Machine Learning in Indoor Positioning and Channel Prediction Systems

by

Yizhou Zhu
B.Eng., Zhejiang University, 2010

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF APPLIED SCIENCE

in the Department of Electrical and Computer Engineering

Machine Learning in Indoor Positioning and Channel Prediction Systems

by

Yizhou Zhu
B.Eng., Zhejiang University, 2010

Supervisory Committee

Dr. Xiaodai Dong, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Wu-Sheng Lu, Departmental Member
(Department of Electrical and Computer Engineering)

Supervisory Committee

Dr. Xiaodai Dong, Supervisor
(Department of Electrical and Computer Engineering)

Dr. Wu-Sheng Lu, Departmental Member
(Department of Electrical and Computer Engineering)

ABSTRACT

In this thesis, the neural network, a powerful tool which has demonstrated its ability in many fields, is studied for the indoor localization system and channel prediction system. This thesis first proposes a received signal strength indicator (RSSI) fingerprinting-based indoor positioning system for the widely deployed WiFi environment, using deep neural networks (DNN). To reduce the computing time as well as improve the estimation accuracy, a two-step scheme is designed, employing a classification network for clustering and several regression networks for final location prediction. A new fingerprinting, which utilizes the similarity in RSSI readings of the nearby reference points (RPs) is also proposed. Real-time tests demonstrate that the proposed algorithm achieves an average distance error of 43.5 inches. Then this thesis extends the ability of the neural network to the physical layer communications by introducing a recurrent neural network (RNN) based approach for real-time channel prediction which uses the recent history channel state information (CSI) estimation for online training before prediction, to adapt to the continuously changing channel to gain a more accurate CSI prediction compared to the other conventional methods. Furthermore, the proposed method needs no additional knowledge, neither the internal properties of the channel itself nor the external features that affect the channel propagation. The proposed approach outperforms the other methods in a changing environment in the simulation test, validating it a promising method for channel prediction in wireless communications.

Contents

List of Tables

List of Figures

List of Abbreviations

AR Autoregressive

BEM Basis-Expansion Model

CDF Cumulative Distribution Function

CE Complex Exponential

CNN Convolutional Neural Network

CSI Channel State Information

DANN Discriminant-Adaptive Neural Network

DNN Deep Neural Network

DPS Discrete Prolate Spheroidal

ELM Extreme Learning Machine

FDD Frequency Division Duplex

GNSS Global Navigation Satellite Systems

GPS Global Positioning System

GRNN Generalized Regression Neural Network

IPS Indoor Positioning System

KNN K Nearest Neighbours

LBS Location-based Service

LMS Least Mean Squares

LSTM Long Short Term with Memory

MDA Multiple Discriminant Analysis

ME Minimum-Energy

MIMO Multiple Input Multiple Output

ML Maximum Likelihood

MLP Multilayer Perceptron

MMSE Minimum Mean Square Error

mmWave Millimetre Wave

MSE Mean Square Error

NN Neural Network

PCA Principal Component Analysis

PRC Parametric Radio Channel

RFID Radio Frequency Identification

RLS Recursive Least Squares

RNN Recurrent Neural Network

RP Reference Point

RSSI Received Signal Strength Indicator

SCM Spatial Channel Model

SISO Single Input Single Output

SVM Support Vector Machine

TDD Time Division Duplex

WLAN Wireless Local Area Network

ACKNOWLEDGEMENTS

I would like to thank:

My wife, my family and my cat seven for supporting me in the low moments. It was the hardest two years for us, but also the most unforgettable, during which we got married and adapted to the new life here in Victoria. I would like to express my endless gratitude to my wife, Yue, for her love, support and encouragement.

Supervisor Dr. Xiaodai Dong for mentoring, support, encouragement, and patience. It was so simple a phone call three years ago which made it real that I could be here to study and research. Furthermore, her guidance, support and encouragement paved the way for me as a researcher, to open mind and think different.

Dr. Tao Lu and Dr. Wu-Sheng Lu for mentoring and guidance. They have provided me with professional guidance and suggestions about the projects that I involved in and the way of thinking.

My colleagues and my friends for their support and help in the last two years. It is them that made the journey here full of memory and pleasure. I would thank my colleagues, Minh Tu Hoang, Ahmed Magdy Elmoogy, Tyler Reese, Brosnan Yuen, Yiming Huo, Jun Zhou, Ping Cheng, and my friends, Weizheng Li, Ji Shi, Kris Haynes, Jeff Martens for all the thing you guys have done for me and my family.

Yizhou Zhu
Victoria, BC, Canada
July, 2018

DEDICATION

To my wife, my family
for
Everytyhing you have done.

Chapter 1

Introduction

1.1 Overview

Though much of the theory was developed 20 years ago, neural networks (NNs) have become very popular in recent years because of the expanding data and the development of the computing infrastructure. With recent events such as the Go match between the 18-time world champion Lee Sedol and AlphaGo, a Go program developed by Deepmind, machine learning has received more and more attention. As a result of Moore's Law, the computing power doubles every 18 months over the past decades. Although GPUs are designed for output to a display device, its parallel computing power is well-suited for training NNs, which are structured in a very uniform manner such that at each layer of the network identical artificial neurons perform the same computation.

In this thesis, we proposed two neural network applications for indoor localization and physical layer communication respectively, to explore the ability of neural networks in different areas.

1.1.1 Neural Network

A neural network system is a system that processes data in a way that biological neurons do. Unlike the other expert systems that need task-specific programming, it can "learn" logic or relation inside of the large data instead of a given model. An NN is based on several collections of nodes called neurons which are connected with each other in a particular way. A neuron that takes the input from other neurons can process it and then send the result to the connected neurons for further use. The

network processes the input this way and compares the output of the network itself with the corresponding known result. Then adjustment based on the error is fed back into the network to modify the weights of the neurons inside to "learn" from the data.

There are three dominant type of NNs that are widely used now, deep neural network (DNN), convolutional neural network (CNN) and recurrent neural network (RNN), which are defined by the network architecture.

DNN

is a network that has multiple hidden layers between the input and output layer to enable the ability to model complex non-linear system. DNNs are typically feedforward networks in which data flows from the input layer to the output layer without looping back. Multilayer perceptron (MLP) is a commonly used feedforward DNN and uses backpropagation to train, which has at least three layers of neurons, and each layer uses a nonlinear activation function except the input layer.

CNN

uses a variation of MLP, which has been successfully applied in visual recognition and classification. A CNN usually consists of convolutional layers, pooling layers, fully connected layers and normalization layers. A CNN is easier to train and have many fewer parameters than fully connected networks with the same number of hidden units.

RNN

is a type of NN that the neurons do not always feedforward, but connect to the neurons in the previous layer or the current layer itself. Thus RNN can use their internal state or memory to process the input so it is suitable for problems that can be formulated into a time sequence.

1.1.2 Indoor Localization with Deep Neural Network

As the fast development of the wireless communication and mobile devices, the location-based services (LBSs) have been driven so fast in many application scenes, which raises a massive demand for high accuracy of localization. Although global navigation satellite systems (GNSS) is widely used in the outdoor environment to obtain

a highly accurate location estimate, it is still a challenge in indoor areas as the GNSS signals from satellites cannot often be seen. Thus an accurate indoor positioning system (IPS) became a fundamental problem for many upper-level applications.

A WiFi based IPS is a promising approach for human because of the widely deployed WiFi infrastructure and the fast increasing WiFi-enabled mobile devices, which makes the system low cost and easy to deploy. Although different sensor-based systems are widely studied, such as Bluetooth [1, 35], radio frequency identification (RFID) [20] and ultrasound [33], they often focus on objective tracking and need corresponding hardware.

IPS could be classified into two classes, ranging based and fingerprinting based. Ranging based ones derive distance between the transmitter and receiver by using different kinds of sensor data and assuming a particular propagation model [22] and then calculate the position based on triangulation. However, due to the changing environment and the multi-path phenomenon, its performance is not comparable to the fingerprinting approaches, which associates a group of physical measurements at each reference point (RP) as a fingerprint, perform like pattern matching systems, comparing the similarity between the target fingerprint and those in the database to return the best match to be the estimation.

Some conventional experts system used in IPSs includes K nearest neighbours (KNN) [4, 29, 39, 42], support vector machine (SVM) [34, 21], filter-based [5, 3] and NN based [14, 19, 7, 24] algorithms.

Depending on the output type of the network, existing NN based IPSs can be grouped into two categories, classification and regression. The classification type outputs the predicted label of the unknown location while the regression type directly returns the exact coordinates. In the literature, DNN [14, 19] is the most frequently used NN for IPS while CNN [7] and RNN [24] are also implemented in some ways.

1.1.3 Channel State Information Prediction with Recurrent Neural Network

Since the rapid development of machine learning in a wide range of applications, researchers explore its employment in communication, such as radio resource management, network optimization and other higher layer aspects. Physical layer designs are also studied, ranging from channel estimation and detection [40], decoding [28, 16] to equalization [8, 10], spectrum usage recognition [15], etc.

Classical communication theory develops statistic models based on assumptions. But 5G and future generation systems may employ a large number of antennas when millimetre wave (mmWave) bands are used, accurate modelling using classical theory is increasingly difficult and complex. Thus machine learning based systems become a promising solution as they establish the internal relationship, which is not easily describable by mathematical formulas, based on a large number of training data.

Obtaining channel state information (CSI) is vital to both transmitter and receiver for high spectral efficiency. Due to the instability of the wireless propagation channel caused by user mobilities and changing dynamics in the environment, a pilot based technique is applied by transmitting known pilot symbols, also called reference symbols, between transmitter and receiver to estimate the channel in real time and then interpolate or extrapolate CSI estimation at non-pilot positions. Channel estimation is often done on the receiver side, thus for the transmitter to know the channel, either CSI feedback is required in frequency division duplex (FDD) or pilots are transmitted in the opposite direction and assume channel reciprocity in time division duplex (TDD). The resource consumed and the time delay caused by the feedback is significant for CSI estimation in a fast-changing channel. Therefore, channel prediction is very useful in this case [13, 12].

The conventional channel prediction techniques can be divided into three groups, the parametric radio channel (PRC) model [2, 36], basis-expansion model (BEM) [41] and the autoregressive (AR) model [23, 17, 13, 18]. These methods predict CSI based on certain theoretical channel propagation models and/or estimation of channel long-term statistics and channel parameters.

On the other hand, instead of deriving equations based on assumptions and propagation models, machine learning based approaches train a learning model, e.g., a DNN [11] or a CNN [25], using a large known dataset to find the internal relation underneath. The performance in simulation and/or extensive experiments shows these approaches are comparable to, or even better than the conventional methods.

1.2 Summary of Contributions

In this thesis, the main results are presented in Chapters 3 and 4, which are summarized below.

Chapter 3 presents a DNN based RSSI fingerprinting indoor localization system that reaches a result of 43.5 inches mean localization error with 84 % of its predictions

under the error of 60 inches as well as reduces the time complexity significantly compared to KNN models in real experiment test. The problem formulation, model establishment, database buildup are present and illustrated and a real experiment is performed to demonstrate the ability of NN in analyzing the RSSI readings and predicting location estimation for indoor localization. Based on the principal idea of fingerprinting systems, a new fingerprinting for RSSI based system is proposed by utilizing the similarity in RSSI readings between two closeby points. To reduce the computing time as well as improve the estimation accuracy, a scheme employing a classification network for clustering and several localization networks for final location prediction is designed. Furthermore, a median filter pre-processor is applied before the data is fed into the network, to reduce the impact of the RSSI fluctuation. Finally, real experiments performed in the lab area provides supportive results for our performance analysis.

Chapter 4 investigates an online training based RNN real-time CSI prediction approach that has the best performance compared to the conventional AR model and the offline trained RNN. Since channel prediction is one of the fundamental techniques in wireless communication, the improvement in prediction achieved by the proposed system could further increase the channel bandwidth and enhance the stability. By introducing an online training scheme using the recent history data and a properly designed time schedule, the proposed system can then adapt to the continuously changing channel to give a better prediction with no additional knowledge of the channel. Initial simulation result shows the proposed method has the best performance, which could demonstrate the RNN has the ability in learning and predicting the time sequential CSI data in a changing environment.

1.3 Organizations

The rest of this thesis is organized as follows.

Chapter 2 briefly describes the basic concept, the architecture of the neural network and the mathematical algorithms used for training and their derivation.

Chapter 3 considers a WiFi RSSI based indoor localization problem. After a full research of the existing approaches, we introduce a new two-step scheme using several DNNs, one of which is a classification network for clustering and the other are regression networks for final location prediction. In addition, a new RSSI fingerprinting pattern based on the RSSI similarity between closeby points is proposed and a me-

dian filter is applied as a pre-processor before the target RSSI is fed into the network. Real experiment result shows the improvement this two-step scheme could achieve.

Chapter 4 extends our neural network study area to physical layer communication, a real-time CSI prediction system. Considering it could be formulated into a time sequential problem, we proposed a simple but efficient RNN, which is suitable for this type of problems, with an online training scheme based on the recent pilot assisted or data assisted CSI estimation. All the technical details are discussed and an initial simulation test is performed to demonstrate the ability of RNN in learning and analyzing the channel information.

Chapter 5 concludes this thesis and enumerates open problems for further research.

Chapter 2

Neural Network

2.1 Perceptron

The materials presented in this section follow [27].

In machine learning, the perceptron is an algorithm for supervised learning of binary classifiers. It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector. Fig. 2.1 shows what a simple perceptron is like.

A perceptron consists of

- **Weight** A perceptron can have multiple inputs $(x[1], x[2], \ldots, x[n] \,|\, x[i] \in \mathbb{R})$, each of which has its corresponding weight $w[i] \in \mathbb{R}$, with an additional weight call **bias** b, which is $w[0]$ in the Fig. 2.1.
- **Activation Function** The **step function** is used.

$$f(x) = \begin{cases} 1 & x > 0 \\ 0 & otherwise \end{cases}$$

- **Output** The output of this perceptron can be calculated as

$$y = f(\mathbf{w}^T\mathbf{x} + b)$$

2.1.1 An Example of Perceptron Implementing Logic AND

We design a simple perceptron to implement the logic AND function, to show the ability of the perceptron. Table 2.1 shows the truth table of the AND function.

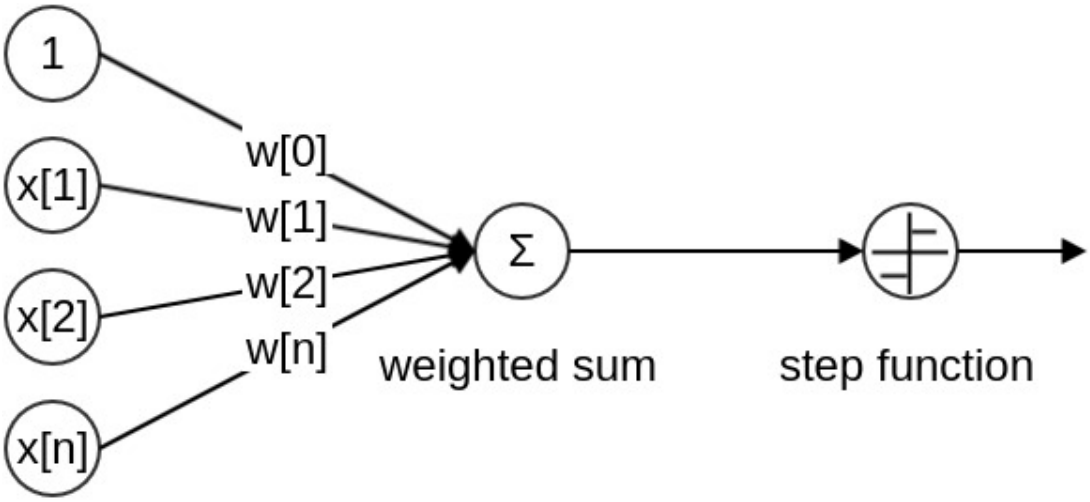

Figure 2.1: A Simple Perceptron

Table 2.1: Truth Table of the Logic AND

x[1]	x[2]	y
0	0	0
0	1	0
1	0	0
1	1	1

We can simply set $w[1] = 0.5$, $w[2] = 0.5$, $b = -0.6$, and choose the step function as the activation function. Thus this perceptron calculates the logic AND function. Check the line 1 in the truth table

$$\begin{aligned} y &= f(\mathbf{w}^T\mathbf{x} + b) \\ &= f(w[1]x[1] + w[2]x[2] + b) \\ &= f(0.5 \times 0 + 0.5 \times 0 - 0.6) \\ &= f(-0.6) = 0 \end{aligned}$$

In fact, any linear classification problem can be solved by a perceptron. But if the data set is not linearly separable, the perceptron will never get to the state with all inputs classified into the correctly, for example, the logic XOR, which is a non-linear function.

2.2 Linear Unit and Gradient Descent

The materials presented in this section follow [27].

2.2.1 Linear Unit

By replacing the **step function** with an **identity function**, a perceptron becomes a simple linear unit. Instead of just two values, 0 or 1, the output of a linear unit can be an arbitrary value, thus it can solve some non-linear regression problems. Fig. 2.2 shows the architecture of a simple linear unit with the activation function to be an identity function.

$$f(x) = x$$

Thus, the output of a linear unit can be calculated

$$y = h(\mathbf{x}) = f(\mathbf{w}^T\mathbf{x} + b) = \mathbf{w}^T\mathbf{x} + b = w[1] \times x[1] + w[2] \times x[2] + \cdots + w[n] \times x[n] + b$$

where function $h(\mathbf{x})$ is called hypothesis, whose parameters consist of $\mathbf{w}$ and b. By setting $b = w[0] \times x[0]$ and $x[0] = 1$, the equation above can be writen as

$$y = h(\mathbf{x}) = f(\mathbf{w}^T\mathbf{x}) = \mathbf{w}^T\mathbf{x}$$

2.2.2 Objective Function

To train a linear unit, we need to minimize the error between the ground truth and the predicted value by the unit. There are a lot of functions to measure the error, among which, the mean square error is the most commonly used.

$$e = \frac{1}{2}(y - \bar{y})^2$$

where y is the actual value while $\bar{y}$ is the predicted valued by the unit.

By assuming n samples in the data set, the summation of the error of all the

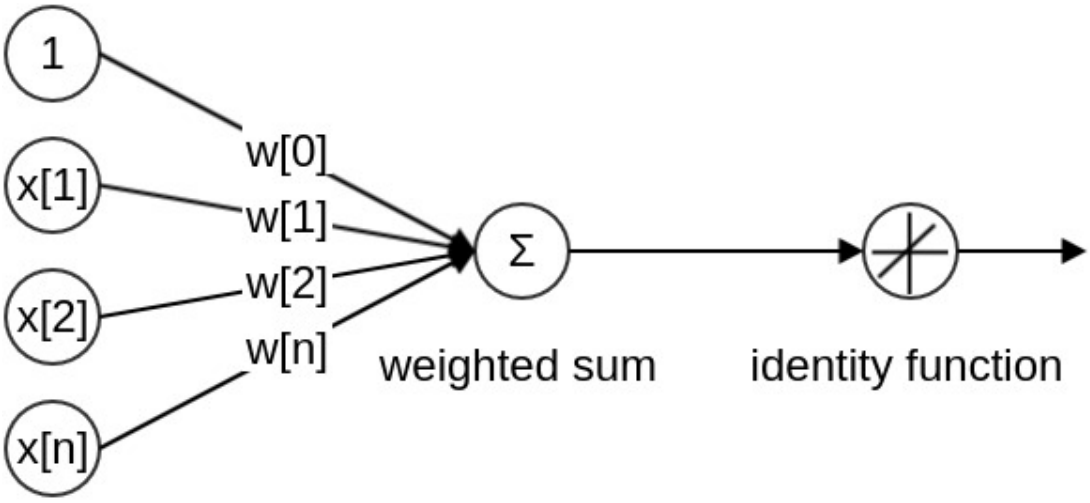

Figure 2.2: A Simple Linear Unit

samples is calculated as the error of the unit, E

$$\begin{aligned} E &= \sum_{i=1}^{n} e^{(i)} \\ &= \frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - \bar{y}^{(i)})^2 \\ &= \frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - h(\mathbf{x}^{(i)}))^2 \\ &= \frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)})^2 \end{aligned}$$

where $\mathbf{x}^{(i)}$, $y^{(i)}$, $\bar{y}^{(i)}$ and $e^{(i)}$ represent the input value, the actual output value, the predicted value and the error of the ith sample in the data set.

Thus it can be seen that the purpose of training a linear unit is to minimize the error E by choosing a proper $\mathbf{w}$, which is an optimization problem in mathematics and the $E(\mathbf{w})$ is called the objective function.

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(y^{(i)} - \mathbf{w}^T\mathbf{x}^{(i)})^2$$

2.2.3 Gradient Descent

Gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or approximate gradient) of the function at the current point. By calculating

$$\mathbf{x}_{new} = \mathbf{x}_{old} - \eta \nabla f(\mathbf{x})$$

one can find the minimum of the function $f(\mathbf{x})$, where η is called the learning rate. For the optimization problem mentioned in the previous subsection, the gradient descent algorithm can be written as

$$\mathbf{w}_{new} = \mathbf{w}_{old} - \eta \nabla E(\mathbf{w})$$

By deriving the equation $\nabla E(\mathbf{w})$, one can get

$$\begin{aligned}
\nabla E(\mathbf{w}) &= \frac{\partial}{\partial \mathbf{w}} E(\mathbf{w}) \\
&= \frac{\partial}{\partial \mathbf{w}} \frac{1}{2} \sum_{i=1}^{n} (y^{(i)} - \bar{y}^{(i)})^2 \\
&= \frac{1}{2} \sum_{i=1}^{n} \frac{\partial}{\partial \mathbf{w}} (y^{(i)} - \bar{y}^{(i)})^2 \\
&= \frac{1}{2} \sum_{i=1}^{n} \frac{\partial (y^{(i)} - \bar{y}^{(i)})^2}{\partial \bar{y}^{(i)}} \frac{\partial \bar{y}^{(i)}}{\partial \mathbf{w}} \\
&= \frac{1}{2} \sum_{i=1}^{n} \frac{\partial}{\partial \bar{y}^{(i)}} ((y^{(i)})^2 - 2y^{(i)}\bar{y}^{(i)} + (\bar{y}^{(i)})^2)^2 \cdot \frac{\partial}{\partial \mathbf{w}} \mathbf{w}^T \mathbf{x}^{(i)} \\
&= \frac{1}{2} \sum_{i=1}^{n} (-2y^{(i)} + 2\bar{y}^{(i)}) \mathbf{x}^{(i)} \\
&= -\sum_{i=1}^{n} (y^{(i)} - \bar{y}^{(i)}) \mathbf{x}^{(i)}
\end{aligned}$$

Thus the iteration step for this particular optimization problem is shown below

$$\mathbf{w}_{new} = \mathbf{w}_{old} + \eta \sum_{i=1}^{n} (y^{(i)} - \bar{y}^{(i)}) \mathbf{x}^{(i)}$$

2.3 Neural Network and Backpropagation

The materials presented in this section follow [27].

2.3.1 Neuron

Essentially, a neuron is the same as a perceptron, but with the activation function replaced by the sigmoid function.

$$f(x) = sigmoid(x) = \frac{1}{1 + e^{-x}}$$

So the output of a neuron is calculated

$$y = sigmoid(\mathbf{w}^T\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}}}$$

The derivative of the sigmoid function is shown below

$$f'(x) = f(x)(1 - f(x))$$

Thus, it is efficient to calculate the derivative of the sigmoid function as soon as one has calculated the value of the function.

2.3.2 Neural Network

An NN is based on several collections of neurons which are connected with each other in a particular way. A neuron that takes the input from other neurons can process it and then send the result to the connected neurons for further use. Fig. 2.3 shows a simple **fully connected neural network**, from which one can figure out the characteristics of a **fully connected neural network** has

- The neurons are connected in layers. The left layer is called the input layer while the right is the output layer, and the layers between them are hidden layers as they are not seen from the outside.
- The neurons in the same layer do not have connections between each other.
- The neurons in one layer are connected to all the neurons in the previous layer.

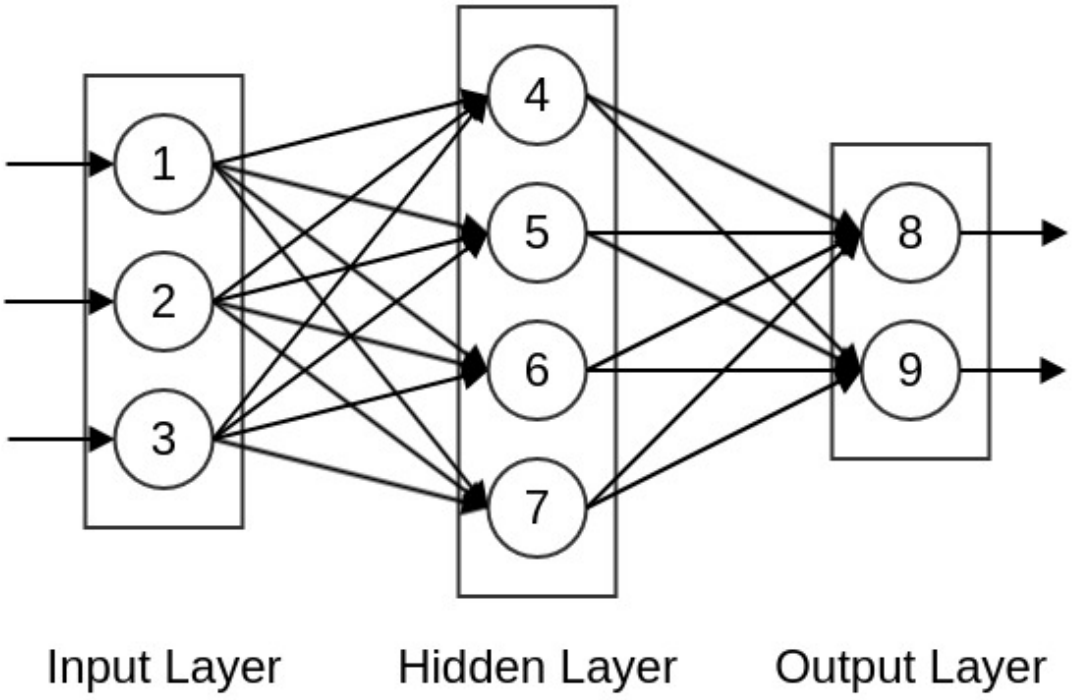

Figure 2.3: A Simple Fully Connected Neural Network

In fact, an NN is a function that maps the input vector $\mathbf{x}$ to the output vector $\mathbf{y}$. To calculate the output of the NN, one needs to assign the input vector to the input layer, and then to calculate the value of each neuron in each layer in turn until all the values are calculated. The values of neurons in the output layer then form the output vector of the network. For example, the output of the fully connected NN in Fig. 2.3 is calculated as follows.

The value of neuron 1, 2, 3 are the input value $x[1]$, $x[2]$, $x[3]$. To calculate the value of neuron 4, a_4

$$\begin{aligned} a_4 &= sigmoid(\mathbf{w}^T\mathbf{x}) \\ &= sigmoid(w_{41}x[1] + w_{42}x[2] + w_{43}x[3] + w_{4b}) \end{aligned}$$

where w_{41}, w_{42} and w_{43} are the weights of the connections between neuron 1, 2, 3 and neuron 4, and w_{4b} is the bias of neuron 4. By keeping doing this calculation, one can get the values of neuron 5, 6 and 7, followed by the value of neuron 8, 9 in the output

layer,

$$\begin{aligned} y[1] &= a_8 \\ &= sigmoid(w_{84}a_4 + w_{85}a_5 + w_{86}a_6 + w_{87}a_7 + w_{8b}) \\ y[2] &= a_9 \\ &= sigmoid(w_{94}a_4 + w_{95}a_5 + w_{96}a_6 + w_{97}a_7 + w_{9b}) \end{aligned}$$

Thus the output of the network $\mathbf{y} = \begin{bmatrix} y[1] \\ y[2] \end{bmatrix}$ is calculated based on the input vector $\mathbf{x} = \begin{bmatrix} x[1] \\ x[2] \\ x[3] \end{bmatrix}$. To be more clear, set

$$\mathbf{x} = \begin{bmatrix} x[1] \\ x[2] \\ x[3] \\ 1 \end{bmatrix}, \mathbf{w}_4 = \begin{bmatrix} w_{41} \\ w_{42} \\ w_{43} \\ w_{4b} \end{bmatrix}, \mathbf{w}_5 = \begin{bmatrix} w_{51} \\ w_{52} \\ w_{53} \\ w_{5b} \end{bmatrix}, \mathbf{w}_6 = \begin{bmatrix} w_{61} \\ w_{62} \\ w_{63} \\ w_{6b} \end{bmatrix}, \mathbf{w}_7 = \begin{bmatrix} w_{71} \\ w_{72} \\ w_{73} \\ w_{7b} \end{bmatrix}$$

Then,

$$\begin{aligned} a_4 &= sigmoid(\mathbf{w}_4^T \mathbf{x}) \\ a_5 &= sigmoid(\mathbf{w}_5^T \mathbf{x}) \\ a_6 &= sigmoid(\mathbf{w}_6^T \mathbf{x}) \\ a_7 &= sigmoid(\mathbf{w}_7^T \mathbf{x}) \end{aligned}$$

By setting

$$\mathbf{a} = \begin{bmatrix} a_4 \\ a_5 \\ a_6 \\ a_7 \end{bmatrix}, \mathbf{W} = \begin{bmatrix} \mathbf{w}_4 & \mathbf{w}_5 & \mathbf{w}_6 & \mathbf{w}_7 \end{bmatrix}^T = \begin{bmatrix} w_{41} & w_{42} & w_{43} & w_{4b} \\ w_{51} & w_{52} & w_{53} & w_{5b} \\ w_{61} & w_{62} & w_{63} & w_{6b} \\ w_{71} & w_{72} & w_{73} & w_{7b} \end{bmatrix}, sigmoid(\begin{bmatrix} z_1 \\ z_2 \\ \vdots \end{bmatrix}) = \begin{bmatrix} sigmoid(z_1) \\ sigmoid(z_2) \\ \vdots \end{bmatrix}$$

Then,

$$\mathbf{a} = sigmoid(\mathbf{W} \cdot \mathbf{x})$$

where $\mathbf{W}$, $\mathbf{x}$ and $\mathbf{a}$ is the weight matrix, the input vector and the output vector of

a certain layer. The above equation demonstrates that the effect of each layer of an NN is to apply a linear transformation to the input vector, followed by an activation function.

The calculation process is the same for each layer, thus to calculate the output of the network, the above equation needs to be calculated repeatedly until the output layer.

2.3.3 Backpropagation

To train an NN is to calculate the value of each weight w_{ij}, but not the way how the neurons connect, the number of layers, the number of neurons in each layer, which are called hyper-parameters that are manually set. As mentioned in the previous section, the objective function needs to be formulated and then a gradient descent optimization algorithm can be applied to find the minimum. By using the mean square error to measure the error between the actual values and the predicted ones, the objective function of NN, E, and the gradient descent equation used are shown below,

$$E = \frac{1}{2} \sum_{i \in outputs} (t_i - a_i)^2$$

$$w_{ji} \leftarrow w_{ji} - \eta \frac{\partial E}{\partial w_{ji}} \tag{2.1}$$

where t_i is the actual value of the ith neuron in the network.

By analysis, one can find the weight w_{ji} will only affect the rest of the network through the input to the neuron j, set s_j to be the weighted summation of neuron j,

$$\begin{aligned} s_j &= \sum_i w_{ji} x_{ji} \\ &= \mathbf{w}_j^T \mathbf{x}_j \end{aligned}$$

where x_{ji} is the value passed from neuron i to neuron j, $\mathbf{x}_j$ is the input vector for neuron j. Thus E is a function of s_j and s_j is a function of w_{ji}, by applying chain

rule one can get

$$\begin{aligned}\frac{\partial E}{\partial w_{ji}} &= \frac{\partial E}{\partial s_j}\frac{\partial s_j}{\partial w_{ji}} \\ &= \frac{\partial E}{\partial s_j}\frac{\partial \sum_i w_{ji}x_{ji}}{\partial w_{ji}} \\ &= \frac{\partial E}{\partial s_j}x_{ji}\end{aligned}$$

By assigning $\delta_j = -\frac{\partial E}{\partial s_j}$, the equation is also written as

$$\frac{\partial E}{\partial w_{ji}} = -\delta_j x_{ji}$$

- For the neuron j in the output layer,

$$\begin{aligned}\frac{\partial E}{\partial s_j} &= \frac{\partial E}{\partial a_j}\frac{\partial a_j}{\partial s_j} \\ &= \frac{\partial}{\partial a_j}(\frac{1}{2}\sum_{i\in outputs}(t_i - a_i)^2)\frac{\partial}{\partial s_j}sigmoid(s_j) \\ &= \frac{\partial}{\partial a_j}(\frac{1}{2}(t_j - a_j)^2)(a_j(1-a_j)) \\ &= -(t_j - a_j)a_j(1-a_j)\end{aligned}$$

  Thus $\delta_j = (t_j - a_j)a_j(1-a_j)$. By plugging it into (2.1), the iteration step writes as

$$\begin{aligned}w_{ji} &\leftarrow w_{ji} - \eta\frac{\partial E}{\partial w_{ji}} \\ &= w_{ji} + \eta(t_j - a_j)a_j(1-a_j)x_{ji} \\ &= w_{ji} + \eta\delta_j x_{ji}\end{aligned}$$

- For the neuron j in the hidden layer,

$$\begin{aligned}
\frac{\partial E}{\partial s_j} &= \sum_{k \in Downstream(j)} \frac{\partial E}{\partial s_k} \frac{\partial s_k}{\partial s_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial s_k}{\partial s_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial s_k}{\partial a_j} \frac{\partial a_j}{\partial s_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k w_{kj} (a_j(1 - a_j)) \\
&= -(a_j(1 - a_j)) \sum_{k \in Downstream(j)} \delta_k w_{kj}
\end{aligned}$$

  where $Downstream(j)$ defines the neurons which are directly connected to the neuron j in the next layer. Thus $\delta_j = (a_j(1 - a_j)) \sum_{k \in Downstream(j)} \delta_k w_{kj}$. By plugging it into (2.1), the iteration step writes as

$$\begin{aligned}
w_{ji} &\leftarrow w_{ji} - \eta \frac{\partial E}{\partial w_{ji}} \\
&= w_{ji} + \eta (a_j(1 - a_j)) \sum_{k \in Downstream(j)} \delta_k w_{kj} x_{ji} \\
&= w_{ji} + \eta \delta_j x_{ji}
\end{aligned}$$

2.4 Recurrent Neural Network and Backpropagation Through Time

2.4.1 Recurrent Neural Network

The materials presented in this subsection follow [6].

RNN is a type of NN that the neurons do not always feedforward, but connect to the neurons in the previous layer or the current layer itself. Thus RNN can use their internal state or memory to process the input so it is suitable for problems that can be formulated into a time sequence, such as natural language process and voice recognition.

The left side of the Fig. 2.4, a simple RNN unit takes a time series of input data x and outputs o. The loop in the RNN unit makes it possible to pass the information

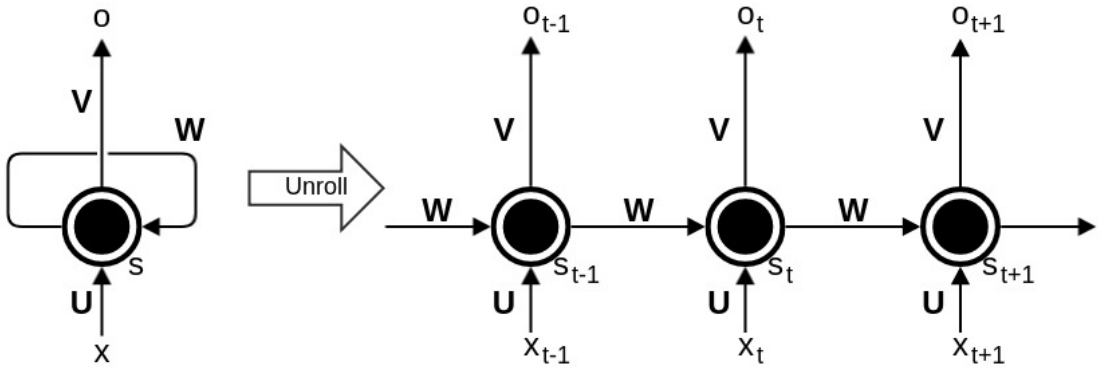

Figure 2.4: A Simple Recurrent Neural Network

from one step to the next. In other words, an RNN unit can be treated as a horizontal expansion of the copies of itself. The right side of the Fig. 2.4 shows the network after the loop is unrolled, where x_t, o_t and s_t denote the input, output and state at time step t.

Thus the mathematical calculation of a general RNN is shown below

$$\mathbf{o}_t = g(\mathbf{V}\mathbf{s}_t) \tag{2.2}$$

$$\mathbf{s}_t = f(\mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{s}_{t-1}) \tag{2.3}$$

where $\mathbf{V}$, $\mathbf{U}$ and $\mathbf{W}$ are the weight matrix for output, input and state transformation, g and f are the activation functions for output and recurrent layer. (2.2) is the formula for the output layer of the network, which is a fully connected layer and (2.3) is the formula for the recurrent layer of the network, which calculates the $\mathbf{s}_t$ based on $\mathbf{x}_t$ and $\mathbf{s}_{t-1}$ with the additional weight matrix $\mathbf{W}$.

2.4.2 Backpropagation Through Time

The materials presented in this subsection follow [32].

Backpropagation Through Time (BPTT) is the algorithm that is used to update the weights in the recurrent layer of the recurrent neural network, the rationale of which is the same as Backpropagation described in the previous subsection. It consists of three steps, each of which is described in the following subsections.

- Calculate the value of each neuron forward.
- Calculate the error of each neuron backward.

- Calculate the gradient for each weight.

Forward for Value

(2.3) is the formula to calculate the value of each neuron forward. Note that by assuming the input $\mathbf{x}$ is an m-dimension vector and the output $\mathbf{o}$ is an n-dimension vector, the dimensions of $\mathbf{U}$ and $\mathbf{W}$ are $n \times m$ and $n \times n$. Unfold the equation to be more clear

$$\begin{bmatrix} s_1^t \\ s_2^t \\ \vdots \\ s_n^t \end{bmatrix} = f(\begin{bmatrix} u_{11} & u_{12} & \dots & u_{1m} \\ u_{21} & u_{22} & \dots & u_{2m} \\ \vdots & & & \\ u_{n1} & u_{n2} & \dots & u_{nm} \end{bmatrix} \begin{bmatrix} x_1^t \\ x_2^t \\ \vdots \\ x_m^t \end{bmatrix} + \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1n} \\ w_{21} & w_{22} & \dots & w_{2n} \\ \vdots & & & \\ w_{n1} & w_{n2} & \dots & w_{nn} \end{bmatrix} \begin{bmatrix} s_1^{t-1} \\ s_2^{t-1} \\ \vdots \\ s_n^{t-1} \end{bmatrix})$$

where the superscript and the subscript of x and s are the time step and the ordinal of the neuron.

Backward for Error

BTPP algorithm propagates the error $\boldsymbol{\delta}_t^l$ in two directions, one into the previous layer $\boldsymbol{\delta}_t^{l-1}$ while the other back to the initial time step $\boldsymbol{\delta}_1^l$.

By setting $\mathbf{net}_t = \mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{s}_{t-1}$, one can get $\mathbf{s}_{t-1} = f(\mathbf{net}_{t-1})$. Thus

$$\begin{aligned} \frac{\partial \mathbf{net}_t}{\partial \mathbf{net}_{t-1}} &= \frac{\partial \mathbf{net}_t}{\partial \mathbf{s}_{t-1}} \frac{\partial \mathbf{s}_{t-1}}{\partial \mathbf{net}_{t-1}} \\ &= \begin{bmatrix} \frac{\partial net_1^t}{\partial s_1^{t-1}} & \frac{\partial net_1^t}{\partial s_2^{t-1}} & \dots & \frac{\partial net_1^t}{\partial s_n^{t-1}} \\ \frac{\partial net_2^t}{\partial s_1^{t-1}} & \frac{\partial net_2^t}{\partial s_2^{t-1}} & \dots & \frac{\partial net_2^t}{\partial s_n^{t-1}} \\ \vdots & \vdots & & \vdots \\ \frac{\partial net_n^t}{\partial s_1^{t-1}} & \frac{\partial net_n^t}{\partial s_2^{t-1}} & \dots & \frac{\partial net_n^t}{\partial s_n^{t-1}} \end{bmatrix} \begin{bmatrix} \frac{\partial s_1^{t-1}}{\partial net_1^{t-1}} & \frac{\partial s_1^{t-1}}{\partial net_n^{t-1}} & \dots & \frac{\partial s_1^{t-1}}{\partial net_n^{t-1}} \\ \frac{\partial s_2^{t-1}}{\partial net_1^{t-1}} & \frac{\partial s_2^{t-1}}{\partial net_1^{t-1}} & \dots & \frac{\partial s_2^{t-1}}{\partial net_n^{t-1}} \\ \vdots & \vdots & & \vdots \\ \frac{\partial s_n^{t-1}}{\partial net_1^{t-1}} & \frac{\partial s_n^{t-1}}{\partial net_1^{t-1}} & \dots & \frac{\partial s_n^{t-1}}{\partial net_n^{t-1}} \end{bmatrix} \\ &= \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1n} \\ w_{21} & w_{22} & \dots & w_{2n} \\ \vdots & & & \\ w_{n1} & w_{n2} & \dots & w_{nn} \end{bmatrix} \begin{bmatrix} f'(net_1^{t-1}) & 0 & \dots & 0 \\ 0 & f'(net_2^{t-1}) & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & f'(net_n^{t-1}) \end{bmatrix} \\ &= \mathbf{W} diag[f'(\mathbf{net}_{t-1})] \end{aligned}$$

Then the equation for the propagation into the initial time step is

$$\begin{aligned}
\boldsymbol{\delta}_k^T &= \frac{\partial E}{\partial \mathbf{net}_k} \\
&= \frac{\partial E}{\partial \mathbf{net}_t}\frac{\partial \mathbf{net}_t}{\partial \mathbf{net}_k} \\
&= \frac{\partial E}{\partial \mathbf{net}_t}\frac{\partial \mathbf{net}_t}{\partial \mathbf{net}_{t-1}}\cdots\frac{\partial \mathbf{net}_{k+1}}{\partial \mathbf{net}_k} \\
&= \boldsymbol{\delta}_t^T \prod_{i=k}^{t-1} \mathbf{W} diag[f'(\mathbf{net}_i)]
\end{aligned}$$

By setting $\mathbf{a}_t^{l-1}$ as the output of the previous layer, one can get $\mathbf{net}_t^l = \mathbf{U}\mathbf{a}_t^{l-1} + \mathbf{W}\mathbf{s}_{t-1}$. Thus the equation for the propagation into the previous layer is

$$\begin{aligned}
(\boldsymbol{\delta}_t^{l-1})^T &= \frac{\partial E}{\partial \mathbf{net}_t^{l-1}} \\
&= \frac{\partial E}{\partial \mathbf{net}_t^{l}}\frac{\partial \mathbf{net}_t^{l}}{\partial \mathbf{net}_t^{l-1}} \\
&= \frac{\partial E}{\partial \mathbf{net}_t^{l}}\frac{\partial \mathbf{net}_t^{l}}{\partial \mathbf{a}_t^{l-1}}\frac{\partial \mathbf{a}_t^{l-1}}{\partial \mathbf{net}_t^{l-1}} \\
&= (\boldsymbol{\delta}_t^{l})^T \mathbf{U}[f'^{l-1}(\mathbf{net}_t^{l-1})]
\end{aligned}$$

where f^{l-1} is the activation function of the previous layer.

Gradient

The calculation of gradient for W and U is separated, but similar.

$$\begin{aligned}
\frac{\partial E}{\partial w_{ji}} &= \sum_{k=1}^{t}\frac{\partial E_k}{\partial w_{ji}} \\
&= \sum_{k=1}^{t}\frac{\partial E_k}{\partial net_j^k}\frac{\partial net_j^k}{\partial w_{ji}} \\
&= \sum_{k=1}^{t}\delta_j^k s_i^{k-1}
\end{aligned}$$

$$\frac{\partial E}{\partial u_{ji}} = \sum_{k=1}^{t} \frac{\partial E_k}{\partial u_{ji}}$$
$$= \sum_{k=1}^{t} \frac{\partial E_k}{\partial net_j^k} \frac{\partial net_j^k}{\partial u_{ji}}$$
$$= \sum_{k=1}^{t} \delta_j^k a_i^k$$

2.4.3 Gradient Vanishing and Explosion

The materials presented in this subsection follow [32].

Unfortunately, RNN cannot achieve good performance for long sequences. One main reason is gradient vanishing and explosion in training, which leads to the fact that the gradient cannot propagate long enough. Check the formula below

$$\begin{aligned} \boldsymbol{\delta}_k^T &= \boldsymbol{\delta}_t^T \prod_{i=k}^{t-1} \mathbf{W} diag[f'(\mathbf{net}_i)] \\ ||\boldsymbol{\delta}_k^T|| &\leqslant ||\boldsymbol{\delta}_t^T|| \prod_{i=k}^{t-1} ||\mathbf{W}|| \, ||diag[f'(\mathbf{net}_i)]|| \\ &\leqslant ||\boldsymbol{\delta}_t^T|| (\beta^W \beta^f)^{t-k} \end{aligned}$$

where β defines the upper bound of the modulus of the matrix. Thus if $t-k$ is big enough, the error δ_k^t will increase or decrease very quickly, depending on the value of β bigger or smaller than 1, which causes the vanishing and explosion problem.

2.5 Long Short Term Memory Networks

The materials presented in this section follow [30, 26].

To solve the gradient vanishing and explosion problem in general RNN, long short term memory (LSTM) networks, a special kind of RNN which are capable of learning long-term dependencies, will be introduced in the section. LSTMs are explicitly designed to avoid the long-term dependency problem and remembering information for long periods of time is practically their default behaviour.

The key to LSTMs is the cell state, $\mathbf{c}_t$ in the Fig. 2.5, to which the LSTM does have the ability to remove or add information, carefully regulated by structures called gates. Gates are a way to optionally let information through. They are composed

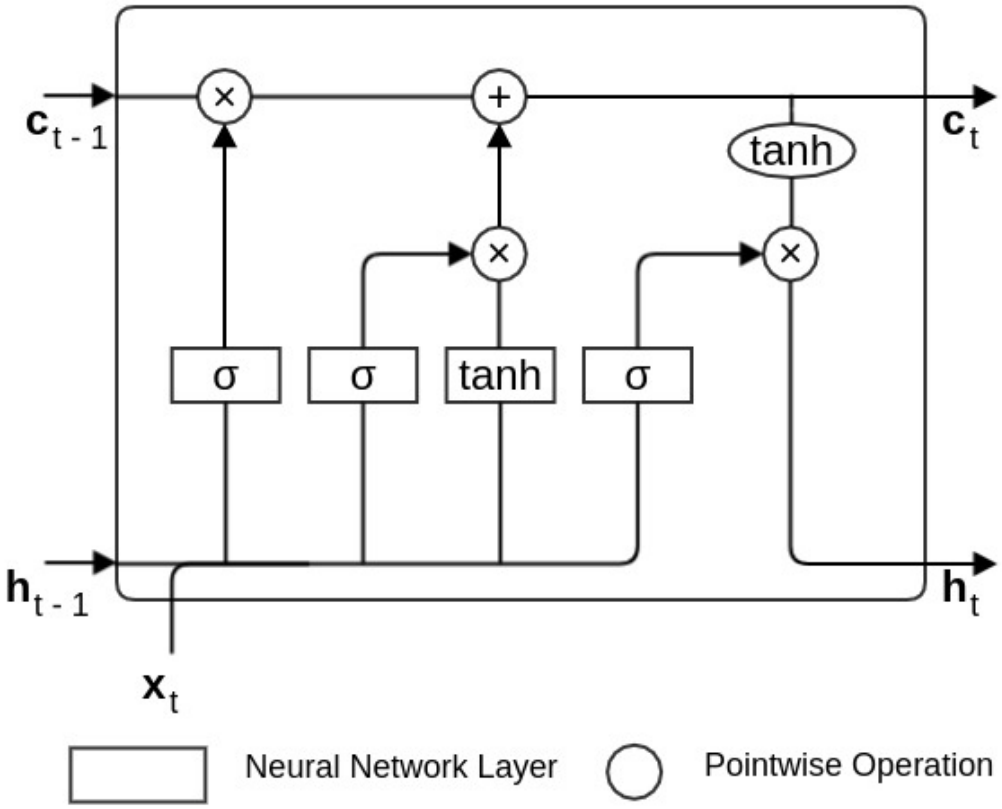

Figure 2.5: The Structure of a LSTM Cell

out of a sigmoid neural net layer and a pointwise multiplication operation. An LSTM has three of these gates, a forget gate, an input gate and an output gate, to protect and control the cell state.

2.5.1 Forward for Value

The first step in LSTM is to decide what information is going to be thrown away from the cell state. This decision is made by a sigmoid layer called the forget gate layer. It looks at $\mathbf{h}_{t-1}$ and $\mathbf{x}_t$ and outputs a number between 0 and 1 for each number in the cell state $\mathbf{c}_{t1}$.

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \cdot \Big[\mathbf{h}_{t-1}\,,\, \mathbf{x}_t\Big])$$

where $\Big[\mathbf{a}\,,\, \mathbf{b}\Big]$ means to concatenate two vectors into one vector.

The next step is to decide what new information is going to be stored in the cell state. This has two parts, a sigmoid layer called the input gate layer decides which

values well update and a *tanh* layer creates a vector of new candidate values, $\tilde{\mathbf{c}}_t$, that could be added to the state. In the next step, these two are combined to create an update to the state.

$$\begin{aligned} \mathbf{i}_t &= \sigma(\mathbf{W}_i \cdot \Big[\mathbf{h}_{t-1}\,,\, \mathbf{x}_t\Big]) && (2.4)\\ \tilde{\mathbf{c}}_t &= tanh(\mathbf{W}_c \cdot \Big[\mathbf{h}_{t-1}\,,\, \mathbf{x}_t\Big]) && (2.5)\\ \mathbf{c}_t &= \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \tilde{\mathbf{c}}_t && (2.6) \end{aligned}$$

where $\circ$ means the pointwise multiplication.

Finally, the output will be based on the cell state but will be a filtered version. A sigmoid layer which decides what parts of the cell state are going to be output is run first and then we put the cell state through *tanh* and multiply it by the output of the sigmoid gate so that we only output the parts we decided to.

$$\begin{aligned} \mathbf{o}_t &= \sigma(\mathbf{W}_o \cdot \Big[\mathbf{h}_{t-1}\,,\, \mathbf{x}_t\Big]) && (2.7)\\ \mathbf{h}_t &= \mathbf{o}_t \circ tanh(\mathbf{c}_t) && (2.8) \end{aligned}$$

2.5.2 Backward for Error

From (2.6), one can get

$$\begin{aligned} \frac{\partial \mathbf{c}_t}{\partial \mathbf{f}_t} &= diag(\mathbf{c}_{t-1})\\ \frac{\partial \mathbf{c}_t}{\partial \mathbf{i}_t} &= diag(\tilde{\mathbf{c}}_t)\\ \frac{\partial \mathbf{c}_t}{\partial \tilde{\mathbf{c}}_t} &= diag(\mathbf{i}_t) \end{aligned}$$

From (2.8), one can get

$$\begin{aligned} \frac{\partial \mathbf{h}_t}{\partial \mathbf{o}_t} &= diag[tanh(\mathbf{c}_t)]\\ \frac{\partial \mathbf{h}_t}{\partial \mathbf{c}_t} &= diag[\mathbf{o}_t \circ (1 - tanh(\mathbf{c}_t)^2)] \end{aligned}$$

Also, the following variables are defined

$$
\begin{aligned}
\mathbf{net}_{f,t} &= \mathbf{W}_f \left[\mathbf{h}_{t-1}\,, \mathbf{x}_t\right] \\
&= \mathbf{W}_{fh}\mathbf{h}_{t-1} + \mathbf{W}_{fx}\mathbf{x}_t \\
\mathbf{net}_{i,t} &= \mathbf{W}_i \left[\mathbf{h}_{t-1}\,, \mathbf{x}_t\right] \\
&= \mathbf{W}_{ih}\mathbf{h}_{t-1} + \mathbf{W}_{ix}\mathbf{x}_t \\
\mathbf{net}_{\tilde{c},t} &= \mathbf{W}_c \left[\mathbf{h}_{t-1}\,, \mathbf{x}_t\right] \\
&= \mathbf{W}_{ch}\mathbf{h}_{t-1} + \mathbf{W}_{cx}\mathbf{x}_t \\
\mathbf{net}_{o,t} &= \mathbf{W}_o \left[\mathbf{h}_{t-1}\,, \mathbf{x}_t\right] \\
&= \mathbf{W}_{oh}\mathbf{h}_{t-1} + \mathbf{W}_{ox}\mathbf{x}_t \\
\boldsymbol{\delta}_{f,t} &= \frac{\partial E}{\partial \mathbf{net}_{f,t}} \\
\boldsymbol{\delta}_{i,t} &= \frac{\partial E}{\partial \mathbf{net}_{i,t}} \\
\boldsymbol{\delta}_{\tilde{c},t} &= \frac{\partial E}{\partial \mathbf{net}_{\tilde{c},t}} \\
\boldsymbol{\delta}_{o,t} &= \frac{\partial E}{\partial \mathbf{net}_{o,t}}
\end{aligned}
$$

Thus,

$$
\begin{aligned}
\frac{\partial \mathbf{f}_t}{\partial \mathbf{net}_{f,t}} &= diag[\mathbf{f}_t \circ (1 - \mathbf{f}_t)] \\
\frac{\partial \mathbf{net}_{f,t}}{\partial \mathbf{h}_{t-1}} &= \mathbf{W}_{fh} \\
\frac{\partial \mathbf{i}_t}{\partial \mathbf{net}_{i,t}} &= diag[\mathbf{i}_t \circ (1 - \mathbf{i}_t)] \\
\frac{\partial \mathbf{net}_{i,t}}{\partial \mathbf{h}_{t-1}} &= \mathbf{W}_{ih} \\
\frac{\partial \tilde{\mathbf{c}}_t}{\partial \mathbf{net}_{\tilde{c},t}} &= diag[1 - \tilde{\mathbf{c}}_t^2] \\
\frac{\partial \mathbf{net}_{\tilde{c},t}}{\partial \mathbf{h}_{t-1}} &= \mathbf{W}_{ch} \\
\frac{\partial \mathbf{o}_t}{\partial \mathbf{net}_{o,t}} &= diag[\mathbf{o}_t \circ (1 - \mathbf{o}_t)] \\
\frac{\partial \mathbf{net}_{o,t}}{\partial \mathbf{h}_{t-1}} &= \mathbf{W}_{oh}
\end{aligned}
$$

Then the equation for the propagation into the previous time step is

$$\begin{aligned}
\boldsymbol{\delta}_{t-1}^T &= \frac{\partial E}{\partial \mathbf{h}_{t-1}} \\
&= \frac{\partial E}{\partial \mathbf{h}_t}\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \\
&= \boldsymbol{\delta}_t^T \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \\
&= \boldsymbol{\delta}_t^T \frac{\partial \mathbf{h}_t}{\partial \mathbf{c}_t}\frac{\partial \mathbf{c}_t}{\partial \mathbf{f}_t}\frac{\partial \mathbf{f}_t}{\partial \mathbf{net}_{f,t}}\frac{\partial \mathbf{net}_{f,t}}{\partial \mathbf{h}_{t-1}} + \boldsymbol{\delta}_t^T \frac{\partial \mathbf{h}_t}{\partial \mathbf{c}_t}\frac{\partial \mathbf{c}_t}{\partial \mathbf{i}_t}\frac{\partial \mathbf{i}_t}{\partial \mathbf{net}_{i,t}}\frac{\partial \mathbf{net}_{i,t}}{\partial \mathbf{h}_{t-1}} \\
&\quad + \boldsymbol{\delta}_t^T \frac{\partial \mathbf{h}_t}{\partial \mathbf{c}_t}\frac{\partial \mathbf{c}_t}{\partial \tilde{\mathbf{c}}_t}\frac{\partial \tilde{\mathbf{c}}_t}{\partial \mathbf{net}_{\tilde{c},t}}\frac{\partial \mathbf{net}_{\tilde{c},t}}{\partial \mathbf{h}_{t-1}} + \boldsymbol{\delta}_t^T \frac{\partial \mathbf{h}_t}{\partial \mathbf{o}_t}\frac{\partial \mathbf{o}_t}{\partial \mathbf{net}_{o,t}}\frac{\partial \mathbf{net}_{o,t}}{\partial \mathbf{h}_{t-1}} \\
&= \boldsymbol{\delta}_t^T \circ [\mathbf{o}_t \circ (1 - tanh(\mathbf{c}_t)^2)] \circ [\mathbf{c}_{t-1}] \circ [\mathbf{f}_t \circ (1 - \mathbf{f}_t)]\mathbf{W}_{fh} \\
&\quad + \boldsymbol{\delta}_t^T \circ [\mathbf{o}_t \circ (1 - tanh(\mathbf{c}_t)^2)] \circ [\tilde{\mathbf{c}}_t] \circ [\mathbf{i}_t \circ (1 - \mathbf{i}_t)]\mathbf{W}_{ih} \\
&\quad + \boldsymbol{\delta}_t^T \circ [\mathbf{o}_t \circ (1 - tanh(\mathbf{c}_t)^2)] \circ [\mathbf{i}_t] \circ [(1 - \tilde{\mathbf{c}}_t)^2]\mathbf{W}_{ch} \\
&\quad + \boldsymbol{\delta}_t^T \circ [tanh(\mathbf{c}_t)] \circ [\mathbf{o}_t \circ (1 - \mathbf{o}_t)]\mathbf{W}_{oh} \\
&= \boldsymbol{\delta}_{f,t}^T \mathbf{W}_{fh} + \boldsymbol{\delta}_{i,t}^T \mathbf{W}_{ih} + \boldsymbol{\delta}_{\mathbf{c},t}^T \mathbf{W}_{ch} + \boldsymbol{\delta}_{o,t}^T \mathbf{W}_{oh}
\end{aligned}$$

By defining the error of the previous layer $\boldsymbol{\delta}_t^{l-1} = \frac{\partial E}{\partial \mathbf{net}_t^{l-1}}$ and the input of the current LSTM layer $\mathbf{x}_t^l = f^{l-1}(\mathbf{net}_t^{l-1})$, where f^{l-1} is the activation function of the previous layer. Thus the equation for the propagation into the previous layer is

$$\begin{aligned}
\frac{\partial E}{\partial \mathbf{net}_t^{l-1}} &= \frac{\partial E}{\partial \mathbf{net}_{f,t}^l}\frac{\partial \mathbf{net}_{f,t}^l}{\partial \mathbf{x}_t^l}\frac{\partial \mathbf{x}_t^l}{\mathbf{net}_t^{l-1}} + \frac{\partial E}{\partial \mathbf{net}_{i,t}^l}\frac{\partial \mathbf{net}_{i,t}^l}{\partial \mathbf{x}_t^l}\frac{\partial \mathbf{x}_t^l}{\mathbf{net}_t^{l-1}} \\
&\quad + \frac{\partial E}{\partial \mathbf{net}_{\tilde{c},t}^l}\frac{\partial \mathbf{net}_{\tilde{c},t}^l}{\partial \mathbf{x}_t^l}\frac{\partial \mathbf{x}_t^l}{\mathbf{net}_t^{l-1}} + \frac{\partial E}{\partial \mathbf{net}_{o,t}^l}\frac{\partial \mathbf{net}_{o,t}^l}{\partial \mathbf{x}_t^l}\frac{\partial \mathbf{x}_t^l}{\mathbf{net}_t^{l-1}} \\
&= \boldsymbol{\delta}_{f,t}^T \mathbf{W}_{fx} \circ f'^{l-1}(\mathbf{net}_t^{l-1}) + \boldsymbol{\delta}_{i,t}^T \mathbf{W}_{ix} \circ f'^{l-1}(\mathbf{net}_t^{l-1}) \\
&\quad + \boldsymbol{\delta}_{\tilde{c},t}^T \mathbf{W}_{cx} \circ f'^{l-1}(\mathbf{net}_t^{l-1}) + \boldsymbol{\delta}_{o,t}^T \mathbf{W}_{ox} \circ f'^{l-1}(\mathbf{net}_t^{l-1})
\end{aligned}$$

2.5.3 Gradient

The calculation of gradient for $\mathbf{W}_{fh}$, $\mathbf{W}_{ih}$, $\mathbf{W}_{ch}$, $\mathbf{W}_{oh}$, $\mathbf{W}_{fx}$, $\mathbf{W}_{ix}$, $\mathbf{W}_{cx}$ and $\mathbf{W}_{ox}$ is separated, but similar.

$$
\begin{aligned}
\frac{\partial E}{\partial \mathbf{W}_{fh}} &= \sum_{j=1}^{t} \frac{\partial E_j}{\partial \mathbf{W}_{fh}} \\
&= \sum_{j=1}^{t} \frac{\partial E_j}{\partial \mathbf{net}_{f,j}} \frac{\partial \mathbf{net}_{f,j}}{\partial \mathbf{W}_{fh}} \\
&= \sum_{j=1}^{t} \boldsymbol{\delta}_{f,j} \mathbf{h}_{j-1}^{T} \\
\frac{\partial E}{\partial \mathbf{W}_{ih}} &= \sum_{j=1}^{t} \boldsymbol{\delta}_{i,j} \mathbf{h}_{j-1}^{T} \\
\frac{\partial E}{\partial \mathbf{W}_{ch}} &= \sum_{j=1}^{t} \boldsymbol{\delta}_{\tilde{c},j} \mathbf{h}_{j-1}^{T} \\
\frac{\partial E}{\partial \mathbf{W}_{oh}} &= \sum_{j=1}^{t} \boldsymbol{\delta}_{o,j} \mathbf{h}_{j-1}^{T} \\
\frac{\partial E}{\partial \mathbf{W}_{fx}} &= \frac{\partial E}{\partial \mathbf{net}_{f,t}} \frac{\partial \mathbf{net}_{f,t}}{\partial \mathbf{W}_{fx}} \\
&= \boldsymbol{\delta}_{f,t} \mathbf{x}_{t}^{T} \\
\frac{\partial E}{\partial \mathbf{W}_{ix}} &= \boldsymbol{\delta}_{i,t} \mathbf{x}_{t}^{T} \\
\frac{\partial E}{\partial \mathbf{W}_{cx}} &= \boldsymbol{\delta}_{\tilde{c},t} \mathbf{x}_{t}^{T} \\
\frac{\partial E}{\partial \mathbf{W}_{ox}} &= \boldsymbol{\delta}_{o,t} \mathbf{x}_{t}^{T}
\end{aligned}
$$

Chapter 3

Deep Neural Network in Indoor Positioning System with a Two-Step Scheme

3.1 Introduction

The LBSs have been driven so fast by the rapid proliferation of wireless communication and mobile devices, which raises a massive demand for high accuracy of localization. For the outdoor open environment, customers can use the GNSS, such as Global Positioning System (GPS) and BeiDou Navigation System, to obtain a highly accurate location estimation. However, the GNSS signals from satellites cannot be seen in many indoor areas, which limits their applications in indoor localization. Therefore, people turn to different sensor-based systems, such as Bluetooth [1, 35], RFID [20] and ultrasound [33], to solve the localization problem in the indoor environment. Among all these available solutions, WiFi-based IPS becomes one of the promising approaches because of the popularity of wireless local area network (WLAN) infrastructure and the fast development of mobile devices, which makes the system low cost and easy to deploy.

In general, WiFi IPSs can be classified into two dominant classes, ranging based and fingerprinting based. Ranging based ones derive the distances between the receiver and different transmitters based on propagation models, using measurements such as time of flight, received signal strength and angle of arrival [22], and then estimate the location based on the distances obtained in the first step by triangula-

tion. However, due to the multi-path phenomenon, the accurate propagation model is difficult to formulate, leading to the inaccuracy of the distance prediction. The fingerprinting based methods, which associate a group of physical measurements at each RP as a fingerprint, perform like pattern matching systems, comparing the similarity between the target fingerprint and those in the database to return the best match to be the estimation. Thus a fingerprinting system consists of two phases, an offline phase, collecting fingerprints at different RPs within the surveillance area and storing them into a database, and an online phase, comparing the target fingerprint with those in the database to return the best match as the prediction based on the desired pattern matching algorithm. Received signal strength indicator (RSSI)-based WiFi fingerprinting IPSs have been intensively studied in the past decade as the RSSI value is available in every 802.11 interface. There are some other approaches using different physical measurements such as CSI [38, 7] as the fingerprints, but they need special WiFi devices and increase the deployment and system cost.

When it comes to the online phase, the IPS needs to calculate the location of the unknown node, which is usually adopted by experts systems, such as KNN [4, 29, 39, 42], SVM [34, 21], filter-based [5, 3] and NN based algorithms. KNN calculates the distance between the target fingerprint and those in the database to get a set of nearest neighbours and return the mean as the final prediction. To calculate the distance, researchers use different metrics, such as Euclidean distance [4], Bhattacharyya distance [29], Spearman distance [39] and so on. Zou et al. proposed a weighted KNN to improve the accuracy by returning a weighted average instead of the mean [42]. SVM, a machine learning algorithm which is simpler than a multi-layer neural network, builds a model that maps the fingerprints into a high dimensional space and then find a hyperplane that differentiates the classes based on all the training points. Principal component analysis (PCA) and Kernel SVM [21] are used to reduce the high dimensional measurements. Recently, some filter based algorithms are designed to improve the accuracy by taking the previous prediction into account. For example, Kalman filter [5, 3] is used to calculate the most likely location assuming a Gaussian noise and linear motion dynamics. In contrast, NN based algorithms build up a neural network that predicts the location from the target input by defining a particular architecture using different activation functions.

In this chapter, we propose a new NN-based IPS that contains multiple NNs including one classification network and several localization networks, to reduce the training complexity and improve the predicting accuracy. The proposed system uti-

lizes the similarity in RSSI readings within a specific region to distribute the target fingerprint into a cluster that it belongs to by the classification network and then applies the corresponding localization network to evaluate a final prediction. There are different approaches to reduce the workload of building up the initial database for offline training, but our RSSI fingerprint database is collected in our lab by a self-developed 3-wheel robot, which is shown in Fig. 3.1. It has multiple sensors including wheel odometer, an inertial measurement unit (IMU), a LIDAR, sonar sensors and a colour and depth (RGB-D) camera. The robot can navigate to a target location to collect WiFi fingerprints automatically. Therefore, the time consumption for building up the fingerprint database is significantly reduced.

The rest of the chapter is organized as follows. Section 3.2 introduces the related work on NN in IPS, followed by the detail model in Section 3.3. Section 3.4 compares the result with other approaches, and the conclusion of the proposed work is given in Section 3.5 for this chapter.

3.2 Related Work

3.2.1 Fingerprinting Technique

The fingerprinting-based IPS works like a pattern matching system, whose primary design can be divided into two parts, an offline and an online phase. In the offline stage, fingerprints at different RPs within the surveillance area need to be collected and stored in a database for the next step, which is also called a site survey. During the online stage, by comparing a target fingerprint at an unknown location with those in the database based on a well-designed algorithm, the system then returns the best match as the current prediction.

While most of the fingerprinting systems are relying on WiFi RSSI, some use Bluetooth [1, 35], RFID [20], and ultrasound [33] devices RSSI and some others use CSI [38, 7] as the fingerprint. Although these systems provide suitable accuracy, they often focus on objective tracking and need corresponding hardware other than the WiFi devices.

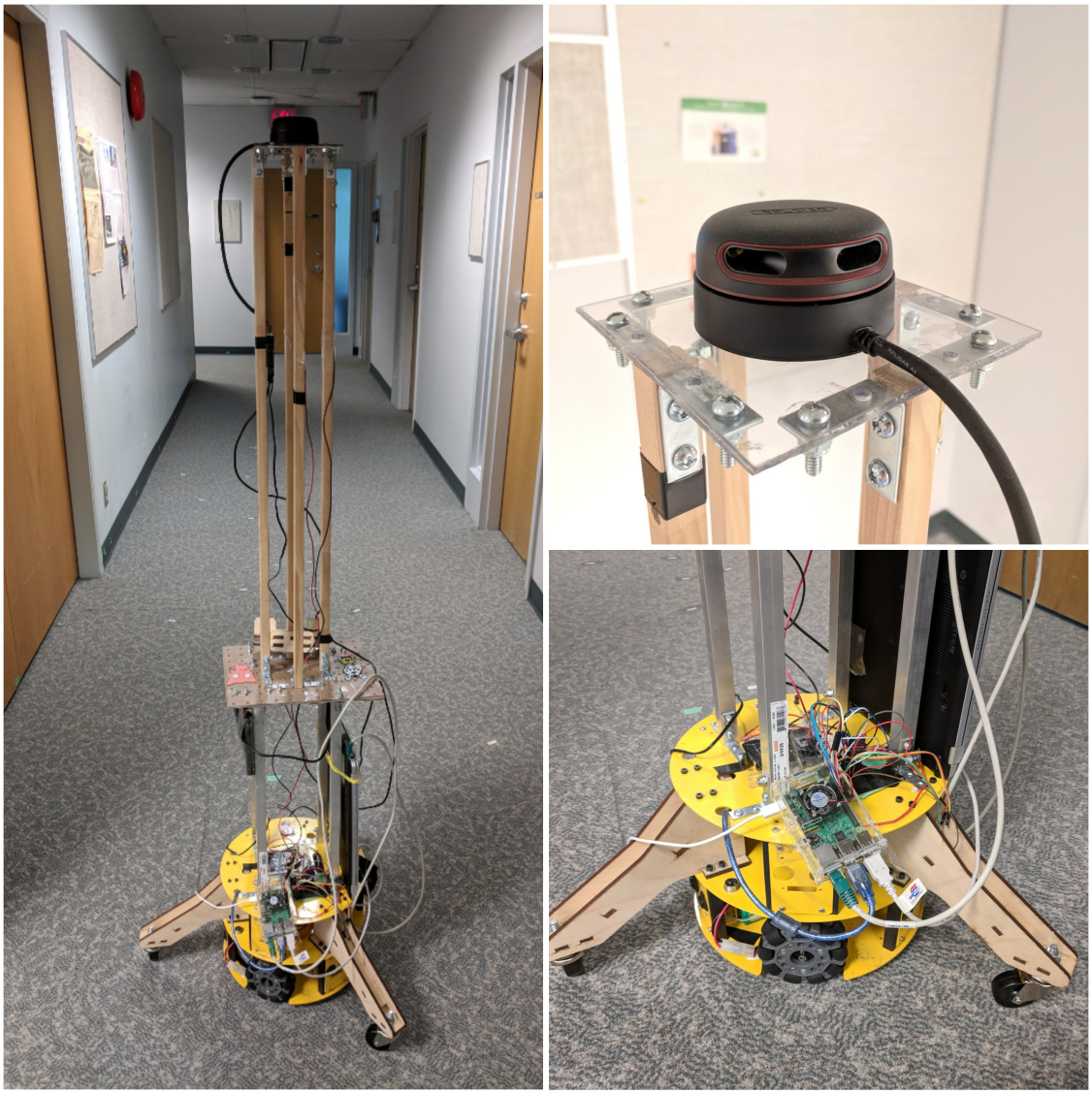

Figure 3.1: The three-wheel robot developed by my colleagues

3.2.2 NN based IPSs

Depending on the output type of the network, existing NN based IPSs can be grouped into two categories, classification and regression. The classification type outputs the predicted label of the unknown location while the regression type directly returns the exact coordinates.

In the literature, multilayer perceptron (MLP) or feed forward neural network is the most frequently used NN for IPS. Fang et al. presented a discriminant-adaptive neural network (DANN), which is implemented by a 3-layer MLP plus multiple dis-

criminant analysis (MDA), outputting the predicted coordinate directly [14]. A real experiment result showed that DANN is more accurate than KNN, maximum likelihood (ML) and simple MLP, with a mean error of smaller than 2 m. In [19], Dai et al. employed an MLP based IPS classifier with a boosting training method, which achieved higher accuracy compared with ML and generalized regression neural network (GRNN) in the experiments.

By using the CSI as fingerprints, Chen et al. developed ConFi, the first system based on CNN, which takes the CSI as input images and is trained to solve a location classification problem and has a mean localization error of 1.36 m and the standard deviation is 0.90 m for the configured experiment [7].

Lukito [24] et al. implemented an RNN model which has two layers of Elman Simple RNN and takes the RSSI readings directly as input and outputs the location labels using Tensorflow. The classification accuracy of this system is 82.47% which is better compared to multi-layer perceptron, Nave Bayes, J48, and SVM, but the network still needs to be tweaked to surpass KNN.

3.2.3 Clustering based IPSs

To reduce the computational complexity and improve the positioning time as well as the positioning accuracy, many researchers came up with the idea of clustering. By utilizing nearest neighbor rule and extreme learning machine (ELM), Xiao et al. proposed a novel clustering base IPS for large scale area [37]. The system applies clustering based on nearest neighbor rules and localization by ELM, giving an approximately 1 m better mean accuracy than that without clustering with the same time complexity. In 2015, Chen et al. proposed a clustering approach: AP similarity clustering and K-weighted nearest node (KWNN) method, which divides the database into clusters based on different APs' similarity [9]. In the online test phase, the system finds out the suitable sub-cluster first and then selects k nodes among the cluster and return the k-weighted prediction, improving the accuracy by 17.14% and reducing the time-consuming by 50% with an average error of 0.77 m compared to k-means+KWNN and KWNN-only.

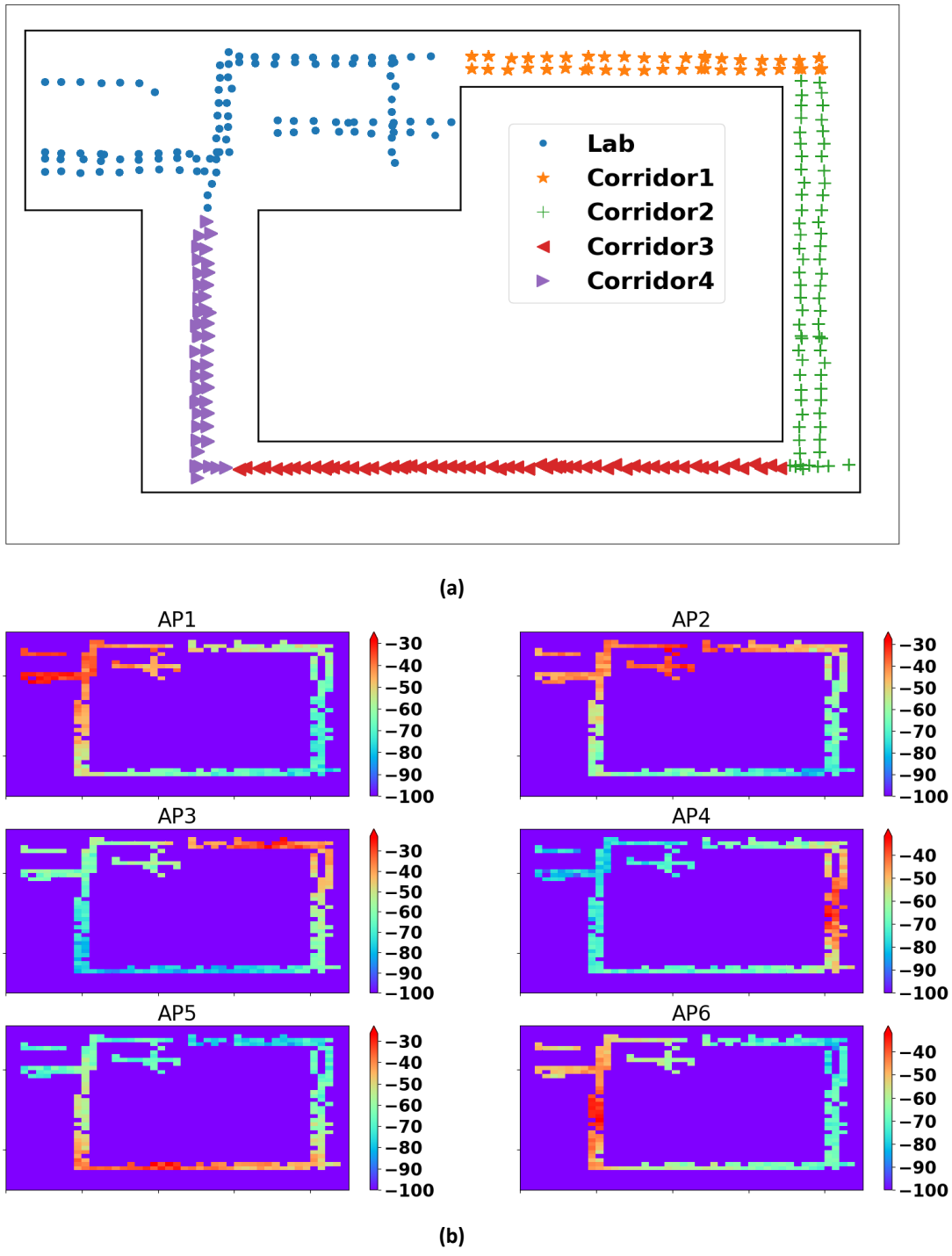

Figure 3.2: (a) Floor map of surveillance area which could be divided into 5 clusters. (b) Heat map of the RSSI strength from 6 APs used in our localization scheme.

3.3 System Model

In this chapter, the notations used are explained as follows. The total number of APs is P while the total number of RSSI readings per scan is N (usually $N \geqslant P$ as each AP may have more than one frequncy band provided), and there are M RPs in the surveillance area. For the ith RP corresponding to the physical coordinate $\boldsymbol{l}_i(x_i, y_i)$ there are S_i fingerprints scanned over a small period of time, jth of which is described as $\boldsymbol{f}^{i,j} = \{F_1^{i,j}, F_2^{i,j}, ..., F_N^{i,j}\}$. For example $F_k^{i,j}$ is the kth RSSI reading of the jth fingerprint at RP i, where $1 \leqslant j \leqslant S_i$, $1 \leqslant k \leqslant N$. Fig. 3.2(a) illustrates our localization floor map with 6 APs, 11 RSSI readings per scan, 332 RPs. Fig. 3.2(b) shows the RSSI heat map of 6 APs, where signal strength is represented by color. Clearly, the signals from 6 APs already cover the whole targeting area including 1 room and 4 corridors.

3.3.1 Database building

The fingerprinting database needs to be built for the network training and testing after the robot finishes collecting RSSI readings. Most work in the literature directly use each RSSI reading and location coordinate or location label pair, $(\boldsymbol{f}^{i,j}, x_i, y_i)$ or $(\boldsymbol{f}^{i,j}, \boldsymbol{l}_i)$, as one database entry. Here we propose a new type of fingerprint, by taking advantage of the similarity in RSSI readings of the nearby locations.

Fig. 3.2(b) indicates that the RSSI readings do not change significantly between two adjacent RPs, based on which we propose to use the combination of these two sets of readings as a new fingerprint to enhance this unique pattern. Consider the two adjacent RPs, $\boldsymbol{l}_{i_1}(x_{i_1}, y_{i_1})$ with S_{i_1} RSSI readings and $\boldsymbol{l}_{i_2}(x_{i_2}, y_{i_2})$ with S_{i_2} RSSI readings, respectively within a threshold distance d, within which we assume the RSSI readings do not have substantial difference in numbers and of which the desired application could bear the error. Instead of treating $(\boldsymbol{f}^{i_1,j_1}, \boldsymbol{l}_{i_1})$ and $(\boldsymbol{f}^{i_2,j_2}, \boldsymbol{l}_{i_2})$ as database entries with a total number of $S_{i_1} + S_{i_2}$ scans, we propose to use the combination $(\boldsymbol{f}^{i_1,j_1}, \boldsymbol{f}^{i_2,j_2}, \boldsymbol{l}_{i_1})$ and $(\boldsymbol{f}^{i_2,j_2}, \boldsymbol{f}^{i_1,j_1}, \boldsymbol{l}_{i_2})$ as database entries with a total number of $2 \times S_{i_1} \times S_{i_2}$, where $1 \leqslant j_1 \leqslant S_{i_1}$ and $1 \leqslant j_2 \leqslant S_{i_2}$.

This representation of the fingerprint combination also assumes that the RSSI readings of each AP fluctuate independently, making the whole training space larger to increase possible cases that the network could see.

In the testing phase, the collected RSSI readings are repeated twice as the target fingerprint.

3.3.2 Clustering

From the Fig. 3.2(b), we see clearly that the RSSI readings are similar within a specific physical area but different from area to area for each existing WiFi infrastructure. To utilize this feature for reducing the searching space, the two methods mentioned in [37, 9] select landmarks or derive clusters based on the RSSI readings by developing a particular algorithm. For simplicity, the proposed method presets the clusters based on the physical space, in this case, one room and four corridors totally 5 clusters, since network administrators often deploy the APs rather evenly in the centre of each physical area for better coverage and maximizing the usage of each WiFi AP.

The network used to predict the cluster id of a target location is a pure MLP that takes the expanded RSSI fingerprint, which has $2 \times N$ individual readings, as input. The output of this NN contains C elements, each of which is a binary classifier, where C is the total number of clusters. Only the ith element of the output vector is 1, which indicates the corresponding fingerprint belongs to the ith cluster, while the others are all 0. Apparently, the network is trying to solve a multi-class classification problem by tuning the number of hidden layers and the number of neurons of each layer.

The loss function we choose is cross entropy loss or log loss, which is used to measure the performance of a classification model whose output is a probability value between 0 and 1.

$$Cross\,Entropy = -\sum_{i=1}^{C} y_i log(\hat{y}_i)$$

where y_i is the binary indicator (0 or 1) if class label i is the correct classification for the observation and $\hat{y}_i$ is the predicted probability the observation is of class i.

3.3.3 Localization

For each cluster, the proposed system builds one specific MLP that only uses the dataset in this cluster for training. In this case, five different MLPs are built, one for the room and four for the other corridors. Likewise, the network takes the expended RSSI fingerprint as input, while the output changes to the corresponding location coordinate in a two-dimension vector, instead of the multi-class vector.

For localization, we choose the mean square error (MSE) loss.

$$MSE = \frac{(\hat{x} - x)^2 + (\hat{y} - y)^2}{2}$$

where x, y are the actual coordinates and $\hat{x}$, $\hat{y}$ are the prediction.

3.3.4 Filtering

Since the RSSI readings fluctuate a lot, the online test results would also vary if there is no mechanism to mitigate this effect, leading to an unstable prediction. To improve the stability of the estimation result in the online stage, a pre-process using a median filter is applied to the N RSSI readings to reduce the noise inside. When a new pair of target RSSI readings comes, instead of directly feeding them into the network, the system calculates the median of each of these N readings along with its $w-1$ previous readings, where w is the window size, and then feed the generated median values to the network.

3.4 Experiment And Analysis

3.4.1 Experimental Setup

We have finished our experiments on the third floor of the Engineering Office Wing (EOW), University of Victoria, BC, Canada. The dimension of the area is 826 inches by 630 inches with 1 large lab area and 4 long corridors as shown in Fig. 3.2(a). The RSSI readings are collected at 332 RPs with a mobile device (Google Nexus 4 running Android 4.4) mounted on a 3-wheel robot to build up the fingerprint database for training. At each RP, 100 RSSI scans are obtained. For 6 APs we used, 5 of them provide 2 distinct frequency bands, 2.4 GHz and 5 GHz respectively, which means there are 11 RSSI readings in total from those 6 APs per scan. To build up the fingerprint dataset for training, the threshold distance $d = 40$ inches is chosen. To summarize, we choose $P = 6$, $N = 11$, $M = 332$, $S = 100$ and $d = 40$ inches to build up the training dataset.

We use Tensorflow and Keras for NN training and prediction in our experiment. In the training phase, we construct one simple MLP for clustering and five individual MLPs for detail localization, defined by the hyper-parameters shown in the Table 3.1 and keep training until no decrement in the training error could be achieved for both types of the MLPs. As we know, the training phase can only determine the value of each weight, but not hyper-parameters, such as the number of layers, the number of neurons in each layer and the activation functions. Thus we brute force a large

Table 3.1: Parameter used for training in Classification and Localization networks

Parameter	Clustering	Localization
Hidden layer count	2	1
Neurons in the input layer	22	22
Neurons in each hidden layer	32	60
Neurons in the output layer	5	2
Activation function in hidden layers	sigmoid	elu
Activation function in the output layer	softmax	linear
Dropout	0.3	0.3
Batch size	1024	1024
Optimization method	Adam	Adam

number of hyper-parameter combinations and the one with lowerest training error is chosen.

In the testing phase, new RSSI readings at different unknown points, totally 117, that cover the whole test area are collected with a total 20 scans at each location as the test dataset. Two different types of tests are performed to evaluate the proposed method, a clustering test and a final localization test. In both tests, a prediction is performed for each scan, not for the mean of several scans at the same location. During the clustering test, we calculate the classification accuracy carried out by the clustering network, while in the final localization test, the location predictions are presented and compared with the other methods in the literature. For both tests, the proposed approach applies the median filter with the window size $w = 3$ to each of the RSSI scans before it is fed into the network and then gives a new prediction back, and does not rely on any previous data collected or any previous prediction.

3.4.2 Clustering Test

The accuracy of the proposed method relies on the accuracy of the clustering. If the clustering network gave a wrong cluster prediction, the whole system can not return a precise location estimation, unless the unknown point is close to the boundary of two different clusters. In this test, we perform a classification test to evaluate the accuracy of the clustering network. To do so, we define 'accurate' and 'potentially accurate' as follows

- The prediction is accurate if and only if the network returns the correct cluster

Table 3.2: Mean localization error and Standard deviation of different methods

Method	Proposed	DANN	Boosting MLNN	AP-Similarity & KWNN
Mean error	43.5 inches	54.0 inches	64.1 inches	50.2 inches
Standard deviation	32.8 inches	36.3 inches	54.1 inches	44.7 inches
Processing time	2.73 ms	1.83 ms	2.01 ms	67.3 ms

to which the target location belongs.

- The prediction is potentially accurate if and only if the predicted cluster the network returns is adjacent to the actual cluster and the target location is within $d = 40$ inches to the boundary of the two clusters.

The result shows that the network gives 95% of the accuracy along with another 4% of potentially accuracy. In other words, we can believe that the network has 99% of the classification accuracy roughly.

3.4.3 Final Localization Test

To evaluate the proposed method, different methods are implemented as described in the literature and the performance of these methods are calculated under the same test environment. In this test, two pre-defined trajectories along which all the test points are randomly chosen are used and the test data is collected by the robot. We use Euclidian distance error as the primary benchmark to compare the performance and the processing time is calculated based on a 2011 Intel core i5 Thinkpad X220.

Table 3.2 shows the errors and processing time of the proposed scheme and existing methods in the literature including DANN [14], Boosting MLNN [19] and AP-Similarity & KWNN [9], while Fig. 3.3 shows the cumulative distribution function (CDF) of the errors. It can be seen that the proposed algorithm outperforms the others in both mean error and standard deviation. Benefiting from using the highly accurate clustering network, the proposed scheme shrinks the possible area into a small cluster for which a particular localization network is responsible. Thus the overall mean error decreases by 18% and 32%, the standard deviation by 8% and 39%, comparing to the two NN based method, DANN and Boosting MLNN. On the

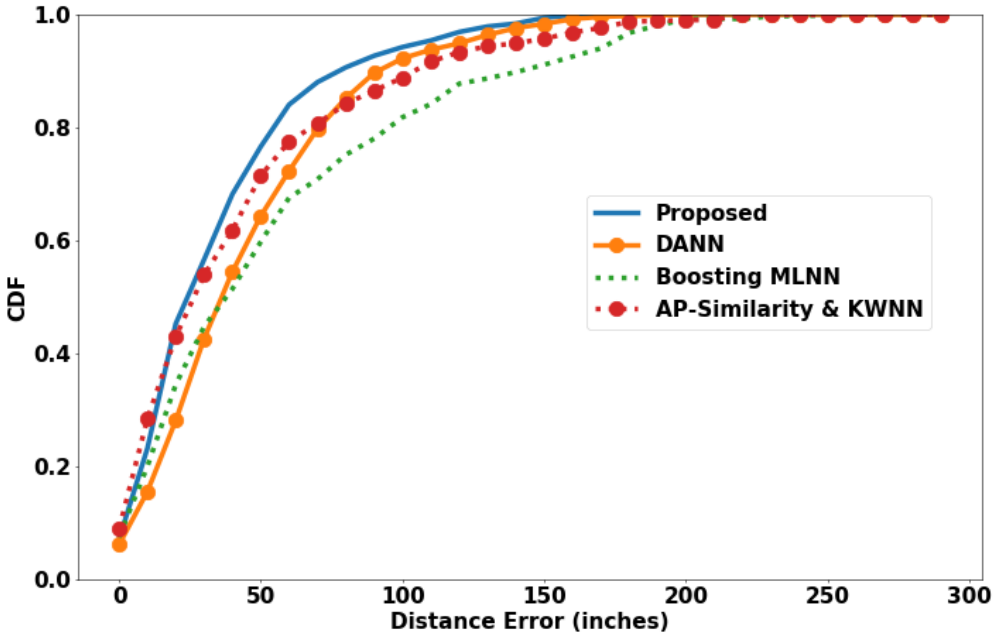

Figure 3.3: CDF of localization errors

other hand, because of the additional prediction needed by the clustering, the proposed scheme takes a little bit more processing time than these two, but still under the reasonable threshold. Although the AP-Similarity & KWNN method has the second best mean error, its processing time is much longer than the other three because of the significant searching space using KNN. The proposed scheme reduces the computing time to only 4% compared to the AP-Similarity & KWNN, while the performance is still better. The CDF indicates that the percentage of the localization error under 60 inches of the proposed is 84%, while the others 72%(DANN), 68%(Boosting MLNN) and 77%(AP-Similarity & KWNN) respectively.

3.5 Conclusions

In conclusion, we propose an NN-clustering-based IPS that uses a new form of fingerprint for WiFi indoor environment, by exploiting the similarities in RSSI readings of the nearby points. The experiment result shows that the proposed algorithm improves the localization accuracy over existing NN designs, reaching the mean error of 43.5 inches and 84% of the error within 60 inches. In the future research, we will

apply RNN for localization as it can be formulated to a time sequence problem that RNN is suitable to solve.

Chapter 4

Recurrent Neural Network in Channel Prediction with an Online Training Scheme

4.1 Introduction

The rapid advancements in machine learning and their successful applications to a broad range of problems in recent years have sparked significant interest in the communications community to adopt learning techniques for communications challenges. While machine learning is actively used for radio resource management, network optimization and other higher layer aspects, it is also being studied for physical layer designs. The general methodology is either using machine learning techniques for a specific physical layer problem ranging from channel estimation and detection [40], decoding [28, 16] to equalization [8, 10], spectrum usage recognition [15], etc., or viewing the communication system design from a new paradigm as an autoencoder in deep learning [31].

Classical communication theory has developed sophisticated statistical models and design principles for different components of a communication system. Such methodology has led to largely efficient and successful wireless systems today. As more demanding performance requirements are envisioned for 5G and future generation systems, the complexity of physical layer design will escalate. For example, massive multiple input multiple output (MIMO) will employ a large number of antennas at the base station and possibly even at the user equipment when mmWave bands are

used. The multipath propagation environment is critical to any wireless communication system design, and as the number of antennas grows the accurate modeling of the huge matrix channel is increasingly difficult and complex. Supervised learning, on the other hand, operates based on a large number of training data to establish the underlying relationship between input and output. This makes it possible to generate a learning model that is not easily describable by mathematical formulas but highly effective. One of the challenges of this machine learning approach is generalization. Although deep learning NN are widely believed to have better generalization ability than traditional signal processing approaches in the field of computer vision, natural language processing, etc., its use in physical layer communications is still in question as the propagation environment is immensely diverse and dynamically changing. Another challenge is the real time requirement of any effective model in physical layer communications. In this chapter, we attempt to address these two challenges by designing neural networks for the most fundamental problem, i.e., channel prediction and estimation, of a single input single output (SISO) communications system. The idea developed can be extended to more sophisticated MIMO channel prediction and estimation, beamforming training and tracking, frequency selective channel prediction, etc.

As well known, obtaining CSI is vital to both transmitter and receiver for high spectral efficiency. Since a wireless propagation channel varies due to user mobility and changing dynamics in the environment such as appearance and disappearance of human and objects, conventional methods employ transmitting known pilot symbols, also called reference symbols, to estimate the channel in real time. This creates pilot overhead. To estimate the channel at non-pilot positions, interpolation or extrapolation (prediction) of the CSI estimation at pilots are carried out. Moreover, channel estimation is done at the receiver side. For the transmitter to know the channel, either CSI feedback is required in FDD or pilots are transmitted in the opposite direction to estimate the CSI of the reverse link and assume channel reciprocity in TDD. CSI feedback consumes much reverse link resources and more importantly introduces a feedback delay. In a dynamic environment, the channel condition may have already changed after the feedback delay. Therefore, channel prediction is very useful in this case [13, 12].

The conventional channel prediction techniques can be divided into three groups, the PRC model [2, 36], BEM [41] and the AR model [23, 17, 13, 18]. These methods predict CSI based on certain theoretical channel propagation models and/or estima-

tion of channel long term statistics (e.g., covariance matrix) and channel parameters (e.g., angle of arrival/departure, Doppler frequency). The errors in the estimated values lead to the inaccuracy of the channel prediction as well.

As aforementioned, machine learning based approaches train a learning model, e.g., neural network using a large known dataset to find the internal relation underneath, instead of deriving equations based on assumptions and propagation models. The limitation in an NN model is the finite training set which only accounts for the scenarios described by the training set. When the network model encounters propagation channels dramatically different from what it has viewed before, the performance drops significantly.

In this chapter, we propose a neural network based prediction which can adapt to the continuously changing channel by regularly training the online model of a small efficient RNN. We present a training and testing schedule that achieves real time channel prediction based on the recent history pilot assisted and data assisted CSI estimation. Initial simulation data generated from the 3GPP Spatial Channel Model (SCM) demonstrate that the RNN has the ability in learning and analyzing the sequential CSI data in a changing environment and outperforms the conventional AR model.

The rest of the chapter is organized as follows. Section 4.2 introduces the related work on channel prediction and machine learning, followed by the detail model in Section 4.3. Section 4.4 compares the result with other approaches, and finally conclusion is given in Section 4.5.

4.2 Related Work

4.2.1 Conventional Techniques

Parametric Radio Channel Model

The PRC models the channel as a linear combination of complex sinusoids whose parameters are determined by its amplitude and Doppler frequency. Given a history segment of known channel coefficients, the PRC estimates the parameters of each sinusoids first and then uses them for channel prediction. Reference [2] used it for SISO channel and [36] for MIMO channel prediction.

Basis-Expansion Model

The BEM is another linear model to represent time-varying channels, which describes the channel as a linear set of several low dimension orthogonal basis, such as complex exponential (CE) functions, polynomials and discrete prolate spheroidal (DPS) sequences, etc. These bases are determined by the channel itself and the coefficients of the basis are calculated from the known data at pilot positions. Reference [41] proposed a DPS-BEM based approach for minimum-energy (ME) band-limited prediction of sampled time-variant flat-fading channels. The numerical simulation results show their proposed predictor with dynamic subspace selection performs better than or similar to a predictor based on CE-BEM with perfectly known frequencies for a prediction horizon of one-eighth of a wavelength.

Autoregressive Model

To predict the future CSI, the AR model treats the channel as a linear combination of the known channel coefficients and applies some filter techniques with the knowledge of the channel correlation matrix. Available filters include minimum mean square error (MMSE) [23], least mean squares (LMS) [17], recursive least squares (RLS) [13] and Kalman filtering [18].

4.2.2 Neural Network based Techniques

In the field of channel prediction, Ding et al. introduced a multilayer complex-valued neural network (ML-CVNN), a variant of the deep neural network (DNN), based channel prediction approach combined with chirp z-transform (CZT) in a fading environment [11]. By utilizing the history CSI, the multilayer neural network is trained offline and can predict the next CSI based on several recent measured CSI. A series of simulations and experiments are done and their analysis indicates the system performance depends on window length, learning iteration number and hidden neuron number to input number ratio. With the assumption of a mathematical Jakes's model, they try to estimate parameters in such a model, for example, amplitude, frequency and phase of each path, based on which the channel is then predicted. While the channel is unfit for the particular model, the accuracy of the proposed method may decrease.

Luo et al. proposed a learning framework, which consists of a 2D convolutional

neural network (CNN), a 1D CNN network, and a long short term with memory (LSTM) network [25]. By taking an input data composed of the current CSI and several critical features that affect the propagation model, such as temperature and humidity, the framework predicts the the CSI for the next step. An offline-online two-step training mechanism is developed to improve the stability of the system. In the offline training step, a well-trained NN is obtained on massive volume of historical data, while the weights of this well-trained NN is updated by the gradients in the online step, which are calculated based on the recently measured data. But the update period of the weights is set as 5 miniutes, thus the channel condition may have already changed.

4.3 System Model

4.3.1 Problem Formulation

For a SISO system, the relation between the source symbols and the corresponding received signals is shown below

$$y[t] = h[t]x[t] + z[t]$$

where $x[t]$ and $y[t]$ represent the symbols transmitted at the transmitter side and the corresponding received signals at the receiver side respectively, $h[t]$ is the complex CSI and $z[t]$ is the additive white Gaussian noise at time t. To obtain channel state information, known pilot symbols $p[t]$ are sent and the measured CSI can be derived from the received signal as

$$\hat{h}[t] = \frac{y[t]}{p[t]} = h[t] + \frac{z[t]}{p[t]} = h[t] + n[t].$$

Since pilot symbols are overhead that do not carry information, communications systems often limit the number of pilots transmitted. Typically pilot blocks are transmitted at the beginning of a transmission, followed by interleaved pilot symbols. In this chapter, we apply channel predictions based on channel estimations from both pilot symbols and data symbols. In a FDD system, the received estimates the CSI from pilot symbols and use the interpolated CSI for data detection. CSI estimation at the data positions can then be refined from the decoded data. The transmitter

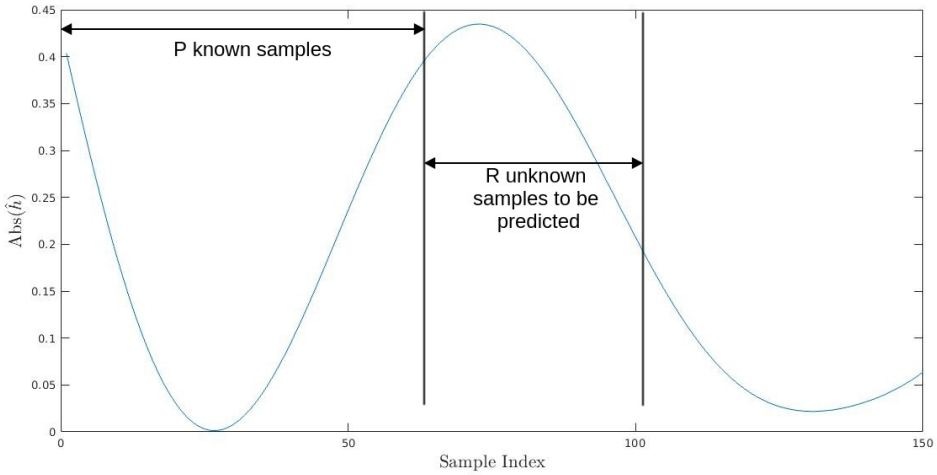

Figure 4.1: SCM channle, and the Prediction Setup

then uses the fed back CSI to predict the channel for transmitter precoding. In a TDD system, the transceiver can use the reverse link signal for CSI estimation and predict the channel for forward link precoding.

Now the prediction problem is summarized and shown in Fig. 4.1. That is, assuming measured CSI (perfect or noisy) are known over the first P time steps, the system predicts the CSI of the next R time steps. For better accuracy, instead of predicting the CSI directly, we propose to predict the CSI difference between two adjacent symbols, defined as

$$\hat{h}^d[t+1] = \hat{h}[t+1] - \hat{h}[t]$$

Then, the problem becomes to predict the next R differences $\tilde{h}^d[t]$, $P+1 \leqslant t \leqslant P+R$, based on the first $P-1$ known differences $\hat{h}^d[t]$, $2 \leqslant t \leqslant P$. To return the final CSI prediction, the system calculates

$$\tilde{h}[t] = \begin{cases} \hat{h}[t-1] + \tilde{h}^d[t] & t = P+1 \\ \tilde{h}[t-1] + \tilde{h}^d[t] & P+2 \leqslant t \leqslant P+R \end{cases} \tag{4.1}$$

where $\tilde{h}^d[t]$ and $\tilde{h}[t]$ are the differences and the final CSI results predicted at time t.

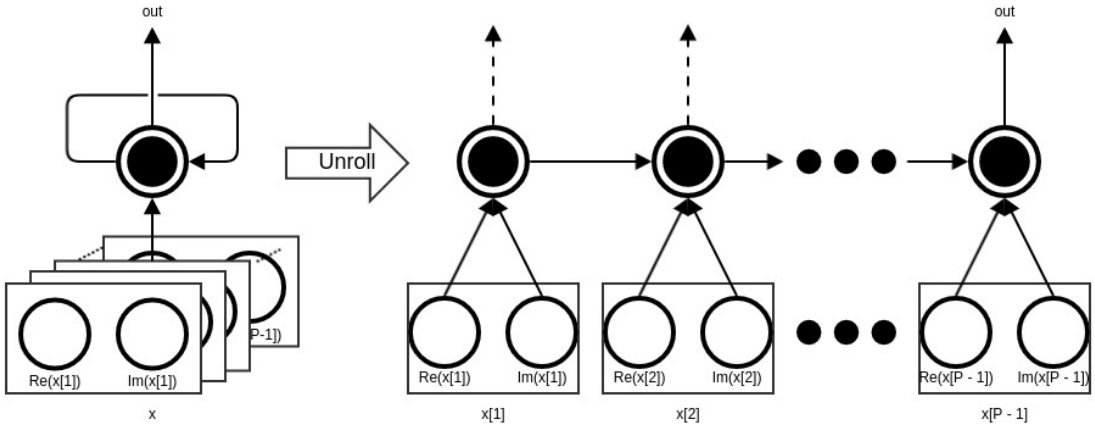

Figure 4.2: Structure of one RNN Unit

4.3.2 Recurrent Neural Network

RNN is a type of NN that the neurons do not always feedforward, but connect to the neurons that in the previous layer or the current layer itself. Thus RNN can use their internal state or memory to process the input so it is suitable for problems that can be formulated into a time sequence.

Fig. 4.2 shows the structure of one RNN unit in our setup, where $x[t]$ denotes the input at time step t, and *out* is the output of this unit. In the left side of the Fig. 4.2, an RNN unit takes a time series of input data x and outputs a value *out*. The loop in the RNN unit makes it possible to pass the information from one step to the next. In other words, an RNN unit can be treated as a horizontal expansion of the copies of itself. The right side of the Fig. 4.2 shows the network after the loop is unrolled. Though the network produces an output at each step, we only take the last one as the output of this unit in our setup.

The type of RNN used in our setup is LSTM network, designed to learn and avoid long-term dependency problems, which standard RNNs are not capable to solve, by introducing an input gate, an output gate and a forget gate in one unit other than a single tanh layer.

4.3.3 Network Structure

As described in Subsection 4.3.1, the objective of the proposed system is to predict the next R unknown differences, based on the first $P-1$ known differences. The structure of the proposed network is shown in Fig. 4.3. Since the proposed system

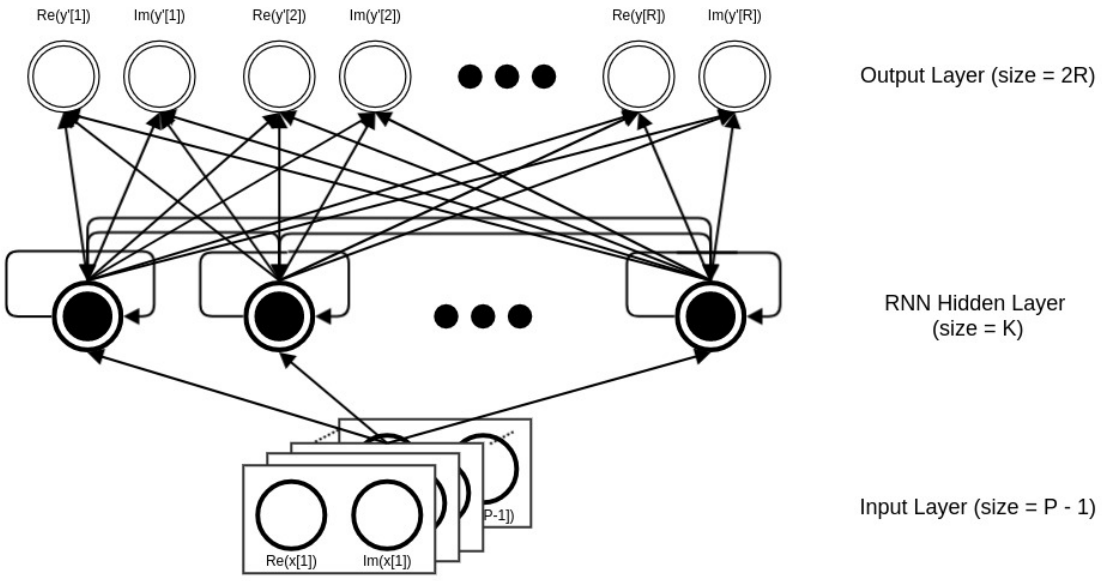

Figure 4.3: Network Structure

is a real-time application and NN training costs a period of time, thus we design the network to be simple and efficient with only one hidden layer.

The input layer contains $P-1$ known CSI differences, $\hat{h}^d[t]$, $2 \leqslant t \leqslant P$, which are then fed into each of the K RNN units in the next hidden layer. Note that, there are synapses between the RNN units in this layer, which give the recurrent network the abilities to share information during the training stage. The output layer, a fully connected layer, generates $2 \times R$ real numbers using a linear activation function, which calculates a weighted sum of the outputs from the hidden layer. The output will later be converted into R complex numbers, $\tilde{h}^d[t]$, $P+1 \leqslant t \leqslant P+R$, to get the final CSI prediction.

4.3.4 Model Training and Testing

The network is trained and tested by the simulation data generated by SCM and additive white Gaussian noise.

For a time sequential training data of length M, we choose $w = P + R$ to be the trace size. Based on a trace of length w, we calculate $w-1$ differences, the former $P-1$ of which are treated as known input differences of the network while the later R as the actual differences of the unknowns to be predicted. We take the first w samples to form the first training trace, and then move the window forward by 1 time

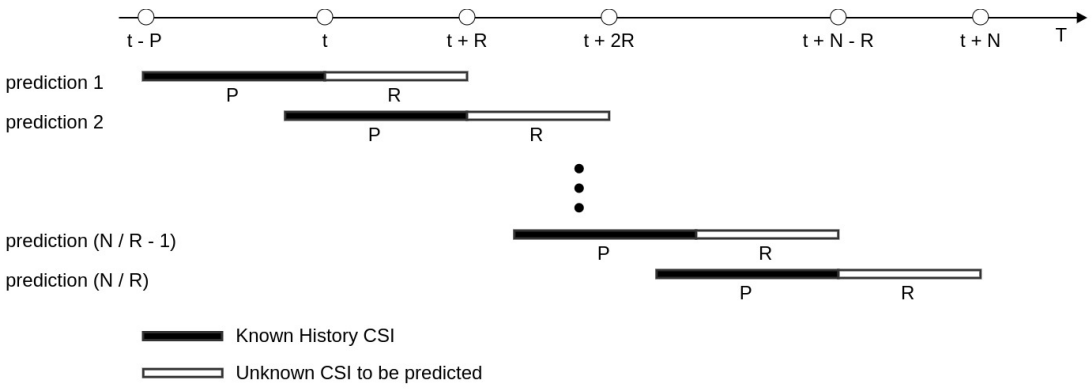

Figure 4.4: Process of Predicting N Unknowns

step for another trace and totally generate $M - P - R + 1$ traces.

The network is trained by minimizing the difference between the output of the network and the actual values. In our model, we choose the mean square error (MSE) loss.

$$MSE = \frac{1}{2R}\sum_{k=1}^{2R}(\hat{y}[k] - y[k])^2$$

where $\hat{y}$ is the prediction and y is the actual value.

To predict a time series of N steps from t to $t + N$, $N > R$, the proposed system groups $b = R$ predictions as a block and predicts block by block. To predict a block, it predicts the R unknown differences based on the $P-1$ previous differences by using the trained network and then calculates the final R predictions using (4.1). Fig. 4.4 shows the process of predicting N unknowns. By utilizing the P previous known CSIs from $t - P + 1$ to t, the proposed system predicts the first R unknowns from $t + 1$ to $t + R$, followed by the next R predictions. The process is repeated until the N predictions are achieved.

4.3.5 Timing Schedule

The proposed system adapts to the continuously changing channel environment by taking advantage of an online training phase using the most recent CSI, which requires a short period of time before the network can be used for prediction. The workflow is shown in Fig. 4.5.

By taking the recent M history CSI for training at time step $t + M$, it costs N time steps to finish training before the newly built network can be used for prediction

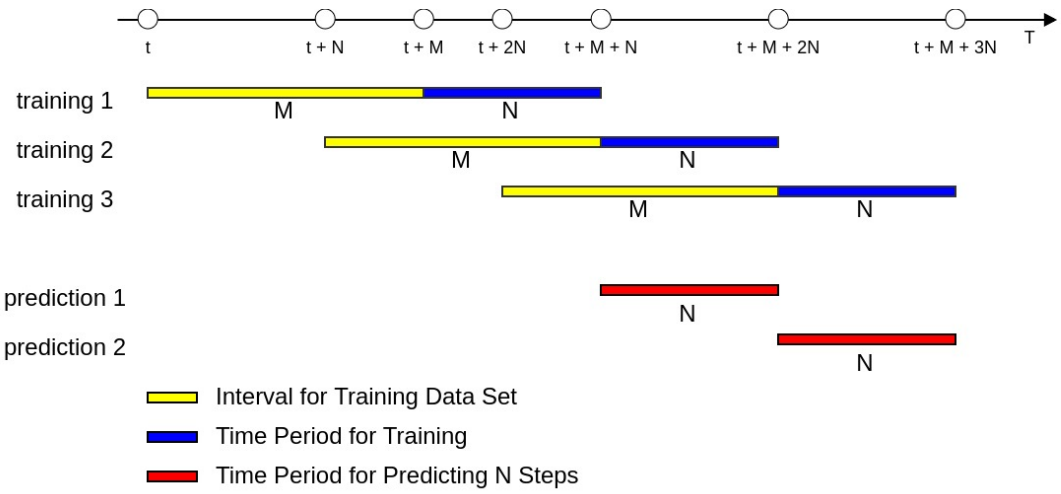

Figure 4.5: Timing Schedule

at time step $t + M + N$. It then takes the most recent M history CSI from time step $t + N$ to $t + M + N$ to modify the existing network, and again from $t + 2N$ to $t + M + 2N$. Simultaneously the system uses the copy of the trained network to do the CSI prediction for a time period of N time steps from time step $t + M + N$ to $t + M + 2N$ for the first time when the network is well trained and then uses the updated network when it is ready at time step $t + M + 2N$.

4.4 Experiment And Analysis

4.4.1 Experimental Setup

Several training and verification traces are generated by SCM to demonstrate the effectiveness of the proposed system, whose performance is compared to an offline RNN model, which uses the same structure, the same parameters but only with offline training, and the conventional AR method with minimum mean square error filter. All the experiments are performed on a 2011 Intel core i5 Thinkpad X220 with 16G RAM running Ubuntu 18.04 by using python 2.7.5, Keras 2.2.0 and TensroFlow 1.9.0 and the results show that the proposed online training and prediction approach outperforms the other two models in most of the cases.

We use the parameters defined in Table 4.1 to generate simulation data with additive noise, based on which the offline RNN model is trained and all the three models are tested.

- For the proposed network, no offline training is performed and online training and prediction are performed as described in Subsection 4.3.4 and 4.3.5 during the test phase.

- For AR model, no training phase is needed and predictions are calculated block by block using the same block size as the proposed.

- To train a particular offline RNN, we generate a corresponding dataset of 250 different training data, each of which is 1000 time steps long with the same *SampleDensity* but random *MsBsDistance*, *ThetaBs*, *ThetaMs*, *MsDirection*, to include all the possible scenarios under the certain Doppler shift for the network to view. Several networks are trained under different scenarios and the system will choose the best-matched network to predict the unknown CSI block by block as AR model does.

To test these three models, we generate several random test data with a length of $200,000$ for each of the particular scenarios which will be described and discussed in the following sections, then run the test and calculate the average performance.

In our experiment, we choose $K = 32$, $P = 20$, $R = 10$, $M = 5000$, $N = 2000$. To construct and train the network, including the proposed and offline RNN, we use parameters defined in Table 4.2.

The proposed approach works as described in Subsection 4.3.4 and 4.3.5 with no additional parameters needed while the other two models predict the CSI block by block with the assumption of some known parameters in model generation. Fig. 4.6 shows a sample prediction using offline RNN, AR and the proposed and the ground truth.

For the timing schedule, each orthogonal frequency division multiplexing (OFDM) symbol duration is $\frac{1}{14}$ millisecond in long-term evolution (LTE). So we are supposing the system can finish training a training data of length 5000 in 2000 time steps, which is $\frac{1}{14} \times 2000 = 143$ milliseconds. The actual time cost of the proposed on the experiment platform is 600 to 800 milliseconds for training, but 100 to 130 milliseconds if equipped with a Nvidia Titan Xp GPU. Though it takes longer time than expected on the CPU-only platform, it is much lower if a GPU or designated hardware are encountered. Furthermore, the channel will not change so fast in the real environment in most cases that the training period can be much longer, such as 1 second or more.

Table 4.1: Parameters used for SCM

Parameter	Value
NumBsElements	1
NumMsElements	1
Scenario	'urban_micro'
SampleDensity	case dependent
NumTimeSamples	case dependent
UniformTimeSampling	'no'
BsUrbanMacroAS	'eight'
NumPaths	1
NumSubPathsPerPath	20
CenterFrequency	2.0000e+09
ScmOptions	'none'
DelaySamplingInterval	1.6276e-08
XpdIndependentPower	'no'
PathLossModelUsed	'no'
ShadowingModelUsed	'no'
PathLossModel	'pathloss'
AnsiC_core	'no'
LookUpTable	0
RandomSeed	[]
BsGainPattern	1
BsGainAnglesAz	[1 × 90 double]
BSGainAnglesEl	0
BsElementPosition	0.5000
MsGainPattern	1
MsGainAnglesAz	[1 × 90 double]
MsGainAnglesEl	0
MsElementPosition	0.5000
InterpFunction	'interp_gain'
InterpMethod	'cubic'
MsBsDistance	case dependent
ThetaBs	case dependent
ThetaMs	case dependent
MsVelocity	10
MsDirection	case dependent
MsHeight	1.5000
BsHeight	32
MsNumber	1

Table 4.2: Parameters used for network configuration and training

Parameter	Value
RNN Model	LSTM
Dropout	0.2
Batch size	100
Optimization method	Adam

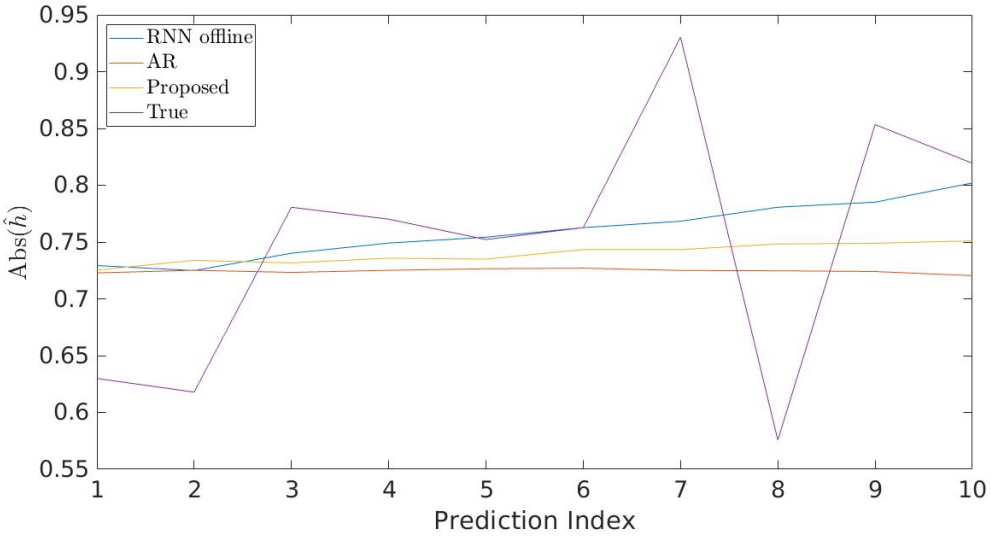

Figure 4.6: Sample Prediction and Ground Truth

4.4.2 Impact of the Normalized Doppler Shift

The proposed method is first compared in the situation when the Doppler shift changes. To generate the test data, all the parameters are fixed except the *SampleDensity*, to simulate that the normalized Doppler shift changes from 0.005 to 0.01 evenly and then white Gaussian noise leading to signal-to-noise ratio SNR= $20, 15, 10$ dB are added. For the other two models, we assume they know the exact SNR and choose the upper and lower bound of the changing normalized Doppler shift value respectively, thus AR model has all the parameters it needs for prediction and offline trained RNN can select the corresponding network to do the prediction.

Fig. 4.7 - 4.9 show the performance of three models under different SNRs, from which we can see clearly that the proposed model has the best performance, while AR is better compared to offline trained RNN. Though the performance is very close between AR and the proposed when SNR= 10 dB, a practical system must estimate the SNR for use in the AR model, while the estimation will contain deviation from real SNR values. The proposed scheme has advantage because it does not need this additional information and regular online training allows the model to adapt to a changing environment.

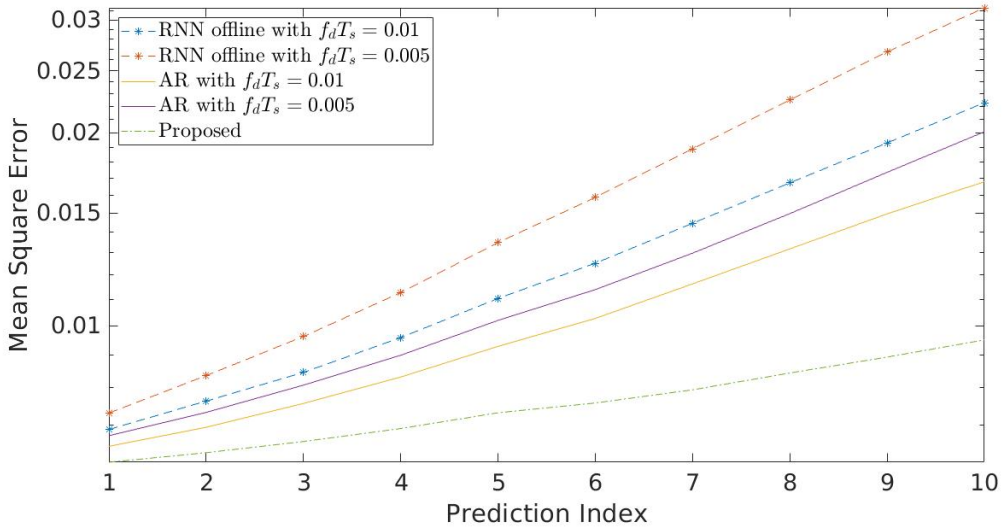

Figure 4.7: Performance Comparison - Normalized Doppler Shift changes from 0.005 to 0.01 evenly and $SNR = 20$ dB

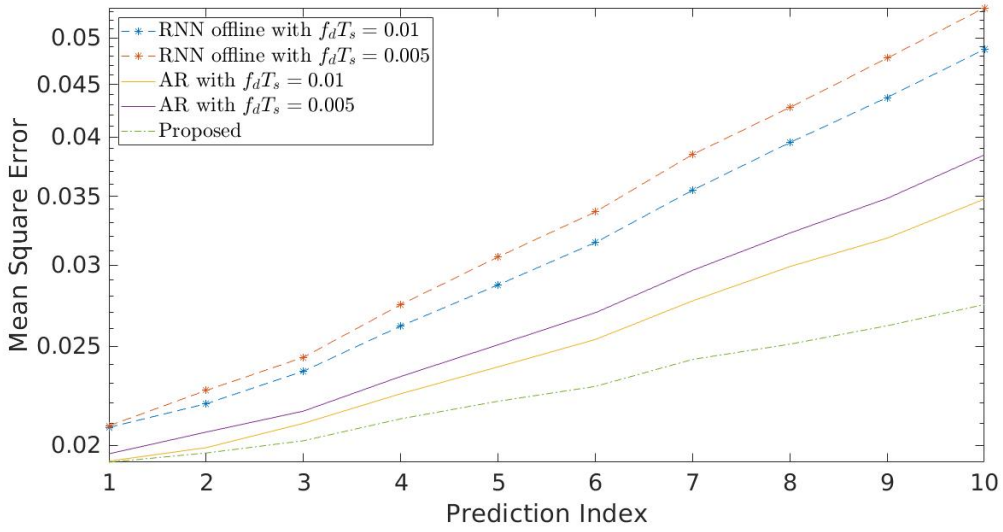

Figure 4.8: Performance Comparison - Normalized Doppler Shift changes from 0.005 to 0.01 evenly and $SNR = 15$ dB

4.4.3 Impact of the Base Station Angular Parameters

As we know, the base station angular parameters will also change continuously in the real environment. We fix all the parameters except the *ThetaBs*, to simulate that it changes from $-180°$ to $180°$ evenly, and then add the white Gaussian noise with

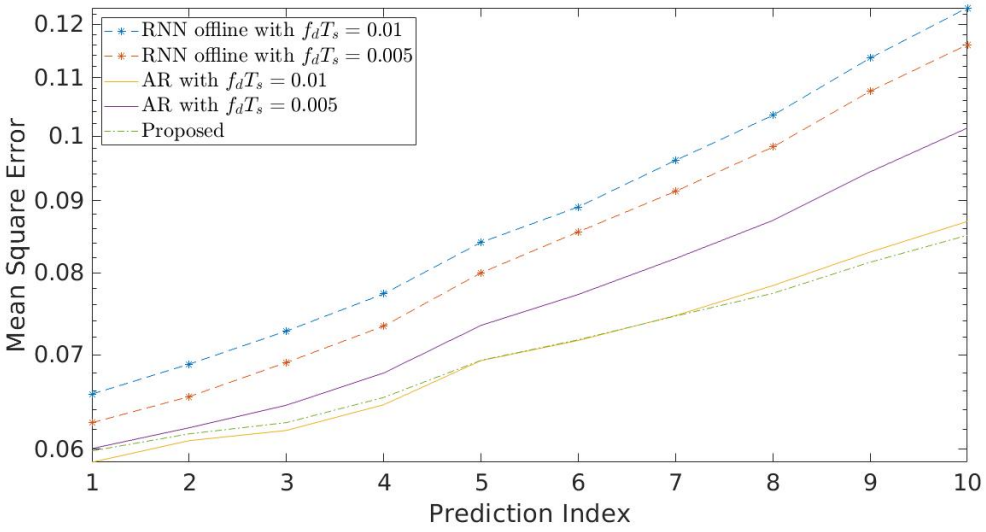

Figure 4.9: Performance Comparison - Normalized Doppler Shift changes from 0.005 to 0.01 evenly and $SNR = 10$ dB

$SNR = 20$ dB. For the other two models, we assume they know the exact normalized Doppler shift and SNR.

Fig. 4.10 illustrates the comparison between the proposed, AR and offline trained RNN. From the figure, the proposed model is the best among the three but with only a bit better than AR model. However, here we assume the other two methods know the exact normalized Doppler shift and SNR, which the proposed model does not need to know.

4.4.4 Impact of the SNR

Due to the time varying and multipath nature of the environment, the SNR will also change during signal transmission. To generate the test data, all the parameters of SCM are fixed to simulate a certain channel while the SNR changes from 20 dB to 10 dB evenly. For the other two models, we assume they know the exact normalized Doppler shift and choose the upper, middle and lower bound of the changing SNR value respectively.

Fig. 4.11 compares the performance of the proposed method, AR and the offline trained RNN when SNR changes. It can be seen that the proposed method is the one with the best performance, while AR is better than the offline trained RNN.

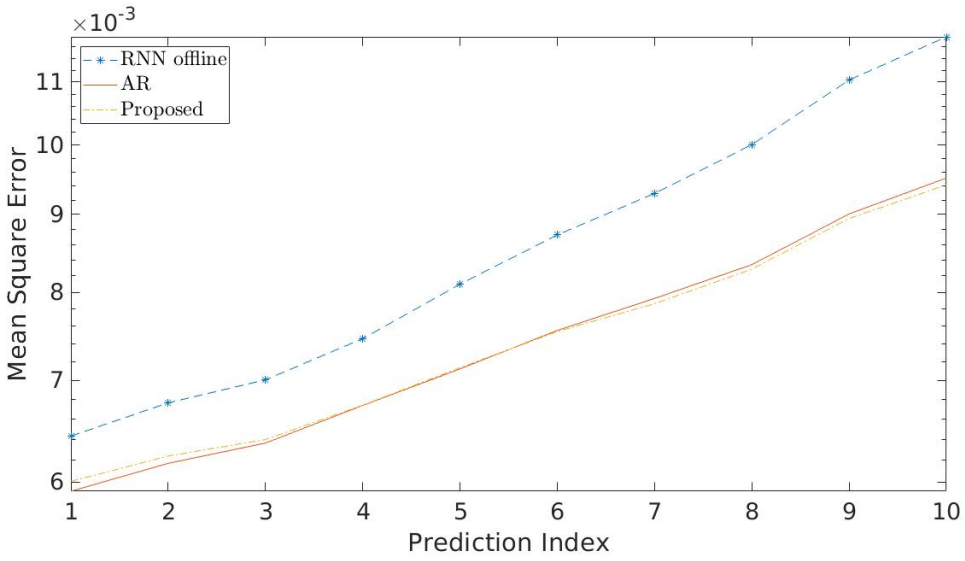

Figure 4.10: Performance Comparison - Base Station Angular Parameters (ThetaBs) changes from $-180°$ to $180°$ evenly

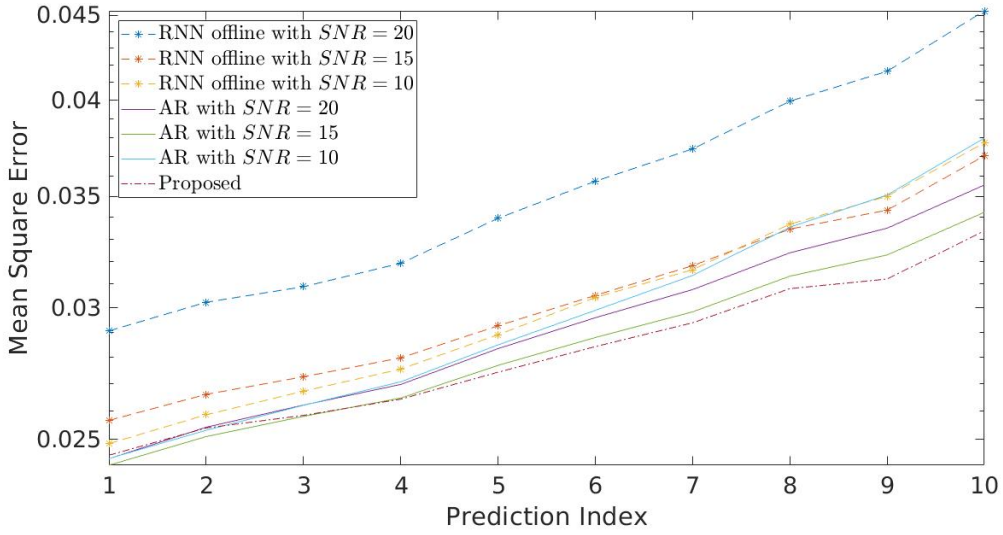

Figure 4.11: Performance Comparison - SNR changes from 20 dB to 10 dB evenly

4.4.5 Impact of the Normalized Doppler Shift and the SNR

To account for more realistic and complex scenario, we simulate a dynamically changing environment in which both the normalized Doppler shift and SNR are changing. The test data is generated in a similar way, fixing all the parameters except the *SampleDensity*, to simulate a channel with the normalized Doppler shift ranging from 0.01

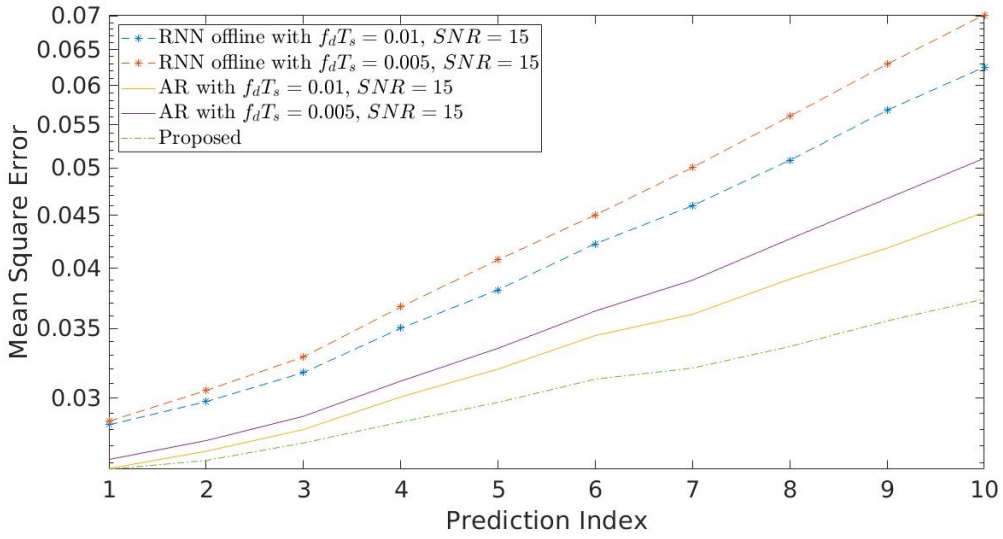

Figure 4.12: Performance Comparison - Normalized Doppler Shift changes from 0.01 to 0.005 and SNR changes from 20 dB to 10 dB evenly

to 0.005 evenly and then adding a sequence of white Gaussian noise resulting in the SNR changing from 20 dB to 10 dB evenly as well. For AR and offline RNN models, we choose $SNR = 15$ dB and the upper and lower bound of the changing normalized Doppler shift value respectively.

Fig. 4.12 demonstrates the fact that the proposed scheme is the best among these three methods. Instead of having to estimate the normalized Doppler shift and SNR value, the proposed method learns the channel from the data itself before prediction, which helps adapt the changing channel and improve the prediction accuracy.

4.4.6 Impact of the Number of RNN Units K and the Known History Length P

To compare the performance of the proposed scheme with different setups, we choose different values for the number of RNN units K and the known history length P. The test data used is as same as the one used in Subsection 4.4.5.

Fig. 4.13 and Fig. 4.14 demonstrate the performance of the proposed under different setups, from which we can see that the performance increases along with the rise of K and P, but the improvements between them are getting smaller. Though we pursue the accuracy as high as we can, the computing time is another factor that we need to take into consideration for system design. As the rise of K and P, the

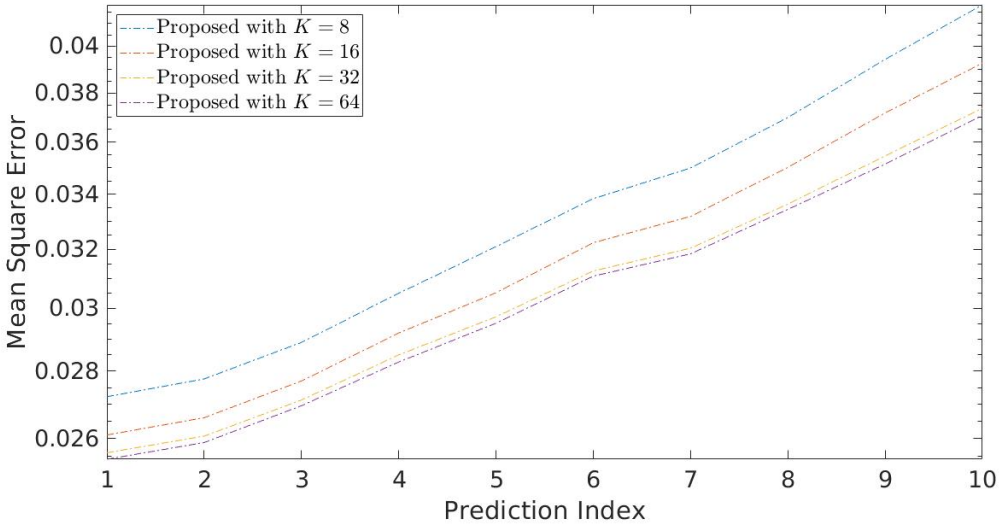

Figure 4.13: Performance Comparison - Different Value of K

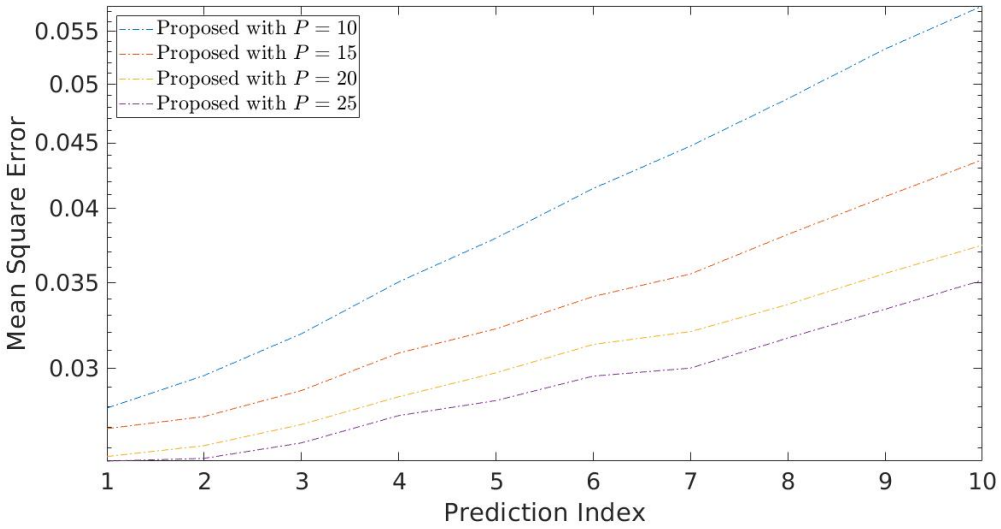

Figure 4.14: Performance Comparison - Different Value of P

computing time for training grows as well. Thus the proper value should be chosen based on the hardware used and the real scenarios.

4.5 Conclusions

In conclusion, we have proposed a small efficient online training based RNN model for real-time channel prediction using the recent history CSI estimation. Initial results has demonstrated the ability of RNN in learning and analyzing the sequential CSI data. The online training and prediction schedule enable the model to adapt to changing environments, which leads to the best performance compared to the conventional AR model and offline trained RNN. The proposed approach does not need any knowledge of the channel such as long term statistics or channel parameters before it can work. More comprehensive experiments with real world measured data and rigorous timing design considering the real hardware capacity are left for the future work.

Chapter 5

Conclusions

This thesis studies neural network applications in two different areas, indoor localization and physical layer communication. Chapter 2 applies a DNN based RSSI fingerprinting IPS with a two-step scheme while Chapter 3 investigates a real-time CSI prediction approach based on the recent history estimation using an RNN structure. This chapter concludes the above research topics and presents some open issues and expectations for the future research.

5.1 DNN based RSSI fingerprinting Indoor localization

In this thesis, we have proposed a DNN based RSSI fingerprinting indoor localization system including multiple neural networks with a two-step scheme. To reduce the time complexity and improve the prediction accuracy, the proposed system first uses the classification network to predict the cluster that the target RSSI belongs to and then uses the corresponding localization network to calculate the final location estimation. By utilizing the similarity in RSSI readings within a specific region, the system applies a new type of fingerprint which is a combination of two RSSI readings obtained from two closeby locations within a threshold distance. Real experiment result shows that the proposed scheme outperforms the existing NN designs, reaching a result of 43.5 inches mean localization error with 84 % of its predictions under the error of 60 inches. The time complexity drops significantly as well, reducing to 4 % compared to AP-Similarity & WKNN while the performance is still better.

5.2 Real-time CSI Prediction Approach using RNN

In this thesis, we also extend the ability of the neural network to physical layer communication by applying an RNN based approach for real-time CSI prediction, which is a small but efficient online training system based on the recent history pilot assisted and data assisted CSI estimation. With the online training scheme using the recent history data and a properly designed time schedule, the proposed system adapts to the continuously changing channel to give a better CSI prediction. Furthermore, the proposed system needs no other input parameters, neither the internal properties of the channel itself nor the external features that affect the channel propagation, but the CSI itself for training and prediction. Initial simulation result using data generated by the 3GPP SCM shows that the proposed approach has the best performance compared to the conventional AR model and the offline trained RNN, which demonstrates an RNN structure has the ability in learning and analyzing the sequential CSI data in a changing environment.

5.3 Future Work

There are several issues appear to be worthwhile for future research. These include:

- Currently only applies DNN for indoor localization in the experiment setup. As this indoor localization problem can be formulated to a time sequence that RNN is suitable for, it would be of interest to extend an RNN scheme to this problem in the future research.

- As an initial study of channel prediction approach using the neural network, in this thesis we have focused on the problem formulation and algorithm development. The test is performed using the simulation data generated by 3GPP SCM. More comprehensive experiments with the measured data and rigorous timing design considering the real hardware capacity should be investigated in the future.

Appendix A

Key Python Code for Indoor Localization with a Two-Step Scheme

```
def training_multi_label(path, file_name, label_file, test_file,
    max_dist=1):
    data = gen_training_data_using_close_position_rssi(path + file_name,
        max_dist)
    labels_map = get_labels_map(path, label_file)

    row_num = data.shape[0]
    label_num = len(labels_map)
    labels = numpy.ndarray(shape=(row_num, 1), dtype=numpy.int)
    for i in range(row_num):
        position = data[i, 0:2]
        key = (position[0], position[1])
        labels[i] = labels_map.get(key, None)
    index_offset = numpy.arange(row_num) * label_num
    labels_one_hot = numpy.zeros((row_num, label_num))
    labels_one_hot.flat[index_offset + labels.ravel()] = 1

    x = data[..., 2:]
    y = labels_one_hot
```

```
    num_layers = 2
    num_neurons = 32
    dropout = 0.3
    batch_size = 1024
    epochs = 10

    my_callback = MyCallback(path)
    model_name = '_'.join([str(num_layers),
                           str(num_neurons),
                           str(dropout),
                           str(batch_size),
                           file_name]) + '_multi_label.h5'

    if isfile(path + model_name):
        model = load_model(path + model_name)
    else:
        model = build_classification_model(22, label_num, num_layers,
            num_neurons, dropout)

    try:
        model.fit(x, y,
                  validation_split=0.3, callbacks=[my_callback],
                  batch_size=batch_size, epochs=epochs)
        model.save(path + model_name)

        print(predict_multi_label_with_history(path, model_name,
            label_file, test_file))
    except KeyboardInterrupt:
        pass

def training_regression(path, file_name, test_file):
    data = read_file(path + file_name)

    position2list_map = gen_position2list_map(data[..., 0:2], data[...,
        2:])
    combination = gen_pre_now_combination(position2list_map, 1)
```

```
x = combination[..., 2:]
y = combination[..., 0:2]

num_layers = 1
num_neurons = 60
dropout = 0.3
activation = 'elu'
batch_size = 1024
epochs = 10000

my_callback = MyCallback(path)
model_name = '_'.join([str(num_layers),
                       str(num_neurons),
                       str(dropout),
                       activation,
                       str(batch_size),
                       file_name]) + '_regression.h5'

if isfile(path + model_name):
    model = load_model(path + model_name)
else:
    model = build_regression_model(22, 2, num_layers, num_neurons,
        dropout, activation)

while True:
    model.fit(x, y,
              validation_split=0.3, callbacks=[my_callback],
              batch_size=batch_size, epochs=epochs)
    model.save(path + model_name)

    predict_regression_with_history(path, model_name, test_file)
```

Appendix B

Key Python Code for Channel Prediction with an Online Training Scheme

```
from data import extract_file_content_binary_float64
from data import difference
from data import gen_scaler
from data import modify_x

import numpy

from keras.layers import Dropout, Input
from keras.layers.core import Dense
from keras.layers.recurrent import LSTM
from keras.models import Model

neurons = 32
dropout = 0.2
batch_size = 100

data_start = 0
data_length = 200000
data_dim = 2
```

```
min_ = -0.05
max_ = 0.05

train_length = 5000
train_time = 2000
data_seg = 1000
predict_length = 10
init_epochs = 20
epochs = 1

known_length = 19


def build_model():
    in1 = Input(shape=(known_length, data_dim))
    x1 = LSTM(neurons, return_sequences=False)(in1)
    x1 = Dropout(dropout)(x1)
    y1 = Dense(data_dim * predict_length, activation='linear')(x1)

    mo = Model(inputs=in1, outputs=y1)
    mo.summary()
    mo.compile(loss="mse", optimizer="adam")

    return mo


def get_scm_train_file_name():
    return 'data/train/SNR20_train_-180-180.dat'


def train(d, m, e):
    diff_x = []
    diff_y = []

    start = 0
    while True:
        end = start + data_seg
```

```
        if end > train_length:
            break

        diff = difference(d[start:end])
        scaled = scaler.transform(diff)

        x = modify_x(scaled, known_length)[:-known_length - predict_length]
        y = modify_x(scaled[known_length:],
            predict_length)[:-predict_length]
        y = y.reshape(-1, data_dim * predict_length)

        diff_x.append(x)
        diff_y.append(y)

        start = start + data_seg

    x = numpy.concatenate(diff_x, axis=0)
    y = numpy.concatenate(diff_y, axis=0)
    m.fit(x, y, batch_size=batch_size, epochs=e, validation_split=0,
        verbose=0)


def test(d, m):
    errors = []
    start = 0
    while True:
        end = start + data_seg
        if end > train_time:
            break

        errors.append(test_seg(d[start:end], m))
        start = start + data_seg
    return numpy.mean(numpy.array(errors), axis=0)


def test_seg(d, m):
    errors = []
```

```
    start = 0
    while True:
        end = start + known_length + 1 + predict_length
        if end > data_seg:
            break

        errors.append(test_single(d[start:end], m))
        start = start + predict_length
    return numpy.mean(numpy.array(errors), axis=0)


def test_single(d, m):
    diff = difference(d)
    scaled = scaler.transform(diff)

    x = scaled[:known_length].reshape(1, known_length, data_dim)

    y = scaler.inverse_transform(m.predict(x)[0].reshape(-1, data_dim))

    prediction = numpy.zeros((predict_length, data_dim))
    prediction[0] = d[known_length] + y[0]
    for i in range(1, predict_length):
        prediction[i] = prediction[i - 1] + y[i]

    return numpy.sum(numpy.power(prediction - d[known_length + 1:], 2),
        axis=1)


data_end = data_start + data_length

scaler = gen_scaler(min_, max_)

data = extract_file_content_binary_float64(get_scm_train_file_name(),
    (data_length, 2))
data = data[data_start:data_end, :data_dim]

model = build_model()
```

```
train_start = 0
train_end = train_length

train(data[train_start:train_end], model, e=init_epochs)

while True:
    test_start = train_end + train_time
    test_end = test_start + train_time
    if test_end >= data_end:
        break
    print(','.join(test(data[test_start:test_end], model).astype(str)))

    train_start = train_start + train_time
    train_end = train_end + train_time
    train(data[train_start:train_end], model, e=epochs)
```

Bibliography

[1] Marco Altini, Davide Brunelli, Elisabetta Farella, and Luca Benini. Bluetooth indoor localization with multiple neural networks. *IEEE 5th International Symposium on Wireless Pervasive Computing 2010*, pages 295 – 300, 2010.

[2] J.B. Andersen, J. Jensen, S.H. Jensen, and F. Frederiksen. Prediction of future fading based on past measurements. *Gateway to 21st Century Communications Village. VTC 1999-Fall. IEEE VTS 50th Vehicular Technology Conference*, pages 151 – 155, 1999.

[3] Anthea Wain Sy Au, Chen Feng, Shahrokh Valaee, Sophia Reyes, Sameh Sorour, Samuel N. Markowitz, Deborah Gold, Keith Gordon, and Moshe Eizenman. Indoor tracking and navigation using received signal strength and compressive sensing on a mobile device. *IEEE Transactions on Mobile Computing*, 12:2050–2062, 2013.

[4] P. Bahl and V. N. Padmanabhan. RADAR: An in-building RF-based user location and tracking system. *IEEE Infocom*, 2000.

[5] A. Besada, A.M. Bernardos, P. Tarrio, and J.R. Casar. Analysis of tracking methods for wireless indoor localization. *Proc. Second Intl Symp. Wireless Pervasive Computing (ISWPC 07)*, pages 493–497, Feb 2007.

[6] DENNY BRITZ. Recurrent neural networks tutorial, part 1 introduction to rnns. `http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/`, September 2015. Last accessed on 2018-09-06.

[7] Hao Chen, Yifan Zhang, Wei Li, Xiaofeng Tao, and Ping Zhang. Confi: Convolutional neural networks based indoor wi-fi localization using channel state information. *IEEE Access*, 5:18066 – 18074, 2017.

[8] S. Chen, G. J. Gibson, C. F. N. Cown, and P. M. Grant. Adaptive equalization of finite non-linear channels using multilayer perceptrons. *Signal Process*, 20:107 – 119, 1990.

[9] Wei Chen, Qiang Chang, Hong tao Hou, and Wei ping Wang. A novel clustering and kwnn-based strategy for wi-fi fingerprint indoor localization. *2015 4th International Conference on Computer Science and Network Technology (ICCSNT)*, pages 49 – 52, 2015.

[10] Jess Cid-Sueiro and Anbal R. Figueiras-Vidal. Digital equalization using modular neural networks: an overview. *Signal Processing in Telecommunications*, pages 337 – 345, 1996.

[11] Tianben Ding and Akira Hirose. Fading channel prediction based on combination of complex-valued neural networks and chirp z-transform. *IEEE Transactions on Neural Networks and Learning Systems*, 25:1686 – 1695, 2014.

[12] A. Duel-Hallen, Shengquan Hu, and H. Hallen. Long-range prediction of fading signals. *IEEE Signal Processing Magazine*, 17:62 – 75, 2000.

[13] Alexandra Duel-Hallen. Fading channel prediction for mobile radio adaptive transmission systems. *Proceedings of the IEEE*, 95:2299 – 2313, 2007.

[14] Shih-Hau Fang and Tsung-Nan Lin. Indoor location system based on discriminant-adaptive neural network in ieee 802.11 environments. *IEEE Transactions on Neural Networks*, 19:1973 – 1978, 2008.

[15] A. Fehske, J. Gaeddert, and J.H. Reed. A new approach to signal classification using spectral correlation and neural networks. *First IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks*, 2005.

[16] Tobias Gruber, Sebastian Cammerer, Jakob Hoydis, and Stephan ten Brink. On deep learning-based channel decoding. *51st Annual Conference on Information Sciences and Systems (CISS)*, 2017.

[17] H. Hallen, A. Duel-Hallen, Shengquan Hu, Tung-Shen Yang, and Ming Lei. A physical model for wireless channels to provide insights for long range prediction. *MILCOM 2002. Proceedings*, 1:627 – 631, 2002.

[18] A. Heidari, A.K. Khandani, and D. Mcavoy. Adaptive modelling and long-range prediction of mobile fading channels. *IET Communications*, 4:39 – 50, 2010.

[19] Wen-hao Ying Huan Dai and Jiang Xu. Multi-layer neural network for received signal strength-based indoor localisation. *IET Communications*, 10:717 – 723, 2016.

[20] Bartosz Jachimczyk, Damian Dziak, and Wlodek J. Kulesza. Performance improvement of nn based rtls by customization of nn structure - heuristic approach. *2015 9th International Conference on Sensing Technology (ICST)*, pages 278 – 283, 2015.

[21] Yajun Gu Lingwen Zhang, Yishun Li and Wenkao Yang. An efficient machine learning approach for indoor localization. *China Communications*, 14:141 – 150, 2017.

[22] Hui Liu, Houshang Darabi, Pat Banerjee, and Jing Liu. Survey of wireless indoor positioning techniques and systems. *IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews*, 37:1067 – 1080, 2007.

[23] Lihong Liu, Hui Feng, Tao Yang, and Bo Hu. Mimo-ofdm wireless channel prediction by exploiting spatial-temporal correlation. *IEEE Transactions on Wireless Communications*, 13:310 – 319, 2014.

[24] Yuan Lukito and Antonius Rachmat Chrismanto. Recurrent neural networks model for wifi-based indoor positioning system. *2017 International Conference on Smart Cities, Automation & Intelligent Computing Systems (ICON-SONICS)*, pages 121 – 125, 2017.

[25] Changqing Luo, Jinlong Ji, Qianlong Wang, Xuhui Chen, and Pan Li. Channel state information prediction for 5g wireless communications: A deep learning approach. *IEEE Transactions on Network Science and Engineering*, 2018.

[26] Arun Mallya. Lstm forward and backward pass. `http://arunmallya.github.io/writeups/nn/lstm/index.html#/`. Last accessed on 2018-09-06.

[27] Tom Michael Mitchell. *Machine Learning*, chapter Artificial Neural Network, pages 81 – 127. McGraw-Hill Education, March 1997.

[28] Eliya Nachmani, Yair Be'ery, and David Burshtein. Learning to decode linear codes using deep learning. *54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2016.

[29] Florian Gain Nicolas Le Dortz and Per Zetterberg. Wifi fingerprint indoor positioning system using probability distribution comparison. *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2301 – 2304, 2012.

[30] Christopher Olah. Understanding lstm networks. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`, August 2015. Last accessed on 2018-09-06.

[31] Timothy OShea and Jakob Hoydis. An introduction to deep learning for the physical layer. *IEEE Transactions on Cognitive Communications and Networking*, 3:563 – 575, 2017.

[32] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, 2013.

[33] Meetha V. Shenoy, K. R. Anupama, Narayan Manjrekar, and Mustafa Murtaza Shamsi. Indoor localization in nlos conditions using asynchronous wsn and neural network. *2017 International Conference on Communication and Signal Processing (ICCSP)*, pages 2130 – 2134, 2017.

[34] Ke Shi, Z.Ma, R.Zhang, W.Hu, , and H.Chen. Support vector regression based indoor location in IEEE 802.11 environments. *Mobile Information Systems*, 2015.

[35] Sevil Tuncer and Taner Tuncer. Indoor localization with bluetooth technology using artificial neural networks. *2015 IEEE 19th International Conference on Intelligent Engineering Systems (INES)*, pages 213 – 217, 2015.

[36] Joel Vanderpypen and Laurent Schumacher. Mimo channel prediction using esprit based techniques. *2007 IEEE 18th International Symposium on Personal, Indoor and Mobile Radio Communications*, pages 1 – 5, 2007.

[37] Wee-Seng Soh Wendong Xiao, Peidong Liu and Guang-Bin Huang. Large scale wireless indoor localization by clustering and extreme learning machine. *2012 15th International Conference on Information Fusion*, pages 1609 – 1614, 2012.

[38] Lingjun Gao Xuyu Wang and Shiwen Mao. Csi phase fingerprinting for indoor localization with a deep learning approach. *IEEE Internet of Things Journal*, 3:1113 – 1123, 2016.

[39] Arumugam Nallanathan Yaqin Xie, Yan Wang and Lina Wang. An improved K-Nearest-Neighbor indoor localization method based on Spearman distance. *IEEE Signal Processing Letters*, 23:351 – 355, 2016.

[40] Hao Ye, Geoffrey Ye Li, and Biing-Hwang Juang. Power of deep learning for channel estimation and signal detection in ofdm systems. *IEEE Wireless Communications Letters*, 7:114 – 117, 2018.

[41] Thomas Zemen, Christoph F. Mecklenbrauker, Florian Kaltenberger, and Bernard H. Fleury. Minimum-energy band-limited predictor with dynamic subspace selection for time-variant flat-fading channels. *IEEE Transactions on Signal Processing*, 55:4534 – 4548, 2007.

[42] Han Zou, Ming Jin, Hao Jiang, Lihua Xie, and Costas J. Spanos. WinIPS: WiFi-based non-intrusive indoor positioning system with online radio map construction and adaptation. *IEEE Transactions on Wireless Communications*, 16:8118–8130, Dec 2017.