MACHINE LEARNING BASED REAL-TIME EARTHQUAKE SIGNAL PREDICTION

by

Sara Tellab

A Thesis presented to the Faculty of the

American University of Sharjah

College of Engineering

In Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in

Mechatronics Engineering

Sharjah, United Arab Emirates

November 2020

Declaration of Authorship

I declare that this thesis is my own work and, to the best of my knowledge and belief, it does not contain material published or written by a third party, except where permission has been obtained and/or appropriately cited through full and accurate referencing.

Signature: Sara Tellab

Date: November 29th, 2020

Approval Signatures

We, the undersigned, approve the Master’s Thesis of

Thesis Title:

Date of Defense:

Name, Title and Affiliation Signature

Dr. Mohamed El-Tarhuni

Vice Provost for Graduate Studies

Office of Graduate Studies

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

Sara Tellab

Machine Learning Based Real-Time Earthquake Signal Prediction

26-Nov-2020

Dr. Usman Tariq

Assistant Professor, Department of Electrical Engineering

Thesis Advisor

Dr. Mohammad AlHamaydeh

Professor, Department of Civil Engineering

Thesis Co-Advisor

Dr. Lotfi Romdhane

Professor of Mechanical Engineering

Thesis Committee Member

Dr. Zahid Khan

Associate Professor, Department of Civil Engineering

Thesis Committee Member

Dr. Mohammad Jaradat

Program Coordinator

Mechatronics Engineering Graduate Program

Acknowledgements

My deepest appreciation goes to my family for their encouragement and guidance through it all. They are the main reason for every single milestone in my life. I would like to thank my advisors, Dr. Usman and Dr. Mohammad, for their guidance and motivation. I am grateful for the American University of Sharjah for fully sponsoring my degree. Working as a graduate teaching assistant was a great learning experience for me. In addition, thanks are due to all the professors in the Mechatronics Engineering program who taught me graduate courses. Finally, my appreciation extends to all my friends who inspire me beyond words.

Dedication

To my family and friends …

Abstract

Processing the ground motion signal at an early stage is beneficial for issuing warnings, applying corrective measures and deploying firstresponders teams, etc. As an earthquake starts, our proposed machine learning systems take in the first arriving points of a ground acceleration signal and predict the upcoming points in all three axes. The training, validation and testing data is acquired from the Pacific Earthquake Engineering Research Center (PEER) NGAWest2 database. It includes shallow crustal earthquakes with hypocenters less than 20 km deep. The research methodology applies different aspects of supervised and unsupervised learning. We analyze the metadata of previous earthquake records such as the magnitude, horizontal distance and peak ground acceleration (PGA). Moreover, we train various structures of artificial neural networks (ANNs) such as convolutional neural networks (CNN), recurrent neural networks (RNN), long shortterm memory (LSTM) networks and CNNLSTMs. The ANN model serves as a baseline for performance evaluation of the other models. We rely on the sliding window approach to split the acceleration signal. It was found that the best model for short term prediction was the LSTM model for a prediction horizon of ten timesteps. It yielded a root mean squared error (RMSE) of 8.43e6 g which is a 95.2% improvement in performance compared to the baseline that yielded an RMSE of 1.74e4 g. In addition, the prediction time for the CNN model is 0.49 ms, which makes it the fastest model. Moreover, the CNN, ANN and CNNLSTM models experimented with in this work, yielded realtime performance. The other models can also produce faster predictions

using more GPUs or a supercomputer.

Keywords: Machine Learning; Ground Motion Prediction; NGAWest2

Database.

Table of Contents

9

List of Figures

List of Tables

Chapter 1. Introduction

About 20,000 earthquakes that are significant enough to be felt without measurement instruments, happen annually around the globe [1]. About 100 of these are of enough magnitude to cause major damage, if they occur near residential areas. Very great earthquakes happen about once a year on average which can trigger landslides, fires and tsunamis [2]. Over the centuries, they caused the loss of millions of lives and inestimable destruction to infrastructures.

At this point in time, it is statistically impossible to predict an earthquake ahead of time. However, detecting ground motion as it begins to happen and alert the public seconds prior to the major shake is a feasible and propitious alternative. Such systems, known as early earthquake warning or EEW, can potentially reduce the casualties caused by an earthquake.

Various EEW systems are currently being tested and deployed around the world. For example, the ShakeAlert system that is developed by The U.S. Geological Survey (USGS) in the US west coast and the Pacific Northwest [3] [4]. It uses a huge network of sensors that sends data to the processing center which sends alert messages to users. This is a significant infrastructure to maintain and operate. It takes about $16 million per year for the ShakeAlert system [4].

Though operational, the system is still under development. For instance, during the 7.1 moment magnitude (Mw) earthquake that struck near Ridgecrest, CA, USA, in

July 2019, the primary algorithm was able to identify the earthquake and send alerts to users. However, it underestimated the earthquake magnitude by 0.8 units [5]. In addition, the finitefault
algorithm failed to correctly characterize the earthquake.
A fully operational EEW system would always need human action in seconds.
That is barely enough time to move away from windows or get protected under a table.
Here comes a pressing question, what if one is asleep or can not move? What about structures such as buildings and bridges?
The motivation for this research is to eliminate human intervention from the process. Using machine learning, this work can be applied to forecast a continuously updated section of the ground acceleration signal beginning from the first few points.
16
As more points become available, they are continuously fed to the system to make new predictions. We refer to the duration of future prediction as the ”forecast horizon”. Our algorithms can be integrated into a mechatronics system installed in structures to help reduce or even eliminate the structure’s movement. The system can consist of sensors, our proposed models and a magnetorheological damper. It contains magnetorheological fluid, which is controlled by a magnetic field, usually from an electromagnet [6].
Varying the strength of the electromagnet can continuously control the damper’s characteristics.

1.1. Thesis Objectives

This study intends to explore the use of several ML techniques for ground motion prediction. The system takes in a window of the acceleration signal, measured by accelerographs, and it forecasts the future signal as the earthquake emerges. The predictions are made in all three axes simultaneously. In addition, the predictions are of a sampling interval of 0.0014 seconds. We follow the sliding window approach to process the signal. The contributions of this research work can be summarized as follows:
1. Propose multiple machine learning systems to predict the ground motion acceleration signal in realtime
in three axes.

2. Compare the performance of various neural network architectures for forecasting.

3. Achieve a prediction delay of less than 10% of the forecast horizon.

The algorithms in this thesis are trained and tested on the records from the NGAWest2 database records. It is worth pointing out that the training and testing subsets do not contain samples from the same earthquake. We experiment with multiple machine learning algorithms. This study can serve as a baseline for future works in this field.

1.2. Proposed Solution

We propose various models that can help support structures as an earthquake unfolds. The models predict the ground motion acceleration in three axes. We consider the

17

NGAWest2

database for the metadata and time series signals. The $k$means

clustering

algorithm as well as principal component analysis (PCA) are applied to the metadata to investigate various groups of earthquakes in the database. We then set the input to 357 points, or half a second, and experiment with different prediction/forecast horizons. An artificial neural network (ANN) acts as a baseline for all the other proposed models. We implement various models for acceleration time series prediction, namely, convolutional neural network (CNN), recurrent neural network (RNN), long shortterm

memory

(LSTM) network and CNNLSTM

network.

1.3. Thesis Organization

For the remainder of the thesis, the chapters are organized as follows. Chapter 2 provides a literature review on important topics related to ground motion and the applications of machine learning in time series forecasting. It also contains the theory of machine learning relevant to this thesis. Chapter 3 discusses the proposed solutions. Chapter 4 contains important results from the trained networks and a discussion of their implications. Finally, Chapter 5 concludes the research and outlines the future work.

Chapter 2. Background and Literature Review

In this chapter, we discuss the ground motion parameters and the fundamentals of ML. Then, we present the prediction techniques for time series and the applications of ML in seismology in the related work. In addition, we present the background knowledge relevant to this work.

2.1. Ground Motion Parameters

When tectonic plates move, they cause pressure to accumulate due to friction and the energy stores. When the pressure surpasses the rocks' strength, they start to move with respect to each other, resulting in a fault [7]. Moreover, the energy gets released and an earthquake occurs. In the event of tectonic movements, sudden bursts of energy are released as waves. The generated seismic waves travel through the crust, and away from the focus or the center of the earthquake, as shown in Figure 2.1. An earthquake is characterized by its magnitude, frequency components and distance measures. The waves result in triaxial ground acceleration which is measured by accelerographs. They record acceleration digitally with respect to time resulting in a time series record also known as an accelerogram.

Figure 2.1. Earthquake source and generated waves (image source: [8]).

Ground motion is divided into vertical and horizontal components, the latter being larger in amplitude. However, this can change, especially near larger earthquakes. One of the main parameters to characterize the motion and its resulting damage is the peak ground acceleration (PGA) [9]. PGA is defined as the maximum amplitude of the ground acceleration in a given location in units of gravity g or $cm/s2$. Another significant parameter is the duration, which is known as the total time of the strong motion. Often, responses are computed to assess structural damage for multiple natural frequencies. These observations are vital to evaluate the seismic hazard of an area.

2.2. Machine Learning

The crux of machine learning (ML) is to learn from data. For example, to

learn to predict signal types or values based upon inputs [10]. The ML algorithms are trained with specific inputs depending on the problem. They can also be updated as new data is available. The learning process can also be modeled probabilistically. ML can be divided into two main groups: supervised learning and unsupervised learning, as shown in Figure 2.2. The difference between them is the presence of the labels. Supervised learning, which includes predictive models for the labeled datasets, can be further subcategorized into classification and regression algorithms. In classification problems, the algorithm's goal is to identify the input as part of a group or class and predict the class as a discrete value.

The classification can be divided to binary classification and multiclass classification

based on the number of classes. The performance is then evaluated based on the accuracy of the predicted class using binary crossentropy

for the binary classification or

the sigmoid loss for the multiclass

classification. In regression, the algorithm maps the

input to a continuous output variable. Unsupervised learning is helpful for exploratory data analysis because it identifies structures in the data automatically. Examples of unsupervised learning include dimensionality reduction and clustering, depending on whether the goal is decreasing the inputs' dimensions or grouping similar data together. In our study, both labeled and unlabeled data are available. The labeled data is the ground acceleration signals and the unlabeled data is the earthquake metadata.

Figure 2.2. Types of machine learning algorithms (image source: [10]).

2.3. Related Work

In this section, we present existing literature in time series prediction. In addition, the methods of predicting ground motion parameters are shown. Finally, we discuss the applications of ML in seismology.

2.3.1. Time series prediction. A time series is defined as a set of numerical observations

taken in consecutive order at equally spaced intervals. Mathematically, time series x is defined as a set of vectors $x(t) = \{x1, x2, ..., xT\}$ [11] [12]. The time variable is the independent one and the target variable is the dependent one. If there is one value per time step in the series, it is termed as a univariate time series. On the other hand, if multiple points occur per time step, it is called a multivariate time series.

Analysis of time series data is crucial for identifying existing trends and patterns. Important decisions can be made from the analysis, for example, like whether one should invest in the stock market based on the trend. Also, historic data of a time series are

used to make predictions into the future. This is called time series forecasting, and it is a thriving research area in many domains such as financial, medical and environmental domains. Selecting the suitable model is crucial as it reflects the underlying form of the time series. The model used to make a prediction is classified as linear or nonlinear based on the manner in which the past data is combined.

Babušiak and Mohylová proposed ANNbased models to predict the next electrocardiogram (ECG) sample, given the five previous ones [13]. Two ANN architectures were tested, namely, the onelayer model and the multilayer back propagation model.

It is noticed that the second network is more accurate. However, it takes more epochs to converge. In 2017, a study was conducted to forecast the upcoming time period in the electroencephalogram (EEG) signal [14]. The input is selected and altered in the training stage using neighborhood structures (NS). The maximum achieved accuracy was around 70% when using 30 models per person. When predicting the maximum forecasting horizon, the accuracy did not change significantly.

The uncertainty in highly nonlinear data imposes a challenge on the modeling process. For example, stock market is a significant area where different prediction

methods are applied to determine the upcoming trend. In [15], CNN, LSTM network and RNN were utilized for stock price predication using a sliding window approach. The paper concluded that CNN generated the most accurate results, since it is not history dependent and only analyzes the window at hand, unlike RNNs, where previous sequences are used to make a prediction.

Recurrent neural networks are the standard when it comes to time series forecasting. In [16], RNN was used to predict temperatures inside building. The input is a combination of parameters such as the outside wind and temperature, which makes the input a multivariate sequence while the output is univariate. The sequence length, which is the number of observations taken into account, was varied to determine the optimal one. It was found that the best prediction accuracy was associated with a sequence length of 120. However, a window of length 12 produced a similar accuracy.

long shortterm memory network is the more powerful variant of RNN. In [17], an LSTM model was used for oneday ahead solar irradiance prediction. The input was composed of 11 timesteps with nine features which makes it a multivariate sequence.

22

The proposed model contained a single LSTM layer followed by a dense layer representing the output. It was found that the LSTM network outperformed the ANN by 18% for an output to input ratio of 9%.

Convolution neural networks gained popularity in recent years as the state of the art technique in computer vision [18], natural language processing [19], and also in time series forecasting. In [20], a 1DCNN model was implemented for one step ahead river flowerstream prediction. Three models were designed with different time intervals, daily, weekly and monthly. The best performance was associated with an

input size of four and an output to input ratio of 25%. The monthly interval had the worst performance as the model could not reflect the dynamic variations. Therefore, sampling data at a suitable time is crucial for the model to perform well.

Combining the pattern extraction capability of CNN and the memory of LSTM network yields a powerful network that can capture the dynamic nature of a time series. In [21], the goal was to predict the daily gold prices with CNNLSTM model. Previous gold prices with today's prices are used to predict the following day's prices. It was found that a model with two CNN layers and one LSTM layer followed by fully connected (FC) layer gave the best performance. In addition, the optimal forecast horizon was six data points.

A similar network was utilized to predict the concentration of air pollutants [22]. The input is a combination of meteorological factors and past pollutant concentration as a multivariate sequence. The forecast horizon length is 24 hours where the input is 72 hours long, so the output to input ratio is 33.3%. The baseline model was a shallow ANN to compare results. The proposed CNNLSTM model RMSE was 36% less than the ANN model and 20.3% less than the conventional LSTM model.

2.3.2. Ground motion estimation. The classical method to estimate some parameters of an earthquake is using ground motion prediction equations (GMPEs). The basic elements of ground motion are the earthquake's source, route and area conditions [23]. Therefore, the classical model relies on these factors to define the ground motion in the form of a simplified linear regression. Newly introduced GMPEs include more regression coefficients to enhance the accuracy, as shown in [24], while increasing the

complexity. A complete and accurate model definition requires the fault characteristics and the development of the rapture, which are primarily unknown and difficult to obtain. In addition, the parameters used in each model should be evaluated and tested properly as they are regiondependent.

Therefore, in regions with a dense network of seismographs, the parameters are well defined [25]. However, that is not the case in most seismically active regions because of lack of measurements.

2.3.3. Applications of ML in seismology. Due to the major increase of available seismic data, ML was integrated heavily in a variety of applications from forecasting to feature extraction. An interesting research was the prediction of a laboratory fault failure [26]. The algorithm listens to the acoustic signal generated by the fault and extracts a signal that was discarded as noise before. Thus, the time remaining before an artificial earthquake was determined.

A study in 2009 applied ML to predict the seismic response of a twofloor building based on selected structural parameters [27]; feed forward back propagation (FFBP) with one hidden layer was trained using real acceleration signals and the computed responses. The results showed good accuracy for both stories. However, the method requires complex computations to extract the desired features. Kerh and Ting employed multilayer feed forward (MLFF) neural network for PGA estimation [28]. Three ANNs, each with one hidden layer were developed, where the input was a combination of epicentral distance, focal depth, and magnitude. The output was PGA in one axis from 21 testing cases. About 85.7% of the testing cases yielded a correlation coefficient $R2$ less than 0.5, which is considered as a low level of correlation.

Another application of MLFF neural network was done by Arjun and Kumar for duration estimation [29]. The proposed models were designed to forecast the duration of strong ground motion from the magnitude, hypocentral distance, shear wave velocity, and the average of the soil characteristics. ANN with six inputs showed 55% accurate results. On the other hand, when using the first three inputs, the accuracy increased to 61%. Another study utilized the same inputs to predict the PGA in three directions based on Turkish records [30]. Three ANN architectures were implemented, namely, radial basis function (RBF), FFBP and generalized regression neural networks (GRNNs). Af24 terwards, the direction which contained the maximum PGA was fed to the network to

determine the maximum PGA value. FFBG with one hidden layer showed better performance in all the three axes. On the other hand, the RMSE of RBF reached 58.17 $cm/s2$, which is considerably high.

In 2013, an ANN model was developed to predict the PGA in one direction from the magnitude, hypocentral distance, and focal depth [31]. FFBP algorithm was utilized with one and two hidden layers. The optimum results were observed when the number of neurons was between 3 and 20 with one hidden layer. For some PGA values, the mean square error reached 1.1 $cm/s2$. In 2017, a similar study predicted the ground motion parameters such as PGA and the first 26 points of spectral acceleration from 13552 shallow earthquakes [32]. The ANN is composed of one hidden layer, five neurons and five input nodes, namely, magnitude, focal mechanism, shear velocity, distance to rapture and its logarithmic value. The focal mechanism is assigned a value from 1 to 3 based on the formation of the fault. The results demonstrate high accuracy because it was optimized with genetic algorithm.

EEW plays a major role in saving lives and structures. On the arrival of the P waves, an alert is issued before the strong onset occurs. The following studies integrate realtime

ML to the EEW to make it faster and reveal more details about the potential earthquake. Leach and Dowla used ANN with FFBP architecture to obtain the shaking intensity, duration and time remaining until PGA from Southern California records [33]. It was shown that the first few seconds of seismic activity play a vital role in the estimation process. Also, the results show a good performance with an $R2$ value of 0.843, given that strong motion was not included in the study. In addition, a realtime

EEW

was implemented based on the seismic activity from all three directions. The system classifies the earthquake as hazardous if the magnitude scale exceeds 0.58.

Another study conducted in the same region explored the use of probabilistic ANN for magnitude prediction based on eight seismicity indicators [34]. The model classifies the magnitude into seven ranges from less than 4.5 to 7.5 on the Richter scale.

The best prediction was for the range 4.5 – 6 with an $R2$ value of 0.78. However, large scale earthquakes yielded an $R2$ value less than 0.5. The authors explained this classification error with the scarcity of big earthquakes in the dataset.

25

Ramirez and Francois employed feature extraction and supervised ML to classify the incoming seismic waves from three directions [35]. The output was either Por Lphases indicating compressional and surface waves respectively. The maximum correct classification accuracy was 67.9% associated with 0.3 acceptance threshold. In 2018, various ML techniques were applied to reduce false EEW [36]. The system learns to differentiate between Pwaves and noise with a high accuracy. Feature extraction was performed on the input signal followed by generative adversarial networks (GANs) and random forests algorithm. The results were promising; the accuracy reached 98.4% for noise signals and 99.2% for P waves.

In the same year, a similar study was conducted to reduce false warnings using classification [24]. The incoming waveforms from multiple channels were classified into phases based on the likelihood function. The results show 50% less error compared to the classical classification function. In [37], the study is based on feed forward ANNs with two hidden layers to estimate the hypocenter and the magnitude. The system relies on the emerging seismic signals from multiple sensors in the Marmara region to issue an EEW. The best accuracy was obtained using four stations and a window of 3.5 seconds. However, enhancing the accuracy here is at the cost of waiting longer which is inefficient for EEW.

A more reliable solution was provided by Kuyuk and Susumu through classifying the earthquake into nearsource or farsource based on the incoming Pwaves [35].

Long Shortterm

Memory Networks were used as the classification function. The input is one second long which is divided into 13 points and the hidden layer contained 100 neurons followed by a classification layer. The training accuracy was more than 95% for both classes. However, during the testing phase, the accuracy dropped to 65.7%. Based on the presented research, all the earthquake predictions are done for some parameters relating to the earthquake and not the acceleration time series itself. It is evident that this study is exploring a new area that is the realtime

prediction of ground

motion resulting from an earthquake. This study begins with using ML models to predict a window of the earthquake signal from the initial points. The used ML algorithms are ANN, CNN, RNN, LSTM network and CNNLSTM

network. The predictions are

updated for future windows of the signal as new points are measured. Moreover, we

conduct some analysis on the metadata like the PCA and $k-$Means clustering. We identify clusters in the metadata and then apply ML models based upon all the clusters which makes the model more robust.

It is worth noting that it is important to have a sufficiently long input window to obtain useful information from it but not overly long as vanishing gradient may occur in long sequences. Therefore, an expanding window is not suitable for our research as the window can become really long after some time (more than 100,000 points). The sliding window approach is utilized with a window size of 357 points or half a second. The timelag

or shift is equivalent to the forecast horizon to obtain nonoverlapping

outputs. In addition, the design should use less layers initially, and if they do not deliver satisfactory performance, we can move to deeper networks as most forecasting problems can be addressed with a small number of layers.

2.4. Theoretical Background

In this section, we shed light on the theory related to our thesis. Namely, Principal component analysis and KMeans clustering. In addition, variants of artificial neural networks such as recurrent and convolutional neural networks are discussed.

2.4.1. Principal component analysis. Principal component analysis (PCA) is a technique used to reduce dimensionality in an unsupervised manner. The number of variables is reduced to a set that contains fewer ones with most information about the original variables. This operation comes at the expense of some accuracy for simplicity. PCA is performed on a vector x ∈ Rm by projecting to a lowerdimensional vector

z = PTx, z ∈ Rn [38]. The columns of P are the principal components. The principal components indicate the direction with maximum variability. The principal components are orthogonal and are the eigenvectors of the covariance matrix S of the data, while the eigenvalues λ of each feature represent the variances in those directions. These are given by

S =

1

n − 1

Õn

i=1

(Xi − ¯X) (Yi − ¯ Y),

Sv = λv,

(1)

27

where S is the covariance matrix between variable X and Y. Since S is a square matrix, v is a vector and λ is a scalar that satisfies equation 1, then λ is called eigenvalue associated with eigenvector v of S. Clustering performance generally decreases with higher dimensions. Therefore, we chose to implement dimensionality reduction through PCA before applying clustering on metadata. PCA also helps us visualize the data in lower dimensions.

Moreover, it eradicates correlated features as the resulting principal components are independent of one another. However, they might be less interpretable compared to the original features as they are linear combination of all features. Figure 2.3 shows two principal components after applying PCA.

Figure 2.3. An example of the two principal components with maximum variance after dimensionality reduction through PCA (image source: [39]).

2.4.2. KMeans

clustering. Clustering is one of the most popular analytical methods to find structure in data. We try to identify homogeneous subgroups such that the observations in each subgroup are as related as possible according to a similarity measure, such as the Euclidean distance. The clustering used in this work is based on features, where the clustering is conducted on observations from each feature. Clustering is a type of unsupervised learning because the ground truths are unavailable to compare with for performance evaluation. The goal is to investigate the data structure by grouping observations into different groups.

Kmeans clustering is an iterative algorithm that partitions the data into K foreknown number of clusters, such that each observation belongs to only one cluster. It

tries to keep the clusters as different as possible while keeping the points as similar as possible based on the similarity measure. The cluster centroid is defined as the arithmetic mean of all the points in that specific cluster. The algorithm will assign points to a cluster if the sum of the squared distance between the points and the centroid is minimum.

The clusters' centroids are initialized randomly and subsequently optimized using the mean of the cluster. Firstly, each point is designated to the nearest cluster based on the Euclidean distance. Afterwards, the mean of all the points in a cluster is calculated and the centroid is moved to the mean. The process continues until the position of the centroids stabilizes. The solution approach can be formulated as ExpectationMaximization.

The expectation is assigning each point to a cluster and maximization is

the computation of the centroid for each cluster. The objective function can be expressed

as

J =

Õk

j=1

Õn

i=1

wji

$x\ j$

i

− $c\ j$

2

, (2)

where wji = 1 if the point xi belongs to the cluster and 0 if not. Here, n, k are the

number of points and clusters, $c\ j$ is the centroid of cluster j , and xi

j is point i in cluster

j . Kmeans

requires prior knowledge about the number of clusters.

1. Relatively efficient: Algorithm complexity is of order $O(tknd)$, where n, k, d,

and t are the number of data points, clusters, dimension and iterations respectively.

Normally, k, t, d << n.

2. Optimum results occur when the data points are wellseparated

from each other.

2.4.3. Artificial neural networks. Artificial neural networks are inspired by the human brain, which is essentially a network of interconnected cells or neurons. ANNs can learn from the data without being explicitly programmed to do so, which makes them data driven models [40]. The learning model relies on weighting the input to each node or neuron. Inputs that contribute more to the output are assigned a higher weight. The data is fed to the input layer which passes it to consecutive layers also known as the hidden layers. If the network contains more than one hidden layer it is referred

to as a deep ANN. Neurons are the base unit that makes up all layers. Each neuron receives the inputs and performs some mathematical operations. The connections to neurons carry a weight each that are constantly updated during the training process. The output of a neuron can be expressed as

zl

j = (Wl

j

)Tx + bl

j , (3)

where Wj are matrices of weights that maps the inputs to the next layer. The $b\ j$ terms represents the bias which is added to all neurons expect the ones in the input layer. To model complex problems and learn more from the data, a nonlinearity is introduced, also known as activation function. The final output from a neuron after it goes through the activation function can be expressed as the output of neuron j in layer l is expressed as,

al

j = φ(

Õ

k

wl

$j\ kal$−1

k

+ bl

j

), (4)

where the sum is over all neurons denoted by k in the previous layer. The set of operations performed by a neuron are shown in Figure 2.4. After one pass of all the training examples through the network and producing an output, forward propagation is completed. The prediction is then compared to the actual output and the error will impact the entire network through back propagation (BP). The goal of the back propagation is to change the weights and bias to minimize the cost function. The amount of change in the weights and bias is determined by the gradient of the cost function with respect to the parameters using the chain rule.

The optimizer is the algorithm responsible for changing the network characteristics to minimize the loss. Gradient descent is a type of optimizer that uses the first derivative of the loss function which makes it simple to implement. However, it is susceptible to be stuck in a local minima [41]. Adding a momentum term can solve this issue, as it continuously increases the size of the step taken towards the minimum. The amount of change in the parameters at each epoch is determined by the learning rate. If the learning rate is small, the convergence process speed will be hindered. On the other hand, a large learning rate can cause divergence in the error. The number of parameters

that the optimizer has to improve are the sum of all the weights and bias in the network.

Figure 2.4. The perceptron, showing an example of the operations described in equation (4) (image source: [42]).

Model selection and the resulting parameters are detrimental to the model performance. When the model is too simple to describe the problem at hand, underfitting occurs. Essentially, the model fails to learn from the training data and will not perform well on the test data. Underfitting can be solved by increasing the complexity of the

model by adding more layers or hidden neurons. When the model is overly complex and has a lot of parameters, overfitting can occur if insufficient data is provided. Overfitted models don't generalize well as they strictly memorize the training data only. Hence, supplying more training data to the model is an appropriate solution to overfitting. Regularization or early stopping can be added to prevent overfitting. Early stopping prevents overfitting by monitoring the validation error, as shown in Figure 2.5. If the loss keeps increasing for a specific number of epochs, also known as patience, it halts the training and restores the weights of the best epoch. Regularization imposes a penalty on the model for the weights, the optimization algorithm has to deal with constraints in addition to minimizing the loss between the actual and predicted output. Regularization

restricts the flexibility of the network, two examples of regularization are $L1$ and $L2$ regularization.

Figure 2.5. Early stopping halts the training as the validation begins to increase, as indicted by the arrow.

$L1$ regularization also known as Lasso regression, adds the absolute value of the weights to the error function as shown in equation 5. If the value of the tunable regularization parameter α is zero, the model will become unregularized. $L1$ regularization works to reduce the number of parameters in the model by forcing some weight to become zero. This is effective in feature selection. $L1$ regularization can be expressed as

$Loss = Error(\hat{y}, y) + \alpha$

$\tilde{O}N$

i=1

$|wi|$, (5)

where N is the total number of training examples, and yi , $\hat{}yi$ are the groundtruth and

predicted points, respectively. $L2$ regularization or ridge regression is the most common type of regularization. $L2$ regularization adds a penalty to the square value of the

weights. Therefore, if the value α is too large, it imposes a high penalty as shown in equation 6. In this technique, no parameters are eliminated but they are all reduced by the same factor.

$Loss = Error(\hat{y}, y) + \alpha$

$\tilde{O}N$

i=1

$w2$

i (6)

2.4.4. Convolutional neural networks. The main issue with utilizing classic artificial neural networks for regression, specifically time series problems, is the huge number of parameters to be optimized. For example, a sequence of length l would require l parameters to connect to a single node in the next layer. Convolutions neural networks are known to have a manageable number of parameters and that makes the training time manageable [43]. The main computation in a CNN is the convolution operation. For a 2D input I of size (H,W), the convolution operation of I with a filter K yields the output feature map O

Okl =

$\tilde{O}H$

i=−H

$\tilde{O}W$

j=−W

$Ii, jKk-i,l-j$. (7)

This can be thought of as the filter K moving across the height and width of the image and taking the sum of the elementwise products of overlapping entries.

Adding a stride of size s to a convolution results in the filter moving s steps for every computation in equation (7), instead of column by column. This decreases the size of the output, and can be used to control for the size of the output feature map, as

well as in pooling layers. Padding is another common practice which consists of adding rows and columns of zeros to the boundaries of the input image to also allow for control of the output dimension.

Similar to ANNs, deep CNN architectures contain many hidden layers with different properties. One of the most common layers are convolutional layers, which convolves an input feature map with a filter as described. Analogous to layer weights in ANNs, filter parameters determine how the output looks like, and is one of the fundamental drivers of CNNbased learning. Max pooling layers are also common. Max

pooling of size M downsamples the input by taking the maximum element from $M \times M$ windows across the input, and do not contain any learned parameters. Fullyconnected layers mimic the hidden layers of ANNs, and are usually placed before the last layer. In addition to these three, many other layers can be found in CNN architectures.

To process 1D signal inputs in CNNs, 1DCNNs have been developed. One of the differences between 1D and 2D CNNs is that the feature maps and filters are onedimensional. The kernel of a filter with size h will cover size l of and slide to the right one time step at a time.

The 1D convolution operation between a $p$dimensional input x and a filter w gives a 1D output y, given by [44]

$$yi = \tilde{O}_{k=-p}^{p} xi-kwk \; . \quad (8)$$

The convolved sequence is then passed through a nonlinear activation function and downsampled.

An example 1DCNN
architecture illustrating the forward propagation
is shown in Figure 2.6, where each hidden layer has 24 filters performing the 1D
convolutions on their respective input feature maps.
Figure 2.6. Example of a 1DCNN
architecture for binary classification (image
source: [44]).
34
The filter parameters are governed by an optimization algorithm with respect to
a loss function that seeks to minimize some error between the network outputs and the
groundtruth
labels. Optimizers use the gradients of the loss function with respect to the
learnable network parameters to find these minima. Some common loss functions are
MSE, RMSE, and crossentropy
loss.
1DCNNs
are significantly less computationally expensive than 2DCNNs,
due
to the number of learnable parameters [44]. They have been used in several applications
related to fault monitoring in vibrating structures, mechanical parts, and multilevel converters
[44]. Avci et al. [45] trained 1DCNNs
using signals obtained from accelerometers
to monitor the directions most susceptible to damage. Abdeljaber et al. [46] used
signals obtained from damage scenarios in structures to train a 1DCNN
for structural
health monitoring.
2.4.5. Recurrent neural networks. In recurrent neural networks, the current state
of the model is affected by the previous states. Unlike feedforward
networks where the

information travels exclusively in one direction from the input to the output layer [47]. The RNN produces an output and feeds it back to the network to be used in the current computation which helps it retain immediate memory from previous computations. The principle of operation can be summarized as

$yt = f\ (ht\ ;\ \theta)$

$ht = g(ht{-}1,\ xt\ ;\ \theta),$

(9)

where yt which is the output at instant t, depends on the current state ht given θ which includes all the network parameters. Given the same parameters θ, the current state ht depends not only on the current input but on the previous state $ht{-}1$ as well. The second equation illustrates how RNNs can remember past computations and carry it to future ones. Figure 2.7 demonstrates the three main variables in an RNN and how they interact. RNNs has the advantage of smaller number of parameters to optimize compared to ANNs because they deploy parameter sharing. It essentially means that the same weights (θx, θy, θh) are reused across all time steps. Parameter sharing gives RNNs the ability to handle variable length inputs.

Figure 2.7. RNN unfolded, an example of the operations happening inside an RNN layer.

In the case demonstrated in Figure 2.7, both the input and output pairs are univariate as the have one value per time step. For multivariate sequences, the dimensionality can be controlled by varying the number of units in each layer. In Figure 2.8, the number of units in the hidden layer is four which equates to the output dimensionality. The output can either be a at every time step or at the end of the sequence. Training RNNs includes obtaining an output then propagating backwards in a process called backpropagation through time (BBTT) to find the gradients of the loss function and adjusting the parameters afterwards.

Figure 2.8. The number of units inside the RNN layer.

RNNs are susceptible to the vanishing/exploding gradient problem that hinders the training process [48]. It occurs because the backpropagation algorithm utilizes the chain rule (more details can be found in [40]). Since the chain rule involves multiplication of partial derivatives, the gradient in the early layers will contain multiplication analogous to the sequence length. If one partial derivative is < 1, the product will be very small resulting in vanishing gradient. In case of derivatives > 1, the product becomes big. Both of these cases are problematic, because vanishing gradient makes the training very slow and exploding gradient renders the training unstable.

2.4.6. Long shortterm

memory networks. Long shortterm

memory, or LSTM

network, is a type of RNN initially created by Hochreiter and Schmidhuber in 1997 [49]. It is deemed powerful for a variety of applications such as machine translation [50] and speech recognition [51] [52]. LSTMs were created as a solution for the RNN's short term memory, since they have internal mechanisms called gates which regulate the flow of information [53]. These gates can learn which data in sequences should be kept and which should be discarded. It learns to use relevant information to make predictions. The gates are the input, forget and output gates. Starting with the forget gate which is expressed as

$ft = \sigma(Wf\ [ht-1, xt\ ] + b\ f\ ])$. (10)

The input xt at instant t is combined with the previous state $ht-1$ into a single vector. The vector is multiplied by weight Wf , added to a bias $b\ f$, and passed to the sigmoid function $sig(t)$ defined by

$sig(t) =$

1

$1 - e-t$. (11)

The output of the sigmoid function is between 0 and 1. When it is closer to 1, the information will be retained and when it is closer to 0, it will be discarded. To update the cell state or memory ˜ ct , the vector of the input and previous state is passed through

tanh which gives an output between 1

and 1. This range will protect from exploding

gradients. In addition, the vector is passed through the sigmoid function, which will decide whether to add the new information to the memory or not. These two operations

37

are expressed as

$it = \sigma(wi\ [ht-1, xt\ ] + bi])$

$\tilde{}\ ct = tanh(wc\ [ht - 1, xt\ ] + bc)$.

(12)

Afterwards, the candidate state $\tilde{}\ ct$ and the input gate output are multiplied which further regulates the information from the memory. At this point, the cell is ready to calculate the current cell state ct by adding the forget gate output ft to the candidate state $\tilde{}\ ct$ as shown in equation 13. Figure 2.9 illustrates the inner connections of an LSTM cell.

$ct = ft * ct-1 + it * \tilde{}\ ct$ (13)

Figure 2.9. Inner connections of an LSTM cell (image source: [54]).

Finally, the output gate decides which information from the current cell state will be passed to the next cell. The vector of the input and previous hidden state is passed through a sigmoid function, as shown in equation 14, which as explained previously acts a filter. Afterwards, the output is multiplied with the tanh of the current cell state to output a value between 1

and 1 and that is the new output that will be used in the next

time step. The aforementioned operation can be expressed as

$ot = \sigma(wo\ [ht-1, xt\ ] + bo])$

$ht = ot * tanh(ct\ )$.

(14)

38

2.4.7. CNNLSTM

networks. The CNNs are capable of extracting information

from the input as feature maps. LSTM networks can handle sequential dependencies

in a time series. For this reason, CNN and LSTM are often utilized together in the same network to forecast a time series. The convolutional layer extracts feature maps representing different channels of the input simultaneously. Each input channel accounts for a distinct timedependent variable in the multivariate sequence. The feature maps are then used as an input to the LSTM layer to make a prediction.

39

Chapter 3. Methodology

In this chapter, we describe the methodology by which we adapt different machine learning algorithms for ground motion acceleration prediction. The database NGAWest2 is introduced as well as the data preprocessing.

3.1. Database

This research employed data from the Pacific Earthquake Engineering Research Center (PEER), specifically the NGAWest2 database [55]. It contains ground motion data from shallow crustal earthquakes worldwide, and their locations are illustrated in Figure 3.1. Events are categorized as shallow crustal if their hypocenter is within the continental crust. The database includes triaxial records from 599 earthquakes. In addition, 21,540 records of metadata that characterize events and recording stations are also present, as shown in Figure 3.1. The moment magnitude ranges from Mw 3.0 to Mw 7.9, and the closest distance ranges from 0.05 to 1,533 km. Data is significantly less for distances that exceed 400 km, as shown in Figure 3.2. The raw ground motion time series is processed to minimize lowand highfrequency noise using acausal Butterworth filter.

Figure 3.1. Map of included events in the NGAWest 2 database, shown in red (image

source: [55]).

40

Figure 3.2. Magnitude versus distance for shallow crust records. (image source: [56]).

3.2. Data Preprocessing

The metadata, which contains 276 features, is processed according to the following criteria:

1. Remove unnecessary features such as the station name and date
2. Remove features which include more than 50% missing data
3. Create a numeric value to represent some values such as the coseismic surface rupture: 1=Yes; 0=No; Replace 999 with 1 (absence of measurement)
4. Feature standardization to obtain zero mean and unity variance

The final database contains a total of 21540 records and 48 features for the metadata. In addition, a total of 17602 triaxial acceleration records are used. That is because not all

41

of the time series records are available on the database. The used measurement unit for acceleration through this research is in terms of gravity g which equates to 9.81 $m/s2$. Interpolation was done to ensure an equal sampling frequency of 714.3 Hz, where each time step is 0.0014 seconds. This small sampling interval results in a high prediction precision. Moreover, we obtain more training examples from each signal which in turn enhances the prediction performance.

The time series is split into input/output pairs where the input is called a window and the output is the forecast horizon, as shown in Figure 3.3. This is following the sliding window approach which will increase the number of training examples available to our models. The prediction horizons from different windows are nonoverlapping. Therefore, the windows should be shifted to the right by the forecast horizon as time progresses.

Figure 3.3. The sliding window approach.

3.3. Proposed Models and Training Details

The ML architectures used are described in this section, followed by the training details for the setups implemented to test the networks. These details include the different hyperparameters used in the different models such as the choices of the loss function and training/test data split.

42

3.3.1. Metadata analysis. The record file for the NGAWest2 database includes 48 features as mentioned in the previous section. Due to the highdimensional features, making sense of this large quantity of information is an issue. To analyze our data and potentially deduce patterns within would be extremely difficult due to the curse of dimensionality. This means that the data becomes more sparse and moves further away from each other as the dimension increases. To sustain the space representation, we need more data examples as they grow exponentially with the number of dimensions. Another issue is analyzing the data, since in high dimensions, the data might be similar yet it appears further.

To solve this issue, PCA reduces the number of dimensions by linearly combining all the original parameters, as explained in the previous chapter, and extracting the principal components with the highest variance. In this way, we do not have to manually select parameters, as some important ones might get dismissed. We let the data explain itself and statistically produce new features that represent all parameters. From there, the $k$means clustering algorithm is applied to find structure in the data. The selected number of clusters is three. These clusters are present in the training, testing and validation datasets to sustain a global model applicable for all clusters.

3.3.2. ANN architecture. The ANN model serves as a baseline to compare the other models' performance. It consists of an input layer, one hidden layer and an output layer. The input layer contains windows from different axes where H1 is the first

horizontal axis, H2 is the second horizontal axis and V is the vertical axis. The number of nodes in the input represents the time steps from three axes, the window size is set to half a second or 357 points. The number of hidden neurons should be less than the input and output sizes. We set the hidden layer size to be the mean of the input and output size. Each output neuron corresponds to a predicted time step in the three axes, and we experiment with multiple prediction horizons.

Since ANNs are prone to overfitting due to the huge number of parameters in the network, regularization is deployed. The $L2$ regularization with a value of 0.0001 is used. For longer prediction horizons, the number of parameters might exceed the number of training examples so we increase the $L2$ value to 0.1. In addition, early

stopping will halt the training if the validation loss does not improve within five epochs. The optimizer was selected upon experimentation. The network configuration is shown in Figure 3.4.

Figure 3.4. Proposed ANN model.

The learning rate is an important parameter in the working of ANNs. It determines how the weights are updated at each epoch to reach the minimum. If the learning rate is too high, it can cause the error metric to diverge and never reach the minimum. If it is too low, the learning process becomes too slow. To avoid excessive tuning, a learning rate scheduler has been deployed. It is initialized to be a high value that reduces every epoch according to equation 15. Fast learning happens at first and the weights optimization occurs as the training progresses.

lr =

$lr0$

1 + $epoch * Rd$

, (15)

where Rd is the decay rate and is set to 0.5, and the $lr0$ is the initial learning rate that is determined by experimentation. Another important parameter is called the batch size. The batch size is the number of training examples fed at once during training. We selected

the batch size to be 128. A batch normalization layer is added after the input and

44

the hidden layer and it standardizes the batch to zero mean and unity variance. This ensures training stability and prevents the weights from exploding and helps reduce the total number of epochs in training.

3.3.3. CNN architecture. Since CNNs are excellent for extracting meaningful feature maps, they are utilized for time series prediction. The proposed model, as shown in Figure 3.5, has an input layer of three channels each from a different axis. After that, a 1Dconvolutional layer with a filter size of three. To downsample the layer's output, a stride of two is utilized. The stride reduces the number of computations needed as the convolutions layer produces a smaller output size. A stride of two is the most commonly used stride [57]. Since a stride of two is used, the feature map size will be reduced to half. A flatten layer is used to turn all the feature maps to a single vector and then feed it to an FC layer. This allows us to control the output size by changing the number of neurons.

Figure 3.5. Proposed CNN model.

Two FC layers are used to reduce the number of parameters, where the FC proceeding the output layer has about 100 neurons. Each output neuron produces a single time step through the three axes. The number of convolutions layers and their filters are selected by experimentation based on the test loss. The batch size is 128 and the ini45 tial learning rate are determined by experimentation as well. The selected optimization algorithm is adaptive moment estimation (ADAM), which is the most popular one for CNNs [21]. ADAM eliminates vanishing learning rate and helps speed up the convergence. However, it tends to be computationally costly [58]. Early stopping is deployed with a patience of five epochs.

3.3.4. RNN architecture. The RNN network is composed of an input layer, RNN layer and an output layer, as shown in Figure 3.6. The input contains the windows from

each axis as a separate channel. H1 axis is channel one, H2 axis is channel two and V axis is channel three. Not only does this reduce the number of computations but also separates the axes so the RNN can handle the input as a multivariate sequence rather than a univariate one in the ANN. The layer does not return a sequence, rather it returns a vector. That vector is the summary of the input and it is fed to the FC layer.

Figure 3.6. Proposed RNN model.

The FC layer is the output layer and it is divided into three segments each is the forecast horizon in a different axis. The number of RNN layers are determined through

experimentation along the number of units in each RNN layer. Early stopping of five epochs is used to prevent overfitting. A batch size of 128 is used to divide the dataset into multiple segments for training.

3.3.5. LSTM architecture. The LSTM network is composed of an input layer, LSTM layer and an output layer, as shown in Figure 3.7. The input contains the windows from each axis as a separate channel. The layer does not return a sequence, rather it returns a vector. That vector is the summary of the input and it is fed to the FC layer. The FC layer is the output layer and it is divided into three segments each is the forecast horizon in a different axis. The number of LSTM layers are determined through experimentation along with the number of units in each LSTM layer. Early stopping of five epochs is used to prevent overfitting. A batch size of 128 is used to divide the dataset into multiple segments for training.

Figure 3.7. Proposed LSTM model.

3.3.6. CNNLSTM

architecture. Combining the CNN ability to extract information with the ability of the LSTM network to remember information and deal with time

dependency should benefit the network in longterm predictions. Not only will the CNN produce more meaningful input to the LSTM, but it will also reduce the size of the input.

The CNNLSTM
model consists of an input layer, 1Dconvolutional
layer, LSTM layer
and an output layer, as shown in Figure 3.8. Maxpooling
is utilized after the convolutional
layer to decrease the size of the feature map. The input is windows from the
three axes ordered as channels. The 1Dconvolutional
layer has a filter size of three.
The output layer is an FC layer that predicts future values in each axis. Batch size of
128 is used along with early stopping with a patience of five epochs.
Figure 3.8. Proposed CNNLSTM
model.
3.3.7. Training details. The implementation code was written in Python 3.4 on
a computer (Intel(R) Core(TM) i910900X
CPU 3.70GHz, 62 Gbyte RAM) running
ubuntu 18.04 operating system. The machine learning models were implemented using
Keras library, on GeForce GTX 1080 GPU running with CUDA 10.2. The dataset was
split as 70% training and 30% testing, 10% of the training data is used as a validation
set to test performance during training. The examples that appear in the training set do
not appear in the validation set nor the test set of our earthquake data. For the signal
prediction, a training set of one million examples was used. A smaller subset of 50,000
examples was used for parameter selection. All models except CNN–LSTM models

were trained for 25 epochs with a batch size equal to 128. Table 3.1 summarizes the
training details for all the proposed models.
Table 3.1. Summary of the training/validation/testing splits and activations used for
each of the proposed models.
ANN CNN RNN LSTM CNNLSTM
Training Data 1M 1M 1M 1M 1M

Validation Data 100k 100k 100k 100k 100k

Testing Data 420k 420k 420k 420k 420k

Activations Relulinear

Tanhlinear

Tanhlinear

Tanhlinear

Tanhlinear

No. of epochs 25 25 25 25 35

3.4. Performance Evaluation

To evaluate the performance of our networks in predicting ground motion time series, we need to select an appropriate loss function. The first method which is documented in [59], is called cosine similarity. This method works by measuring the similarity between the actual output y and the predicted output $\hat{y}$ vectors. The cosine similarity is expressed as

$sim(y, \hat{y}) =$

$y.\hat{y}$

$\|y\| \|\hat{y}\|$, (16)

where $\|y\|$ is the Euclidean norm of vector $y = (y1, y2, ...yn)$ which is expressed as

$\|y\| =$

$\sqrt{\sum_{i=1}^{n} y2}$

i . The cosine similarity computes the cosine of the angle between the two vectors. A cosine value of zero means the two vectors are orthogonal or dissimilar. As the value approaches one, the angle decreases and the vectors are more similar until they point in the same direction. However, if either vector is zero the dot product is zero which makes the cosine similarity zero. That indicates dissimilarity but that can be deceiving as the vectors might be matching and therefore similar. This makes cosine similarity unsuitable for earthquake time series prediction as a considerable portion of the signal is zero.

Another heavily documented method in literature is called root mean squared error (RMSE). This measure was used by Cheng et al. [60] for evaluating the performance of a multistep prediction model in time series. In addition, Geng et al. [61]

utilized RMSE for seismic energy prediction from time series and added parameters. RMSE is expressed as

$RMSE$ =

vut

1

N

ÕN

i=1

(yi − ˆyi)2, (17)

where N is the total number of training examples, and yi , ˆyi are the groundtruth and predicted points, respectively. This is a commonly used metric that constitutes a measure of the overall deviation of the predicted points from the actual points. Another model evaluation metric is the execution time since our objective is to have a realtime system.

It is crucial to take a substantially lower time to make a prediction compared to the forecast horizon. We aim for a delay of less than 10% of the forecast horizon.

Chapter 4. Experimental Results and Discussion

In this chapter, we present the parameter selection for all the models. In addition, different prediction horizons are shown, the best performing model compared to the baseline is highlighted. Finally, a summary of the results is discussed along with the execution time for each model.

4.1. Results of Metadata Analysis

In this section, we present the results for PCA and the kMeans clustering algorithm.

These unsupervised ML techniques were implemented on the earthquake metadata.

4.1.1. Results for PCA. The PCA algorithm is used to reduce the dimensional space from 48 to three to visualize the data, as illustrated in Figure 4.1. Thus, we show here three dimensions with the largest eigenvalues. Loadings squared are the variance in each variable per component. Loadings are interpreted as the coefficients of the linear combination of the initial variables from which the principal components are constructed. The variables that influence the principal components the most or have the biggest coefficient in each linear combination are shown in Table 4.1. The strongest variable is the magnitude which is the most common parameter to describe an earthquake. The first principal component represents the earthquake characteristics like the magnitude, magnitude type and the fault rapture width. The magnitude type is either moment magnitude, local magnitude, surfacewave magnitude or body wave magnitude [55].

The second principal component relates to the distance and it is a defining factor in the strength of an earthquake at a given location. The epicentral distance is defined as the distance from the earthquake's epicenter to the recording station. The hypocentral distance refers to the distance from the earthquake's focus or hypocenter to the recording station. In addition, JoynerBoore is defined as the distance from the recording station to the surface projection of the rupture surface. The third component describes the ground

motion time series which are the lowest usable frequency, peak ground acceleration and velocity.

Table 4.1. Variables with highest coefficients in each principal component.

PC 1	PC 2	PC 3
Magnitude	Epicentral distance	Lowest usable frequency V
Fault rapture width	Hypocentral distance	PGA

Magnitude type JoynerBoore distance PGV

Figure 4.1. Dimensionality reduction to three dimensions.

4.1.2. Results for KMeans

clustering. Using the Kmeans

clustering algorithm,

the data is clustered to three parts as shown in Figure 4.2 and 4.3 respectively. The clustering is consistent with the visual one as there are three obvious groups. These groups are not specific to certain earthquake parameters, but to all 48 parameters. That is because the clustering was performed on the principal components and they are a linear combination of all the original parameters. However, the clusters help us structure our dataset. All clusters are present in the training, validation and testing to obtain a robust and global model.

52

Figure 4.2. Three clusters in three dimensions from the PCA.

Figure 4.3. Three clusters in three dimensions from the PCA top

view.

53

4.2. Parameters Selection

In this section, we present the experiments conducted to obtain the parameters for each model. The experimental dataset is composed of 50,000 examples.

4.2.1. Results for ANN. In order to finalize the ANN architecture, the first thing to select was the optimizer. A mini dataset of 50,000 examples was used to experiment with the optimizer and it was trained for 25 epochs. Figure 4.4 shows the training loss for the following optimizers: Adagarad, Adamax, Nadam, Adam and RMSprop. It is clear that the Adagrad started from a smaller RMSE value and converged to the final value in less than 10 epochs. As opposed to other optimizers that started from a bigger value and suffered from high fluctuations. The validation loss for the same optimizers as shown in Figure 4.5 paints a similar story, Adagrad displayed a superior performance. The Adagrad validation loss converged in three epochs. Table 4.2 demonstrates the test

loss on the mini dataset and the Adagrad has the least test loss. Therefore, it is the chosen optimization algorithm for the ANN model.

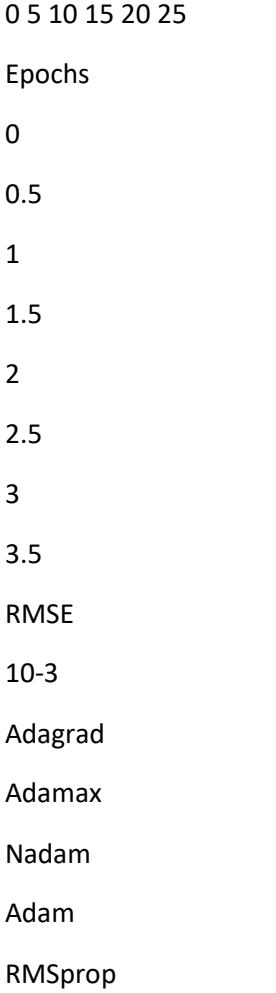


Figure 4.4. The training loss for different optimizers.

0 5 10 15 20 25
Epochs
0
1
2
3
4
5

6

RMSE

10-3

Adagrad

Adamax

Nadam

Adam

RMSprop

Figure 4.5. The validation loss for different optimizers.

Table 4.2. The test loss across different optimizers.

Optimizer Test Loss

Adagrad 2.74e6

Adam 9.30e5

RMSprop 2.61e4

Adamax 1.82e5

Nadam 3.86e5

A learning rate scheduler was utilized that starts with an initial value and decays with a factor of 0.5. Table 4.3 shows the test loss for different learning rates both fixed and decaying. We can see that the decaying performed better than the fixed one. This is because the scheduled learning rate optimizes the weights at the beginning of training and finetunes

them as the training progresses. Moreover, an initial learning rate of 0.01 performed better than 0.001 as it was able to change the weights more and then decay so it would not cause divergence as sometimes would happen with larger learning rates.

55

Therefore, a learning rate scheduler of an initial value of 0.01 was deployed for the training process.

Table 4.3. The test loss for different learning rates.

Learning Rate Test Loss

0.01 6.72e5

0.001 7.41e5

0.01 Decay 2.35e5

0.001 Decay 0.13

4.2.2. Results for CNN. Our proposed model has a 1Dconvolutional

layer with

a filter size of three and a stride of two. The number of convolutional layers was tested on the mini dataset from one layer to three. The best test loss was 3.58e5

g for a single

layer, as shown in Table 4.4 which is less than half the loss associated with three layers. So a single layer was utilized and the number of filters is varied from 3 to 32. Table 4.5 demonstrates that the least number of filters which is three produced the best RMSE, again less than half the loss of 32 filters.

Table 4.4. Investigating the effect of different layers on the test loss.

Layers Test Loss

1 3.58e5

2 5.29e5

3 6.90e5

Table 4.5. One layer with different filters and the corresponding test loss.

Filters Test loss

3 3.58e5

8 3.94e5

16 7.12e5

32 6.93e5

The number of units in the FC layer for our model is set to 100. However, for a forecast horizon of size one, the result was not satisfactory. Therefore, different units in the FC layer proceeding the output layer were tried out. From Table 4.6, we can see that

the optimal number is five. For the learning rate, fixed and decaying values are tested.

From Table 4.7, the best test loss is associated with a value of 0.01 in a learning rate scheduler.

Table 4.6. For a forecast horizon of 1, the effect of the units in the FC layer on the test loss.

FC Units	Test loss
1	1.07e4
5	5.41e5
10	1.87e4
20	3.39e4
50	2.09e4

Table 4.7. The test loss for different learning rates.

Learning Rate	Testing Loss
0.01	3.17e4
0.005	1.87e4
0.01 Decay	3.58e5
0.005 Decay	4.33e5

The batch normalization layer standardizes each batch to have a zero mean and unity variance. We tried inserting this layer in different places in the model before the activation function. It turns out the model that does not use batch normalization gave the best test loss, as shown in Table 4.8.

Batch normalization does not work well in the prediction as it does during the training. This is due to the fact that when we train we use a batch size of 128 and a size of one in the prediction phase as we feed every input to the network as soon as it becomes available. In addition, batch normalization increases the training time due to the additional computation for each batch.

Table 4.8. Batch normalization effect on the test loss.

Batch Norm.	Test loss
None	3.58e5
Only after input	4.49e5

All expect output 1.65e3

57

4.2.3. Results for RNN and LSTM network. Since the RNN and the LSTM network are both recurrent and have a similar architecture, they exhibit similar behavior. The RNN model that was presented in Chapter 3 needed some parameter selection, such as the number of layers and number of units as well as the learning rate and the batch normalization. Table 4.9 shows how changing the number of layers affects the test loss in the mini dataset. A single layer yielded the lowest test RMSE and as we described in Chapter 2, a small number of layers is sometimes sufficient to model the problem. The same result is shown with LSTM network, where a single layer yielded an error of 2.75e5

g. Three units in the RNN layer had the best performance compared to the rest as shown in Table 4.10, as it is equal to the number of input channels and the number of desired output vectors. The LSTM network shows the same behavior where the least loss is associated with three units. Finally, the initial learning rate is set to 0.005 that decays exponentially.

Table 4.9. Investigating the effect of different layers each with three units on the test loss.

Layers RNN Test Loss LSTM Test Loss

1 3.55e5

2.76e5

2 6.41e5

4.75e5

3 6.63e5

7.64e5

Table 4.10. One layer with different units and the corresponding test loss.

Units RNN Test Loss LSTM Test Loss

3 3.55e5

2.76e5

8 8.02e5

5.44e4

16 8.28e5

5.44e4

32 1.40e4

5.99e4

4.2.4. Results for CNNLSTM

network. The CNNLSTM

architecture needed

tuning in terms of the number of layers and the filters in the 1Dconvolutional

layer and

the number of units in the LSTM layer. Table 4.11 shows the effect of different LSTM

and 1Dconvolutional

layers on the test loss. We can see that the minimum loss was

with a single LSTM layer and a single convolutional layer with an RMSE of 1.17e5

58

g. The number of units in the LSTM layer and the number of filters in the convolution

layer were varied together. Table 4.12 shows that the optimal performance was achieved

with six units in the LSTM layer and three filters on the convolutional layer. The RMSE

was equal to 2.51e5

g. However, for a forecast horizon of size one, the result was not

satisfactory, the predicted signal was significantly smaller than the actual one in axis

H2 and V and the test error was 1.1e4

g. Therefore, the number of LSTM units was

increased from six to seven and the issue was rectified. Finally, the initial learning rate

is set to 0.005 that decays exponentially.

Table 4.11. Effect of number of LSTM and convolutional layers on the test loss.

LSTM Layers Test Loss Conv. layers Test Loss

1 1.17e5

1 1.17e5

2 2.35e5

2 2.36e5

3 2.87e5

3 5.07e5

Table 4.12. Number of convolutional filters and LSTM units vs the test loss.

CNN Filters LSTM Units Test loss

3 3 6.66e05

3 6 2.51e5

5 3 3.40e5

5 6 5.61e5

4.3. Acceleration Signal Prediction

This section presents the prediction results for different prediction horizons. In addition, a visual comparison is conducted based on the plotted predictions. Finally, we discuss the overall performance and the execution time for each model.

4.3.1. Results for forecast horizon of size 1. Starting from this subsection, the prediction results are reported based on the test set RMSE which is composed of 450,000 examples. In addition, we show the ANN performance which is our baseline and the best performing model for visualization. The rest of the figures can be found in the appendices. Here we have an input of 357 points or half a second and we try to predict one point or 0.0014 seconds in each axis. From Table 4.13 it is evident that the LSTM

model exhibited the best performance with an RMSE of 1.56e5 g. The RMSE dropped by 98.8% compared to the baseline that has an error of 1.35e3 g. This is expected as the LSTM network is superior in short term predictions because it can retain memory that represents the temporal nature of our data. The model that yielded the secondhighest RMSE was the CNNLSTM

but it still has 94.7% less error compared to the baseline.

Figure 4.6 shows the baseline prediction for one point for a signal with a magnitude of Mw 7.9, which is the maximum magnitude existing in the NGAWest2 database,

from cluster 3. We can see that the baseline prediction was very noisy. When we zoom into an interval of five seconds, indicated by the black box in Figure 4.6, it is clear that the baseline does not keep up with the variations at all. That is because it could not handle the dynamic nature or learn it well. The predictions are much bigger in magnitude compared to the actual signal. On the other hand, Figure 4.7 shows the LSTM network prediction and it is almost identical to the actual acceleration in all three axes. The patterns are modeled accurately and smoothly with minimal variations.

Figure 4.6. ANN prediction vs actual acceleration from cluster 3, window size is 1.

60

0 100 200 300

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 100 200 300

-0.02

-0.01

0

0.01

Acceleration in g

H2

50 51 52 53 54 55

-4

-2

0

2

10-3 H2

0 100 200 300

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-2

0

2

10-3 V

Figure 4.7. LSTM network prediction vs actual acceleration from cluster 3, window size is 1.

Table 4.13. The test loss for all models for a forecast horizon of size 1.

Model Test Loss Improvement %

ANN 1.35e3

CNN

5.41e5

95.9%

RNN 1.96e5

98.5%

LSTM 1.56e5

98.8%

CNNLSTM

7.21e5

94.7%

4.3.2. Results for forecast horizon of size 10. The input window size is 357 points or half a second and we try to predict ten points or 0.014 seconds in each axis. From Table 4.14, it is evident that the LSTM model exhibited the best performance with an RMSE of 8.43e6

g. That is a 95.1% improvement in performance compared to the baseline that yielded an error of 1.74e4

g. This is expected as the LSTM network is superior in short term predictions because it can retain memory that represents the

61

temporal nature of our data. The model that yielded the secondhighest RMSE was the

CNNLSTM

but it still has 71.8% less error compared to the baseline.

Figure 4.8 shows the baseline prediction for 10 points for a signal with a magnitude of Mw 7.9 from cluster 3, and we can see that the baseline prediction was very noisy. When we zoom into an interval of five seconds, it is clear that the baseline models the behavior but it is very noisy. That is due to the fact that it could not handle the dynamic nature or learn it well. On the other hand, Figure 4.9 shows the LSTM network prediction and it is almost identical to the actual acceleration in all three axes.

Table 4.14. The test loss for all models for a forecast horizon of size 10.

Model Test Loss Improvement %

ANN 1.74e4

CNN

3.32e5

80.9%

RNN 3.93e5

77.1%

LSTM 8.43e6

95.1%

CNNLSTM

4.90e5

71.8%

0 100 200 300

-0.02

0

0.02

H1

Predicted

Actual

50 51 52 53 54 55

-5

0

5

10-3 H1

0 100 200 300

-0.02

0

0.02

Acceleration in g

H2

50 51 52 53 54 55

-5

0

5

10-3 H2

0 100 200 300

Time in Seconds

-0.02

0

0.02

V

50 51 52 53 54 55

-5

0

5

10-3 V

Figure 4.8. ANN prediction vs actual acceleration from cluster 3, window size is 10.

0 100 200 300

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 100 200 300

-0.02

-0.01

0

0.01

Acceleration in g

H2

50 51 52 53 54 55

-4

-2

0

2

10-3 H2

0 100 200 300

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-2

0

2

10-3 V

Figure 4.9. LSTM network prediction vs actual acceleration from cluster 3, window size is 10.

4.3.3. Results for forecast horizon of size 50. The input window size is 357 points or half a second and we try to predict 50 points or 0.07 seconds in each axis. The output represents about 14% of the input. From Table 4.15, it is clear that the LSTM model still resulted in the best performance with an RMSE of 3.89e5

g. That is a 90.7%
improvement in performance compared to the baseline that yielded an error of 4.17e4
g.
This is expected as the LSTM network is superior in short term predictions because it can retain memory that represents the temporal nature of our data. The model that yielded the secondhighest
RMSE was the CNN but it still has 85.6% less error compared to the
baseline. We notice that the improvement in performance drops as the forecast horizon grows.
Figure 4.10 shows the baseline prediction for 50 points for a signal with a magnitude of Mw 7.9, which is the maximum magnitude existing in the NGAWest2
database,
from cluster 3. We observed that the baseline prediction was very noisy. When we zoom into an interval of five seconds, the predictions do not keep up with the variations and
63
they are much bigger in magnitude than the actual acceleration. That is due to the fact that it could not handle the dynamic nature or learn it well. On the other hand, Figure 4.11 shows the LSTM network prediction and it is almost identical to the actual acceleration in all three axes and it models the pattern well for the whole earthquake duration
of 300 seconds. However, the maximums and minimums of the signal are not predicted accurately as the magnitude of the prediction is slightly less than the actual earthquake signal.
Table 4.15. The test loss for all models for a forecast horizon of size 50.

Model	Test Loss	Improvement %
ANN	4.17e4	
CNN	6.02e5	85.6%
RNN	4.52e5	

89.2%

LSTM 3.895

90.7%

CNNLSTM

4.65e5

88.8%

0 100 200 300

-0.02

0

0.02

H1

Predicted

Actual

50 51 52 53 54 55

-5

0

5

10-3 H1

0 100 200 300

-0.02

0

0.02

0.04

Acceleration in g

H2

50 51 52 53 54 55

-4

-2

0

2

10-3 H2

0 100 200 300

Time in Seconds

-0.02

0

0.02

V

50 51 52 53 54 55

-2

0

2

10-3 V

Figure 4.10. ANN prediction vs actual acceleration from cluster 3, window size is 50.

64

0 100 200 300

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 100 200 300

-0.02

-0.01

0

0.01

Acceleration in g

H2

50 51 52 53 54 55

-4

-2

0

2

10-3 H2

0 100 200 300

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-2

0

2

10-3 V

Figure 4.11. LSTM network prediction vs actual acceleration from cluster 3, window size is 50.

4.3.4. Results for forecast horizon of size 100. The input window size is 357 points or half a second and we try to predict 100 points or 0.14 seconds in each axis. The output represents about 28% of the input. From Table 4.16, the CNNLSTM model

yielded the best performance with an RMSE of 2.76e5

g. That is a 94.2% improvement

in performance compared to the baseline that exhibited an error of 4.76e4

g. As the

forecast horizon became longer, the LSTM network accuracy dropped as it is optimal

for shorter ranges. The model that yielded the secondhighest

RMSE was the CNN but

it still has 90% less error compared to the baseline.

Figure 4.12 shows the baseline prediction for 100 points for a signal with a

magnitude of Mw 7.9, which is the maximum magnitude existing in the NGAWest2

database, from cluster 3. We can see that the baseline prediction was very noisy. When

we zoom into an interval of five seconds, we notice that the baseline's zero regions be65

came longer and the predictions are bigger in magnitude than the actual acceleration.

On the other hand, Figure 4.13 shows the CNNLSTM

network prediction that models

the acceleration patterns in a somewhat accurate manner. The patterns are modeled with

minimal variations. However, the maximums and minimums of the signal are not predicted

accurately. The predictions experience surges to the maximums and minimums

and decay in an exponential manner.

Table 4.16. The test loss for all models for a forecast horizon of size 100.

Model Test Loss Improvement %

ANN 4.76e4

CNN

4.75e5

90.0%

RNN 4.24e5

91.1%

LSTM 4.75e5

90.0%

CNNLSTM

2.76e5

94.2%

0 100 200 300

-0.02

0

0.02

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 100 200 300

-0.02

0

0.02

Acceleration in g

H2

50 51 52 53 54 55

-4

-2

0

2

10-3 H2

0 100 200 300

Time in Seconds

-0.02

0

0.02

V

50 51 52 53 54 55

-2

0

2

10-3 V

Figure 4.12. ANN prediction vs actual acceleration from cluster 3, window size is 100.

0 100 200 300

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 100 200 300

-0.02

-0.01

0

0.01

Acceleration in g

H2

50 51 52 53 54 55

-4

-2

0

2

10-3 H2

0 100 200 300

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-2

0

2

10-3 V

Figure 4.13. CNNLSTM

prediction vs actual acceleration from cluster 3, window size

is 100.

4.3.5. Results for forecast horizon of size 200. The input window size is 357

points or half a second and we try to predict 200 points or 0.28 seconds in each axis.

The output represents about 56% of the input. From Table 4.17, the CNN model yielded

the best performance with an RMSE of 1.47e3

g. That is an 80.2% improvement in

performance compared to the baseline that exhibited an error of 7.43e3

g. The model

that yielded the secondhighest

RMSE was the CNNLSTM

but it still has 79.4% less

error compared to the baseline. It is worth noting that all the test errors in the experiments are of the same order.

Figure 4.14 shows the baseline prediction for 200 points for a signal with a magnitude of Mw 7.9, which is the maximum magnitude existing in the NGAWest2 database, from cluster 3. We observed that the baseline prediction was very noisy. When we zoom into an interval of five seconds, we notice that the baseline's zero re67 gions became longer and the predictions are much bigger in magnitude than the actual acceleration. That is due to the fact that it could not handle the dynamic nature or learn it well as the prediction horizon became longer. On the other hand, Figure 4.15 shows the CNN predictions which are very attenuated in magnitude. In addition, the maximums and minimums of the signal are not predicted accurately especially in axis V, where the predictions fluctuate around zero.

Table 4.17. The test loss for all models for a forecast horizon of size 200.

Model Test Loss Improvement %

ANN 7.43e3

CNN

1.47e3

80.2%

RNN 1.51e3

79.7%

LSTM 1.52e3

79.5%

CNNLSTM

1.53e3

79.4%

0 100 200 300

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 100 200 300

-0.02

-0.01

0

0.01

Acceleration in g

H2

50 51 52 53 54 55

-4

-2

0

2

10-3 H2

0 100 200 300

Time in Seconds

-0.01

0

0.01

V

50 51 52 53 54 55

-2

0

2

10-3 V

Figure 4.14. ANN prediction vs actual acceleration from cluster 3, window size is 200.

0 100 200 300

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 100 200 300

-0.02

-0.01

0

0.01

Acceleration in g

H2

50 51 52 53 54 55

-4

-2

0

2

10-3 H2

0 100 200 300

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-2

0

2

10-3 V

Figure 4.15. CNN prediction vs actual acceleration from cluster 3, window size is 200.

4.3.6. Overall performance. As we mentioned in Chapter 3, the proposed solution deployed a sliding window algorithm to split the earthquake acceleration into input and output pairs. The outputs from different windows are nonoverlapping which means we need to shift the window by the forecast horizon to make a new prediction. The window size is set to 357 points but we tried different predictions from 1200 points. Table 4.18 shows the performance for all of the models across different forecast horizons. The best RMSE value was achieved with the LSTM model for a prediction horizon of 10. For short term prediction, namely, 1 to 50 points, we notice that the LSTM network yielded the best performance. This is expected as the LSTM network is indeed more suitable for short term prediction. Moving on to an output of 50 where it is 14% of the input, the findings in [17] are similar to this work. The study was conducted to predict solar irradiance. It was found that the LSTM network outperformed the shallow ANN by 18% for an output to input ratio of 9%. Our improvement is 90.7% compared

to the baseline. Our architecture was identical to the one in the paper but we used three LSTM units while in [17], 30 units were used and the training lasted for 100 epochs. Our findings agree that the LSTM network performed best compared to the shallow ANN for similar output to input ratios.

At an output of 100 where it is 28% of the input, we compare our findings to [22]. The study was performed to predict air pollutants concentration in the air. The forecast horizon length is 24 hours where the input is 72 hours long, so the output to input ratio is 33.3%. The proposed CNNLSTM model RMSE was 36% less than the ANN model and 20.3% less than a conventional LSTM model. In our results, the CNNLSTM network is better by 94.2% compared to the ANN model and 41.9% better than the LSTM model. In addition, the RMSE is 34.9% less than the RNN model. Therefore, our findings agree that the CNNLSTM network performed best compared to the ANN model for similar output to input ratios.

As the forecast horizon reached 200 points, all the models generated an RMSE of the same order, and all of them are not that accurate. With a forecast horizon of 200, the ratio became 56% and a higher error occurred as expected. The best performing model was the CNN model. However, all our proposed models performed better than the baseline by at least 70%.

Table 4.18. The testing loss for all the proposed models.

Forecast Horizon	ANN	RNN	LSTM	CNN	CNNLSTM
1	1.35e3				
	1.96e5				
	1.56e5				
	5.41e5				
	7.21e5				

10 1.74e4

3.93e5

8.43e6

3.32e5

4.90e5

50 4.17e4

4.52e5

3.90e5

6.02e5

4.65e5

100 4.76e4

4.24e5

4.75e5

4.75e5

2.76e5

200 7.01e3

1.51e3

1.52e3

1.47e3

1.53e3

Another important metric for performance evaluation is the realtime

aspect.

Figure 4.16, shows the average prediction for each model. The fastest model is the CNN as it requires fewer computations and it takes about 0.49 ms to make a prediction. Table 4.19 shows the delay ratios for all the prediction horizons for all the proposed models. The CNN model is considered realtime

along with ANN and CNNLSTM

network.

The slowest model is the RNN which takes about 41.7 ms to make a prediction. It is not realtime

in prediction horizons of 1 and 10 only. The predictions can be made faster

using more sophisticated hardware that parallelizes the computations more efficiently.

CNN ANN CNN-LSTM LSTM RNN

0

5

10

15

20

25

30

35

40

45

Avg. Prediction Time in ms

Figure 4.16. Average time to make a prediction for each proposed model.

Table 4.19. Prediction delay for all the proposed models.

Prediction Horizon ANN RNN LSTM CNN CNNLSTM

1 45% 2975.7% 222.9% 35% 46.5%

10 4.5% 297.5% 22.3% 3.5% 4.65%

50 0.9% 59.5% 4.46% 0.7% 0.93%

100 0.45% 29.76% 2.23% 0.35% 0.47%

200 0.22% 14.9% 1.12% 0.1% 0.23%

Chapter 5. Concluding Remarks

In this thesis, we explored the possibility of ground motion acceleration prediction

in realtime.

The prediction was achieved in three axes simultaneously. We utilized

PCA to reduce the number of parameters in the metadata and then cluster based on the new principal components. The signals from the three clusters were present in the training and testing datasets. However, the training and testing datasets were disjoint and do not contain data from any common earthquake. In addition, various machine learning algorithms were used to predict the time series such as the CNN, RNN, LSTM network and CNNLSTM
network. The ANN was considered the baseline in this work to compare
all prediction performances. To optimize each model's parameters, a small subset
of 50,000 examples was used.
The used dataset is NGAWest2
from the PEER research center in California.
It contains earthquake records from around the globe. The earthquakes are shallow crustal with a magnitude range between Mw 3.0 and Mw 7.9. For training, one million input/output sequence pairs were used. The input was fixed to half a second and we tested the models' performance for different prediction horizons. More specifically, the horizons go from one point long to 200. We utilized the sliding window approach to obtain nonoverlapping
prediction horizons.
The general performance was compared to similar studies for time series prediction. It was found that the models that performed best for the other studies agreed with our study but with more improvements compared to the ANN baseline. It was observed that the best model for shortrange
prediction was the LSTM model for a prediction
horizon of ten points. It gave an error of 8.43e6
g which is a 95.2% improvement in
performance compared to the baseline that yielded an error of 1.74e4
g. This is expected
as the LSTM network is superior in short term predictions because it can retain memory that represents the temporal nature of our data. In addition, the prediction time

for the CNN model is 0.49 ms, which makes it the fastest model. Moreover, the CNN, ANN and CNNLSTM
models experimented with in this work, yielded realtime
performance.
The other models can also produce faster predictions using more GPUs or a supercomputer.

The proposed models can be integrated into a mechatronics system installed in structures to dampen the effect of the ground motion caused by an earthquake. The main components are an accelerograph, our models and an MR damper. The accelerograph records the ground motion acceleration which is used as an input to the models. The resulting forecast horizon can change the characteristics of the MR damper for optimum performance to support the structure.
For further investigation and improvement, future work related to this thesis can include the following:

1. Integrate the proposed models into a mechatronics system that supports structures during an earthquake.
2. Explore the implementation of deep neural networks with more sophisticated and dedicated hardware such as a supercomputer. Such machines can reduce the prediction time significantly.
3. Extend the training set to include other earthquake databases from around the world with different parameters compared to the NGAWest2
database. This step
will make the models more robust and produce better predictions.
4. Extend the preliminary work on clustering the metadata to more sophisticated clustering algorithms to identify groups in the dataset. This can then be followed by designing a separate model for each cluster, such that each new signal is directed to the suitable model based on its features.

References

[1] J. F. Cassidy, “Earthquake,” in Encyclopedia of Natural Hazards, P. T. Bobrowsky, Ed. Netherlands: Springer, Jan. 2013, p. 208.

[2] A. J. Crone, “The geology of earthquakes,” Seismological Research Letters, vol. 68, no. 5, pp. 778–779, Sep. 1997.

[3] E. R. Burkett, D. D. Given, and L. M. Jones, ShakeAlert—An earthquake early warning system for the United States west coast, US Geological Survey, 20143083, Jan 2014.

[4] D. D. Given, E. S. Cochran, T. Heaton, E. Hauksson, R. Allen, P. Hellweg, J. Vidale, and P. Bodin, “Technical implementation plan for the ShakeAlert production system: an earthquake early warning system for the west coast of the united states,” US Geological Survey, Rep. no. 2014–1097, May 2014.

[5] A. I. Chung, “The development of earthquake early warning methods,” Nature Reviews Earth & Environment, vol. 1, no. 7, p. 331, Jun. 2020.

[6] D. Truong and K. Ahn, “MR fluid damper and its application to force sensorless damping control system,” in Smart Actuation and Sensing Systems Recent Advances and Future Challenges. InTech, Oct. 2012, ch. 15, pp. 383–424.

[7] M. Ohnaka, The Physics of Rock Failure and Earthquakes. New York: Cambridge University Press, Apr. 2013, ch. 5, pp. 148–150.

[8] E. J. Tarbuck, F. K. Lutgens, and D. Tasa, Earth Science, 10th ed. New Jersey: Prentice Hall, Jan. 2002.

[9] J. Bommer and A. MartínezPereira, “Strong motion parameters: definition, usefulness and predictability,” in 12th World Conference on Earthquake Engineering, vol. 13, Jan. 2000, pp. 127–172.

[10] Q. Kong, D. Trugman, R. Zachary, M. Bianco, B. Meade, and P. Gerstoft, “Machine learning in seismology: Turning data into insights,” Seismological Research Letters, vol. 90, no. 1, pp. 3–14, Nov. 2018.

[11] K. Hipel, Time series modelling of water resources and environmental systems. New York: Elsevier, 1994, ch. 2, pp. 63–64.

[12] T. Raicharoen, C. Lursinsap, and P. Sanguanbhokai, “Application of critical support vector machine to time series prediction,” in Proceedings of the 2003 International Symposium on Circuits and Systems, vol. 5, May 2003, pp. 741–744.

[13] B. Babusiak and J. Mohylová, “The EEG signal prediction by using neural network,” Advances in Electrical and Electronic Engineering, vol. 7, pp. 342–345, Jan. 2008.

[14] V. N. Coelho, I. M. Coelho, B. N. Coelho, M. J. F. Souza, F. G. Guimaraes, E. J. da S. Luz, A. C. Barbosa, M. N. Coelho, G. G. Netto, R. C. Costa, A. A. Pinto, A. de P. Figueiredo, M. E. V. Elias, D. C. O. G. Filho, and T. A. Oliveira, “EEG time series learning and classification using a hybrid forecasting model calibrated with GVNS,” Electronic Notes in Discrete Mathematics, vol. 58, pp. 79–86, Apr. 2017.

[15] S. Selvin, R. Vinayakumar, E. A. Gopalakrishnan, V. K. Menon, and K. P. Soman, "Stock price prediction using LSTM, RNN and CNNsliding window model," in 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Sep. 2017, pp. 1643–1647.

[16] A. Romazanov, A. Zakharov, and I. Zakharova, "Temperature prediction in a public building using artificial neural network," in Proceedings of the 8th Scientific Conference on Information Technologies for Intelligent Decision Making Support (ITIDS). Atlantis Press, Nov. 2020, pp. 30–34.

[17] X. Qing and Y. Niu, "Hourly dayahead solar irradiance prediction using weather forecasts by LSTM," Energy, vol. 148, pp. 461–468, Apr. 2018.

[18] S. Khan, H. Rahmani, S. Shah, and M. Bennamoun, "A guide to convolutional neural networks for computer vision," Synthesis Lectures on Computer Vision, vol. 8, no. 1, pp. 1–207, Feb. 2018.

[19] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," IEEE Computational Intelligence Magazine, vol. 13, no. 3, pp. 55–75, Aug. 2018.

[20] D. Hussain, T. Hussain, A. Khan, S. Naqvi, and A. Jamil, "A deep learning approach for hydrological timeseries prediction: A case study of Gilgit river basin," Earth Science Informatics, vol. 13, no. 3, pp. 915–927, Jun. 2020.

[21] I. E. Livieris, E. Pintelas, and P. Pintelas, "A CNN–LSTM model for gold price timeseries

forecasting," Neural Computing and Applications, vol. 32, no. 23, pp. 17 351–17 360, Apr. 2020.

[22] D. Qin, J. Yu, G. Zou, R. Yong, Q. Zhao, and B. Zhang, "A novel combined prediction scheme based on CNN and LSTM for urban PM2.5 concentration," IEEE Access, vol. 7, pp. 20 050–20 059, Feb. 2019.

[23] D. Boore, "Stochastic simulation of highfrequency ground motions based on seismological

models of the radiated spectra," Bulletin of the Seismological Society of America, vol. 73, no. 6A, pp. 1865–1894, Dec. 1983.

[24] Z. Gülerce, R. Kamai, N. A. Abrahamson, and W. J. Silva, "Ground motion prediction equations for the vertical ground motion component based on the NGAW2 database," Earthquake Spectra, vol. 33, no. 2, pp. 499–528, May 2017.

[25] M. Wyss, "Ten years of realtime earthquake loss alerts," in Earthquake Hazard, Risk and Disasters. Boston: Academic Press, Dec. 2014, ch. 9, pp. 143–165.

[26] B. RouetLeduc, C. L. Hulbert, N. Lubbers, K. M. Barros, C. Humphreys, and P. A. Johnson, "Machine learning predicts laboratory earthquakes," Geophysical Research Letters, vol. 44, no. 18, pp. 9276–9282, Sep. 2017.

[27] S. Chakraverty, P. Gupta, and S. Sharma, "Neural networkbased simulation for response identification of twostorey shear building subject to earthquake motion," Neural Computing and Applications, vol. 19, pp. 367–375, Apr. 2010.

[28] T. Kerh and S. Ting, "Neural network estimation of ground peak acceleration at stations along Taiwan highspeed rail system," Engineering Applications of Artificial Intelligence, vol. 18, no. 7, pp. 857–866, Oct. 2005.

[29] C. Arjun and A. Kumar, “Neural network estimation of duration of strong ground motion using Japanese earthquake records,” Soil Dynamics and Earthquake Engineering, vol. 31, no. 7, pp. 866–872, Jul. 2011.

[30] G. Kemal and G. Ayten, “Peak ground acceleration prediction by artificial neural networks for northwestern Turkey,” Mathematical Problems in Engineering, vol. 2008, Article ID: 919420, Nov 2008.

[31] A. PozosEstrada, R. Gomez, and H. Hong, “Use of neural network to predict the peak ground accelerations and pseudo spectral accelerations for Mexican inslab and interplate earthquakes,” Geofísica internacional, vol. 53, pp. 39–57, Mar. 2014.

[32] J. Dhanya and S. Raghukanth, “Ground motion prediction model using artificial neural network,” Pure and Applied Geophysics, vol. 175, no. 3, pp. 1035–1064, Dec. 2017.

[33] R. R. Leach and F. U. Dowla, “Earthquake early warning system using realtime signal processing,” in Neural Networks for Signal Processing VI. Proceedings of the 1996 IEEE Signal Processing Society Workshop, Sep. 1996, pp. 463–472.

[34] H. Adeli and A. Panakkat, “A probabilistic neural network for earthquake magnitude prediction,” Neural networks: the official journal of the International Neural Network Society, vol. 22, pp. 1018–1024, Jun. 2009.

[35] H. Kuyuk and O. Susumu, “Realtime classification of earthquake using deep learning,” Procedia Computer Science, vol. 140, pp. 298–305, Jan. 2018.

[36] J. Ramirez and F. Meyer, “Machine learning for seismic signal processing: Phase classification on a manifold,” in 2011 10th International Conference on Machine Learning and Applications and Workshops, vol. 1, Dec. 2011, pp. 382–388.

[37] M. Böse, F. Wenzel, and M. Erdik, “Preseis: A neural networkbased approach to earthquake early warning for finite faults,” Bulletin of the Seismological Society of America, vol. 98, no. 1, pp. 366–382, Feb. 2008.

[38] E. Alpaydin, Introduction to Machine Learning, 2nd ed. Massachusetts: MIT Press, Dec. 2009, ch. 11, pp. 237–243.

[39] F. Li, L. Tran, K. Thung, S. Ji, D. Shen, and J. Li, “A robust deep model for improved classification of AD/MCI patients,” IEEE Journal of Biomedical and Health Informatics, vol. 19, no. 5, pp. 1610–1616, Oct. 2015.

[40] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Massachusetts: MIT Press, 2016, ch. 1, pp. 2–3.

[41] S. Ruder, “An overview of gradient descent optimization algorithms,” CoRR, vol. abs/1609.04747, Jun 2017.

[42] I. Nunes Silva, D. Hernane Spatti, R. Andrade Flauzino, L.H.B. Liboni, and S. F. dos Reis Alves, Artificial Neural Networks: A Practical Course, 1st ed. Springer Publishing Company, Incorporated, Aug. 2016.

[43] L. Marchi, Handson neural networks: learn how to build and train your first neural network model using Python. Birmingham: Packt Publishing, May 2019, ch. 3, pp. 65–67.

[44] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. Inman, "1D convolutional neural networks and applications: A survey," CoRR, vol. abs/1905.03554, May 2019.

[45] O. Avci, O. Abdeljaber, S. Kiranyaz, and D. J. Inman, "Structural damage detection in real time: Implementation of 1D convolutional neural networks for SHM applications," in Structural Health Monitoring & Damage Detection, Volume 7, C. Niezrecki, Ed. Springer International Publishing, Mar. 2017, pp. 49–54.

[46] O. Abdeljaber, O. Avci, S. Kiranyaz, M. Gabbouj, and D. J. Inman, "Realtime vibrationbased structural damage detection using onedimensional convolutional neural networks," Journal of Sound and Vibration, vol. 388, pp. 154–170, Feb. 2017.

[47] C. C. Aggarwal, "Training deep neural networks," in Neural Networks and Deep Learning. Springer International Publishing, Aug. 2018, ch. 3, pp. 105–167.

[48] Y. Bengio, P. Simard, and P. Frasconi, "Learning longterm dependencies with gradient descent is difficult," IEEE Transactions on Neural Networks, vol. 5, no. 2, pp. 157–166, Mar. 1994.

[49] S. Hochreiter and J. Schmidhuber, "Long shortterm memory," Neural Computation, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[50] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics, Oct. 2014, pp. 1724–1734.

[51] A. Graves, A. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, May 2013, pp. 6645–6649.

[52] H. Sak, A. Senior, and F. Beaufays, "Long shortterm memory based recurrent neural network architectures for large vocabulary speech recognition," CoRR, vol. abs/1402.1128, Feb 2014.

[53] Z. Voulgaris and Y. Bulut, AI for Data Science: Artificial Intelligence Frameworks and Functionality for Deep Learning, Optimization, and Beyond. New Jersey: Technics Publications, 2018, ch. 9, pp. 173–179.

[54] L. Jiang and G. Hu, "Dayahead price forecasting for electricity market using longshort term memory recurrent neural network," in 2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV), Nov. 2018, pp. 949–954.

[55] T. D. Ancheta, R. B. Darragh, J. P. Stewart, E. Seyhan, W. J. Silva, B. S.J. Chiou, K. E. Wooddell, R. W. Graves, A. R. Kottke, D. M. Boore, T. Kishida, and J. L. Donahue, "PEER NGAWest2 database," Pacific Earthquake Engineering Research Center, California, Rep. no. 2013/03, May 2013.

[56] T. D. Ancheta, R. B. Darragh, J. P. Stewart, E. Seyhan, W. J. Silva, B. S.J. Chiou, K. E. Wooddell, R. W. Graves, A. R. Kottke, D. M. Boore, T. Kishida, and J. L. Donahue, "NGAWest2 database," Earthquake Spectra, vol. 30, no. 3, p. 4, Aug. 2014.

[57] C. C. Aggarwal, “Convolutional neural networks,” in Neural Networks and Deep Learning. Springer International Publishing, Aug. 2018, ch. 8, pp. 315–371.

[58] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in International Conference on Learning Representations, vol. abs/1412.6980, Feb 2017.

[59] J. Han, M. Kamber, and J. Pei, Data Mining: Concepts and Techniques, 3rd ed. California: Morgan Kaufmann Publishers Inc., Jun. 2011, pp. 77–78.

[60] H. Cheng, P. Tan, J. Gao, and J. Scripps, “Multistepahead time series prediction,” in Advances in Knowledge Discovery and Data Mining. Springer Berlin Heidelberg, Apr. 2006, pp. 765–774.

[61] Y. Geng, L. Su, Y. Jia, and C. Han, “Seismic events prediction using deep temporal convolution networks,” Journal of Electrical and Computer Engineering, vol. 2019, pp. 1–14, Apr. 2019.

Appendix A: Time Series Prediction ANN

50 51 52 53 54 55

-2

0

2

4

10-3 H2

0 20 40 60

Time in Seconds

-0.02

0

0.02

V

50 51 52 53 54 55

-2

0

2

10-3 V

Figure 5.1. ANN prediction vs actual acceleration from cluster 1, window size is 1.

0 50 100 150 200

-0.02

0

0.02

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 50 100 150 200

-0.01

0

0.01

Acceleration in g

H2

50 51 52 53 54 55

-5

0

5

10-3 H2

0 50 100 150 200

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-2

0

2

10-3 V

Figure 5.2. ANN prediction vs actual acceleration from cluster 2, window size is 1.

0 20 40 60

-0.04

-0.02

0

0.02

H1

Predicted

Actual

50 51 52 53 54 55

-5

0

5

10

10-3 H1

0 20 40 60

-0.02

0

0.02

Acceleration in g

H2

50 51 52 53 54 55

-4

-2

0

2

10-3 H2

0 20 40 60

Time in Seconds

-0.02

0

0.02

V

50 51 52 53 54 55

-2

0

2

10-3 V

Figure 5.3. ANN prediction vs actual acceleration from cluster 1, window size is 10.

0 50 100 150 200

-0.02

0

0.02

H1

Predicted

Actual

50 51 52 53 54 55

-5

0

5

10-3 H1

0 50 100 150 200

-0.01

0

0.01

Acceleration in g

H2

50 51 52 53 54 55

-5

0

5

10-3 H2

0 50 100 150 200

Time in Seconds

-5

0

5

10

10-3 V

50 51 52 53 54 55

-2

0

2

10-3 V

Figure 5.4. ANN prediction vs actual acceleration from cluster 2, window size is 10.

80

0 20 40 60

-0.02

0

0.02

H1

Predicted

Actual

50 51 52 53 54 55

-5

0

5

10-3 H1

0 20 40 60

-0.02

0

0.02

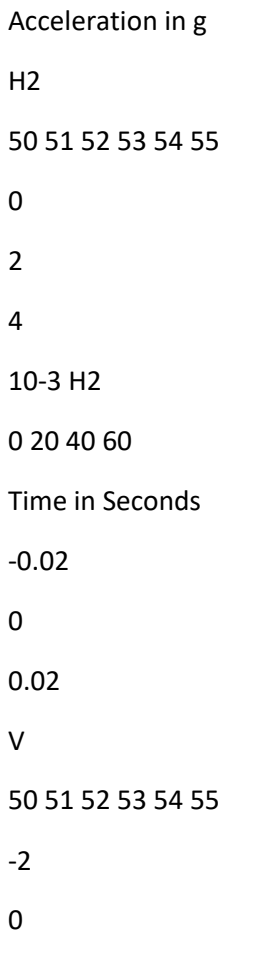
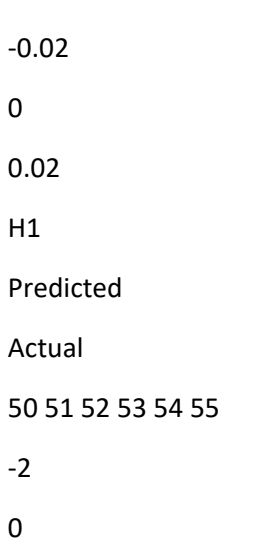


Figure 5.5. ANN prediction vs actual acceleration from cluster 1, window size is 50.



2

10-3 H1

0 50 100 150 200

-0.01

0

0.01

Acceleration in g

H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 50 100 150 200

Time in Seconds

-10

-5

0

5

10-3 V

50 51 52 53 54 55

-2

-1

0

1

10-3 V

Figure 5.6. ANN prediction vs actual acceleration from cluster 2, window size is 50.

0 20 40 60

-0.02

0

0.02

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

4

10-3 H1

0 20 40 60

-0.02

0

0.02

Acceleration in g

H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 20 40 60

Time in Seconds

-0.02

0

0.02

V

50 51 52 53 54 55

-1

0

1

10-3 V

Figure 5.7. ANN prediction vs actual acceleration from cluster 1, window size is 100.

0 50 100 150 200

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 50 100 150 200

-5

0

5

Acceleration in g

10-3 H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 50 100 150 200

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-2

-1

0

1

10-3 V

Figure 5.8. ANN prediction vs actual acceleration from cluster 2, window size is 100.

0 20 40 60

-0.02

0

0.02

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

4

10-3 H1

0 20 40 60

-0.02

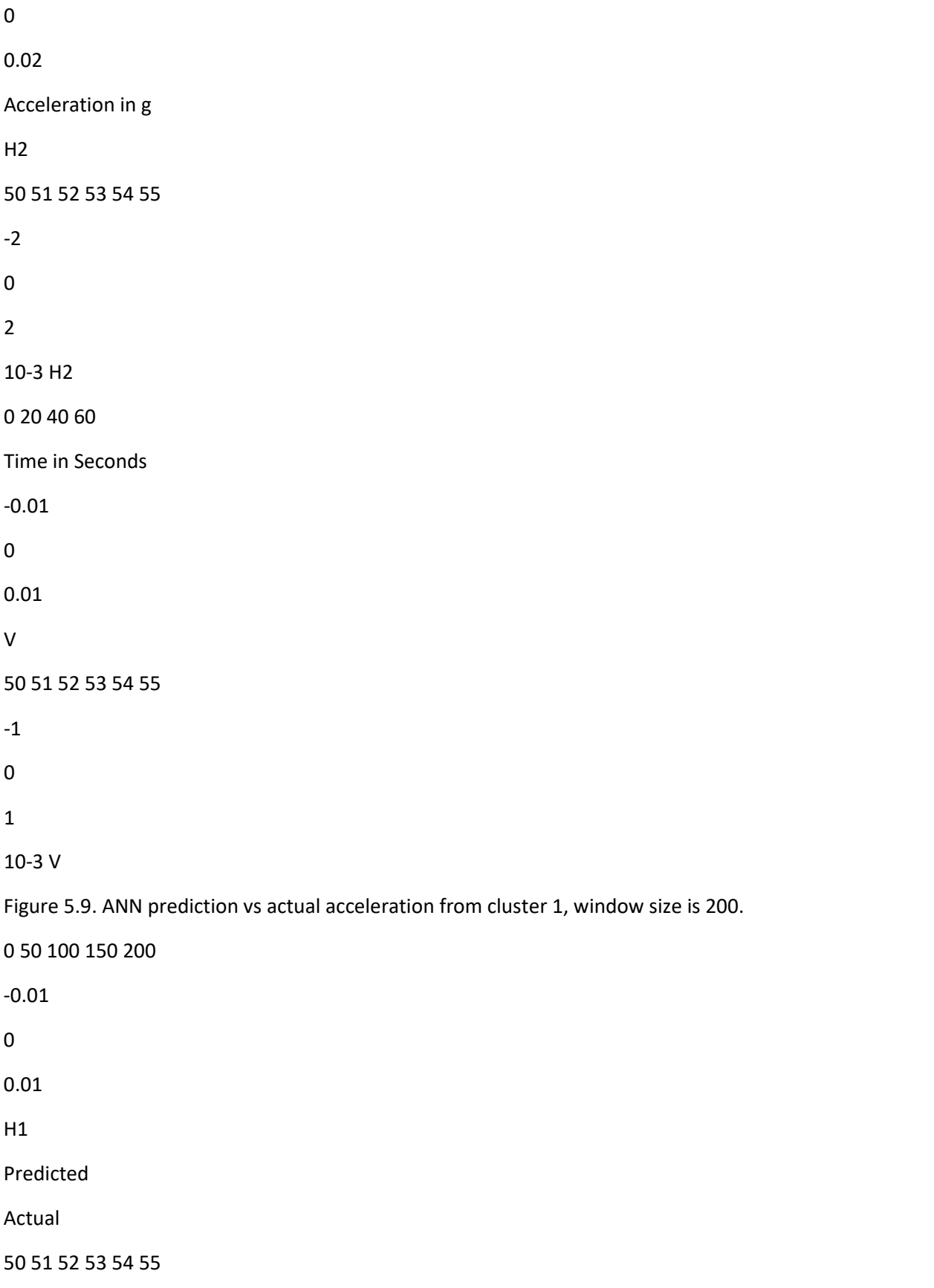


Figure 5.9. ANN prediction vs actual acceleration from cluster 1, window size is 200.

-2

0

2

10-3 H1

0 50 100 150 200

-5

0

5

Acceleration in g

10-3 H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 50 100 150 200

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-2

-1

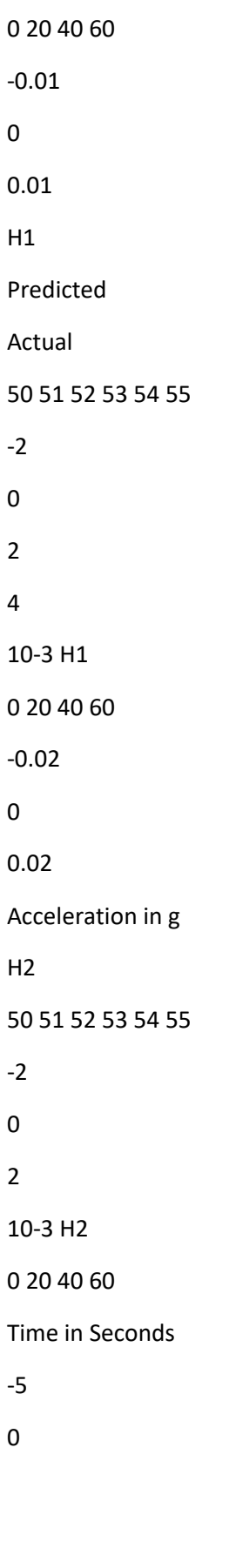
0

1

10-3 V

Figure 5.10. ANN prediction vs actual acceleration from cluster 2, window size is 200.

Appendix B: Time Series Prediction CNN



5

10-3 V

50 51 52 53 54 55

-1

0

1

10-3 V

Figure 5.11. CNN prediction vs actual acceleration from cluster 1, window size is 1.

0 50 100 150 200

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 50 100 150 200

-5

0

5

Acceleration in g

10-3 H2

50 51 52 53 54 55

-2

0

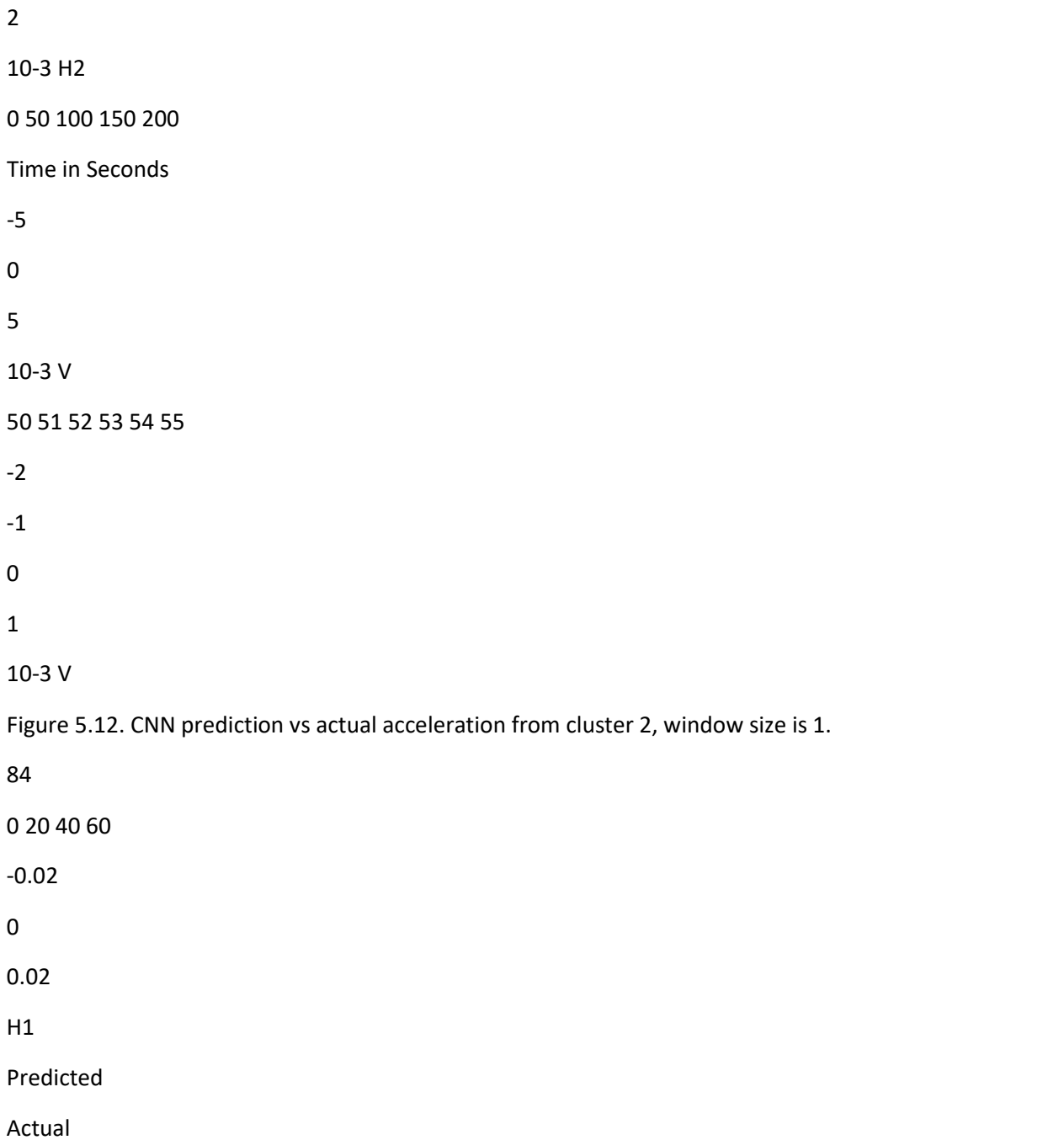


Figure 5.12. CNN prediction vs actual acceleration from cluster 2, window size is 1.

0 20 40 60

-0.02

0

0.02

Acceleration in g

H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 20 40 60

Time in Seconds

-0.01

0

0.01

V

50 51 52 53 54 55

-1

0

1

10-3 V

Figure 5.13. CNN prediction vs actual acceleration from cluster 1, window size is 10.

0 50 100 150 200

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 50 100 150 200

-5

0

5

Acceleration in g

10-3 H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 50 100 150 200

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-2

-1

0

1

10-3 V

Figure 5.14. CNN prediction vs actual acceleration from cluster 2, window size is 10.

5

10

10-3 V

50 51 52 53 54 55

-1

0

1

10-3 V

Figure 5.15. CNN prediction vs actual acceleration from cluster 1, window size is 50

0 50 100 150 200

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 50 100 150 200

-5

0

5

Acceleration in g

10-3 H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 50 100 150 200

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-2

-1

0

1

10-3 V

Figure 5.16. CNN prediction vs actual acceleration from cluster 2, window size is 50.

0 20 40 60

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

4

10-3 H1

0 20 40 60

-0.02

0

0.02

Acceleration in g

H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 20 40 60

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-1

0

1

10-3 V

Figure 5.17. CNN prediction vs actual acceleration from cluster 1, window size is 100.

0 50 100 150 200

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 50 100 150 200

-5

0

5

Acceleration in g

10-3 H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 50 100 150 200

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-2

-1

0

1

10-3 V

Figure 5.18. CNN prediction vs actual acceleration from cluster 2, window size is 100.

0 20 40 60

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

4

10-3 H1

0 20 40 60

-0.02

0

0.02

Acceleration in g

H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 20 40 60

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-1

0

1

10-3 V

Figure 5.19. CNN prediction vs actual acceleration from cluster 1, window size is 200.

0 50 100 150 200

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 50 100 150 200

-5

0

5

Acceleration in g

10-3 H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 50 100 150 200

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-2

-1

0

1

10-3 V

Figure 5.20. CNN prediction vs actual acceleration from cluster 2, window size is 200.

Appendix C: Time Series Prediction RNN

0 20 40 60

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

4

10-3 H1

0 20 40 60

-0.02

0

0.02

Acceleration in g

H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 20 40 60

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-1

0

1

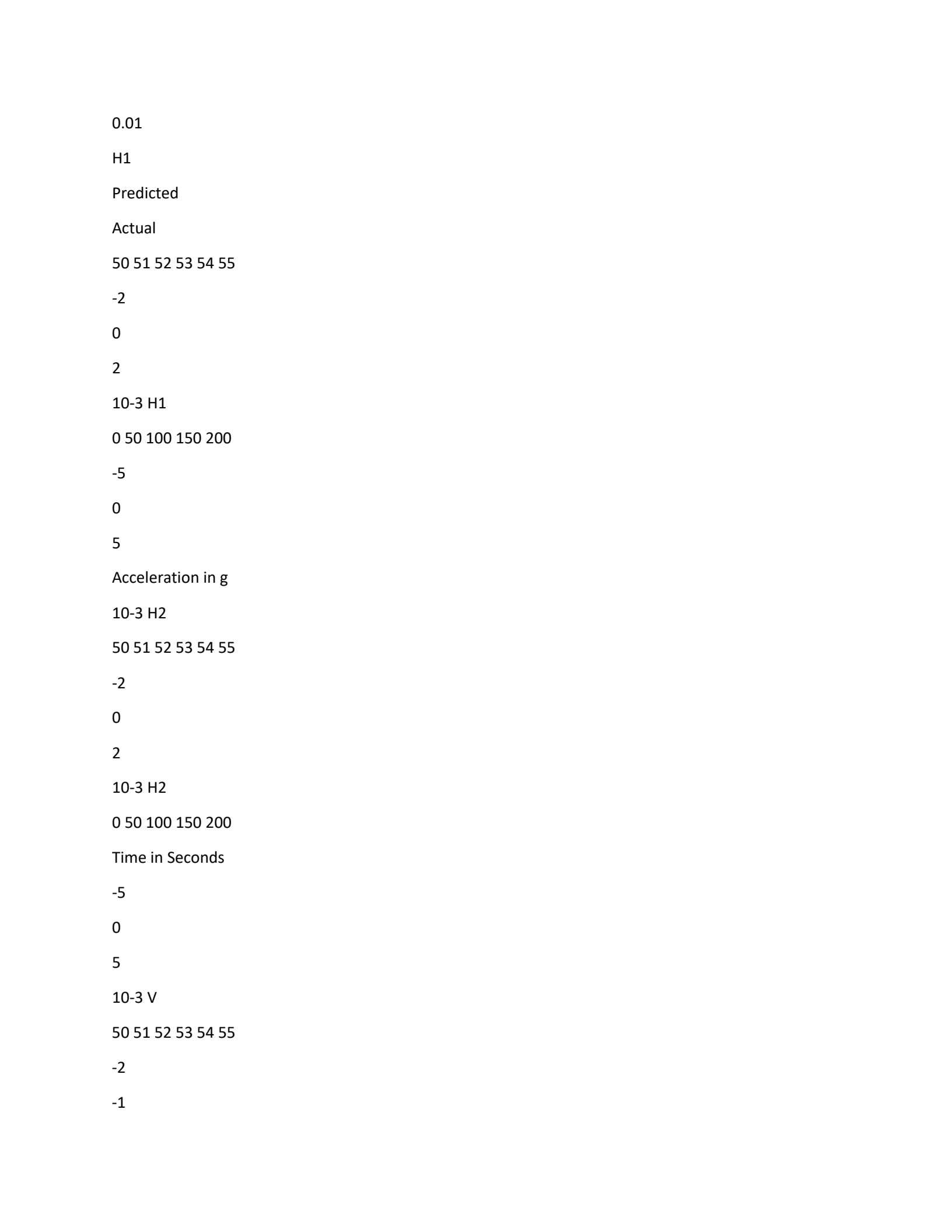
10-3 V

Figure 5.21. RNN prediction vs actual acceleration from cluster 1, window size is 1.

0 50 100 150 200

-0.01

0


0.01
H1
Predicted
Actual
50 51 52 53 54 55
-2
0
2
10-3 H1
0 50 100 150 200
-5
0
5
Acceleration in g
10-3 H2
50 51 52 53 54 55
-2
0
2
10-3 H2
0 50 100 150 200
Time in Seconds
-5
0
5
10-3 V
50 51 52 53 54 55
-2
-1

0

1

10-3 V

Figure 5.22. RNN prediction vs actual acceleration from cluster 2, window size is 1.

89

0 20 40 60

-0.01

0

0.01

0.02

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

4

10-3 H1

0 20 40 60

-0.02

0

0.02

Acceleration in g

H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 20 40 60

Time in Seconds

-0.01

0

0.01

V

50 51 52 53 54 55

-1

0

1

10-3 V

Figure 5.23. RNN prediction vs actual acceleration from cluster 1, window size is 10.

0 50 100 150 200

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 50 100 150 200

-5

0

5

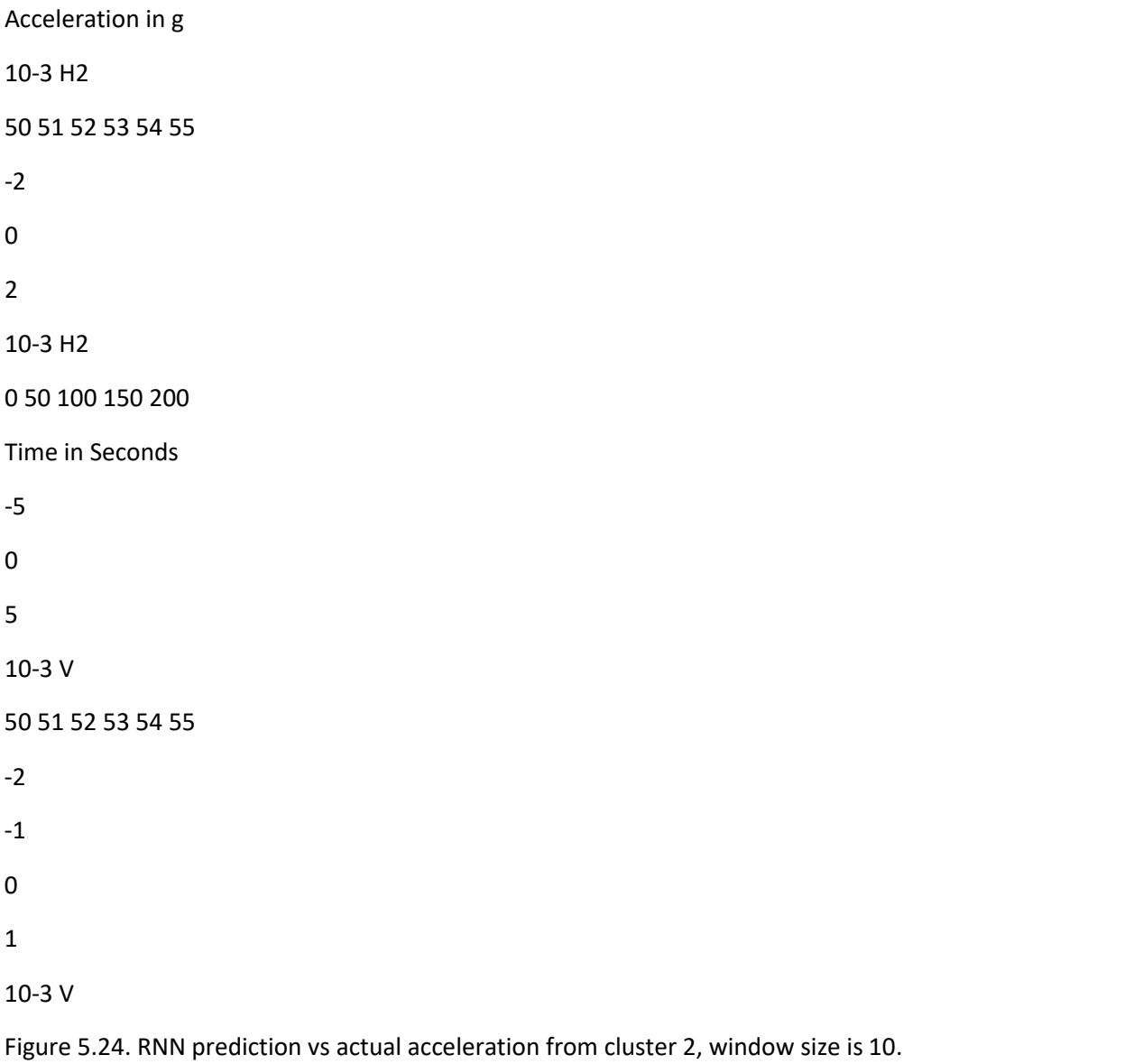


Figure 5.24. RNN prediction vs actual acceleration from cluster 2, window size is 10.

-2

0

2

4

10-3 H1

0 20 40 60

-0.02

0

0.02

Acceleration in g

H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 20 40 60

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-1

0

1

10-3 V

Figure 5.25. RNN prediction vs actual acceleration from cluster 1, window size is 50.

0 50 100 150 200

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 50 100 150 200

-5

0

5

Acceleration in g

10-3 H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 50 100 150 200

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-2

-1

0

1

10-3 V

Figure 5.26. RNN prediction vs actual acceleration from cluster 2, window size is 50.

0 20 40 60

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

4

10-3 H1

0 20 40 60

-0.02

0

0.02

Acceleration in g

H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 20 40 60

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-1

0

1

10-3 V

Figure 5.27. RNN prediction vs actual acceleration from cluster 1, window size is 100.

0 50 100 150 200

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 50 100 150 200

-5

0

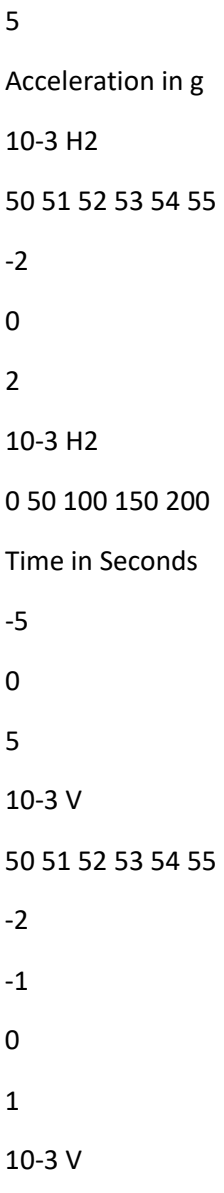
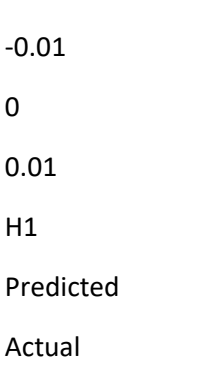


Figure 5.28. RNN prediction vs actual acceleration from cluster 2, window size is 100.



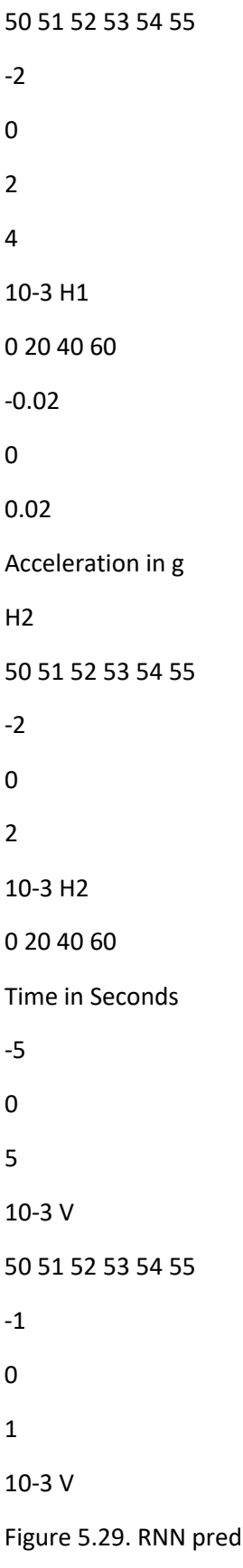


Figure 5.29. RNN prediction vs actual acceleration from cluster 1, window size is 200.

0 50 100 150 200

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 50 100 150 200

-5

0

5

Acceleration in g

10-3 H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 50 100 150 200

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-2

-1

0

1

10-3 V

Figure 5.30. RNN prediction vs actual acceleration from cluster 2, window size is 200.

Appendix D: Time Series Prediction LSTM

Network

0 20 40 60

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

4

10-3 H1

0 20 40 60

-0.02

0

0.02

Acceleration in g

H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 20 40 60

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-1

0

1

10-3 V

Figure 5.31. LSTM network prediction vs actual acceleration from cluster 1, window size is 1.

0 50 100 150 200

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 50 100 150 200

-5

0

5

Acceleration in g

10-3 H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 50 100 150 200

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-2

-1

0

1

10-3 V

Figure 5.32. LSTM network prediction vs actual acceleration from cluster 2, window size is 1.

0 20 40 60

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

4

10-3 H1

0 20 40 60

-0.02

0

0.02

Acceleration in g

H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 20 40 60

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-1

0

1

10-3 V

Figure 5.33. LSTM network prediction vs actual acceleration from cluster 1, window size is 10.

0 50 100 150 200

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 50 100 150 200

-5

0

5

Acceleration in g

10-3 H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 50 100 150 200

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-2

-1

0

1

10-3 V

Figure 5.34. LSTM network prediction vs actual acceleration from cluster 2, window size is 10.

95

0 20 40 60

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

4

10-3 H1

0 20 40 60

-0.02

0

0.02

Acceleration in g

H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 20 40 60

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-1

0

1

10-3 V

Figure 5.35. LSTM network prediction vs actual acceleration from cluster 1, window size is 50.

0 50 100 150 200

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 50 100 150 200

-5

0

5

Acceleration in g

10-3 H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 50 100 150 200

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-2

-1

0

1

10-3 V

Figure 5.36. LSTM network prediction vs actual acceleration from cluster 2, window size is 50.

0

5

10-3 V

50 51 52 53 54 55

-2

-1

0

1

10-3 V

Figure 5.37. LSTM network prediction vs actual acceleration from cluster 1, window size is 100.

0 50 100 150 200

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 50 100 150 200

-5

0

5

Acceleration in g

10-3 H2

Figure 5.38. LSTM network prediction vs actual acceleration from cluster 2, window size is 100.

0

2

4

10-3 H1

0 20 40 60

-0.02

0

0.02

Acceleration in g

H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 20 40 60

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-1

0

1

10-3 V

Figure 5.39. LSTM network prediction vs actual acceleration from cluster 1, window size is 200.

0 50 100 150 200

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 50 100 150 200

-5

0

5

Acceleration in g

10-3 H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 50 100 150 200

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-2

-1

0

1

10-3 V

Figure 5.40. LSTM network prediction vs actual acceleration from cluster 2, window size is 200.

Appendix E: Time Series Prediction CNNLSTM Network

0 50 100 150 200

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 50 100 150 200

-5

0

5

Acceleration in g

10-3 H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 50 100 150 200

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-2

-1

0

1

10-3 V

Figure 5.41. CNNLSTM

network prediction vs actual acceleration from cluster 1,

window size is 1.

0 50 100 150 200

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 50 100 150 200

-5

0

5

Acceleration in g

10-3 H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 50 100 150 200

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-2

-1

0

1

10-3 V

Figure 5.42. CNNLSTM

network prediction vs actual acceleration from cluster 2,

window size is 1.

0 20 40 60

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

4

10-3 H1

0 20 40 60

-0.02

0

0.02

Acceleration in g

H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 20 40 60

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-1

0

1

10-3 V

Figure 5.43. CNNLSTM

network prediction vs actual acceleration from cluster 1,

window size is 10.

0 50 100 150 200

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 50 100 150 200

-5

0

5

Acceleration in g

10-3 H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 50 100 150 200

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-2

-1

0

1

10-3 V

Figure 5.44. CNNLSTM

network prediction vs actual acceleration from cluster 2,

window size is 10.

100

0 20 40 60

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

4

10-3 H1

0 20 40 60

-0.02

0

0.02

Acceleration in g

H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 20 40 60

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-1

0

1

10-3 V

Figure 5.45. CNNLSTM

network prediction vs actual acceleration from cluster 1,

window size is 50.

0 50 100 150 200

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 50 100 150 200

-5

0

5

Acceleration in g

10-3 H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 50 100 150 200

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-2

-1

0

1

10-3 V

Figure 5.46. CNNLSTM

network prediction vs actual acceleration from cluster 2,

window size is 50.

101

0 50 100 150 200

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 50 100 150 200

-5

0

5

Acceleration in g

10-3 H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 50 100 150 200

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-2

-1

0

1

10-3 V

Figure 5.47. CNNLSTM

network prediction vs actual acceleration from cluster 1,

window size is 100.

0 50 100 150 200

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

Figure 5.48. CNNLSTM

network prediction vs actual acceleration from cluster 2,

window size is 100.

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

4

10-3 H1

0 20 40 60

-0.02

0

0.02

Acceleration in g

H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 20 40 60

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-1

0

1

10-3 V

Figure 5.49. CNNLSTM

network prediction vs actual acceleration from cluster 1,

window size is 200.

0 50 100 150 200

-0.01

0

0.01

H1

Predicted

Actual

50 51 52 53 54 55

-2

0

2

10-3 H1

0 50 100 150 200

-5

0

5

Acceleration in g

10-3 H2

50 51 52 53 54 55

-2

0

2

10-3 H2

0 50 100 150 200

Time in Seconds

-5

0

5

10-3 V

50 51 52 53 54 55

-2

-1

0

1

10-3 V

Figure 5.50. CNNLSTM

network prediction vs actual acceleration from cluster 2,

window size is 200.

Vita

Sara Tellab was born in Algeria and received her bachelor of science degree in Electrical Engineering / Communication from Ajman University, UAE. Sara graduated top of her class in 2017. She joined the Mechatronics Graduate Program at AUS in 2018, where she both studied and worked as a graduate teaching assistant. Her current research interests include deep learning, image processing, computer vision and mobile robots.