GESTURE RECOGNITION IN TENNIS BIOMECHANICS

A Thesis

Submitted to

the Temple University Graduate Board

In partial Fulfillment

of the Requirements for the Degree

MASTER OF SCIENCE OF ELECTRICAL ENGINEERING

by

Victor C. Espinoza Bernal

Diploma Date: August 2018

Abstract

The purpose of this study is to create a gesture recognition system that interprets motion capture data of a tennis player to determine which biomechanical aspects of a tennis swing best correlate to a swing efficacy. For our learning set this work aimed to record 50 tennis athletes of similar competency with the Microsoft Kinect performing standard tennis swings in the presence of different targets. With the acquired data we extracted biomechanical features that hypothetically correlated to ball trajectory using proper technique and tested them as sequential inputs to our designed classifiers.

This work implements deep learning algorithms as variable-length sequence classifiers, recurrent neural networks (RNN), to predict tennis ball trajectory. In attempt to learn temporal dependencies within a tennis swing, we implemented gate-augmented RNNs. This study compared the RNN to two gated models; gated recurrent units (GRU), and long short-term memory (LSTM) units. We observed similar classification performance across models while the gated-methods reached convergence twice as fast as the baseline RNN. The results displayed 1.4 entropy loss and 30 % classification accuracy indicating that the hypothesized biomechanical features were loosely correlated to swing efficacy or that they were not accurately depicted by the sensor.

Dedication

This project is dedicated the undocumented community, who has given so much more than they've ever taken.

Acknowledgements

Thank you to my family. Thank you to Dr. Obeid and the members of the committee. I would also like to thank the Neural Instrumentation Lab members.

Table of Contents

List of Figures

List of Tables

1 Introduction

The goal of this work is to use machine learning and computer vision to identify which biomechanical movements are best correlated with effective tennis swings based on motion-capture data of subjects playing tennis. In this work we aim to develop a set of machine learning algorithms that will learn proper technique of the three standard tennis swings; forehand, backhand, and serve. Within each of these three tennis swings, we will aim for the algorithm to learn how to properly hit a tennis ball to the left and right side of the tennis court.

In practice, these general directions, or ball trajectories, are referred to as hitting the ball "down-the-line" and "cross-court" when speaking of forehands and backhands. When speaking of the tennis serve, the two common ball trajectories are referred to as "down-the-'T'" or "out-wide". These common ball trajectories will be explained more in depth in the background section. As for the serve portion of the study, this work will analyze the biomechanics of how to serve to two different general directions. Once we collected and annotated skeletal tracking data of skilled tennis players performing these swings, we can extract biomechanical features such as joint angles and distal landmarks to develop ~~a~~ machine learning algorithm that will classify these swings and correlate them to ball trajectories.

In order to obtain a proper learning data set, we chose highly skilled tennis athletes to perform the mentioned tennis swings in the presence of different targets that demonstrate good tennis practice. In this work, we utilized the Microsoft Kinect v2 as the motion capturing tool and a custom designed software suite to record tennis athletes.

Previous research and applications have shown the Kinect’s spatial measurement limitations and how it compares to a laboratory grade motion capture system.

2 Motivation

In recent years, a variety of new technologies have been introduced to tennis that have changed the game. Technologies like "Hawk-Eye" which track the ball's trajectory and landing have started to diminish the need of official line judges during tennis matches since it can determine whether the ball landed in or out. Not only has this made line-calling more accurate and reliable, it has also opened up a field of more in-depth statistical analysis of the sport. This multi-camera system uses triangulation to track the tennis ball up to 2.6 mm accuracy [1] and has attracted research institutions and companies like IBM and SAP to develop statistical analysis software on professional match play. Ultimately, the work here aspired to develop a statistical model based on biomechanical features of tennis players while drawing a bridge to the field of study that focuses on the trajectory of the ball.

In addition, this work could also contribute to the development of tennis teaching software. If the Kinect's skeletal tracking algorithm is accurate enough, future work could use this data to classify a subject's swing technique as proper or poor. Private tennis lessons, work towards to perfecting a customer's swing technique. Furthermore, a professional tennis instructor will provide tips and detailed adjustments to correct technical mistakes, but not to the extent of instruction that requires precision down to the millimeter. Consequently, we anticipated that the accuracy of the Kinect's body tracking would be sufficient for our purposes since millimeter precision is not vital to the instruction or the learning of standard tennis swings.

This work could also contribute to making tennis video games or simulators more realistic. The value of our results will be significant to applications that refer to tennis and that include motion capture. In addition, the skeletal tracking data collected of highly skilled players will record the current biomechanical approach of advanced players as it changes overtime as players and equipment technology evolve.

3 Research Objectives

The overall objective of this work is to determine if a machine learning algorithm can correlate biomechanics to swing efficacy based on the skeletal tracking data obtained with the Kinect. More specifically, to determine if we are able to design a set of algorithms that are capable of recognizing patterns of a proper forehand, backhand, and serve with a desired target across multiple subjects. With this main objective in mind, we can derive specific aims. The success of this study will require us to accomplish the following set of goals:

A. Determine features of body movement that are correlated to a tennis player's swing
B. Design an algorithm that will be able to predict the tennis ball's general trajectory from body tracking data obtained from the Kinect v2.

We will exploit the Kinect's skeletal tracking data with various discrete-time signals processing techniques and pattern recognition techniques in order to meet these goals.

4 Background

4.1 Kinect v2

The Microsoft Kinect v2 comprises of three different sensors; depth sensor, infrared sensor, and an RGB camera, which do not match in resolution or physical location. Microsoft has developed a proprietary machine learning algorithm to detect up to six human bodies with the Kinect using these three layers of data streams [2]. This algorithm is what makes this motion capture technology a markerless approach. The skeletal tracking data is encapsulated in the "body frame" in which we can obtain 3-dimensional data of 25 different body joint centers of each body at a frame rate of up to 30 fps. The joint data includes *x*, *y*, *z* components and. The Kinect-estimated joint centers are demonstrated in Figure 1.

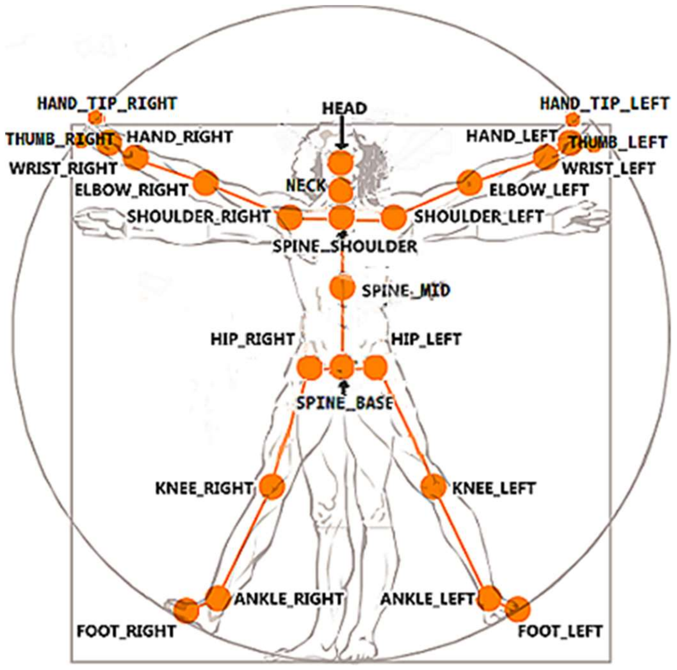


Figure 1: Kinect skeletal tracking [3]

As previously mentioned in the introduction, section 1.0, it is important to be mindful of the Kinect's skeletal tracking's accuracy, and how this accuracy will affect our machine learning algorithm design. Studies show that the Kinect v2's accuracy and frame rate is outperformed by multi-camera laboratory grade motion capture systems, but it is reliable enough to be considered a valid clinical measurement tool [4]. Although the Kinect v2 is a low-grade motion capture system, displaying low accuracy especially in ankle and feet detection [4], it is portable, low-cost, and does not require calibration or the use of markers on subjects like a Vicon system of 2 mm accuracy at 100 fps. Without the constraints of markers and a research lab setting, subjects that will partake in our study will be able perform as freely as they would during match play as long as they stay in the Kinect's field of vision and within four meters of the sensor.

The Kinect v2 outputs 3-dimensional space coordinates for 25 joints of each body tracked with reference to the actual device. In more detail, the origin (point 0,0,0) of the 3-dimensional coordinate system obtained by the Kinect is the actual location of the Kinect as shown in Figure 2. The *z* component represents the orthogonal distance from a joint to the Kinect. The *y* component represents the orthogonal distance from the joint to the floor plane, and the *x* component entails the distance of the joint that extends to the right or left of the sensor. With the discussion of the coordinate system, it also important to note that the floor plane is also encapsulated within the Kinect's body frame. This floor detection is also a component of Microsoft's proprietary skeletal tracking since the *y* coordinates are dependent on it.

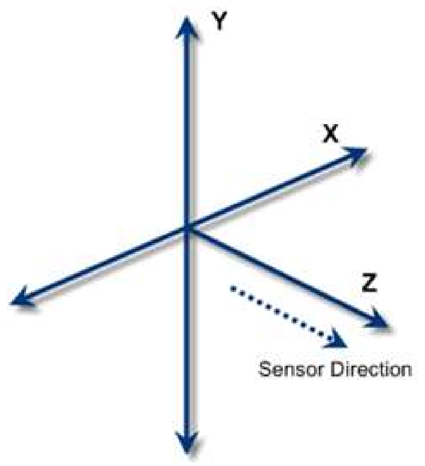


Figure 2: Kinect Cartesian Coordinate System [5]

Since the position of the subject with respect to the Kinect will not be constant for every swing across all subjects, we will have to measure distances and angles with respect to the subject's own body. Additionally, we can manipulate the data by removing parts of the *x* and z components of the 3D coordinates in a way to relocate the body coordinates of into a fixed position for each frame of the data collected.

4.2 Tennis Background

Generally speaking, an intermediate tennis player is able to hit the three standard tennis swings; forehand, backhand, and serve. This level of player should also have control over the general direction of the ball from all of these previously mentioned swings. These directions refer to the ball's trajectory with respect to the subject who hit the ball. The forehand and backhand, often referred to as groundstrokes, can be struck "down-the-line" or "cross-court". The serve can also be struck in two general targets, "down-the-T", which refers to the center of the court, and "out-wide" which pertains to aiming the serve at the singles side line that belongs to that service box. The clarification of theses swings and associated directions are important since they can vary depending on the dominant hand of the subject and on which side of the court the subject who is hitting the ball from. This set of tennis swings and general directions create a general baseline for tennis knowledge and ability, as well as a better understanding of the objective of this work.

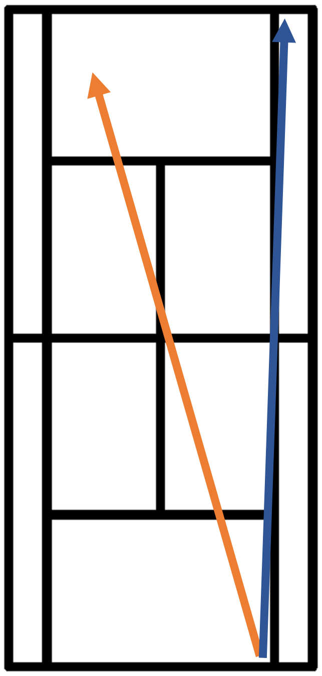


Figure 3: Forehand down-the-line (blue) crosscourt (orange) for a right-handed player

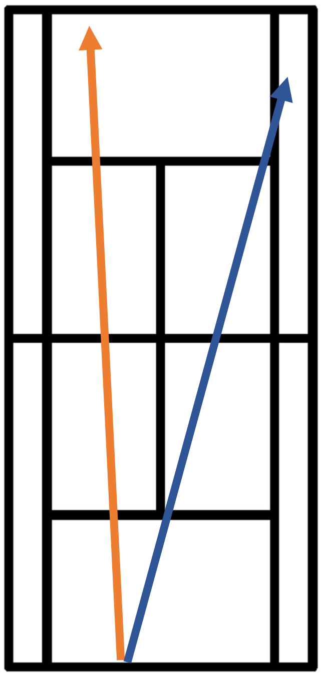


Figure 4: Backhand down-the-line (orange) crosscourt (blue) for a right-handed player

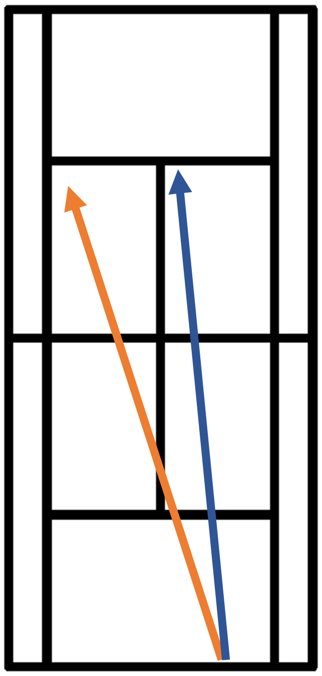


Figure 5: Deuce side serve down-the-T (blue) out-wide (orange)

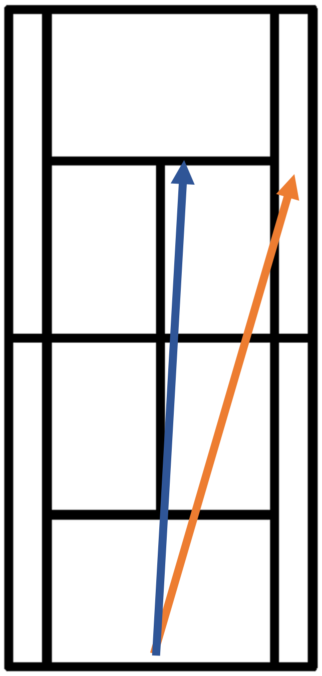


Figure 6: Ad Side down-the-T (blue) out-wide (orange)

4.3 Pattern Recognition Techniques

In this section we will introduce the pattern recognition approaches that we anticipated using in this work. We will consider two methods that are feature dependent, support vector machines and hidden Markov models.

4.3.1 Support Vector Machines

Support vector machines are a supervised machine learning technique used to classify data between two classes. SVMs are essentially the use of a decision hyperplane of dimension (p -1) to classify p-dimensional data points in the feature space. This decision hyperplane is designed based on a subset of the training data points referred to as *support vectors* which lie close to estimated boundary between the two classes*;* the remaining training data samples becomes essentially ignorable. The design procces of an SVM classifier starts by considering the following basic classifier in slope intercept form displayed by equation 1.

$$f(x) = \beta x + b = 0 \tag{1}$$

If we impose the constraints of equation 2 where $y_j = \pm 1$ as the classification variable, and x_j denotes data points, we can define the decision hyperplane becomes by finding the optimal value of β. The data points that are chosen as the support vectors satisfy equation 3.

$$y_j(\beta x_j + b) \geq 1 \tag{2}$$

$$y_j f(x_j) \geq 1 \tag{3}$$

$$y_j(\beta x_j + b) = 1 \tag{4}$$

To optimize our classifier with the given constraints we solve for optimal β using LaGrange multipliers. The form of our optimized classifier then becomes in form of equation 5.

$$f(x) = \sum_{j=1}^{N} \alpha_j y_j \, x_j \cdot x_i + b \tag{5}$$

If the classes are not linearly separable, the training data are transformed to a higher dimension where the data can be separated by a linear surface. Kernel functions can be used to avoid the need for explicitly mapping each data point into a higher dimensional space. This use of a kernel function leads us to the final form of the classifier as shown in equation 6.

$$f(x) = \sum_{j=1}^{N} a_j y_j K\left(x_j, x_i\right) + b \tag{6}$$

Commonly used kernel functions in practice are Linear, RBF, and Polynomial kernels shown in equations 7, 8 and 9 respectively:

$$k(x, y) = x_i x_j + c \tag{7}$$

$$k(x, y) = e^{-\gamma \|x_i - x_j\|^2} \tag{8}$$

$$k(x, y) = \left(x_i x_j + c\right)^d \tag{9}$$

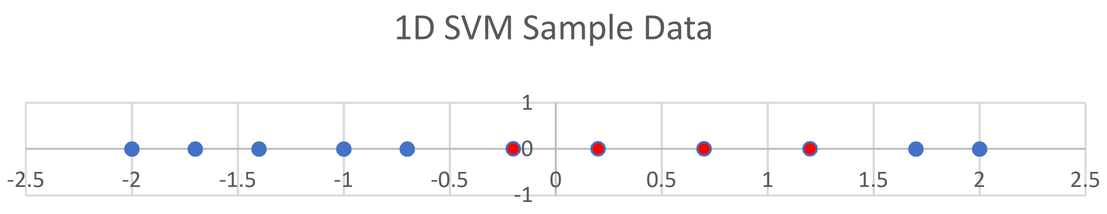


Figure 7: 1D Sample Data

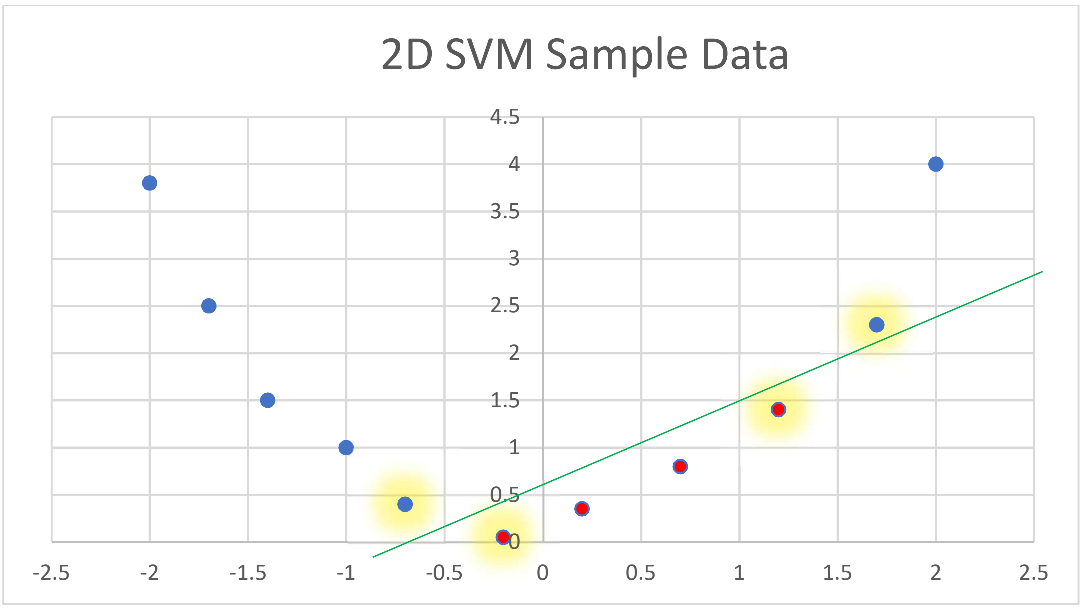


Figure 8: 2D SVM Sample Data

4.3.2 Random Forest Algorithm

In order to comprehend the random forest approach, we will briefly go over the decision tree, which is the building block of the random forest algorithm. The decision tree is a classification technique that consists of a set of questions and decides a path that will ultimately lead to the classification of the data. The questions can be thought of as nodes or visualized as a point where branches of the tree split. As data travels up a finite number of nodes it will ultimately end up in a leaf, where the data is classified. An example of a decision tree classifier is demonstrated in figure 10 of the given sample data found in figure 9 Although this is a practical and computationally inexpensive approach it is easy to overfit the data, meaning that complex trees will not classify non-training data very well.

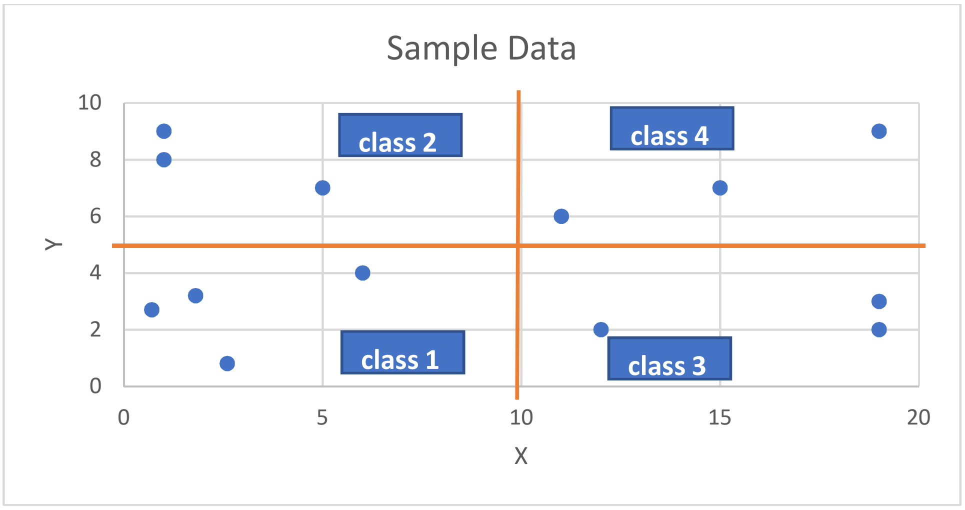


Figure 9: Random Forest Data

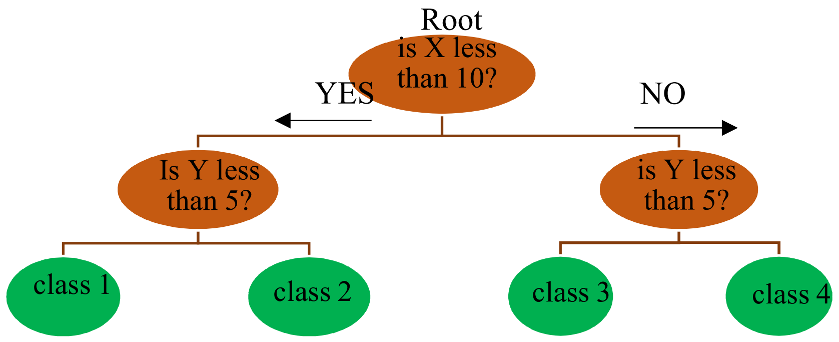


Figure 10: Decision Tree Classifier

In order to quantify for the efficacy of the trees we will implement ways to measure the impurity in the classifications. Two common methods to quantify this impurity in classification regions are by measuring entropy and the Gini index, shown in Equation 10 and 11 respectively. In the worst scenario, the entropy measurement will return a value of 1, indicating that the probability of the classes w_j in that split or region is equally likely. On the other hand, the Gini index takes the sum of the square of the probability of each class in the region or split and subtracts this sum from 1. If two classes are equally likely in a region or split, the Gini index would return a value of 0.5.

$$i(N) = -\sum_{j} P(w_j)\, log_j\, P(w_j) \tag{10}$$

$$i(N) = 1 - \sum_{j} P^2(w_j) \tag{11}$$

As the number of features of the training data increases, the decision tree becomes greater in depth, indicating more nodes and more classification leaves that will decrease in variance. In order to account for these weaknesses in decision trees, it is common to practice pruning. Pruning entails setting a predetermined maximum depth, or path from root to leaf, for the decision tree, as well as setting a minimum number of samples of training data per leaf, or class. In addition, we introduce the random forest algorithm as an "ensemble" machine learning method in which we deploy various weak learners, individual decision trees. This algorithm is essentially a set of multiple decision trees that can be used for classification and regression. In use as a classification method, the data sample will go through each of the decision trees while we keep count of the classifications, leaves, that point to each class in the whole forest. The random forest will ultimately classify the new data object based on the leaf with the most votes. The general workflow of random forests is depicted in 11.

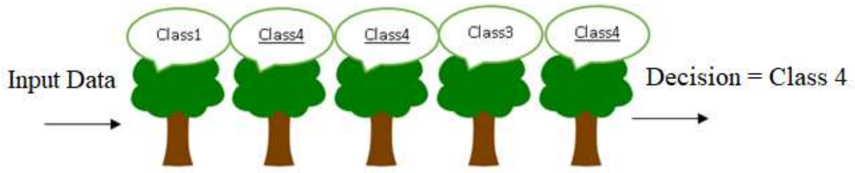


Figure 11: Decision Tree Workflow

4.3.3 Hidden Markov Model

Hidden Markov models, HMMs, are a well-established technique in machine learning with applications involving sequences of data like speech recognition and handwriting recognition. HMMs are represented by a graph of finite number of states, which are connected by transitions. At each of these states, there are two sets of probabilities; transition probabilities and emission probabilities. Transition probabilities represent the chances of transferring to the next state while the emission probabilities represent the chance of emitting an output, or symbol, at the current state. Figure 6.2 demonstrates an example of a network of 3 hidden states denoted by w_i, with transition probabilities a_{ij}, and emission probabilities b_{jk} of emitting discrete symbol v_k. HMMs are a double stochastic process, the red component of the figure indicates the visible, or observable, portion of the model, while the black components of the model indicate the hidden, non-observable part. These transition and emission probabilities create a transition matrix and emission matrix respectively and are denoted in equations 12 and 13. Equation 14 displays the probabilities of starting at each state, which are referred to as the initial probabilities. An HMM is considered ergodic if every element of matrix A is a non-zero value. In other words, each state in the model is reachable from any other states in a finite number of transitions.

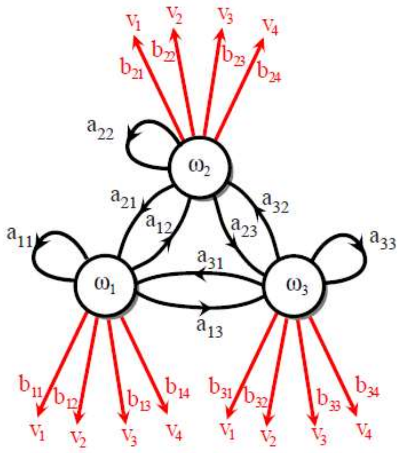


Figure 12: HMM

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \tag{12}$$

$$B = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \end{bmatrix} \tag{13}$$

$$\Pi = [\pi_1 \quad \pi_2 \quad \pi_3] \tag{14}$$

HMMs are expressed in the form of equation 15.

$$\lambda = (A, B, \Pi) \tag{15}$$

If we apply this machine learning technique to sequential data in time, like speech or motion capture data, we can focus on a subset of HMM's that is non- ergodic. This type of HMM, known as the Left-Right Model or Bakis model, is more suited for sequential data. Since it is not possible to go back in time, the transition probability of states become the option to stay at the current state or move on to the next one as depicted in figure 13.

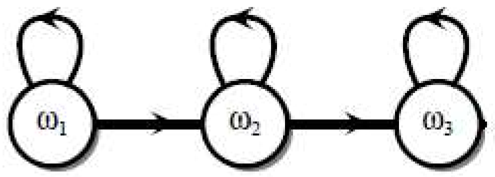


Figure 13: Bakis Model

In this case, the transition matrix becomes in the formm of Eq. 16 while the initial state probabilities become in the form in Eq. 17

$$A = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 \\ 0 & a_{22} & a_{23} & 0 \\ 0 & 0 & 0 & a_{33} \end{bmatrix} \tag{16}$$

$$\Pi = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \tag{17}$$

4.3.4 Multi-layer Perceptron

Since all previously mentioned machine learning algorithms take calculated biomechanical features from Kinect based estimations as inputs, we heavily rely on the limited accuracy of the sensor's skeletal tracking. In this section we introduce a common deep learning approach, the multilayer perceptron, which can take raw data as input. The building block of the multilayer perceptron, commonly referred to as MLP or feed-forward neural network, is the neuron displayed in orange in figure 14. The input layer consists of all the x_m nodes, the hidden layer is made up of the computational unit, and output layer is comprised of $\hat{y}$. Output $\hat{y}$ is computed with the use of an activation function φ. The activation function is typically designed to return a binary value based on a threshold, or a probability value ranging from 0 and 1.

$$\hat{y} = \varphi\left(\sum_{i=1}^{m}(w_i x_i) + bias\right) \tag{18}$$

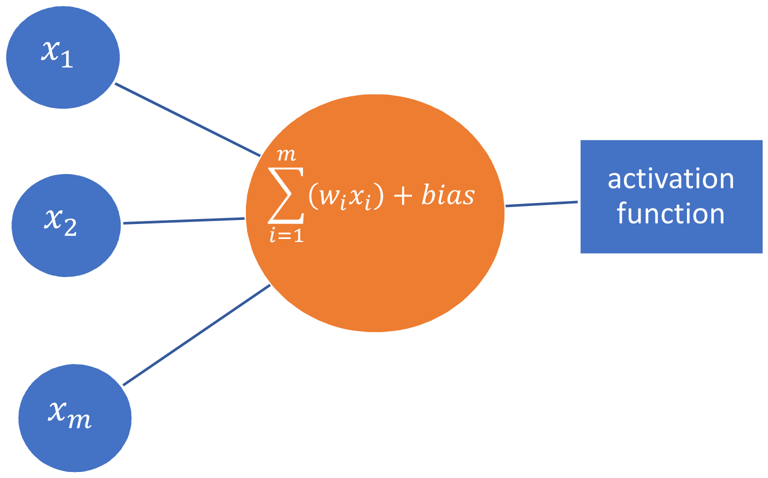


Figure 14: MLP

The training process consists of two phases, the feedforward phase and the back-propagation phase. In the feedforward mode, the weights are estimated or randomized, and the input is introduced to the network and are computed in the forward direction. The output $\hat{y}$ is produced and then compared to the true value of y and the error is calculated between the two using a cost function $J(w)$. The backpropagation phase then takes this computed error and adjust the weights across the network to minimize the error using stochastic gradient descent. Stochastic gradient descent utilizes the partial derivative of the loss function with respect to parameters, referred to as the gradient shown in equation 19.

$$\nabla_w J(w) = \frac{d\big(L(y,\hat{y})\big)}{dw} \tag{19}$$

After accumulating error over a batch of training samples, stochastic gradient descent updates the set of weights w at a learning rate η as shown in equation 20. The ultimate goal of the training is to go through each training sample and solve for the optimal values of each weight to minimize the error.

$$w = w - \eta \nabla_w J(w) \tag{20}$$

The output layer of neural network model consists of a densely connected layer, where all the units in the last hidden layer are all connected to the units in the output layer. For categorical classification models, the output layer contains the same number of output units as there are categories, which are all activated by the SoftMax activation function. In categorical prediction tasks, the loss function is defined by categorical cross entropy show in 21. Where M is the number of classes, $p_{o,c}$ is the predicted probability observation *o* of class c, and *y* is a binary variable which indicates if classification *c* of observation *o* is in fact a correct classification. Alternatively, for binary or decimal target prediction, the output layer contains one single densely connected unit.

$$\mathcal{L}(\hat{y}, y) = -\sum_{c=1}^{M} y_{0,c} \, log(p_{o,c}) \tag{21}$$

4.4 Deep Learning

Further investigation led to the realization that the majority of the machine learning algorithms mentioned in section 4.3 lose temporal information with the exception of the HMM. In light of this realization we redirected our efforts towards deep learning strategies. We further investigated more appropriate models that are capable of classifying sequences of spatial-temporal data. More specifically, algorithms capable of classifying sequences of variable length since the duration of tennis swings fluctuate. In this section we describe what the development of a deep learning task entails.

The objective of deep learning tasks, where one is given a pair of inputs and targets to design an algorithm, is referred to as *supervised learning* task. "Unsupervised learning, where no training signal exists at all, and the algorithm attempts to uncover the structure of the data by inspection alone."[6]. This scope of this study will be solely focused on supervised sequence classification. A supervised learning task requires a training set of input-target pairs (x,y) where x may be an element, vector, or matrix from space X, and y is an element of the correct target space Y. Additionally, another input-target disjoint set is created and referred to as the *validation set*. In practice, an additional set is reserved separately from the training set and validation set, referred to as the *test set*, and is tested on once optimization is finalized. The goal is to use the input-target pairs in the training set to design an algorithm that will minimize the error function between the algorithm output and the correct target in both training and validation sets. In development the trained algorithm is tested on the validation set, input-target pairs outside of the training. The algorithms considered in this work are deep recurrent networks, which are parametric models that are fine-tuned in training to obtain a specified value or target. The end goal is to optimize the training parameters in the network to minimize the error across all three mentioned sets in uniform fashion.

In practice these parameters, or weights, are fine-tuned incrementally throughout the whole training set and then tested in the validation. One iteration of training the algorithm on every sample in the training set and testing it on the validation set is referred to as an epoch. As one increases the number or epochs a common obstacle that arises is overfitting. This means that the algorithm's loss decreases over the number of epochs in the training set, while the loss on the validation set increases or becomes stagnant. As a result of overfitting, the model learns very specific features that can be found the training set, but these features are not as defined or evident in the validation set. In the deep learning field there are a wide variety of model architectures which have shown to be advantageous in classifications of specific types of data.

4.4.1 Supervised Sequence Learning

In this work we aim correlate biomechanical features over time with tennis ball trajectory. More specifically, classify these features sampled at 30 Hz with the Microsoft Kinect. In this section we will go more into detailed discussion of algorithms designed for sequential data labeling which have shown to be efficient previous work. Additionally, discuss the obstacles that arise in classifying a sequence of temporal features. These deep learning models are; the recurrent neural network, the gated recurrent network, and the Long-Short Term memory model.

4.4.2 Recurrent Neural Network

The recurrent neural network, RNN, does not deviate too far from the previously mentioned model in section 4.3.4, the MLP. "A recurrent network is an extension of conventional feed forward neural network, which is able to handle variable length sequence input" [7]. The RNN and MLP differ in that the hidden units in an RNN have an additional option to transition back into itself. In other words, a transition probability from the hidden unit at time *t* back to itself at *t+1* exists. At the beginning of the time sequence, this transition probability is set to zero, but previous studies show that it can be advantageous to initialize this as a non-zero value [8].

$$h_t = \begin{cases} 0, & t = 0 \\ \emptyset(h_{t-1}, x_t) & t \neq 0 \end{cases} \tag{22}$$

The hidden state equation then becomes a function of the input *x* at time *t*, an input weight, the recurrent weight, the previous hidden state and a bias as displayed in equation 23.

$$h_t = sigmoid(W_x x_t + W_r h_{t-1} + b_a) \tag{23}$$

This generative model is used to generate a series of outputs based on input sequences, and is commonly found in applications that generate sentences, or lyrics. Using this approach, the probability of the input sequence of length T becomes

$$p(x_1, \dots, x_T) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2) \dots p(x_T|x_1, \dots x_{T-1}) \tag{24}$$

Using a special output symbol to represent the end of the sequence, this model is capable of handling variable length sequences by modeling a conditional probability distribution at each time step as show in Eq.25.

$$p(x_t|x_1, \dots x_{t-1}) = h_t \tag{25}$$

Alternate to the generative model, the predictive RNN trains on a sequential data to predict one singular target. The activation of a hidden unit in a predictive RNN becomes and additive function over the length of the sequence, flattened by a differentiable nonlinearity, or gate as shown in Eq. 26. This predictive model is commonly applied to applications that predict a single specific target value like weather or stock prices.

$$y_t = g\left(\sum_{t=1}^{T} w_x x_t + \sum_{t=1}^{T} w_r h_{t-1}\right) \tag{26}$$

4.4.3 Back Propagation Through Time

RNNs take in feature data as function of time, and classiffiy each sequence into a single binary or categorical value. Despite the newly considered sequential data, we still depend on a cost function to quantify the model's error for the input sequence. Similarly, we optimize our network using stochastic gradient decent, or the derivative of the cost function with respect to the weight path. This process becomes more complex because with every time step in the training sequence, the gradient may become much larger or smaller.

Back propagation through time, or BPPT, can be thought out of as a traditional stochastic gradient descent over an RNN that has been unrolled through time. Once we define a loss function for a simple RNN, BPTT entails computing the gradient for every time step in the training sequence. Ultimately, the loss or error for one training sequence becomes the sum of the error over every time step.

$$\frac{dE}{dW} = \sum_{i<k<t} \frac{dE_t}{dx_t} \tag{27}$$

In order to calculate the error at each time step, we are required to apply the chain rule since each calculated output at *t* depends on the previous output at t-1. Figure 15 demonstrates how the error at the 4th step of a sequence of length 5, becomes a composite function of previous activations h_t.

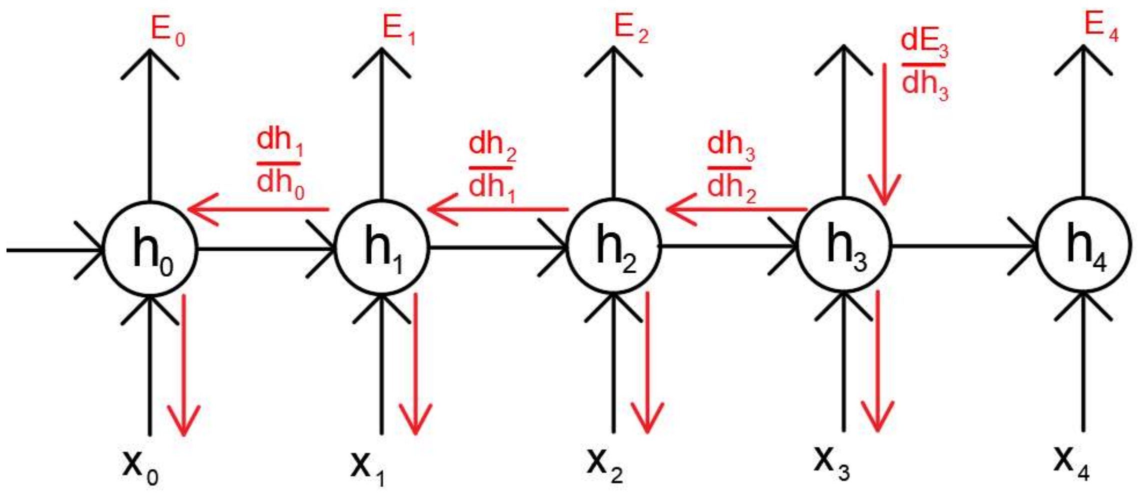


Figure 15: BPTT at t=3

This method begins at the highest value of t, *T*, and ends at the lowest value of t, which is why it is referred to as backpopagation through time. If we consider the chain rule formula (28) , we can take the product of all gradients in each time step across the training time sequences.

$$\frac{d}{dx}[f(g(x))] = f'(g(x))g'(x) \tag{28}$$

Therefore, the gradient at time *t* over a sequnce of length *T*, becomes a function of all timesteps *k*, where k is less than *t*, as shown in equation 29.

$$\frac{dE_t}{dW} = \sum_{i<k<t} \frac{dE_t}{dx_t}\frac{dx_t}{dx_k}\frac{dx_k}{dW} \tag{29}$$

Intuively, we can observe that by multiplying T many numbers together that are greater than 1 can potentially leads to the exploding gradient problem. On the opposite hand, if we multiply many number less than zero we are led to vanishing gradient that converges to zero. These two obstatcles are two of the main reasons why RNNs are so difficult to train [9].

4.4.4 Gated Recurrent Unit

The gated recurrent unit is an apporach which augments the basic RNN with two gating units, the *update* gate and the *reset* gate. Each of these two gates feature an activation and a trainable parameter creating a small submodel at every node of the hidden layers of the network. In back propagation, these parameters, or weights, are trained to determine how to retain relevant information as well as to filter out irrelevant data.

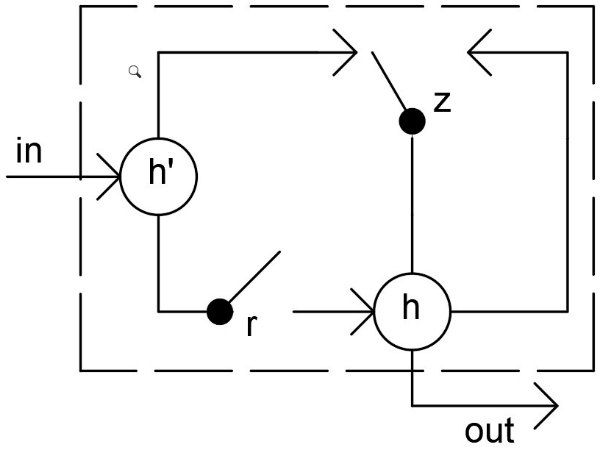


Figure 16: Gated Recurrent Unit

The update gate z controls how much information from the previous hidden state will carry over to the current hidden state. [10]

$$z_t = sigmoid(W_z x_t + U_z h_{t-1}) \tag{30}$$

Secondly, the reset gate r_t, also referred to as relevance gate, which can be aplied to the hidden state before or after the matrix multiplication depicted in Equation 31. This gate will advise the cell wether to update with new values or not. The candidate activation h'_t, is the new suggesed value that is trying to get through reset gate

$$r_t = tanh(W_r x_t + U_r h_{t-1}) \tag{31}$$

$$h'_t = tanh\big(W x_t + U(r_t \odot h_{t-1})\big) \tag{32}$$

Ultimately, the activation at time *t* combines the candidate activation and the previous activation at *t-1* using the current update gate value

$$h_t = z_t\,\widehat{h_t} + (1 - z_t)h_{t-1} \tag{33}$$

4.4.5 Long Short Term Memory

Long-short term memory, LSTM, units are also closely related to the simple RNN. Similar to GRU models, the LSTM architecture is comprised of units at each node of the hidden layer as opposed to a single perceptron. In the LSTM approach, each unit contains three gates; the *input*, *output* and *forget* gates. "The multiplicative gates allow LSTM memory cells to store and access information over long periods of time, thereby mitigating the vanishing gradient problem." [6] Additionally, LSTM utilizes these gates to retain a sufficiently large gradient in order to keep optimizing the network model towards an error minimum. The LSTM gates shown in figure 17 are mathematically defined as follows:

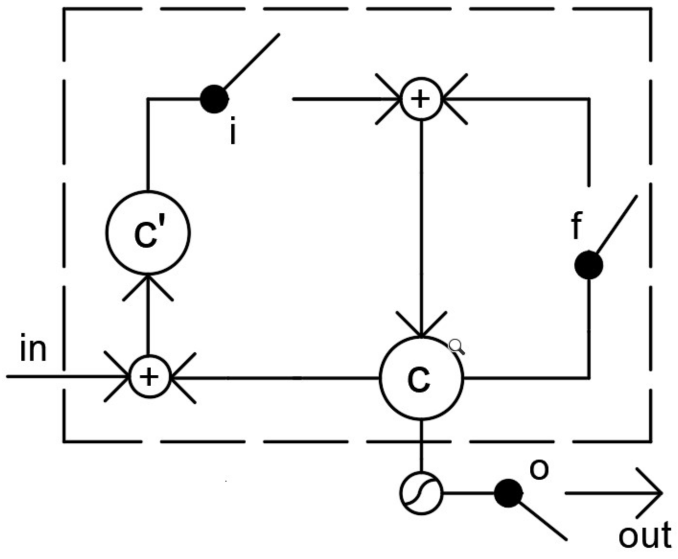


Figure 17: Long Short-Term Memory Unit

The *input* gate, which informs the cell on what incoming data is useful while also contributing to the cell state c_t.

$$i_t = \sigma(W_i\, x_t + U_i h_{t-1} + V_i c_{t-1}) \tag{34}$$

The *forget* gate, which is utilized to help the cell let go of what information is irrelevant

$$f_t = \sigma\big(W_f x_t + U h_{t-1} + V_f c_{t-1}\,\big) \tag{35}$$

At the core of the LSTM unit, one can find the *constant error carousel*, CEC, which closely resembles a standard RNN. “By recirculating activation and error signals indefinitely, the CEC provides short-term memory storage for extended time periods.”[11] The output of the becomes cell’s memory which is then defined as a function of the forget gate and input gate values shown in equation 36.

$$c_t = f_t c_{t-1} + i_t \hat{c}_t \tag{36}$$

where $\hat{c}_t$ is the candidate new cell memory candidate at time *t* regularized by the value of the input gate. $c'_t$is defined using x_t, the input at time *t,* and h_{t-1}, the activation at time *t-1* as shown Equation 37.

$$c'_t = tanh(W_c x_t + U_c h_{t-1}) \tag{37}$$

where the activation of the unit h_t,is defined by the hyperbolic tangent of the cell memory at time *t* element-wise multiplied by the output gate value as show in Equation 38

$$h_t = o_t * tanh(C_t) \tag{38}$$

where the output gate, o_t, whose purpose is to regulate the amount of memory content to be visible by the rest of the network at time t is defined by 39

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + V_o c_t) \tag{39}$$

4.4.6 Algorithm Discussion

It is important to note that the GRU designed by Cho et all in 2004 has two gates; the reset and update gate. More importantly to note, if we set the reset gate to 0 and update gate to 1, we end up with the simple RNN model. On the other hand, the LSTM architecture brought forth by Hochreiter and Schmidhuber in 1997, has 3 gates; input gate, output gate and forget gate. In similar fashion, if the input gate and forget gate are open, set to 1, then the CEC is updated with every new input and recurrent state just like an RNN unit. In nature, gates are differentiable nonlinearities operating on trainable parameters that transmit or receive information to or from the rest of the network, whose task is to be more selective in accessing sequential data. The LSTM unit has a memory cell and the GRU does not, but both have an additive component in their updating features from *t* to *t+1*. The purpose of the LSTM memory cell is to overwrite part of existing memory with incoming data. In similar fashion the GRU uses the update gate value to add a factor of old and add new data. Since the GRU has less trainable parameters than the LSTM, it is less computationally expensive to train and a faster algorithm when implemented. Although the two mentioned augmented algorithms in 9.3 and 9.4 are more efficient in learning long temporal range approaches than the basic RNN approach, there is very little evidence to support the idea that one consistently outperforms the other.

4.5 Optimization

This section is dedicated to cover the techniques that were investigated and implemented in training the recurrent network models covered in the background section.

4.5.1 Regularizers

Regularizing is practice of normalizing the activations or parameters of a specific hidden layer of n trainable parameters and apply penalties to the loss function of the network model. The way regularization is applied to the model's loss function is by adding a regularization term to the loss function. In order to monitor the effect of the of the regularization term on the loss function, we define a small regularization parameter, λ.

$$L_r = L(y, \hat{y}) + \lambda \sum_{1}^{n} \theta_j \tag{40}$$

Regularizers help in when the model learns too many features, and what it does is decrease or regulate the magnitude of the learned parameters. In doing so, we tame the learned parameters in the training set to not be as dominant or deterministic in the model's performance on the test set and ultimately attempt to prevent overfitting.

$$\eta \frac{dL_r}{d\theta} = \eta \frac{d\left(L(y, \hat{y}) + \lambda \sum_1^n \theta_j\right)}{d\theta} = \frac{dL(y, \hat{y})}{d\theta} + \frac{d\left(\lambda \sum_1^n \theta_j\right)}{d\theta} \tag{41}$$

4.5.2 Batch Normalization

In practice, a big setback in training a model is overfitting due to when there is a shift in distribution between the training set and the validation set. In more detail, the internal covariate shift is defined at the layer level where the distribution of the input of a hidden layer and the distribution of its output differs or changes after the activation function. As a model increases in hidden layers, the distributions after each layer might differ, and may lead to complicating the optimization process. A way to mitigate this is to whiten the output of each layer. Whitening entails linearly transforming the input matrix to have zero means, unit variances, and decorrelated. This entails computing the covariance matrix along with its inverse square root.

Previous works shows that whitening between each hidden layer would result in obtaining an input of fixed distribution for each hidden layer and ultimately remove the effects of the internal covariate shift. Additionally, whitening transformation between layers has also demonstrated to lead to faster model convergence.[12] However, this transformation is computationally expensive, and would significantly slow down the network in producing a prediction, even more so as the number of hidden layers increases.

Instead of performing whitening transformations at every layer, previous work suggests that batch normalization also helps mitigate the ill effect of the covariate shift[13]. It is also important to note that batch normalization can be implemented before or after applying the non-linearity of the hidden layer. In training we use the minibatch computed mean and variance, μ_B and σ_B^2 respectively and shown in Equation 42. During inference, validation, the true value of the mean and variance of the population, E[x] and var[x] respectively are used and demonstrated in Equation 43.

$$\widehat{x_\iota} = \frac{x - \mu_B}{(\sigma_B^2 + \in)^{\frac{1}{2}}} \tag{42}$$

$$\widehat{x_\iota} = \frac{x - E[x]}{(Var[X] + \in)^{\frac{1}{2}}} \tag{43}$$

4.5.3 Dropout Regularization

Dropout can be considered as another way of regularizing a neural network and has shown to be effective in preventing overfitting in previous work[14]. Dropout disconnects certain connections to a subset of units within a hidden layer and refreshed after every iteration of weight optimization. In practice, a subset of hidden units within a hidden layer are given a probabilistic value to be omitted from the model. Deactivating hidden units in training forces the model to generate various paths to obtain a desired target and ultimately lead to better results in validation set. When testing the model's performance in the validation set, dropout is not used. This technique can be thought of as training multiple models within one fixed-sized model. Ultimately, dropout regularization tackles overfitting prevention as well as model combining, which is an approach of training different models separately with different data and combing them into one.

4.5.4 Gradient Clipping

When one encounters the exploding gradient problem, it is good practice to clip the gradient in order to keep training the parameters toward a loss minimum. In practice, once the gradient is calculated it is normalized by a threshold value and the L1 or L2 norm of the gradient

$$\nabla_w J(w) \ x \frac{threshold}{norm_L(\nabla_w J(w)\,)} \tag{44}$$

Equations 45 and 46 define the L1 and L2 norm of a vector x of size n.

$$L1\ norm = \sum_{1}^{n} |x_n| \tag{45}$$

$$L2\ norm = \left(\sum_{1}^{n} x_n^2\right)^{0.5} \tag{46}$$

4.5.5 Momentum-Augmented SGD

In this section we introduce augmented versions of stochastic gradient descent that have shown to outperform the standard SGD approach in section 4.2.2 in sequential data applications. More specifically, we review two first-order techniques that apply the concept of momentum to SGD in order to accelerate through regions of low curvature in the loss function. The concept of classical momentum, CM, can be described as iteratively accumulating a directional vector towards a minimum of a given function that requires optimization. For deep learning applications, the objective function becomes loss function of the model. Equation 46 shows how CM is applied to gradient descent, where μ is a predefined momentum coefficient ranging from 0 to 1, and v_t is the directional vector. Next, we can use the directional vector, v_{t+1},to iteratively update the parameters towards a local minimum as shown in Eq 47.

$$v_{t+1} = \mu v_t - \eta \frac{dL(y,y)}{dW} \tag{47}$$

$$\theta_{t+1} = \theta_t + v_{t+1} \tag{48}$$

The Nesterov Accelerated Gradient, or NAG, is similar to classical momentum approach, expect for the difference in the way that NAG calculates the gradient. This technique computes the gradient by first adding the cost function to the momentum coefficient μ multiplied by velocity vector v_t. The gradient then becomes the derivative of this sum with respect to the model parameters, and it is used to update the parameters identically to first approach, shown in Equation 49

$$v_{t+1} = \mu v_t - \eta \frac{d(L(y,y) + \mu v_t)}{dW} \tag{49}$$

Previous experiments show that momentum accelerated SGD, with a strong μ coefficient had favorable effects in the optimization of RNN model [15].

5 Methods

In this section we describe how we collected data from highly skilled athletes performing the three standard tennis swings and managing the raw extracted motion capture data.

5.1 Protocol Design

The protocol for this study will start off by setting the Kinect v2 on a tripod located on the corner of the tennis court. This corner is defined by the baseline and the outer edge of one of the two doubles alleys and will vary depending on which direction the subject is facing while executing a tennis groundstroke. On the opposite side of the net, we will set up a grid of 8.4x7.5 feet rectangles. A cone will be set in the center of one these rectangles which will define the target rectangle. Once the target is set and the subject has warmed up, we will give the subject 10 attempts try to hit the target while the research team records the landing of each attempt in the defined grid. If a subject missed by not making contact with the ball or by hitting the ball into the net, they will be granted one supplementary attempt. Each attempt will consist of the co-investigator hitting a ball at slow pace from the opposite side net to the subject who will be waiting for the ball at the baseline. The ball will be fed to the subject in such a way that the subject will be able to strike the ball while staying the Kinect's field of vision while the Kinect is recording. There will be six different targets set up for the forehand portion of the study, as well for the backhand portion of the study. Similarly, the subject will be given 10 attempts for each target.

For the serve portion of the study, a different size grid will be set up consisting of 3.5x3.5 feet squares and will be set on a service box diagonally across from the subject. A cone will then be placed at the center of one of the rectangles to define the target rectangle. Starting from the “deuce side” of the court, the subject will stand 3 feet from the center of the baseline and take 10 attempts at hitting the target while performing their serve motion. For each side of serve position, “deuce” and “ad” side, the subject will be required to hit two different targets from each of these serving sides. From each of the two sides, one target will be located in the corner of the service box that is closest to the center of the court, the second target will be located on the opposite corner of the box close to one of the doubles alley. By placing targets in opposite sides of the courts we can expect to be able to denote biomechanical features that correlate to the different trajectories of the tennis ball. Figure 18 below displays the grid with reference to a regulation size tennis court, the targets of the study, and the locations where the Kinect would be placed for recording depicted by the blue dots.

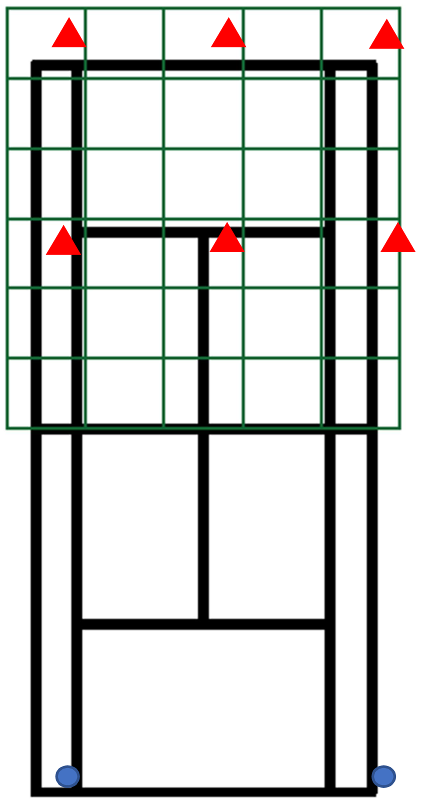


Figure 18: Ground stroke portion of the study where the red triangles represent targets

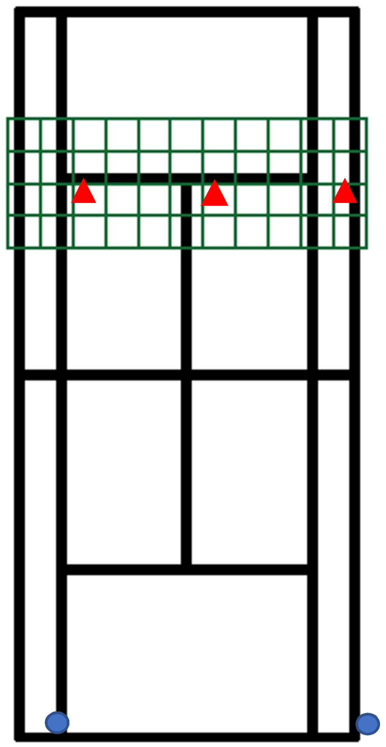


Figure 19: Serve portion of the study

5.2 Signal Processing

The Kinect's frame rate can fluctuate below the advertised 30 frames per seconds. This variable frame rate is not controllable by the user and mainly relies on the hardware components of the sensor and the user's computer resources. In order to compensate for fluctuation in frame rate, the Kinect raw data will be interpolated up to the advertised frame rate of 30 frames per second. And then it will be passed through a 4th order Butterworth filter. Once the data is interpolated and filtered, we will gather the data of the actual swing. Taking into consideration different swing speed across subjects, we have determined that the dominant side wrist of the participant has denotable maxima and minima that indicate the start and end of each swing. More specifically, the Y component of the wrist joint signal demonstrates that tennis athletes accelerate their dominant arm by swinging in a more vertical direction. These spatial-temporal landmarks were considered in the annotative process of the data.

Since the position of the subject with respect to the Kinect will not be constant for every swing across all subjects, we will have to measure distances and angles with respect to the subject's own body. Alternatively, we can manipulate the data by removing part of the *x* component of the 3D coordinates in a way to relocate the body coordinates into a fixed position for each frame of the data collected.

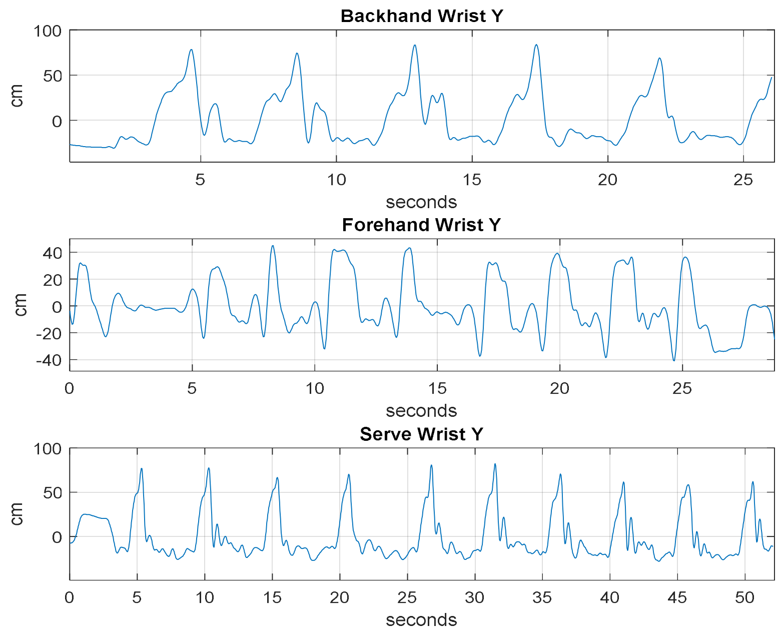


Figure 20: Dominant Hand Y-Component

5.3 Feature Extraction

In this section we describe how we extracted biomechanical features from the acquired data from section 5.1 to the deep learning techniques mentioned in 4.4 and apply them to a set hypothesized biomechanical features in order to meet our objectives. One of the features were hypothesized to be indicative of the direction of the ball is the angular displacement of the subject's dominant-side hip and shoulder. In other words, we predict that the angular displacements of the two mentioned joints will have a correlation to the amount of change in direction required to hit a present target. Figure 18 denotes the rotation of the hips in the Kinect's XZ plane where the blue denotes the frame after the black frame. The midpoint between hips and between shoulders are defined by *spine base* and *spine shoulder* points respectively.

$$angular\ velocity = \frac{d\theta}{dt} \tag{50}$$

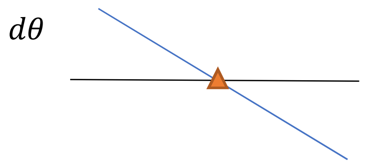


Figure 21: Angular Velocity

We also predicted that the angles created by the subjects' ulna and humerus during the swing are correlated to the distance between the subject and the target as well as the direction. More specifically, that the mentioned angle will become more obtuse throughout the swing as the subject aims for a target that is further away from him or herself. In order to create an input for this model we will take into consideration three angles from the dominant side of the subject as a function of time. These three angles were calculated using three joints from the skeletal data and Equation 51. For each calculated angle, we will need the three joints with one common joint to create two vectors.

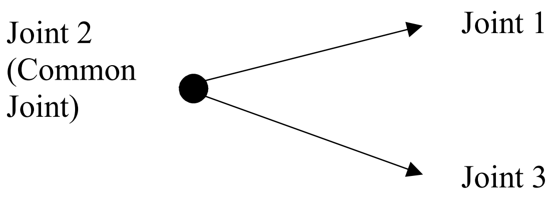


Figure 22: Joint Angle

$$\emptyset_i = cos^{-1}\left(\frac{u \cdot v}{||u||\ ||v||}\right) \tag{51}$$

Table 1: Joint Angle Features

Angle	**Joint 1**	**Joint 2**	**Joint 3**	**Sequence**
$\emptyset$	Shoulder	Elbow	Wrist	$\emptyset_1, \emptyset_2, \ldots . \emptyset_T$
θ	Elbow	Shoulder	Neck	$\theta_1, \theta_2, \ldots \theta_T$
ω	Elbow	Shoulder	Hip	$\omega_1, \omega_2, \ldots \omega_T$

5.4 Data and Software Development

In this work, data was gathered across 14 eligible participants using a Dell Inspiron 17 7559 laptop with a dedicated GPU, the Microsoft Kinect and a custom designed software suite. The motion capture data was recorded in a .csv file in the presence of a single target at a time. In other words, each .csv file contained the ten attempts of a subject swinging at one of the targets mentioned in 5.1. The data was annotated using a custom designed software suite in MatLab based on the maxima and minima features in 5.2. These minima and maxima were not as observable in the backhand and serve swings due to the dominant hand not being in the sensor's line of sight during swing execution. Consequently, the remainder of the study focused on forehand target prediction. In the annotation process, the full-body motion capture data was visually inspected, before correlating every peak and valley to the start and end of a swing. The ball landing location was then mapped to one of five categories across the width of the tennis court shown in table Consequently, each annotated swing was stored in its own .csv file. Along with it's true target outcome.

Table 2: Target Classification

Court reference	**Cross-Court**	**Left-of center**	**Center**	**Right of center**	**Down-the-Line**
Target mapping	4	3	2	1	0

The feature extraction was developed in Matlab to export angles and distances mentioned in the section 5.3**,** along with the swing's trajectory classification. The features were then imported into the scripts containing the models mentioned in the section 9.0 using python 3. This work features the RNN as the baseline model whose performance was then compared to GRU and LSTM's performance. All three models were implemented using the Keras API with a TensorFlow backend. The constraints on each model was a maximum of two hidden generative hidden layers, while maintaining the number of trainable parameters in the same order of sequences gathered. Stacking hidden generative layers has shown to be effective at the sub-model level to achieve predictive tasks. Ultimately, the workflow of this study was defined as shown figure 23.

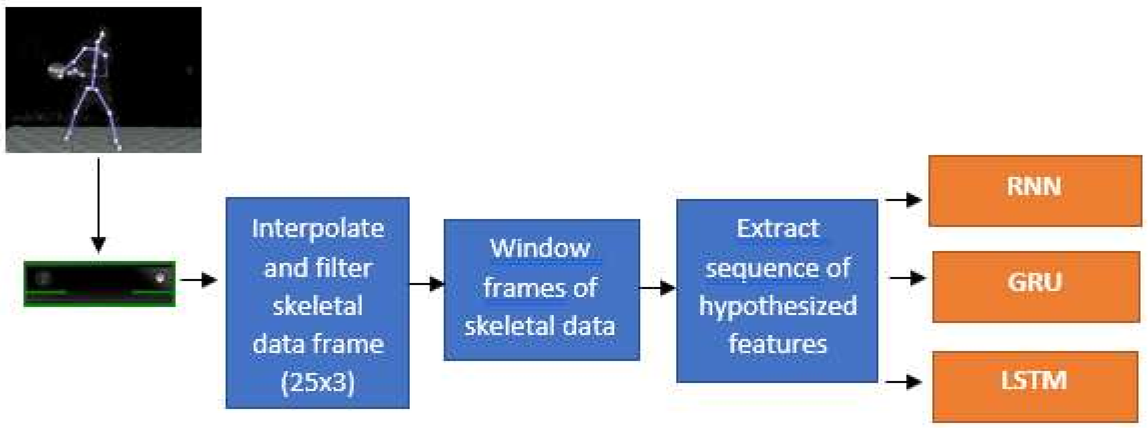

Interpolate and filter skeletal data frame (25x3)
Window frames of skeletal data
Extract sequence of hypothesized features
RNN
GRU
LSTM

Figure 23: Design cycle

This work considered a categorical output layer in attempting ball trajectory prediction. This attempt featured an output layer of five units with the use of a SoftMax activation function. These five units represent the five categories in the grid defined in table across the width of a tennis court. The targets of the swing sequences were one-hot encoded, mapped to a binary vector with length of the number of categories where the only bit set to 1 corresponds to the correct target as shown in table. Using this approach, the loss function implemented was categorical cross-entropy shown in equation 21.

Table 3: One-hot encoded targets

Decimal value	**One-hot encoded**
0	[1 0 0 0 0]
1	[0 1 0 0 0]
2	[0 0 1 0 0]
3	[0 0 0 1 0]
4	[0 0 0 0 1]

6 Results

In this section we visualize the hypothesized features in the training set data that we expected to be correlated to ball trajectory as well as analyze the subject population and the data target distribution. The participant population is analyzed in table 4 displaying binary variables that were permissible in the data gathering process. The target outcome distribution is displayed in figure 27 across all targets and subjects.

Table 4: Participant Population

Gender (Male/female)	Dominant Side (Righty/Lefty)	Session Environment (Indoor/Outdoor)
9/5	14/0	4/10

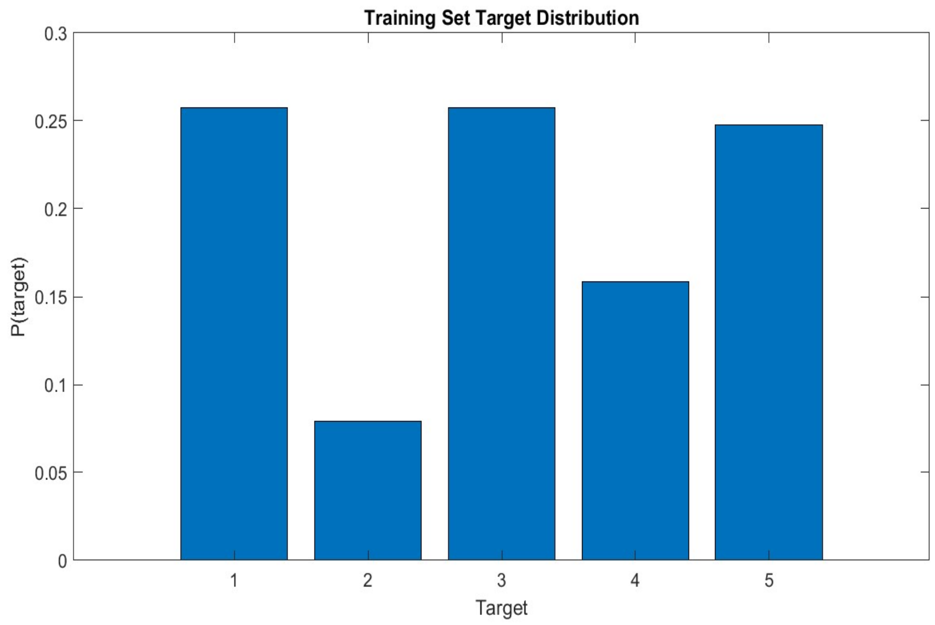


Figure 24: Training set target distribution

In chapter 3 we aimed to design an algorithm that would be able to classify biomechanical feature sequential data into one of two general directions, cross-court and down-the-line. We attempted to find visible depiction between the two, to test our hypothesized features as well as answering for the reliability of the sensor. The features displayed were taken from the swings that resulted in target 0 and target 4 over a two-second time window, demonstrated in blue and red respectively. The first three features evaluated, were the ones mentioned in table 1; elbow flexion, shoulder flexion and axillary, as shown in figures 24, 25 and 26. In addition, we evaluated how likely these angles were over this twosecond window. This was done to visualize the classifier's task in selecting between the two opposing targets.

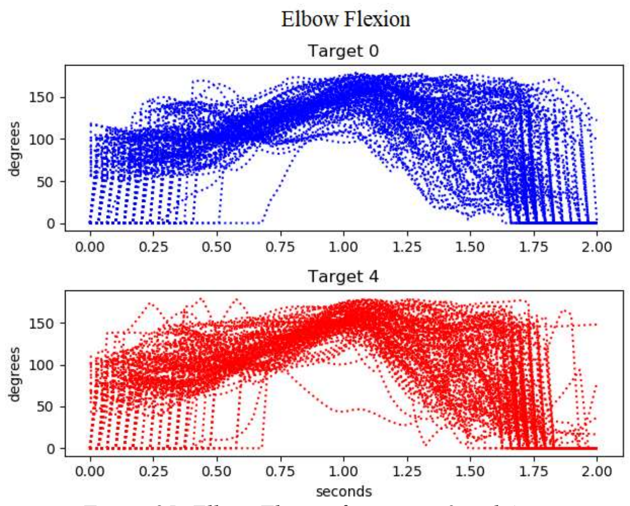


Figure 25: Elbow Flexion for targets 0 and 4.

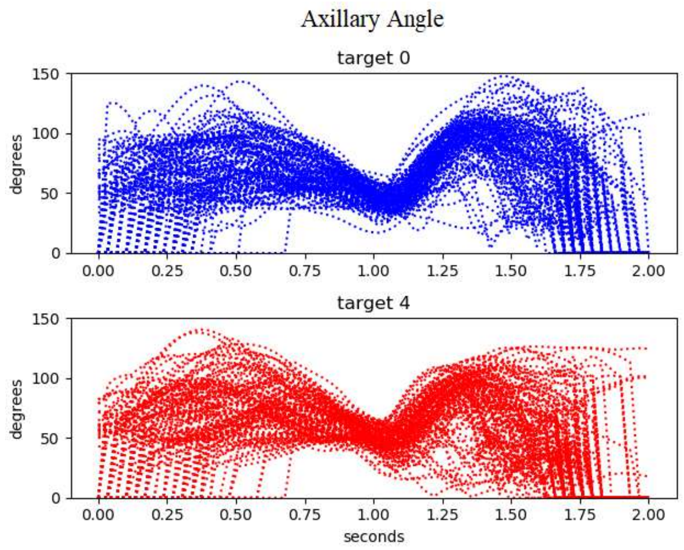


Figure 26: Axillary Angle for targets 0 and 4.

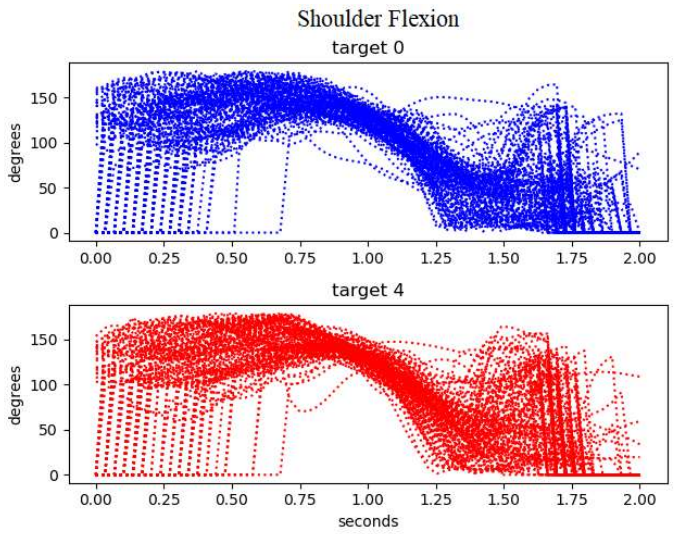


Figure 27: Shoulder Flexion for targets 0 and 4.

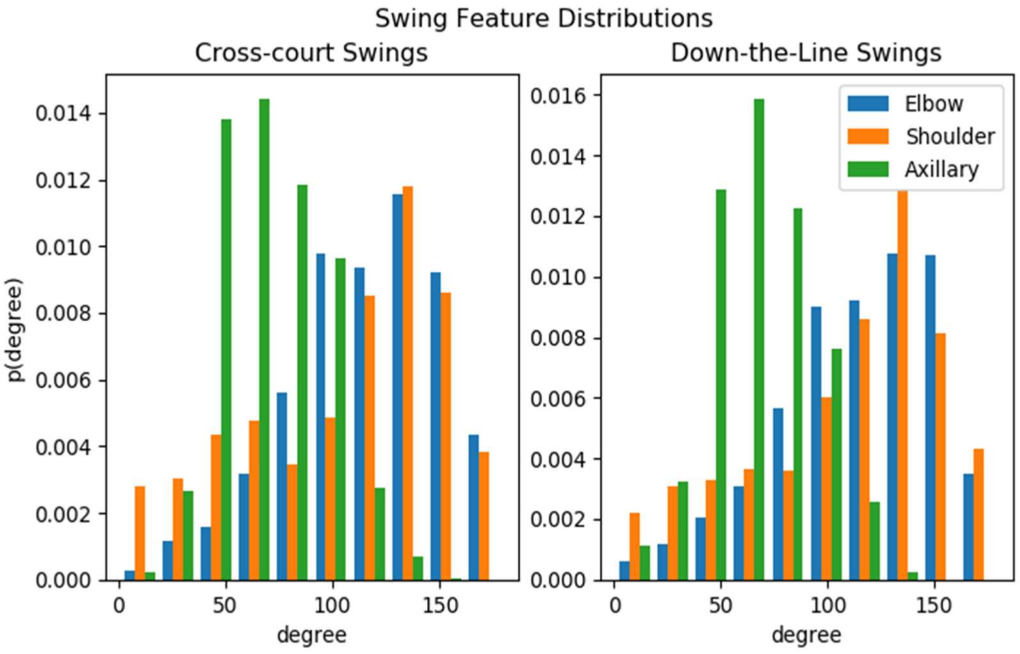


Figure 28: Swing Feature Distribution

We were not able to observe significant differences in any of the proposed feature-target pairs, or across the feature distributions in figure 28. In attempt to find more indicative features, we extracted more features shown in figures 29, 30, and 31.

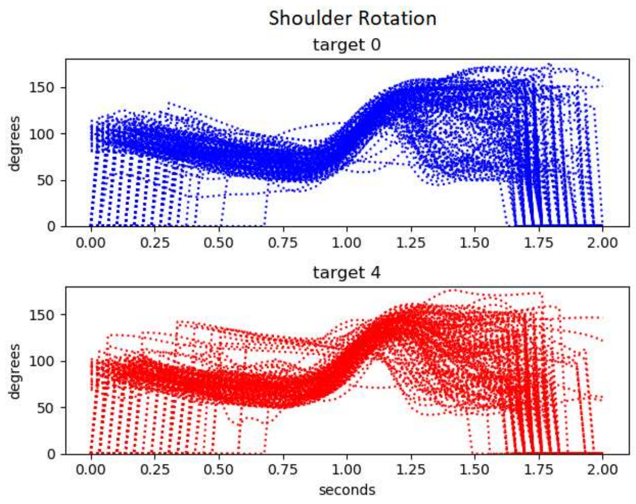


Figure 29: Shoulder Rotation with respect to Kinect

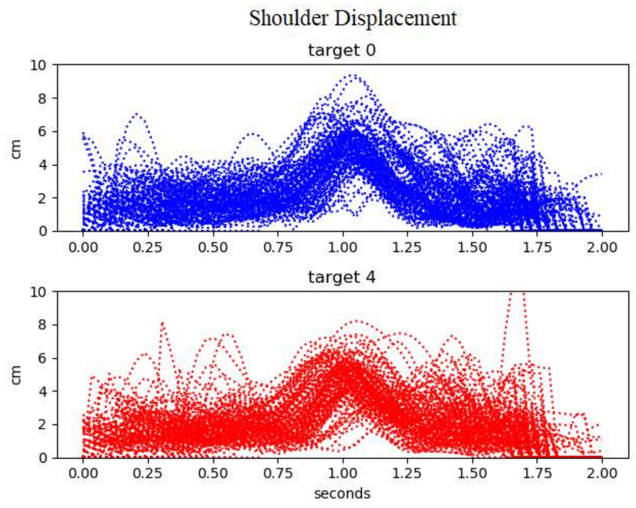


Figure 30: Shoulder Displacement in 3D space throughout Swing

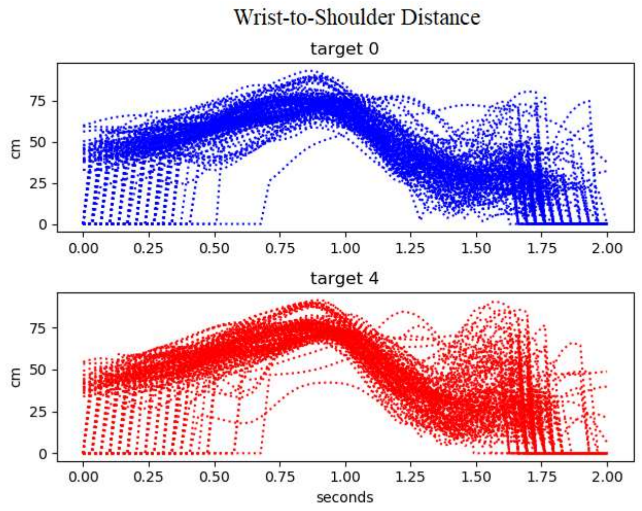


Figure 31: Wrist-to-Shoulder Distance

The training data was then time-aligned and randomly shuffled across the whole set. We then trained and tested the model with the mentioned sequential feature data. The models each individually comprised RNN, GRU, LSTM units. Our results in terms of loss and categorical accuracy are shown in figures 32 and 33 respectively.

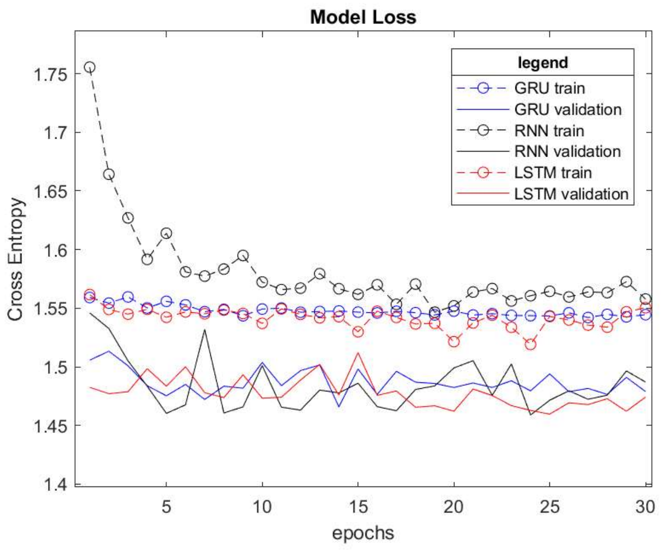


Figure 32: Model Loss

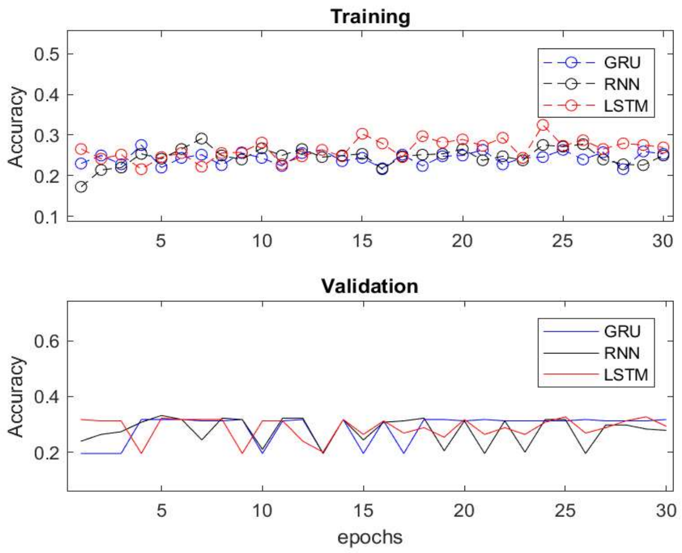


Figure 33: Model Accuracy

7 Discussion

In figure 32 we demonstrate the loss of the three models over 30 epochs. Our results indicate that the that regularization techniques successfully prevented the models from overfitting. The plot also demonstrates that RNN converges after10 epochs, while the gated-unit models converge at 5 epochs. Batch normalization and NAG did in fact accelerate the learning process but ultimately led to convergence at 1.4 cross entropy loss. We can deduct that from this loss convergence difference that the gated approaches were beneficial in our application.

The progression of classification accuracy in each model over 30 epochs is displayed in figure 26 with the training and validation results in individual sublots. We can see that in training the accuracy subplot remained below 30%, but the LSTM accuracy reached higher than 30% at epoch 25. In the validation process we see similar pattern of 30%, but we noted that the GRU model converged the fastest out of the three models. We also noticed that the stability of the GRU accuracy was more reliable after it reach convergence at epoch 15. The 33% accuracy is predicted to be a result from the training set containing targets 0, 2 and 4, close to equally likely in outcome as shown in the distribution shown in figure 24.

In attempt to obtain better results we implemented a method in which we fed the model a batch of sequences of similar target. In detail, we trained the models on batches of singular target across randomized subjects. This was done in aim to force the model to learn features correlated to target outcome across different subjects. All three models were then evaluated twice; once with batches of random subjects swinging at random defined targets, and another with batches of random subjects swinging at one defined target. We did not expect to see a big increase in classification accuracy for targets 1 and 3 since they're probabilistic low as well as uneven in favor of 3. Our results showed improvement not quantitatively in terms of entropy or accuracy a but in distance between predicted and corrected target. We evaluated our results by analyzing the confusion matrices of each model trained in both mentioned manners. The confusion matrix is a 2D matrix used to visualize the predictions vs the correct outcome, where the correct outcome is always the *ith* element of the *ith* row. We observed the most favorable results from the RNN and LSTM models. The comparison in loss between target-batch training and random batch training can be seen in appendix A.

Table 5: Random batch training vs. Single-target training confusion matrices

Model	**Random Batch**	**Single-Target Batch**
RNN	$\begin{array}{ccccc} 0 & 0 & 0 & 0 & 43 \\ 0 & 0 & 0 & 0 & 6 \\ 0 & 0 & 0 & 0 & 43 \\ 0 & 0 & 0 & 0 & 18 \\ 0 & 0 & 0 & 0 & 26 \end{array}$	$\begin{array}{ccccc} 12 & 0 & 7 & 0 & 24 \\ 3 & 0 & 1 & 0 & 2 \\ 11 & 0 & 6 & 0 & 26 \\ 5 & 0 & 5 & 0 & 8 \\ 10 & 0 & 2 & 0 & 14 \end{array}$
GRU	$\begin{array}{ccccc} 43 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 \\ 43 & 0 & 0 & 0 & 0 \\ 18 & 0 & 0 & 0 & 0 \\ 26 & 0 & 0 & 0 & 0 \end{array}$	$\begin{array}{ccccc} 0 & 0 & 0 & 0 & 43 \\ 0 & 0 & 0 & 0 & 6 \\ 0 & 0 & 0 & 0 & 43 \\ 0 & 0 & 0 & 0 & 18 \\ 0 & 0 & 0 & 0 & 26 \end{array}$
LSTM	$\begin{array}{ccccc} 37 & 0 & 0 & 0 & 6 \\ 6 & 0 & 0 & 0 & 0 \\ 31 & 0 & 0 & 0 & 12 \\ 9 & 0 & 0 & 0 & 9 \\ 19 & 0 & 0 & 0 & 7 \end{array}$	$\begin{array}{ccccc} 34 & 0 & 0 & 0 & 9 \\ 5 & 0 & 0 & 0 & 1 \\ 23 & 0 & 0 & 0 & 20 \\ 6 & 0 & 0 & 0 & 12 \\ 15 & 0 & 0 & 0 & 11 \end{array}$

8 Conclusion

The primary goal of this work to was to determine features of body movement that correlated to tennis swing efficacy. In doing so, we would have been able to artificially interpret a humanistic skill with a neural network model and an accessible motion capture sensor. We tested our hypothesized features by applying them as inputs to three established supervised sequence learning techniques. In this work we implemented the RNN as the baseline model whose performance was then compared to the GRU and the LSTM models developed by, Cho and, Hochreiter and Schmidhuber respectively. These two gate-augmented models have shown to be advantageous in training as well as recognizing longer range time dependencies better than RNN models.

Our results show that the gated-unit approaches did in fact converged faster than the RNN model at similar loss values, but the gated-models did not outperform the RNN in classification accuracy. In methodically generating batches in training, we observed more favorable values which indicated that there was some correlation between the features and swing efficacy. In analyzing the features, we can consider that the feature selections are not indicative of swing-target pair, while also not being accurately depicted by the Kinect sensor. From a deep learning perspective, we believe that the model could benefit from a convolutional layer as feature-generating layer prior to the RNN units. From a signal processing standpoint, we expect the model could benefit from implementing adaptive filtering techniques that are used in pose estimation. Considering that the GRU and LSTM units are adaptive to longer range time dependencies, and that they converged faster than the RNN, we can infer that this temporal-memory feature is beneficial in classifying variable length swing patterns to an extent.

References Cited

[1] D. Whiteside and M. Reid, “Spatial Characteristics of Professional Tennis Serves with Implications for Serving Aces: A Machine Learning Approach,” *J. Sports Sci.*, vol. 35, no. 7, pp. 648–654, Apr. 2017.

[2] “BodyFrame Class.” [Online]. Available: https://msdn.microsoft.com/en-us/library/windowspreview.kinect.bodyframe.aspx. [Accessed: 14-Dec-2017].

[3] “JointType Enumeration.” [Online]. Available: https://msdn.microsoft.com/en-us/library/microsoft.kinect.jointtype.aspx. [Accessed: 14-Dec-2017].

[4] K. Otte *et al.*, “Accuracy and Reliability of the Kinect Version 2 for Clinical Measurement of Motor Function,” *PLOS ONE*, vol. 11, no. 11, p. e0166532, Nov. 2016.

[5] “Coordinate Spaces.” [Online]. Available: https://msdn.microsoft.com/en-us/library/hh973078.aspx. [Accessed: 14-Dec-2017].

[6] A. Graves, “Supervised Sequence Labelling,” in *Supervised Sequence labelling with Recurrent Neural Networks*, Springer, 2012, pp. 5–13.

[7] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling,” *ArXiv14123555 Cs*, Dec. 2014.

[8] M. Zimmermann, J. C. Chappelier, and H. Bunke, “Offline Grammar-Based Recognition of Handwritten Sentences,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 5, pp. 818–821, May 2006.

[9] R. Pascanu, T. Mikolov, and Y. Bengio, "On the Difficulty of Training Recurrent Neural Networks," in *International Conference on Machine Learning*, 2013, pp. 1310–1318.

[10] K. Cho *et al.*, "Learning Phrase Representations Using RNN Encoder-Decoder for statistical machine translation," *ArXiv Prepr. ArXiv14061078*, 2014.

[11] F. Gers, N. Schraudolph, and J. Schmidhuber, "Learning Precise Timing with LSTM Recurrent Networks," *J. Mach. Learn. Res.*, vol. 3, pp. 115–143, Jan. 2002.

[12] "A Convergence Analysis of Log-Linear Training - Semantic Scholar." [Online]. Available: /paper/A-Convergence-Analysis-of-Log-Linear-Training-Wiesler-Ney/24e4eed7a161769705dfcfa5b659c3dd428de6bf. [Accessed: 02-Jul-2018].

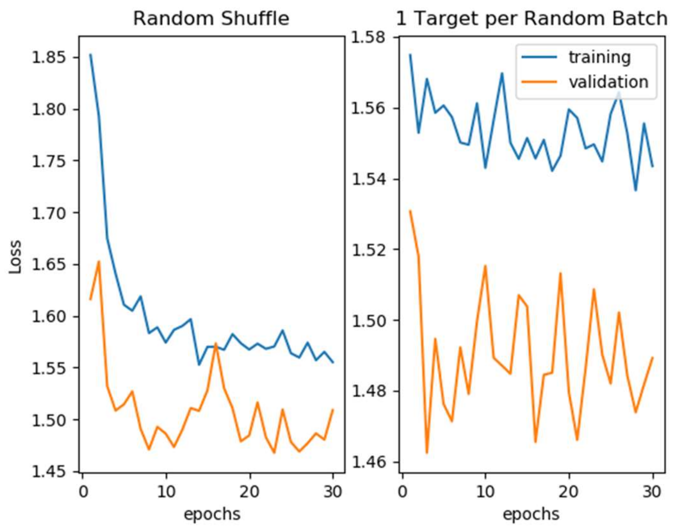
[13] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *ArXiv Prepr. ArXiv150203167*, 2015.

[14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Qay to Prevent Neural Networks from Overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

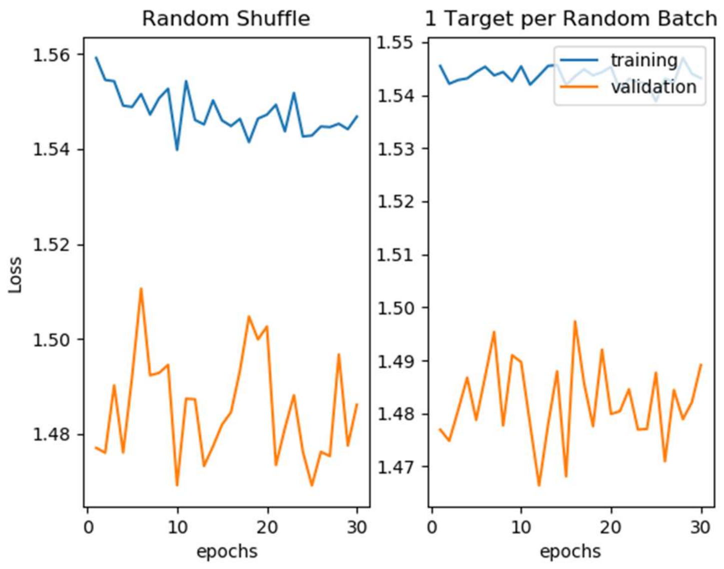
[15] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the Importance of Initialization and Momentum in Deep Learning," in *International conference on machine learning*, 2013, pp. 1139–1147.

APPENDIX A: RANDOM BATCH TRAINING VS SINGLE-TARGET TRAINING LOSS ACROSS 3 MODELS

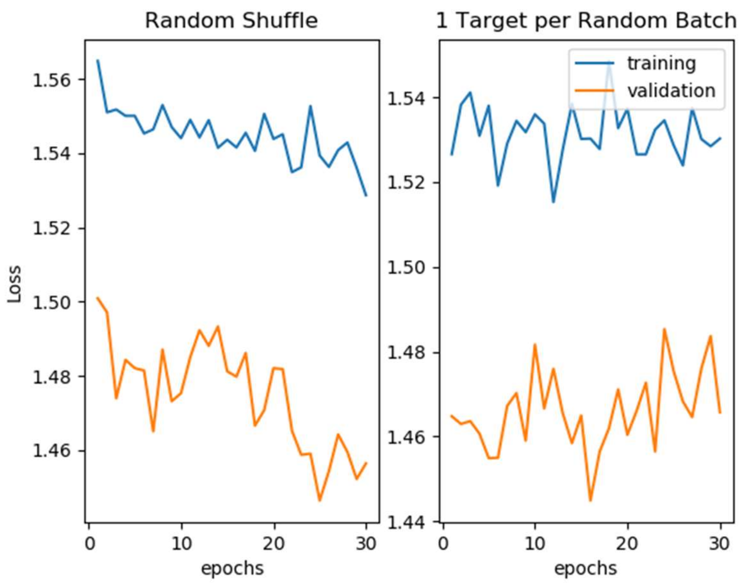
RNN



GRU



LSTM


Random Shuffle
1 Target per Random Batch
Loss
epochs
epochs
training
validation
1.56
1.54
1.52
1.50
1.48
1.46
1.54
1.52
1.50
1.48
1.46
1.44
0
10
20
30