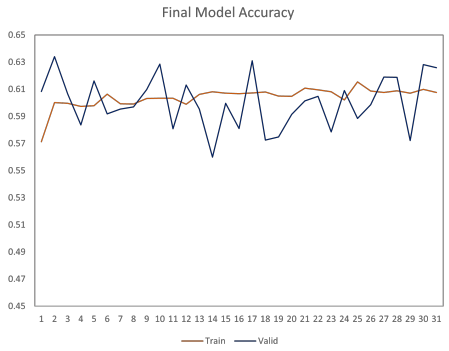
Figure 4.4: Inv. CNN Model Training Results

likely overfitting. We can take the overfit loss and accuracy as indications of the maximum/minimum values obtainable with our available features. It is also unlikely it is possible to fit a more complex model to this problem without additional data.

Naturally, using accuracy in this way does not give a complete overview of the prediction quality due to the unbalanced nature of the data. One could gain 90% accuracy by predicting "null" in every case as this class makes up 90% of the target outcomes. Therefore we must also find a way to measure the accuracy while punishing the incorrect predictions of rarer classes. A straightforward and well-known metric meeting these criteria is the F1 score, which, in a two class problem, takes the harmonic mean of precision and recall. The F1 score is calculated using the following equations:

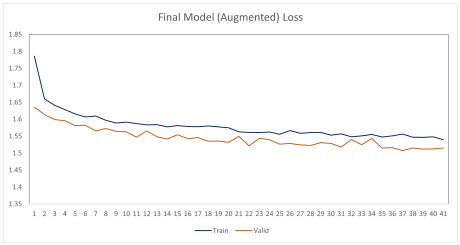


(a) Final Model Loss

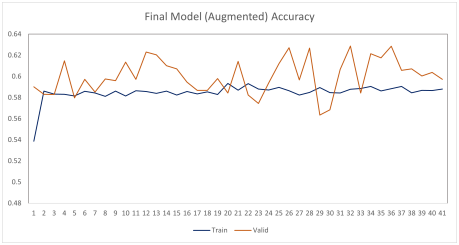


(b) Final Model Accuracy

Figure 4.5: Final Model Training Results



(a) Final Model (Aug) Loss



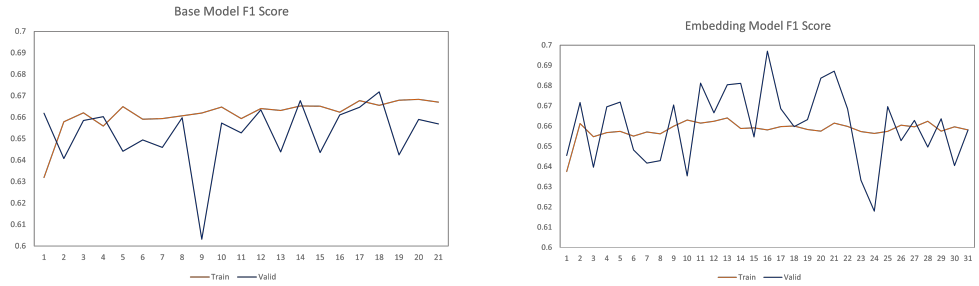
(b) Final Model (Aug) Accuracy

Figure 4.6: Final Model (Augmented) Training Results

$$F1 = 2\frac{P \times R}{P + R}$$
$$P = \frac{TP}{TP + FP}$$
$$R = \frac{TP}{TP + FN}$$

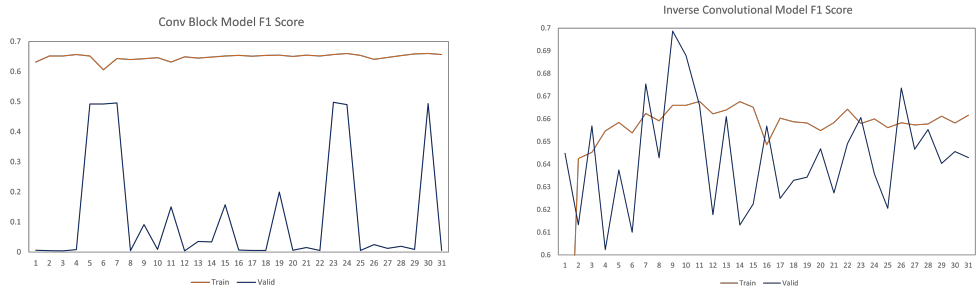
where P and R are precision and recall respectively, and TP, FP and FN are the number of true positives, false positives and false negatives respectively. As we use the F1 score in a multiclass problem, the F1 score formula is adjusted slightly to average the F1 scores across our classes. There are multiple methods of doing this: micro, macro and weighted averaging. Micro averaging computes the global average of the precision and recall before computing the F1 score. For example, it takes the total number of true positives across all classes over the total number of predictions for both precision and recall, this means when each point can be classified as only one class, as in our case, micro-averaged F1

scores are equivalent to our previously calculated accuracy. Macro-averaged F1 scores take the simple arithmetic mean of the F1 scores for each class, while the weighted average F1 score will give more weight to classes with higher numbers of target occurrences. For this study we will use the weighted-F1 score as although we are being careful to ensure our models do not tend too far towards predicting the most common classes, we also must acknowledge that the occurrences of shots and null events are somewhat more impactful on a game, and less random, than the rarer turnovers or fouls. This can be paired with the accuracy of each model to ensure that our predictions are not unbalanced, while also measuring the relative usefulness of each model in real-world applications. A general guideline for acceptable F1 scores for two-class problems is approximately 0.7. Therefore, for our four-class problem we would consider this a good level of achievement.



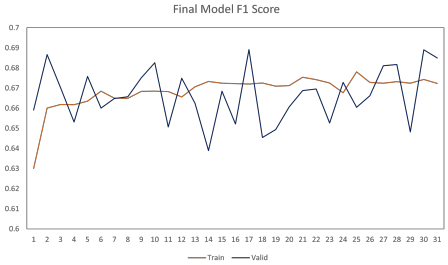
(a) Base Model F1 Score (b) Embedded Model F1 Score

Figure 4.7: Base Model and Embedded Model F1 Scores

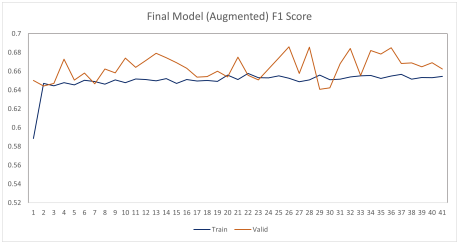


(a) Conv. Block Model F1 Score (b) Inv. CNN Model F1 Score

Figure 4.8: Conv. Block and Inv. CNN Model F1 Scores



(a) Final Model F1 Score



(b) Final Model (Aug) F1 Score

Figure 4.9: Final Model and Final Model (Augmented) F1 Scores

We can see that similarly to the loss and accuracy graphs, the F1 score for our models quickly plateau at the same level. The convolutional block model F1 scores can again be used as an indication of the upper end of attainable scores. Pairing the results of our accuracy and F1 measurements, it is clear that our model does not only output the most common class as our accuracy is lower than could be achieved by predicting "null" in every case. This indicates that for a significant portion of these cases our model predicts another terminal outcome. Our F1 scores also indicate that despite this, our model is finding a good balance between precision and recall, with F1 scores approaching our rough benchmark of 0.7.

The final model was also tested on our reserved test set. Additionally, the MSE of the class predictions and value predictions are measured separately as we are able to compare these to the results achieved by Sicilia, Pelechrinis, and Goldsberry [53], the closest model model to ours in terms of aim and application. The model is assessed using a sampled test set as well as full games. The sampling scheme is performed similarly to that of the training and validation sets, while full games are tested as this is closest to the proposed application of the model.

Our experimental results show an improvement over the closest model reported in literature, showing the effectiveness of our approach. While we must be aware that the MSE for the DeepHoops model is calculated using five classes, where fouls are split between shooting and non-shooting fouls, it is

	Final Model (Full)	Final Model (Sampled)	DeepHoops
Loss	1.4197	1.4671	0.3598
Class MSE	0.1205	0.1202	0.3598
Point MSE	0.5300	0.5325	-
Class F1	0.5620	0.6855	-
Class Accuracy	0.6286	0.6302	-

Table 4.1: Final Model Results on Test Set

unlikely this is able to explain the difference between the results of the two models.

The individual f1 scores for each class are also calculated, along with the confusion matrix from our sampled test experiment.

Null	Shot	Foul	Turnover
0.75223	0.65287	0.37545	0.16569

Table 4.2: Test Set F1 Scores by Class

True\Pred	Null	Shot	Foul	TO
Null	55507	6624	9417	13852
Shot	3236	22118	2892	8641
Foul	520	329	4010	645
TO	2138	745	1053	2823

Figure 4.10: Final Model Prediction Confusion Matrix

We find that the model is perhaps overconfident with predictions of turnovers and fouls, as seen by the confusion matrix, which is also evident in the class F1 scores. The precision of the model is relatively high while the recall is fairly lacking. It is common for there to be a trade-off between precision and recall, and it is likely that the current balance is due to the weighting aspect of the loss function prioritising rare classes in our imbalanced dataset.

We can also inspect the predictions of our model on a more granular scale, by looking at the outputs across a single game. This allows us to view the

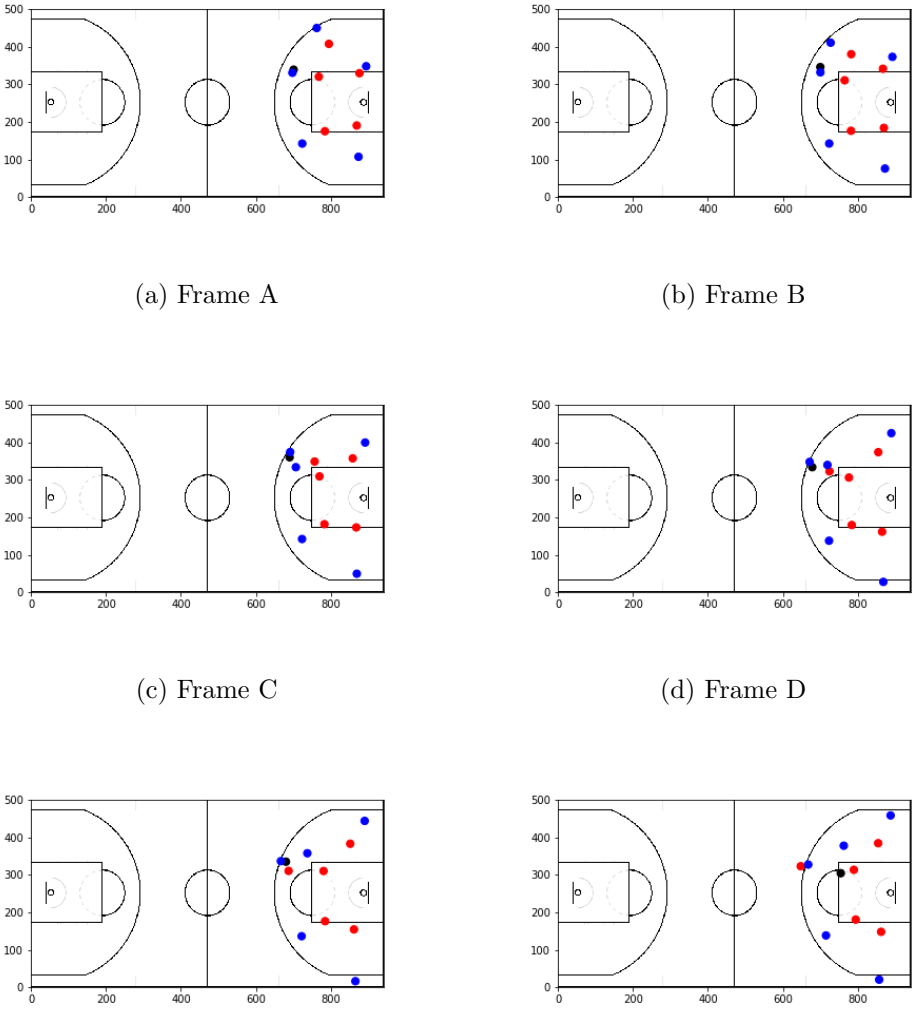
model in a more realistic application. As an example the game 0021500321 is analysed in more depth. The game was between the Houston Rockets and the Washington Wizards on the 9^{th} December 2015. (Predictions can be found in Appendix A.)

Generally the model predictions line up with what is observed, particularly with predictions pertaining to shots. Furthermore, the model predicts shot occurrences when none are later observed, potentially indicating the model is identifying situations where a shot could have been taken but was not for some reason. The same can be said for both fouls and turnovers, albeit slightly less reliably: we see the model identifying possible situations where a foul or turnover becomes more likely.

The graphs for our possession value also mirror these observations, predominantly hovering around the 0.6 points mark, with spikes or dips at key moments. We see that the model is also able to differentiate between opportunities of differing quality, where spikes have comparatively higher or lower peaks. Encouragingly, predictions are generally higher for situations where points are scored, which could indicate that players are generally making good decisions offensively. Moreover, the model is able to output values above 2, further differentiating between 2 and 3 point opportunities.

The model output across a single possession is inspected further to see how predictions evolve as play progresses. These outputs are shown in Figures 4.11, 4.12 and 4.13.

We can see that the model does not detect any significant opportunities for the blue team to score until frame D. At this point the blue player has turned to face the hoop with the ball, aided by a team mates off-ball screen to gain additional space. The Expected Points drops as the red defender closes down the blue player, but ultimately the blue player is able to get a shot off. The rise in Expected Points after frame E, where the shot was taken, can be attributed to the red player failing to block the shot. The maximum Expected Points for this play, even post-shot, is below 1. This fits our intuition as the



(a) Frame A

(b) Frame B

(c) Frame C

(d) Frame D

(e) Frame E

(f) Frame F

Figure 4.11: Key Frames of Highlighted Play

blue player has taken a long-range 2-point shot, which we would expect to be scored less than 50% of the time.

If we then isolate a single player, player 1, at a given moment and rerun the model with variations on this players positioning, we are able to see the degree to which this player's positioning influences the value of the play. Furthermore, we are also able to determine the effectiveness of the player's positioning in affecting the possession by comparing the observed value to the optimal value for the defensive team. The model is re-run with variations on this player's positioning, where the velocity components are computed by subtracting the

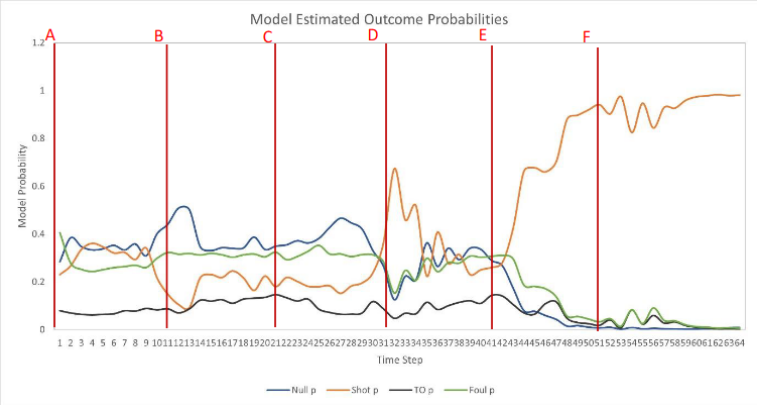


Figure 4.12: Play Predicted Outcomes

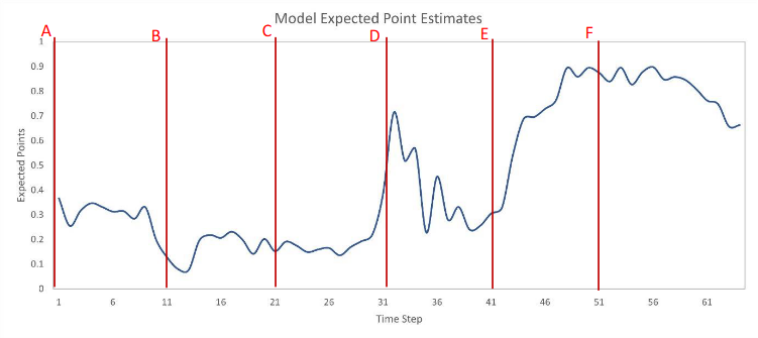


Figure 4.13: Play Predicted Points

coordinates of the player at the previous timestep from the new simulated coordinates. Essentially we are finding what the player should have done at the previous timestep, and as such only positions near the current player's position are simulated to ensure positions are realistic.

With a current possession value of 0.386 and an optimal value of 0.153, for the defending team, a simple measure of the player's positioning can be computed by dividing the optimal value, V_o, by the observed value, V_c.

$$P_d = \frac{V_o}{V_c} = \frac{0.153}{0.386} = 0.396$$

Which shows our model believes a mistake was made in the positioning of player 1. We can visualise this by plotting a heat map of the expected value against the position of player 1 in Figure 4.14. In this case darker red locations indicate the Expected Points for the play is predicted to be lower should player

1 be positioned at that point.

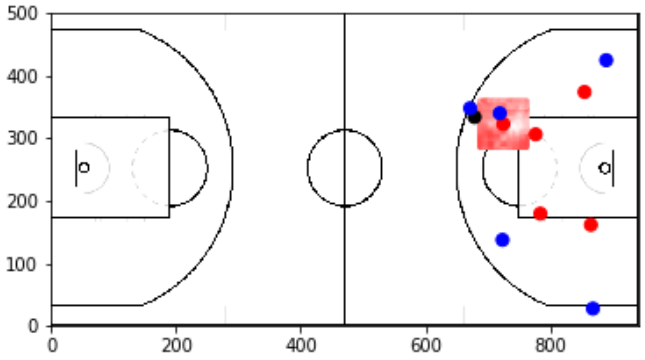


Figure 4.14: Heatmap of Expected Points

Our model predicts a lower Expected Points for the play if player 1 were to be positioned closer to the blue attacker with the ball, and more directly in-front of the attacking player. There is also a very clear white area near player 1's location, and another darker patch near their team mate. This could be indicating that the "optimal" location doesn't offer any great opportunities to cut off the scoring opportunity. The model output could suggest that positioning a defender to either try to block the blue player's shot, guarding against a drive towards the hoop, or even just being in the shooter's eye-line would be better than being located behind a supporting attacker's screen.

Naturally, there exists an equivalent measure for offensive positioning with an identical formula. Clearly, the optimal values will be higher than the observed in attacking contexts, and lower in defensive contexts, meaning P_o will take values $\in [0, 1]$ and P_d will take values $= 1$, where 1 is the optimal value for each. If one were to take some average of this across a player's gametime (where key situations may be weighted more heavily), we would have measures of a players offensive and defensive positioning efficiency. We can also hypothesise a metric which satisfies the desire among analysts for all-in-one measures, by taking P_o/P_d. This would yield another value $\in [0, 1]$ again where 1 indicates perfect positioning according to the model, measuring the overall

effectiveness of a player's positioning in both phases of the game.

Chapter 5

Conclusion

The aim of this thesis was to investigate the use of deep Artificial Neural Networks within the context of professional sport as a method of quantifying and measuring player positioning. Of particular interest were the effects of defensive player positioning due to the historic lack of adequate metrics with which to rate player's defensive play. We chose to study the application of these models specifically within professional basketball, largely due to the combination of data availability relative to other sports and the rule-set of basketball providing an environment in which the design and implementation of machine learning models is relatively straightforward. In published literature this problem is still being explored, with similar problems being addressed using a range of machine learning models. A range of Deep Learning techniques have been applied previously within this context, with the closest to addressing this particular problem being a stacked LSTM model. We used two datasets, the first detailing the locations of every player on court and the ball 25 times per second for 649 games in the 2015/16 NBA season, and the second being the play-by-play events of every game in the same season. We combined these datasets to remove overlapping frames and label the data appropriately. To address the problem, a Convolutional Neural Network was used as the basis of the model. This was intended to capture the spatial effects of the players. It was found that the performance of the model quickly plateaued with improvements diffi-

cult to obtain with multiple modules trialled to compliment the existing model structure, unfortunately little in terms of added performance was found. The final model consisted of four convolutional layers followed by four fully connected layers, with leaky ReLU activations. We opted for a two headed model output, with possession outcome predictions and expected point value heads. Therefore, the loss function was a multi-objective loss function comprised of a Cross Entropy Loss component for the class predictions and a Mean Squared Error component for the expected point predictions. This model yielded effective results, with accuracy and multi-class F1 scores indicating the model produces good predictions for both outcome class and expected point outputs while not over-fitting towards the dominant class or point value. We also computed the Mean Squared Error of our possession outcome predictions on a reserved test set as a means of comparison to similar research, and found that our model produces a lower Mean Squared Error than the closest application in literature. Finally, we outlined a method of computing the optimal position for a single player at a given moment, and a way of quantifying and comparing the positioning efficiency across players, both in and out of possession.

5.1 Future Work

Further extensions to this work could improve upon the result achieved here. The clearest area of improvement would be the introduction of more independent data. This study has shown that merely augmenting the data through flipping the frames along the x-axis is insufficient for achieving significant improvements. As stated, this method assumed that teams were not already running mirrored tactical schemes, where another season of data would likely introduce further tactical developments. Another way of conceptualising this issue is that our current model is well suited for evaluating games in the 2015/16 season, but this may not necessarily translate as well into previous or subsequent seasons. Larger alterations to the network architecture could

also lead to improvements. Our model is largely based on convolutional layers, while other model types such as Graph Neural Networks or Transformers may in fact produce superior results. As discussed in Chapter 2, other publications have found success with larger sets of input features, usually the result of previous modelling done by authors. Introducing additional features at the input stage could allow for some aspects of the game to be introduced that our model is unable to extract from the images used in this thesis.

Additionally, we introduced a potential metric to measure a player's positioning efficiency. This too can be further developed. Through collaboration with professionals in the field, the metric may be further refined to better suit the use cases found in professional sports. As mentioned, the metric could be weighted based on the importance of a situation or the distance to the defender's hoop, giving greater importance to more crucial situations. One could also study how this metric, or one similar, performs as a predictive or recruitment tool. For example, if all players in the league were measured and ranked, how effective would the metric be as a way of predicting a team's points conceded? And is the spread of scores wide enough to differentiate between high and low performers when accounting for noise? Finally, computing this metric for all players across a season is time consuming, and improvements to the run-time of this would likely be beneficial, whether this takes the form of a smaller network or improved hardware.

Bibliography

[1] Jim Albert. "Sabermetrics: The Past, the Present, and the Future". In: (Mar. 2010). DOI: 10.5948/UPO9781614442004.002.

[2] Chris Anderson and David Sally. *The Numbers Game*. Penguin Group, 2014.

[3] Baseball Hall of Fame. *Henry Chadwick*. URL: https://baseballhall.org/hall-of-famers/chadwick-henry (visited on 01/07/2024).

[4] Baseball Savant. *Statcast Search*. URL: https://baseballsavant.mlb.com/statcast_search (visited on 01/08/2024).

[5] Ted Berg. *The Metrics System: How MLB's Statcast is creating baseball's new arms race*. 2015. URL: https://ftw.usatoday.com/2015/05/mlb-statcast-stats-data-launch-angle-route-efficiency (visited on 01/08/2024).

[6] Murray Campbell, A. Joseph Hoane Jr., and Feng-hsiung Hsu. "Deep Blue". In: *Artificial Intelligence* 134 (2002), pp. 57–83.

[7] Daniel Cervone et al. "A Multiresolution Stochastic Process Model for Predicting Basketball Possession Outcomes". In: *arxiv* 1408.0777v3 (2016).

[8] Albert Chen. *The Metrics System: How MLB's Statcast is creating baseball's new arms race*. 2016. URL: https://www.si.com/mlb/2016/08/26/statcast-era-data-technology-statistics (visited on 01/08/2024).

[9] François Chollet. *Deep Learning with Python*. Manning Publications Co., 2108.

[10] Dimitrije Curcic. *Racial Bias in NBA Commentary (Analysis)*. https://runrepeat.com/racial-bias-in-nba-commentary, Last accessed on 2024-01-06. 2023.

[11] Neil Davidson. *Raptors hope technology leads to success with new multimedia analytic board*. 2022. URL: https://www.cbc.ca/sports/basketball/nba/toronto-raptors-multimedia-analytic-board-1.6613209 (visited on 01/12/2024).

[12] Uwe Dick and Ulf Brefeld. "Learning to Rate Player Positioning in Soccer". In: *Big Data* 7.1 (2019), pp. 71–82. DOI: 10.1089/big.2018.0054.

[13] Uwe Dick, Maryam Tavakol, and Ulf Brefeld. "Rating Player Actions in Soccer". In: *Frontiers in Sports and Active Living* 3 (2021). DOI: 10.3389/fspor.2021.682986.

[14] Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning". In: *arxiv* 1603.07285v1 (2016).

[15] Dunks & Threes. *What is Estimated Plus-Minus (EPM)?* URL: https://dunksandthrees.com/about/epm (visited on 01/12/2024).

[16] ESPN. *THE GREAT ANALYTICS RANKINGS*. 2015. URL: http://www.espn.com/espn/feature/story/_/id/12331388/the-great-analytics-rankings# (visited on 01/12/2024).

[17] Javier Fernández, Luke Bornn, and Daniel Cervone. "A framework for the fine-grained evaluation of the instantaneous expected value of soccer possessions". In: *Machine Learning* 110 (2021), pp. 1389–1427.

[18] Frederic Friedel. *Reconstructing Turing's "Paper Machine"*. 2017. URL: https://en.chessbase.com/post/reconstructing-turing-s-paper-machine (visited on 01/08/2024).

[19] Larry Greenemeier. *20 Years after Deep Blue: How AI Has Advanced Since Conquering Chess*. 2017. URL: https://www.scientificamerican.com/article/20-years-after-deep-blue-how-ai-has-advanced-since-conquering-chess/ (visited on 01/08/2024).

[20] Chuan Guo et al. “On Calibration of Modern Neural Networks”. In: *arxiv* 1706.04599v2 (2017).

[21] Larry Hardesty. *Explained: Neural networks*. 2017. URL: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414 (visited on 01/15/2024).

[22] Simon Haykin. *Neural Networks A Comprehensive Foundation*. Prentice Hall International, Inc., 1999.

[23] *History of Go Ratings*. URL: https://www.goratings.org/en/history/ (visited on 01/08/2024).

[24] Yoshifusa Ito. “Representation of functions by superpositions of a step or sigmoid function and their applications to neural network theory”. In: *Neural Networks* 4.3 (1991), pp. 385–394. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(91)90075-G. URL: https://www.sciencedirect.com/science/article/pii/089360809190075G.

[25] Mike James. *Number Of Legal Go Positions Finally Worked Out*. 2016. URL: https://www.i-programmer.info/news/112-theory/9384-number-of-legal-go-positions-finally-worked-out.html (visited on 01/09/2024).

[26] Neil M. Johnson. *nba-movement-data*. 2016. URL: https://github.com/sealneaward/nba-movement-data (visited on 01/15/2024).

[27] Krešimir Josić. *The Mechanical Turk*. 2012. URL: https://engines.egr.uh.edu/episode/2765 (visited on 01/08/2024).

[28] Bryan Kalbrosky. *What is the best advanced statistic for basketball? NBA executives weigh in*. 2021. URL: https://hoopshype.com/lists/advanced-stats-nba-real-plus-minus-rapm-win-shares-analytics/ (visited on 01/12/2024).

[29] Patrick Kiernan. *Which is greater? The number of atoms in the universe or the number of chess moves?* URL: https://www.liverpoolmuseums.org.uk/stories/which-greater-number-of-atoms-universe-or-number-of-chess-moves (visited on 01/09/2024).

[30] Diederik P. Kingma and Jimmy Lei Ba. "Adam: A Method for Stochastic Optimization". In: 2015.

[31] *Komodo*. URL: https://www.chessprogramming.org/Komodo#Evaluation (visited on 01/08/2024).

[32] Emanuel Lasker. *Lasker's Manual of Chess*. E P Dutton, New York, 1926.

[33] Daniel Link, Steffen Lang, and Philipp Seidenschwarz. "Real Time Quantification of Dangerousity in Football Using Spatiotemporal Tracking Data". In: *PLOS ONE* (2016). DOI: 10.1371/journal.pone.0168768.

[34] Steven A. Lopez. *An "easy" engine for the Fritz/Rybka interface*. 2011. URL: https://uscfsales.wordpress.com/2011/07/01/an-%E2%80%9Ceasy%E2%80%9D-engine-for-the-fritzrybka-interface/ (visited on 01/08/2024).

[35] Zach Lowe. *Lights, Cameras, Revolution*. 2013. URL: https://grantland.com/features/the-toronto-raptors-sportvu-cameras-nba-analytical-revolution/ (visited on 01/12/2024).

[36] Major League Baseball. *Fielding Independent Pitching (FIP)*. URL: https://www.mlb.com/glossary/advanced-stats/fielding-independent-pitching (visited on 01/08/2024).

[37] Major League Baseball. *Fielding Percentage (FPCT)*. URL: https://www.mlb.com/glossary/standard-stats/fielding-percentage (visited on 01/08/2024).

[38] Major League Baseball. *Wins Above Replacement (WAR)*. URL: https://www.mlb.com/glossary/advanced-stats/wins-above-replacement (visited on 01/08/2024).

[39] Voros McCracken. *Pitching and Defense: How Much Control Do Hurlers Have?* 2001. URL: https://www.baseballprospectus.com/news/article/878/pitching-and-defense-how-much-control-do-hurlers-have/ (visited on 01/08/2024).

[40] Danny McLoughlin. *Racial Bias in Football Commentary (Study): The Pace and Power Effect*. 2023. URL: https://runrepeat.com/racial-bias-study-soccer (visited on 01/06/2024).

[41] Kostya Medvedovsky. *Daily Adjusted and Regressed Kalman Optimized projections — DARKO*. URL: https://apanalytics.shinyapps.io/DARKO/ (visited on 01/12/2024).

[42] Andreas C. Müller and Sarah Guido. *Introduction to Machine Learning with Python*. O'Reilly Media, Inc., 2016.

[43] Vinod Nair and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27 th International Conference on Machine Learning* (2010).

[44] NBAStuffer. *Analytics Movement in the NBA*. URL: https://www.nbastuffer.com/analytics101/nba-analytics-movement/ (visited on 01/12/2024).

[45] Thomas Nielsen. *NBA Basketball Datasets & CSV Files*. URL: https://sports-statistics.com/sports-data/nba-basketball-datasets-csv-files/ (visited on 01/15/2024).

[46] *official-stockfish*. 2024. URL: https://github.com/official-stockfish/Stockfish/blob/master/src/evaluate.cpp (visited on 01/08/2024).

[47] James A. Rada and K. Tim Wulfemeyer. "Color Coded: Racial Descriptors in Television Coverage of Intercollegiate Sports". In: *Journal of Broadcasting & Electronic Media* (2005).

[48] RM. *Juego de posición – A short explanation*. 2014. URL: https://spielverlagerung.com/2014/11/26/juego-de-posicion-a-short-explanation/ (visited on 01/12/2024).

[49] Sarah Rudd. *A Framework for Tactical Analysis and Individual Offensive Production Assessment in Soccer Using Markov Chains*. URL: http://nessis.org/nessis11/rudd.pdf. 2012.

[50] Julian Schrittwieser et al. "Mastering Atari, Go, chess and shogi by planning with a learned model". In: *Nature* 588 (2020), pp. 604–612.

[51] John Schuhmann. *NBA dives headlong into new era of statistical analysis*. 2009. URL: https://web.archive.org/web/20170330044407/http://www.nba.com/2009/news/features/john_schuhmann/10/23/stats.analysis/ (visited on 01/12/2024).

[52] Shrot. *Do Basketball Games Switch Sides? (All You Need To Know)*. 2022. URL: https://americansportsplanet.com/do-basketball-games-switch-sides/ (visited on 01/16/2024).

[53] Anthony Sicilia, Konstantinos Pelechrinis, and Kirk Goldsberry. "DeepHoops: Evaluating Micro-Actions in Basketball Using Deep Feature Representations of Spatio-Temporal Data". In: *arxiv* 1902.08081v1 (2019).

[54] David Silver et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362 (2018), pp. 1140–1144.

[55] David Silver et al. "Mastering the game of Go with deepneural networks and tree search". In: *Nature* 529 (2016), pp. 484–503.

[56] David Silver et al. "Mastering the game of Go without human knowledge". In: *Nature* 550 (2017), pp. 354–371.

[57] Nate Silver. *How Our RAPTOR Metric Works*. 2019. URL: https://fivethirtyeight.com/features/how-our-raptor-metric-works/ (visited on 01/12/2024).

[58] Karun Singh. *Introducing Expected Threat (xT)*. 2018. URL: https://karun.in/blog/expected-threat.html (visited on 01/10/2024).

[59] Piper Slowinski. *FIP*. 2010. URL: https://library.fangraphs.com/pitching/fip/ (visited on 01/08/2024).

[60] Piper Slowinski. *What is WAR?* 2010. URL: https://library.fangraphs.com/misc/war/ (visited on 01/08/2024).

[61] Piper Slowinski. *wOBA*. 2010. URL: https://library.fangraphs.com/offense/woba/ (visited on 01/08/2024).

[62] William Spearman. "Beyond Expected Goals". In: *MIT Sloan Sports Analytics Conference* (2018).

[63] William Spearman. *Quantifying Pitch Control*. Feb. 2016. DOI: 10.13140/RG.2.2.22551.93603.

[64] William Spearman et al. "Physics-Based Modeling of Pass Probabilities in Soccer". In: Mar. 2017.

[65] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958.

[66] Stats Perform. *Assessing The Performance of Premier League Goalscorers*. 2012. URL: https://www.statsperform.com/resource/assessing-the-performance-of-premier-league-goalscorers/ (visited on 01/10/2024).

[67] Howard Staunton. "The Chess-Player's Handbook". In: Henry G. Bohn, 1847. Chap. 4.

[68] Leigh Steinberg. *CHANGING THE GAME: The Rise of Sports Analytics*. 2015. URL: https://www.forbes.com/sites/leighsteinberg/2015/08/18/changing-the-game-the-rise-of-sports-analytics/?sh=70f2c3744c1f (visited on 01/07/2024).

[69] David Sumpter. *Welcome to Soccermatics*. 2022. URL: https://soccermatics.readthedocs.io/en/latest/ (visited on 01/10/2024).

[70] David Sumpter and Aleksander Andrzejewski. *Expected Threat - Action-based*. 2022. URL: https://soccermatics.readthedocs.io/en/latest/lesson4/xTAction.html (visited on 01/10/2024).

[71] David Sumpter and Aleksander Andrzejewski. *Fitting the xG model*. 2022. URL: https://soccermatics.readthedocs.io/en/latest/gallery/lesson2/plot_xGModelFit.html (visited on 01/10/2024).

[72] Dan Vatterott. *Creating Videos of NBA Action With Sportsvu Data*. 2016. URL: https://danvatterott.com/blog/2016/06/16/creating-videos-of-nba-action-with-sportsvu-data/ (visited on 01/16/2024).

[73] Edward Winter. *The Value of the Chess Pieces*. 2022. URL: https://www.chesshistory.com/winter/extra/value.html (visited on 01/08/2024).

[74] Tom Worville. *How football's finest are using analytics to find an edge*. 2021. URL: https://theathletic.com/2882187/2021/10/12/how-to-find-the-edge-examining-premier-league-analytics-trends-what-is-to-come/ (visited on 01/16/2024).

[75] Calvin C. K. Yeung, Tony Sit, and Keisuke Fujii. "Transformer-Based Neural Marked Spatio Temporal Point Process Model for Football Match Events Analysis". In: *arxiv* 2302.09276v1 (2023).

[76] Aston Zhang et al. *Dive into Deep Learning*. 2021.

Appendix A

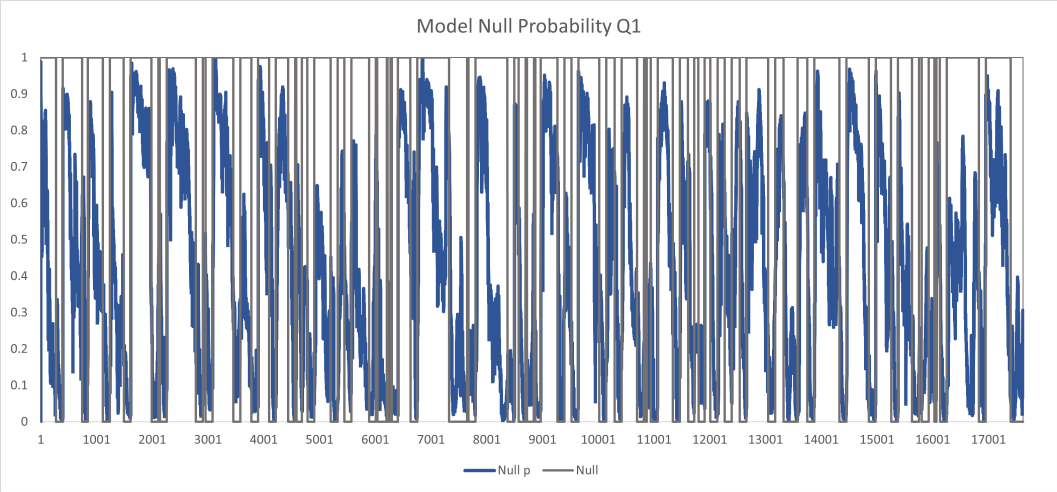


Figure A.1: Quarter 1 Null Prediction Probability

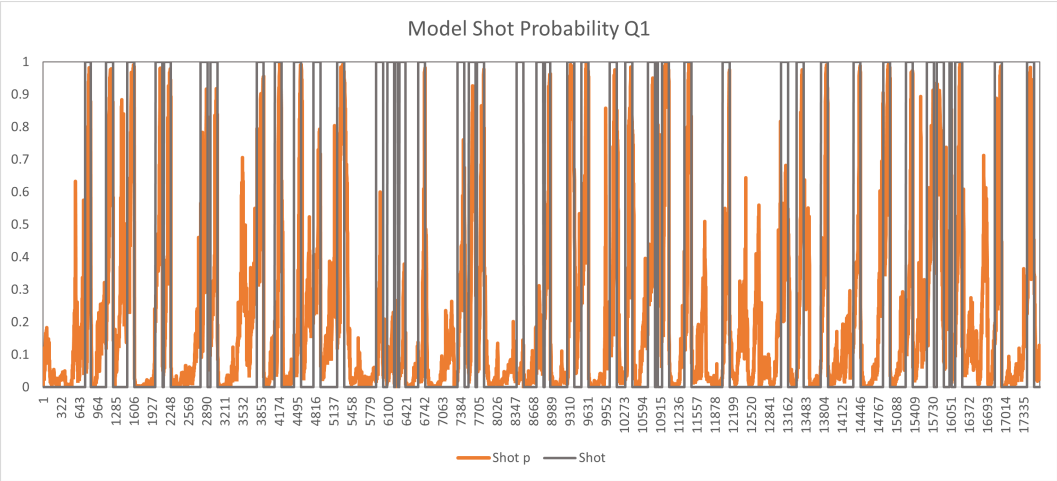


Figure A.2: Quarter 1 Shot Prediction Probability

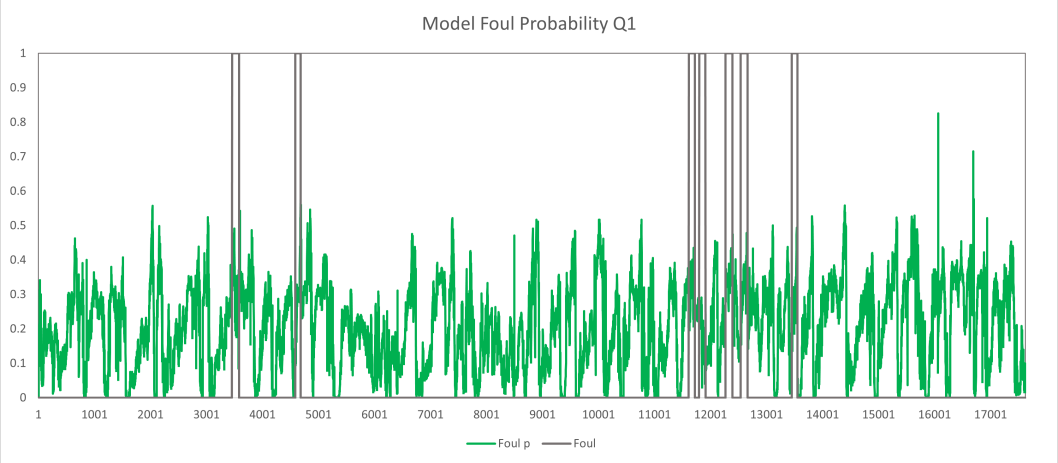


Figure A.3: Quarter 1 Foul Prediction Probability

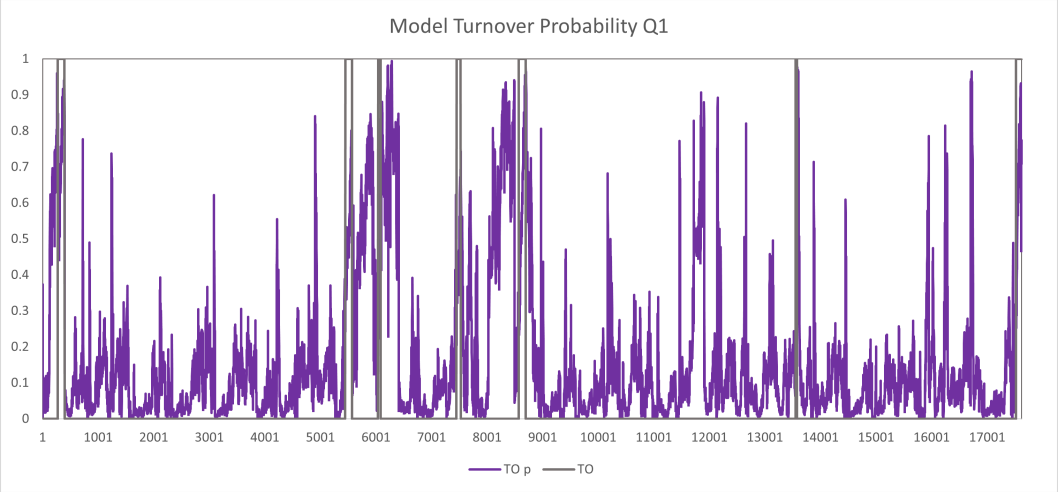


Figure A.4: Quarter 1 Turnover Prediction Probability

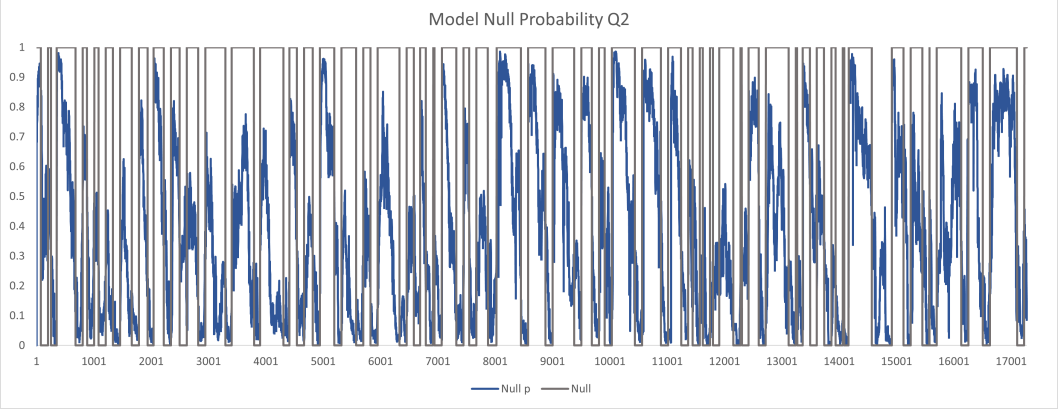


Figure A.5: Quarter 2 Null Prediction Probability

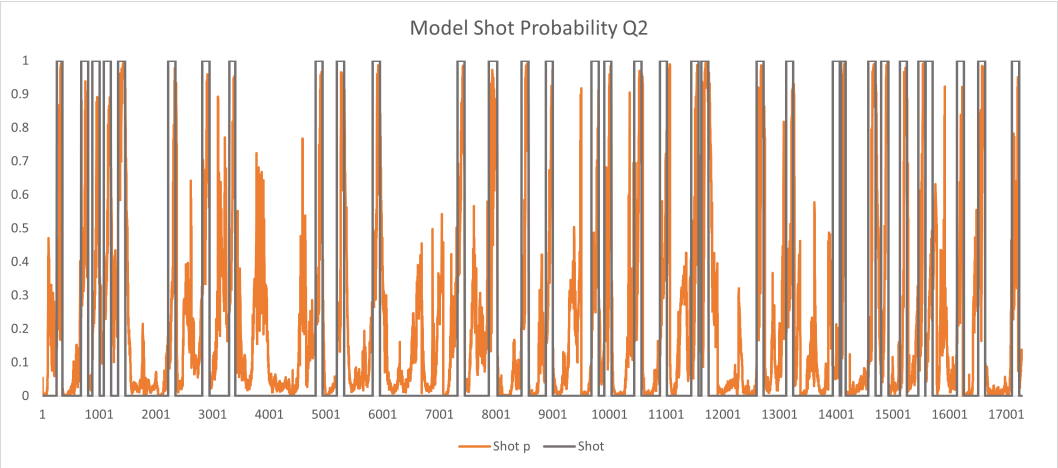


Figure A.6: Quarter 2 Shot Prediction Probability

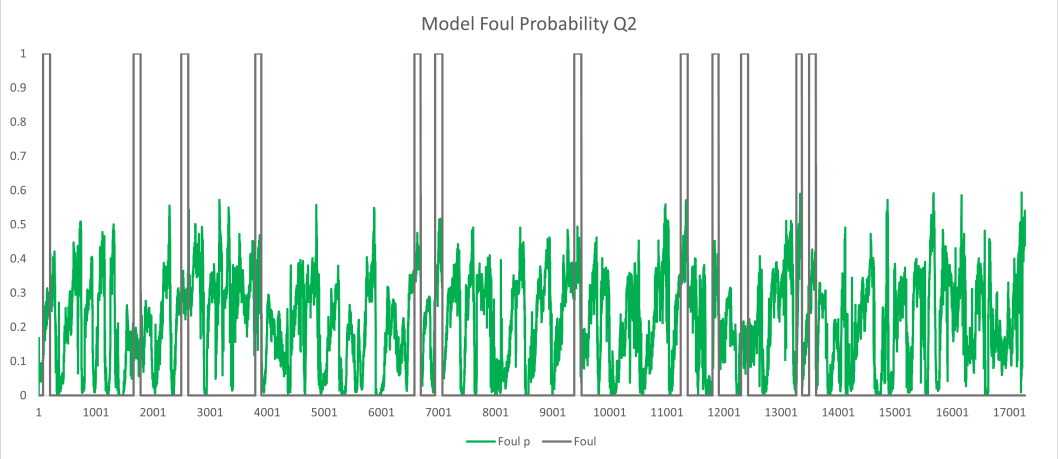


Figure A.7: Quarter 2 Foul Prediction Probability

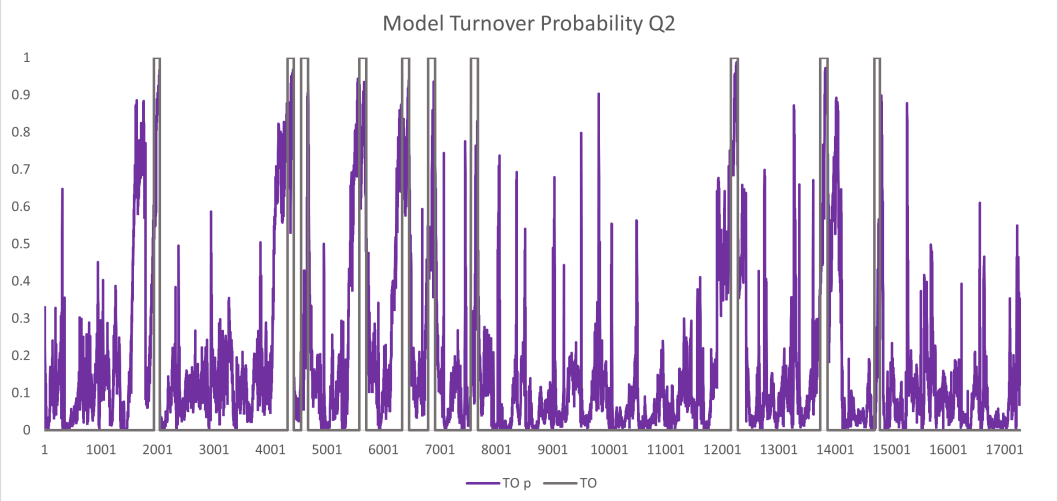


Figure A.8: Quarter 2 Turnover Prediction Probability

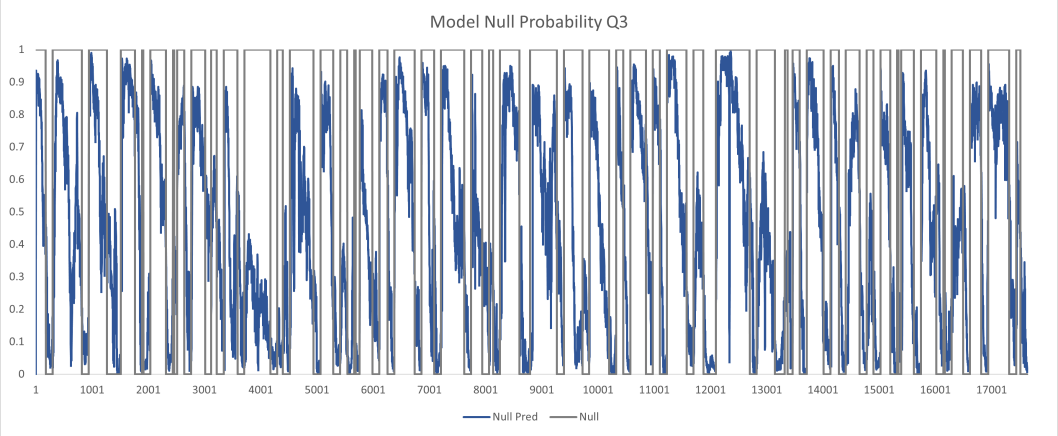


Figure A.9: Quarter 3 Null Prediction Probability

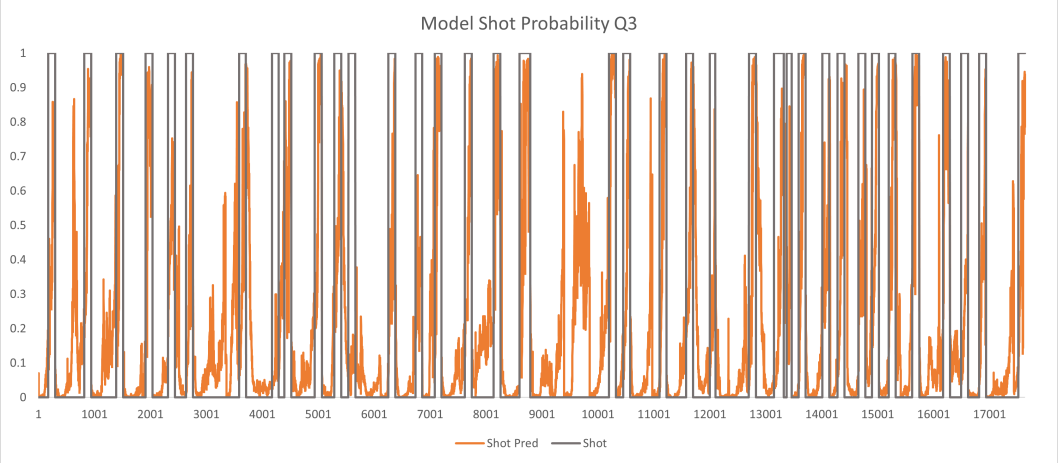


Figure A.10: Quarter 3 Shot Prediction Probability

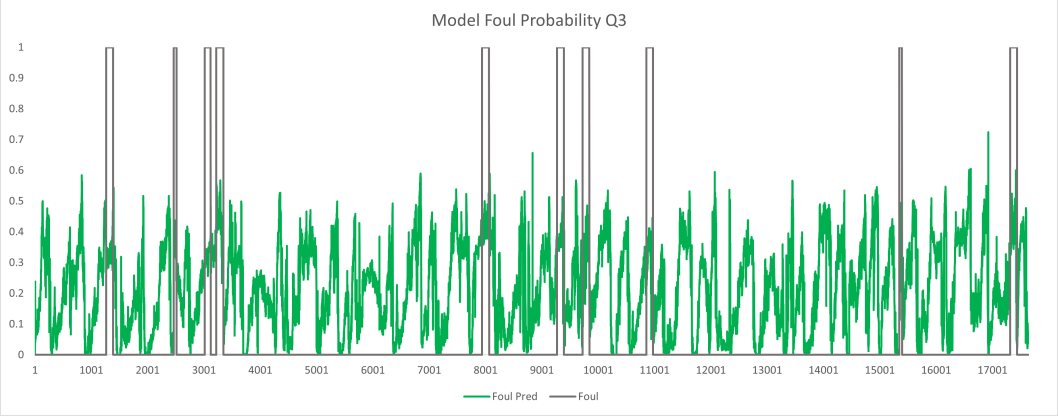


Figure A.11: Quarter 3 Foul Prediction Probability

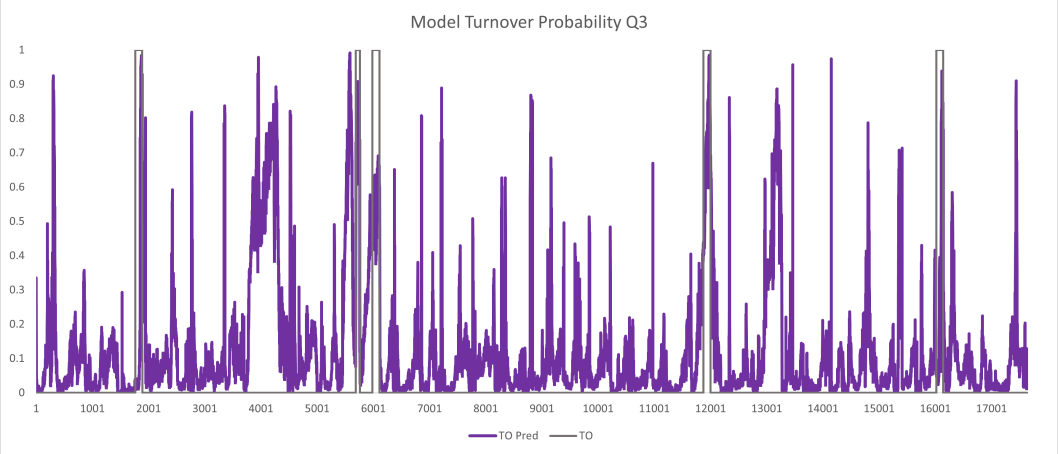


Figure A.12: Quarter 3 Turnover Prediction Probability

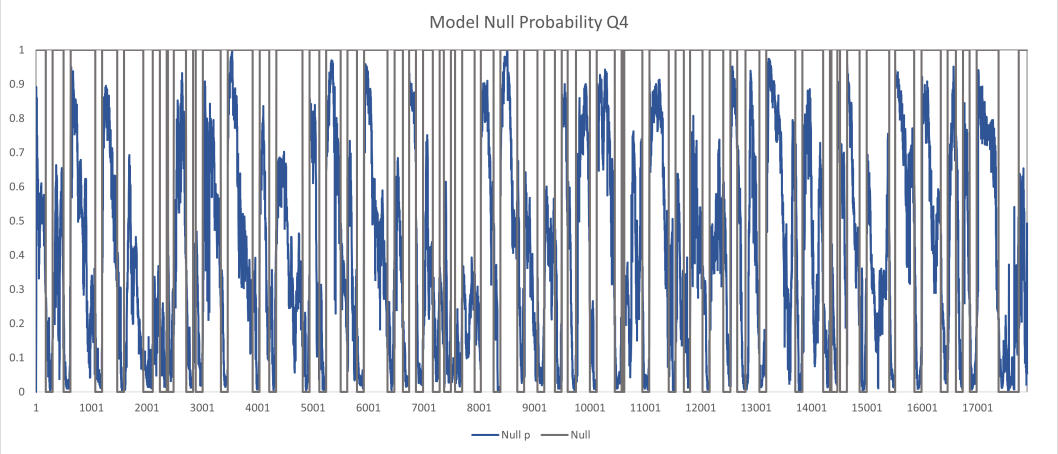


Figure A.13: Quarter 4 Null Prediction Probability

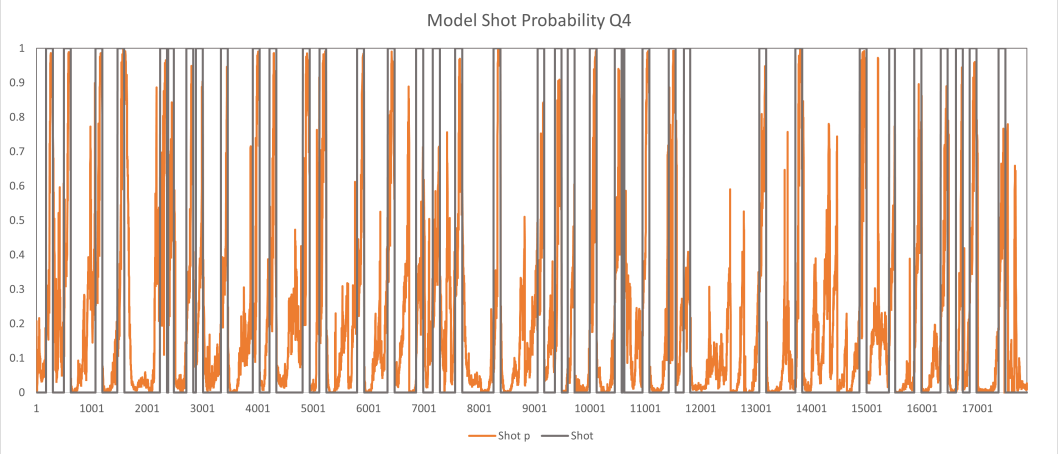


Figure A.14: Quarter 4 Null Prediction Probability

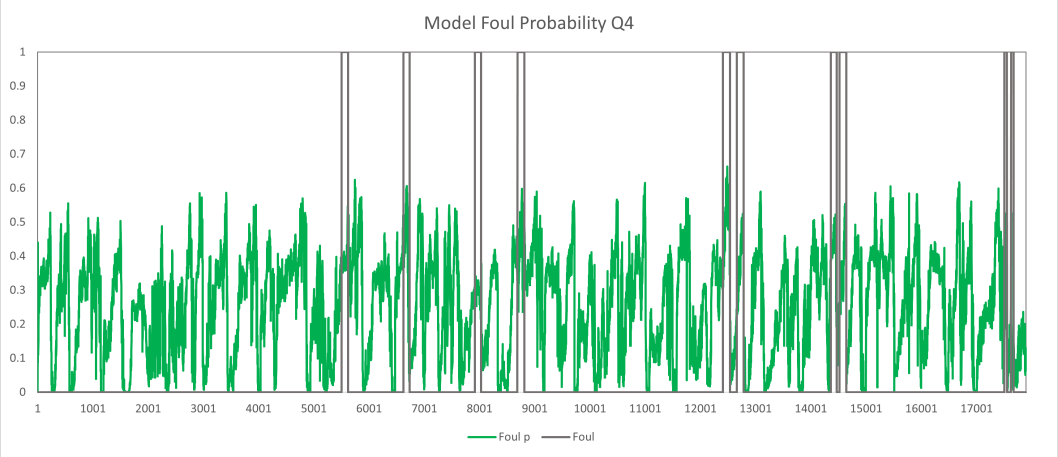


Figure A.15: Quarter 4 Null Prediction Probability

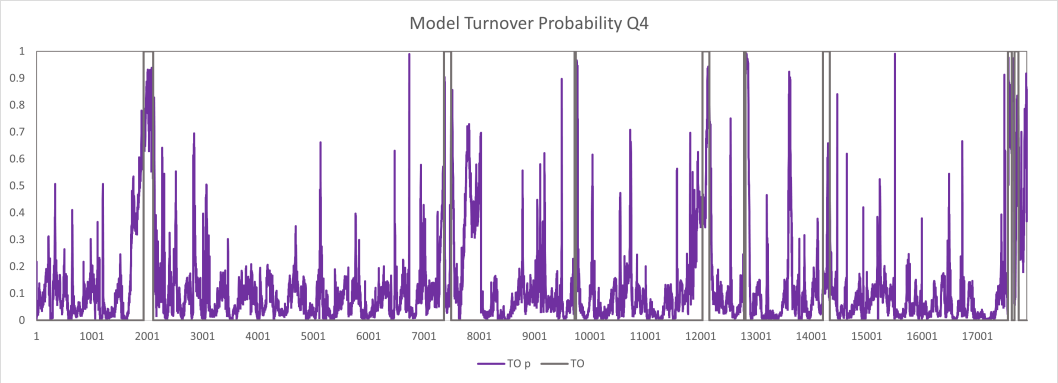


Figure A.16: Quarter 4 Null Prediction Probability

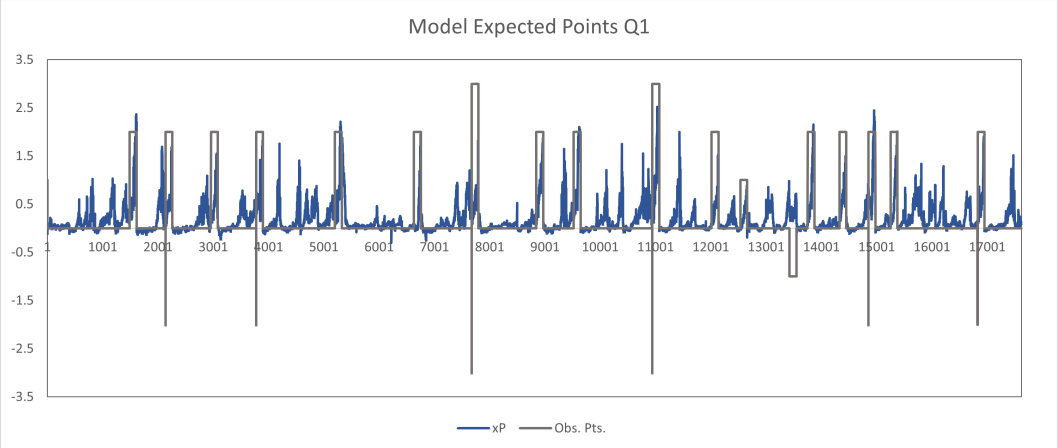


Figure A.17: Quarter 1 Expected Points Prediction

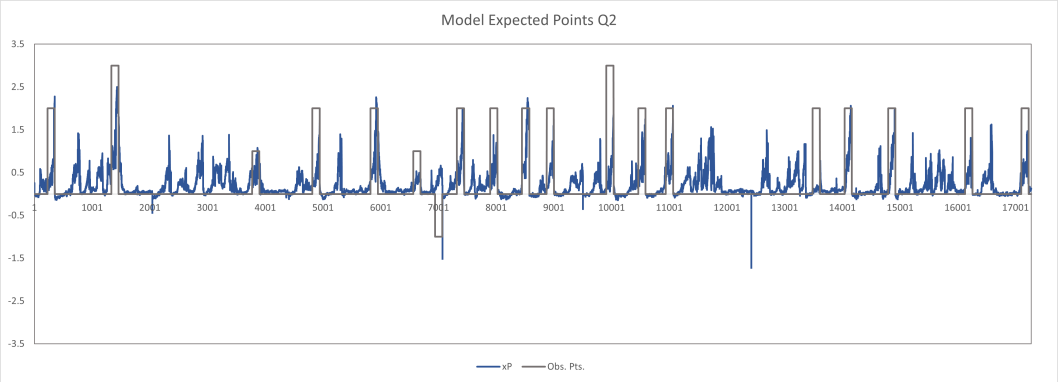


Figure A.18: Quarter 2 Expected Points Prediction

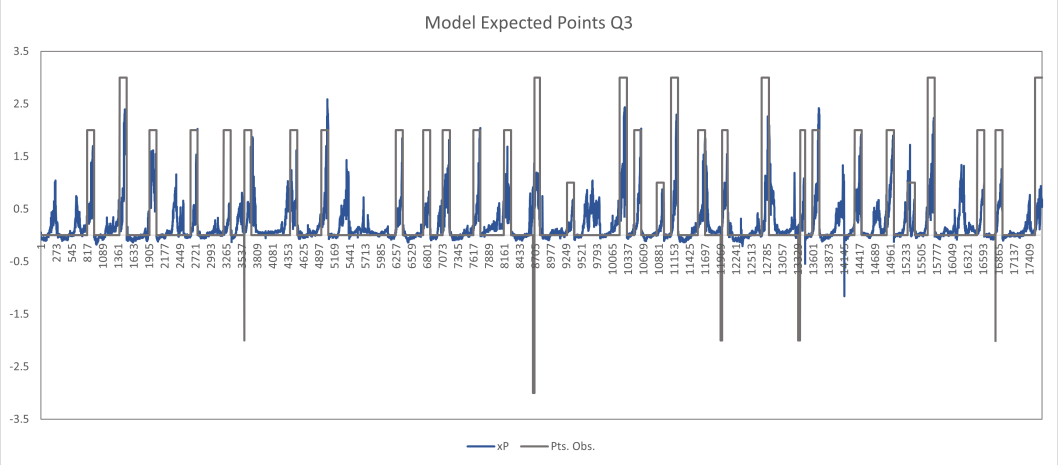


Figure A.19: Quarter 3 Expected Points Prediction

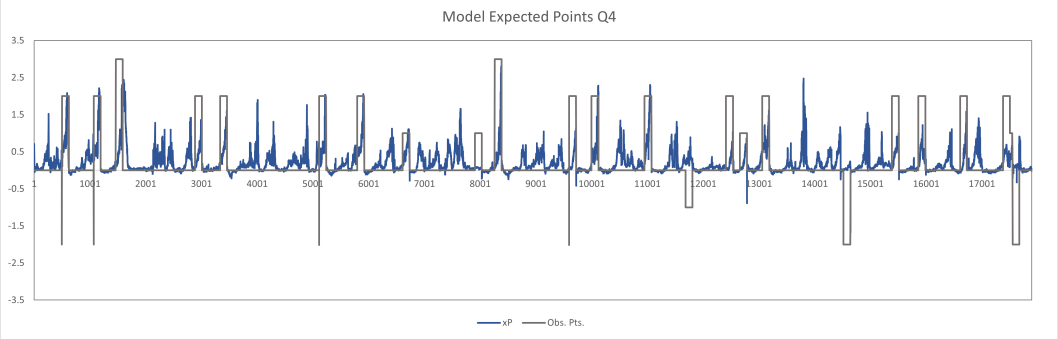


Figure A.20: Quarter 4 Expected Points Prediction