DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING, SECOND CYCLE, 30 CREDITS

STOCKHOLM, SWEDEN 2017

Predicting trajectories of golf balls using recurrent neural networks

ANTON JANSSON

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF COMPUTER SCIENCE AND COMMUNICATION

Predicting trajectories of golf balls using recurrent neural networks

ANTON JANSSON

Master in Computer Science
Date: June 30, 2017
Supervisor: Pawel Herman
Examiner: Danica Kragic
Swedish title: Förutspå bollbanan för en golfboll med neurala nätverk
School of Computer Science and Communication

Abstract

This thesis is concerned with the problem of predicting the remaining part of the trajectory of a golf ball as it travels through the air where only the three-dimensional position of the ball is captured. The approach taken to solve this problem relied on recurrent neural networks in the form of the long short-term memory networks (LSTM). The motivation behind this choice was that this type of networks had led to state-of-the-art performance for similar problems such as predicting the trajectory of pedestrians. The results show that using LSTMs led to an average reduction of 36.6 % of the error in the predicted impact position of the ball, compared to previous methods based on numerical simulations of a physical model, when the model was evaluated on the same driving range that it was trained on. Evaluating the model on a different driving range than it was trained on leads to improvements in general, but not for all driving ranges, in particular when the ball was captured at a different frequency compared to the data that the model was trained on. This problem was solved to some extent by retraining the model with small amounts of data on the new driving range.

Sammanfattning

Detta examensarbete har studerat problemet att förutspå den fullständiga bollbanan för en golfboll när den flyger i luften där endast den tredimensionella positionen av bollen observerades. Den typ av metod som användes för att lösa problemet använde sig av recurrent neural networks, i form av long short-term memory nätverk (LSTM). Motivationen bakom detta var att denna typ av nätverk hade lett till goda resultatet för liknande problem. Resultatet visar att använda sig av LSTM nätverk leder i genomsnitt till en 36.6 % förminskning av felet i den förutspådda nedslagsplatsen för bollen jämfört mot tidigare metoder som använder sig av numeriska simuleringar av en fysikalisk modell, om modellen användes på samma golfbana som den tränades på. Att använda en modell som var tränad på en annan golfbana leder till förbättringar i allmänhet, men inte om modellen användes på en golfbana där bollen fångades in med en annan frekvens. Detta problem löstes till en viss mån genom att träna om modellen med lite data från den nya golfbanan.

Contents

Chapter 1

Introduction

In sports, the speed of the game often makes the viewing experience less enjoyable for the spectators (both live and on TV), as these people often lack the key insight into the game that the players have. Therefore, different kind of technologies can be applied to increase the viewing experience, such as showing the trajectory of the ball on TV screens. The most common approach for this is to use computer vision (via cameras) to capture the trajectory of the ball, which has been applied to sports such as tennis [27], volleyball [8], basketball [9] and baseball [11]. For the case of golf, the work that have been published have been on the underlying physics of the problem [30, 41] instead of the engineering behind creating a complete system for visualizing the trajectory. It is worth pointing out that there exist many commercial products such as launch monitors [44, 31] that computes the length of the shot, among others, depending on how the player hit the ball.

Ideally, the camera would track the ball the entire trajectory, from start to finish. However, this is not always possible, as the ball may not always be in the field of view of the camera, or can be too far away from it. Therefore, after the ball can no longer be seen, the trajectory should instead be simulated so the entire trajectory can be visualized for the viewers. In the previous mentioned sports, the size of the playing area is quite small. This meant that for the frames that the ball was not captured, the ball would not travel very far. This meant that the ball was assumed to follow a parabolic arc in the missing frames, where the parameters could easily be estimated. This is in contrast to golf, where the size of the playing area is large, with variations depending on the particular golf course where the game is played. However, this

does not mean that the motion itself must be complex, in other words depending on many different parameters. At a first glance, the motion of a golf ball traveling through the air might not be too different from a parabolic arc. This is not the case, as in the beginning of the trajectory[1] the golf ball follows an almost straight line, and then falls to the ground in a motion similar (but still different) from a parabolic arc [28], as shown in figure 1.1. This suggests that there are more forces involved than just the forces of gravity and drag (which slows the ball down), which makes the physics more complicated.

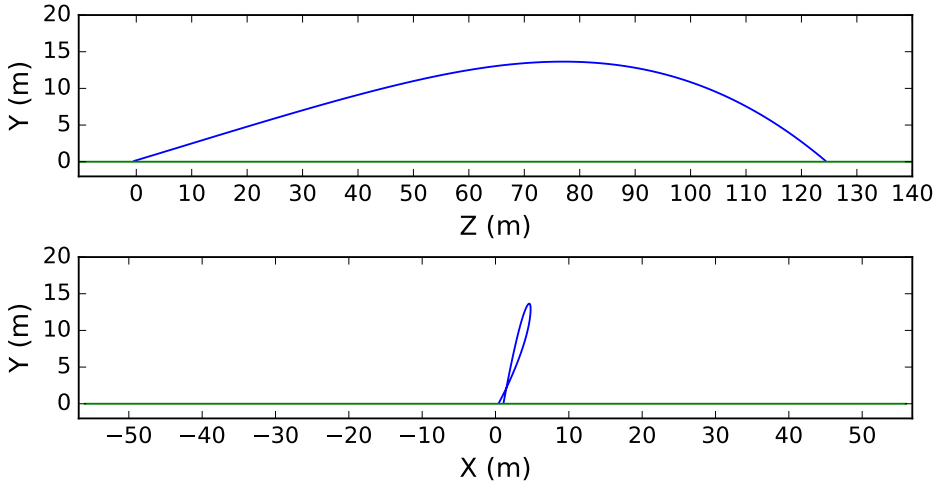


Figure 1.1: An example of how a golf shot looks. The player is located at $(0, 0, 0)$. The top figure shows how it looks from the side and the bottom one how it looks from the perspective of the player. The green line denotes the ground level.

The most prominent method to simulate the trajectory of a golf ball seems to be numerical simulations of an approximative physical model of the golf boll [28, 30]. In this physical model, the input data are the initial velocity and rotation of the ball. In this thesis, a specific version of the problem will be studied. Only the position of the ball will be captured, not the initial velocity or the rotation of the ball. This makes the problem more complicated, as these parameters must first be estimated from the sequence of ball positions, before the numerical simulations of the model can be applied.

This thesis will instead investigate an alternative approach, using recurrent neural networks applied directly to the captured positions. Re-

[1] This is the case for a golf ball shoot by a driver.

current neural networks constitute a suitable method for this specific version of the problem as it does not need to rely on any underlying physical model to simulate the trajectory. Compared to other machine learning methods, recurrent neural networks have a key feature that they can be applied to sequences of varying length without making any assumptions on how many previous points are needed for the predictions. Secondly, recurrent neural networks have been shown to perform better than other methods in similar problems, such as predicting the trajectory of pedestrians [1].

1.1 Problem definition

This thesis will investigate the following question: *How well does recurrent neural networks perform in the task of extrapolating the trajectory of a golf ball, compared to a model based on numerical simulations of a physical model?*

In this problem, a trajectory is defined as a sequence of 3D spatial coordinates with time information. The data set that will be only contains trajectories, with some additional information such as where (i.e. at which golf course) and when they were recorded.

The purpose of the model is then to extrapolate a partial trajectory to a full trajectory based on the previous tracked positions of the ball. As the camera system that supplies input data to the model does not start tracking the ball from its starting position, but instead at some time in the air, the model must be able to extrapolate backwards in time also.

1.2 Delimitation

This thesis will only predict the motion of the golf ball as it travels through the air. The thesis will not deal with how the ball will move after impacting the ground (how it will bounce and roll on the ground).

1.3 Contributions

The main contributions of this thesis are:

- Applying recurrent neural networks to a new time series prediction problem.
- Making comparison between conventional methods: numerical simulations and recurrent neural networks.
- Evaluating impact different factors such as: the amount of training data used, noise in the observed data, the initialization of the model, and how much of the trajectory that must be observed have on the accuracy of the predictions.
- Examining the effect of additional physical constraints incorporated into a recurrent neural network have on the accuracy of the predictions.

1.4 Thesis outline

The outline for this thesis is the following. In chapter 2, the theory required to understand the problem is outlined. This includes both the underlying physics of the problem and recurrent neural networks. In chapter 3, state-of-the-art research is presented for similar problems. In chapter 4, the method used to train and evaluate the prediction models is described. In chapter 5, the results are presented and discussed. In chapter 6, different properties of the methods used are discussed, and finally, the conclusions are formulated in chapter 7.

Chapter 2

Background

In this chapter, the underlying physics of the problem will be presented, the existing model for predicting the trajectories, and the underlying theory behind neural networks.

2.1 Aerodynamics of golf balls

The defining characteristic of the golf ball is its geometry, with small dimples scattered across the surface, which can be seen in figure 2.1. According to the rules of golf [3], the diameter of the golf ball can be at least $42.67\,\mathrm{mm}$ and have a maximum weight of $45.93\,\mathrm{g}$. However, there are no restrictions on the maximum diameter or the minimum weight. A study conducted by Alam et al. [2] found that the average physical dimensions were close to these limits, with an average diameter of $42.7\,\mathrm{mm}$ and mass of $45.5\,\mathrm{g}$ for a set of eight different commercial golf balls. In the same study, they also found out that the aerodynamic characteristics of the balls varied, most notably the amount of drag that was generated.

The aerodynamics of a golf ball is not yet completely understood, as it is affected by multiple factors such as the physical characteristics of the ball: size, shape, depth and pattern of the dimples, weather conditions: wind, temperature, humidity and air pressure, velocity and spin of the ball. [5, 2]

The three main forces that have been identified are gravity, drag and lift [41, 35]. Both the drag and lift forces are hard to calculate, but

Figure 2.1: An example of a golf ball.

both can be approximated using the *drag equation* (which describe the magnitude of these forces) [28]:

$$F = \frac{1}{2} C A \rho u^2 \tag{2.1}$$

where F denotes the force in Newtons, ρ the density of the air, u the velocity of the air relative to the object, C the drag (or lift) coefficient, and A the reference area of the object (how much of the total area of the object that faces the air). The drag and lift coefficients depends on the geometry of the object, and can be calculated experimentally, for example using a wind tunnel [2]. The relative velocity, u can be calculated as $u = v - v_w$ where v denotes the velocity of the object (relative to the ground) and v_w the velocity of the wind [28].

The drag acts in the opposite direction of the motion of the ball, which slows it down. The lift of the ball is generated by how the ball spins, and is due to *Magnus effect*. As the ball spins, the flow of the air is pushed down, which in turn generates the lift, and as the ball rotates faster, more lift is generated. The spin of the ball is denoted by an axis of rotation and the amount of rotation, which is known as the *spin vector*, $\boldsymbol{\alpha}$. The lift is then perpendicular to both the spin and velocity vectors, in other words the same direction as $\boldsymbol{\alpha} \times \mathbf{v}$. [32] The lift force causes acceleration both in the vertical axis (which causes the golf ball travel longer) and in the lateral axis (which causes the golf ball to move sideways). The component of the spin that causes the acceleration in the vertical axis is known as the *backspin*, which means that the ball rotates backwards (compared to the motion of the ball). It is also possible to have *topspin* — that the golf ball rotates forwards instead which has the opposite effect of making the golf ball travel shorter, as the acceleration downwards increases. However, this is uncommon in practice. The component of the spin that causes the ball to move sideways is

known as *sidespin*. [28] The total amount of spin is not constant in the trajectory, as noted by Smits and Smith in [41], it decreases as the ball travels in the air. For more detail of the forces involved, see figure 2.2.

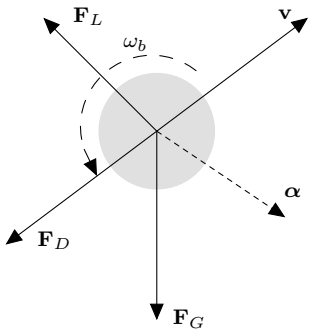


Figure 2.2: A model of the forces involved. $\mathbf{F}_G$ denotes the gravitational force, $\mathbf{F}_D$ the drag force, $\mathbf{F}_L$ the lift force and ω_b the backspin. The spin vector $\boldsymbol{\alpha}$ points outward from the figure, towards the reader.

The shape of the golf ball has a large influence on how it behaves in the air. A ball with dimples flies longer than a ball without, in other words a smooth ball. This is the reason why golf balls are dimpled. [24] Ting demonstrated in [45] that as the size of the dimples increase, the amount of drag decrease. However, there is a limit, where the drag instead increases. The shape of the dimple also influences the amount of drag, and two common shapes are circular and hexagonal dimples [2].

2.1.1 Wind

As mentioned in the previous section, the lift and drag forces depend on the velocity of the wind. It is also possible to assume no wind, that $v_w = 0$. The wind can be described by two components: the crosswind and headwind/tailwind. The crosswind acts sideways, similar as the sidespin — causing the ball to travel sideways. The headwind or tailwind describe the effect that the wind has in the direction of motion, where the headwind is in the opposite direction of the motion and the tailwind in the same direction of the motion. [30]

If only the position of the ball is observed, then the effect that the wind and the spin have on the trajectory of the ball can be hard to isolate,

as the result of these effects become coupled [30]. This makes it hard in practice to determine the spin and wind of the ball, if no additional instruments are used.

2.2 Existing model

An existing model for predicting the trajectory of a golf ball was provided by Joakim Hugmark. This model used the random sample consensus (RANSAC) algorithm [43] in combination with a physical model (see section 2.1 and 4.6.1) to predict the remaining part of the trajectory.

The RANSAC algorithm is a general algorithm for finding a parametric model that describe a set of points that are noisy and contains outliers such as lines or more complex models such as the trajectory of a golf ball. A short description of the algorithm for lines are: sample two points (the minimum number of points required to describe a line), using these two points, find the parameters for a model of the line: $y = kx + m$. For the remaining set of points, calculate the number of points that are close to this line (using some metric and definition of closeness). These points are denoted as *inliers*. Repeat this process for N iterations. When this is done, select the parameters which had the most number of inliers as the parameters of the model.

For the case of golf ball trajectories, the algorithm works as the following: sample four points, solve an optimization problem using the previously mentioned physical model for the initial velocity, spin vector, drag coefficient and lift coefficient, the other parameters such as the mass and radius of the ball are fixed using measured values. From these parameters, a new trajectory is created by using the estimated parameters in combination with numerical integration of the physical model to compute the full trajectory. From this full trajectory, the number of inliers are then calculated.

In addition to this, for trajectories that has been determined to be long, the existing model tries to weight the start and end of the trajectory more than the middle part of the trajectory. This is performed by dividing the trajectory into two parts, and then performing simulations independently for each part. These parts are then combined to create a full trajectory.

2.3 Noise reduction

One important realization of data captured from the real world is that there is noise in the data. The output from a machine learning method might also appear to be noisy, which in this case means that the predicted trajectory does not seem smooth, making small (or large) oscillations from the true trajectory. Both cases can be handled by the same algorithms.

2.3.1 Geometric algorithms

A geometric algorithm looks only at the position (the raw trajectory), and does not use any knowledge of the problem itself to smooth out the noise in the trajectory. One possible algorithm is fitting a curve of some kind to the data, such as a polynomial of a certain degree. When we have this curve, we can then generate new, smooth data points by calculating the position at the point in the curve with the same time as the original position. Using a polynomial is problematic though, as the data can most likely not be described by a polynomial curve, and to have the most flexibility, we would like to use a polynomial with a high degree. However, using a very high degree polynomial can lead to instability, causing the curve to oscillate. Therefore, the most promising solution is to combine several simple polynomials at different parts of the curve, to describe the entire curve. This is the idea behind *splines*. [6]

2.3.2 Kalman filter

A Kalman filter is an algorithm that handles data with uncertainty (noise), both to make predictions, combine several different data sources and to filter out the noise from the data (which is our use case). We will use the version and notation of the algorithm as defined in [7]. In a Kalman filter, we define the state of the system as X_k, which is the property of the system that we would like to obtain. The state of the system is hidden, something that we cannot observe directly. To estimate the state, we have an observation Y_k (also known as the measurement) which depend on the distribution of X_k. That is, we cannot

measure X_k directly, but we can however measure the outcome of X_k through Y_k.

Using this, the dynamics of the system can be defined as:

$$\begin{aligned} X_{k+1} &= A_k X_k + R_k U_k \\ Y_k &= B_k X_k + S_k V_k \end{aligned} \tag{2.2}$$

where A_k defines the state transition matrix (how the state changes), B_k the measurement transition matrix (how the measurements changes depending on the state), R_k the square root of the state noise covariance matrix, S_k the square root of the measurement noise covariance, U_k the noise of the state and V_k the noise of the measurement. In a Kalman filter, it is assumed that the noise follows a normal distribution with zero mean and unit covariance. Note that the Kalman filter is sometimes defined with covariance matrices instead of R_k, S_k. The relationship is given by: $Cov[S_k V_k] = S_k S_k^T$ and $Cov[R_k U_k] = R_k R_k^T$. In the univariate case, this means that in this notation we describe the standard deviation instead of the variance.

A Kalman filter works by using only the previous state of the system and the current measurement, to predict the next state of the system. This means that a Kalman filter does not use the full history of previous states and measurements when making a prediction, we are assuming that this can be described completely by the current state and measurement (the Markov assumption). If we want to extend the algorithm to consider the full history, this extension is known as *Kalman smoothing*. This works by first running the normal Kalman filter on the data, saving the intermediate calculations which are used to determine the state. We then perform similar calculations, but backwards instead (from end to start). Finally, the forwards and backwards calculations are combined to create a new estimate, that considers all the other states and observations in the entire sequence for each estimate.

2.4 Feedforward neural networks

A neural network is a type of computational model, often represented as a graph that can approximate any continuous function (with some constraints). The model consists of simple nodes, where each operation between the different nodes in the graph are differentiable. The

output of the network is determined by the strengths of the connections between the nodes, which is known as the weights or parameters of the network. The network contains two modes: the forward and backward pass. The forward pass calculates the output (i.e. the predicted value) for a given input. The backward pass calculates the gradient with an algorithm known as *backpropagation*, which is used to update the weights (train the model). One common neural network architecture is the *feedforward architecture*. In this architecture, the data always flow forwards, that is, there exists no cycles in the graph. [18]

A common type of feedforward architecture is the *multilayer perceptron* (MLP), which consist of layers that are fully connected, meaning that all the nodes in the previous layer are connected to all the nodes in the next layer. In this architecture, the first layer is denoted as the *input layer* which takes the input. This is then sent to one or more *hidden layers*. Finally, the output of the last hidden layer is sent as input to the *output layer* with the purpose of taking the high-dimensional output of the layer and turning it into an output vector that matches the dimension of the function to learn. [18]

2.5 Time series prediction

Time series prediction is the problem of predicting future data based on previous, historical data. A time series is defined as a sequence of scalars or vectors which depend on the time. It is formally defined as $\mathbf{x}(t_0), \mathbf{x}(t_1), \mathbf{x}(t_2), ...$ where $\mathbf{x}(t)$ denotes a data point at time t. Using this notation, a prediction can then be defined as[1]

$$\hat{\mathbf{x}}(t+s) = f(\mathbf{x}(t), \mathbf{x}(t-1), \mathbf{x}(t-2), ...) \tag{2.3}$$

where f is our predictor, $\hat{\mathbf{x}}$ our prediction and s denotes how far into the future the prediction is, also known as the *horizon*. [33]

If our predictor has a horizon of s_1, but our desired prediction is s_2 where $s_2 > s_1$, it is still possible to make a prediction. For the sake of simplicity assume that our horizon is $s_1 = 1$, then this prediction can be made in an iterative fashion. Assume that t is the current time, and $t + s_2$ is the desired prediction. Then we can predict the value at $t + s_2$

[1] $t - 1$ should be interpreted as the previous time step.

by first predicting $t+1$, then using this as the input to our predictor to predict $t+2$, and so on until we reach $t+s_2$. This is known as an *iterated prediction*, in contrast to a *direct prediction* where we predict $t+s_2$ directly. The disadvantage of iterated prediction is that this does not consider the errors that accumulate through predictions in the intermediate steps. The disadvantage of direct prediction is that separate models must be created to predict each specific time horizon [29]

There exists many possible predictors f, one possibility is using neural networks and one possible neural network architecture is the sliding window architecture. In this architecture, a prediction is made by feeding data points from several times in the past into the network. In other words, the input layer will consist of nodes for each point $\mathbf{x}(t), \mathbf{x}(t-1), \mathbf{x}(t-2), ..., \mathbf{x}(t-d-1)$, where $t, t-1, ..., t-d-1$ is known as the *time window*. Looking at figure 2.3, as we make predictions further into the future, the window is moved, by moving the previous values in the window one step to the left, and replacing the right most value with the new value. [14]

(a) Predicting the value at $t = 5$

(b) Predicting the value at $t = 6$

Figure 2.3: The sliding window architecture. The shaded area denotes the current window (of size three in this case), as we make predictions further into the future, it is moved to the right.

Using this architecture, a choice must be made on the number of data points to feed into the network, since a neural network requires that the input of the network to be of fixed size. Choosing the right size of the time window is a key factor in the overall performance of the network. Using too few data points results in the network having problems with distinguishing data points, and using too many will result in the network treating some of the input as noise, as the most important part of the data is located within some subset of the window. [14]

An alternative architecture is to allow the network to have connections to data points in the past, which means that a choice does not need to

be made on the number of data points to feed in, the network remembers the previous data. These kinds of networks are known as *recurrent neural networks*, which section 2.6 will be about.

2.6 Recurrent neural networks

A recurrent neural network (RNN) is a type of neural network that is used for processing sequential data, for example time series. The network has connections to the previous time steps. That is, the network at time step t is connected to the state of network at the time $t-1$, which is connected to the state at time $t-2$, and so on. This makes the graph of the network appear as a chain, see figure 2.4.

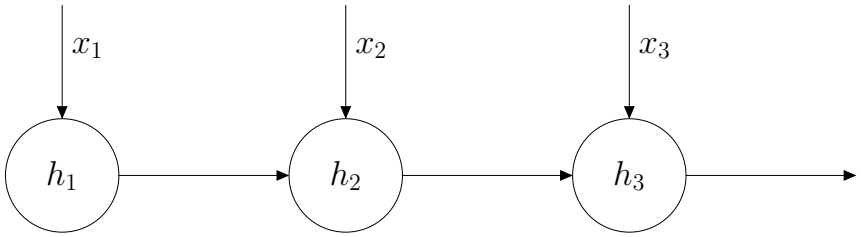


Figure 2.4: A recurrent neural network.

This structure means that the network can remember previous input data, and uses this in combination with latest input data to make new predictions. The memory of the previous input data can either be modeled implicitly, by how the data flows through the network or explicitly, that the network has memory used to store data. This memory is called the *state*, and is denoted at the time t as h_t. How the state changes can be described by the following equation:

$$h_t = f(h_{t-1}, x_t; \theta) \tag{2.4}$$

where f is the network and θ is the parameters of the network. To compute the state at time t, we need to know the state at time $t-1$:

$$h_t = f(f(h_{t-2}, x_{t-1}; \theta), x_t; \theta) \tag{2.5}$$

We also need know the state at time $t-2$, and so on, until we reach $t = 0$. This is a recursive relation, and explicitly calculating the state further back in time is known as *unrolling* the network. [18]

Compared to the sliding window architecture, this introduces two advantages: the network can handle sequences of varying length without making any assumptions on how many data points that are important for a prediction and allows the parameters to be shared at every time step. [18] Parameter sharing means that the same parameters are used for all input points. For the sliding window architecture, the parameters are not shared as different parameters are used for each of the $\mathbf{x}(t), \mathbf{x}(t-1), ...$ input nodes [15].

2.6.1 Regression

In machine learning, regression is the task of predicting the numerical output of some real-valued function f. For neural networks, vector-valued outputs can be handled by using multiple output nodes in the network. [18]

For the optimization to work, a cost function J needs to be defined, which the network will try to optimize. The cost function is defined as the expected output of a loss function L, which is the cost for a specific example [18]:

$$J(\theta) = \mathbb{E}[L(y, \mathbf{x}; \theta)] \tag{2.6}$$

One common loss function is the *squared error*:

$$L(y, \mathbf{x}; \theta) = \frac{1}{2}(y - \hat{y}(\mathbf{x}))^2 \tag{2.7}$$

where y denotes the true value and $\hat{y}$ the prediction [18]. One problem with this loss function is that it is sensitive to outliers, in other words, the overall cost is dominated by the examples where the error is large. An alternative loss function is the *Huber loss* [23], which is less sensitive to outliers. It consists of two parts, a squared part and a linear part (for outliers), see figure 2.5. It is given by the following equation:

$$L(y, \mathbf{x}; \theta) = \begin{cases} \frac{1}{2}(y - \hat{y}(\mathbf{x}))^2 & \text{if } |y - \hat{y}(\mathbf{x})| \leq \delta \\ \delta|(y - \hat{y}(\mathbf{x}))| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases} \tag{2.8}$$

where δ denotes where the loss becomes linear instead of squared.

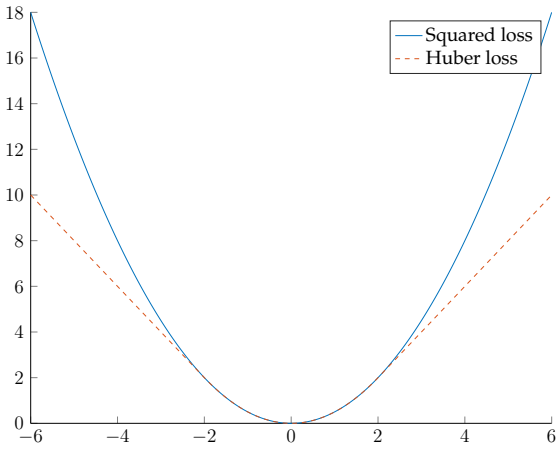


Figure 2.5: Plot comparing the squared- and Huber loss (with $\delta = 2$). The x-axis is the error and the y-axis the loss.

Multiple outputs

As mentioned earlier, regression can be generalized to multiple outputs by adding n number of nodes in the output layer, if the dimension of the output of the function to be learned is n. A straightforward way to define the loss in this case is (using the squared loss function):

$$L(\mathbf{y}, \mathbf{x}; \theta) = \frac{1}{2} \sum_{i=0}^{n} (\mathbf{y}_i - \hat{\mathbf{y}}(\mathbf{x})_i)^2 \tag{2.9}$$

This is the same as assuming that the different outputs are uncorrelated, which means that in our optimization method, we do not use the fact that in most cases the outputs are in fact correlated. Methods that exploit this fact learns in addition to the weights, the covariance between the outputs, and uses this in the overall loss function. [37]

2.6.2 Backpropagation through time

To train a recurrent neural network, the gradient must be calculated. This is done by the backpropagation through time algorithm (BPTT), which works like the regular backpropagation algorithm. In this algorithm, the network is unrolled, and the gradient is calculated in a sequential fashion: the gradient at time t is propagated back to the

state of the network at time $t-1$, and so on, until $t=0$. [18]

Ideally, we would like to propagate the gradient all the way back to $t=0$, but if the sequence is long this will take too long to compute. Therefore, we typically limit the number of steps that the gradient is propagated through during training for an example which is known as *truncation*. [48]

If the purpose of the network is to predict the next state of the network, and the output is then fed as input to the next prediction, a procedure known as *teacher forcing* can be used during training. This works by instead of feeding the predicted value as input to the next stage, the ground truth is instead fed as input. As the state of the network is preserved between predictions in the same sequence, the information about the previous input data is not lost. [18] If the training data contains information when a new sequence start, the state of the network is then reset before starting the training of each new sequence (e.g. by setting it to zero). [15]

2.6.3 Vanishing and exploding gradients

Two of the main problems when training deep neural networks[2] are the vanishing and exploding gradient problems [18]. Intuitive, both problems are related to the fact that when calculating the gradient through backpropagation, it is calculated by multiplying the gradient of the current layer by the gradient of the deeper layer.

The vanishing gradient problem means that as we propagate through the layers, the gradient goes to zero exponentially fast. This means that for deep networks the model does not learn anything at all, as the gradient is used to update the parameters of the model. The problem is related to the fact that the derivative of an activation function is less than 1 for all inputs. [18]

The vanishing gradient problem is a serious problem for recurrent neural networks, as they can be seen as network with many layers during training. According to Hochreiter and Schmidhuber [22], back-propagating through just 5-10 time steps makes the vanishing gradient affect apparent. This means that if this problem is not solved, we cannot learn long-term dependencies as the gradient approaches zero.

[2]Depth denotes both the number of layers and unrollings.

Solutions to this problem for recurrent neural networks are presented in section 2.6.4.

The exploding gradient problem is the opposite — the gradient approaches infinity exponentially fast as we propagate through the layers. The exploding gradient problem leads to the training becoming unstable (i.e. the numerical values tends to be ∞ or NaN). One solution to the problem is to simply constrain the norm of the gradient, which is known as *gradient clipping*. This works by having an upper limit of the norm, denoted $threshold$, and if the norm is greater than this limit, clip the norm such that the norm of the gradient is $threshold$ but with the same direction. [18]

2.6.4 Long short-term memory

Today, one of the most popular type of recurrent neural networks are the long short-term memory (LSTM) networks. Using this type of network had led to state-of-the-art performance in different tasks such as speech recognition [19] and unsegmented handwriting recognition [21], meaning that the input image has not been preprocessed into smaller images for each character.

This type of network was first introduced in 1997 by Hochreiter and Schmidhuber [22], and the version that is mostly used is by Graves and Schmidhuber [20]. The network is explicitly designed to avoid the vanishing gradient problem and to handle long-term dependencies. At the heart of these networks are cells, with explicit memory and functions to control the flow of information (known as gates). The memory cell is of fixed size and acts a loose summary of all the previous input data sent to the network.

To control the flow of information, LSTM uses three gates: forget (f_t), input (i_t) and output (o_t). The forget gate controls when we should remember old information, input gate controls when we should store information, and finally the output gate when we should read from the memory. The forget gate allows the network to learn when it should forget stored data, which is useful when the training sequence consist of one long sequence which has not been segmented into smaller sequences, but are in fact several smaller sequences that has been concatenated together [15].

Let c_t be the memory of the cell, h_t the output, W, U the weights and b the biases. This gives the following equations for the forward pass of the network (with $\circ$ as element-wise multiplication):

$$\begin{aligned} f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\ i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\ o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\ c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\ h_t &= o_t \circ \sigma_h(c_t) \end{aligned} \tag{2.10}$$

Typically, σ_g is the sigmoid function ($\frac{1}{1+e^{-x}}$) which has output in the range $(0, 1)$, which typically is interpreted as a probability. For, σ_c, σ_h the $tanh$ function is typically used, which has output in the range $(-1, 1)$. However, [16] makes the case that $\sigma_h(x) = x$ should instead be used, as there is no empirical evidence that a non-linearity is needed for the output.

The forget gate introduces some problems in the early phase of the training. If the bias term for the forget gate is initialized to zero, which it typically is for neural networks, then the forget gate will output something close to zero (i.e. forget everything). This means that the network will have a hard time learning long-term dependencies in the beginning. To avoid this problem, the bias for the forget gate is initialized to some large value, typically 1, which causes the network to not use the forget gate in the beginning of the training phase. [49]

The key feature of LSTM networks is that these networks avoids the vanishing gradient problem. The reason for this is that only the memory cell (c_t) is connected recurrently to the past, and the "strength" of this connection is controlled by the gates. For the case when the gates do not activate, that is, the memory cell it not reset (f_t close to one) or updated (i_t close to zero), the value of the cell remains the same, that is, $c_{t+1} = c_t$, which in other words is the identity function. This function has a gradient magnitude of one, which neither vanishes or explodes. This means that the network can learn long-term dependencies, as the memory of the network is only updated when it needs to. [22, 15]

2.6.5 Regularization

The most important thing for a model is its ability to generalize to unseen data. Different strategies for increasing the ability to generalize is known as *regularization*. The simplest form of regularization is to use more data [26].

One common type of regularization is *weight decay*, which adds a cost on the values of the parameters itself, rather than just the loss function used for optimizing the task. One type is L_2 regularization, which forces the weights towards the origin. This works by penalizing the parameters by the square of the magnitude of the parameter vector θ [18]:

$$\frac{||\theta||_2^2}{2} \tag{2.11}$$

Another type is *dropout*, where the network with the probability p ignores the output of a node (which means that the node is ignored both in the forward and the backward pass). This means that during training we sample different architectures that shares the same weights, which leads to the network learning more robust features. When the network is used (i.e. not in training), the dropout effect is replaced by multiplying each node's activation by the geometric mean of the distribution produced by all the possible networks. [26]

Care must be taken when applying dropout to recurrent networks. Applying the dropout effect to the recurrent connections (h_{t-1}, c_{t-1}) hurts the learning, and to correct apply it, it should be on the input connections (x_t). [50] Results from [51] suggest that combining L_2 with dropout yields impressive results, which are better than just using one of them.

2.6.6 Deep recurrent neural networks

One very important realization in recent years is that increasing the depth of the model, that is, adding more hidden layers in the network, leads to better performance. The intuition behind this is that this allows the network to learn a better representation of the data that is expressed in the form of a simpler representation. One common example is for image classification. The input layer consists of raw pixels, the first hidden layer learns a very simple representation, for example

edges. Using edges, a more complex representation can be expressed in the next hidden layer, for example contours. If we continue to add more hidden layers, the representations get more complex for each layer. This in turn makes it easier for the last layer, the classifier, of identifying the correct class for the input image, even though the classifier can be a simple linear classifier. [18]

This concept of adding more layers for feedforward networks has yielded impressive results in tasks such as image classification [26] and a question then arises if adding more layers for recurrent networks would yield large performance improvements, as they already are deep in time. Results from [19] in the task of speech recognition suggest that this is also the case for recurrent networks. Adding more layers works similar as for feedforward networks, the output from hidden layer 1 is feed as input to hidden layer 2, and so on.

Chapter 3

Related work

In this chapter, works related to predicting the trajectory of golf balls and other type of objects are presented. Emphasis are put on approaches that used recurrent neural networks instead of other types of machine learning based methods.

3.1 Golf ball trajectory prediction

Sköld [40] studied a similar problem, by inferring the 3D-trajectory of a golf ball using data only captured from one camera (a 2D-trajectory) and a database of 3D-trajectories with the corresponding 2D-trajectories. Several different approaches were tested, but the one that performed the best used nearest neighbor search in combination with a Kalman filter (see section 2.3.2 for information about Kalman filters). The nearest neighbor search was used to find other 2D trajectories that were similar, using a Euclidean metric. Using the trajectories that were determined to be similar, a complete 3D-trajectory was then computed using a combination of the 3D-trajectories. Different ways were investigated how to combine these trajectories, such as an unweighted average and a weighted average, giving more weight depending on how similar the candidate trajectory was to the input trajectory. To make the final trajectory appear smoother and to remove the underlying noise, a Kalman filter was applied.

Bačić [4] used a system that captured the movements of a golf club as it hit the ball, and used this information to classify which pattern (from

a list of nine predefined patterns) the trajectory would have using machine learning. The methods that were tested were neural networks and support vector machines, achieving similar performance for both methods in term of classification error. However, the purpose of this work was to create an expert system, that given which pattern the trajectory had, give the player advice how the performance of the player could be improved, not to predict the trajectory of the ball after being hit by the club.

A related problem is to instead simulate the entire trajectory, based on parameters such as the initial velocity, rotation of the ball and weather conditions. McPhee and Andrews [30] investigated how the trajectory of a golf ball could be simulated, using the initial speed, launch angle, rotation of the ball and wind speed as input data. To perform these simulations, they used a physical model like the one discussed in section 2.1, but with two important differences: they assumed a linear impact of the velocity on the drag force instead of a squared one, and that the motion along each axis was independent of the other axes. This lead to the motion being described by three independent differential equations, one for each axis. This made it possible to derive an analytical expression for the trajectory, instead of using numerical integration. However, in the model that was used it was assumed that the rotation of the ball was constant, but according to [41] this is not true in practice, as it decays during the flight.

Kim and Baek [5] created a simulator that incorporated many different factors, that previously had not been considered. The factors included wind, air pressure, temperature, dimple geometry, and more. However, the application of their simulator was to create a realistic physical model for the use in video games, not for predicting real trajectories. This meant that many of the equations that were used to describe these variables were not supported by experimental evidence, but instead by arguments that made the simulated trajectories appear realistic.

3.2 Sports trajectory prediction

Kumar et al. [27] studied a similar problem for the game of tennis. They constructed a system where the camera tracked the tennis ball, and when the ball could not be tracked anymore, the system predicted

the remaining part of the trajectory based on the previous tracked positions of the ball. To make the predictions, they used a simple physical model that assumed that the ball followed a parabolic arc, and only considered the force of gravity in the model (see equation 3.1). For the prediction, only a limited amount of information was used, the last two recorded positions of the ball, which were used to estimate the initial position and velocity of the ball in the parabolic arc.

$$\begin{aligned} x_t &= x_0 + v_x t \\ y_t &= y_0 + v_y t \\ z_t &= z_0 + v_z t + \frac{gt^2}{2} \end{aligned} \tag{3.1}$$

Equation 3.1: The equation for parabolic motion, the only parameters are the initial position (x_0, y_0, z_0) and velocity (v_x, v_y, v_z). It is assumed that z is the vertical axis and g is the gravitational constant (≈ -9.8)

A similar method was applied for predicting the trajectories of the ball in the sport of volleyball by Chen et al. [8]. They also assumed that the ball followed a parabolic arc, with gravity as the only acting force. However, compared to the work made by Kumar et al. they used all the previous recorded positions of the ball, by forming an equation system of the parabolic arc equation for each position, and solved it to get a better approximation of the initial position and velocity of the ball.

Chen et al. [9] studied how the trajectories of basketballs could be predicted using only 2D data, that is, using only a single camera. For this to work, they used knowledge of the problem itself, the location of specific features on the basketball court, such as the court lines and the backboard[1] and the physical characteristics of the motion of the ball, by assuming that it followed a parabolic arc.

Chu et al. [11] studied how trajectories of baseballs could be captured from 2D data only. They first extracted all possible trajectories from a single camera and applied a Kalman filter to filter out noise and fill in parts where the trajectory was not captured. As their method generated many possible trajectories where there was only one, they applied a verification scheme using a physical model. This worked by

[1]The thing that the ball is thrown in.

simulating different trajectories by varying the initial conditions of the trajectory, such as the velocity. The physical model considered the spin of the ball, the drag and gravitational forces. The captured trajectories were then compared to these simulated ones, and the trajectories that did not fit the simulations were filtered out.

3.3 General trajectory prediction

The problem of predicting the trajectory of an object can be generalized to other domains, not just for object traveling in the air. Choi and Hebert [10] studied a generalized version of the trajectory prediction problem where the input data only contained the position and speed of the object at different points in time. They considered a trajectory to be constructed of smaller segments. A Markov model was then constructed to model the transitions between the different segments, which was then used to make predictions of the trajectory, by composing different segments. To construct the smaller segments used by the model, the training data was divided into segments of equal length in time and clustered using the K-means algorithm to identify similar segments. These similar segments were then used when predicting the trajectory.

Alahi et al. [1] applied recurrent neural networks to predict the trajectories of humans walking in an environment, where the trajectories for different persons were considered at the same time. Compared to other state-of-the-art methods used for the problem, their method only took the raw data as input (the sequence of x- and y-coordinates of the person to predict) instead of using handcrafted features. Compared to other trajectory prediction problems, it could not be assumed that for trajectories that were close in time were independent of each other, as the trajectory for a specific person depended on the trajectory of other persons. To handle this problem, they introduced a new type of recurrent neural network. They first let each person be represented by a separate network, but connected networks that were spatially close through a special type of layer they called “social pooling”. This layer worked in the following way: when the position needed to be predicted for time step $t+1$ for a specific person, the input was taken both from the persons own network at time t, but also from nearby persons' networks at time t. Using this approach, they achieved state-of-the-art performance on different datasets used for the problem.

3.4 Physical modeling using machine learning

Wang et al. [46] studied how neural networks could be applied to model a physical system assumed to follow an (unknown) ordinary differential equation. They looked at problems where the purpose was to estimate the state of the system, $\mathbf{x}(t)$ which was assumed to depend on a differential equation, $\dot{\mathbf{x}}(t) = f(\mathbf{x})$. Then they let the neural network learn this differential equation in order to make future predictions, given $\mathbf{x}(t)$: predict $\mathbf{x}(t+1)$. To make the prediction using this learned differential equation, they used a fourth order Runge-Kutta method, which is known from numerical analysis to be a very stable numerical integrator. The output of the numerical integrator was used to compute the loss, by computing the difference between the predicted state of the system and the true state. They make the argument that the main advantage of this approach, compared to predicting the state directly, is that this produces more stable predictions, and allows previous physical models to be incorporated to make better predictions.

Ehrhardt et al. [12] studied how the trajectory of an object sliding down a plane with a varying degree of friction could be estimated by just observing the object for a few video frames in the beginning of the trajectory using deep neural networks. In this work, they did not use any previous physical knowledge of the problem, but let the network predict the trajectory directly. The problem that was studied was a simplified one, with synthetically generated data. This was performed using an existing 3D rendering engine with physics simulations. This system was given an initial configuration of the world (which was not known to the neural network) such as the initial position of the object and the friction of the plane. Then the simulator performed both the physical simulations of this interaction and the rendering of the world.

Fragkiadaki et al. [13] studied how long-term predictions could be made of the trajectory of the balls in the game of billiard. In the setup that was used, a virtual environment was used instead of a real game, and instead of having a player with a cue stick, an agent was used that pushed the ball instead. To make predictions, a convolutional neural network was combined with a recurrent neural network (in the form

of a long short-term memory network, see section 2.6.4). The input to the network was a rendering (using a 3D engine) of the scene. The input was first sent to the convolutional network, and then the output of this network, was sent to the recurrent neural network, to handle the long-term dependencies, in combination with the force that was applied to push the ball. The output was then the position of the ball for the next 20 frames.

Chapter 4

Method

In this chapter, the method that was used to train and evaluate the models are presented.

4.1 Definitions

A trajectory T was defined as an ordered sequence of 3D points $\mathbf{p}(t)$ at specific points in time t, where $\mathbf{p} = (x, y, z)$, more formally defined as:

$$T = [\mathbf{p}(t_1), \mathbf{p}(t_2), ..., \mathbf{p}(t_m)] \tag{4.1}$$

In the program that captured the input data, the difference in time between each data point was constant (known as the time step or Δt). This meant that a trajectory was a discrete sequence of points instead, which simplified the problem. Note, in the data that was captured, for some of the trajectories, points were missing in the middle of the trajectory. This meant that the time between two captured points might not be one time step.

The coordinate system that was used defined the y-axis as the vertical axis with a positive value pointing upwards, and $y = 0$ as the ground level. The z-axis denoted the forward axis, with a positive value pointing forward, and finally, the x-axis as the lateral axis with a positive value pointing towards the right.

4.2 Data capture

The trajectories were captured from a stereo camera system that consisted of two cameras separated by a certain amount of distance. The output from this camera system were 3D coordinates, which meant that no extra processing was required to obtain the 3D trajectory from the data set. Two (or more) cameras are required to estimate the distance to the object, for information about how this is performed in general, refer to any book computer vision (e.g. [43]).

4.2.1 Sources of noise

There are multiple sources of noise that exist in the data set. Some of the sources are:

- Extrinsic camera calibration: to be able to calculate the 3D coordinate, the extrinsic calibration between the two cameras was needed. This consist of the relative translation and rotation between the cameras. These values are hard to estimate exactly.
- Intrinsic camera calibration: the lens of the camera causes nonlinear distortion of the captured image. These effects are hard to model.
- Image noise: this can cause the tracking software to estimate the position of the ball in the cameras incorrectly.

4.3 Data set

The data set consisted of trajectories recorded from different outdoor driving ranges (denoted a site) around the world. Each site had a varying degree of length, width, number of bays and active players during each recorded day. This meant that both the number of recorded shots and the characteristics between the sites were different. For information about the type of the trajectories that were used for each site, see appendix A.

In total, five sites were used (denoted Site 1, 2, 3, 4, 5). For each site, the data that was used were all in the same time period (e.g. the

same month). The data set for each site were partitioned into three parts: training, validation and test. The partition was made randomly, such that each part had a uniform representation of the entire time period. For training, 15 000 trajectories were used, 1 000 for validation and 3 000 for the test set. For the sites used, the cameras used to capture the trajectories had a different frame rate, and this information is presented in table 4.1

Table 4.1: The time steps each site used.

Site #	Time step (seconds)
1	1/25
2	1/25
3	1/25
4	1/29.97
5	1/29.97

4.3.1 Filtering

In the data set, not all trajectories had been captured from start to finish. To present as compete trajectories as possible during training, only trajectories that had been captured for a long time was used. How long a trajectory had been tracked was defined as the ratio between the number of tracked positions and the number of points estimated of the trajectory by the existing model. The ratio that was used is 85 %, meaning that each trajectory had been tracked for at least 85 % of the complete trajectory. Looking at figure 4.1, for a typical trajectory, most of the missing data was in the beginning of the trajectory, with some missing towards the end.

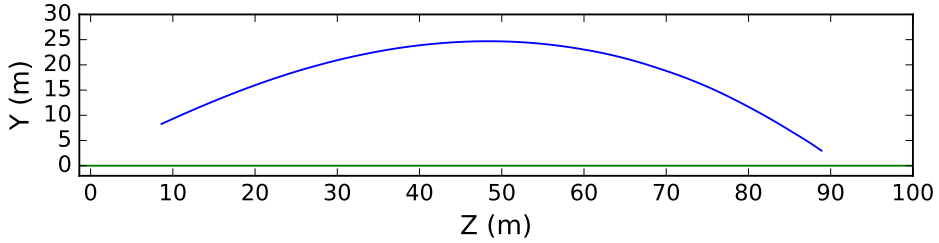


Figure 4.1: A plot in the Z-Y plane of how a typical shot looks.

4.4 Data processing

The raw trajectories were processed in different ways to reduce the amount of noise and to create a coordinate system that was common for all trajectories. The overall process is visualized in figure 4.2. The first stage subtracted the starting position from all other points in the trajectory, so that the trajectory was described relative to the starting position.

The next stage was to normalize the coordinate system of the trajectory, so that the forward direction of the trajectory aligned with the z-axis and the lateral direction with the x-axis. This was done by calculating the direction in the Z-X plane from the first point of the trajectory to the 20:th point of the trajectory. The choice of the 20:th point was arbitrary, but is related to the fact that the models that used a sliding window had a window size of 20. This direction then formed the forward direction of the coordinate system. The vertical axis was kept the same (i.e. $(0, 1, 0)$), and the lateral axis was defined as the cross product between these two axis vectors. The trajectory was then transformed to this coordinate system.

After this was performed, the trajectories were mean-normalized, such that if all the trajectories for a site were concatenated to one long sequence, then the mean of each coordinate axis would be zero and the standard deviation 1. For a site, the training set was used to calculate the mean and standard deviation, and was then applied to the training, validation and test set.

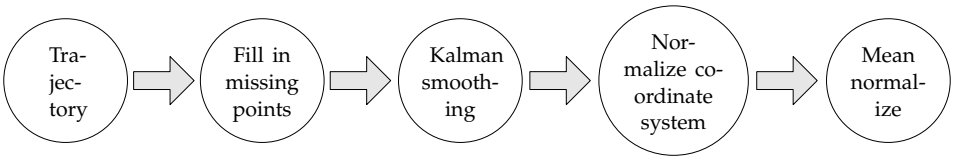


Figure 4.2: An overview of the different data processing steps.

4.4.1 Noise reduction

The noise in the data was reduced using the Kalman smoothing algorithm (see section 2.3.2). The primary reason for choosing the Kalman smoothing algorithm over a spline was that the algorithm only re-

moved the noise without changing the shape of the trajectory in anyway. The smoothing was not applied to the validation or testing data, so that errors introduced by the noise reduction algorithm was not used in the evaluation of the models.

Before applying the noise reduction algorithm, missing points were filled in, using linear interpolation between the points before and after. In most cases, the points were just missing for a few time steps. The errors introduced by the linear interpolation was then easily handled by the Kalman smoothing algorithm.

To use a Kalman filter, a model of the system's dynamics was needed. First, the state of the system was defined as both the position and velocity of the ball, $X_k = (x_k, y_k, z_k, \dot{x}_k, \dot{y}_k, \dot{z}_k)$ (where $\dot{x}_k$ denotes the first order derivative, the velocity). As only the position of the ball was observed, the measurement was then defined as the observed noisy position of the ball at time t_k, $Y_k = \mathbf{p}(t_k) = (x_k, y_k, z_k)$. This gives that X_k describe the position where the noise in the observed position Y_k has been reduced.

Using this definition of the state and measurement, the dynamics can then be described by kinematics (only shown for the x-axis here, but it is the same for y and z):

$$\begin{aligned} X_{k+1,x} &= X_{k,x} + X_{k,\dot{x}}\Delta t + \frac{u_{k,x}\Delta t^2}{2} \\ X_{k+1,\dot{x}} &= X_{k,\dot{x}} + u_{k,\dot{x}}\Delta t \\ Y_{k,x} &= X_{k,x} + \frac{v_{k,x}\Delta t^2}{2} \end{aligned} \tag{4.2}$$

where $X_{k,x}$ denoted the x value of the state tuple X_k, $X_{k,\dot{x}}$ the x component of the velocity of the state tuple, $Y_{k,x}$ the x value of the measurement tuple Y_k, $u_{k,x}$ the noise term for the x component of the state, $u_{k,\dot{x}}$ the noise term for the x component of the velocity of the state and finally, $v_{k,x}$ the noise term for the x component of the measurement.

The noise terms for the position of the state (u_x, u_y, u_z) modeled the change in position that was not described by the velocity term. In the model used, this corresponded to the acceleration. For the measurement (third equation), the noise (v) was assumed to appear due to the acceleration. Incorporating the acceleration into the state would require a model of the change in the acceleration. As seen in section

2.1, the physics behind this was complicated, and as the purpose of the Kalman filter being smoothing out the noise (obtaining the position), a more complex model that described the physics better was not needed. From these formulas, the corresponding matrices were defined as:

$$A_k = \begin{pmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad R_k = \begin{pmatrix} \frac{\Delta t^2}{2} & 0 & 0 \\ 0 & \frac{\Delta t^2}{2} & 0 \\ 0 & 0 & \frac{\Delta t^2}{2} \\ \Delta t & 0 & 0 \\ 0 & \Delta t & 0 \\ 0 & 0 & \Delta t \end{pmatrix} \tag{4.3}$$

$$B_k = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \quad S_k = \begin{pmatrix} \frac{\Delta t^2}{2} & 0 & 0 \\ 0 & \frac{\Delta t^2}{2} & 0 \\ 0 & 0 & \frac{\Delta t^2}{2} \end{pmatrix}$$

4.5 Multilayer perceptron

A multilayer perceptron (MLP) was used as a baseline method for the machine learning methods that were tested. The main reason was that this method was simple, often easy to train and had been extensively studied in literature.

The input to the network consisted of a window of points ($\mathbf{p}(t), \mathbf{p}(t-1), ..., \mathbf{p}(t-d-1)$), and the purpose of the network was to predict the position at time $t+1$. As mentioned earlier, each point was defined by three coordinates (x, y, z). This meant that if the window size was d, then the number of inputs to the network was $3d$. The output layer consisted of three nodes, one for each coordinate of the position to predict. The activation function that was used was the rectified linear activation function (ReLU) [18]: $f(x) = max(x, 0)$. The activation function was not applied to the output layer, only the input and hidden layers.

To perform the training, each training example was defined to consist of a window of a certain size as input, with the value directly following the window as the output. After this was done for each trajectory, all examples in each trajectory was concatenated to one long sequence. This meant that the training process did not consist of complete trajec-

tories, but just small fragments. This was one of the disadvantages of this network architecture, as the network was not trained on complete trajectories, some information of the overall trajectory was lost.

The loss function that was used considered each component of the predicted position to be independent, which lead to sum of the loss of all the components as the total loss for the prediction. The loss function that was used for each component was the Huber loss function with $\delta = 1$ (see section 2.6.1). The cost was then defined to be the sum of the loss for all examples in the batch, instead of the average, as this performed better. A formal definition of this cost is:

$$J = \sum_{j=1}^{m} \sum_{i=1}^{3} L(\hat{f}(\bar{\mathbf{p}}_j, \theta)_i - \mathbf{p}'_{j,i}) \tag{4.4}$$

where m denotes the batch size, j a particular example in this batch, $\bar{\mathbf{p}}_j$ the input window for the j:th example, $\mathbf{p}'_j$ the expected output and i a component ($x = 1, y = 2, z = 3$).

4.5.1 Predictions

A partial trajectory T_p was extended to a full trajectory T in the following way: compute the sliding windows for T_p, and select the last window as the starting point. The next position was then predicted, and used as input data to the next prediction. This was continued until it was determined that the ball had impacted the ground. This was determined to be when the y coordinate of the last predicted position was below zero. If one of the following conditions were true, then the network instead returned "FAILED", and stopped the prediction.

- The number of predicted points exceeded 300.
- The ball started to increase altitude again after reaching the highest point.
- The ball started traveling backwards.

4.5.2 Hyperparameters

The network consisted of an input layer that took 20 points as input (i.e. window size of 20) and one hidden layer with 40 units. As the

MLP only was used as a baseline, no extensive study of the hyperparameters was conducted.

4.6 Recurrent neural network

Using the sliding window architecture (the MLP in this case) implied an assumption that a prediction only depended on a fixed number of previous positions. However, at different stages of the extrapolation of the trajectory, different number of previous positions may be required to get optimal performance. An alternative is to let the network learn the number of previous positions required at each step. A type of neural network with this capability are recurrent neural networks.

The type of RNN that was used was the long short-term memory network. The input layer to this network was the same as the MLP, but used a window size of 1, in other words, no window at all. After the input layer, several LSTM layers were connected to each other, so that the output of the first LSTM layer was used as the input to the second LSTM layer. After the LSTM layers, an output layer was used that took the high dimensional output of the last LSTM layer and produced a prediction of the next position. Instead of predicting the next position directly, the displacement from the input position (at time t) to the position at $t + 1$ was predicted instead.

4.6.1 Output layer

Following the previous section, the output of the network (a 3D vector) was defined to be $\Delta\mathbf{p}(t)$ (the displacement), which was used to produce a prediction of the next position $\hat{\mathbf{p}}(t + 1)$ in the following way:

$$\hat{\mathbf{p}}(t + 1) = \mathbf{p}(t) + \Delta\mathbf{p}(t) \tag{4.5}$$

Different approaches to computing the displacement using the output of the last layer of the network are now be presented.

Direct prediction

The first approach predicted the displacement directly; the output of the network was therefore equal to $\Delta\mathbf{p}(t)$. This meant that no extra constraints were added to the predictions made by the network, the network directly influenced the output.

Force prediction

From the underlying physics of the problem, some of the predictions that the model could make were invalid, or were very unlikely to correspond to the actual position of the ball. That is, there exist some inherent constraints of the motion of the ball, such as there should exist an acceleration downwards due to the force of gravity and the resistance of the air should slow down the ball. Incorporating these constraints to the model explicitly, instead of just letting the model learning these constraints implicitly, should improve the overall accuracy of the model.

The approach that was taken in this thesis was to use an approximative physical model of the forces of the ball to constrain the output of the network. The network estimated the unknown parameters in this model, which were then used to calculate the change in position. This meant that the network was not used to predict the change in position directly, only indirectly by how the parameters lead to the change in position. This was performed in the following way. First, it was assumed that the acceleration in one time step was constant (the time from t to $t + \Delta t$), which lead to the following equation for the displacement of the ball:

$$\Delta\mathbf{p}(t) = \mathbf{v}\Delta t + \frac{\mathbf{a}\Delta t^2}{2} \tag{4.6}$$

where $\mathbf{v}$ denoted the velocity and $\mathbf{a}$ the acceleration of the ball in the time step. The network then predicted the velocity and the parameters that described the acceleration at each step. Following section 2.1, the following model of the forces was used:

$$
\begin{aligned}
\mathbf{F} &= \mathbf{F}_G + \mathbf{F}_D + \mathbf{F}_L \\
\mathbf{F}_G &= m\mathbf{g} \\
\mathbf{F}_D &= -\frac{1}{2} C_D A \rho \|\mathbf{v}\| \mathbf{v} \\
\mathbf{F}_L &= \frac{1}{2} C_L A \rho (\boldsymbol{\alpha} \times \mathbf{v}) \\
A &= \pi r^2
\end{aligned}
\tag{4.7}
$$

At each step, the network predicted the value of the parameters that were unknown, parameters where no ground truth existed for, and used measured values for the other parameters. Secondly, the purpose of using measured values was also to increase the possibility that the model predicted the true meaning of the unknown parameters. The model predicted the parameters $C_D, C_L, \rho, \mathbf{v}, \boldsymbol{\alpha}$ at each step and used fixed values for the parameters $m, r, \mathbf{g}$. The following values were used:

$$
\begin{aligned}
m &= 0.0455\,\mathrm{kg} \\
r &= 0.0213\,\mathrm{m} \\
\mathbf{g} &= (0, -9.81, 0)
\end{aligned}
\tag{4.8}
$$

For the density of the air, ρ, there were two possibilities. Use a standard value for the parameter, which would not likely correspond to the true value of the parameter at the time when the trajectory was recorded, or let the network predict this parameter at each step. The approach that was taken in this thesis was to let the model predict a new value of the parameter at each step.

The main disadvantage of this approach was that it required a physical model of the problem. The model that was used in this thesis was only an approximation of the true physics. This meant that some factors was not accounted for in the model such as the wind. Possible consequences of this may be loss of accuracy, as the effect of these factors must be expressed in the form of the explicitly represented factors instead.

The reason for choosing an RNN instead of trying to estimate these

parameters directly were that most of the parameters were hard to estimate, such as the rotation of the ball, as only the position of the ball was recorded. The only requirement from the network to estimate these parameters was that the model could be used to calculate the change in position. This makes it easy to implement the physical model in the network, and forces the network to learn how to estimate these parameters.

4.6.2 Forward predictions

The prediction phase for the RNN worked similar as the MLP, in the sense that the predictions started from the last point in the input trajectory. However, before the predictions started, the input trajectory was loaded into the network. This was achieved using a "loading procedure" that loaded all the previous points into the networks memory. This worked as the following: send the first point as input to the network and calculate the forward pass (see equation 2.10). The effect of this was that the memory of the network (c_t) changed and the network produced an estimate of the next position (h_t). This estimate was not sent as input to the next step during the loading procedure, but was used in the forward pass for the next step (as h_{t-1}). After this, the next point in the trajectory was sent as input to the model, and the same steps were performed for this point. This was continued for all the points in the input trajectory except for the last point. When the loading was complete, it meant that the memory of the network summarized the input trajectory instead of explicitly representing it, as the size of the memory was fixed.

4.6.3 Backward predictions

Predictions backwards in time were performed similar as the predictions forwards in time, except that the input trajectory was processed in reverse (starting by the last point instead of the first). A separate model was then trained for predicting backwards in time, using the same network architecture and training procedure, except that the trajectories were processed in reverse.

4.6.4 Complete trajectory

To produce one final trajectory, the input trajectory was first sent into the model that performed predictions forwards in time. The result from this extrapolation was then appended to the input trajectory. This was then sent as the input to the model that performed predictions backwards in time, which resulted in a complete trajectory. Finally, the Kalman smoothing algorithm was applied to the complete trajectory to remove the noise from the input trajectory.

4.6.5 Training

The network was trained using complete trajectories. This was done using the teacher forcing approach. As mentioned earlier, this worked in the following way: reset the state of the network to zero, send the first point as input. The prediction of the next position was then computed (and was used in the loss function). Instead of sending the predicted position as the input to the next stage, the ground truth was used instead (i.e. the second point in the trajectory). This was then continued for all the remaining points (except the last).

After this was done, the gradient was calculated. As the number of points in a trajectory was limited (less than 200), the number of backpropagation steps were not truncated, and was backpropagated all the way back to the start of the trajectory for each prediction. This increased the processing time, but was a small price to pay to let the network train on complete trajectories.

For each point in a trajectory, the loss was defined similar as for the MLP, that each component was assumed to be independent, computed using the Huber loss function with $\delta = 1$ (see section 2.6.1). The loss for the entire trajectory was then defined as the sum of the loss of all the points in the trajectory. As many trajectories were trained on at the same time, the cost was then defined as the average loss of all the trajectories in the batch. A formal definition of this cost is:

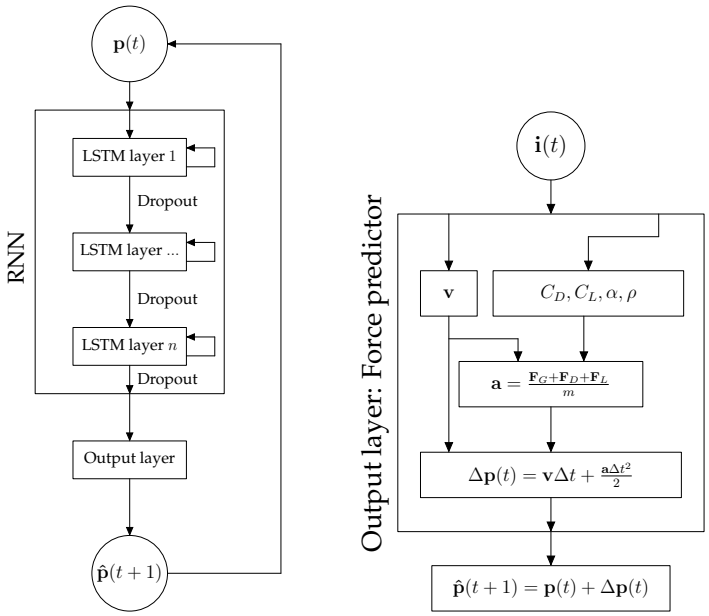
$$J = \frac{1}{m} \sum_{j=1}^{m} \sum_{k=1}^{|T_j|-1} \sum_{i=1}^{3} L(\hat{f}(\mathbf{p}_{j,k}, \theta)_i - \mathbf{p}_{j,k+1,i}) \tag{4.9}$$

where m denotes the batch size, j a particular trajectory in this batch,

$|T_j|$ the number of points in a trajectory, $\mathbf{p}_{j,k}$ the input for the k:th point in the j:th trajectory, $\mathbf{p}_{j,k+1}$ the expected output and i a component ($x = 1, y = 2, z = 3$).

4.6.6 Network architecture

The network used two LSTM layers, each with 320 hidden nodes. Each LSTM layer used the $tanh$ function as cell and output activation function. After each LSTM layer, the dropout regularization technique was applied with $p = 0.10$, in other words, the output of a node was ignored with 10 % probability. The network architecture is visualized in figure 4.3.



(a) The input to the network is a position $\mathbf{p}(t)$, which is sent as input to a RNN. The RNN consist of several LSTM layers, and the output of the last of these layers is sent as input to the output layer. The output is then sent back as input to the network, to continue the prediction.

(b) The input to the output layer is $\mathbf{i}(t)$, which are used to compute the physical parameters and the velocity. This is then sent as input to a physical model which computes the acceleration. Using the velocity and acceleration, the displacement can then be computed.

Figure 4.3: A overview of the network architecture that was used for the RNN.

4.7 Training

The training for all the models was performed using the Adam optimization algorithm [25] with a learning rate of 10^{-5}, batch size of 128 and trained for 1 000 epochs. After each epoch, the order of the batches was shuffled. The parameters of the models were initialized using the Xavier method [17]. The parameters that gave the lowest value for the "mean point difference" metric (see section 4.8.1) on the validation set were selected as the final parameters of the model. For an RNN-based model, a gradient clipping threshold of 5 was used.

4.8 Performance evaluation

To evaluate the performance of the models, different performance metrics were used, allowing different characteristics of the models to be captured.

4.8.1 Mean point difference

Given a trajectory T and a trajectory $\hat{T}$ predicted by a model, then the point difference was defined as the Euclidean distance at the same point in time. The mean point difference was then defined as the difference over all the points in all the trajectories on a site.

Mathematically this was defined as:

$$\begin{aligned} \text{Mean point difference} &= \frac{1}{N} \sum_{j=1}^{m} \sum_{k=1}^{|T_j|} ||T_{j,k} - \hat{T}_{j,k}|| \\ N &= \sum_{j=1}^{m} |T_j| \end{aligned} \tag{4.10}$$

where m denotes the number of trajectories, T_j a particular trajectory, $|T_j|$ the number of points in the trajectory, $T_{j,k}$ the k:th point in this trajectory and $||\cdot||$ the Euclidean norm.

4.8.2 Impact point difference

Given a trajectory T and a trajectory $\hat{T}$ predicted by a model, then the impact point difference was defined as the Euclidean distance between the points at $y = 0$. First, the trajectory was extended following the velocity at the last point, such that a point below $y = 0$ was obtained (to handle cases when the last few points of the trajectory were not captured). Then linear interpolation was applied to the closest point above and below the zero level so that a point with $y = 0$ was obtained.

Mathematically this was defined as:

$$\text{Impact point difference} = ||AtGround(T) - AtGround(\hat{T})|| \quad (4.11)$$

where $AtGround(T)$ is the point at $y = 0$.

The values that are used in the tables are the mean of this metric over all the trajectories on a site.

4.8.3 Last point difference

Given a trajectory T and a trajectory $\hat{T}$ predicted by a model, then the last point difference was defined as the Euclidean distance between the last recorded point in trajectory T, and the point at the same time in the trajectory $\hat{T}$.

Mathematically this was defined as:

$$\text{Last point difference} = ||\mathbf{p}_{last} - \hat{T}(time(\mathbf{p}_{last}))|| \quad (4.12)$$

where $\mathbf{p}_{last}$ denotes the last recorded point in the trajectory T, and $time(\mathbf{p})$ is the time which the point was recorded.

The values that are used in the tables are the mean of this metric over all the trajectories on a site.

4.8.4 Apex point difference

Given a trajectory T and a trajectory $\hat{T}$ predicted by a model, then the apex point difference was defined as the Euclidean distance between

the points with the maximum height (y-value) in each trajectory.

Mathematically this was defined as:

$$\text{Apex point difference} = ||MaxHeight(T) - MaxHeight(\hat{T})|| \quad (4.13)$$

where $MaxHeight(T)$ is the point with the maximum height for the trajectory T.

The values that are used in the tables are the mean of this metric over all the trajectories on a site.

4.9 Statistical significance

To determine the statistical significance of the result, several statistical methods were used.

4.9.1 Confidence intervals

To indicate the uncertainty in the means, confidence intervals were calculated for the means of the different metrics. For a given confidence level C and a set of values X_i, then the confidence interval was calculated as [34]:

$$(\bar{X} - t^* \frac{s}{\sqrt{n}}, \bar{X} + t^* \frac{s}{\sqrt{n}}) \quad (4.14)$$

where the $\bar{X}$ denoted the sample mean and s the sample standard deviation, calculated as:

$$\begin{aligned} \bar{X} &= \frac{1}{n} \sum_{i=1}^{n} X_i \\ s &= \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (X_i - \bar{X})^2} \end{aligned} \quad (4.15)$$

The value t^* is known as the critical value, and depended on the confidence level C and the number of samples (n) used to calculate the

mean. As the true standard deviation was not known, the t-distribution was used instead. Compared to a normal distribution, the t-distribution considers the uncertainty in the estimation of the standard deviation. [47] The t^* value was obtained from the `scipy.stats.t.ppf` Python function [39].

4.9.2 ANOVA

The ANOVA test [34] was used to determine if there exist any statistical significant differences between the mean prediction error rates produced by the predictors. From the result of the test, a p-value can be calculated which indicates that the null hypothesis (there exists no difference between the means) should be rejected if $p < \alpha$ where α denotes the significance level. This test was conducted using the `scipy.stats.f_oneway` Python function [38].

4.9.3 Post-hoc analysis

If the null hypothesis about the equality of the means is rejected, it must be established which pairs of predictors generated predictions of different accuracy. This is known as *post-hoc analysis* and Tukey's range test was used [34]. For each pair of predictors, the method computed a q-value. If this value was greater than the critical value q^* (which depend on the significance level and number of predictors), then the hypothesis that there exists no difference between the means should be rejected, that is, the means between a pair of methods differ significantly. This test was conducted using the `pairwise_tukeyhsd` function from the `statsmodels.stats.multicomp` Python package [42].

Chapter 5

Results

In the tables and figures presented in this chapter, the model names correspond to the following models:

- Existing model: the existing model (see section 2.2).
- MLP (Site x): the MLP (see section 4.5) trained on site x.
- RNN direct (Site x): the RNN with the direct prediction output layer (see section 4.6.1) trained on site x.
- RNN force (Site x): the RNN with the force prediction output layer (see section 4.6.1) trained on site x.

The following experiments were conducted:

- The effect that different number of LSTM layers for the RNN force model had on the accuracy, see section 5.1.
- The effect that the initialization of the RNN force model had on the accuracy, see section 5.2.
- The effect that the size of the training set had on the accuracy for the RNN force model, see section 5.3.
- The accuracy for models trained and evaluated on the same site, see section 5.4.
- The accuracy for models trained and evaluated on different sites, see section 5.5.
- The effect that using an already trained model as the initialization had on the accuracy for the RNN force model, see section 5.6.

- The effect that the amount of data sent as input during evaluation had on the accuracy, see section 5.7.
- The impact that the weather had on the accuracy, see section 5.8.
- The effect that noise had on the extrapolated trajectory, see section 5.9.
- The effect that different output layers had on the accuracy for RNN-based models, see section 5.10.
- The time required to extrapolate a trajectory, see section 5.11.

All results for the neural network based models were for the network that predicted forwards in time, as the data in the beginning of the trajectory was missing, which would have been needed for evaluating the performance of the network predicting backwards in time.

As mentioned in section 4.3, the number of training trajectories were 15 000, 1 000 used for the validation set and 3 000 for the test set. All models were trained for 1 000 epochs, using the method described in section 4.7. For the results presented in this section, each model was supplied 40 % of the overall trajectory as input, starting from the beginning of the recorded trajectory[1], and predicted the remaining 60 % of the trajectory.

The statistical tests were conducted for the mean of the different metrics used where each predictor was considered a group in the test. The statistical tests conducted used the significance level $\alpha = 0.05$. The confidence level for the confidence intervals were 95 %. The error bars presented in this chapter are the width of the confidence intervals, and for the tables presented in the appendices, $\mu \pm w$ denoted the confidence interval $(\mu - w, \mu + w)$ with the mean μ.

5.1 Hyperparameters

To determine how many LSTM layers were needed for the RNN model using the force prediction output layer, different number of layers were tested where each layer used 320 nodes. This experiment was only

[1]Keep in mind that the recorded trajectory does not begin at the beginning of the actual trajectory.

conducted for a model trained on site 1. The result from this experiment on the test set for the different metrics are shown in figure 5.1.

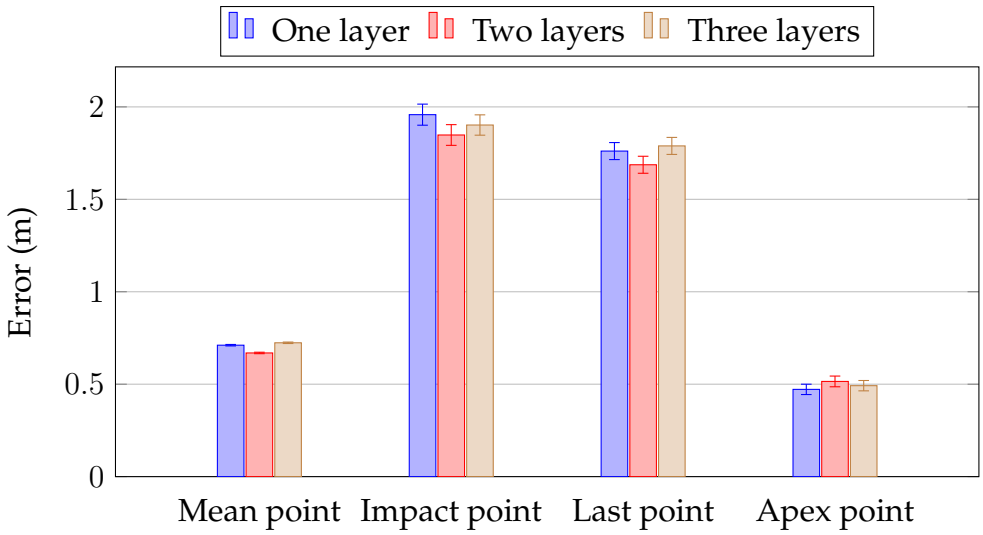


Figure 5.1: The result of the metrics for different network architectures for the RNN force model. The results are from the test set on site 1.

Table 5.1: The p-tests for the different metrics performed on the test set for site 1.

Metric	P-value
Mean point	10^{-89}
Impact point	0.026
Last point	0.007
Apex point	0.112

Table 5.2: Pair-wise comparisons using Tukey's range test for the mean point difference metric evaluated on test set for site 1. The critical value was 3.315.

Model #1	Model #2	q	Reject null	Desired
One layer	Two layers	20.876	True	Reject
One layer	Three layers	6.561	True	Reject
Two layers	Three layers	27.437	True	Reject

Table 5.3: Pair-wise comparisons using Tukey's range test for the impact point difference metric evaluated on test set for site 1. The critical value was 3.315.

Model #1	Model #2	q	Reject null	Desired
One layer	Two layers	3.815	True	Reject
One layer	Three layers	1.934	False	Reject
Two layers	Three layers	1.880	False	Reject

Table 5.4: Pair-wise comparisons using Tukey's range test for the last point difference metric evaluated on test set for site 1. The critical value was 3.315.

Model #1	Model #2	q	Reject null	Desired
One layer	Two layers	3.128	False	Reject
One layer	Three layers	1.203	False	Reject
Two layers	Three layers	4.331	True	Reject

From this figure, it can be seen that using two layers gave the best result for the mean point, impact point and last point difference metrics. This difference was statistically significant for the mean point difference metric compared to the other models (see table 5.2), but only against the model that used one layer for the impact point difference metric (see table 5.3) and against the model that used three layers for the last point difference metric (see table 5.4). For the apex point different metric, all models performed roughly equal, with no statistically significant differences (the p-value was greater than 0.05, see table 5.1).

5.2 Effect of parameter initialization

The parameters for the neural network based models were initialized from a random distribution. To study the effect that the initialization of the parameters had on the accuracy of the models, several models were trained. However, due to the computational time required to train each model, only the RNN model with the force prediction output layer was investigated, and only on site 1. The result from this experiment on the test set for the different metrics are shown in figure 5.2

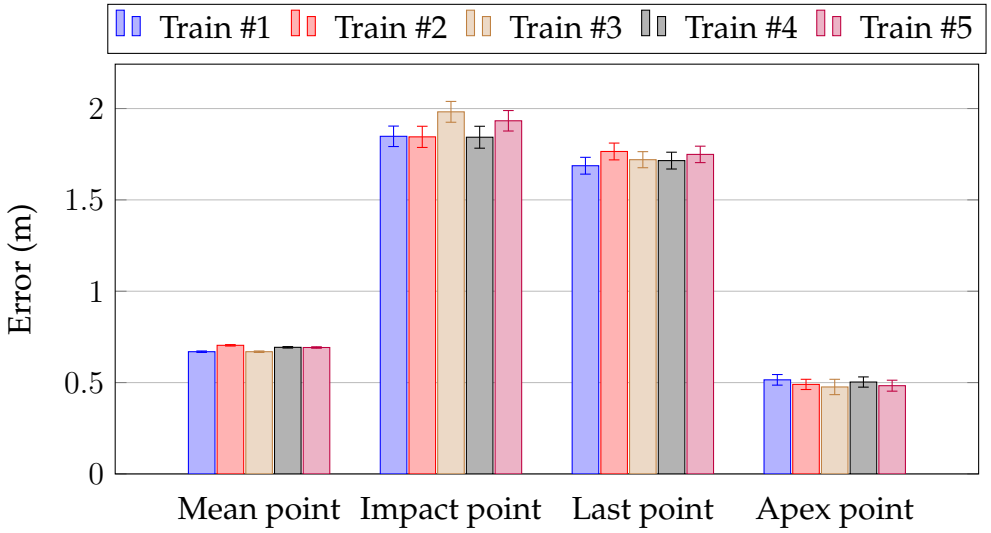


Figure 5.2: The result of the metrics for training the RNN force model with different random initializations. Train #1 denotes the first run, Train #2 the second run, etc. The results are from the test set on site 1.

Table 5.5: The p-tests for the different metrics performed on the test set for site 1.

Metric	P-value
Mean point	10^{-51}
Impact point	10^{-3}
Last point	0.147
Apex point	0.439

From table 5.5, it can be seen that there was a statistically significant difference between the trained models for the mean point and impact point difference metrics (as the p-values was less than 0.05). However, this difference was very small for the mean point metric. For the last point and apex point difference metrics, there was no statistically significance difference between the trained models (as the p-values was greater than 0.05).

5.3 Amount of training data

The following plots show how the performance of the RNN model using the force prediction output layer depended on the number of

trajectories used for training. Other than a different number of trajectories, the same network architecture and training procedures was used as mentioned in chapter 4. This experiment was only conducted for site 1. The results presented below are from the test set.

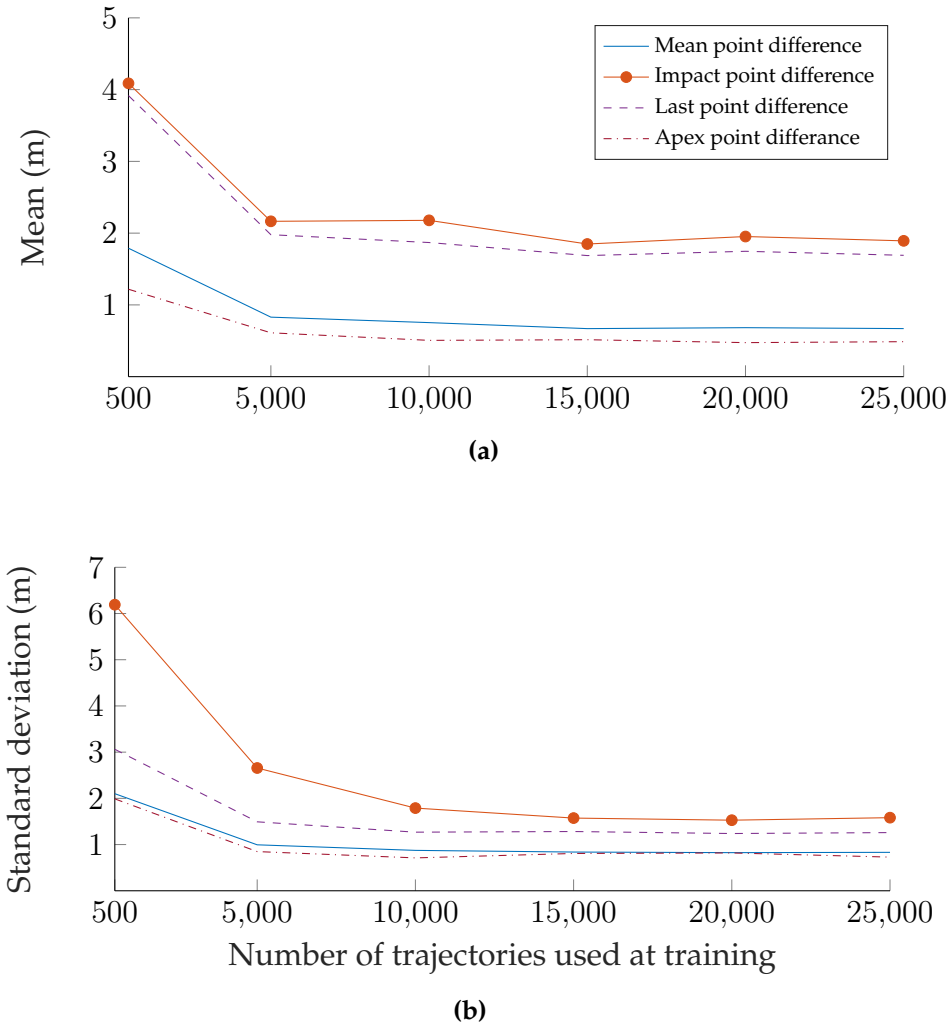


Figure 5.3: Plots showing how the performance of the model depended on the number of trajectories used during training. Figure a) shows the mean of the metrics as a function of the number of trajectories used during training, and figure b) the same for the standard deviation.

From these plots, it can be seen that using more than 10 000 trajectories had a marginal impact of the mean and variance of the metrics.

5.4 Results per site

In this section, the result for all sites are presented. The hyperparameters for all sites were the same and based on the experiments conducted for site 1 in the previous sections. The primary reason was due to the time required to train the models for each experiment. In addition, using the same network architecture makes it possible to use models from other sites as initialization (see section 5.6) when training a new model. This meant that the accuracy for other sites may not be optimal, but this is not taken into consideration, it was assumed that the same network architecture would be optimal for each site.

In figures 5.4–5.7, summaries are presented for the different metrics for models trained and evaluated on the same site. In these figures, *Site (1, 1)* denotes a model trained on site 1 and evaluated on the test set for site 1. For a metric, the values are given relative to the existing model on the site in question:

$$\text{Relative metric} = \frac{\text{Metric model}}{\text{Metric existing model}} \tag{5.1}$$

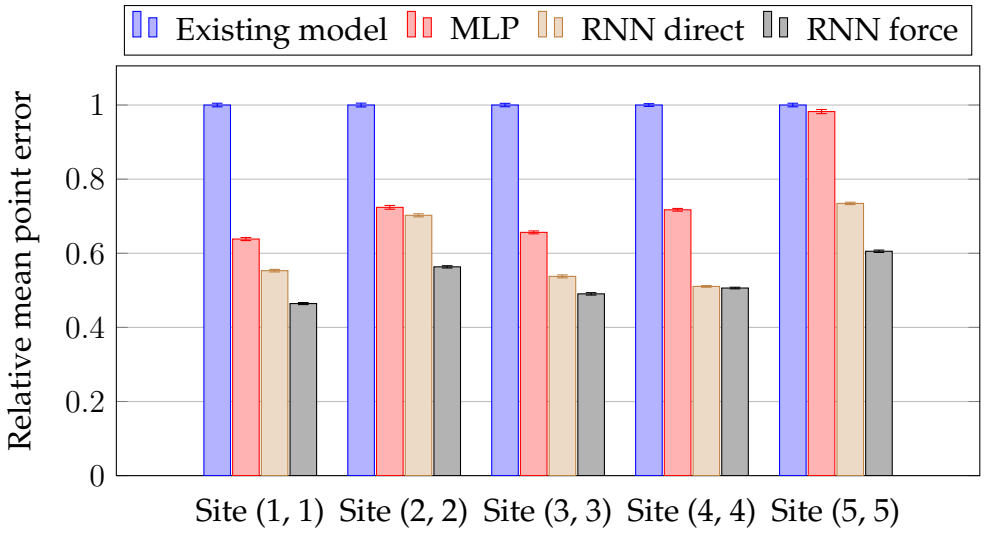


Figure 5.4: Summary of the mean point difference metric for models trained and evaluated on the same site. The results are from the test sets.

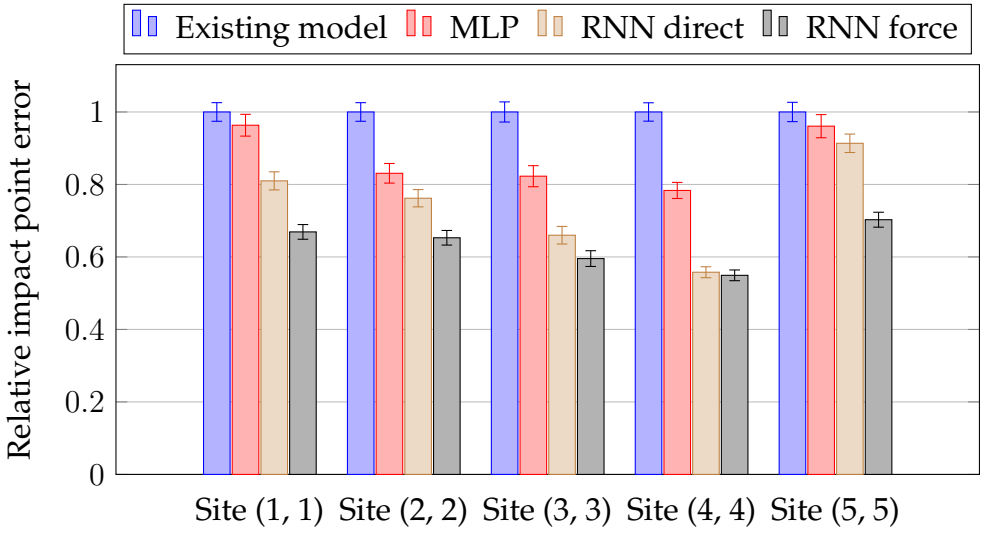


Figure 5.5: Summary of the impact point difference metric for models trained and evaluated on the same site. The results are from the test sets.

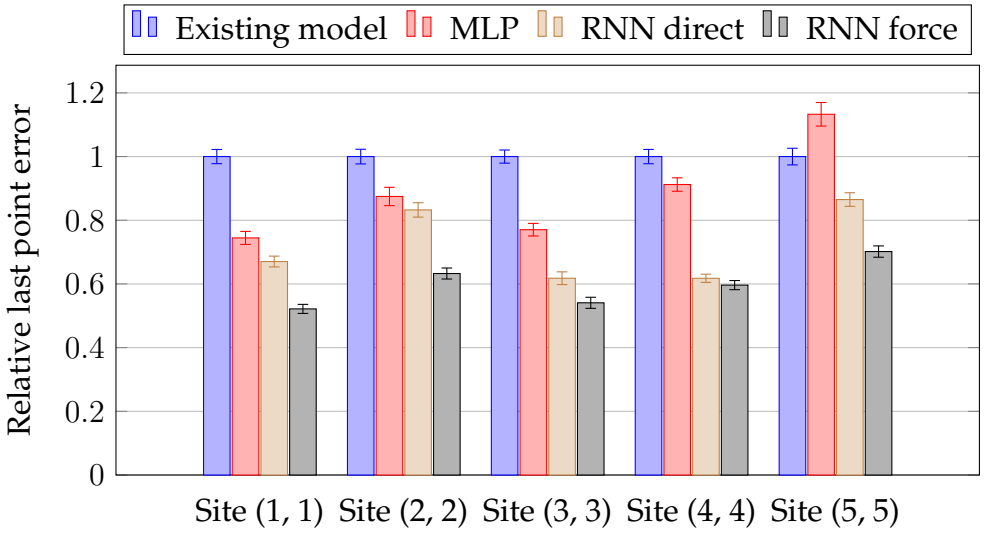


Figure 5.6: Summary of the last point difference metric for models trained and evaluated on the same site. The results are from the test sets.

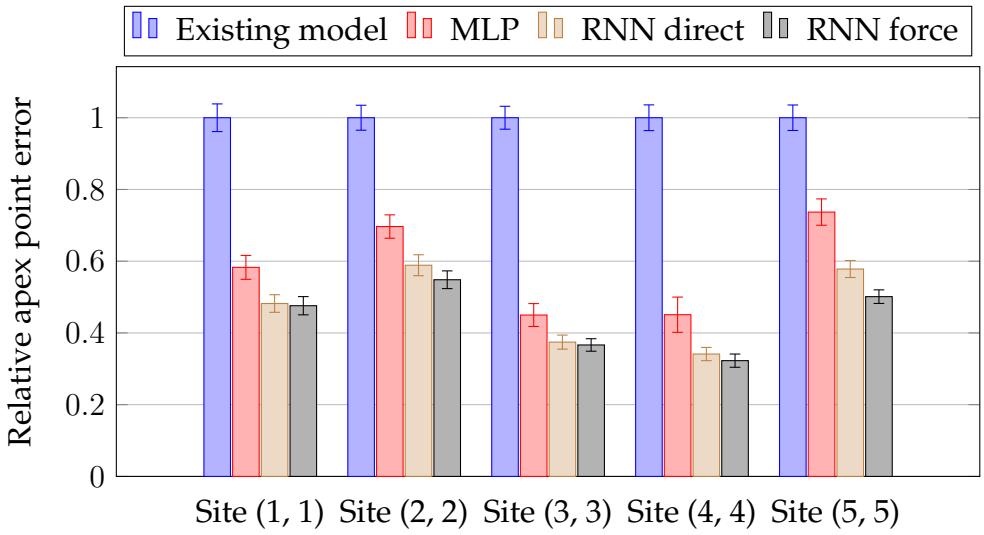


Figure 5.7: Summary of the apex point difference metric for models trained and evaluated on the same site. The results are from the test sets.

5.4.1 All sites

To study the general trend across sites, the mean of the metrics over all sites for a given model was calculated, and the results are shown in figure 5.8. For the tests that determined the statistical significance, see appendix B.2.

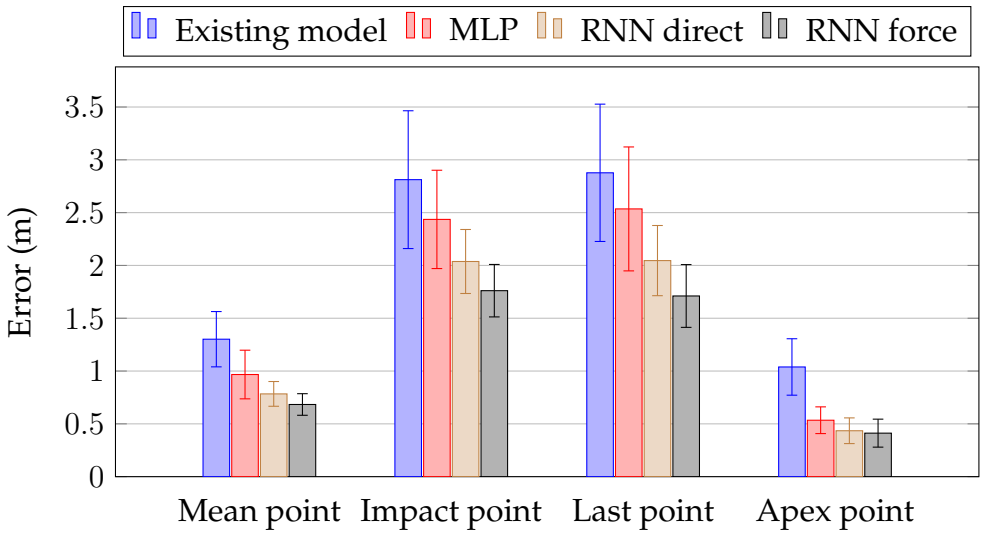


Figure 5.8: The result for all metrics averaged over the results for all sites for a model. The results are from the test set on the respective sites.

Mean point difference metric

The general trend for this metric was that the neural network based models performed better than the existing model, and the RNN-based models performed better than the MLP. However, the difference was not statistically significant between the MLP and RNN direct models. Between the RNN-based models, the RNN force performed better, but the difference was not statistically significant.

Impact and last point difference metrics

The trend for these metrics were similar as the trend for the mean point difference metric, except that the difference between the existing model and the MLP was not statistically significant.

Apex point difference metric

For this metric, the neural network based models performed roughly similar, with no statistically significant difference between them. However, all of these models performed better than the existing model.

5.4.2 Analysis for each site

In this section, the results for each site are analyzed with additional figures and statistical tests. The heatmaps presented in this section visualizes the spread of the error in the predicted impact position for a specific model and site. The blue dot denotes the expected impact position. As the impact position depended on the direction of the trajectory, the coordinate system was normalized as described in section 4.4.

Site 1

In this section, the result for models that were trained on site 1 are presented, and for the complete data, see appendix B.1.1. As can be seen in figure 5.9, the neural network based models performed better

than the existing model. It was only the case for the MLP on the impact point difference metric that the improvement was not statistically significant.

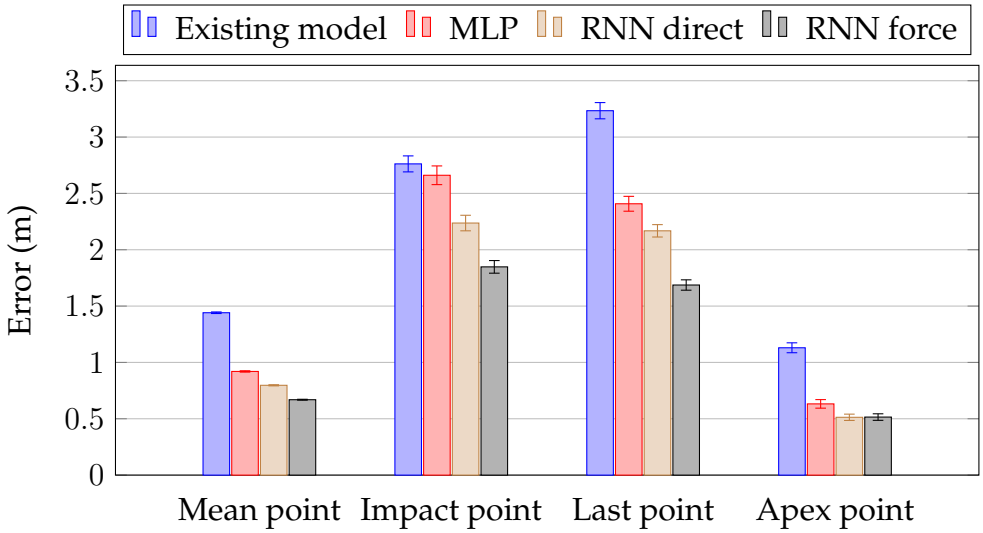
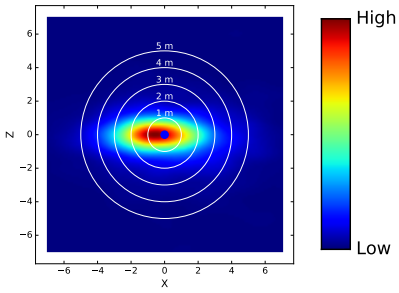
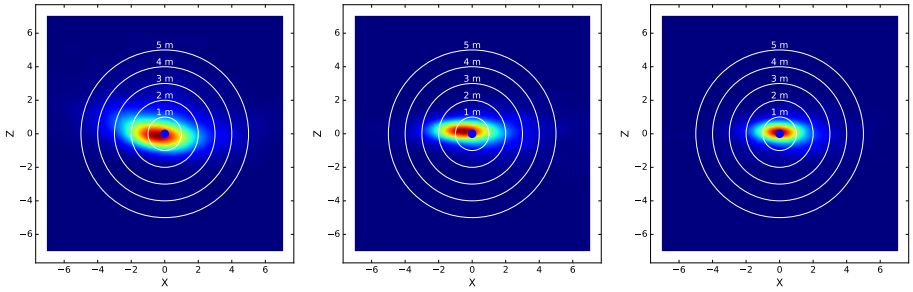


Figure 5.9: The results for the different metrics for models trained and evaluated on site 1. The results are from the test set on site 1.

Looking at the spread of the error in the predicted impact position (see figure 5.10), the spread was smaller for the neural network based models compared to the existing model, especially for the RNN-based models. For the center-of-density of the spread, it was only the case for the RNN force model that it aligned well with the expected impact position, the blue dot.



(a) Existing model.



(b) MLP (Site 1). **(c)** RNN direct. **(d)** RNN force.

Figure 5.10: The spread of the error in the impact position displayed as heatmaps on the test set for site 1.

Site 2

In this section, the result for models that were trained on site 2 are presented, and for the complete data, see appendix B.1.2. As can be seen in figure 5.11, the behavior for the models that were trained on this site were consistent with the models on site 1. However, one difference was that for the RNN force model, the center-of-density was not aligned with the expected impact position, which was the case for site 1 (see figure 5.12).

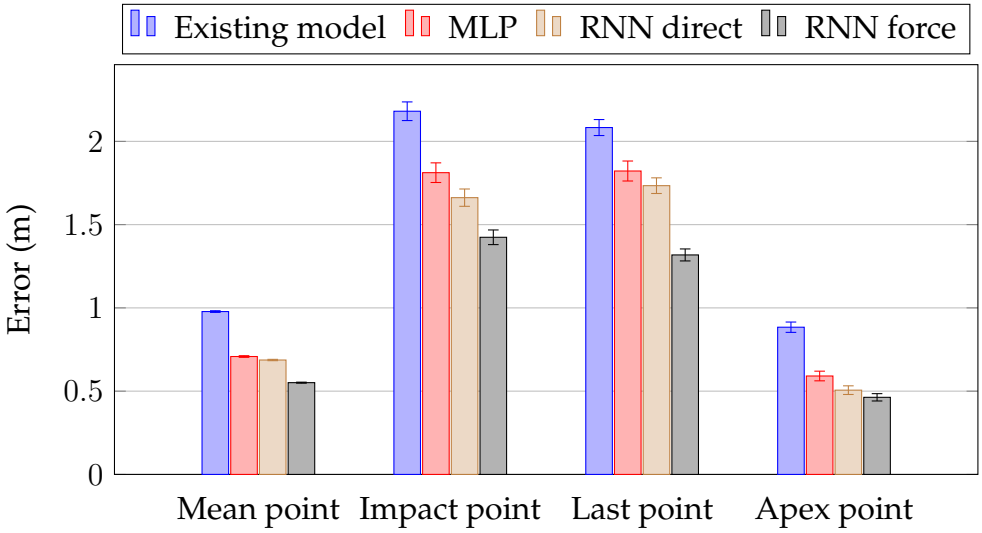
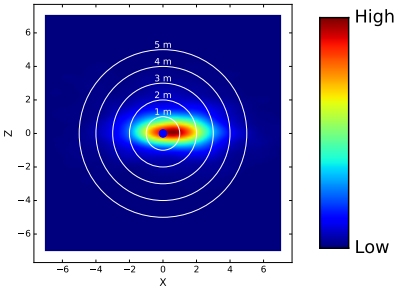
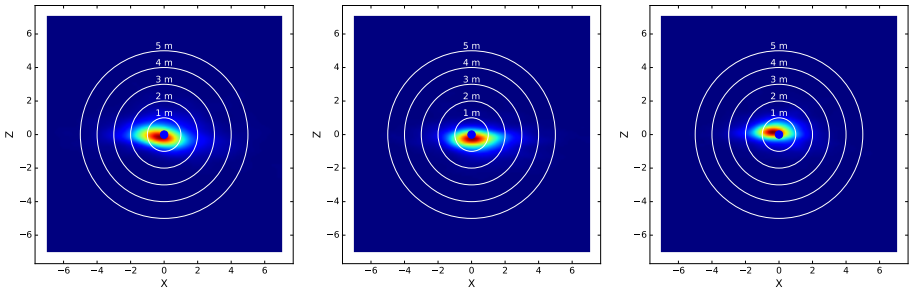


Figure 5.11: The results for the different metrics for models trained and evaluated on site 2. The results are from the test set on site 2.



(a) Existing model.



(b) MLP. **(c)** RNN direct. **(d)** RNN force.

Figure 5.12: The spread of the error in the impact position displayed as heatmaps on the test set for site 2.

Site 3

In this section, the result for models that were trained on site 3 are presented, and for the complete data, see appendix B.1.3. As can be seen in figures 5.13 and 5.14, the behavior for the models that were trained on this site were consistent with the models on site 1 and 2.

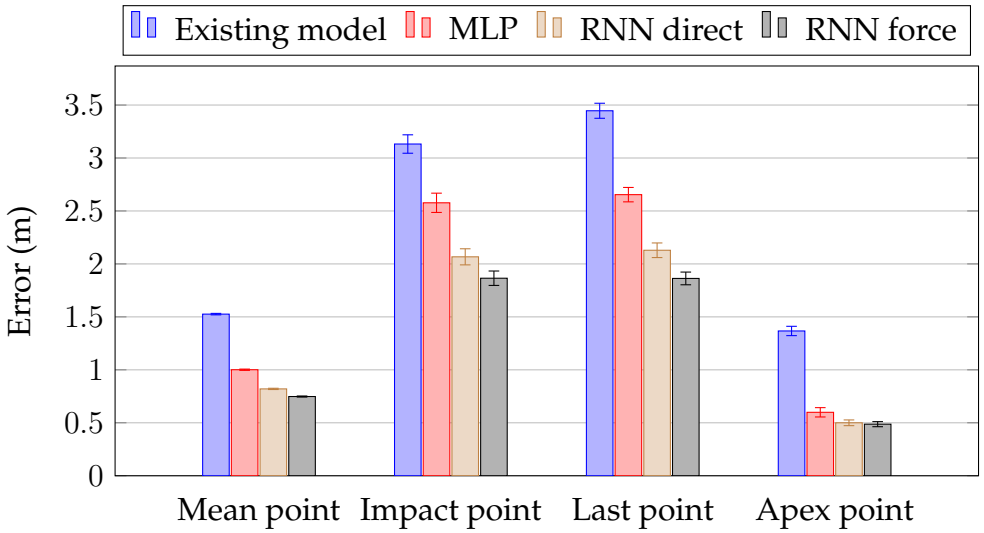
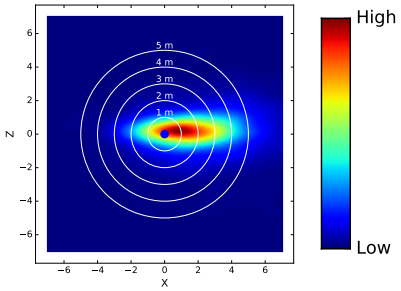
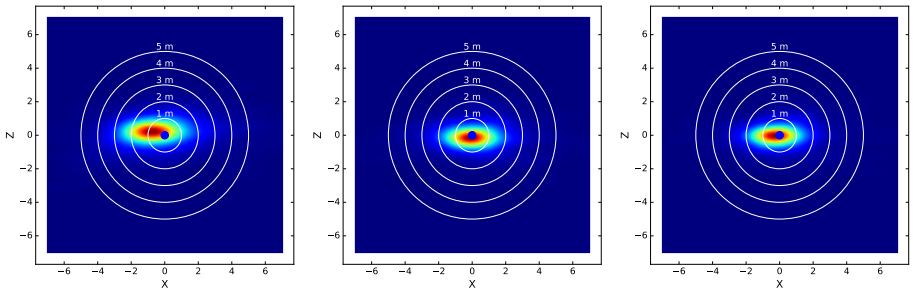


Figure 5.13: The results for the different metrics for models trained and evaluated on site 3. The results are from the test set on site 3.



(a) Existing model.



(b) MLP. **(c)** RNN direct. **(d)** RNN force.

Figure 5.14: The spread of the error in the impact position displayed as heatmaps on the test set for site 3.

Site 4

In this section, the result for models that were trained on site 4 are presented, and for the complete data, see appendix B.1.4. As can be seen in figure 5.15, the largest difference compared to the result on the other sites were that the accuracy of the RNN-based models was very similar, and the difference in accuracy was not statistically significant.

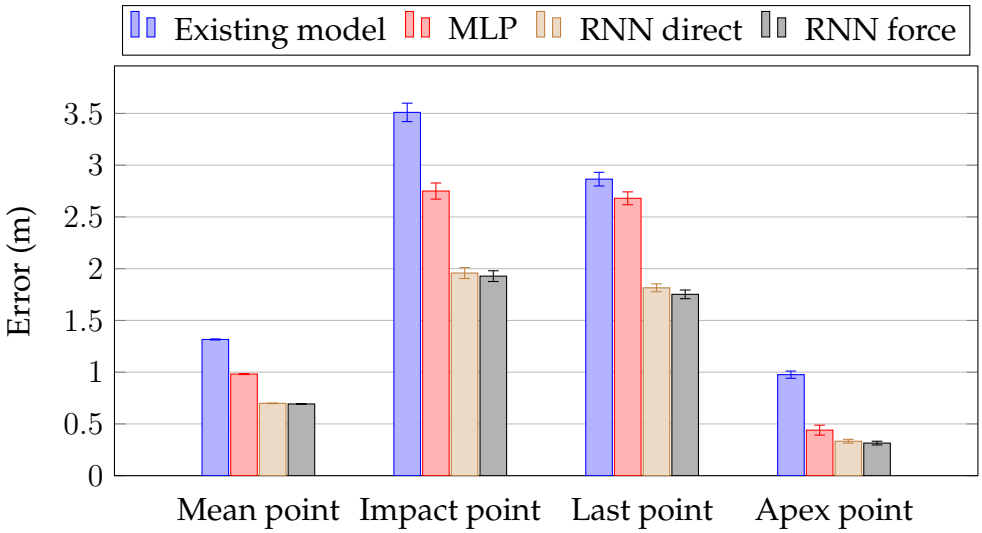
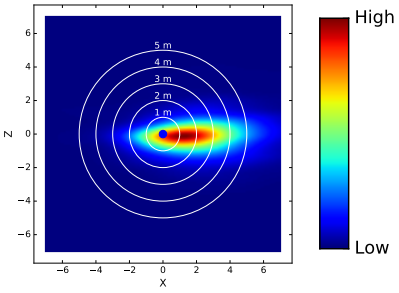
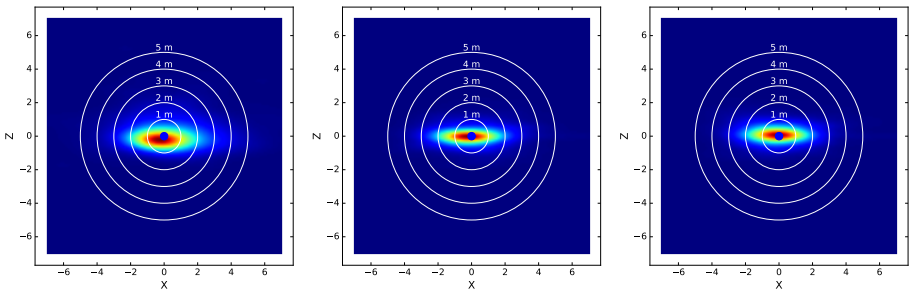


Figure 5.15: The results for the different metrics for models trained and evaluated on site 4. The results are from the test set on site 4.



(a) Existing model.



(b) MLP. **(c)** RNN direct. **(d)** RNN force.

Figure 5.16: The spread of the error in the impact position displayed as heatmaps on the test set for site 4.

Site 5

In this section, the result for models that were trained on site 5 are presented, and for the complete data, see appendix B.1.5. As can be seen in figure 5.17, the behavior on this site were consistent with the behavior on the other sites, except for the impact position. Comparing the difference between existing model and the MLP for the impact position, then the difference was not statistically significant. This was also the case when comparing between the MLP and the RNN direct models. Finally, for the spread of the predicted impact position (see figure 5.18), the center-of-density for the existing model was better aligned with the expected impact position, which was not the case for the other sites.

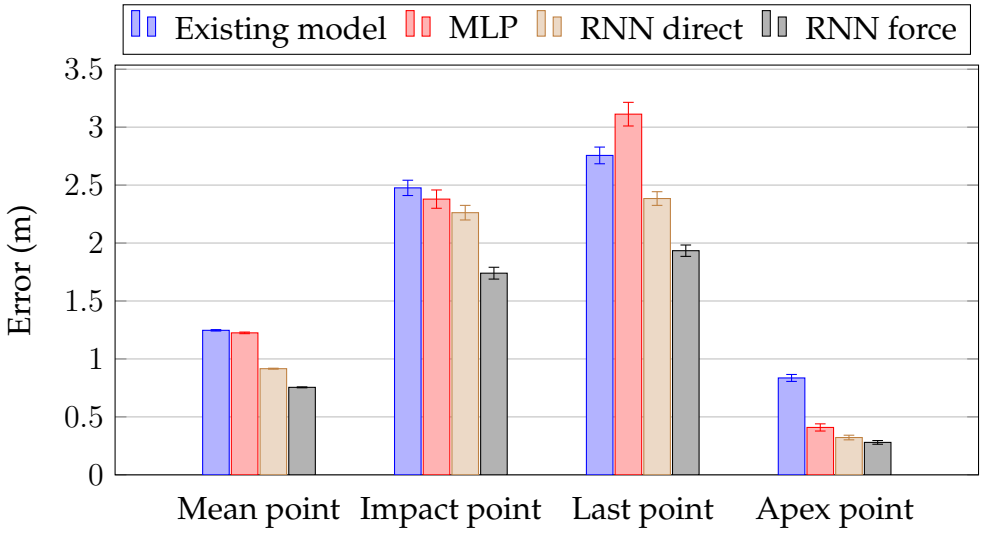
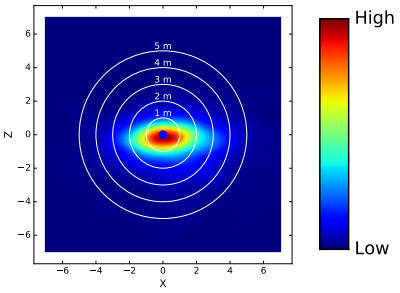
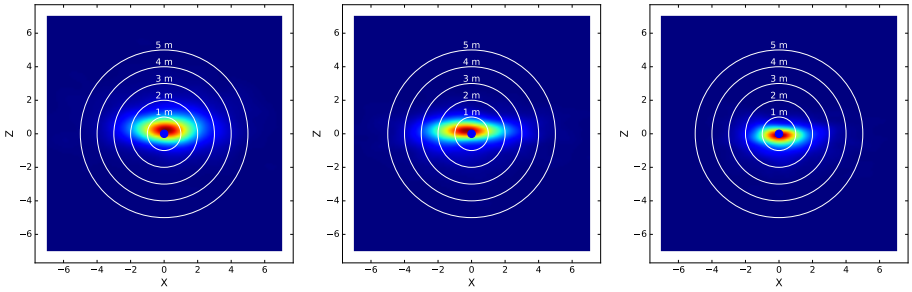


Figure 5.17: The results for the different metrics for models trained and evaluated on site 5. The results are from the test set on site 5.



(a) Existing model.



(b) MLP. **(c)** RNN direct. **(d)** RNN force.

Figure 5.18: The spread of the error in the impact position displayed as heatmaps on the test set for site 5.

5.5 Cross site generalization

In this section, the generalization capabilities across sites are presented, that is, taking a model trained on one site and evaluating it on a site it was not trained on. Due to the number of sites and models, results are only shown for models evaluated on site 1 in this section, and the complete data for all sites are presented in appendix B.1.

The results for the different metrics are shown in figures 5.19–5.22. In general, the behavior of the metrics were similar, with the models trained with the same time step (site 2 and 3) generalizing well, performing better than the existing model. However, for the models trained with a different time step (site 4 and 5), the models generalized poorly, not leading to any improvements in accuracy compared to the existing model.

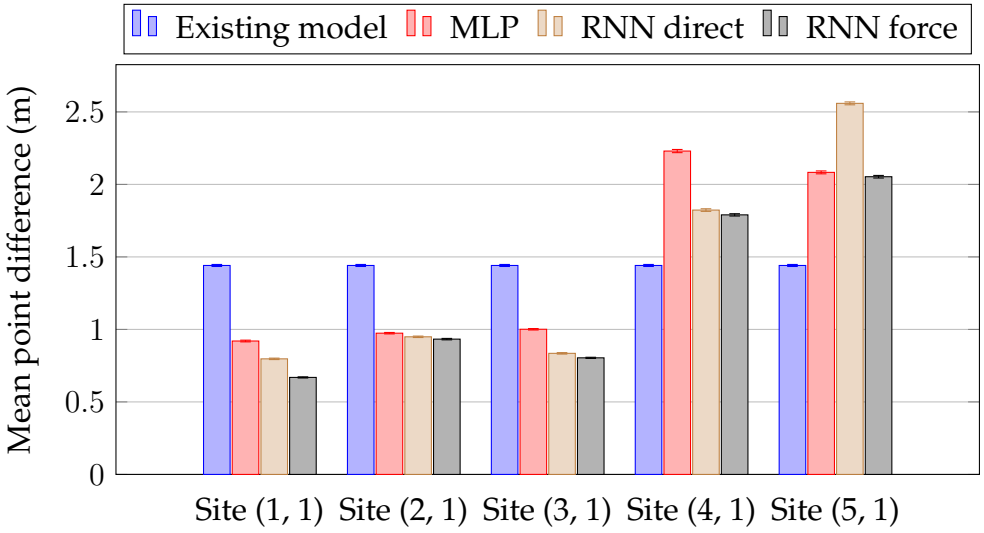


Figure 5.19: The results for the mean point difference metric for models trained on other sites. The results are from the test set on site 1.

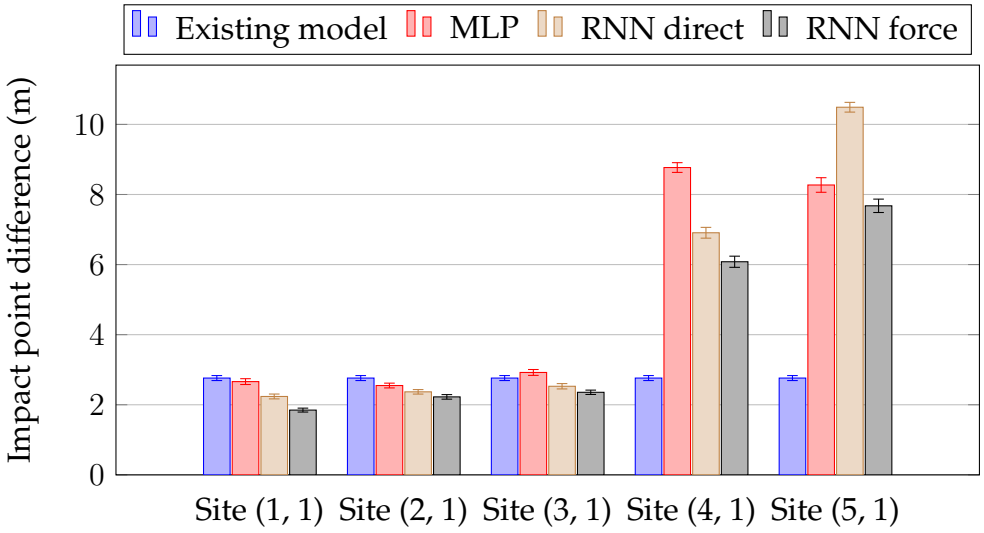


Figure 5.20: The results for the impact point difference metric for models trained on other sites. The results are from the test set on site 1.

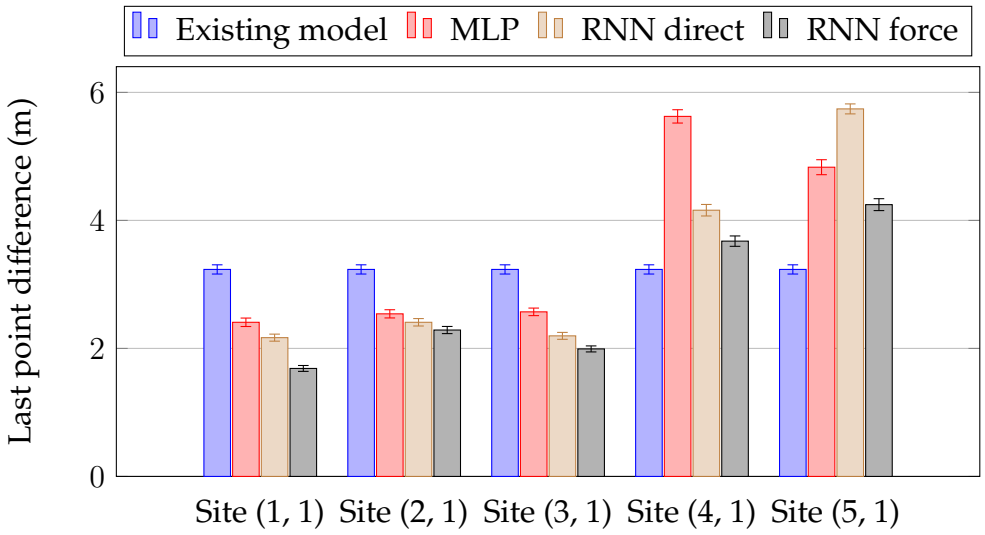


Figure 5.21: The results for the last point difference metric for models trained on other sites. The results are from the test set on site 1.

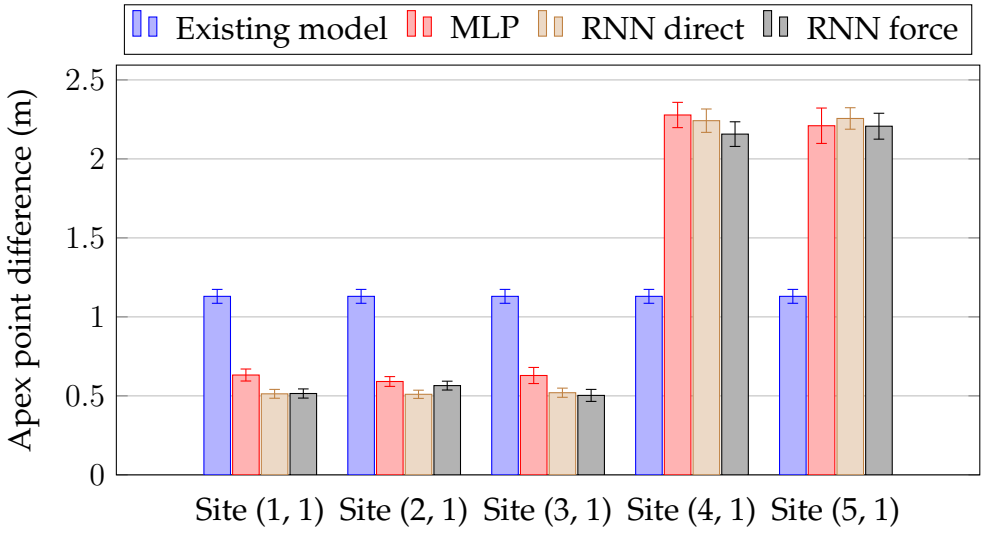
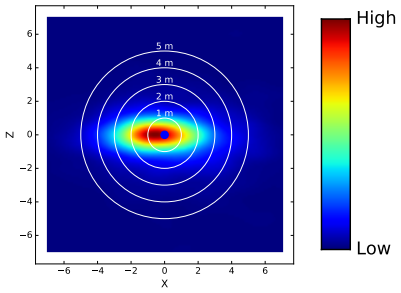


Figure 5.22: The results for the apex point difference metric for models trained on other sites. The results are from the test set on site 1.

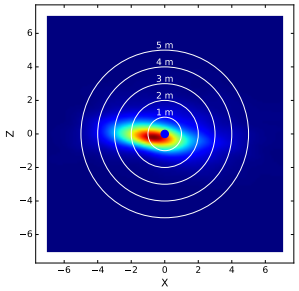
5.5.1 Spread of the impact position

In this section, the spread of the impact position for models trained on the other sites are presented. As shown in figure 5.23, the spread for the models trained on sites with the same time step as site 1 (5.23b–

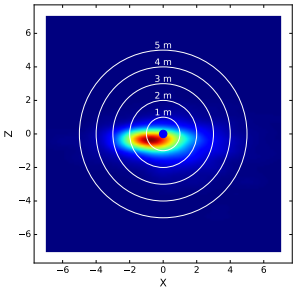
5.23g) was smaller than the existing model, but having an offset in some direction for the center-of-density. For the models trained with a different time step (5.23h–5.23m), the spread was very large with a large offset for the center-of-density.



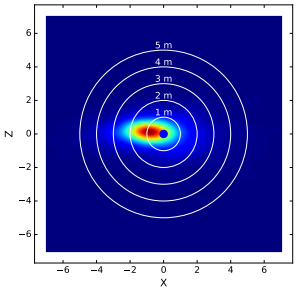
(a) Existing model.



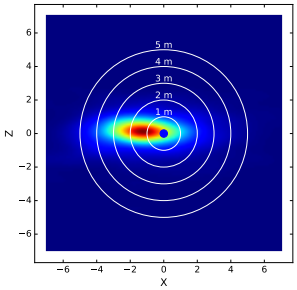
(b) MLP (Site 2).



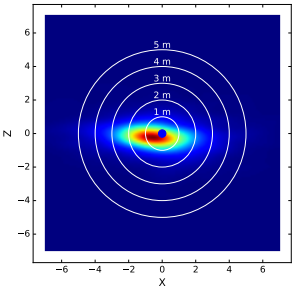
(c) RNN direct (Site 2).



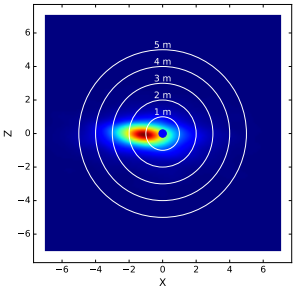
(d) RNN force (Site 2).



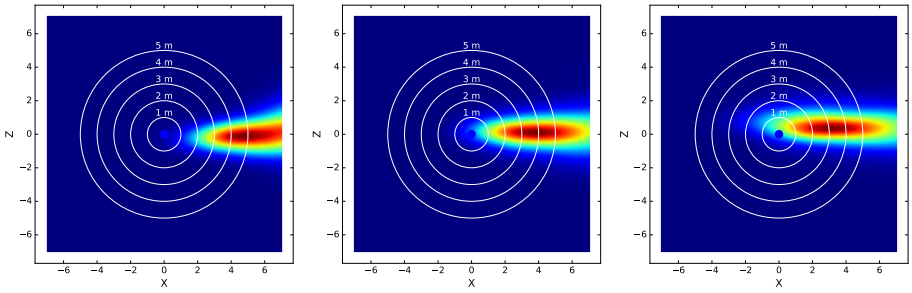
(e) MLP (Site 3).



(f) RNN direct (Site 3).



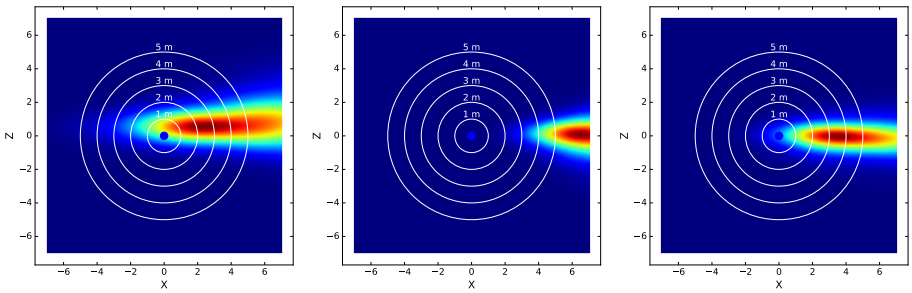
(g) RNN force (Site 3).



(h) MLP (Site 4).

(i) RNN direct (Site 4).

(j) RNN force (Site 4).



(k) MLP (Site 5).

(l) RNN direct (Site 5).

(m) RNN force (Site 5).

Figure 5.23: The spread of the error in the impact position displayed as heatmaps on the test site for site 1.

5.6 Pretraining

When training a model on a new site, there exist two main problems: there does not exist enough training data or it takes too long to train a new model from scratch. An alternative is then to initialize the model from an already trained model.

To demonstrate this, new models were trained with only 500 trajectories, but instead initialized from an existing model, the model trained on site 2. The reason for not choosing the model trained on site 1 (which most of the experiments were conducted for) was to compare the model with pretraining to a model where the hyperparameters had been carefully chosen based on experiments. The choice for the model from site 2 was then arbitrarily. Secondly, this experiment was only conducted for the RNN model using the force prediction output layer.

In the following figures, "No pretraining" corresponded to a model initialized at random, and trained with 500 trajectories, and "Pretraining" a model initialized from an existing model (and trained with 500 trajectories). As reference (denoted "RNN force (Site x)" for site x in the figures), the performance of a model trained with all the data from the site and initialized at random was used. The training procedure and network architecture were the same as in the other experiments.

The results for a model retrained on site 1 are shown in figure 5.24 and in figure 5.25 for a model retrained on site 4. For the tests that determined the statistical significance of the results, see appendix B.3.

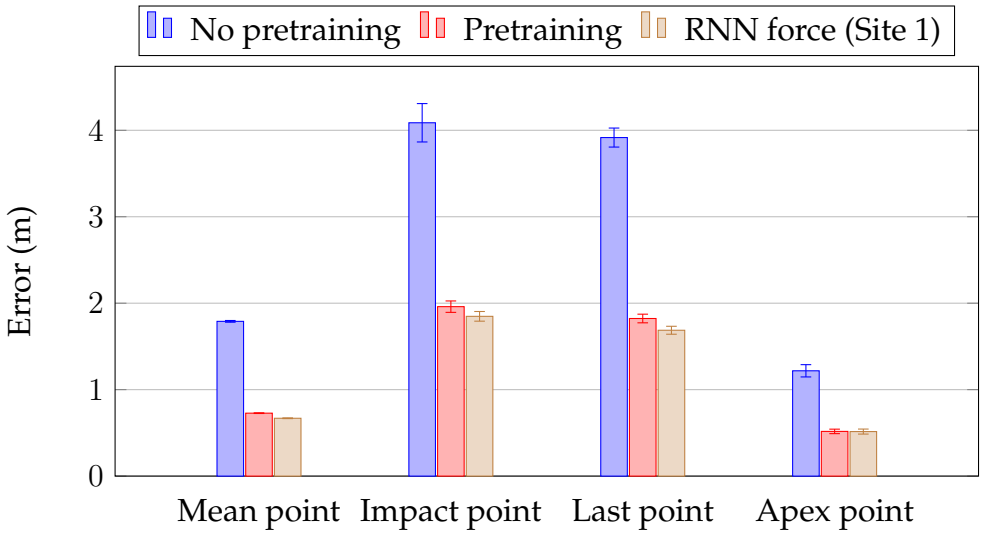


Figure 5.24: The results for the different metrics for models initialized with pretraining compared to models without pretraining. The results are from the test set on site 1.

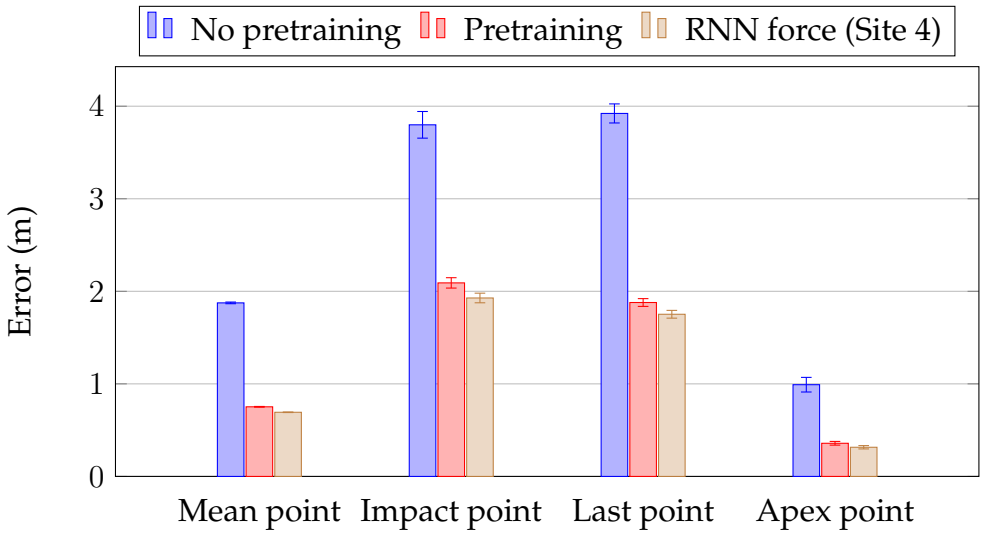
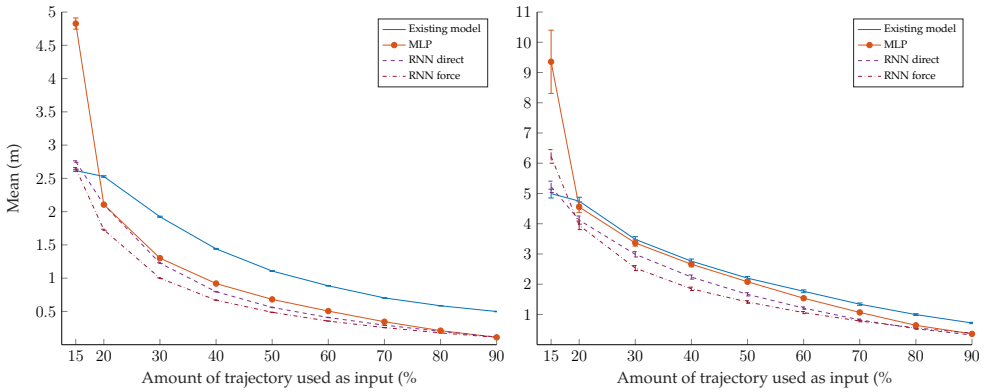


Figure 5.25: The results for the different metrics for models initialized with pretraining compared to models without pretraining. The results are from the test set on site 4.

From these figures, it can be seen than using pretraining greatly improved the accuracy of the model, compared to a model initialized at random with the same amount of training data, not too far from the model that was initialized at random and trained with all the data on the site in question. This was also the case if the model that was used as initialization used a different time step (as can be seen in section 5.5 performed poorly on a site with a different time step), that is, the result for a model retrained on site 4.

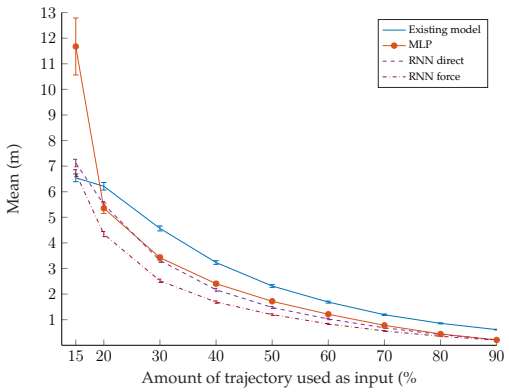
5.7 Amount of input data

The following plots show how the models performed based on how much of the trajectory was sent as input during evaluation. This experiment was only conducted for models trained on site 1 and evaluated on the test set for site 1.

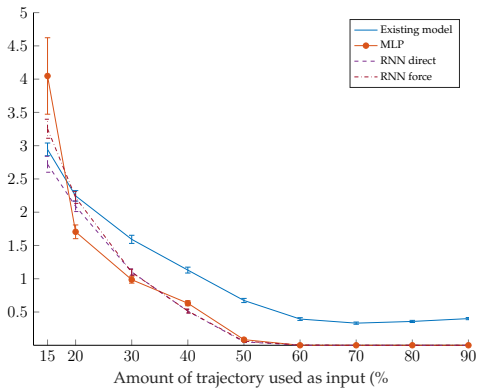


(a) Mean point difference.

(b) Impact point difference.



(c) Last point difference.



(d) Apex point difference.

Figure 5.26: Plot showing how the performance of the models depended on the amount of the trajectory that was sent as input during evaluation.

From these figures, it can be seen that the neural network based models used the available input data better than the existing model, requiring less input data in order to get the same level of accuracy. It is interesting that even if almost the entire trajectory was given as input, the existing model struggles to estimate the position of the apex point. It is worth pointing out that the existing model generated a new trajectory based on the input data, while the neural network based models only extrapolated it. This meant as more input data was sent as input, the network needed to extrapolate less of the trajectory.

5.8 Weather conditions

To study the effect that the weather had on the accuracy of the models, the evaluation data was partitioned into different parts depending on the different weather conditions. This information was not incorporated into the model (not sent as input), but only used for evaluation purposes. Two weather conditions were defined: low wind if the daily average wind speed was less than $5\,\mathrm{m/s}$ and high wind otherwise. Results are only shown for sites that had any variation on this definition. Note that the weather data was obtained from third party sources (online weather sites), which did not contain which direction the wind was from the perspective of the trajectory nor the actual speed. This meant that it was unlikely that the weather data correspond perfectly to the actual wind conditions.

5.8.1 Site 2

This site contained:

- Low wind: 677 validation and 1909 test.
- High wind: 323 validation and 1091 test.

For the results presented below, all models were trained on site 2 with all the data (i.e. both high and low wind at the same time).

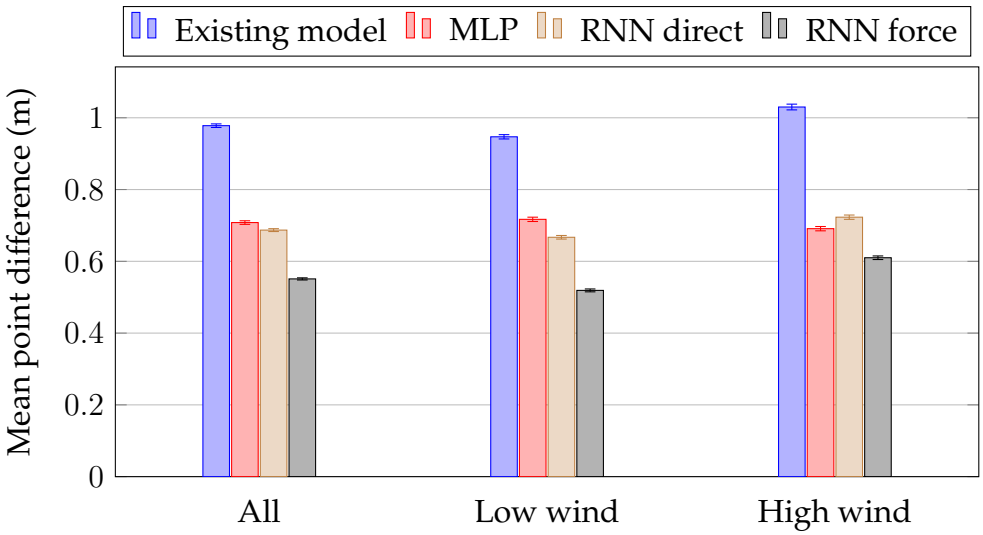


Figure 5.27: The results for the mean point difference metric for different weather conditions. The results are from the test set on site 2.

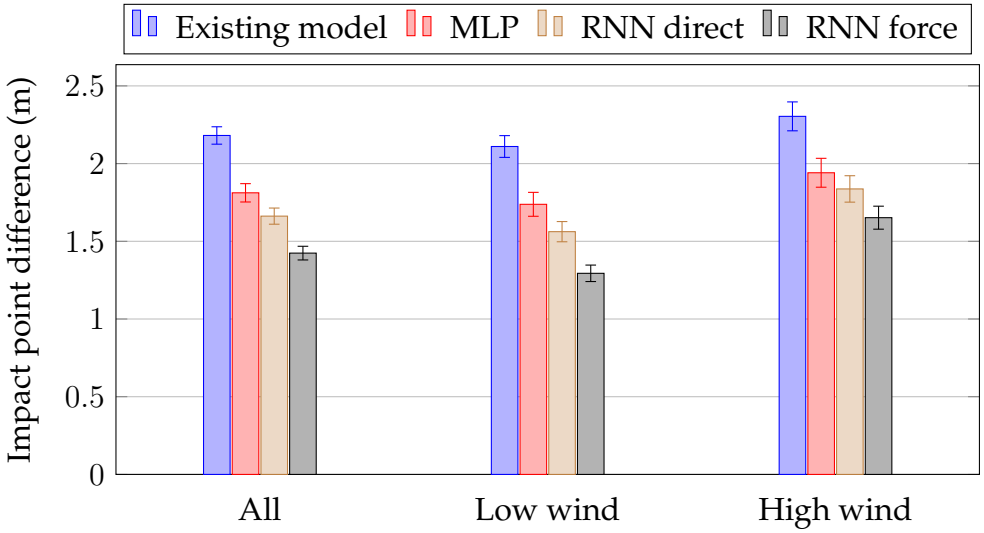


Figure 5.28: The results for the impact point difference metric for different weather conditions. The results are from the test set on site 2.

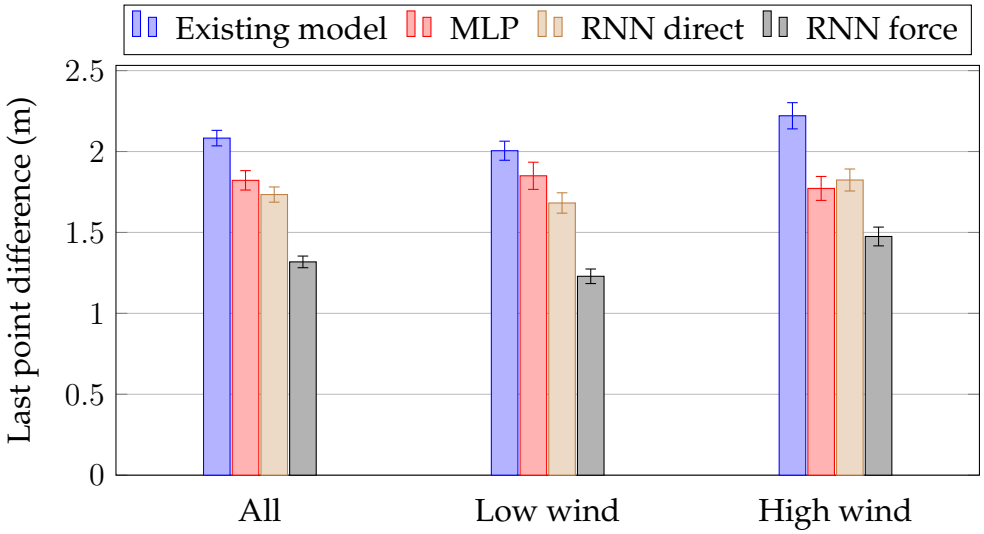


Figure 5.29: The results for the last point difference metric for different weather conditions. The results are from the test set on site 2.

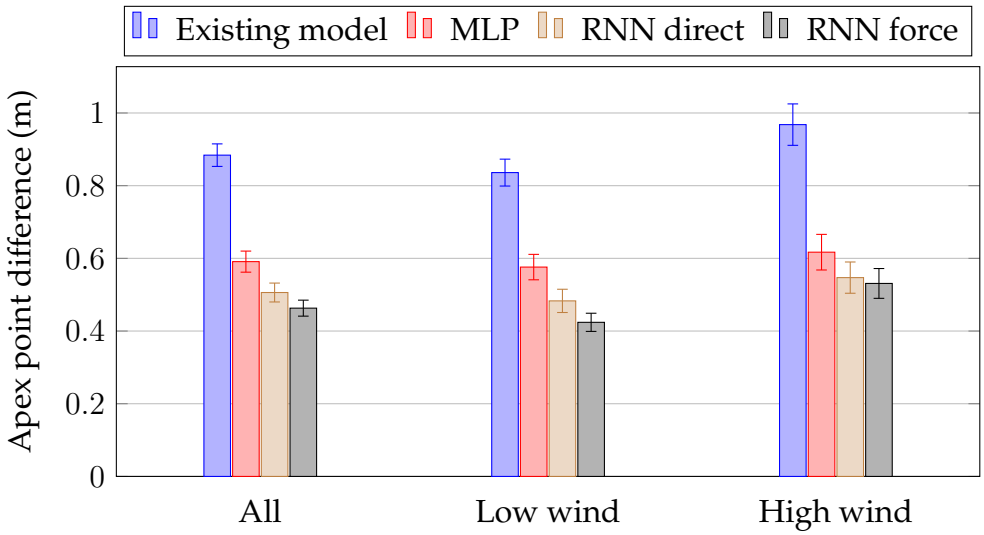


Figure 5.30: The results for the apex point difference metric for different weather conditions. The results are from the test set on site 2.

Table 5.6: The p-values for the means computed between the result with low and high wind for a model. The results are from the low/high wind test sets on site 2.

Metric	Existing model	MLP	RNN direct	RNN force
Mean point	10^{-66}	10^{-16}	10^{-39}	10^{-167}
Impact point	0.001	0.020	10^{-6}	10^{-14}
Last point	10^{-5}	0.057	0.005	10^{-10}
Apex point	10^{-4}	0.565	0.019	10^{-5}

Comparing the result with low wind against the result with high wind for each model (see figures 5.27–5.30), the wind led to statistically significance difference between the means for all models (see table 5.6). It was only the case for the MLP on the apex point difference metric that the difference was not statistically significant. For the MLP on the last point difference metric, the model actually performed better with high wind compared to low wind. However, this difference was not statistically significant. Comparing between the metrics, the impact position was the metric where the wind had the largest impact. Looking at the models individually, then the RNN force model was the model that was affected the most by the wind.

5.9 Effect of noise

To study the effect that noise had on the extrapolated trajectory during evaluation, artificial noise was added to the input before performing the extrapolation. In each point, normal distributed noise was added to each component, with zero mean and standard deviation 0.025. The result from this experiment is presented in figure 5.31 and are for models trained (without added noise) and evaluated (with added noise) on site 1.

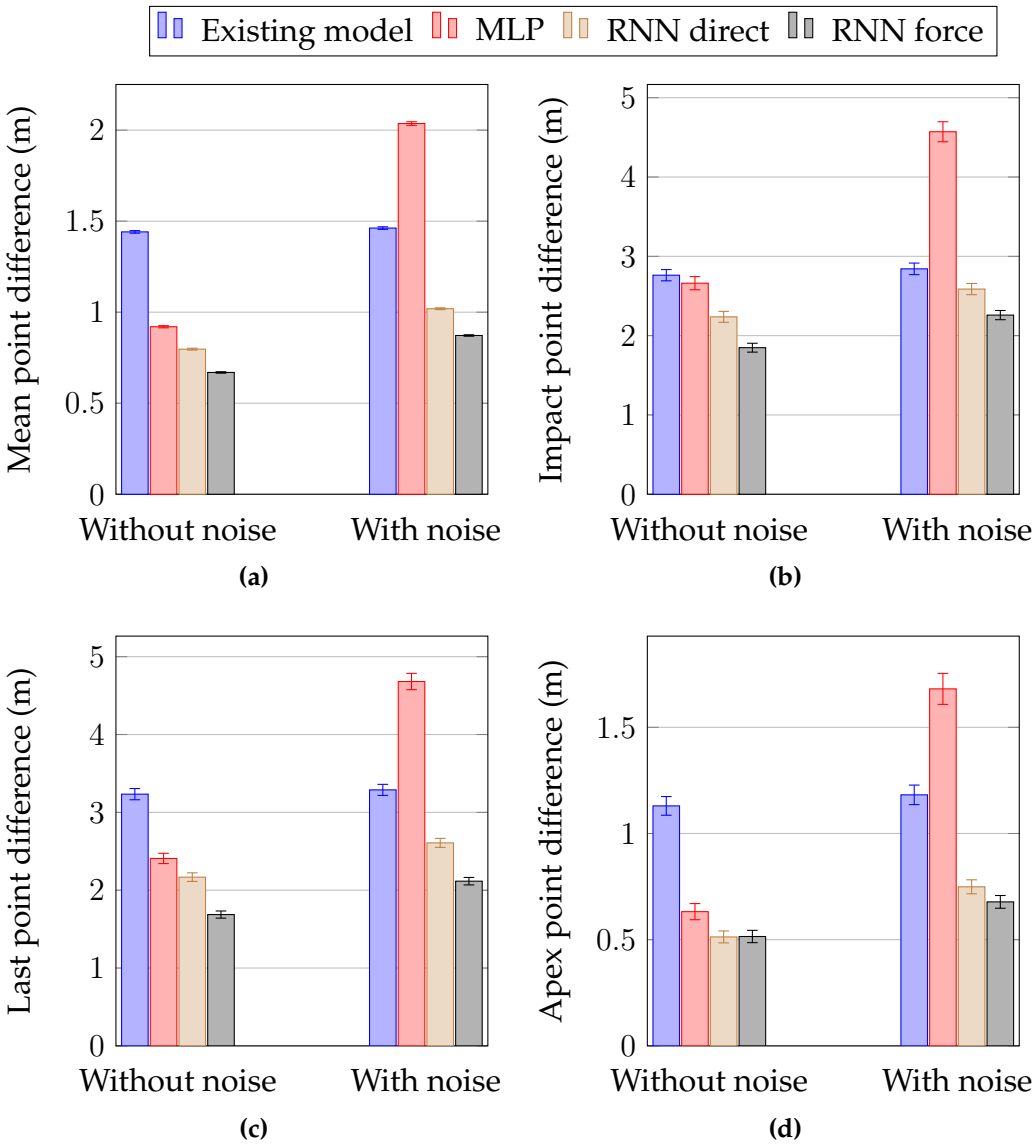


Figure 5.31: The results for the different metrics when artificial noise was added to the input to the models during evaluation. The results are from the test set on site 1.

Table 5.7: The p-values for the means computed between the result without and with added noise for a model. The results are from the test set on site 1.

Metric	Existing model	MLP	RNN direct	RNN force
Mean point	10^{-5}	0	0	0
Impact point	0.124	10^{-129}	10^{-11}	10^{-22}
Last point	0.280	10^{-254}	10^{-26}	10^{-36}
Apex point	0.112	10^{-130}	10^{-26}	10^{-14}

Comparing the results with noise to the result without noise, the existing model was the model that was most robust against noise, in the sense that the result was only marginally affected by the added noise. As seen in table 5.7, this did only result in a statistically significant difference for the mean point metric. All neural network based models were affected greatly by the added noise, with the differences being statistically significant for all the metrics (see table 5.7). Compared between the models, the MLP was the neural network based model that was affected the most by the added noise. The effect that the noise had on the extrapolated trajectory can be seen in figures 5.32–5.34, where the noise still existed in the extrapolated part for the MLP, but not for the RNN-based models.

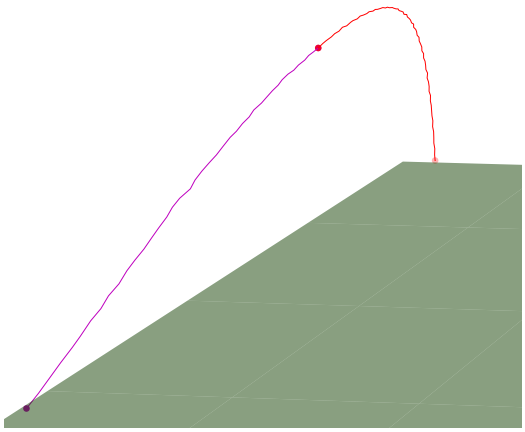


Figure 5.32: The result of extrapolating a noisy trajectory using the MLP trained on site 1. The purple line denotes the input, and the red line the extrapolated part.

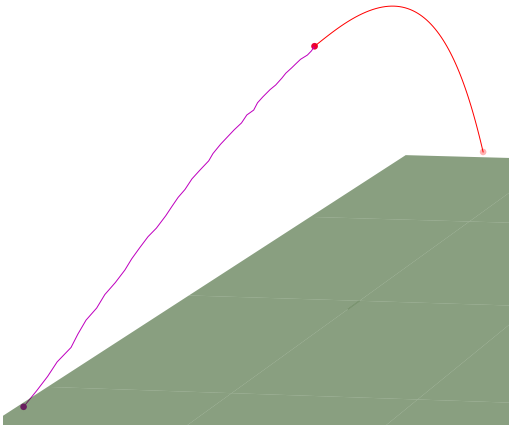


Figure 5.33: The result of extrapolating a noisy trajectory using the RNN with the direct prediction output layer trained on site 1.

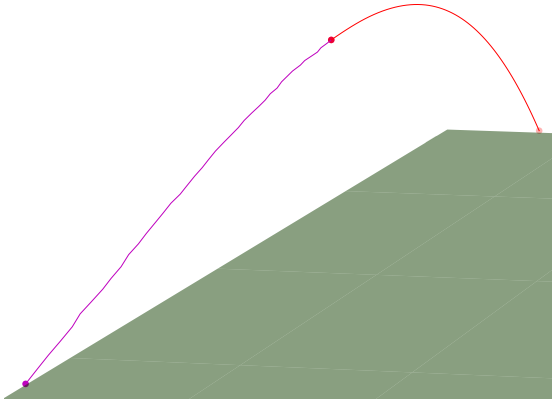


Figure 5.34: The result of extrapolating a noisy trajectory using the RNN with the force prediction output layer trained on site 1.

5.10 Meaning of force prediction output layer

One possible explanation for the effect that the force prediction output layer (see section 4.6.1) had on the predictions were that the layer acted as a loss function, constraining the form of the output. To better study this effect, an alternative output layer was tested, that assumed a specific form of the output. This output layer consisted of a second-degree polynomial. The motivation behind this was that in the physical model (see 2.1 and 4.6.1), many variables had a squared

impact on the final predictions. This output layer was defined as the following:

$$\begin{aligned} \Delta x(t) &= aw_0^2 + bw_0 + c_0 \\ \Delta y(t) &= aw_1^2 + bw_1 + c_1 \\ \Delta z(t) &= aw_2^2 + bw_2 + c_2 \end{aligned} \tag{5.2}$$

where $a, b, c_0, c_1, c_2, w_0, w_1, w_2$ were parameters that the network predicted at each step.

This model was trained with the same procedure as the other experiments. For the results presented below, all models were trained and evaluated on site 1. The result from this experiment is shown in figure 5.35. For the tests that determined the statistical significance of the results, see appendix B.4.

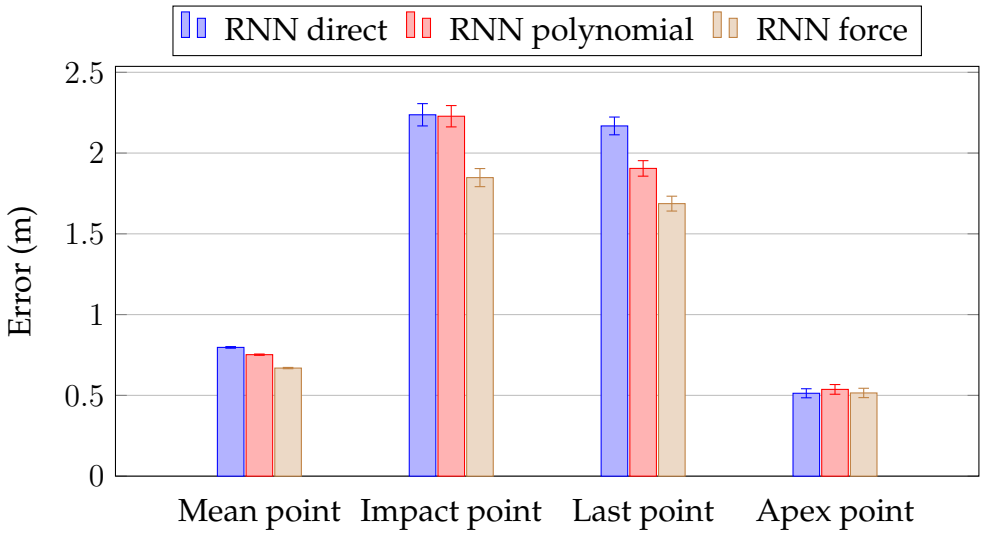


Figure 5.35: The results for the different metrics for different RNN models. The results are from the test set on site 1.

From this figure, it can be seen for the impact point difference metric and the apex point difference metric assuming a specific form of the output without explicit physical constraints does not lead to any improvements compared to predicting the displacement directly. For the mean point difference metric and last point difference metric, this results in a statistically significant improvement compared to predicting the displacement directly, but not improvements compared to the force prediction output layer.

5.11 Computational time for extrapolation

The following table show how the neural network based models compared to the existing model in terms of the time required to extrapolate a partial trajectory to a full trajectory. For this experiment, each model was given 4 000 trajectories, and the total time it took to extrapolate all trajectories were measured. In the experiment, each model was supplied 40 % of the trajectory as input, and the number of points sent as input affect the overall computational time.

Table 5.8: The results of predicting 4 000 trajectories, averaged over three runs (time in seconds).

Model	**Mean**	**Std.**	**Shots/second**
Existing model	326.4	3.000	12.3
MLP	69.1	0.276	57.9
RNN	205.9	0.456	19.4

Chapter 6

Discussion

In this chapter, different features and properties of the models are discussed, but also the method used for the evaluation is analyzed.

6.1 Analysis of metrics

For this problem, there are several things that can be measured to determine if one model was better than the other. Each metric used in this thesis had weaknesses, and these are now discussed. The mean point metric computed the metric as the average over all the errors. However, the error in the first predicted position was much lower than the last predicted position, as the errors propagated from the first to the last position. This made the spread of this metric hard to visualize.

The impact point difference metric cannot always be calculated, since the impact point may not actually be recorded, this metric was calculated by extending the reference trajectory following the velocity of the last recorded position. If the trajectory needed to be extended for many steps, then the metric did not actually measure the impact position. In addition, the metric did not consider if the ball impacted at the correct time, as only the position of the impact point was considered. The spread of this metric could easily be visualized as heatmaps.

The last point difference metric solved one of the problems with the impact point metric, by considering that the last position, which in ideal cases would correspond to the impact position, was predicted

at the correct position and time. This metric, however, depended on how much of the trajectory was captured (and was lower if less of the trajectory was captured). This made it more complicated to visualize the spread of the last point, compared to the spread of the impact point.

The last metric, the apex point difference, could easily be interpreted and did not depend on how much of the trajectory that was captured (given that the trajectory contained the apex point, which was the case for all trajectories used in this thesis), however, this metric only considered one point roughly in the middle of the trajectory. In other words, it did not consider what happened after the highest point in the trajectory had been reached.

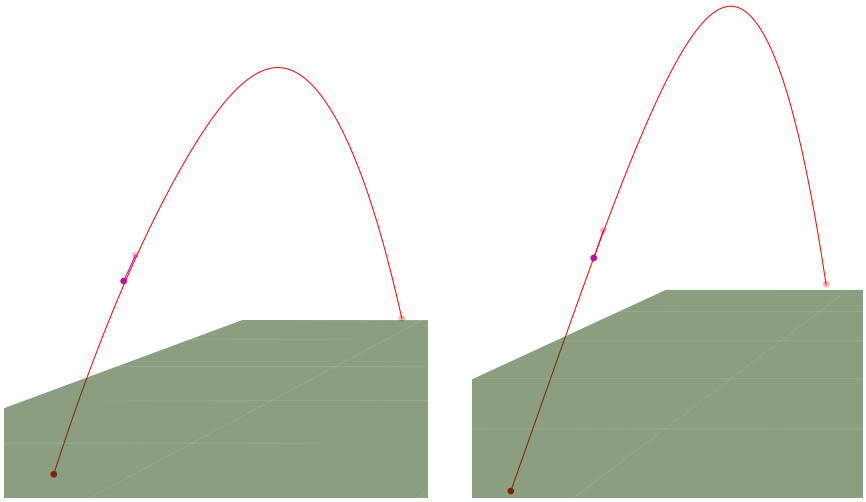
In conclusion, having more than one metric made the analysis of the performance of a model more complicated, as more variables must be considered, but was crucial to capture the true performance.

6.2 Comparing MLP and RNN

As can be seen in the chapter 5, the accuracy of a RNN-based model was much better than a model that used a MLP. It is worth pointing out that the focus of the thesis was the RNN-based models, which meant that more time was spent on trying to improve the accuracy of these models. Except for the difference in accuracy, there seems to be a more fundamental difference between the inner workings of the models. If artificial noise was added to the trajectory (as seen in figure 5.32), then the noise still remained in the trajectory for the MLP. This was not the case for the RNN-based models (see figures 5.33 and 5.34). This suggest that the predictions that the MLP make takes the same form as the input data, instead of first finding the trajectory that best matches the input data, and then extrapolating that trajectory.

To demonstrate that the RNN-based models more likely worked as the second approach, the minimum number of input points can be supplied as input. If the model made predictions that were only based on the form of the input data, then it is likely that the extrapolated trajectory does not have the typical shape. However, if the network finds a trajectory that fits the input data, then the extrapolated trajectory still should have the typical “arc shape”. The minimum number

of points for a RNN-based model in this case was two, and the results can be seen in figure 6.1. As can be seen in the figure, the trajectory still had the typical arc shape, even though the input data that the model received contained very little information about the overall shape of the trajectory. This suggest that the shape is stored in the network instead.



(a) RNN direct **(b)** RNN force

Figure 6.1: The RNN-based models were supplied two points (purple line) as input, and the red line is the extrapolated trajectory.

In terms of other factors, such as the computational time, the MLP was much faster to make predictions (see table 5.8) than the RNN-based models. The difference in computational time was due to two things: different number of hidden layers and nodes in each layer and no loading procedure of the input points. As a reminder, the loading procedure for the RNN-based models loaded all the input data into the network before starting the extrapolation, which the MLP did not perform. The computational time of the MLP was also affected by the size of the sliding window, where a larger window size implied a higher computational time. However, the size of the window for the MLP cannot be decreased too much, otherwise the accuracy of the model is affected too much. If the same number of hidden layers and nodes was used, the cost per time step would be lower for the RNN, but the time before the first prediction could be performed would be much higher, as the input data must be loaded in the network first.

6.3 Generalization

As can be seen in the figures and tables in section 5.5, the generalization capabilities of the models were often satisfactory, if the models were evaluated on a site with the same time step as the site the model was trained on. Looking at the characteristics of the trajectories used (see appendix A), the characteristics between sites were quite different. This suggest that the model had some true generalization capabilities across sites, not just used on a site with the same type of trajectories. However, as also seen in section 5.5, if the time step was different, then the predictions were of inferior quality, much worse than the existing model.

The most likely explanation for this was that the network had been overfitted to the velocity implied by the time step of the training data used. For example, if the network was trained on with data with a time step of 0.04, but the data that was supplied as input used a time step of 0.02, then it appears the network predicted the data as it was supplied with the first time step, which meant that the underlying velocity calculated by the predictions were incorrect. This could in theory be solved by using data from several sites with different time steps during training, or transform the data such that all trajectories had the same time step, regardless of the frame rate of the camera.

This might not be necessary; if the model was initialized from an existing model instead of randomly, even from a different site and time step (see section 5.6), and retrained with very little data on the new site, then the results after training was very near the accuracy of a model that was trained with all the data for the site in question. A possible explanation for this behavior is that only a small part of the network depended on the time step (e.g. the last layer) of the data, instead of the entire network. From a practical stand point, this means that a model can be trained with large amounts of data and for a long time, and then retrained with small amounts of data for a new site, without any major loss of accuracy.

6.4 Comparing existing model and RNN

Comparing the existing model and the neural network based models, the accuracy of the neural network based models were much better,

leading on average to a reduction of 36.6 % of the error in the predicted impact position, if the best model was evaluated on the same site it was trained on. In this case, the best model was the RNN using the force prediction output layer. However, the existing model must in some sense be a general model, as it must be applied to all possible trajectories, where the neural network based models can be specialized for a set of trajectories (in this case the set of trajectories for a site). This is the main advantage of a neural network based model, as it can easily be specialized for a set of trajectories.

Comparing the computational time (see section 5.11), the neural network based models were faster to make predictions, especially the MLP. It is worth pointing out that there exist overhead in calculating the trajectories using the existing model, as new a process needs to be created for each trajectory, and the input/output data needs to be parsed from the model. Finally, the computational time can easily be controlled for the neural network based models, by changing the number of hidden layers and units in each layer, to reach the desired computational time, at the expense of the accuracy of the model.

There exists a difference at the fundamental level on how the models work, as the existing model computes a new trajectory that fits the input data to extrapolate the trajectory, instead of just performing an extrapolation to fill in the missing parts. This makes it computational expensive to make on-line predictions, calculating the position of the ball a few steps ahead in real time, as a completely new trajectory needs to be calculated. However, computing a completely new trajectory has advantages, as it eliminates any noise in the input data. To solve this problem for a neural network based model, a noise reduction algorithm was needed, which can be costly to compute, or introduce performance degradations.

6.5 Meaning of the force prediction output layer

As mentioned earlier, the purpose of the force prediction output layer was to add constraints on the type of predictions that the model could make. There are many possible answers to what effect that these constraints have on the model, and some of these will now be discussed.

One possibility was that the network predicted the true physical parameters, which were then used to calculate the displacement of the ball. That is, the physical model served its intended purpose. However, this was unlikely for many reasons. The approach that was used to calculate the displacement in this thesis was not strictly correct from a numerical integration point of view. Using a numerical integrator, the acceleration would be used to update the velocity, which in turn would be used to calculate the change in position (discussed more in section 6.9), not used in a simple formula as done in this thesis. As there does not exist ground truth for the physical parameters, no direct loss could be added for the predictions of these parameters. This meant the network did not need to predict something that was close to the true value, just something that led to a decrease in the overall loss.

Another possibility was that the physical model regularized the model, i.e. the physical model acted as a loss function that penalized the predictions during training that were from a physical point of view unlikely to correspond to the actual position. To study this hypothesis further, a new model was trained that used a second-degree polynomial (see section 5.10). Compared to predicting the displacement directly, this led to improvements for the mean point and last point metrics, but not for the impact point and apex point difference metrics. The first two metrics are more closely related to the loss that the model was trained on than the impact point metric. This suggest that assuming a specific form of the output leads to minor improvements, but, it is only when the constraints are deeply connected to the task that the improvements become significant for all metrics. Finally, without any ground truths for the physical parameters, the accuracy of these predicted parameters cannot be determined.

6.6 Ethical and sustainability considerations

The data used in this thesis was collected from shots by real golfers. These people may not have given their consent, or have any desire, for their golf shots to be used for scientific research. Also, the model developed in this thesis could in theory be used to determine if two shots are hit by the same player, which in turn could be used to deanonymize the data.

A machine learning based model requires considerable computing time for training which needs electrical power. Deep neural networks are known to take a long time to train and need a lot of computational resources. For example, simulations reported in [36] took one month of training using four power hungry GPUs (graphics processing units, special type of processor suitable for training). This means that the model developed in this thesis could impact the environment more, due to excess use of computational power, compared to other type of methods.

6.7 System improvements

The component that is missing for creating a production quality system is a better method for selecting the training data as there exist more data than used in this thesis. The method used in this thesis selected only trajectories that were tracked for at least 85 %. For the different type of trajectories that exist (such as different lengths), no selection was performed to a more balanced distribution of the different types. This meant that for very short or very long trajectories, the models were trained with very few examples of these types. The consequence of this was that there existed a bias towards a specific type of trajectory in the thesis, but this bias must not exist for a real system.

6.8 Applicability to other problems

As mentioned in chapter 1, the problem that was studied in this thesis exist for other sports, not just for golf. The systems designed for solving the problem for these sports used a similar setup as the one used in this thesis with a stereo camera system capturing the 3D position of the ball. Therefore, the RNN-based model without the physical constraints could be used to train a model that solved the problem for these sports as well. The reason for this was that this model made very few assumptions that were specific for golf and performed the extrapolation directly to the captured trajectory. The only remaining problem for applying this model is then obtaining enough training data.

For trajectory prediction problems in general, the most interesting observation was that even if the captured input data contained noise, the extrapolated part of the trajectory was smooth without any noise. This is most likely a property of the LSTM network rather than the specific application of the network to this problem. This means that LSTM networks may be a good choice of method, if these properties are desired.

6.9 Future work

The main problem with the neural network based models were that they did not generalize all that well to sites with a different time step. One possible solution to this is fitting a spline to the input data. From this spline, new points can be obtained that have a desired difference in time between the points. Using this method, the time step for all sites can be equal, say 0.04 seconds. This should in theory solve this problem, and should therefore be investigated in future work.

If the previous mentioned data modifications are applied, then the acceleration can in fact be calculated noise free, from the underlying polynomials that make up the spline. An alternative approach is then to let the network only predict the acceleration, or the parameters that describe the acceleration (and use the known acceleration as the ground truth). From this, a trajectory can then be extrapolated using numerical integration, which uses the predicted acceleration to update the velocity at each time step, which is then used to update the position. This should in theory lead to better predictions of the acceleration (and better estimates of the physical parameters), as the network is directly penalized for predicting the acceleration incorrectly. One possible downside is that since the network does not predict the position, then the accuracy of the predicted position may suffer, which in the end is the main objective. Using a spline also *assumes* a specific form of the acceleration, that it is piecewise linear if the spline consists of polynomials of degree three. This can introduce problems, if this assumption does not hold.

Finally, it should be investigated how one model can be created that predicts both forwards and backwards in time, instead of using two separate models, as done in this thesis. Having one model may lead to better overall accuracy, as the model is trained on both tasks simultaneously.

Chapter 7

Conclusions

The conclusion that can be drawn from this thesis is that machine learning based methods, in the form of neural networks such as RNNs, can be applied to predict the trajectories of golf balls, even if only the 3D position of the ball was captured. The accuracy of neural network based models were on average much better than a model based on numerical simulations, especially for the case when the model was trained and evaluated on the same driving range. There exist some generalization problems across different driving ranges, but this can be solved to some extent by retraining a model with a small amount of data for the driving range in question. Further, conclusions can also be drawn on the last layer of the network. As shown in this thesis, adding constraints to the form of the output improved the accuracy, most noticeable if physical knowledge of the problem was used.

Bibliography

[1] Alexandre Alahi et al. "Social lstm: Human trajectory prediction in crowded spaces". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 961–971.

[2] Firoz Alam et al. "A study of golf ball aerodynamic drag". In: *Procedia Engineering* 13 (2011), pp. 226–231.

[3] United States Golf Association. *Guide to the Rules on Clubs and Balls*. `http://www.usga.org/rules/equipment-rules.html#!rule-14611`. [Online; accessed 31-January-2017]. 2016.

[4] Boris Bačić. "Predicting golf ball trajectories from swing plane: An artificial neural networks approach". In: *Expert Systems with Applications* 65 (2016), pp. 423–438.

[5] Seongmin Baek and Myunggyu Kim. "Flight trajectory of a golf ball for a realistic game". In: *International Journal of Innovation, Management and Technology* 4.3 (2013), p. 346.

[6] Richard H Bartels, John C Beatty, and Brian A Barsky. *An introduction to splines for use in computer graphics and geometric modeling*. Morgan Kaufmann, 1995.

[7] Olivier Cappé, Eric Moulines, and Tobias Ryden. *Inference in Hidden Markov Models (Springer Series in Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005. ISBN: 0387402640.

[8] Hua-Tsung Chen et al. "Ball tracking and 3D trajectory approximation with applications to tactics analysis from single-camera volleyball sequences". In: *Multimedia Tools and Applications* 60.3 (2012), pp. 641–667.

[9] Hua-Tsung Chen et al. "Physics-based ball tracking and 3D trajectory reconstruction with applications to shooting location estimation in basketball video". In: *Journal of Visual Communication and Image Representation* 20.3 (2009), pp. 204–216.

[10] Patrick Pakyan Choi and Martial Hebert. "Learning and predicting moving object trajectory: a piecewise trajectory segment approach". In: *Robotics Institute* (2006), p. 337.

[11] Wei-Ta Chu, Chia-Wei Wang, and Ja-Ling Wu. "Extraction of baseball trajectory and physics-based validation for single-view baseball video sequences". In: *Multimedia and Expo, 2006 IEEE International Conference on*. IEEE. 2006, pp. 1813–1816.

[12] Sebastien Ehrhardt et al. "Learning A Physical Long-term Predictor". In: *arXiv preprint arXiv:1703.00247* (2017).

[13] Katerina Fragkiadaki et al. "Learning visual predictive models of physics for playing billiards". In: *arXiv preprint arXiv:1511.07404* (2015).

[14] Ray J Frank, Neil Davey, and Stephen P Hunt. "Time series prediction and neural networks". In: *Journal of intelligent and robotic systems* 31.1-3 (2001), pp. 91–103.

[15] Felix Gers. "Long short-term memory in recurrent neural networks". PhD thesis. Universität Hannover, 2001.

[16] Felix A Gers and Jürgen Schmidhuber. "Recurrent nets that time and count". In: *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*. Vol. 3. IEEE. 2000, pp. 189–194.

[17] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *International conference on artificial intelligence and statistics*. 2010, pp. 249–256.

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[19] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. "Speech recognition with deep recurrent neural networks". In: *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE. 2013, pp. 6645–6649.

[20] Alex Graves and Jürgen Schmidhuber. "Framewise phoneme classification with bidirectional LSTM and other neural network architectures". In: *Neural Networks* 18.5 (2005), pp. 602–610.

[21] Alex Graves et al. "A novel connectionist system for unconstrained handwriting recognition". In: *IEEE transactions on pattern analysis and machine intelligence* 31.5 (2009), pp. 855–868.

[22] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[23] Peter J Huber et al. "Robust estimation of a location parameter". In: *The Annals of Mathematical Statistics* 35.1 (1964), pp. 73–101.

[24] Theodore P Jorgensen. *The physics of golf*. Springer Science & Business Media, 1999.

[25] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[27] Anil Kumar et al. "3D Estimation and Visualization of Motion in a Multicamera Network for Sports". In: *Machine Vision and Image Processing Conference (IMVIP), 2011 Irish*. IEEE. 2011, pp. 15–19.

[28] William M MacDonald and Stephen Hanzely. "The physics of the drive in golf". In: *American Journal of Physics* 59.3 (1991), pp. 213–218.

[29] James McNames. "A nearest trajectory strategy for time series prediction". In: *Proceedings of the International Workshop on Advanced Black-Box Techniques for Nonlinear Modeling*. KU Leuven Belgium. 1998, pp. 112–128.

[30] John J McPhee and Gordon C Andrews. "Effect of sidespin and wind on projectile trajectory, with particular application to golf". In: *American Journal of Physics* 56.10 (1988), pp. 933–939.

[31] *Mevo – portable multi-sport radar, practice with purpose*. `http://flightscopemevo.com/#!/#mevo-for-golf`. [Online; accessed 8 June 2017]. 2017.

[32] Roland B. Minton. *Sports Math: An Introductory Course in the Mathematics of Sports Science and Sports Analytics*. first. Chapman and Hall/CRC, Oct. 2016.

[33] *Neural Networks for Time Series Prediction*. `https://www.cs.cmu.edu/afs/cs/academic/class/15782-f06/slides/timeseries.pdf`. [Online; accessed 23 January 2017]. 2006.

[34] Peter Olofsson and Mikael Andersson. *Probability, statistics, and stochastic processes*. John Wiley & Sons, 2012.

[35] A Raymond Penner. "The physics of golf". In: *Reports on Progress in Physics* 66.2 (2002), p. 131.

[36] Alec Radford, Rafal Jozefowicz, and Ilya Sutskever. "Learning to generate reviews and discovering sentiment". In: *arXiv preprint arXiv:1704.01444* (2017).

[37] Piyush Rai, Abhishek Kumar, and Hal Daume. "Simultaneously leveraging output and task structures for multiple-output regression". In: *Advances in Neural Information Processing Systems*. 2012, pp. 3185–3193.

[38] *scipy.stats.f_oneway*. `https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.stats.f_oneway.html`. [Online; accessed 8 June 2017]. 2017.

[39] *scipy.stats.t*. `https://docs.scipy.org/doc/scipy-0.19.0/reference/generated/scipy.stats.t.html`. [Online; accessed 24 May 2017]. 2017.

[40] Jonas Sköld. "Estimating 3D-trajectories from Monocular Video Sequences". MA thesis. Sweden: KTH Royal Institute of Technology, 2015.

[41] AJ Smits and DR Smith. "A new aerodynamic model of a golf ball in flight". In: *Science and golf II* 340 (1994), p. 347.

[42] *statsmodels.stats.multicomp.pairwise_tukeyhsd*. `http://www.statsmodels.org/stable/generated/statsmodels.stats.multicomp.pairwise_tukeyhsd.html`. [Online; accessed 8 June 2017]. 2017.

[43] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[44] *The SkyTrak Difference*. `http://www.skytrakgolf.com/about-skytrak`. [Online; accessed 8 June 2017]. 2017.

[45] LL Ting. "Effects of dimple size and depth on golf ball aerodynamic performance". In: *ASME/JSME 2003 4th Joint Fluids Summer Engineering Conference*. American Society of Mechanical Engineers. 2003, pp. 811–817.

[46] Yi-Jen Wang and Chin-Teng Lin. "Runge-Kutta neural network for identification of dynamical systems in high accuracy". In: *IEEE Transactions on Neural Networks* 9.2 (1998), pp. 294–307.

[47] Eric Weisstein. *Student's t-Distribution*. `http://mathworld.wolfram.com/Studentst-Distribution.html`. [Online; accessed 24 May 2017]. 2017.

[48] Ronald J Williams and Jing Peng. "An efficient gradient-based algorithm for on-line training of recurrent network trajectories". In: *Neural computation* 2.4 (1990), pp. 490–501.

[49] Wojciech Zaremba. "An empirical exploration of recurrent network architectures". In: (2015).

[50] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. "Recurrent neural network regularization". In: *arXiv preprint arXiv:1409.2329* (2014).

[51] Albert Zeyer et al. "A comprehensive study of deep bidirectional LSTM RNNs for acoustic modeling in speech recognition". In: *arXiv preprint arXiv:1606.06871* (2016).

Appendix A

Detailed data set statistics

The following tables present detailed statistics of the characteristics of the trajectories used. Curve denotes the maximum distance for a point in the trajectory from an imaginary line drawn from the start of the trajectory to the end.

Table A.1: The distribution of the length for the trajectories used at a particular site.

Length	0-50	50-75	75-100	100-150	150-200	200-250
Site 1 (%)	1.3	19.5	39.4	35.7	4.0	0.12
Site 2 (%)	1.9	31.0	41.5	23.3	2.2	0.02
Site 3 (%)	0.14	14.4	41.9	41.0	2.5	0.09
Site 4 (%)	2.6	17.3	34.1	42.4	3.6	0.00
Site 5 (%)	0.04	9.3	33.3	51.8	5.5	0.01

Table A.2: The distribution of the curve for the trajectories used at a particular site.

Curve	0-0.5	0.5-1	1-2.5	2.5-5	5-7.5	7.5-10	10-55
Site 1 (%)	10.2	11.2	25.8	25.4	12.9	6.9	7.6
Site 2 (%)	11.2	11.0	25.9	26.4	12.9	6.2	6.5
Site 3 (%)	8.3	9.5	23.4	26.1	15.0	7.8	9.9
Site 4 (%)	7.0	7.7	20.2	23.3	14.8	9.8	17.2
Site 5 (%)	5.8	7.2	19.3	24.3	16.3	10.6	16.6

Appendix B

Extended results

In this appendix, the full results are given for different experiments conducted.

B.1 Results for all sites

The following tables present the complete results for all trained models and evaluated on all sites. In the tables, "Std." denotes the standard deviation and "Max" the maximum of all the errors.

B.1.1 Site 1

In this section, the result for models that were evaluated on site 1 are presented.

Table B.1: The p-tests for the different metrics performed on the site 1 test set.

Metric	P-value
Mean point	0
Impact point	10^{-86}
Last point	10^{-271}
Apex point	10^{-167}

Table B.2: Pair-wise comparisons using Tukey's range test for the mean point difference metric evaluated on site 1 test set. The critical value was 3.633.

Model #1	Model #2	q	Reject null	Desired
Existing model	MLP (Site 1)	194.331	True	Reject
Existing model	RNN direct (Site 1)	240.458	True	Reject
Existing model	RNN force (Site 1)	288.148	True	Reject
MLP (Site 1)	RNN direct (Site 1)	45.813	True	Reject
MLP (Site 1)	RNN force (Site 1)	93.290	True	Reject
RNN direct (Site 1)	RNN force (Site 1)	47.504	True	Reject

Table B.3: Pair-wise comparisons using Tukey's range test for the impact point difference metric evaluated on site 1 test set. The critical value was 3.633.

Model #1	Model #2	q	Reject null	Desired
Existing model	MLP (Site 1)	2.824	False	Reject
Existing model	RNN direct (Site 1)	14.627	True	Reject
Existing model	RNN force (Site 1)	25.479	True	Reject
MLP (Site 1)	RNN direct (Site 1)	11.758	True	Reject
MLP (Site 1)	RNN force (Site 1)	22.576	True	Reject
RNN direct (Site 1)	RNN force (Site 1)	10.842	True	Reject

Table B.4: Pair-wise comparisons using Tukey's range test for the last point difference metric evaluated on site 1 test set. The critical value was 3.633.

Model #1	Model #2	q	Reject null	Desired
Existing model	MLP (Site 1)	26.736	True	Reject
Existing model	RNN direct (Site 1)	34.584	True	Reject
Existing model	RNN force (Site 1)	50.201	True	Reject
MLP (Site 1)	RNN direct (Site 1)	7.752	True	Reject
MLP (Site 1)	RNN force (Site 1)	23.308	True	Reject
RNN direct (Site 1)	RNN force (Site 1)	15.596	True	Reject

Table B.5: Pair-wise comparisons using Tukey's range test for the apex point difference metric evaluated on site 1 test set. The critical value was 3.633.

Model #1	**Model #2**	**q**	**Reject null**	**Desired**
Existing model	MLP (Site 1)	27.593	True	Reject
Existing model	RNN direct (Site 1)	34.300	True	Reject
Existing model	RNN force (Site 1)	34.196	True	Reject
MLP (Site 1)	RNN direct (Site 1)	6.612	True	Reject
MLP (Site 1)	RNN force (Site 1)	6.496	True	Reject
RNN direct (Site 1)	RNN force (Site 1)	0.119	False	Reject

Table B.6: The results of the mean point difference metric evaluated on site 1. Values are in meters.

	Validation ($n = 60\,972$)			*Test* ($n = 182\,303$)		
Model	**Mean**	**Std.**	**Max**	**Mean**	**Std.**	**Max**
Existing model	1.51 ± 0.01	1.52	16.79	1.44 ± 0.01	1.46	20.42
MLP (Site 1)	0.95 ± 0.01	1.23	15.72	0.92 ± 0.01	1.20	22.86
RNN direct (Site 1)	0.80 ± 0.01	0.98	9.57	0.80 ± 0.00	1.00	14.88
RNN force (Site 1)	**0.68** ± 0.01	**0.84**	**9.03**	**0.67** ± 0.00	**0.84**	13.58
MLP (Site 2)	1.01 ± 0.01	1.27	20.11	0.97 ± 0.01	1.16	16.47
RNN direct (Site 2)	0.93 ± 0.01	1.03	10.50	0.95 ± 0.00	1.08	18.08
RNN force (Site 2)	0.96 ± 0.01	1.10	11.52	0.93 ± 0.00	1.07	14.85
MLP (Site 3)	1.03 ± 0.01	1.17	15.79	1.00 ± 0.01	1.13	17.41
RNN direct (Site 3)	0.86 ± 0.01	1.05	10.77	0.83 ± 0.00	1.02	**12.62**
RNN force (Site 3)	0.82 ± 0.01	0.93	9.84	0.80 ± 0.00	0.93	13.95
MLP (Site 4)	2.28 ± 0.02	2.38	21.30	2.23 ± 0.01	2.31	25.34
RNN direct (Site 4)	1.87 ± 0.02	1.93	19.88	1.82 ± 0.01	1.86	27.57
RNN force (Site 4)	1.82 ± 0.01	1.69	16.93	1.79 ± 0.01	1.67	23.59
MLP (Site 5)	2.16 ± 0.02	2.45	28.17	2.08 ± 0.01	2.28	31.18
RNN direct (Site 5)	2.56 ± 0.02	2.10	18.05	2.56 ± 0.01	2.10	22.30
RNN force (Site 5)	2.09 ± 0.02	1.96	19.60	2.05 ± 0.01	1.93	27.74

Table B.7: The results of the impact point difference metric evaluated on site 1. Values are in meters.

	Validation ($n = 1\,000$)			*Test* ($n = 3\,000$)		
Model	**Mean**	**Std.**	**Max**	**Mean**	**Std.**	**Max**
Existing model	2.82 ± 0.13	2.03	15.59	2.76 ± 0.07	1.99	19.01
MLP (Site 1)	2.74 ± 0.15	2.34	19.40	2.66 ± 0.08	2.31	27.44
RNN direct (Site 1)	2.25 ± 0.12	1.86	14.69	2.24 ± 0.07	1.92	19.13
RNN force (Site 1)	**1.89** ± 0.10	**1.59**	**12.27**	**1.85** ± 0.06	**1.57**	**15.40**
MLP (Site 2)	2.55 ± 0.12	1.89	14.87	2.55 ± 0.07	1.90	19.49
RNN direct (Site 2)	2.36 ± 0.11	1.70	13.93	2.37 ± 0.06	1.77	15.82
RNN force (Site 2)	2.28 ± 0.11	1.69	15.14	2.23 ± 0.07	1.82	47.74
MLP (Site 3)	2.89 ± 0.13	2.15	25.28	2.92 ± 0.08	2.32	40.14
RNN direct (Site 3)	2.56 ± 0.14	2.19	27.24	2.53 ± 0.08	2.13	22.98
RNN force (Site 3)	2.38 ± 0.11	1.75	16.54	2.36 ± 0.06	1.73	36.06
MLP (Site 4)	8.81 ± 0.25	4.00	28.61	8.77 ± 0.14	3.86	33.61
RNN direct (Site 4)	6.93 ± 0.27	4.39	29.12	6.91 ± 0.15	4.31	36.77
RNN force (Site 4)	6.13 ± 0.28	4.55	33.73	6.08 ± 0.16	4.43	35.90
MLP (Site 5)	8.30 ± 0.37	5.84	39.47	8.27 ± 0.21	5.76	56.67
RNN direct (Site 5)	10.47 ± 0.24	3.90	31.08	10.49 ± 0.14	3.87	35.64
RNN force (Site 5)	7.91 ± 0.34	5.55	38.88	7.68 ± 0.19	5.34	56.57

Table B.8: The results of the last point difference metric evaluated on site 1. Values are in meters.

	Validation ($n = 1\,000$)			*Test* ($n = 3\,000$)		
Model	**Mean**	**Std.**	**Max**	**Mean**	**Std.**	**Max**
Existing model	3.36 ± 0.13	2.10	16.79	3.23 ± 0.07	2.00	20.42
MLP (Site 1)	2.47 ± 0.12	1.87	15.72	2.41 ± 0.07	1.84	22.86
RNN direct (Site 1)	2.18 ± 0.09	1.48	9.57	2.17 ± 0.06	1.54	14.88
RNN force (Site 1)	**1.70** ± 0.08	**1.27**	**9.03**	**1.69** ± 0.05	**1.28**	13.58
MLP (Site 2)	2.61 ± 0.13	2.00	20.11	2.54 ± 0.06	1.78	16.47
RNN direct (Site 2)	2.34 ± 0.09	1.47	10.50	2.41 ± 0.06	1.61	18.08
RNN force (Site 2)	2.34 ± 0.10	1.59	11.52	2.29 ± 0.06	1.55	14.85
MLP (Site 3)	2.61 ± 0.10	1.65	15.79	2.57 ± 0.06	1.65	17.41
RNN direct (Site 3)	2.23 ± 0.09	1.52	10.77	2.20 ± 0.05	1.51	**12.62**
RNN force (Site 3)	2.00 ± 0.08	1.32	9.84	1.99 ± 0.05	1.32	13.95
MLP (Site 4)	5.73 ± 0.19	3.04	21.30	5.63 ± 0.10	2.89	25.34
RNN direct (Site 4)	4.24 ± 0.16	2.64	19.88	4.16 ± 0.09	2.51	27.57
RNN force (Site 4)	3.73 ± 0.14	2.31	16.93	3.68 ± 0.08	2.26	23.59
MLP (Site 5)	4.97 ± 0.22	3.56	28.17	4.83 ± 0.12	3.25	31.18
RNN direct (Site 5)	5.73 ± 0.14	2.22	18.05	5.74 ± 0.08	2.18	22.30
RNN force (Site 5)	4.33 ± 0.17	2.66	19.60	4.25 ± 0.09	2.60	27.74

Table B.9: The results of the apex point difference metric evaluated on site 1. Values are in meters.

	Validation ($n = 1\,000$)			*Test* ($n = 3\,000$)		
Model	**Mean**	**Std.**	**Max**	**Mean**	**Std.**	**Max**
Existing model	1.16 ± 0.08	1.27	10.16	1.13 ± 0.04	1.24	14.95
MLP (Site 1)	0.69 ± 0.09	1.37	24.31	0.63 ± 0.04	1.04	17.56
RNN direct (Site 1)	0.54 ± 0.05	0.75	6.08	0.51 ± 0.03	0.78	13.35
RNN force (Site 1)	0.52 ± 0.05	0.73	7.56	0.51 ± 0.03	0.81	21.22
MLP (Site 2)	0.61 ± 0.05	0.81	**5.77**	0.59 ± 0.03	0.87	14.07
RNN direct (Site 2)	0.54 ± 0.05	0.76	7.09	0.51 ± 0.03	**0.74**	**8.19**
RNN force (Site 2)	0.58 ± 0.05	0.85	6.83	0.56 ± 0.03	0.79	9.84
MLP (Site 3)	0.66 ± 0.06	0.97	14.59	0.63 ± 0.05	1.41	49.82
RNN direct (Site 3)	0.52 ± 0.05	0.76	6.42	0.52 ± 0.03	0.81	9.08
RNN force (Site 3)	**0.52** ± 0.04	**0.72**	6.26	**0.50** ± 0.04	1.07	44.98
MLP (Site 4)	2.35 ± 0.15	2.34	20.53	2.28 ± 0.08	2.23	32.70
RNN direct (Site 4)	2.28 ± 0.13	2.10	16.23	2.24 ± 0.07	2.06	26.13
RNN force (Site 4)	2.23 ± 0.14	2.27	20.55	2.16 ± 0.08	2.19	31.44
MLP (Site 5)	2.27 ± 0.21	3.33	30.20	2.21 ± 0.11	3.11	40.16
RNN direct (Site 5)	2.26 ± 0.12	1.86	14.99	2.26 ± 0.07	1.91	23.33
RNN force (Site 5)	2.29 ± 0.15	2.45	22.80	2.21 ± 0.08	2.29	32.20

B.1.2 Site 2

In this section, the result for models that were evaluated on site 2 are presented.

Table B.10: The p-tests for the different metrics performed on the site 2 test set.

Metric	**P-value**
Mean point	0
Impact point	10^{-88}
Last point	10^{-106}
Apex point	10^{-119}

Table B.11: Pair-wise comparisons using Tukey's range test for the mean point difference metric evaluated on site 2 test set. The critical value was 3.633.

Model #1	Model #2	q	Reject null	Desired
Existing model	MLP (Site 2)	128.315	True	Reject
Existing model	RNN direct (Site 2)	138.774	True	Reject
Existing model	RNN force (Site 2)	203.446	True	Reject
MLP (Site 2)	RNN direct (Site 2)	10.021	True	Reject
MLP (Site 2)	RNN force (Site 2)	74.274	True	Reject
RNN direct (Site 2)	RNN force (Site 2)	64.440	True	Reject

Table B.12: Pair-wise comparisons using Tukey's range test for the impact point difference metric evaluated on site 2 test set. The critical value was 3.633.

Model #1	Model #2	q	Reject null	Desired
Existing model	MLP (Site 2)	13.622	True	Reject
Existing model	RNN direct (Site 2)	19.255	True	Reject
Existing model	RNN force (Site 2)	28.128	True	Reject
MLP (Site 2)	RNN direct (Site 2)	5.520	True	Reject
MLP (Site 2)	RNN force (Site 2)	14.325	True	Reject
RNN direct (Site 2)	RNN force (Site 2)	8.852	True	Reject

Table B.13: Pair-wise comparisons using Tukey's range test for the last point difference metric evaluated on site 2 test set. The critical value was 3.633.

Model #1	Model #2	q	Reject null	Desired
Existing model	MLP (Site 2)	10.588	True	Reject
Existing model	RNN direct (Site 2)	14.239	True	Reject
Existing model	RNN force (Site 2)	31.185	True	Reject
MLP (Site 2)	RNN direct (Site 2)	3.568	False	Reject
MLP (Site 2)	RNN force (Site 2)	20.397	True	Reject
RNN direct (Site 2)	RNN force (Site 2)	16.922	True	Reject

Table B.14: Pair-wise comparisons using Tukey's range test for the apex point difference metric evaluated on site 2 test set. The critical value was 3.633.

Model #1	Model #2	q	Reject null	Desired
Existing model	MLP (Site 2)	21.128	True	Reject
Existing model	RNN direct (Site 2)	27.421	True	Reject
Existing model	RNN force (Site 2)	30.606	True	Reject
MLP (Site 2)	RNN direct (Site 2)	6.133	True	Reject
MLP (Site 2)	RNN force (Site 2)	9.281	True	Reject
RNN direct (Site 2)	RNN force (Site 2)	3.162	False	Reject

Table B.15: The results of the mean point difference metric evaluated on site 2. Values are in meters.

	Validation ($n = 57\,686$)			*Test* ($n = 172\,030$)		
Model	**Mean**	**Std.**	**Max**	**Mean**	**Std.**	**Max**
Existing model	0.98 ± 0.01	0.97	12.94	0.98 ± 0.00	0.96	20.28
MLP (Site 1)	0.83 ± 0.01	1.10	12.88	0.82 ± 0.01	1.07	23.54
RNN direct (Site 1)	0.91 ± 0.01	1.03	11.76	0.91 ± 0.00	1.05	18.84
RNN force (Site 1)	0.74 ± 0.01	0.89	10.58	0.73 ± 0.00	0.87	20.38
MLP (Site 2)	0.68 ± 0.01	0.89	13.06	0.71 ± 0.00	0.98	20.56
RNN direct (Site 2)	0.67 ± 0.01	0.82	9.39	0.69 ± 0.00	0.85	21.53
RNN force (Site 2)	**0.56** ± 0.01	**0.67**	**8.17**	**0.55** ± 0.00	**0.66**	19.48
MLP (Site 3)	0.81 ± 0.01	1.02	13.61	0.78 ± 0.00	0.96	19.97
RNN direct (Site 3)	0.71 ± 0.01	0.84	8.55	0.71 ± 0.00	0.84	20.48
RNN force (Site 3)	0.66 ± 0.01	0.80	8.94	0.65 ± 0.00	0.78	20.46
MLP (Site 4)	2.06 ± 0.02	2.05	18.85	2.03 ± 0.01	2.01	22.57
RNN direct (Site 4)	1.90 ± 0.01	1.78	18.23	1.85 ± 0.01	1.70	**17.69**
RNN force (Site 4)	1.97 ± 0.01	1.77	21.18	1.94 ± 0.01	1.69	17.95
MLP (Site 5)	2.43 ± 0.02	2.49	29.89	2.37 ± 0.01	2.38	34.11
RNN direct (Site 5)	2.76 ± 0.02	2.25	19.67	2.71 ± 0.01	2.18	22.63
RNN force (Site 5)	2.31 ± 0.02	2.08	18.85	2.28 ± 0.01	2.02	20.75

Table B.16: The results of the impact point difference metric evaluated on site 2. Values are in meters.

	Validation ($n = 1\,000$)			*Test* ($n = 3\,000$)		
Model	**Mean**	**Std.**	**Max**	**Mean**	**Std.**	**Max**
Existing model	2.18 ± 0.10	1.63	13.30	2.18 ± 0.06	1.56	26.03
MLP (Site 1)	2.09 ± 0.14	2.16	33.35	1.97 ± 0.06	1.75	**21.39**
RNN direct (Site 1)	2.23 ± 0.12	1.96	17.19	2.19 ± 0.06	1.80	24.67
RNN force (Site 1)	1.66 ± 0.09	1.48	14.11	1.65 ± 0.05	1.47	30.28
MLP (Site 2)	1.88 ± 0.11	1.73	14.90	1.81 ± 0.06	1.64	39.33
RNN direct (Site 2)	1.69 ± 0.09	1.49	11.91	1.66 ± 0.05	1.44	26.66
RNN force (Site 2)	**1.47** ± 0.08	**1.25**	**10.53**	**1.42** ± 0.04	**1.22**	29.65
MLP (Site 3)	2.21 ± 0.12	1.88	22.57	2.16 ± 0.06	1.77	24.64
RNN direct (Site 3)	1.84 ± 0.12	1.86	17.89	1.83 ± 0.06	1.77	25.76
RNN force (Site 3)	1.68 ± 0.09	1.47	14.26	1.64 ± 0.05	1.37	28.72
MLP (Site 4)	8.57 ± 0.24	3.82	44.93	8.52 ± 0.13	3.54	42.29
RNN direct (Site 4)	7.43 ± 0.25	4.01	34.26	7.37 ± 0.13	3.64	35.86
RNN force (Site 4)	7.09 ± 0.28	4.53	44.29	7.06 ± 0.15	4.06	45.79
MLP (Site 5)	9.69 ± 0.38	6.09	37.13	9.54 ± 0.21	5.87	70.90
RNN direct (Site 5)	11.11 ± 0.23	3.75	29.34	11.06 ± 0.12	3.38	32.93
RNN force (Site 5)	8.80 ± 0.32	5.13	38.27	8.78 ± 0.17	4.81	46.20

Table B.17: The results of the last point difference metric evaluated on site 2. Values are in meters.

	Validation ($n = 1\,000$)			*Test* ($n = 3\,000$)		
Model	**Mean**	**Std.**	**Max**	**Mean**	**Std.**	**Max**
Existing model	2.09 ± 0.08	1.34	12.94	2.08 ± 0.05	1.34	20.28
MLP (Site 1)	2.11 ± 0.11	1.73	12.88	2.08 ± 0.06	1.69	23.54
RNN direct (Site 1)	2.36 ± 0.09	1.42	11.76	2.37 ± 0.05	1.44	18.84
RNN force (Site 1)	1.75 ± 0.08	1.28	10.58	1.74 ± 0.05	1.28	20.38
MLP (Site 2)	1.78 ± 0.09	1.45	13.06	1.82 ± 0.06	1.65	20.56
RNN direct (Site 2)	1.69 ± 0.08	1.23	9.39	1.73 ± 0.05	1.32	21.53
RNN force (Site 2)	**1.32** ± 0.06	**0.96**	**8.17**	**1.32** ± 0.04	**0.99**	19.48
MLP (Site 3)	2.03 ± 0.10	1.52	13.61	1.97 ± 0.05	1.43	19.97
RNN direct (Site 3)	1.84 ± 0.08	1.22	8.55	1.84 ± 0.04	1.25	20.48
RNN force (Site 3)	1.62 ± 0.07	1.15	8.94	1.62 ± 0.04	1.15	20.46
MLP (Site 4)	4.82 ± 0.16	2.51	18.85	4.76 ± 0.09	2.49	22.57
RNN direct (Site 4)	4.06 ± 0.13	2.14	18.23	4.02 ± 0.07	2.00	**17.69**
RNN force (Site 4)	3.87 ± 0.14	2.18	21.18	3.86 ± 0.07	2.00	17.95
MLP (Site 5)	5.28 ± 0.21	3.34	29.89	5.17 ± 0.11	3.14	34.11
RNN direct (Site 5)	5.94 ± 0.14	2.28	19.67	5.87 ± 0.08	2.18	22.63
RNN force (Site 5)	4.72 ± 0.16	2.55	18.85	4.70 ± 0.09	2.45	20.75

Table B.18: The results of the apex point difference metric evaluated on site 2. Values are in meters.

	Validation ($n = 1\,000$)			*Test* ($n = 3\,000$)		
Model	**Mean**	**Std.**	**Max**	**Mean**	**Std.**	**Max**
Existing model	0.89 ± 0.06	0.93	8.30	0.88 ± 0.03	0.87	9.52
MLP (Site 1)	0.67 ± 0.08	1.23	23.74	0.66 ± 0.03	0.96	15.21
RNN direct (Site 1)	0.58 ± 0.05	0.81	6.86	0.57 ± 0.03	0.77	11.84
RNN force (Site 1)	0.53 ± 0.05	0.75	9.42	0.50 ± 0.03	0.82	28.84
MLP (Site 2)	0.64 ± 0.06	0.96	9.21	0.59 ± 0.03	0.79	8.35
RNN direct (Site 2)	0.51 ± 0.05	0.80	7.63	0.51 ± 0.03	0.71	8.53
RNN force (Site 2)	0.50 ± 0.04	0.72	8.42	**0.46** ± 0.02	**0.62**	**6.37**
MLP (Site 3)	0.62 ± 0.06	0.90	15.29	0.63 ± 0.04	1.02	27.95
RNN direct (Site 3)	0.56 ± 0.06	1.04	19.20	0.53 ± 0.03	0.85	19.12
RNN force (Site 3)	**0.50** ± 0.04	**0.69**	**5.99**	0.48 ± 0.02	0.64	7.29
MLP (Site 4)	2.20 ± 0.17	2.68	52.30	2.18 ± 0.09	2.47	49.12
RNN direct (Site 4)	2.15 ± 0.11	1.84	14.30	2.09 ± 0.06	1.59	12.96
RNN force (Site 4)	2.18 ± 0.15	2.42	43.55	2.09 ± 0.07	1.91	35.81
MLP (Site 5)	2.74 ± 0.26	4.14	41.18	2.61 ± 0.13	3.55	34.87
RNN direct (Site 5)	2.32 ± 0.11	1.75	12.96	2.29 ± 0.06	1.60	19.50
RNN force (Site 5)	2.28 ± 0.15	2.36	30.27	2.20 ± 0.07	1.98	32.02

B.1.3 Site 3

In this section, the result for models that were evaluated on site 3 are presented.

Table B.19: The p-tests for the different metrics performed on the site 3 test set.

Metric	**P-value**
Mean point	0
Impact point	10^{-119}
Last point	10^{-255}
Apex point	10^{-321}

Table B.20: Pair-wise comparisons using Tukey's range test for the mean point difference metric evaluated on site 3 test set. The critical value was 3.633.

Model #1	Model #2	q	Reject null	Desired
Existing model	MLP (Site 3)	182.267	True	Reject
Existing model	RNN direct (Site 3)	245.316	True	Reject
Existing model	RNN force (Site 3)	270.550	True	Reject
MLP (Site 3)	RNN direct (Site 3)	62.771	True	Reject
MLP (Site 3)	RNN force (Site 3)	87.936	True	Reject
RNN direct (Site 3)	RNN force (Site 3)	25.180	True	Reject

Table B.21: Pair-wise comparisons using Tukey's range test for the impact point difference metric evaluated on site 3 test set. The critical value was 3.633.

Model #1	Model #2	q	Reject null	Desired
Existing model	MLP (Site 3)	13.418	True	Reject
Existing model	RNN direct (Site 3)	25.792	True	Reject
Existing model	RNN force (Site 3)	30.681	True	Reject
MLP (Site 3)	RNN direct (Site 3)	12.345	True	Reject
MLP (Site 3)	RNN force (Site 3)	17.225	True	Reject
RNN direct (Site 3)	RNN force (Site 3)	4.884	True	Reject

Table B.22: Pair-wise comparisons using Tukey's range test for the last point difference metric evaluated on site 3 test set. The critical value was 3.633.

Model #1	Model #2	q	Reject null	Desired
Existing model	MLP (Site 3)	23.064	True	Reject
Existing model	RNN direct (Site 3)	38.400	True	Reject
Existing model	RNN force (Site 3)	46.151	True	Reject
MLP (Site 3)	RNN direct (Site 3)	15.291	True	Reject
MLP (Site 3)	RNN force (Site 3)	23.029	True	Reject
RNN direct (Site 3)	RNN force (Site 3)	7.743	True	Reject

Table B.23: Pair-wise comparisons using Tukey's range test for the apex point difference metric evaluated on site 3 test set. The critical value was 3.633.

Model #1	Model #2	q	Reject null	Desired
Existing model	MLP (Site 3)	41.701	True	Reject
Existing model	RNN direct (Site 3)	47.153	True	Reject
Existing model	RNN force (Site 3)	47.838	True	Reject
MLP (Site 3)	RNN direct (Site 3)	5.400	True	Reject
MLP (Site 3)	RNN force (Site 3)	6.078	True	Reject
RNN direct (Site 3)	RNN force (Site 3)	0.677	False	Reject

Table B.24: The results of the mean point difference metric evaluated on site 3. Values are in meters.

	Validation ($n = 65\,778$)			*Test* ($n = 198\,071$)		
Model	**Mean**	**Std.**	**Max**	**Mean**	**Std.**	**Max**
Existing model	1.52 ± 0.01	1.48	18.19	1.53 ± 0.01	1.49	17.98
MLP (Site 1)	1.10 ± 0.01	1.50	22.67	1.09 ± 0.01	1.53	31.86
RNN direct (Site 1)	1.04 ± 0.01	1.50	35.24	1.06 ± 0.01	1.64	49.35
RNN force (Site 1)	0.92 ± 0.01	1.33	30.84	0.91 ± 0.01	1.31	28.22
MLP (Site 2)	1.04 ± 0.01	1.44	24.01	1.08 ± 0.01	1.53	29.08
RNN direct (Site 2)	0.97 ± 0.01	1.36	29.66	1.00 ± 0.01	1.67	53.55
RNN force (Site 2)	0.94 ± 0.01	1.44	35.34	0.95 ± 0.01	1.45	37.32
MLP (Site 3)	1.00 ± 0.01	1.26	**17.53**	1.00 ± 0.01	1.25	**16.29**
RNN direct (Site 3)	0.81 ± 0.01	1.18	26.24	0.82 ± 0.01	1.26	39.18
RNN force (Site 3)	**0.73** ± 0.01	**1.07**	26.25	**0.75** ± 0.00	**1.10**	26.85
MLP (Site 4)	2.72 ± 0.02	2.72	25.53	2.73 ± 0.01	2.77	33.43
RNN direct (Site 4)	2.47 ± 0.02	2.48	30.02	2.48 ± 0.01	2.55	38.59
RNN force (Site 4)	2.47 ± 0.02	2.31	30.13	2.49 ± 0.01	2.34	33.36
MLP (Site 5)	2.97 ± 0.02	3.08	30.41	2.97 ± 0.01	3.11	44.70
RNN direct (Site 5)	3.17 ± 0.02	2.69	35.47	3.17 ± 0.01	2.74	39.07
RNN force (Site 5)	2.82 ± 0.02	2.68	36.61	2.85 ± 0.01	2.74	39.00

Table B.25: The results of the impact point difference metric evaluated on site 3. Values are in meters.

	Validation ($n = 1\,000$)			*Test* ($n = 3\,000$)		
Model	**Mean**	**Std.**	**Max**	**Mean**	**Std.**	**Max**
Existing model	3.01 ± 0.14	2.33	20.12	3.13 ± 0.09	2.43	26.97
MLP (Site 1)	2.79 ± 0.17	2.67	24.70	2.85 ± 0.10	2.90	41.74
RNN direct (Site 1)	2.87 ± 0.18	2.87	42.30	2.92 ± 0.10	2.86	45.27
RNN force (Site 1)	2.23 ± 0.13	2.03	21.85	2.24 ± 0.08	2.12	35.32
MLP (Site 2)	2.54 ± 0.14	2.30	20.32	2.59 ± 0.09	2.46	**23.54**
RNN direct (Site 2)	2.31 ± 0.16	2.56	44.22	2.37 ± 0.11	3.08	70.04
RNN force (Site 2)	1.98 ± 0.14	2.28	45.23	2.05 ± 0.09	2.40	45.60
MLP (Site 3)	2.51 ± 0.15	2.36	39.03	2.58 ± 0.09	2.53	39.93
RNN direct (Site 3)	2.06 ± 0.12	2.00	33.18	2.07 ± 0.08	2.13	37.60
RNN force (Site 3)	**1.79** ± 0.09	**1.46**	**13.81**	**1.87** ± 0.07	**1.90**	47.63
MLP (Site 4)	10.46 ± 0.27	4.34	42.00	10.65 ± 0.16	4.54	43.49
RNN direct (Site 4)	9.17 ± 0.30	4.85	47.07	9.34 ± 0.19	5.24	81.02
RNN force (Site 4)	8.25 ± 0.31	4.93	36.36	8.49 ± 0.18	5.02	48.64
MLP (Site 5)	11.40 ± 0.43	6.82	79.69	11.76 ± 0.26	7.29	156.25
RNN direct (Site 5)	12.77 ± 0.29	4.60	49.78	12.96 ± 0.18	5.05	91.37
RNN force (Site 5)	10.76 ± 0.41	6.55	48.60	11.03 ± 0.24	6.58	54.95

Table B.26: The results of the last point difference metric evaluated on site 3. Values are in meters.

	Validation ($n = 1\,000$)			*Test* ($n = 3\,000$)		
Model	**Mean**	**Std.**	**Max**	**Mean**	**Std.**	**Max**
Existing model	3.43 ± 0.12	1.95	18.19	3.45 ± 0.07	1.99	17.98
MLP (Site 1)	2.88 ± 0.15	2.38	22.67	2.84 ± 0.09	2.44	31.86
RNN direct (Site 1)	2.62 ± 0.14	2.18	35.24	2.69 ± 0.09	2.51	49.35
RNN force (Site 1)	2.23 ± 0.12	1.92	30.84	2.24 ± 0.07	1.95	28.22
MLP (Site 2)	2.84 ± 0.15	2.40	24.01	2.96 ± 0.09	2.59	29.08
RNN direct (Site 2)	2.46 ± 0.13	2.04	29.66	2.57 ± 0.09	2.62	53.55
RNN force (Site 2)	2.32 ± 0.13	2.16	35.34	2.37 ± 0.08	2.20	37.32
MLP (Site 3)	2.64 ± 0.12	1.94	**17.53**	2.65 ± 0.07	1.90	**16.29**
RNN direct (Site 3)	2.07 ± 0.11	1.75	26.24	2.13 ± 0.07	1.93	39.18
RNN force (Site 3)	**1.82** ± 0.10	**1.60**	26.25	**1.86** ± 0.06	**1.68**	26.85
MLP (Site 4)	6.81 ± 0.20	3.27	25.53	6.86 ± 0.12	3.36	33.43
RNN direct (Site 4)	5.65 ± 0.20	3.22	30.02	5.71 ± 0.12	3.36	38.59
RNN force (Site 4)	5.16 ± 0.19	3.02	30.13	5.26 ± 0.11	3.06	33.36
MLP (Site 5)	6.91 ± 0.26	4.25	30.41	6.97 ± 0.16	4.32	44.70
RNN direct (Site 5)	7.18 ± 0.17	2.78	35.47	7.22 ± 0.11	2.95	39.07
RNN force (Site 5)	6.11 ± 0.21	3.46	36.61	6.22 ± 0.13	3.56	39.00

Table B.27: The results of the apex point difference metric evaluated on site 3. Values are in meters.

	Validation ($n = 1\,000$)			*Test* ($n = 3\,000$)		
Model	**Mean**	**Std.**	**Max**	**Mean**	**Std.**	**Max**
Existing model	1.36 ± 0.08	1.30	12.88	1.37 ± 0.04	1.24	18.98
MLP (Site 1)	0.67 ± 0.06	0.98	9.27	0.67 ± 0.03	0.91	9.10
RNN direct (Site 1)	0.58 ± 0.06	0.99	16.04	0.57 ± 0.03	0.85	10.32
RNN force (Site 1)	0.55 ± 0.05	0.76	9.70	0.54 ± 0.03	0.71	10.95
MLP (Site 2)	0.62 ± 0.06	0.97	10.97	0.64 ± 0.03	0.95	10.71
RNN direct (Site 2)	0.55 ± 0.06	1.01	18.75	0.53 ± 0.03	0.81	12.58
RNN force (Site 2)	0.65 ± 0.07	1.11	20.21	0.61 ± 0.03	0.94	16.75
MLP (Site 3)	0.59 ± 0.05	0.85	**8.86**	0.60 ± 0.04	1.22	46.71
RNN direct (Site 3)	0.50 ± 0.05	0.88	15.68	0.50 ± 0.03	0.77	8.26
RNN force (Site 3)	**0.47** ± 0.04	**0.71**	10.01	**0.49** ± 0.02	**0.67**	**7.05**
MLP (Site 4)	2.75 ± 0.15	2.46	24.71	2.81 ± 0.09	2.53	30.43
RNN direct (Site 4)	2.56 ± 0.14	2.24	19.56	2.62 ± 0.08	2.28	25.87
RNN force (Site 4)	2.59 ± 0.15	2.38	19.51	2.67 ± 0.09	2.47	21.08
MLP (Site 5)	3.09 ± 0.24	3.84	30.51	3.30 ± 0.15	4.26	38.36
RNN direct (Site 5)	2.56 ± 0.12	1.90	14.81	2.55 ± 0.07	1.88	20.79
RNN force (Site 5)	2.64 ± 0.16	2.59	34.69	2.75 ± 0.09	2.63	23.17

B.1.4 Site 4

In this section, the result for models that were evaluated on site 4 are presented.

Table B.28: The p-tests for the different metrics performed on the site 4 test set.

Metric	**P-value**
Mean point	0
Impact point	10^{-277}
Last point	10^{-278}
Apex point	10^{-223}

Table B.29: Pair-wise comparisons using Tukey's range test for the mean point difference metric evaluated on site 4 test set. The critical value was 3.633.

Model #1	Model #2	q	Reject null	Desired
Existing model	MLP (Site 4)	151.586	True	Reject
Existing model	RNN direct (Site 4)	281.448	True	Reject
Existing model	RNN force (Site 4)	283.946	True	Reject
MLP (Site 4)	RNN direct (Site 4)	128.529	True	Reject
MLP (Site 4)	RNN force (Site 4)	131.040	True	Reject
RNN direct (Site 4)	RNN force (Site 4)	2.543	False	Reject

Table B.30: Pair-wise comparisons using Tukey's range test for the impact point difference metric evaluated on site 4 test set. The critical value was 3.633.

Model #1	Model #2	q	Reject null	Desired
Existing model	MLP (Site 4)	21.320	True	Reject
Existing model	RNN direct (Site 4)	43.846	True	Reject
Existing model	RNN force (Site 4)	44.703	True	Reject
MLP (Site 4)	RNN direct (Site 4)	22.275	True	Reject
MLP (Site 4)	RNN force (Site 4)	23.133	True	Reject
RNN direct (Site 4)	RNN force (Site 4)	0.869	False	Reject

Table B.31: Pair-wise comparisons using Tukey's range test for the last point difference metric evaluated on site 4 test set. The critical value was 3.633.

Model #1	Model #2	q	Reject null	Desired
Existing model	MLP (Site 4)	6.791	True	Reject
Existing model	RNN direct (Site 4)	38.802	True	Reject
Existing model	RNN force (Site 4)	41.142	True	Reject
MLP (Site 4)	RNN direct (Site 4)	31.809	True	Reject
MLP (Site 4)	RNN force (Site 4)	34.140	True	Reject
RNN direct (Site 4)	RNN force (Site 4)	2.354	False	Reject

Table B.32: Pair-wise comparisons using Tukey's range test for the apex point difference metric evaluated on site 4 test set. The critical value was 3.633.

Model #1	Model #2	q	Reject null	Desired
Existing model	MLP (Site 4)	32.436	True	Reject
Existing model	RNN direct (Site 4)	39.168	True	Reject
Existing model	RNN force (Site 4)	40.267	True	Reject
MLP (Site 4)	RNN direct (Site 4)	6.484	True	Reject
MLP (Site 4)	RNN force (Site 4)	7.586	True	Reject
RNN direct (Site 4)	RNN force (Site 4)	1.111	False	Reject

Table B.33: The results of the mean point difference metric evaluated on site 4. Values are in meters.

	Validation ($n = 75\,055$)			*Test* ($n = 223\,895$)		
Model	**Mean**	**Std.**	**Max**	**Mean**	**Std.**	**Max**
Existing model	1.34 ± 0.01	1.30	19.70	1.32 ± 0.01	1.30	20.07
MLP (Site 1)	2.81 ± 0.02	2.76	31.10	2.82 ± 0.01	2.80	37.33
RNN direct (Site 1)	2.68 ± 0.02	2.65	21.68	2.68 ± 0.01	2.64	29.88
RNN force (Site 1)	2.91 ± 0.02	2.84	23.78	2.89 ± 0.01	2.82	26.51
MLP (Site 2)	3.12 ± 0.02	2.84	18.99	3.13 ± 0.01	2.88	22.95
RNN direct (Site 2)	3.53 ± 0.03	3.33	23.81	3.50 ± 0.01	3.28	28.54
RNN force (Site 2)	3.49 ± 0.02	3.32	24.65	3.44 ± 0.01	3.27	26.54
MLP (Site 3)	3.34 ± 0.02	3.16	26.24	3.34 ± 0.01	3.18	35.44
RNN direct (Site 3)	2.90 ± 0.02	2.95	24.50	2.89 ± 0.01	2.96	39.43
RNN force (Site 3)	3.41 ± 0.02	3.11	23.38	3.39 ± 0.01	3.08	23.42
MLP (Site 4)	1.01 ± 0.01	1.21	15.67	0.98 ± 0.00	1.17	14.06
RNN direct (Site 4)	0.72 ± 0.01	0.79	10.25	0.70 ± 0.00	**0.77**	**10.72**
RNN force (Site 4)	**0.70** ± 0.01	**0.78**	**8.18**	**0.69** ± 0.00	0.82	13.32
MLP (Site 5)	1.32 ± 0.01	1.62	27.29	1.31 ± 0.01	1.75	52.52
RNN direct (Site 5)	1.03 ± 0.01	1.19	11.62	1.00 ± 0.01	1.21	22.28
RNN force (Site 5)	0.94 ± 0.01	1.19	14.08	0.90 ± 0.00	1.12	16.37

Table B.34: The results of the impact point difference metric evaluated on site 4. Values are in meters.

	Validation ($n = 1\,000$)			*Test* ($n = 3\,000$)		
Model	**Mean**	**Std.**	**Max**	**Mean**	**Std.**	**Max**
Existing model	3.61 ± 0.18	2.92	51.05	3.51 ± 0.09	2.47	20.97
MLP (Site 1)	8.29 ± 0.24	3.91	47.53	8.28 ± 0.13	3.51	28.53
RNN direct (Site 1)	8.40 ± 0.27	4.25	49.73	8.43 ± 0.14	3.99	42.09
RNN force (Site 1)	9.46 ± 0.28	4.52	28.82	9.37 ± 0.16	4.47	27.75
MLP (Site 2)	10.11 ± 0.26	4.18	27.51	10.20 ± 0.15	4.16	32.24
RNN direct (Site 2)	11.57 ± 0.35	5.39	27.39	11.49 ± 0.20	5.28	28.01
RNN force (Site 2)	11.67 ± 0.32	5.19	32.05	11.52 ± 0.18	5.11	29.92
MLP (Site 3)	10.40 ± 0.26	4.09	35.34	10.47 ± 0.15	4.03	31.81
RNN direct (Site 3)	8.81 ± 0.31	4.94	29.00	8.71 ± 0.17	4.86	26.17
RNN force (Site 3)	10.54 ± 0.29	4.63	29.60	10.48 ± 0.16	4.52	26.81
MLP (Site 4)	2.80 ± 0.15	2.46	32.17	2.75 ± 0.08	2.13	25.03
RNN direct (Site 4)	1.98 ± 0.11	1.75	26.34	1.96 ± 0.05	1.48	15.29
RNN force (Site 4)	**1.96** ± 0.10	**1.69**	26.35	**1.93** ± 0.05	**1.45**	**11.26**
MLP (Site 5)	3.34 ± 0.16	2.64	26.39	3.23 ± 0.09	2.55	37.43
RNN direct (Site 5)	3.23 ± 0.14	2.29	**17.53**	3.12 ± 0.08	2.28	21.14
RNN force (Site 5)	2.98 ± 0.15	2.49	26.50	2.87 ± 0.08	2.19	18.75

Table B.35: The results of the last point difference metric evaluated on site 4. Values are in meters.

	Validation ($n = 1\,000$)			*Test* ($n = 3\,000$)		
Model	**Mean**	**Std.**	**Max**	**Mean**	**Std.**	**Max**
Existing model	2.89 ± 0.11	1.84	19.70	2.87 ± 0.07	1.83	20.07
MLP (Site 1)	7.02 ± 0.22	3.48	31.10	7.10 ± 0.13	3.59	37.33
RNN direct (Site 1)	6.64 ± 0.19	3.09	21.68	6.64 ± 0.11	3.06	29.88
RNN force (Site 1)	7.13 ± 0.22	3.62	23.78	7.02 ± 0.13	3.59	26.51
MLP (Site 2)	7.54 ± 0.19	3.05	18.99	7.56 ± 0.11	3.16	22.95
RNN direct (Site 2)	8.70 ± 0.26	4.02	23.81	8.61 ± 0.15	4.01	28.54
RNN force (Site 2)	8.52 ± 0.25	4.01	24.65	8.40 ± 0.14	3.98	26.54
MLP (Site 3)	8.18 ± 0.23	3.65	26.24	8.17 ± 0.13	3.70	35.44
RNN direct (Site 3)	7.70 ± 0.21	3.30	24.50	7.62 ± 0.12	3.39	39.43
RNN force (Site 3)	7.68 ± 0.23	3.65	23.38	7.64 ± 0.13	3.64	23.42
MLP (Site 4)	2.69 ± 0.11	1.81	15.67	2.68 ± 0.06	1.70	14.06
RNN direct (Site 4)	1.84 ± 0.07	**1.07**	10.25	1.82 ± 0.04	**1.06**	**10.72**
RNN force (Site 4)	**1.75** ± 0.07	1.09	**8.18**	**1.75** ± 0.04	1.18	13.32
MLP (Site 5)	3.28 ± 0.15	2.38	27.29	3.26 ± 0.10	2.66	52.52
RNN direct (Site 5)	2.71 ± 0.11	1.69	11.62	2.61 ± 0.06	1.77	22.28
RNN force (Site 5)	2.42 ± 0.11	1.77	14.08	2.34 ± 0.06	1.65	16.37

Table B.36: The results of the apex point difference metric evaluated on site 4. Values are in meters.

	Validation ($n = 1\,000$)			*Test* ($n = 3\,000$)		
Model	**Mean**	**Std.**	**Max**	**Mean**	**Std.**	**Max**
Existing model	0.96 ± 0.06	1.00	11.49	0.98 ± 0.04	0.97	10.66
MLP (Site 1)	1.29 ± 0.14	2.29	57.80	1.27 ± 0.06	1.73	55.87
RNN direct (Site 1)	1.44 ± 0.18	2.86	65.61	1.31 ± 0.05	1.50	11.61
RNN force (Site 1)	1.42 ± 0.10	1.58	11.65	1.41 ± 0.06	1.56	10.88
MLP (Site 2)	1.26 ± 0.08	1.31	9.13	1.25 ± 0.05	1.37	23.19
RNN direct (Site 2)	1.26 ± 0.09	1.37	9.55	1.23 ± 0.05	1.37	9.92
RNN force (Site 2)	1.38 ± 0.10	1.58	10.54	1.37 ± 0.06	1.59	11.79
MLP (Site 3)	1.61 ± 0.13	2.01	40.05	1.59 ± 0.07	1.84	35.57
RNN direct (Site 3)	1.52 ± 0.15	2.38	34.40	1.46 ± 0.07	2.05	37.05
RNN force (Site 3)	1.63 ± 0.11	1.73	11.04	1.61 ± 0.06	1.69	11.93
MLP (Site 4)	0.40 ± 0.04	0.67	5.88	0.44 ± 0.05	1.33	43.18
RNN direct (Site 4)	**0.35** ± 0.03	**0.54**	**4.44**	0.33 ± 0.02	0.51	**5.11**
RNN force (Site 4)	0.36 ± 0.06	1.02	27.87	**0.32** ± 0.02	**0.51**	5.53
MLP (Site 5)	0.57 ± 0.12	1.98	55.95	0.57 ± 0.04	1.16	23.97
RNN direct (Site 5)	0.46 ± 0.05	0.84	12.20	0.46 ± 0.03	0.89	26.71
RNN force (Site 5)	0.37 ± 0.03	0.56	4.96	0.36 ± 0.02	0.56	7.28

B.1.5 Site 5

In this section, the result for models that were evaluated on site 5 are presented.

Table B.37: The p-tests for the different metrics performed on the site 5 test set.

Metric	**P-value**
Mean point	0
Impact point	10^{-62}
Last point	10^{-115}
Apex point	10^{-242}

Table B.38: Pair-wise comparisons using Tukey's range test for the mean point difference metric evaluated on site 5 test set. The critical value was 3.633.

Model #1	Model #2	q	Reject null	Desired
Existing model	MLP (Site 5)	7.732	True	Reject
Existing model	RNN direct (Site 5)	120.559	True	Reject
Existing model	RNN force (Site 5)	179.175	True	Reject
MLP (Site 5)	RNN direct (Site 5)	113.748	True	Reject
MLP (Site 5)	RNN force (Site 5)	172.843	True	Reject
RNN direct (Site 5)	RNN force (Site 5)	59.090	True	Reject

Table B.39: Pair-wise comparisons using Tukey's range test for the impact point difference metric evaluated on site 5 test set. The critical value was 3.633.

Model #1	Model #2	q	Reject null	Desired
Existing model	MLP (Site 5)	2.903	False	Reject
Existing model	RNN direct (Site 5)	6.429	True	Reject
Existing model	RNN force (Site 5)	22.136	True	Reject
MLP (Site 5)	RNN direct (Site 5)	3.524	False	Reject
MLP (Site 5)	RNN force (Site 5)	19.224	True	Reject
RNN direct (Site 5)	RNN force (Site 5)	15.704	True	Reject

Table B.40: Pair-wise comparisons using Tukey's range test for the last point difference metric evaluated on site 5 test set. The critical value was 3.633.

Model #1	Model #2	q	Reject null	Desired
Existing model	MLP (Site 5)	9.511	True	Reject
Existing model	RNN direct (Site 5)	9.973	True	Reject
Existing model	RNN force (Site 5)	22.008	True	Reject
MLP (Site 5)	RNN direct (Site 5)	19.479	True	Reject
MLP (Site 5)	RNN force (Site 5)	31.512	True	Reject
RNN direct (Site 5)	RNN force (Site 5)	12.032	True	Reject

Table B.41: Pair-wise comparisons using Tukey's range test for the apex point difference metric evaluated on site 5 test set. The critical value was 3.633.

Model #1	**Model #2**	**q**	**Reject null**	**Desired**
Existing model	MLP (Site 5)	33.093	True	Reject
Existing model	RNN direct (Site 5)	39.860	True	Reject
Existing model	RNN force (Site 5)	43.108	True	Reject
MLP (Site 5)	RNN direct (Site 5)	6.754	True	Reject
MLP (Site 5)	RNN force (Site 5)	9.991	True	Reject
RNN direct (Site 5)	RNN force (Site 5)	3.237	False	Reject

Table B.42: The results of the mean point difference metric evaluated on site 5. Values are in meters.

	Validation ($n = 81\,639$)			*Test* ($n = 248\,890$)		
Model	**Mean**	**Std.**	**Max**	**Mean**	**Std.**	**Max**
Existing model	1.24 ± 0.01	1.33	14.93	1.25 ± 0.01	1.37	20.30
MLP (Site 1)	3.01 ± 0.02	2.90	24.12	3.04 ± 0.01	2.96	29.01
RNN direct (Site 1)	3.09 ± 0.02	3.13	25.46	3.08 ± 0.01	3.06	27.12
RNN force (Site 1)	2.90 ± 0.02	2.68	18.28	2.97 ± 0.01	2.76	22.42
MLP (Site 2)	3.42 ± 0.02	3.08	21.67	3.49 ± 0.01	3.12	24.81
RNN direct (Site 2)	4.02 ± 0.03	3.68	23.37	4.08 ± 0.02	3.71	28.67
RNN force (Site 2)	3.66 ± 0.03	3.48	23.26	3.68 ± 0.01	3.50	33.55
MLP (Site 3)	3.52 ± 0.02	3.25	21.98	3.57 ± 0.01	3.32	37.00
RNN direct (Site 3)	2.86 ± 0.02	2.88	23.35	2.91 ± 0.01	2.89	20.78
RNN force (Site 3)	3.53 ± 0.02	3.19	25.51	3.57 ± 0.01	3.21	36.23
MLP (Site 4)	1.31 ± 0.01	1.54	17.20	1.32 ± 0.01	1.50	17.14
RNN direct (Site 4)	1.08 ± 0.01	1.29	15.31	1.11 ± 0.01	1.32	20.49
RNN force (Site 4)	1.06 ± 0.01	1.24	20.97	1.06 ± 0.00	1.21	27.11
MLP (Site 5)	1.21 ± 0.01	1.84	29.48	1.23 ± 0.01	1.76	31.73
RNN direct (Site 5)	0.87 ± 0.01	1.07	**14.55**	0.92 ± 0.00	1.11	21.17
RNN force (Site 5)	**0.73** ± 0.01	**0.91**	15.21	**0.76** ± 0.00	**0.91**	**11.68**

Table B.43: The results of the impact point difference metric evaluated on site 5. Values are in meters.

	Validation ($n = 1\,000$)			*Test* ($n = 3\,000$)		
Model	**Mean**	**Std.**	**Max**	**Mean**	**Std.**	**Max**
Existing model	2.43 ± 0.11	1.85	14.64	2.48 ± 0.07	1.84	14.35
MLP (Site 1)	9.31 ± 0.25	4.05	35.97	9.50 ± 0.15	4.13	53.62
RNN direct (Site 1)	9.47 ± 0.27	4.28	31.19	9.58 ± 0.15	4.13	33.34
RNN force (Site 1)	9.36 ± 0.26	4.13	26.99	9.69 ± 0.15	4.21	28.50
MLP (Site 2)	10.47 ± 0.25	4.08	28.37	10.73 ± 0.14	3.97	31.37
RNN direct (Site 2)	11.84 ± 0.33	5.14	31.13	12.17 ± 0.19	5.12	33.44
RNN force (Site 2)	11.29 ± 0.32	5.08	31.29	11.57 ± 0.18	5.15	63.63
MLP (Site 3)	11.06 ± 0.27	4.28	31.23	11.33 ± 0.16	4.34	32.42
RNN direct (Site 3)	9.33 ± 0.29	4.63	29.48	9.61 ± 0.17	4.72	29.75
RNN force (Site 3)	10.86 ± 0.29	4.65	30.59	11.15 ± 0.17	4.66	38.98
MLP (Site 4)	2.80 ± 0.12	1.86	17.26	2.85 ± 0.06	1.81	13.69
RNN direct (Site 4)	2.69 ± 0.11	1.73	**10.72**	2.78 ± 0.06	1.76	**12.90**
RNN force (Site 4)	2.65 ± 0.11	1.84	15.37	2.72 ± 0.08	2.16	59.19
MLP (Site 5)	2.37 ± 0.15	2.35	34.87	2.38 ± 0.08	2.19	27.39
RNN direct (Site 5)	2.20 ± 0.11	1.78	18.55	2.26 ± 0.06	1.75	21.92
RNN force (Site 5)	**1.73** ± 0.09	**1.50**	23.49	**1.74** ± 0.05	**1.41**	13.42

Table B.44: The results of the last point difference metric evaluated on site 5. Values are in meters.

	Validation ($n = 1\,000$)			*Test* ($n = 3\,000$)		
Model	**Mean**	**Std.**	**Max**	**Mean**	**Std.**	**Max**
Existing model	2.71 ± 0.12	1.90	14.93	2.76 ± 0.07	2.02	20.30
MLP (Site 1)	7.29 ± 0.22	3.51	24.12	7.48 ± 0.13	3.59	29.01
RNN direct (Site 1)	7.02 ± 0.25	3.91	25.46	7.09 ± 0.14	3.74	27.12
RNN force (Site 1)	6.79 ± 0.20	3.26	18.28	7.05 ± 0.12	3.33	22.42
MLP (Site 2)	7.95 ± 0.21	3.38	21.67	8.21 ± 0.12	3.41	24.81
RNN direct (Site 2)	9.21 ± 0.28	4.37	23.37	9.46 ± 0.16	4.35	28.67
RNN force (Site 2)	8.39 ± 0.26	4.17	23.26	8.56 ± 0.15	4.14	33.55
MLP (Site 3)	8.46 ± 0.24	3.76	21.98	8.66 ± 0.14	3.86	37.00
RNN direct (Site 3)	7.11 ± 0.21	3.41	23.35	7.33 ± 0.12	3.36	20.78
RNN force (Site 3)	7.88 ± 0.24	3.82	25.51	8.08 ± 0.14	3.78	36.23
MLP (Site 4)	3.57 ± 0.13	2.06	17.20	3.59 ± 0.07	2.00	17.14
RNN direct (Site 4)	2.77 ± 0.11	1.79	15.31	2.85 ± 0.07	1.86	20.49
RNN force (Site 4)	2.71 ± 0.11	1.74	20.97	2.74 ± 0.06	1.71	27.11
MLP (Site 5)	3.04 ± 0.19	3.00	29.48	3.11 ± 0.10	2.84	31.73
RNN direct (Site 5)	2.27 ± 0.10	1.56	**14.55**	2.38 ± 0.06	1.65	21.17
RNN force (Site 5)	**1.88** ± 0.09	**1.39**	15.21	**1.93** ± 0.05	**1.37**	**11.68**

Table B.45: The results of the apex point difference metric evaluated on site 5. Values are in meters.

	Validation ($n = 1\,000$)			*Test* ($n = 3\,000$)		
Model	**Mean**	**Std.**	**Max**	**Mean**	**Std.**	**Max**
Existing model	0.83 ± 0.05	0.88	8.72	0.84 ± 0.03	0.85	6.43
MLP (Site 1)	1.02 ± 0.09	1.39	14.75	1.08 ± 0.06	1.73	56.87
RNN direct (Site 1)	1.06 ± 0.15	2.33	59.85	1.01 ± 0.05	1.27	10.07
RNN force (Site 1)	1.05 ± 0.08	1.32	12.39	1.08 ± 0.05	1.31	9.74
MLP (Site 2)	0.97 ± 0.08	1.25	14.02	0.98 ± 0.04	1.18	9.07
RNN direct (Site 2)	0.95 ± 0.08	1.23	13.60	0.96 ± 0.04	1.19	9.33
RNN force (Site 2)	1.02 ± 0.08	1.34	14.48	1.03 ± 0.05	1.28	9.27
MLP (Site 3)	1.28 ± 0.09	1.45	10.67	1.36 ± 0.06	1.76	28.16
RNN direct (Site 3)	1.12 ± 0.09	1.44	15.36	1.12 ± 0.05	1.39	11.27
RNN force (Site 3)	1.30 ± 0.10	1.58	13.32	1.32 ± 0.06	1.58	11.67
MLP (Site 4)	0.34 ± 0.05	0.75	11.77	0.33 ± 0.02	0.64	20.55
RNN direct (Site 4)	0.30 ± 0.03	0.53	7.36	0.31 ± 0.02	0.49	5.40
RNN force (Site 4)	0.28 ± 0.03	0.47	**3.81**	0.30 ± 0.02	0.47	**5.19**
MLP (Site 5)	0.41 ± 0.06	0.97	13.68	0.41 ± 0.03	0.88	17.02
RNN direct (Site 5)	0.32 ± 0.04	0.59	6.27	0.32 ± 0.02	0.54	6.14
RNN force (Site 5)	**0.27** ± 0.03	**0.45**	3.91	**0.28** ± 0.02	**0.45**	5.55

B.2 Combined results for all sites

The following tables shows the statistical tests that were conducted to determine the statistical significance for the combined results across all sites.

Table B.46: The p-tests for the different metrics performed for when the results were averaged over all sites for a model.

Metric	**P-value**
Mean point	10^{-4}
Impact point	0.002
Last point	0.001
Apex point	10^{-5}

Table B.47: Pair-wise comparisons using Tukey's range test for the mean point difference metric for when the results were averaged over all sites for a model. The critical value was 4.046.

Model #1	Model #2	q	Reject null	Desired
Existing model	MLP	4.864	True	Reject
Existing model	RNN direct	7.533	True	Reject
Existing model	RNN force	8.992	True	Reject
MLP	RNN direct	2.669	False	Reject
MLP	RNN force	4.127	True	Reject
RNN direct	RNN force	1.458	False	Reject

Table B.48: Pair-wise comparisons using Tukey's range test for the impact point difference metric for when the results were averaged over all sites for a model. The critical value was 4.046.

Model #1	Model #2	q	Reject null	Desired
Existing model	MLP	2.342	False	Reject
Existing model	RNN direct	4.823	True	Reject
Existing model	RNN force	6.543	True	Reject
MLP	RNN direct	2.481	False	Reject
MLP	RNN force	4.201	True	Reject
RNN direct	RNN force	1.720	False	Reject

Table B.49: Pair-wise comparisons using Tukey's range test for the last point difference metric for when the results were averaged over all sites for a model. The critical value was 4.046.

Model #1	Model #2	q	Reject null	Desired
Existing model	MLP	1.932	False	Reject
Existing model	RNN direct	4.697	True	Reject
Existing model	RNN force	6.590	True	Reject
MLP	RNN direct	2.765	False	Reject
MLP	RNN force	4.658	True	Reject
RNN direct	RNN force	1.893	False	Reject

Table B.50: Pair-wise comparisons using Tukey's range test for the apex point difference metric for when the results were averaged over all sites for a model. The critical value was 4.046.

Model #1	Model #2	q	Reject null	Desired
Existing model	MLP	8.091	True	Reject
Existing model	RNN direct	9.689	True	Reject
Existing model	RNN force	10.054	True	Reject
MLP	RNN direct	1.598	False	Reject
MLP	RNN force	1.963	False	Reject
RNN direct	RNN force	0.365	False	Reject

B.3 Pretraining

The following tables shows the statistical tests that were conducted to determine the statistical significance of the pretraining experiments. The results for site 1 are shown in tables B.51–B.55 and in tables B.56–B.60 for site 4.

Table B.51: The p-tests for the different metrics performed on the site 1 test set with pretraining.

Metric	P-value
Mean point	0
Impact point	10^{-136}
Last point	0
Apex point	10^{-121}

Table B.52: Pair-wise comparisons using Tukey's range test for the mean point difference metric evaluated on site 1 test set with pretraining. The critical value was 3.315.

Model #1	Model #2	q	Reject null	Desired
No pretraining	Pretraining	322.756	True	Reject
No pretraining	RNN force (Site 1)	340.652	True	Reject
Pretraining	RNN force (Site 1)	17.789	True	Reject

Table B.53: Pair-wise comparisons using Tukey's range test for the impact point difference metric evaluated on site 1 test set with pretraining. The critical value was 3.315.

Model #1	Model #2	q	Reject null	Desired
No pretraining	Pretraining	30.340	True	Reject
No pretraining	RNN force (Site 1)	31.939	True	Reject
Pretraining	RNN force (Site 1)	1.591	False	Reject

Table B.54: Pair-wise comparisons using Tukey's range test for the last point difference metric evaluated on site 1 test set with pretraining. The critical value was 3.315.

Model #1	Model #2	q	Reject null	Desired
No pretraining	Pretraining	55.123	True	Reject
No pretraining	RNN force (Site 1)	58.696	True	Reject
Pretraining	RNN force (Site 1)	3.559	True	Reject

Table B.55: Pair-wise comparisons using Tukey's range test for the apex point difference metric evaluated on site 1 test set with pretraining. The critical value was 3.315.

Model #1	Model #2	q	Reject null	Desired
No pretraining	Pretraining	29.313	True	Reject
No pretraining	RNN force (Site 1)	29.401	True	Reject
Pretraining	RNN force (Site 1)	0.081	False	Reject

Table B.56: The p-tests for the different metrics performed on the site 4 test set with pretraining.

Metric	P-value
Mean point	0
Impact point	10^{-192}
Last point	0
Apex point	10^{-101}

Table B.57: Pair-wise comparisons using Tukey's range test for the mean point difference metric evaluated on site 4 test set with pretraining. The critical value was 3.315.

Model #1	Model #2	q	Reject null	Desired
No pretraining	Pretraining	383.425	True	Reject
No pretraining	RNN force (Site 4)	403.590	True	Reject
Pretraining	RNN force (Site 4)	19.959	True	Reject

Table B.58: Pair-wise comparisons using Tukey's range test for the impact point difference metric evaluated on site 4 test set with pretraining. The critical value was 3.315.

Model #1	Model #2	q	Reject null	Desired
No pretraining	Pretraining	35.500	True	Reject
No pretraining	RNN force (Site 4)	38.915	True	Reject
Pretraining	RNN force (Site 4)	3.401	True	Reject

Table B.59: Pair-wise comparisons using Tukey's range test for the last point difference metric evaluated on site 4 test set with pretraining. The critical value was 3.315.

Model #1	Model #2	q	Reject null	Desired
No pretraining	Pretraining	58.347	True	Reject
No pretraining	RNN force (Site 4)	62.006	True	Reject
Pretraining	RNN force (Site 4)	3.635	True	Reject

Table B.60: Pair-wise comparisons using Tukey's range test for the apex point difference metric evaluated on site 4 test set with pretraining. The critical value was 3.315.

Model #1	Model #2	q	Reject null	Desired
No pretraining	Pretraining	25.868	True	Reject
No pretraining	RNN force (Site 4)	27.641	True	Reject
Pretraining	RNN force (Site 4)	1.762	False	Reject

B.4 Meaning of force prediction output layer

The following tables shows the statistical tests that were conducted to determine the statistical significance of the experiment.

Table B.61: The p-tests for the different metrics performed on the test set for site 1.

Metric	P-value
Mean point	0
Impact point	10^{-20}
Last point	10^{-39}
Apex point	0.427

Table B.62: Pair-wise comparisons using Tukey's range test for the mean point difference metric evaluated on test set for site 1. The critical value was 3.315.

Model #1	Model #2	q	Reject null	Desired
RNN direct	RNN polynomial	20.919	True	Reject
RNN direct	RNN force	59.599	True	Reject
RNN polynomial	RNN force	38.704	True	Reject

Table B.63: Pair-wise comparisons using Tukey's range test for the impact point difference metric evaluated on test set for site 1. The critical value was 3.315.

Model #1	Model #2	q	Reject null	Desired
RNN direct	RNN polynomial	0.294	False	Reject
RNN direct	RNN force	11.953	True	Reject
RNN polynomial	RNN force	11.664	True	Reject

Table B.64: Pair-wise comparisons using Tukey's range test for the last point difference metric evaluated on test set for site 1. The critical value was 3.315.

Model #1	Model #2	q	Reject null	Desired
RNN direct	RNN polynomial	10.341	True	Reject
RNN direct	RNN force	18.912	True	Reject
RNN polynomial	RNN force	8.575	True	Reject