Temporal Consistency in Learning Action Values for Volleyball

by

Martin Ambrozic

M.Sc., Friedrich–Alexander University Erlangen–Nürnberg, 2013
M.Sc., Delft Technical University, 2013
B.Sc., University of Ljubljana, 2011

Project Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

Declaration of Committee

Name:	**Martin Ambrozic**
Degree:	**Master of Science**
Thesis title:	**Temporal Consistency in Learning Action Values for Volleyball**
Committee:	**Chair:** Ghassan Hamarneh Professor Computing Science
	Oliver Schulte Supervisor Professor Computing Science
	Mo Chen Committee Member Assistant Professor Computing Science
	Steven Ruuth Examiner Professor Mathematics

Abstract

Learning actions values is a key idea in sports analytics with applications such as player ranking, tactical insight and outcome prediction. We compare two fundamentally different approaches for learning action values on a novel play-by-play volleyball dataset. In the first approach, we employ regression models that implicitly assume statistical independence of data samples. In the second approach, we use a deep reinforcement learning model, explicitly enforcing the sequential nature of the data in the learning process. We find that temporally independent regression can in certain settings outperform the reinforcement learning approach in terms of predictive accuracy, but the latter performs much better when temporal consistency is required. We also consider a mimic regression tree as a way to add interpretability to the deep reinforcement learning approach. Finally, we examine the computed action values and perform a number of example analyses to verify their validity.

Keywords: Action Values; Volleyball; Reinforcement Learning; Machine Learning

Dedication

To Patrick, for being my anchor through 2020.

Acknowledgements

I would like to extend my deepest thanks to my supervisor Oliver Schulte for his unwavering guidance and support throughout my studies in Computing Science and for his help in the creation of this project.

I would also like to thank everyone in the UBC Men's Volleyball program for their contributions to the creation of the dataset used in this project and for allowing me to use that data in my research.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Increasing volume of collected data in all kinds of sports has made machine learning techniques a quickly developing area in sports science, but progress varies depending on the particular sport in question [6]. Learning the value of actions taken during a match is key to sports analytics. It offers applications such as player evaluation (by aggregating a player's action values) strategic analysis (by comparing actions in a given context) or even match outcome prediction [1, 13, 16, 18, 20].

An action value function maps the match state and an action to a real value which measures the team's chance of success after the action is taken. Two fundamental approaches have been explored for learning action values:

1. A regression approach where state-action inputs are treated as the independent variable and success chance as the dependent target variable. The regression approach implicitly assumes that the values for different state-action pairs are statistically independent of each other. However, in the actual data, the chances of success for successive state-action pairs are highly correlated. Examples of this approach include work on soccer data in [7] and [17].

2. Temporal difference learning, which exploits data correlation by forcing success predictions for successive state-action pairs to be consistent with each other, such as in [13] and [20].

This project compares the two approaches on a novel volleyball dataset collected in the past three seasons through a collaboration with the University of British Columbia Men's Volleyball program.

1.1 Preliminaries and Objectives

1.1.1 Markov Decision Process

A Markov decision process (abbreviated as MDP) is a discrete-time stochastic process consisting of a set of states S, a set of actions A, and a transition function $P_a(s, s')$ which gives the transition probability from state s into state s' given that action a was taken.

1.1.2 Action Value Function

Commonly, a reward function $R_a(s, s')$, which outputs a real-valued reward given a state transition, is provided along with an MDP. This allows for evaluation of actions taken in terms of expected future reward. The Q function, given for an action taking policy π as:

$$Q_\pi(s, a) = \mathbb{E}\left[R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1}) \,|\, S_t = s, A_t = a\right] \tag{1.1}$$

formalizes this. Here R_{t+1} is shorthand for $R_a(S_t, S_{t+1})$, i.e. the reward obtained by taking action a in state S_t and proceeding to state S_{t+1}. A discounting factor γ is used to avoid infinitely large accumulation of rewards in some applications, but following the example in hockey-related work [13, 21, 20], we set $\gamma = 1$ in our experiments. The main motivation is that points scored are worth the same regardless of when they are scored and longer rallies do not diminish the reward at the end.

1.1.3 Objectives

The aim of this project is to:

1. Construct an MDP from volleyball play-by-play event data. The states of the MDP represent match context and the actions correspond to player actions on the court.
2. Compute an action value function using:
   (a) regression learning that assumes temporal independence,
   (b) reinforcement learning that enforces temporal consistency

   and compare results.

1.2 Related Work

1.2.1 Computing Action Values

The concept of an action value function has been applied to event data in a number of different sports. In [5], expected value of possessions in basketball (equivalent to a Q function) was used on an NBA dataset. Similarly, soccer actions were evaluated in terms of scoring probability in [7]. However, this project relies mostly on work in the hockey context done by Liu, Routley and Schulte [13, 19, 20, 21] on large NHL event datasets. Their work has further been extended to consider pairs of players appearing on the ice together in [14]. A similar action value based approach was also taken in [10].

1.2.2 Volleyball Statistical Analysis

In volleyball, there have been previous applications of statistical methods involving Markov chains, but the dataset concerned actions of only one of the two teams and no game context such as current score was included. This is in contrast to our dataset, where both opponents' data is included. Skill importance was computed in this fashion in [15], rally winning probabilities were computed in [8] and home advantage was brought into question in [2].

1.2.3 Machine Learning in Volleyball

Moving to more recent research than the purely statistical methods listed in the previous section, machine learning has been applied to volleyball data with the goal of predicting future team rankings from win/loss data in [24]. Deep models were employed for activity recognition on video in [9] and [11], with the objective of determining training load and injury risk. In [27] and [25], analysis and prediction of attack patterns was carried out using artificial intelligence methods and [26] tackled classification of players into skill levels based on motion data from wearable devices.

Despite progress of machine learning in a variety of fields in volleyball, however, we did not encounter a context-aware approach to estimating action values that this project applies to a volleyball event dataset.

1.3 Report Structure

This report is structured as follows:

- In Chapter 2, we describe the volleyball dataset used and discuss the modelling decisions made in the process of constructing our MDP.
- Chapter 3 gives details on the various machine learning models used in computing the action value function.
- In Chapter 4, we discuss comparison metrics between the models and give results.
- Chapter 5 gives some examples of potential applications of the action value function in a practical setting.
- Chapter 6 draws conclusions and suggests future work.

Chapter 2

Dataset and the Markov Decision Process

2.1 Dataset Description

The volleyball dataset was collected in collaboration with the University of British Columbia's Men's Volleyball program. The UBC Thunderbirds compete in the Canada West division of U-Sports volleyball, which is the highest level of volleyball university competition in Canada. Their regular season consists of a double round robin phase, where they play each of the other Canada West teams twice, and a play-off phase, where top seeded teams play a single elimination bracket to determine the conference champion as well as participants in the U-Sports national final tournament. During the data collection time window, 12 teams participated in the Canada West competition, except in 2017/2018, where 13 teams competed.

The collection process was manual, using specialized software to record play-by-play data, either live at the game or after the fact using game video. In the former case, a second pass was needed to ensure quality and completeness of data. The format used was introduced by the company Data Project based in Salerno, Italy and has become widely adopted in professional and international volleyball. Both teams' actions were included for every game.

The dataset contains data from all games played by UBC over three seasons. Additionally, a number of matches between other teams in Canada West were included, originally intended for the purpose of tactical preparation for an upcoming opponent.

Table 2.1 describes the dataset size in terms of games, sets, rallies and game events.

	Games	**Sets**	**Rallies**	**Events**
Contained in Data	204	764	34,242	146,050

Table 2.1: Dataset size description in various terms.

Each entry in the dataset corresponds to an action event by a player in the game. We differentiate 7 action types:

- **Serve**: first contact of the serving team, the action of putting the ball into play over the net from behind the baseline.
- **Receive**: first contact of the team receiving the opponent's serve.
- **Set**: Second contact on either side, the aim of which is to deliver the ball to an attacker.
- **Attack**: Usually the third contact of either team, spiking the ball over the net with the aim of scoring by hitting the ground in the opponent's court.
- **Block**: The action of one or more players by the net attempting to defending against an attack by deflecting the ball back to the opponent's court as it crosses the net.
- **Dig**: The defensive action against an opponent's attack, volleying the ball before it contacts the ground.
- **Free-ball**: The first contact of a team after the opposing team failed to produce a proper attack and volleyed an 'easy ball' over the net.

Further, the outcome of each action is recorded, ranging from '=' (double negative), through '-' (negative), '/', '!', '+' (positive) and '#' (double positive). The meanings of these symbols vary by action type, but in general double negative corresponds to an error (awarding the rally to the opponent) and double positive is a winning or perfect outcome, eg. a scoring attack.

For most actions, the execution location is also recorded, as well as the trajectory location for attack and serve actions. Finally, we also record the context, i.e. the state of the game, including the current set number (1 through maximum 5) and the points scored by each team in the current set (usually between 0 and 25, but can be higher in a 'win by 2' scenario). Table 2.2 summarizes the context and action-related fields recorded in the dataset.

Action Field	**Player, Team**	**Type**	**Outcome**	**Location, Trajectory**
Range	categorical	7 categories	6 categories	Discretized 12-by-6 grid of locations

Context Field	**Set Number**	**Set Scores**
Range	Integers 1-5	Integers 0-31

Table 2.2: Summary of fields in the volleyball dataset. Action-related fields on top, context-related fields on the bottom.

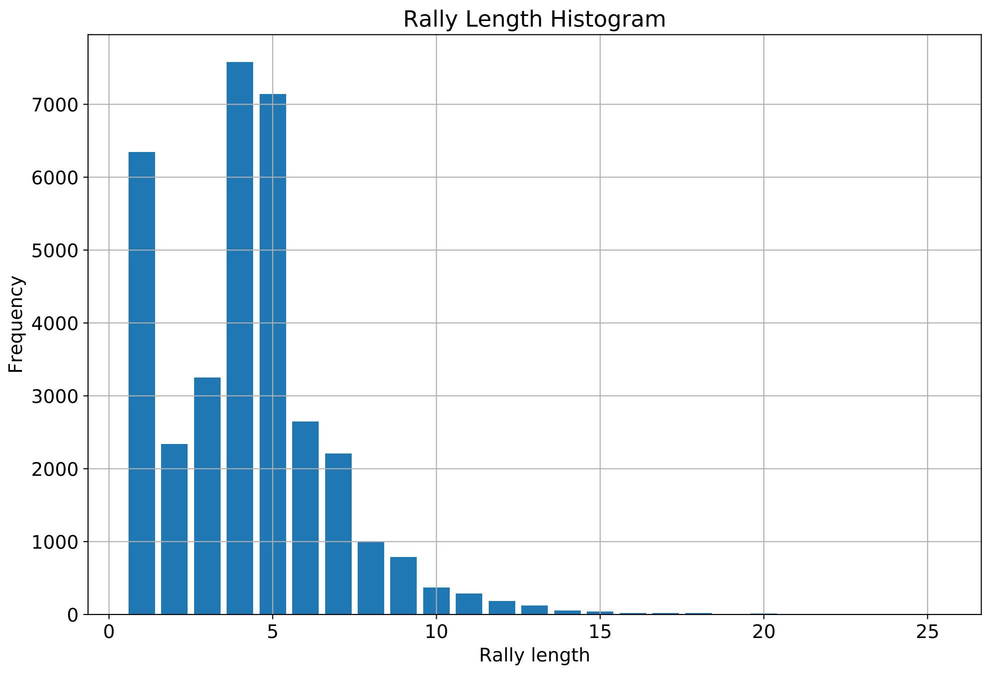

Figure 2.1: Rally length histogram for the volleyball dataset.

2.2 The Markov Decision Process

In volleyball, a rally is a sequence of actions beginning with the serve and ending with one of the teams scoring a point. We model every rally as a sequence of states and actions of a Markov decision process (abbreviated as MDP). Similarly to the NHL related work in [20], we include in the state space the match context as well as a limited history of previous actions taken by either team.

In determining the extent of action history to include, we consider the distribution of rally lengths (in terms of the number of actions). Figure 2.1 shows a histogram of rally lengths found in the dataset. We adopt a maximum history window of 9 actions, which means 96% of all rallies are encompassed in full from their beginning.

The state space S is therefore defined as a set of vectors $(c, a_0, a_1, ..., a_9)$, where c contains the match context features, i.e. the current set score and set number, and a_0 through a_9 contain action features as described in Table 2.2 for the current action and up to 9 past actions.

Every rally corresponds to a walk through the states of the MDP, where the action taken by either team determines the next state. Eventually one of the teams wins the point and the walk terminates. We model this by a reward function $r : S \times A \to \mathbb{Z}$, where S is the state space, A is the action space and

$$r(s,a) = \begin{cases} 1 & \text{if home team wins the rally} \\ -1 & \text{if away team wins the rally} \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

Our quantity of interest in this setting is the action value function (1.1). It expresses the expected future reward given the current state and action assuming future actions follow policy π.

Often in reinforcement learning applications, the objective is to compute the optimal policy π^* that maximizes future expected reward. This is *not* the case in this context, as the objective is only to *evaluate* the match states and actions *given* the policy observed in the data. The observed policy represents the behaviour of the average volleyball player. We thus omit the subscript π in (1.1) from here on. Additionally, as discussed in Section 1.1.2, we use a discounting factor of $\gamma = 1$ for all derivations and experiments to follow.

Chapter 3

Models

This chapter gives an overview of the various learning models employed in our experiments. As mentioned previously, we make the distinction between models that assume data independence on one hand and models that incorporate sequential dependence of the data on the other hand. The former are considered first in Section 3.1 and the latter follow in Section 3.2.

3.1 Temporally Independent Models

We first consider regression models that treat data samples as independent. The Q function gives the expected future reward in an episode starting from a given state-action pair by equation (1.1). Treating the current state and action as the independent variable x and the eventual reward $R(x)$ as the output, we can formulate this as a regression problem, where the target value for each sample in the training data is 1 or -1, depending on which team eventually won the rally. If we train a function estimator Q' that minimizes the mean squared error

$$\frac{1}{|X|}\sum_{x\in X}(Q'(x)-R(x))^2, \tag{3.1}$$

then $Q'(s,a)$ will approximate the conditional expected value

$$\mathbb{E}\left[R\,|\,x\right].$$

This follows from the property that conditional expectation of the target in general minimizes mean squared error. See, e.g. Bishop [3] Section 1.5.5 for a derivation of this property. In other words, since x is a state-action pair sample and $R(x)$ is the eventual reward, $Q'(x)$ will approximate the Q function. We now consider specific models that follow this approach.

3.1.1 Decision Tree

Decision trees are an appealing model because they offer interpretable function approximation modelled as a sequence of simple decisions. We use a regression tree class from the

Python scikit-learn package. Due to categorical fields in the dataset, a pre-processing step is needed, encoding the categorical data using a number of binary 'dummy' columns.

To avoid overfitting, we limit the maximum depth of the tree and use 5-fold cross-validation to determine the best value for maximum depth (i.e. the one that gives the minimal mean squared error averaged over the 5 cross-validation runs).

Table 3.1 shows experiment results for a range of maximal depth values. Setting the depth limit to 10 is adopted as the best model based on the mean squared error values shown.

max_depth	8	9	10	11	12	13
mean cross-validation MSE	0.6476	0.6302	0.6248	0.6260	0.6335	0.6448

Table 3.1: Mean MSE values for 5-fold cross-validation using a regression tree model with varying maximal depth.

A distinct feature of tree models is the ability to plot and examine the tree structure. We show the first three levels of the adopted tree in Figure 3.1. We notice the top level splits mostly concern the most recent two action outcomes, which seems sensible. Interestingly, it looks like action outcomes are considered before action types. This is likely due to the fact that outcome symbols, especially the extremes '#' and '=', share the meaning of 'point scored' and 'point lost' across several volleyball skills and splitting on those values correlates with winning or losing the rally without knowing the action type.

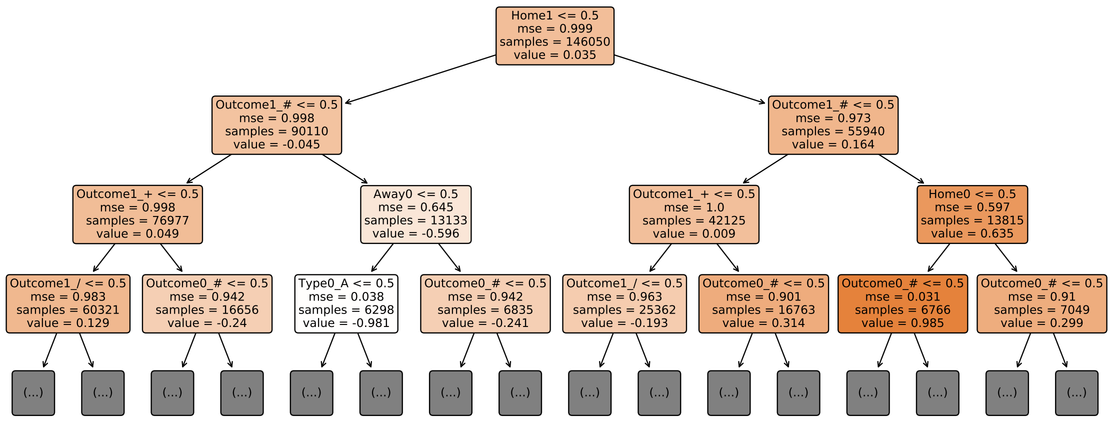

Figure 3.1: First levels of the regression tree structure.

3.1.2 Random Forest

While decision trees are an appealing model in terms of simplicity and interpretability, we wish to examine a potentially stronger regression model by using ensemble methods. The random forest model from the Python scikit-learn package allows for a straightforward implementation of an ensemble of trees. We use the same pre-processing steps as above and train an ensemble of 10 decision tree estimators (experiments showed that using more estimators beyond 10 did not provide a significant decrease in mean squared error). Table 3.2 shows mean MSE values for 5-fold cross-validation using a random forest model. An improvement over using a single decision tree can clearly be observed.

max_depth	10	11	12	13	14
mean cross-validation MSE	0.6177	0.6159	0.6170	0.6192	0.6222

Table 3.2: Mean MSE values for 5-fold cross-validation using a regression random forest model with varying maximal depth.

3.1.3 Regression Neural Network

We also explore the option of a neural network as a regression model. After experimenting with a range of hidden layers, activation functions and node counts, we decide on the following architecture:

- First dense layer of 200 nodes with RelU activation,
- Second dense layer of 20 nodes with RelU activation,
- Output layer with tanh activation.

Similarly to previous models, we employ 5-fold cross validation to determine the best number of training epochs to prevent overfitting to the training data. Table 3.3 shows mean MSE values for a range of training epoch counts. We adopted 7 epochs as the best value in further experiments.

epochs	4	5	6	7	8	9	10
mean cross-val. MSE	0.6162	0.6145	0.6139	0.6116	0.6147	0.6142	0.6154

Table 3.3: Mean MSE values for 5-fold cross-validation using a regression neural network model with varying number of training epochs.

3.2 Temporally Consistent Models

In this section we consider a reinforcement learning (abbreviated as RL) approach, where the underlying MDP is exploited and the sequential nature of the data is explicitly used in the learning process. More specifically, the action value function has the recurrent property:

$$Q(s,a) = r(s,a) + \sum_{s',a' \in S \times A} T(s,a,s',a')\, Q(s',a') \tag{3.2}$$

where $T(s,a)$ denotes the probability that the state-action pair (s',a') follows (s,a) in the MDP. Equation (3.2) includes assumptions made in section 2.2.

3.2.1 Temporal Difference Learning

One of the central ideas in reinforcement learning is the notion of temporal difference (TD) learning [23], which performs updates to action values after each state transition without waiting until the end of the episode to obtain rewards. This is accomplished by approximating the true action values through bootstrapping of the current estimate. In its simplest form, the action value for a state-action pair (s,a) is updated by

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r(s,a) + \gamma Q(s',a') - Q(s,a)),$$

where the pair (s',a') follows (s,a) in the current episode and the reward $r(s,a)$ is given for the transition. The target $r(s,a)+\gamma Q(s',a')$ is a one-step bootstrapped estimate of the true action value and the current estimate $Q(s,a)$ is updated towards that value by learning rate α. We use this concept in our RL Neural Network approach to computing action values.

3.2.2 The RL Neural Network

Our neural network architecture is based on the approach in [13] that proved successful for deep learning using NHL hockey data. After some experiments with the number of nodes, we adopt the following structure for our reinforcement learning based neural network:

- First layer containing 256 LSTM nodes for sequence processing,
- Hidden dense layer of 1000 nodes with RelU activation,
- Output layer.

The input sequence contains vectors of the current state-action pair and a history of up to 9 previous actions in the same rally. As in [13], a dynamic trace length parameter is passed to the network to ensure the correct history length is taken into account. The network outputs a single number, which is the estimated expected reward value of the given (partial) state sequence. This differs from the implementation in [13], where separate values are output for 'home score', 'away score' and 'no score' events. Because of the nature of scoring in volleyball, every rally ends with one of the two teams scoring and only a single number is sufficient to describe all outcomes.

Denoting the weight vector of the neural network by $\mathbf{w}$ and the corresponding approximated action value function as $Q(s, a; \mathbf{w})$, we apply one-step temporal difference bootstrapping [23] to approximate the true action values $Q(s, a)$ and we minimize the loss function:

$$J(\mathbf{w}) = \mathbb{E}[(r(s,a) + \gamma Q(s', a'; \mathbf{w}) - Q(s, a; \mathbf{w}))^2],$$

where the state-action pair s, a is followed by the pair s', a', producing a reward of $r(s, a)$. For terminal states s' where no successor state exists, expected future reward becomes trivial - for those states we set $Q(s, a; \mathbf{w}) = r(s, a)$.
The training process uses gradient backpropagation by the Adam algorithm [12] as implemented in the Tensorflow Python package.

We monitor convergence of the action values through the 1-norm of successive Q function iterates, i.e. denoting by $\mathbf{Q}^i$ the vector of all action values produced by the i-th iteration through the training data, we consider $\|\mathbf{Q}^i - \mathbf{Q}^{i-1}\|_1$ as a measure of convergence and stop when its value falls below a predefined threshold.

3.3 Mimic Tree

Mimic learning attempts to combine the predictive power of a deep neural network with the transparency of a decision tree model. In [22], this approach is employed to model action values learned from hockey data in an interpretable way. We follow the same idea using volleyball action values, but we stop short of using linear model trees and use a regression tree instead.

Specifically, we train a regression tree model with the same input as our regular regression tree, but instead of targeting the binary outcome of home or away rally win, we use the output of the RL neural network (i.e. the computed action value) as the regression target.

Just as before, we use 5-fold cross-validation to determine the optimal maximal depth of the tree. Table 3.4 shows an excerpt of the cross-validation mean squared error values for various tree depths. We adopt a maximal depth of 20 as the optimal choice and note that the mimic tree appears to achieve a good fit to the action values with MSE value about 2.5×10^{-3}.

max_depth	16	18	20	22	24
mean cross-validation MSE	0.00261	0.00257	0.00250	0.00257	0.00259

Table 3.4: Mean MSE values for 5-fold cross-validation using a regression random forest model with varying maximal depth.

The tree is much too large to include in its entirety, but we can examine some of the top level splits shown in Figure 3.2. Comparison to Figure 3.1 reveals a great deal of similarity, but there is a single node on the lowest level shown that is different between the two models, indicating that the mimic tree is close to the original regression tree, but not identical.

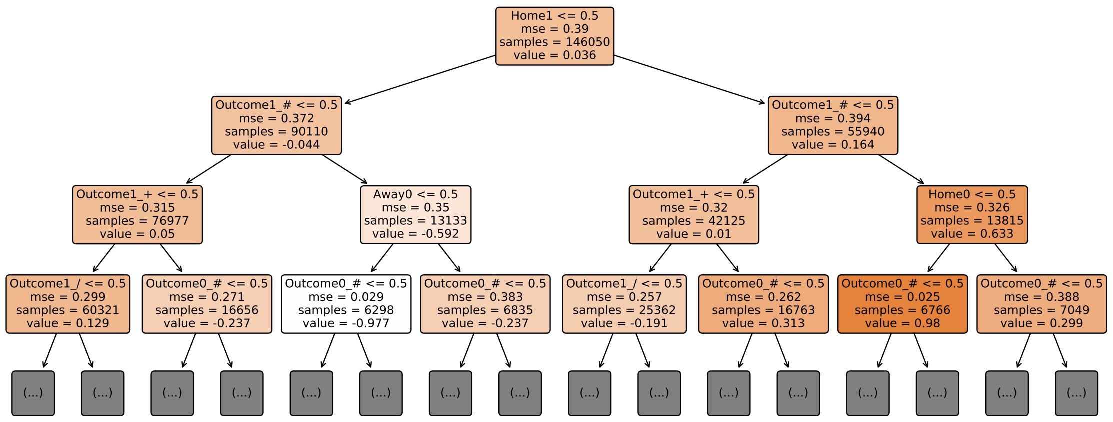

Figure 3.2: First levels of the mimic regression tree structure.

3.4 Summary

To summarize, Table 3.5 gives a list of all models considered in this project and whether or not they enforce temporal consistency stemming from the sequential nature of the data and the underlying MDP. Note that, while the mimic tree does not enforce temporal consistency in its own right, it learns from the output of the RL Neural Network, indirectly incorporating some temporal information.

Model	**Temporal Consistency**
Decision Tree	No
Random Forest	No
Regression Neural Network	No
Reinforcement Learning Neural Network	Yes
Mimic Tree	No*

Table 3.5: Summary of models used to compute action values.

Chapter 4

Model Evaluation

4.1 Mean Squared Error

We evaluate the mean squared error (3.1) between observed rewards and computed action values for all our models over the entire dataset. Table 4.1 shows a summary of MSE error values for the various models as discussed in the previous chapter. The best regression fit was achieved by using a random forest model, followed by the reinforcement learning neural network. The decision tree model produced the highest mean squared error. These results give the impression of the random forest as the superior model - however, as discussed below, temporal consistency is sacrificed as a result.

Model	Dec. Tree	Rand. Forest	Reg. Neural Net.	RL Neural Net.	Mimic Tree
MSE	0.609	0.592	0.606	0.603	0.604

Table 4.1: Mean squared error values between Q function estimates and eventual reward values for various models.

4.2 Temporal Consistency

The action value function over a Markov decision process satisfies a recurrence relationship in terms of the current state-action pair (s, a) and possible future state-action pairs (S_{t+1}, A_{t+1}):

$$\begin{aligned} Q_\pi(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1}) \,|\, S_t = s, A_t = a\right] && (4.1) \\ &= R_{t+1} + \sum_{s' \in S, a' \in A} \gamma P(S_{t+1} = s', A_{t+1} = a' \,|\, S_t = s, A_t = a) Q_\pi(s', a'). && (4.2) \end{aligned}$$

This property is known as the Bellman equation [23] and it gives us the opportunity to evaluate action value estimates in an aspect that is different from the regression error considered in the previous section. If our approximation is indeed an action value function, we expect it to (at least approximately) agree with the Bellman equation in addition to minimizing the loss function against the training data. We use this notion to measure the temporal consistency of our models.

In our context we use a discount factor of $\gamma = 1$ and rewards only occur at episode-ending states. If s is not a terminal state, $R_{t+1} = 0$ and the Bellman equation becomes

$$Q(s,a) = \sum_{s',a'} P(S_{t+1} = s', A_{t+1} = a' \mid S_t = s, A_t = a) Q(s', a'). \tag{4.3}$$

Assuming we've computed an action value function estimate for all the states in our dataset, we can proceed to verify this equality using the data, since the transition probability P can be estimated with respect to maximum likelihood (see e.g. [23] Chapter 6) by counting occurrences of state transitions in the dataset as

$$P(S_{t+1} = s', A_{t+1} = a' | S_t = s, A_t = a) = \frac{c(S_{t+1} = s', A_{t+1} = a', S_t = s, A_t = a)}{c(S_t = s, A_t = a)},$$

where we denote by $c(x)$ the number of times x appears in the dataset.

Since the state space is large, we wish to avoid examining temporal consistency with respect to a single state due to potential data sparsity effects. We are instead interested in a set of state-action pairs X, summing all the corresponding action values as they appear in the dataset, namely the quantity:

$$\sum_{s,a \in X} c(s,a) Q(s,a).$$

Using (4.3), we obtain the right hand side:

$$\sum_{s,a \in X} c(s,a) Q(s,a) = \sum_{s,a \in X} c(s,a) \sum_{s',a' \in S \times A} \frac{c(s',a',s,a)}{c(s,a)} Q(s',a'), \tag{4.4}$$

which simplifies to

$$\sum_{s,a \in X} c(s,a) Q(s,a) = \sum_{s,a \in X} \sum_{s',a' \in S \times A} c(s',a',s,a) Q(s',a'). \tag{4.5}$$

This final form is the one we use to compute the degree of temporal inconsistency, since both the left and right hand sides can be computed in a simple loop over the dataset sequences. More specifically, we are interested in how much our action value estimates violate (4.5), i.e. the quantity

$$e_X = | \sum_{s,a \in X} c(s,a) Q(s,a) - \sum_{s,a \in X} \sum_{s',a' \in S \times A} c(s',a',s,a) Q(s',a') |. \tag{4.6}$$

Model	Serve	Receive	Attack	Set	Block	Dig
Decision Tree	0.0158	0.0151	0.0376	0.0074	0.0641	0.0662
Random Forest	0.0172	0.0183	0.0075	0.0066	0.0025	0.0148
Regr. Neural Network	0.0621	0.0162	0.0616	0.0401	0.0322	0.0508
RL Neural Network	0.0001	0.0005	0.0003	0.0019	0.0008	0.0002
Mimic Tree	0.0001	0.0001	0.0076	0.0018	0.0125	0.0009

Table 4.2: Mean degree of temporal inconsistency e_X for various sets of non-terminal states X. For each volleyball action type (columns), the set X includes actions performed by the home team with a non-terminal outcome.

We choose the state-action subset X to include actions of a single type with non-terminal outcomes (for example, we include all serve actions that do not result in an immediate error or point won). We compute the quantity e_X averaged over the number of data samples for various action types and compare results across different action value function approximations.

Looking at the e_X values shown in Table 4.2, the reinforcement learning neural network exhibits overall highest temporal consistency and the single decision tree is the worst of our models in this respect. Since the RL neural network training incorporates the temporal difference update, explicitly correcting toward the TD target

$$R(s, a) + \gamma Q(s', a') - Q(s, a)$$

where the state-action pair (s, a) is followed by (s', a') in the data, it is not surprising that temporal consistency is best maintained using this model. The Decision tree and other regression models are trained under the assumption of independently sampled data and do not incorporate any information about the sequential nature of the volleyball dataset. Their e_X values are accordingly much larger. In particular, the regression-based neural network exhibits high e_X values overall. In the case of the random forest, that drawback of low temporal agreement could outweigh its lower mean squared error compared to the RL neural network when used to compute action values.

The mimic tree model is by design a middle ground between regression models and reinforcement learning, and its e_X values seem to align with that premise. They are close to the RL neural network's in most cases, but significantly higher for attack and block actions.

4.3 Action Value Comparison

We compare action values computed by the different models for the home and away team, including all possible action types and outcomes. Tables 4.3 and 4.4 contain mean values of $Q(s, a)$ over all state-action pairs where the action type and outcome for a are fixed, separately for the home team and away team, respectively.

Home Action Values

Action		Dec. Tree	Rand. For.	Regr. NN	RL NN	Mim. Tree	Data
S	=	-1	-1	-1	-0.998	-0.998	-1
	-	**-0.293**	-0.291	-0.262	-0.29	-0.29	-0.293
	!	0.103	0.011	-0.066	**-0.101**	**-0.101**	-0.106
	+	0.103	**0.246**	0.298	0.225	0.225	0.238
	#	**1**	**1**	0.985	0.997	0.998	1
R	=	**-1**	**-1**	**-1**	-0.998	-0.998	-1
	-	**-0.167**	**-0.167**	-0.118	-0.157	-0.157	-0.166
	!	0.104	0.039	0.192	**0.11**	0.108	0.118
	+	0.287	**0.323**	0.424	0.325	0.325	0.321
	#	0.437	0.427	0.409	**0.446**	**0.446**	0.446
A	=	-0.991	-0.987	**-0.997**	-0.994	-0.993	-1
	/	-0.984	-0.968	-0.984	**-0.993**	-0.986	-1
	-	0.02	-0.052	-0.021	**-0.072**	**-0.072**	-0.071
	!	0.107	0.082	0.190	**0.122**	0.117	0.127
	+	0.302	0.284	0.439	0.372	**0.368**	0.362
	#	0.947	0.99	0.988	**0.994**	**0.994**	1
B	=	-0.87	-0.969	**-1**	-0.99	-0.961	-1
	-	**-1**	**-1**	-0.986	-0.993	-0.993	-1
	!	0.103	0.209	**0.051**	0.063	0.071	0.052
	+	0.218	0.154	0.246	0.146	**0.145**	0.142
	#	0.999	**1**	0.998	0.993	0.993	1
D	=	**-1**	**-1**	**-1**	-0.995	-0.995	-1
	!	-0.289	-0.262	-0.187	**-0.278**	-0.275	-0.282
	#	0.217	**0.152**	0.211	0.155	0.155	0.147
E	=	-0.941	-0.986	**-1**	-0.994	-0.989	-1
	+	0.364	0.375	0.432	**0.371**	**0.371**	0.372

Table 4.3: Comparison of home action values computed by different models and directly from data. Entries closest to data values in bold.

Note that there is no inherent order in the outcome symbols and the values for the different outcomes are entirely learned by the models. The fact that action values increase for outcomes generally considered more favourable is a good sanity check and provides validation for our models. We also compute mean eventual rewards directly from the data for comparison. Overall the model values seem to agree well with the data averages with a few exceptions. For example, home values for serve outcomes '+' and '!' computed by the decision tree model are identical and seem to be somewhere in between the outcome frequencies observed in the data set, suggesting that a split failed to occur and samples were grouped together in the same leaf node. The same can be observed for the '!' and '+' outcomes of the away receive action. The random forest model corrects this to some extent, but still outputs a positive value for the '!' outcome, contrary to what the data suggests.

Away Action Values

Action		Dec. Tree	Rand. For.	Regr. NN	RL NN	Mim. Tree	Data
S	=	**1**	**1**	0.999	0.999	0.999	1
	-	**0.362**	0.351	0.342	0.364	0.364	0.362
	!	0.107	**0.11**	0.076	**0.11**	**0.11**	0.118
	+	-0.166	**-0.167**	-0.170	-0.157	-0.158	-0.167
	#	**-1**	**-1**	-0.992	-0.998	-0.997	-1
R	=	**1**	**1**	**1**	0.998	0.998	1
	-	**0.239**	**0.239**	0.184	0.225	0.225	0.238
	!	-0.185	-0.097	-0.135	**-0.101**	-0.1	-0.106
	+	-0.185	-0.251	-0.266	**-0.252**	**-0.252**	-0.256
	#	**-0.369**	-0.368	-0.377	-0.373	-0.373	-0.369
A	=	0.986	0.987	0.991	**0.994**	0.986	1
	/	0.927	0.965	0.987	0.993	**0.99**	1
	-	0.114	0.151	0.150	**0.155**	0.156	0.153
	!	0.091	0.103	-0.025	**0.067**	0.072	0.051
	+	-0.204	-0.206	-0.264	-0.287	**-0.275**	-0.277
	#	**-0.998**	-0.992	-0.989	-0.994	-0.993	-1
B	=	0.951	0.969	0.996	0.992	0.991	1
	-	0.999	**1**	0.994	0.994	0.994	1
	!	0.116	0.109	0.142	**0.123**	0.119	0.127
	+	-0.052	-0.07	-0.063	**-0.072**	-0.071	-0.075
	#	**-1**	-0.999	-0.998	-0.993	-0.993	-1
D	=	**1**	**1**	**1**	0.995	0.995	1
	!	**0.37**	0.363	0.326	0.377	0.378	0.371
	#	-0.069	-0.071	-0.073	**-0.064**	**-0.064**	-0.062
E	=	0.971	0.987	0.972	**0.993**	0.989	1
	+	**-0.281**	-0.287	-0.311	-0.283	-0.284	-0.281

Table 4.4: Comparison of away action values computed by different models. See also Table 4.3.

4.4 Rally Win Prediction Accuracy

The action value function has a range of $[-1, 1]$ where positive values correspond to higher expected reward for the home team and negative values correspond to higher expected reward for the away team. Denoting by (s, a) the current state-action pair, action values can be converted to rally-winning probabilities by using the formula:

$$P_H(s,a) = \frac{Q(s,a)+1}{2} \tag{4.7}$$

for the home team and

$$P_A(s,a) = \frac{1-Q(s,a)}{2} \tag{4.8}$$

for the away team. These relationships can easily be derived by noting that one (and only one) of the two teams always wins the rally, hence

$$P_H(s,a) + P_A(s,a) = 1. \tag{4.9}$$

Furthermore, the reward function (2.1) values depend precisely on those two events and therefore

$$Q(s,a) = P_H(s,a) - P_A(s,a).$$

Finally, substitution of (4.9) into this equation yields the formulas (4.7). These probabilities can be used to predict the eventual winner of the rally and evaluate accuracy differences between our models. Table 4.5 shows prediction accuracies as well as cross entropy values over the entire dataset for the different models. Cross entropy for a single prediction is computed using the formula

$$-\log P(y|P_H) = -(y \log P_H + (1-y)\log(1-P_H)), \tag{4.10}$$

where $y \in \{0, 1\}$ is the indicator variable for the home team eventually winning the current rally and P_H is the according predicted home win probability. Dependency of y and P_H on s and a omitted here for brevity.

We consider some baselines in this context, against which to compare model accuracy. The home team has a slight advantage, so always predicting home to win the rally is a possible baseline. Furthermore, the receiving team normally scores more often than the serving team, so always predicting the receiving team to win the rally is another viable baseline. We find the following baseline accuracies are obtained:

- Always predicting home team wins the rally: 0.5174,
- Always predicting receiving team wins the rally: 0.5807.

Model	**Accuracy Score**	**Log Loss**
Decision Tree	0.7493	0.4345
Random Forest	0.8092	0.4017
Regr. Neural Network	0.7612	0.4203
RL Neural Network	0.7551	0.4283
Mimic Tree	0.7529	0.4301

Table 4.5: Comparison of models for rally win prediction.

From Table 4.5 accuracy values, clearly all models perform well above the considered baselines, but the random forest model outperforms the rest with a prediction accuracy of just over 80%, which suggests it as the best choice in the win prediction context. Despite the prediction accuracy advantage however, we will find that random forest is not the best choice when it comes to preserving temporal consistency inherited from the Markov decision process.

To further analyse the win prediction capabilities of our models, we consider predictions depending on how many actions remain in the rally. Recall that the model input is a partial sequence of actions leading up to the current state and action, which means model performance likely depends significantly on how much play remains in the current rally. Figure 4.1 shows a graph of this dependency. When the observed action is the last one in the rally (i.e. zero actions remain), win prediction accuracy is perfect for all considered models. It remains significantly better than guessing for up to 4 actions away from the rally terminating, but for distances of 5 or higher, accuracy is significantly impaired and only the random forest model continues to give predictions with around 60% accuracy.

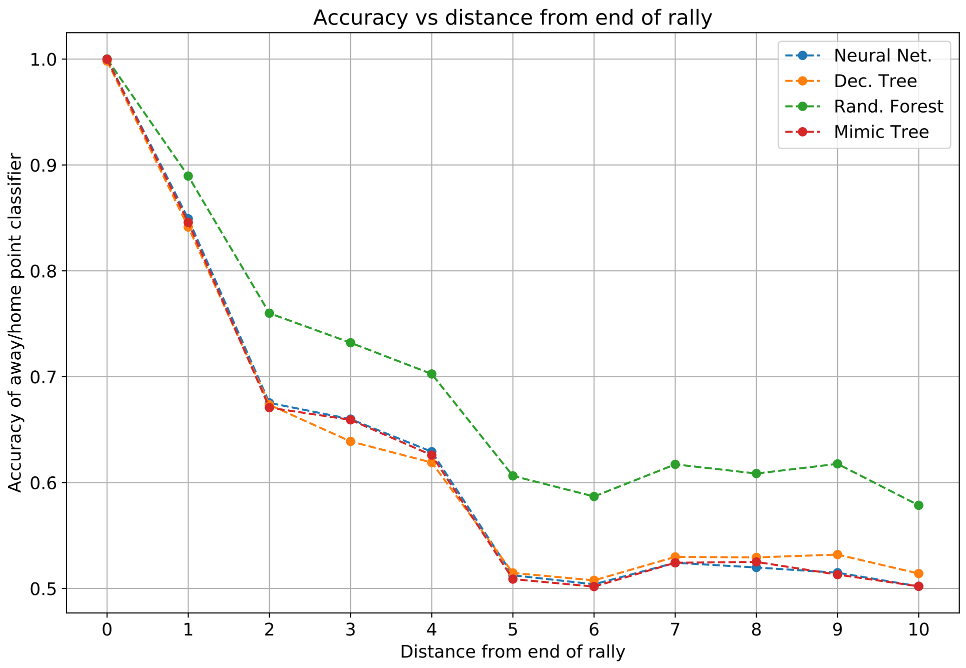

Figure 4.1: Model win prediction accuracy plotted against distance from the end of rally in terms of actions remaining.

Chapter 5

Action Value Use Examples

5.1 Serve Risk Assessment

The serve action is one of the most crucial actions types in volleyball as its outcome has a large impact on the odds of either team winning the rally. It is also the only action in volleyball where the player executing it has complete control independently of other players' actions. Optimizing the risk/reward profile for serving is therefore an important problem coaches and players frequently face [4]. In this section we examine serve action value depending on the score context, namely how close to ending the current set is in terms of score. We take $\max(\text{home score}, \text{away score})$ to be the progress measure of the current set and compare the action values of serves resulting in the '-' and '!' outcomes. The '-' outcome corresponds to the opponent team receiving the ball well enough to have all attack options available - this is the expected outcome of a serve executed with low risk. The '!' serve outcome means the opponent reception landed away from the net, removing the quick middle attack option and making opponent offence more predictable - this is generally the outcome sought when executing a serve with some (controlled) risk.

Figure 5.1 shows action values converted to point scoring probability (see equation (4.7)) for the serve outcomes above depending on the current set score. For simplicity, the set score is bracketed into intervals of 0-5, 6-10, 11-15, 16-20 and 21-25. The x axis labels correspond to the upper end of each interval. From the graph we can observe a decreasing trend in the '-' outcome value and an increasing trend in the '!' outcome value. This would suggest a strategy using lower risk serving in the beginning of a set and gradually increasing the risk towards the end of the set, since '!' outcomes become more valuable as the set progresses.

This is of course not a complete risk assessment, since we did not include any information regarding serve errors, but it can serve as an example of potential use of the action value function.

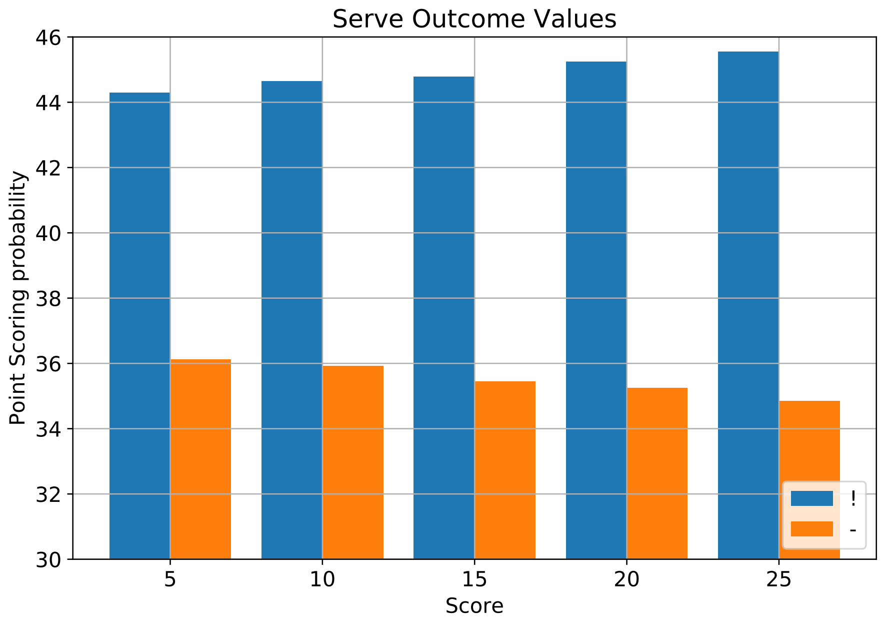

Figure 5.1: Home serve values converted to point scoring probability for the '-' and '!' serve outcomes depending on current set score.

5.2 Serve Receive Location Analysis

Since the previous section concerned analysis of actions through time, we now consider an example related to spatial location. We focus on the action of receiving an opponent's serve and examine the associated end location of the ball, i.e. where the ball ended up after the receive action was performed. This is a good validation example because there is ample domain knowledge on the ideal target location that we can use to compare against computed action values.

Figure 5.2 depicts home win probabilities by end location for the home team receiving the serve (bird's eye view of half the court is depicted with the net on the top side of the image). These can easily be computed from action values using equation (4.7). Locations in the half-court furthest from the net are not included due to the fact they are not present in detail in the data.

The location with highest winning probability coincides perfectly with the usual notion of a perfect reception, i.e. close to the net and shifted slightly to the right. Values generally decrease as distance from the net increases and as the location is shifted away from the ideal vertical line. This confirms our action values agree with domain knowledge and provides validation to our model.

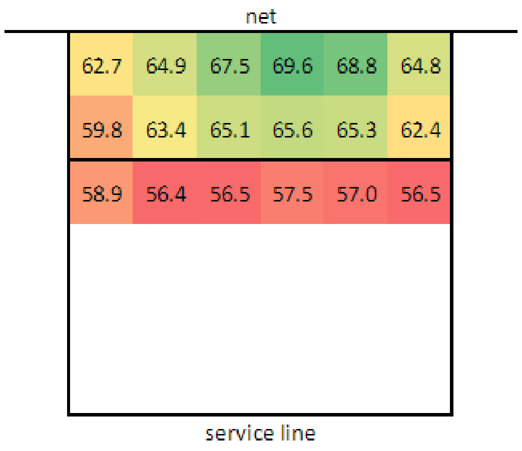

Figure 5.2: Home win probabilities (percent) for home serve receive actions by end location.

5.3 Action Impact

One of the motivations of computing the action-value function for sports data is the ability to numerically rank teams and players according to their action values. To achieve this, we need to consider that the context of an action performed by a player influences the quality of their actions. Namely, scoring a point from a favorable situation (eg. a successful attack following a perfect pass) should be treated differently than scoring in a situation that is deemed difficult. This leads us to the notion of action impact.

As discussed in [19], there are several options of how we could choose to value actions. We adopt the difference of consecutive action values as a measure of impact. Namely, for a transition from state s_{t-1} to s_t, with actions a_{t-1} and a_t we define the impact of (s_t, a_t) as:

$$\text{impact}(s_t, a_t) = Q(s_t, a_t) - Q(s_{t-1}, a_{t-1}). \tag{5.1}$$

This allows us to capture to some degree how the flow of the game changed when action a_t was performed. This version of valuing player actions was successfully used in the hockey context in [13] for player evaluations.

Note that since action impact relates to the values of both current and previous states, we need to find a treatment for the first action of any episode (in volleyball this is always a serve action), where s_{t-1} and a_{t-1} are not defined. The most obvious solution is to use an initial state-action value of 0, but since the serving team is generally at a disadvantage in terms of scoring probability, this approach gives serve actions a negative value on average and renders serve values disproportionate when compared to other action types. We therefore proceed as follows. Define Q_{SH} to be the mean value of all home serve actions and Q_{SA} to be the mean value of all away serve actions. We then use these values to compute action impacts as:

$$\text{impact}(s_t, a_t) = \begin{cases} Q(s_t, a_t) - Q_{SH} & \text{if } a_t \text{ is home serve,} \\ Q(s_t, a_t) - Q_{SA} & \text{if } a_t \text{ is away serve,} \\ Q(s_t, a_t) - Q(s_{t-1}, a_{t-1}) & \text{otherwise.} \end{cases} \tag{5.2}$$

The intuition behind this approach is to have the serve impact measure the difference between the current serve and the average serve. We also know the home team tends to have a slight advantage, which is the reason for separate averaging of home and away serves. The two means, Q_H and Q_A do in fact differ in absolute value by about 0.06 in favour of the home team.

Firstly, we consider team collective action impact and investigate how well it aligns with team performance in the Canada West volleyball competition in the last three seasons. To this end, we compute the mean impact of all actions performed by players of a particular team using equation (5.2). These values are collected in Table 5.1 for all teams participating in the Canada West men's volleyball competition (with the exception of Regina, who ceased participation after the 2017/2018 season and is not well represented in the dataset). We compare action impact values to mean ranking of the teams following regular season across the 2017/2018, 2018/2019 and 2019/2020 seasons, also included in Table 5.1. Teams in the table are sorted by mean action impact. Good alignment against team ranking can be seen from inspection of the table, but a plot is also included in Figure 5.3 for a more visual representation. With the exception of three teams, an increase in mean action impact corresponds to higher ranking, which is desirable behaviour for our model, as higher action values lead to more wins. We also compute the Pearson correlation coefficient between mean action impacts and mean rankings and obtain a value of -0.854, confirming a high correlation and a near linear relationship.

We note here again that the dataset contains all UBC games, but not all games among other Canada West teams, meaning that the amount of data for those teams is significantly lower than for UBC. The bias introduced by this could explain the outliers in Figure 5.3.

We can compute mean action impacts for individual players in the same fashion as we did for teams above, which enables us to track and rank player performance. Table 5.2 shows the top 10 players ranked by mean action impact in Canada West. The caveat, compared to team ranking, is that there are fewer obvious ways to check the validity of these values. In professional leagues such as the NHL, player salary has been used for this purpose, e.g. in [21], but there is no such information for university athletes. Regardless, we still attempt some discussion. Notably, most of the players listed come from the top ranked teams - the best 4 teams in Canada West (by mean ranking in the last 3 seasons) are represented 8 out of 10 times, which can be taken as a good sign.

Team	Mean Action Impact	R20	R19	R18	Mean Ranking
Trinity Western	0.0220	1	2	1	1.33
Alberta	0.0175	2	3	3	2.67
UBC	0.0124	3	7	2	4.00
Brandon	0.0077	4	1	4	3.00
Calgary	0.0073	5	8	6	6.33
Winnipeg	0.0053	6	10	5	7.00
Mount Royal	-0.0053	8	4	10	7.33
Manitoba	-0.0067	10	9	7	8.67
MacEwan	-0.0128	12	11	12	11.67
Thompson Rivers	-0.0144	9	6	8	7.67
Saskatchewan	-0.0210	7	5	9	7.00
UBCO	-0.0235	11	12	11	11.33

Table 5.1: Canada West teams sorted by mean action impact. Columns R20, R19, R18 are team rankings after regular season in 2020, 2019 and 2018, respectively and Mean Ranking is the mean of those three columns.

Player Name	Team	Mean Action Impact
Elliot Viles	Brandon	0.0838
Eric Loeppky	Trinity Western	0.0796
Jackson Kennedy	Alberta	0.0580
George Hobern	Alberta	0.0566
Hamish Hazelden	Calgary	0.0539
Joel Regher	UBC	0.0509
Daniel Thiessen	Winnipeg	0.0504
Gerard Murray	UBC	0.0471
Jordan Canham	Alberta	0.0459
Arran Chambers	Alberta	0.0451

Table 5.2: Top 10 individual players in Canada West by mean action impact as given in equation (5.2).

On a more individual level, player awards and recognition can provide further validity to our ranking. Elliot Viles, who ranks first in Table 5.2, was named a conference all-star in his rookie season and both Canada West as well as U SPORTS National player of the year in the 2018/2019 season. He also represented his home country Australia in the prestigious FIVB World League competition. Eric Loeppky, who follows closely in the mean impact ranking, also received the title of U SPORTS player of the year in 2020 as well as multiple other awards and has played on the Canadian senior national team, which is rare for players still in their university years. Jackson Kennedy and Daniel Thiessen have furthermore been selected as Canada West all-star team members alongside Loeppky and Viles. Prominent players like that featuring as leaders in impact ranking is further validation of our action impact model.

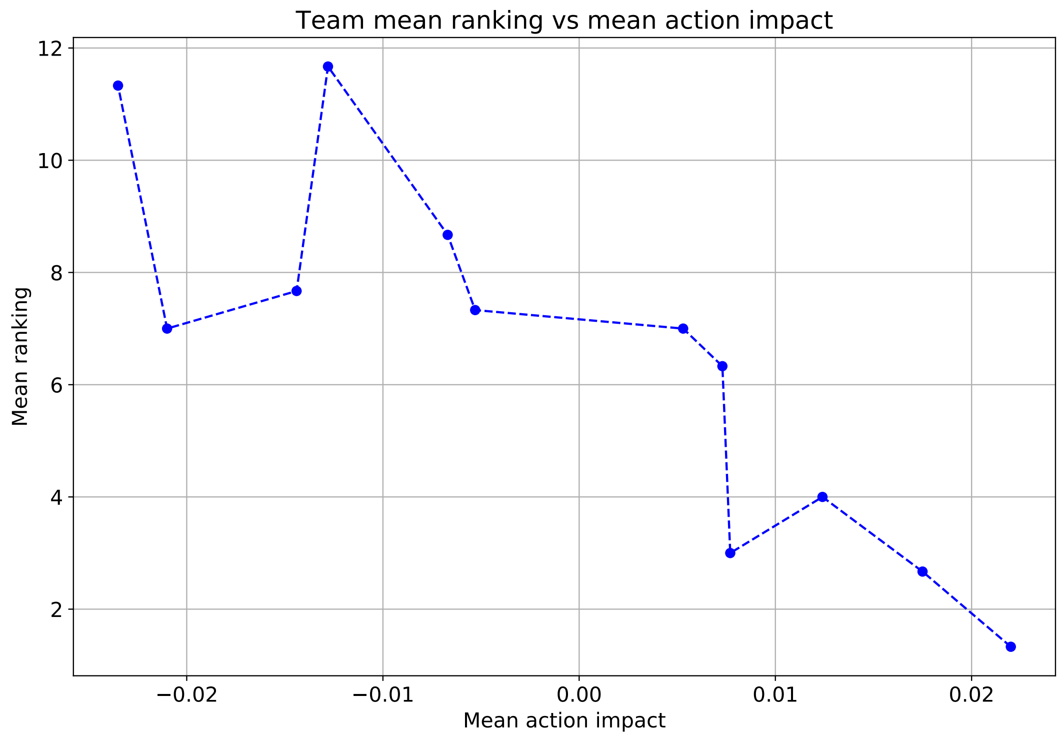

Figure 5.3: Team mean ranking plotted against mean action impact for Canada West participating teams, see Table 5.1 for values shown in this graph.

Chapter 6

Conclusion and Future Work

This project explores learning action values from volleyball data using two fundamentally different approaches. In the first approach, regression models are used to estimate the eventual reward (winning the rally) using partial action sequences and match context under an independence assumption. In the second approach, deep reinforcement learning is used, explicitly incorporating the sequential nature of the data into the learning process.

In terms of mean squared error, a common measure used in regression, the random forest regression model achieved the best fit out of all experiments, followed by the RL neural network in second place. However, when considering temporal consistency, the reinforcement learning approach produced much more consistent action values. This suggests that the random forest might be of better use in a strictly predictive context, but since action values can be used in a diverse range of sports analyses (a sample of which we include in Chapter 5), we believe the temporal consistency of deep reinforcement learning makes it the best choice.

As an additional experiment, we include a tree regression model based on mimic learning. We find that it can achieve good fidelity to the RL neural network's outputs (see Table 3.4), while providing insight into their interpretation through its tree structure.

In Chapter 5, a number of example analyses using action values are carried out using the values computed by the RL neural network. Comparing those against domain knowledge we find a high degree of agreement, which adds validity to our model.

In terms of future work, this project examines only a small subset of all available machine learning models, so there are numerous possible extensions:

- Additional ensemble methods for regression could be tested, e.g. gradient boosted trees.
- When performing mimic learning, we restrict ourselves to a regression tree, but a linear model tree could be used such as in [22].
- Finally, there are numerous architecture choices in neural network design that could be considered, such as convolutional neural networks.

Bibliography

[1] Jim Albert, Mark E Glickman, Tim B Swartz, and Ruud H Koning. *Handbook of statistical methods and analyses in sports.* CRC Press, 2017.

[2] Laios Alexandros, Kountouris Panagiotis, and Kyprianou Miltiades. The existence of home advantage in volleyball. *International Journal of Performance Analysis in Sport*, 12(2):272–281, 2012.

[3] Christopher M Bishop. *Pattern recognition and machine learning.* springer, 2006.

[4] Tristan Burton and Scott Powers. A linear model for estimating optimal service error fraction in volleyball. *Journal of Quantitative Analysis in Sports*, 11(2):117–129, 2015.

[5] Dan Cervone, Alexander D'Amour, Luke Bornn, and Kirk Goldsberry. Pointwise: Predicting points and valuing decisions in real time with nba optical tracking data. In *Proceedings of the 8th MIT Sloan Sports Analytics Conference, Boston, MA, USA*, volume 28, page 3, 2014.

[6] João Gustavo Claudino, Daniel de Oliveira Capanema, Thiago Vieira de Souza, Julio Cerca Serrão, Adriano C Machado Pereira, and George P Nassis. Current approaches to the use of artificial intelligence for injury risk assessment and performance prediction in team sports: a systematic review. *Sports medicine-open*, 5(1):28, 2019.

[7] Tom Decroos, Lotte Bransen, Jan Van Haaren, and Jesse Davis. Actions speak louder than goals: Valuing player actions in soccer. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1851–1861, 2019.

[8] Lindsay W Florence, Gilbert W Fellingham, Pat R Vehrs, and Nina P Mortensen. Skill evaluation in women's volleyball. *Journal of Quantitative Analysis in Sports*, 4(2), 2008.

[9] Moustafa Ibrahim. *Deep models for multi-person activity understanding.* PhD thesis, Applied Sciences: School of Computing Science, 2018.

[10] Edward H Kaplan, Kevin Mongeon, and John T Ryan. A markov model for hockey: manpower differential and win probability added. *INFOR: Information Systems and Operational Research*, 52(2):39–50, 2014.

[11] Thomas Kautz, Benjamin H Groh, Julius Hannink, Ulf Jensen, Holger Strubberg, and Bjoern M Eskofier. Activity recognition in beach volleyball using a deep convolutional neural network. *Data Mining and Knowledge Discovery*, 31(6):1678–1705, 2017.

[12] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[13] Guiliang Liu and Oliver Schulte. Deep reinforcement learning in ice hockey for context-aware player evaluation. *arXiv preprint arXiv:1805.11088*, 2018.

[14] Dennis Ljung, Niklas Carlsson, and Patrick Lambrix. Player pairs valuation in ice hockey. In Ulf Brefeld, Jesse Davis, Jan Van Haaren, and Albrecht Zimmermann, editors, *Machine Learning and Data Mining for Sports Analytics*, pages 82–92, Cham, 2019. Springer International Publishing.

[15] Michelle A Miskin, Gilbert W Fellingham, and Lindsay W Florence. Skill importance in women's volleyball. *Journal of Quantitative Analysis in Sports*, 6(2), 2010.

[16] Luca Pappalardo and Paolo Cintia. Quantifying the relation between performance and success in soccer. *Advances in Complex Systems*, 21(03n04):1750014, 2018.

[17] Luca Pappalardo, Paolo Cintia, Paolo Ferragina, Emanuele Massucco, Dino Pedreschi, and Fosca Giannotti. Playerank: data-driven performance evaluation and player ranking in soccer via a machine learning approach. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(5):1–27, 2019.

[18] Stephen Pettigrew. Assessing the offensive productivity of nhl players using in-game win probabilities. In *9th annual MIT sloan sports analytics conference*, volume 2, page 8, 2015.

[19] Kurt Douglas Routley. A markov game model for valuing player actions in ice hockey. Master's thesis, Applied Sciences:, 2015.

[20] Oliver Schulte, Mahmoud Khademi, Sajjad Gholami, Zeyu Zhao, Mehrsan Javan, and Philippe Desaulniers. A markov game model for valuing actions, locations, and team performance in ice hockey. *Data Mining and Knowledge Discovery*, 31(6):1735–1757, 2017.

[21] Oliver Schulte, Zeyu Zhao, Mehrsan Javan, and Philippe Desaulniers. Apples-to-apples: Clustering and ranking nhl players using location information and scoring impact. In *Proceedings of the MIT Sloan Sports Analytics Conference*, 2017.

[22] Xiangyu Sun, Jack Davis, Oliver Schulte, and Guiliang Liu. Cracking the black box: Distilling deep sports analytics. *arXiv preprint arXiv:2006.04551*, 2020.

[23] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

[24] Abdullah Erdal Tümer and Sabri Koçer. Prediction of team league's rankings in volleyball by artificial neural network method. *International Journal of Performance Analysis in Sport*, 17(3):202–211, 2017.

[25] Jan Van Haaren, Horesh Ben Shitrit, Jesse Davis, and Pascal Fua. Analyzing volleyball match data from the 2014 world championships using machine learning techniques. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 627–634, 2016.

[26] Yufan Wang, Yuliang Zhao, Rosa HM Chan, and Wen J Li. Volleyball skill assessment using a single wearable micro inertial measurement unit at wrist. *IEEE Access*, 6:13758–13765, 2018.

[27] Sebastian Wenninger, Daniel Link, and Martin Lames. Performance of machine learning models in application to beach volleyball data. *International Journal of Computer Science in Sport*, 19(1):24–36, 2020.

Appendix A

Code

The following is the core Python code used to produce results listed in this report. Non-essential exploratory analysis and plot-producing code is omitted.

A.1 Regression Models

```python
import pandas as pd
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
from sklearn.model_selection import cross_val_score

def loadDataset():
# loads volleyball data, returns input columns X as well as target column y
    vb = pd.read_csv("../data/vb_data_3_numZone.csv")

    # get input columns
    X = vb.drop(['Season', 'GameID', 'PlayerTeam', 'PlayerName', \
                 'RewardDistance', 'RewardValue'], axis=1)

    # generate dummy columns for categorical values
    print(len(X.columns), 'columns in dataframe before dummies')
    cols = [col for col in list(X.columns) if X[col].dtype == 'object']
    X = pd.get_dummies(data=X, columns = cols)
    print(len(X.columns), 'columns in dataframe after dummies')

    # get target column
    y = vb.RewardValue
    return X,y
```

```
# --- Decision Tree

def crossValRegressionTree(X, y, min_val=8, max_val=16):
# runs 5-fold cross-validation on a decision tree regression model
# with varying max_depth
    for depth in range(min_val,max_val+1):
        print('Checking max_depth =', depth)
        decTree = DecisionTreeRegressor(max_depth=depth)
        # perform 5-fold cross validation
        scores = cross_val_score(estimator=decTree, X=X, y=y, cv=5, \
                    scoring='neg_mean_squared_error')
        print('Mean cross validation MSE:', scores.mean())

def buildRegressionTree(X, y, max_depth=10):
# creates and trains a decision tree regression model
    decTree = DecisionTreeRegressor(max_depth=max_depth).fit(X,y)
    y_predicted = decTree.predict(X)
    print("Mean squared error:", metrics.mean_squared_error(y, y_predicted))
    return decTree

# --- Random Forest

def crossValRandomForest(X, y, min_val=8, max_val=16):
# runs 5-fold cross-validation on a random forest regression model
# with varying max_depth
    for depth in range(min_val, max_val+1):
        print('Checking max_depth =', i)
        randForest = RandomForestRegressor(n_estimators=10, max_depth=i)
        # Perform 5-fold cross validation
        scores = cross_val_score(estimator=randForest, X=X, y=y, cv=5, \
                                 scoring='neg_mean_squared_error')
        print('Mean cross validation MSE:', scores.mean())

def buildRandomForest(X, y, max_depth=11):
# creates and trains a random forest regression model
    randForest = RandomForestRegressor(n_estimators=10, max_depth=max_depth)
    randForest.fit(X, y)
    y_predicted = randForest.predict(X)
    print("Mean squared error:", metrics.mean_squared_error(y, y_predicted))
    return randForest

# --- Neural Network

from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor
```

```
def regressionNNModel():
    # create neural network
    model = Sequential()
    model.add(Dense(200, input_dim=204, activation='relu'))
    model.add(Dense(20, activation='relu'))
    model.add(Dense(1, activation='tanh'))
    # compile model
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model

def crossValNN(X, y, min_val=1, max_val=10):
# performs 5-fold cross validation with varying
# number of training epochs
    for epochs in range(min_val, max_val+1):
        estimator = KerasRegressor(build_fn=regressionNNModel, \
                                   epochs=epochs, batch_size=64, verbose=0)
        results = cross_val_score(estimator, X, y, cv=5)
        print("CV MSE for %d epochs: %.5f" % (epochs, results.mean()))

def buildNN(X, y, epochs=7):
# builds and trains a neural network regression model
    neural_net = KerasRegressor(build_fn=regressionNNModel, epochs=epochs,
                                batch_size=64, verbose=1)
    neural_net.fit(X,y)
    y_predicted = neural_net.predict(X)
    print("Mean squared error:", metrics.mean_squared_error(y, y_predicted))
    return neural_net
```

A.2 Reinforcement Learning Model

The following Python code is based on the NHL deep learning model from [13], but has been substantially adapted for our purposes.

A.2.1 Neural Network Class

```
import tensorflow as tf

class vball_lstm:
    def __init__(self, feature_count, lstm_cell_count, max_trace_length,
        learning_rate):
    # define a dynamic LSTM neural network

        with tf.name_scope("LSTM_layer"):
            self.rnn_input = tf.placeholder(tf.float32,
                [None, max_trace_length, feature_count],
                name="rnn_input")
            self.trace_lengths = tf.placeholder(tf.int32, [None],
                name="trace_lengths")

            self.lstm_cell = tf.contrib.rnn.LSTMCell(
                num_units=lstm_cell_count,
                state_is_tuple=True,
                initializer=tf.random_uniform_initializer(-0.05, 0.05))

            self.rnn_output, self.rnn_state = tf.nn.dynamic_rnn(
                inputs=self.rnn_input, cell=self.lstm_cell,
                sequence_length=self.trace_lengths,
                dtype=tf.float32,
                scope='bp_last_step_rnn')

            self.outputs = tf.stack(self.rnn_output)
            self.batch_size = tf.shape(self.outputs)[0]
            self.index = tf.range(0, self.batch_size) * max_trace_length
                        + (self.trace_lengths - 1)
            self.rnn_last = tf.gather(tf.reshape(self.outputs,
                        [-1, lstm_cell_count]), self.index)

        num_layer_1 = lstm_cell_count
        num_layer_2 = 1000
        with tf.name_scope("dense_layer_1"):
            self.W1 = tf.get_variable('w1_xaiver',
                [num_layer_1, num_layer_2],
                initializer=tf.contrib.layers.xavier_initializer())
            self.b1 = tf.Variable(tf.zeros([num_layer_2]), name="b_1")
            self.y1 = tf.matmul(self.rnn_last, self.W1) + self.b1
            self.activation1 = tf.nn.relu(self.y1, name='activation')

        with tf.name_scope("dense_layer_2"):
            self.W2 = tf.get_variable('w2_xaiver', [num_layer_2, 1],
                initializer=tf.contrib.layers.xavier_initializer())
            self.b2 = tf.Variable(tf.zeros([1]), name="b_2")
            self.read_out = tf.matmul(self.activation1, self.W2) + self.b2

        self.y = tf.placeholder("float", [None, 1])

        with tf.name_scope("cost"):
            self.readout_action = self.read_out
            self.cost =
                tf.reduce_mean(tf.square(self.y - self.readout_action))
            self.diff =
                tf.reduce_mean(tf.abs(self.y - self.readout_action))
        tf.summary.histogram('cost', self.cost)
```

```
        with tf.name_scope("train"):
            self.train_step =
                tf.train.AdamOptimizer(learning_rate=learning_rate)
                    .minimize(self.cost)
```

A.2.2 Main Learning Class

```
from nn_lstm import vball_lstm
from data_utils import get_next_batch, prepare_data, shuffle_data
import tensorflow as tf
import numpy as np
import time
import pickle

# config variables
FEATURE_COUNT = 41
LSTM_CELL_COUNT = 256
MAX_TRACE_LENGTH = 10
LEARNING_RATE = 1e-6
THRESHOLD = 0.0001
BATCH_SIZE = 32
GAMMA = 1

SAVE_DIR = '../output'
DATA_FILE = '../data/vb_data_3_numZone.csv'

def train_network(sess, model, state_data, reward_data, trace_lengths,
    reward_targets, episode_ids, max_iterations):

    converge_flag = False
    merge = tf.summary.merge_all()
    saver = tf.train.Saver()
    sess.run(tf.global_variables_initializer())
    start_iteration = 1

    state_data_orig = state_data.copy()
    trace_lengths_orig = trace_lengths.copy()

    restore_model = True
    if restore_model:
        checkpoint = tf.train.get_checkpoint_state(SAVE_DIR)
        if checkpoint and checkpoint.model_checkpoint_path:
            start_iteration =
                int((checkpoint.model_checkpoint_path.split("-"))[-1]) + 1
            saver.restore(sess, checkpoint.model_checkpoint_path)
            print("Successfully loaded:",
                checkpoint.model_checkpoint_path)
        else:
            print("Could not find old network weights.")

    # initial action values
    [rnn_outputs_prev, q_predictions_prev] =
        sess.run([model.outputs, model.read_out],
            feed_dict={model.trace_lengths: trace_lengths_orig,
            model.rnn_input: state_data_orig})

    for iteration in range(start_iteration, max_iterations+1):
        t = time.time()
        print('Iteration %d ...' % (iteration))
        costs = []
        sample_index = 0

        # shuffle data, preserving episodes
        state_data, reward_data, trace_lengths, episode_ids =
            shuffle_data(state_data, reward_data, trace_lengths,
                episode_ids)

        while sample_index < len(state_data):
            # prepare batch
            states0, states1, rewards, traces0, traces1, end_index =
                get_next_batch(state_data, reward_data, trace_lengths,
                    sample_index, BATCH_SIZE)

            # compute NN prediction
            [rnn_outputs_t1, q_predictions_t1] =
                sess.run([model.outputs, model.read_out],
                    feed_dict = {model.trace_lengths: traces1,
                        model.rnn_input: states1})

            q_target = np.zeros([len(rewards), 1])
            for i in range(len(rewards)):
                if rewards[i] != 0:
                    # in this case, we're at the end of an episode
                    q_target[i,0] = rewards[i]
                else:
                    q_target[i,0] = rewards[i] + GAMMA *
                        q_predictions_t1[i,0]

            # update with gradient
            [diff, read_out, cost, summary_train, _] =
                sess.run([model.diff, model.read_out, model.cost, merge,
                    model.train_step], feed_dict={model.y: q_target,
                    model.trace_lengths: traces0,
                    model.rnn_input: states0})

            sample_index = end_index + 1
            costs.append(cost)
        # end of iteration loop

        iteration_cost = np.mean(costs)
        print('... done. Time elapsed: %.2f seconds' % (time.time() - t))
        print('Iteration cost: %.3f' % (iteration_cost))

        [rnn_outputs, q_predictions] =
            sess.run([model.outputs, model.read_out],
                feed_dict={model.trace_lengths: trace_lengths_orig,
                model.rnn_input: state_data_orig})


        # compute mean change in value predictions
        avg_change = np.mean(np.abs(q_predictions - q_predictions_prev))
        print('Mean change from last iteration: %.5f' % (avg_change))
        q_predictions_prev = q_predictions

        saver.save(sess, SAVE_DIR + '/nn-iter', global_step=iteration)

        # check for convergence
        if avg_change < THRESHOLD:
            converge_flag = True

        if converge_flag:
            print('Convergence detected, exiting.')
            print('Saving action values.')
            pickle.dump(q_predictions,
                open(SAVE_DIR + '/q_values.pkl', 'wb'))
            break
    print('Done.')
```

A.3 Mimic Learning

```
import pandas as pd
import numpy as np
import pickle
from sklearn.tree import DecisionTreeRegressor
from sklearn import metrics
from sklearn.model_selection import cross_val_score

def crossValMimicTree(min_val=10, max_val=30):
```

```
# runs 5-fold cross-validation on a mimic tree regression model
    # load data
    X, _ = loadDataset()
    # load reinforcement learning neural net outputs
    act_values_raw = pickle.load(
        open('../output/q_values_RL_NN.pkl', 'rb'))
    act_values = np.asarray(
        [q_values_raw[i,0] for i in range(len(q_values_raw))])
    # clipping
    act_values[act_values > 1] = 1
    act_values[act_values < -1] = -1
    for depth in range(min_val, max_val+1):
        print('Checking max_depth =', depth)
        mimicTree = DecisionTreeRegressor(max_depth=depth)
        # Perform 5-fold cross validation
        scores = cross_val_score(estimator=mimicTree,
            X=X, y=act_values, cv=5, scoring='neg_mean_squared_error')
        print('Mean cross validation MSE:', scores.mean())

def buildMimicTree(max_depth=20):
# build and train a mimic tree regression model
    # load data
    X, _ = loadDataset()
    # load reinforcement learning neural net outputs
    act_values_raw = pickle.load(
        open('../output/q_values_new.pkl', 'rb'))
    act_values = np.asarray(
        [q_values_raw[i,0] for i in range(len(q_values_raw))])
    # clipping
    act_values[act_values > 1] = 1
    act_values[act_values < -1] = -1
    # build mimic tree and print MSE to NN action values
    mimicTree = DecisionTreeRegressor(max_depth=max_depth)
        .fit(X, act_values)
    q_mimic = mimicTree.predict(X)
    print("Mean squared error:",
        metrics.mean_squared_error(act_values, q_mimic))
    return mimicTree
```