Investigation of Neural ODE LSTM for RSSI Indoor Localization

by

Charles Chang

B.A.Sc., Simon Fraser University, 2019

A Report Submitted in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF ENGINEERING

in the Department of Electrical Computer Engineering

Investigation of Neural ODE LSTM for RSSI Indoor Localization

by

Charles Chang

B.A.Sc., Simon Fraser University, 2019

Supervisory Committee

_____

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

_____

Dr. Aaron Gulliver, Departmental Member
(Department of Electrical and Computer Engineering)

ABSTRACT

Traditionally, indoor localization using received signal strength indicator (RSSI) measurements ignores the elapsed time between positions. This project investigates the use of elapsed time between positions as a feature with RSSI measurements for indoor localization. We use the recently proposed neural ordinary differential equations long short-term memory (ODE-LSTM) to handle the varying time gaps between each position. Our experiment compares the performance of the bidirectional LSTM (BiLSTM) model and the bidirectional ODE-LSTM (Bi-ODE-LSTM) model. The result shows that using time as a training feature for RSSI indoor localization can improve the accuracy. However, the benefit of the ODE-based model decreases as the number of RSSI features increases. Finally, the benefit brought by the Bi-ODE-LSTM model should be taken into account for the extra model complexity in practical applications.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

APs Access Points

AdaGrad Adaptive Gradient Algorithm

ANNs Artificial Neural Networks

BiLSTM Bidirectional Long-Short Term Memory

Bi-ODE-LSTM Bidirectional ODE Long-Short Term Memory

BiRNN Bidirectional RNN

BLE Bluetooth Low Energy

CSI Channel State Information

CNNs Convolutional Neural Networks

ECDF Empirical Cumulative Distribution Function

FNNs Feedforward Neural Networks

GPS Global Positioning System

GRU Gated Recurrent Unit

LSTM Long-Short Term Memory

MAE Mean Absolute Error

MLP Multi-Layer Perceptron

MSE Mean Squared Error

ODE Ordinary Differential Equations

ReLU Rectified Linear Unit

ResNet Residual Neural Network

RMSE Root Mean Square Error

RMSProp Root Mean Square Propagation

RNN Recurrent Neural Network

RSSI Received Signal Strength Indicator

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. Xiaodai Dong, for guiding me throughout my master's program and this project. I also would like to thank Brosnan Yuen for suggesting the topic for this project.

I would like to also thank my family for supporting me during my time at UVic.

Chapter 1

Introduction

Localization technology such as the widely adapted Global Positioning System (GPS) enables the positioning and tracking of electronic devices. Moreover, the advancement in electronics has allowed GPS, Wi-Fi, and Bluetooth technology to be integrated into a single device, like smartphones. Although GPS is the most common position tracking technology, it is limited to the outdoor environment because GPS signals require line-of-sight toward the satellites in the sky. Therefore, indoor localization uses radio wave signals, namely Wi-Fi or Bluetooth signals. Over the years there has been a wide range of applications for indoor localization in, for example, shopping malls [1] and contact tracing for the recent covid pandemic [2]. Wi-Fi fingerprinting [3] and Bluetooth Low Energy (BLE) using a received signal strength indicator (RSSI) has become popular choice because of the wide range adaption of Wi-Fi and BLE technology in personal electronics devices [4]. RSSI indoor localization uses the RSSI measurement to predict the corresponding position of the device. This approach ignores the elapsed time between positions. The purpose of this project is to investigate the use of elapsed time between each position as a feature with RSSI measurements for indoor localization. Because the speed of the device moving through the space varies, the elapsed time between each position also varies. Traditionally, data with varying time gaps are divided equally to fit the recurrent neural network (RNN) model. The recent neural ordinary differential equations (ODE) [5] and ODE-RNN [6] paper proposed to incorporate the irregular time gap between samples to train the neural network. Our goal is to examine the use of elapsed time by comparing the RNN model with the ODE-RNN model. We will be using the bi-directional ODE long-short term memory (Bi-ODE-LSTM) model, a variation of ODE-RNN.

This report starts by discussing the background information in the proposed Bi-ODE-LSTM model. Chapter 2 begins with the artificial neural network basics followed by the RNN and LSTM. Finally, we will explain the Neural ODE and the intuition behind the architecture.

Chapter 3 focuses on the proposed Bi-ODE-LSTM algorithm and the experiments. This chapter begins with related work in ODE-RNN followed by explaining indoor localization. Then, the dataset used in the experiment and the baseline algorithm, BiLSTM, and the proposed algorithm, Bi-ODE-LSTM will be introduced. Lastly, we will detail the experiment setup and the results with discussion on the proposed algorithm.

Chapter 4 finishes the report with conclusion remarks.

Chapter 2

Background

2.1 Artificial Neural networks

The artificial neural networks (ANNs), inspired by the neurological connection in the human brain, are the building block of modern deep learning algorithms. ANNs contain three parts, an input layer, one or many hidden layer(s), and an output layer. The neural network performs calculations for the incoming inputs throughout the neurons in the network and produces the output in the output layers. The three main types of neural networks are feedforward neural networks (FNNs) or multi-layer perceptron (MLP), convolutional neural networks (CNNs), and recurrent neural networks (RNNs). Fig. 2.1 shows the structure of an FNN. The input layer is a single input with "N" feature(s), hidden layers contain "N" layers and "J" neuron(s), and the output with "N" variable(s). The number of neurons for each layer can vary depending on the model.

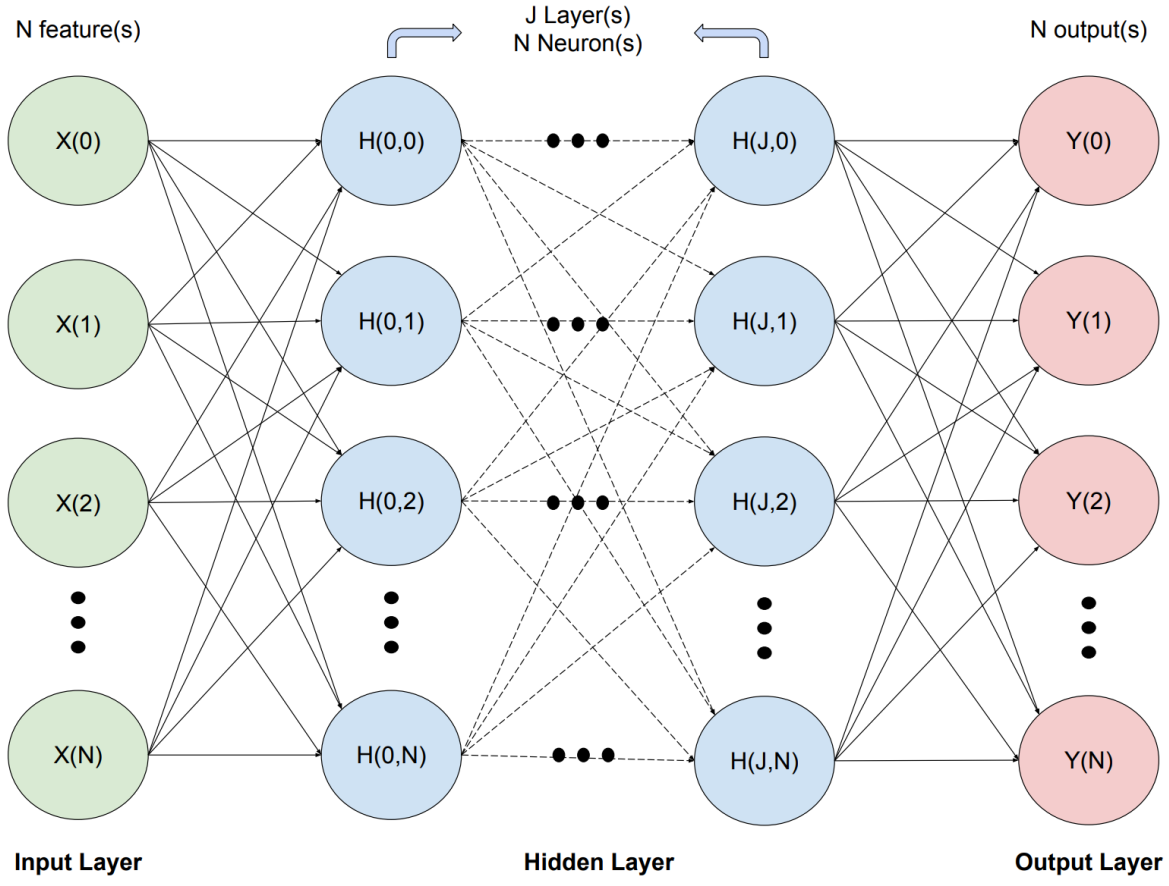


Figure 2.1: Feedforward neural network

2.1.1 Neuron Breakdown

Neurons make up the neural network. A neuron is a mathematical function that takes the weighted sum of the input and computes the neuron's output using an activation function. Fig. 2.2 is a simplified model of the biological neuron on the left and the mathematical model on the right. The input signal multiplies with the weights at the synapse. The product follows the dendrites and reaches the cell body. The axon sums up the inputs and calculates to generate the output signal. The signal is then branched to other neurons. The perceptron, a threshold activation function, is the earliest form of neuron in the neural network. If the sum of the inputs exceeds a threshold, the output will be 1 or it will stay at 0 [7]. Modern neural network neurons use nonlinear activation functions such as the sigmoid function to handle nonlinear data [8].

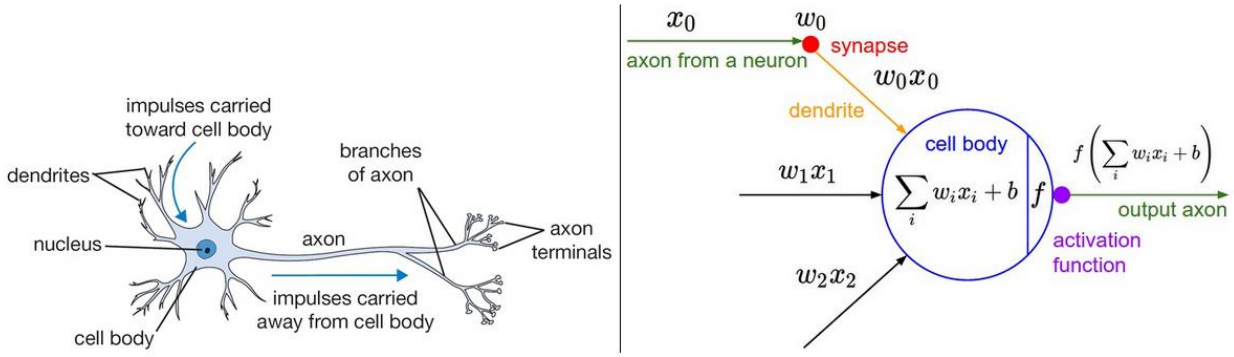


Figure 2.2: Biological neuron (left), Mathematical model of neuron (right) [9, Fig. 1]

2.1.2 Activation Function

Sigmoid and tanh are common non-linear activation functions used in the feedforward network. Sigmoid activation function scales to form tanh. The output of tanh is between -1 to 1. Unlike sigmoid where the output is between 0 and 1, tanh is zero-centred, which makes training smoother by avoiding weight jumping between positive and negative values during the weight updates [9]. Like sigmoid, tanh also suffers from the same vanishing gradient problem. The graph and the equation of sigmoid and tanh are shown in Figs. 2.3 and 2.4.

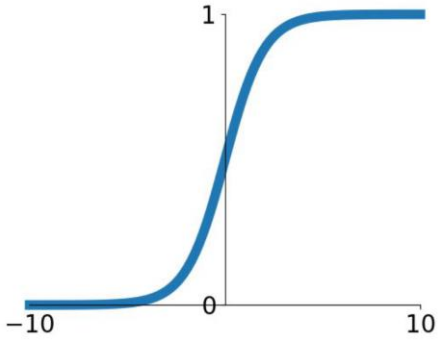


Figure 2.3: Sigmoid function [10, pp. 15]

$$\sigma(x) = \frac{1}{(1+e^{-x})} \qquad \textit{Sigmoid Function } (1)$$

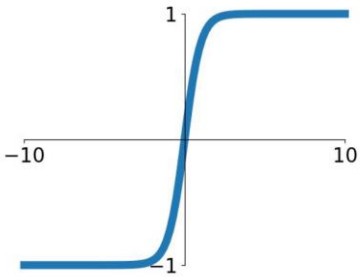


Figure 2.4: Tanh function [10, pp. 15]

$$\tanh(\mathrm{x}) = 2\sigma(2x) - 1 \tag{2}$$

Leaky ReLU, Fig. 2.6, is an improved version of Rectified Linear Unit (ReLU) function, Fig. 2.5. It addresses the dying ReLU problem. When the input is negative, the output is zero, which means the weights will never be updated. It will never recover from this because the gradient is zero when the input is negative. Leaky ReLU attempts to solve this problem by adding a small slope in the negative x area. Like the ReLU function, leaky ReLU does not need an exponential function which is less efficient to compute. Also, leaky ReLU converges faster compared to sigmoid and tanh and does not suffer from vanishing gradient problems in sigmoid and tanh [10].

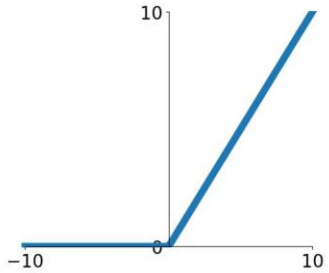


Figure 2.5: Rectified Linear Unit (ReLU) function [10, pp. 15]

$$f(x) = \max(0, x) \qquad \textit{ReLU} \tag{3}$$

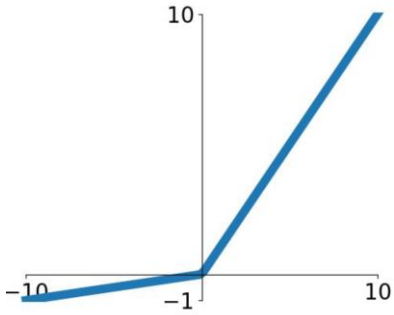


Figure 2.6: Leaky ReLU function [10, pp. 15]

$$f(x) = \max(0, x) + \alpha \min(0, x), \alpha = slope \qquad \textit{Leaky ReLU} \tag{4}$$

2.1.3 Loss Function

The loss function evaluates the accuracy of the neural networks by comparing the predicted output with the expected output during training. The goal is to minimize the loss function to train the algorithm to learn from the training data. Then, test the accuracy of the neural network using test data. Overfitting happens when the neural network learns the features specific to the training data. Test data ensures the neural network learns the features of the problem dataset and thus prevents overfitting. There are many different types of loss functions for different purposes [11]. Since our model is a regression problem, we will use the mean squared error (MSE) function [11]:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{5}$$

2.1.4 Optimization

The first step to minimizing the loss function is to calculate the gradients of the loss function with respect to each individual weight. We calculate the gradient using the chain rule, by backpropagating back through the network [12]. Fig. 2.7 is an example of calculating the gradient of the loss function, L, with respect to the weights of the neurons, $\frac{\partial L}{\partial w_x}$ and $\frac{\partial L}{\partial w_y}$. Next, the optimization algorithm uses the gradient to find the weights that minimize the loss function. Adam algorithm, Fig. 2.8, is computationally efficient and works well in large and sparse gradient datasets [13]. Adam combines the advantage of the Adaptive Gradient Algorithm (AdaGrad) optimizer and Root Mean Square Propagation (RMSProp) optimizer with momentum [13]. We will be adopting Adam as our optimization algorithm with a learning rate of 0.001 for the best result.

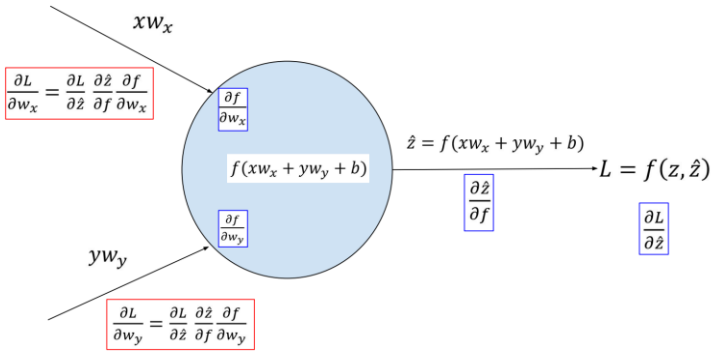


Figure 2.7: Backpropagation example [14, pp. 29]

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
  dx = compute_gradient(x)
  first_moment = beta1 * first_moment  + (1 - beta1) * dx
  second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
  first_unbias = first_moment / (1 - beta1 ** t)
  second_unbias = second_moment / (1 - beta2 ** t)
  x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7))
```

Momentum

Bias correction

AdaGrad / RMSProp

Figure 2.8: Adam optimization algorithm [15, pp. 37]

2.2 Recurrent Neural Networks

Unlike common feedforward neural networks in which output depends on one input, RNN receives input from the past and the present [16]. The networks preserve information from the past to make a prediction in the present. RNN enables machine learning algorithms to handle sequence data, like natural language processing and text generation. Fig. 2.9 below shows the basic flow of RNN. The output is fed into the input of the next state and so on. The A in the box represents an RNN cell, a mathematical operation. Vanilla RNN is illustrated in Fig. 2.10 and Eq. (6). The output of vanilla RNN cell is the weighted sum of current input and the output of the last state.

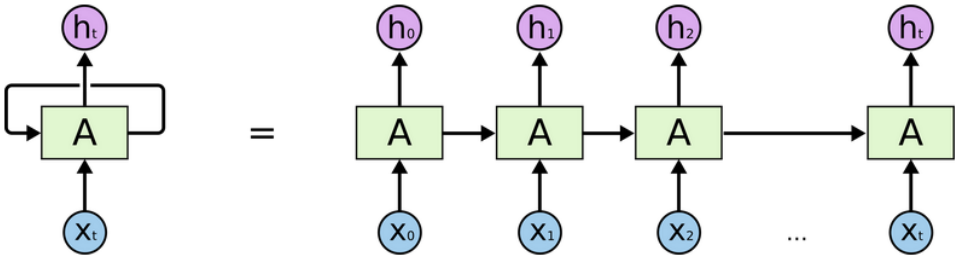


Figure 2.9: RNN structure [16, Fig. 2]

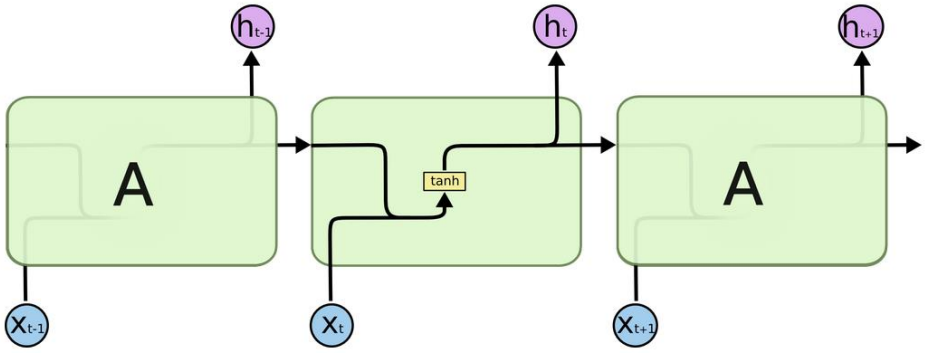
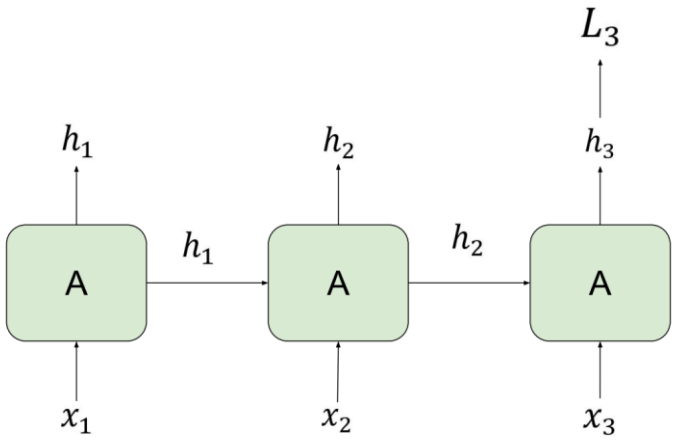


Figure 2.10: Vanilla RNN [16, Fig. 5]

$$h_t = \tanh\left(w_{xh}x_t + w_{hh}h_{t-1} + b\right) \tag{6}$$

One of the drawbacks of vanilla RNN is the vanishing and exploding gradient problem [17]. The network performs the chain rule to find the gradient with respect to the earlier state, illustrated in Fig. 2.11. L_3 is the loss function at the third input and the gradient of L_3 with respect to h_1 is $\frac{\partial L_3}{\partial h_3} \times \frac{\partial h_3}{\partial h_2} \times \frac{\partial h_2}{\partial h_1}$. If the partial derivatives are small or large, the gradient can become very large or very small. The mathematical proof in Eq. (7-9) shows the increase in the multiplication of, w_{hh}^{n-i}, as the n, the latest state, and i, the earlier state becomes further apart. If the weight is smaller than one, the gradient is going to vanish as it gets smaller and smaller. If the weight is greater than one, it will get larger and larger. As a result, RNN has a hard time learning long-term dependency with vanish gradient problems. On the other hand, the exploding gradient causes the optimization algorithm to take large steps and unable to learn the weights [18].



$$\frac{\partial L_3}{\partial h_1} = \frac{\partial L_3}{\partial h_3} \times \frac{\partial h_3}{\partial h_2} \times \frac{\partial h_2}{\partial h_1}$$

Figure 2.11: RNN backpropagation

$$\frac{\partial h_t}{\partial h_{t-1}} = \tanh'(w_{xh}x_t + w_{hh}h_{t-1} + b)\, w_{hh} \tag{7}$$

$$\frac{\partial L_n}{\partial h_i} = \frac{\partial L_n}{\partial h_n}\left(\prod_{i<t\leq n} \frac{\partial h_t}{\partial h_{t-1}}\right) \tag{8}$$

$$\frac{\partial L_n}{\partial h_i} = \frac{\partial L_n}{\partial h_n}\left(\prod_{i<t\leq n} \tanh'(w_{xh}x_t + w_{hh}h_{t-1} + b)\right) w_{hh}^{n-i} \tag{9}$$

2.2.1 Long-Short Term Memory

LSTM was proposed to minimize the risk of vanishing and exploding gradient problem in the vanilla RNN [16]. The key contribution is the additional cell state, c_t, that stores long-term information. The cell state is modified through various gates in the LSTM cell. Fig. 2.12 illustrates the internal structure of an LSTM cell. Eq. (10-15) are for the LSTM cell. The forget gate decides if the information coming from the previous hidden state and the input of the current state is discarded or kept. Since the activation function is sigmoid, between 0 and 1, if forget gate output is 1 the information is kept completely; thrown away if it is 0. Next, the input gate controls the new information added to the cell, and the "gate" gate is a tanh layer that is the new information to the cell. The output gate decides what information to output. Finally, the cell state is updated using the information from the forget, input, last cell state, and "gate" gate. The hidden state uses the output gate and cell state to calculate the output of the LSTM cell.

$$f_t = \sigma\left(w_{xf}x_t + w_{hf}h_{t-1} + b_f\right) \quad \textit{Forget gate} \tag{10}$$

$$i_t = \sigma(w_{xi}x_t + w_{hi}h_{t-1} + b_i) \quad \textit{Input gate} \tag{11}$$

$$o_t = \sigma(w_{xo}x_t + w_{ho}h_{t-1} + b_o) \quad \textit{Output gate} \tag{12}$$

$$g_t = \tanh\left(w_{xg}x_t + w_{hg}h_{t-1} + b_i\right) \quad \textit{"Gate" gate} \tag{13}$$

$$c_t = f_t * c_{t-1} + i_t * g_t \qquad \textit{Cell state} \ (14)$$

$$h_t = o_t * \tanh(c_t) \qquad \textit{Hidden state} \ (15)$$

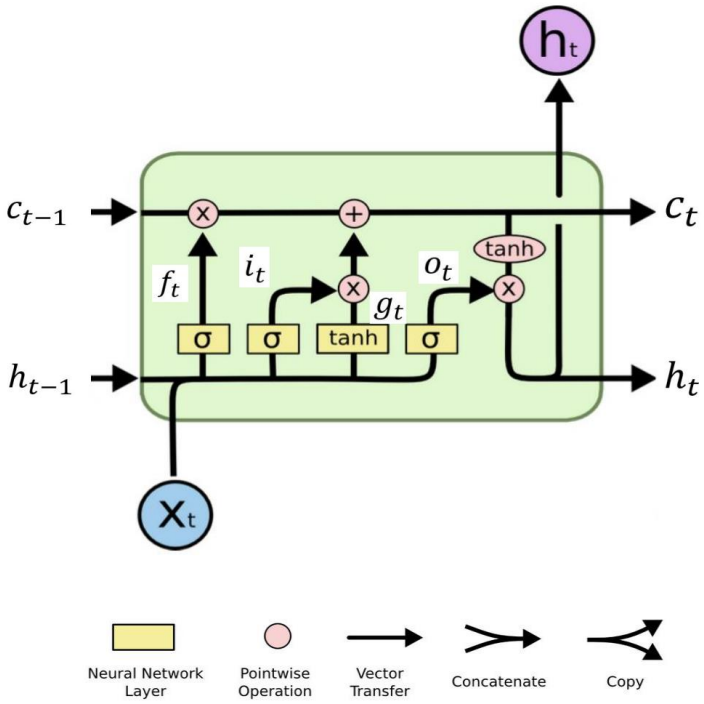


Figure 2.12: LSTM structure [16, Figs. 6-7]

2.2.2 Bidirectional RNN

One variation of the vanilla RNN is the Bidirectional RNN (BiRNN). Vanilla RNN leverage the information of the past to make the current prediction. Bidirectional RNN, on the other hand, uses information from the future and the past for current prediction. By running forward states RNN and backward states RNN, we can construct a BiRNN for any type of RNN. Fig. 2.13 shows the structure of BiRNN where h_0 is the forward hidden state, and $h_0{}'$ is the backward hidden state. BiRNN is only possible if the data set has the entire sequence [18]. In this report, we will be using BiRNN.

2.2.3 Multilayer RNN

In addition to running a backward RNN on top of a forward RNN, we can also stack multiple layers of RNN cell together to form multilayer RNN. The output of first layer is the input for the next layer, and so on. The hidden state progress to the next states in the same layer. Deep neural network can improve the performance for complex dataset, but there is always a possibility of overfitting if the network is too deep. Fig. 2.14 below is a three-layer RNN. The RNN cell is the green box, and the blue and red are the input and output.

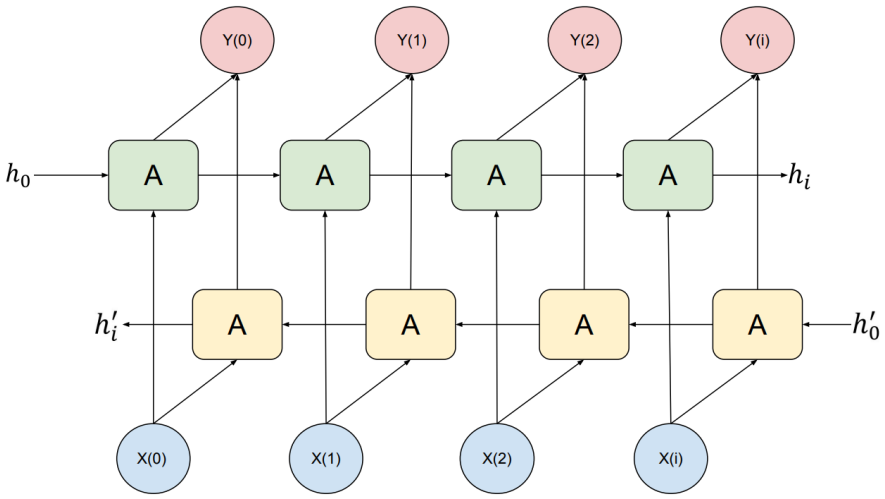


Figure 2.13: Bidirectional RNN structure

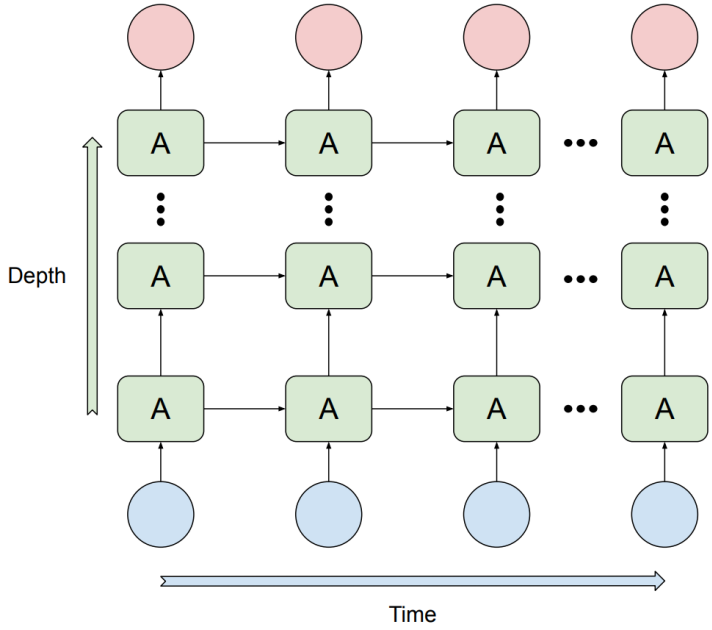


Figure 2.14: Multilayer RNN structure

2.3 Neural ODE

Before jumping straight into Neural ODE, we will first build the intuition behind Neural ODE through the differential equation and the residual neural network.

2.3.1 Ordinary Differential Equation (ODE)

The ODE describes how one variable changes with respect to another variable. We are interested in how the function evolves. The solution of the ODE is a function that satisfies the original equation. In this report, we will focus on first-order differential equations, which means the largest derivative in the ODE function

is first order. First-order ODE can be solved analytically or numerically. One of the numerical methods to solve the first order initial value problem (IVP), Eq. (16-17), is the Euler's method, Eq. (18). Euler's method works by calculating the tangent line of the initial point, t_0, and a next point, t_n, on the tangent line. Once we have t_n, we can calculate for t_{n+1} and so on. By setting the next point close to the initial point, we can estimate the actual solution. Fig. 2.15 illustrates the concept of Euler's method.

$$\frac{dy}{dt} = f(y,t) \tag{16}$$

$$y(t_0) = y_0 \tag{17}$$

$$y_{n+1} = y_n + f(y_n, t_n)(t_{n+1} - t_n) \tag{18}$$

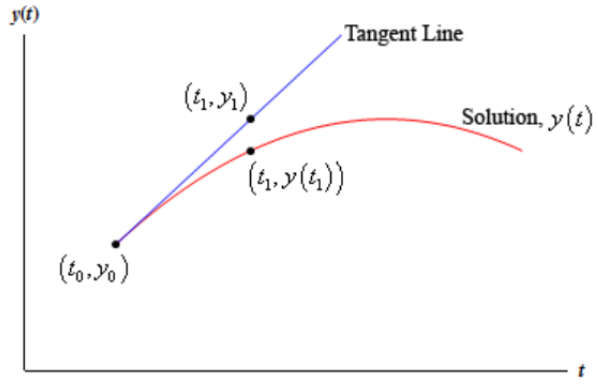


Figure 2.15: Euler's method [19, Fig. 1]

2.3.2 Residual Neural Networks

One of the problems with deep neural networks is the vanishing gradient problem. A residual neural network (ResNet) is a neural network architecture developed to mitigate vanish gradient problems in a deep neural network [20]. Unlike an FNN, where input is always transformed through the layers, Eq. (19), residual neural network implements skip connection, illustrated in Eq. (20) and Fig. 2.16 (right). A skip connection is a shortcut where the input can skip a layer or multiple layers in the network and rejoin at a later layer. The skip connection allows information to travel deeper into the network without getting lost. A residual neural network can be stacked to form a deep network, and largely avoid vanishing gradient problems [21]. With a deeper neural network, the system has a better chance of learning the parameter of the data and thus resulting in better performance.

$$h_{t+1} = f(h_t) \tag{19}$$

$$h_{t+1} = f(h_t) + h_t \tag{20}$$

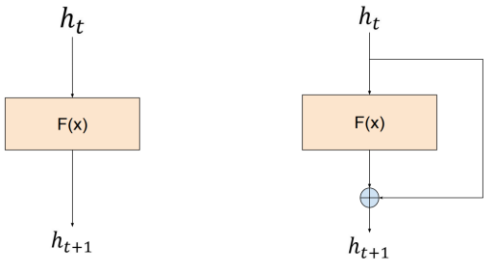


Figure 2.16: Building block of neural network: Feedforward network (left), Residual network (right)

2.3.3 Neural Ordinary Differential Equations

We can see from ResNet and Euler's method that they resemble each other. A series of a ResNet blocks is the solution to the ODE using Euler's method [5]. The t_n in Euler's method represents the layers in the Neural ODE. The input to the Neural ODE is the initial condition, Eq. (21), of the IVP. Neural ODE network is represented by Eq. (23), where t is the layers and θ is the parameters shared among each layer. The final condition Eq. (22) is the output of the Neural ODE. We can solve the ODE using adaptive or fixed-step solvers. Unlike a ResNet, where layers are connected in discrete time sequence, Neural ODE is a continuous-depth model [5]. Eq. (21-23) summarized the Neural ODE equation. Fig. 2.17 illustrates the structure of the residual network on the left and the Neural ODE network on the right. Neural ODE combines the function in the residual network into one function and uses t as the layers in the network.

$$x = \boldsymbol{z}(t_0) \qquad \textit{Input to the network } (21)$$

$$y = \boldsymbol{z}(t_N) \qquad \textit{Output to the network } (22)$$

$$\frac{d\boldsymbol{z}(t)}{dt} = f(\boldsymbol{z}(t), t, \theta) \qquad \textit{Neural Network } (23)$$

$$L\big(\boldsymbol{z}(t_N)\big) = L\left(\boldsymbol{z}(t_0) + \int_{t_0}^{t_N} f(\,\boldsymbol{z}(t), t, \theta)\, dt\,\right) = L(ODESolve(\boldsymbol{z}(t_0), f, t_0, t_N, \theta)) \quad \textit{Loss function } (24)$$

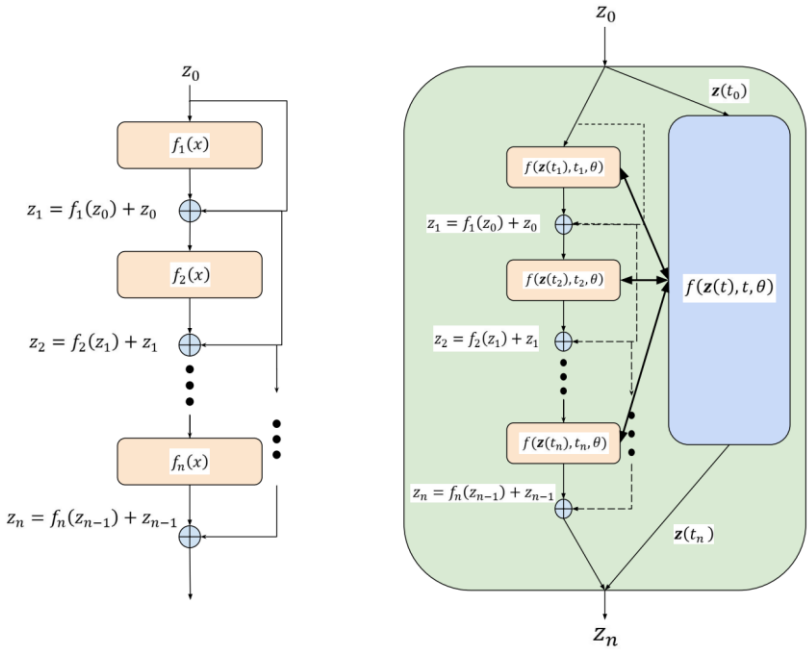


Figure 2.17: Structure of Residual Network (left) and Neural ODE (right)

Once we calculate the output and the loss function, Eq. (24), we can backpropagate through the ODE solver; however, it requires high memories [5]. An adjoint sensitivity method is proposed to minimize memory cost and reduce errors [5]. The proposed adjoint sensitivity method is to solve the gradient of the loss function with respect to the parameter at the initial state, t_0, using the ODE solver with the initial value of the loss function gradient at the later state t_N. The adjoint state at Eq. (25) refers to the gradient of loss function with respect to the hidden state $z(t)$. The adjoint dynamics in Eq. (26), is the derivative of the adjoint state. We can now run the ODE solver backward with the initial condition, the adjoint state, and the adjoint dynamics to solve for the adjoint state at t_0, Eq. (27). A similar method is employed to find the gradient to the other parameters $\frac{\partial L}{\partial \boldsymbol{z}(t_0)}$, $\frac{\partial L}{\partial \theta}$, $\frac{\partial L}{\partial t_0}$, and $\frac{\partial L}{\partial t_N}$. The full derivations are available in [5]. Fig. 2.18 illustrates the forward and backward pass of Neural ODE. From $z(t_0)$ we can solve for $z(t_n)$ at any time and the backward pass uses an adjoint state to solve for the gradients.

$$\boldsymbol{a}(t) = \frac{dL}{d\boldsymbol{z}(t)} \qquad \textit{Adjoint state} \ (25)$$

$$\frac{d\boldsymbol{a}(t)}{dt} = -\boldsymbol{a}(t)\ \frac{\partial f(\boldsymbol{z}(t),t,\theta)}{\partial \boldsymbol{z}(t)} \qquad \textit{Dynamics of adjoint state} \ (26)$$

$$\boldsymbol{a}(t_0) = \boldsymbol{a}(t_N) - \int_{t_N}^{t_0} \frac{d\boldsymbol{a}(t)}{dt}\, dt \ = \boldsymbol{a}(t_N) - \int_{t_N}^{t_0} \boldsymbol{a}(t)\ \frac{\partial f(\boldsymbol{z}(t),t,\theta)}{\partial \boldsymbol{z}(t)} \qquad (27)$$

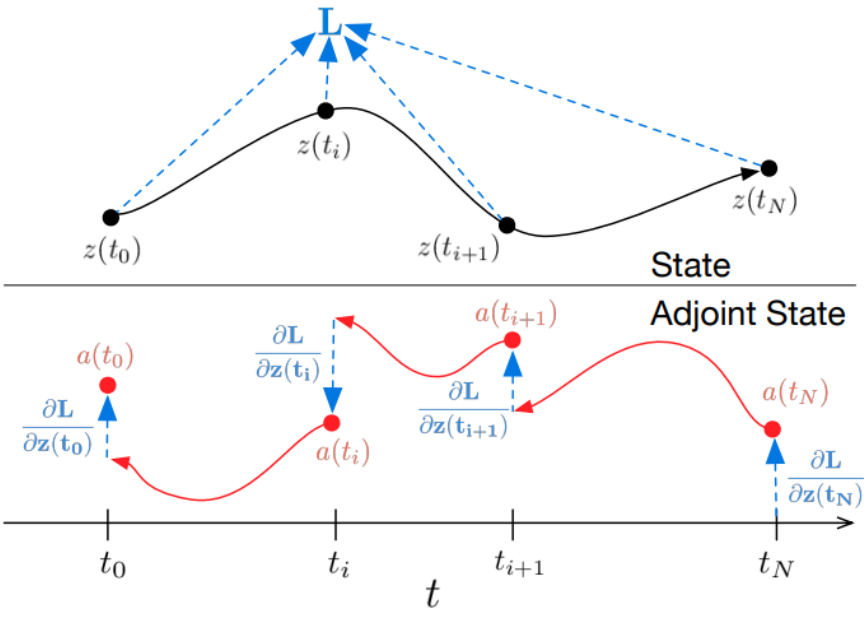


Figure 2.18: Neural ODE forward and backward pass [5, Fig. 2]

Chapter 3

Bi-directional ODE LSTM

3.1 Related Work

RNN is best known for handling even sampled sequences and regular time-gap data, such as language model. To handle irregular-sampled data, the data are divided into equal intervals for the RNN. This method ignores the time dimension and thus can affect the performance of the model [22], [5]. An ODE-RNN model [6] is introduced to incorporate Neural ODE [5] and RNN to use the time information in an irregularly sampled sequence data. The algorithm works by using Neural ODE to calculate the hidden state for the time gap between the previous and current sample, followed by an RNN cell updating the hidden state. Like the vanilla RNN suffers from vanishing and exploding gradient problem, ODE-RNN also has the same drawback [23]. The solution [23] proposed is to use LSTM followed by the Neural ODE to update the hidden state meanwhile the cell state is untouched through the whole sequence to preserve long-term information.

3.2 Indoor Localization

Indoor localization enables the tracking of devices in an indoor environment. There is a wide range of applications from shopping malls [1], smart homes [24], and contacting tracing during the covid pandemic [2]. Although GPS is a common practice in an outdoor environment, walls and ceilings interfere with the GPS signal thus making it unreliable. Wi-Fi and BLE are used in indoor localization because Wi-Fi and BLE are integrated into most personal devices. RSSI and channel state information (CSI) are the two common types of Wi-Fi and BLE fingerprinting. RSSI is popular among the two because RSSI is widely available in most receivers, despite suffers from fading and multipath [25]. On the other hand, CSI provides more information [1], but requires specific hardware [26].

3.3 Data Set

The raw dataset for training and testing was collected from an autonomous robot scanning for RSSI on the third floor in the Engineering Office Wing (EOW), University of Victoria [2]. The robot collects multiple samples of RSSI measurements at the 81 unique positions in the room. Each sample contains 24 RSSI readings. There are 11 Wi-Fi access points (APs) and 7 BLE APs. Each AP can have up to two RSSI measurements using 2.4 GHz and 5 GHz frequency bands. An algorithm loops through the raw dataset to generate unique trajectories for training and testing. The algorithm first randomly selects the initial position to be the start of a trajectory. Next, a second position is randomly selected. If the Euclidean distance between the two positions falls within a threshold, the second position is now part of the next position in the trajectory [2]. The process repeats until the number of positions is reached. The time increment of each position is calculated from the distance between the two positions and the walking speed between 0.4 m/s to 2 m/s [27], [28]. In the training data, there are a total of 20,000 unique trajectories with 20 positions in each trajectory. Although, in theory, the testing data is evenly sampled. In practice, there are going to be missing data or unusable measurements due to interference. Therefore, in testing the data

will reflect the irregular time gap between samples. The testing data includes one trajectory with 65 positions.

3.4 BiLSTM Model

The BiLSTM [2] is used to compare against the proposed Bi-ODE-LSTM model here. Initially, we used directional RNN and LSTM, but the result has a high variance. Furthermore, we also tested the bidirectional gated recurrent unit (GRU) and RNN, but there was no improvement. Therefore, BiLSTM was used. The BiLSTM model consists of two bidirectional LSTM layers and an output layer. The input features are the 24 RSSI readings with a batch size of 100 to speed up the training time. The hidden layer size is 168 for the LSTM. The first output layer is a linear layer with input from the output of the bidirectional LSTM, 2×168 features, and outputs 168 features. The second output linear layer is followed by a Leaky ReLU activation function and outputs the x and y positions. Fig. 3.1 is the algorithm flowchart for the BiLSTM algorithm.

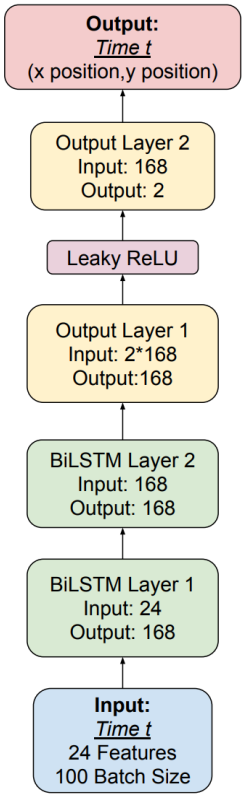


Figure 3.1: BiLSTM structure

3.5 Bi-ODE-LSTM Model

The proposed Bi-ODE-LSTM algorithm is based on ODE-LSTM [23] and BiLSTM algorithm [2], we added the backward states LSTM to make it a Bi-ODE-LSTM. Fig. 3.2 below is the overall architecture of the algorithm, and the pseudocode is in Appendix A. The number of input features to the ODE-LSTM cell is 24, the hidden state size is 168, and the batch size is 100. The input is a sequence of data from t_0 to t_n. In the forward

states, the input starts from the beginning of the sequence until the end of the sequence and the output for each time stamp is saved. The backward states start at the end of the sequence toward the start of the sequence. The output of the backward states for each timestamp is combined with the output of the forward states for the output layer. The ODE-LSTM cell, shown in Fig. 3.3 (left), consisting of two layers of ODE-LSTM and calculates the output for each time stamp. The size of the hidden state is selected from experimenting with different values. A larger hidden size leads to overfitting. And smaller hidden size produces worse results in training and testing because there are not enough parameters to learn from the data. Furthermore, directional ODE-LSTM was first attempted but the result was unusable due to high variance, so bidirectional was used. After the initial result from the Bi-ODE-LSTM, we replace LSTM with GRU, but it did not show better performance. Also, we tested training with irregular time features and testing with regular time features, and the result is worse compared to irregular elapsed time testing data.

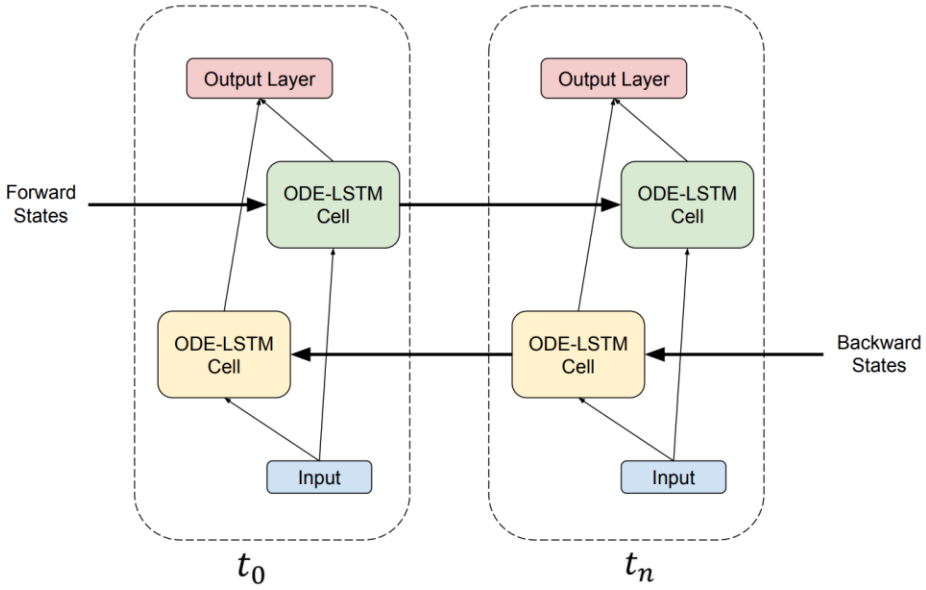


Figure 3.2: Bi-ODE-LSTM structure

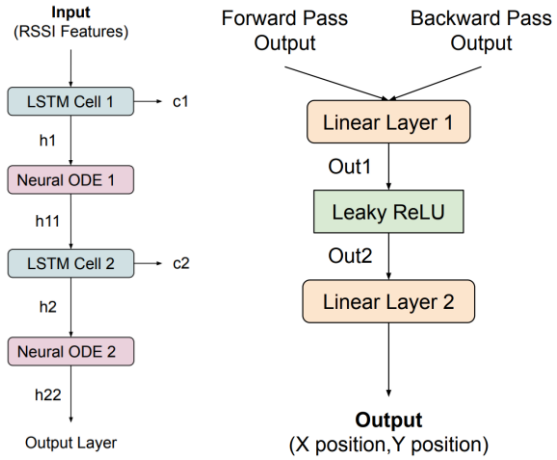


Figure 3.3: ODE-LSTM cell (left), Output layer (right)

The output layers are two linear layers with a Leaky ReLU activation function in between and output the x and y positions, as illustrated in Fig. 3.3 (right). The linear layer has the equation of $y = xw + b$. The y is the output of the layer and the x is the input. The w and the b are the weights and bias of the linear layer. The input size to the "Linear Layer 1" is 2×168 features with an output of 168 features. And the "Linear Layer 2" outputs the x and y positions. The Neural ODE uses hidden state output from the LSTM cell and the time stamp from the last position to solve for the current time stamp, illustrated in Fig. 3.4 (right). The Neural ODE uses a fixed-step solver, Fourth-order Runge-Kutta. The Neural ODE network is two linear layers with a tanh activation function, shown in Fig. 3.4 (left). The mathematical representation of the Neural ODE is shown in Eq. (28-30).

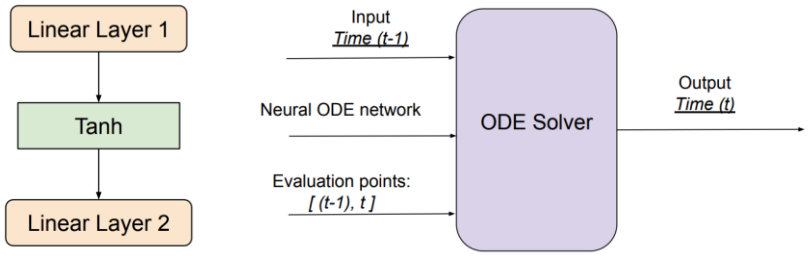


Figure 3.4: Neural ODE network (left), ODE solver (right)

$$\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), \theta) \qquad \textit{Neural ODE network } (28)$$

$$z(t_0) = h_{t-1} \qquad \textit{Input } (29)$$

$$z(t_N) = h_t \qquad \textit{Output } (30)$$

3.6 Experiment

3.6.1 Setup

The initial setup for the experiment is in Table 3.1. The experiment compares the performance of BiLSTM and Bi-ODE-LSTM using input with all 24 RSSI features and 12 RSSI features. The training data contain 20 positions in each of the 20,000 unique trajectories. The testing dataset is a trajectory with 65 positions. The Neural ODE uses the backpropagation and adjoint sensitivity method to analyze any performance difference. Time increment for each position is set by randomly choosing a walking speed between 0.4 m/s to 2 m/s.

Category	Value
Optimization algorithm	Adam
Learning rate	0.001
Loss Function	Mean Squared Error
Batch Size	100
ODE Solver	Fixed-step 4th-order Runge-Kutta

Table 3.1: Initial setup parameters for training

3.6.2 Experiment Result

The experiment is separated into two parts: using 24 RSSI features and 12 RSSI features. The table below summarizes the result of the experiment. The Backprop. uses backpropagation and the adjoint uses

adjoint sensitivity method for the Neural ODE. There is a total of 200 iterations per epoch. The models are evaluated using Mean Absolute Error (MAE) and Root Mean Square Error (RMSE).

Model	Number of RSSI Features	MAE (m)	RMSE (m)	Training time (s)	Epochs	Training Parameters
BiLSTM	24	0.636	0.918	752	2	997 K
Bi-ODE-LSTM + Backprop.	24	0.555	0.794	63627	2	1.5 M
Bi-ODE-LSTM + Adjoint	24	0.580	0.857	64483	2	1.5 M
BiLSTM	12	0.673	1.008	507	3	250K
Bi-ODE-LSTM + Backprop.	12	0.530	0.837	34737	4	386K
Bi-ODE-LSTM + Adjoint	12	0.595	0.887	36581	4	386K

Table 3.2: Experiment results

A) 24 RSSI Input Features:

The test results from training with all 24 RSSI features using the BiLSTM and the Bi-ODE-LSTM are shown in Table 3.2. The Bi-ODE-LSTM model has MAE and RMSE at 0.555 meters and 0.794, respectively, compared with BiLSTM at 0.636 meters and 0.981 meters. Bi-ODE-LSTM shows better outcomes but at the cost of longer training time because of the added Neural ODE network. The adjoint sensitivity method performs similarly to regular backpropagation with MAE at 0.580 meters and RMSE at 0.857 meters.

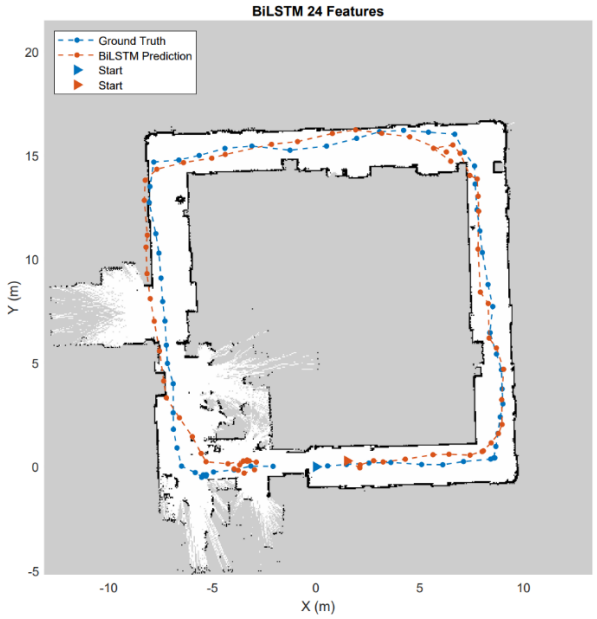


Figure 3.5: BiLSTM ground truth vs prediction path

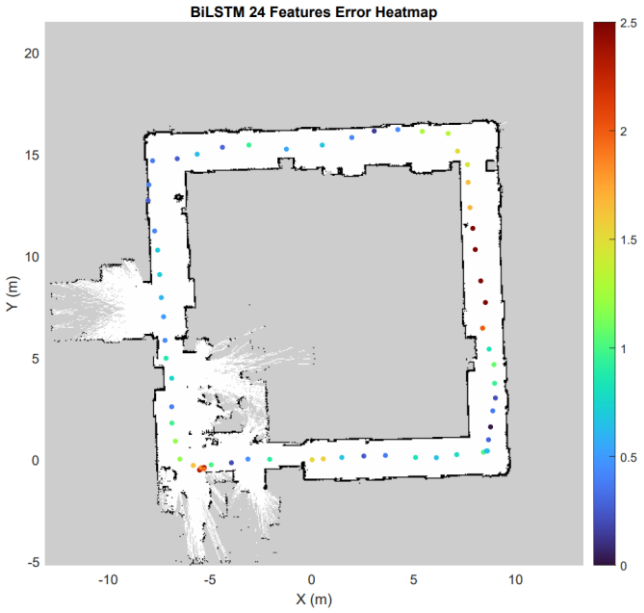


Figure 3.6: BiLSTM error heatmap

Figs. 3.5 and 3.6 are the path prediction of the BiLSTM model with 24 RSSI input features. Fig. 3.5 is the predicted path and the ground truth path. The error heatmap, Fig. 3.6, shows high errors in the east hallway, northeast, and southwest corners. The high errors may attribute to objects in the hallway or the location of the routers, which can cause weak or distorted signals. Bi-ODE-LSTM results in Fig. 3.7 and Fig. 3.8 show a similar trend. The northeast corners, southwest corners, and east hallway have higher errors compared to other parts of the hallway. The Empirical Cumulative Distribution Function (ECDF) plot shows that the BiLSTM has around 80% of errors less than 1.63 meters. On the other hand, the Bi-ODE-LSTM

model has around 1.45 meters. The max error for the BiLSTM is 3.56 meters and 2.82 meters for the Bi-ODE-LSTM.

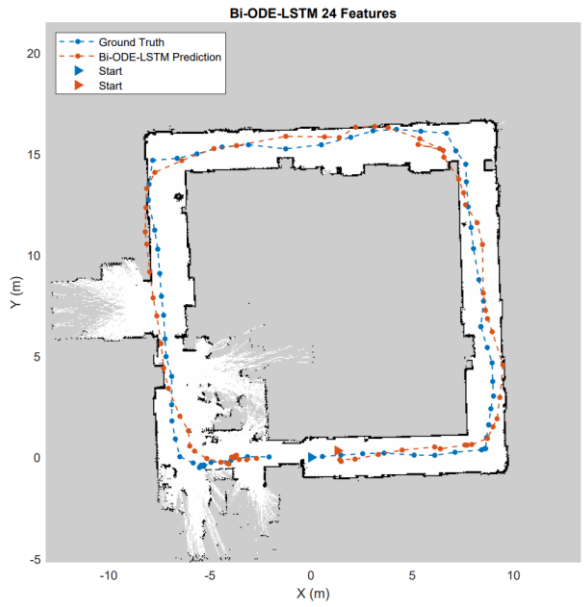


Figure 3.7: Bi-ODE-LSTM ground truth vs prediction paths

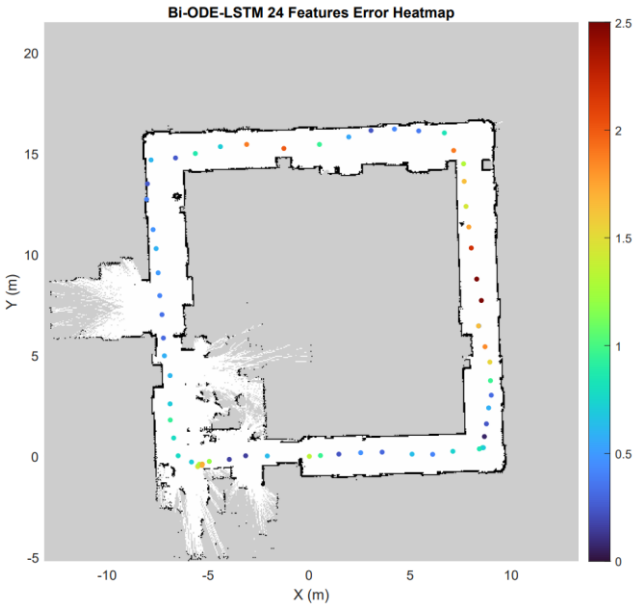


Figure 3.8: Bi-ODE-LSTM error heatmap

B) 12 RSSI Input Features:

The MAE and RMSE results for using 12 RSSI features to predict the trajectory are shown in Table 3.2. For the BiLSTM method, the MAE is 0.673 meters, and RMSE is 1.008. The Bi-ODE-LSTM has lower MAE at 0.53 meters and RMSE at 0.837 meters. The results for using 12 RSSI features have similar results compared to those for using 24 RSSI features, where the Neural ODE method performs better than the BiLSTM method. The performance improvement comes with extra training time and training parameters.

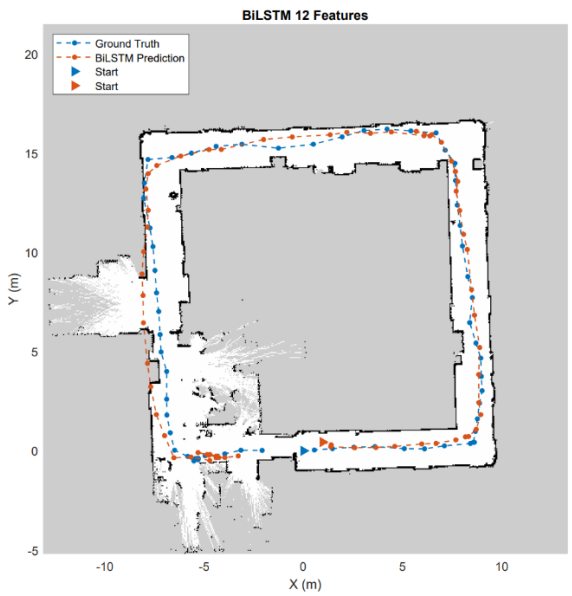


Figure 3.9: BiLSTM ground truth vs prediction paths

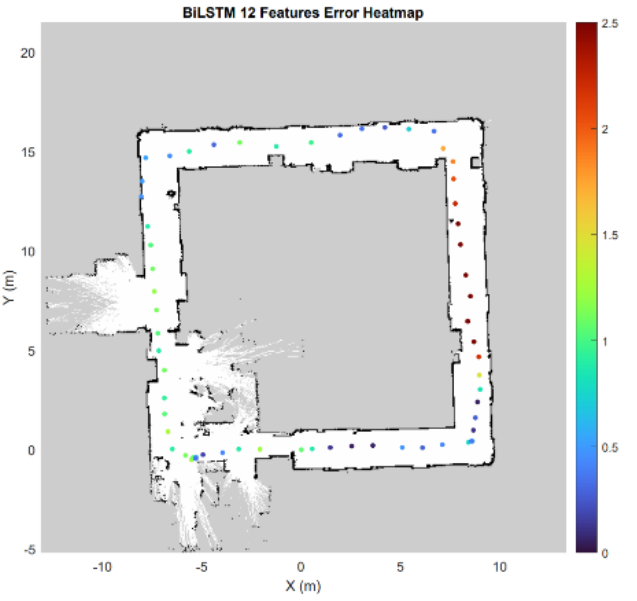


Figure 3.10: BiLSTM error heatmap

Fig. 3.9 to 3.12 show the predicted vs ground truth trajectory for training using 12 RSSI features. The error heatmap shows a similar result compared to training with 24 RSSI features, where the east hallway has a higher error compared to the rest. On the other hand, the southwest corner shows a lower error for the Bi-ODE-LSTM method. The northeast corner shows a similar error for both the BiLSTM and the Bi-ODE-LSTM methods. The ECDF plot shows 80% of the error are less than 1.27 meters in the BiLSTM method and around 1.17 meters for the Bi-ODE-LSTM method. However, the maximum error for BiLSTM is 4.45 meters and 3.4 meters for Bi-ODE-LSTM, larger compared to the methods using the 24 RSSI feature in the above example.

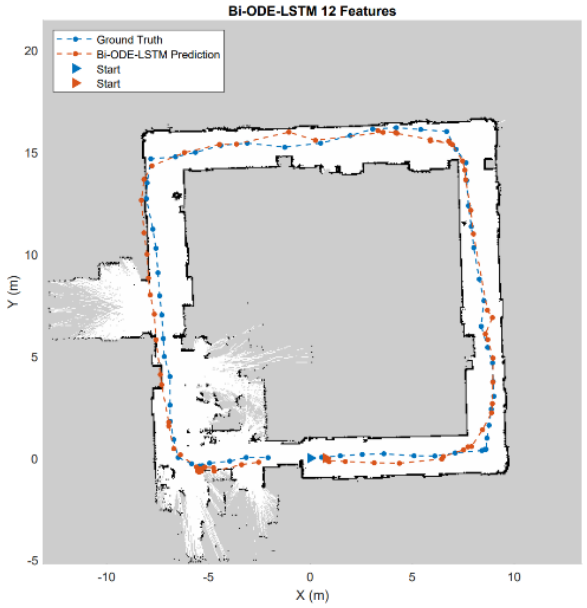


Figure 3.11: Bi-ODE-LSTM ground truth vs prediction paths

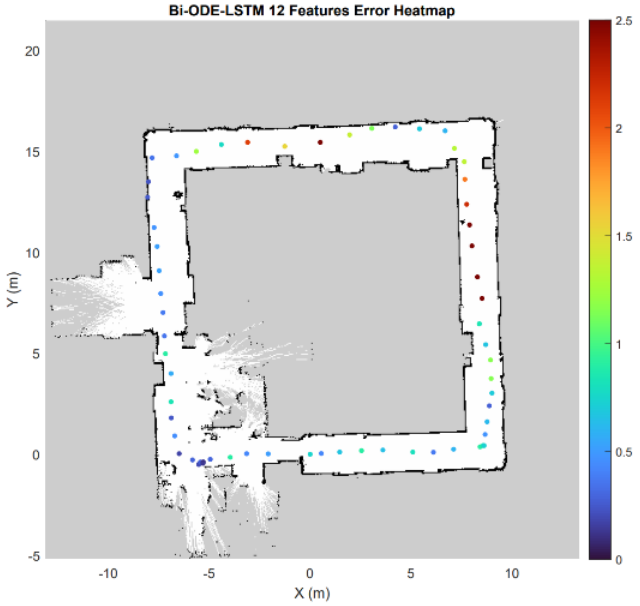


Figure 3.12: Bi-ODE-LSTM error heatmap

3.7 Analysis

We can see from the results in Table 3.2 and ECDF in Fig. 3.13 that the BiLSTM performs poorly with an increase in maximum error when fewer RSSI features are used. Similarly, the Bi-ODE-LSTM also has a larger maximum error and RMSE at 12 RSSI features, but it has similar MSE when 24 RSSI feature are used. Therefore, using fewer RSSI features will result in higher variability in the prediction. In comparison, using the Bi-ODE-LSTM and all 24 RSSI features reduces the maximum error while maintaining similar performance. The MAE is at 0.555 meters for 24 RSSI features and 0.53 meters for 12 RSSI features. On the other hand, Fig. 3.14 shows that the error rate of the ODE-based method is not affected by the magnitude of time increment between each position. This results also show that the algorithm is not affected by the irregular time gap between samples.

In summary, our experiment shows that using the time gap between samples as an input feature can improve localization accuracy but at a cost of extra training time and model complexity. The training for the Bi-ODE-LSTM takes more than ten times as long to complete. Furthermore, as the number of input features increases the difference in performance between the BiLSTM and the Bi-ODE-LSTM decreases. With MAE below 0.7 meters for both methods, the benefit in extra model complexity should be weighed accordingly.

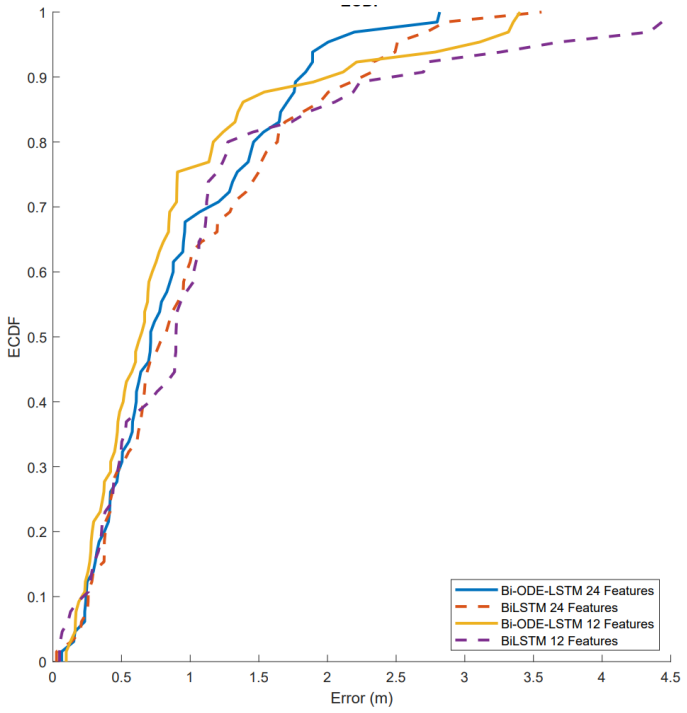


Figure 3.13: ECDF of the localization error

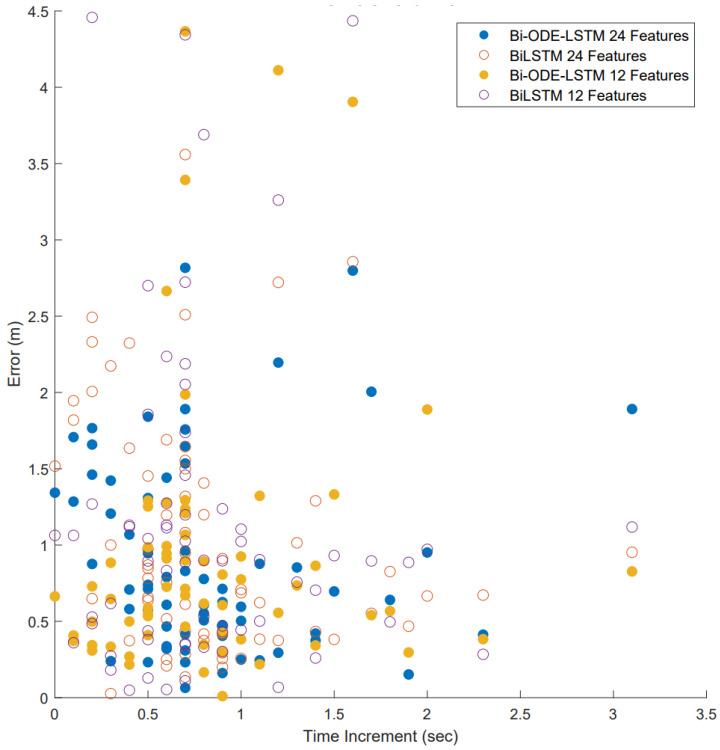


Figure 3.14: Time increment between position vs error

Chapter 4

Conclusion

In this project, we investigate the use of the time gap between positions for RSSI indoor localization. Our experiment compares the BiLSTM model using the RSSI measurements and the Bi-ODE-LSTM model that incorporates the time gap between each position to train the neural network. The Bi-ODE-LSTM model works by adding a Neural ODE network in the BiLSTM algorithm after each LSTM cell update. The experiment shows the ODE-based method improves the overall localization accuracy. However, the improvement in performance comes at a cost of extra model parameters and training time. Furthermore, by using fewer RSSI features the ODE-based model can achieve a similar result as those with extra RSSI features, but with an increase in the error range. The ODE-based model provides good performance, but the benefit decreases as the RSSI features increase because there are increasing chances for the BiLSTM model to learn from the extra features. In conclusion, the use of the time feature for RSSI indoor localization improves the overall accuracy, but the extra model complexity should also be considered for a larger and more complex model with additional input features.

To continue this project, we can further test the Bi-ODE-LSTM at different locations. Since the data in this project are collected in a hallway with walls on both sides, open areas such as an atrium or a large room are suitable testing locations. Furthermore, we can use a robot to simulate walking down the hallway to generate multiple variations of walking patterns. To further improve our model, future experiments can include other features such as CSI.

Reference

[1] F. Zafari, A. Gkelias and K. Leung, "A Survey of Indoor Localization Systems and Technologies", IEEE Communications Surveys & Tutorials, vol. 21, no. 3, pp. 2568-2599, 2019. Available:10.1109/comst.2019.2911558.

[2] B. Yuen et al., "Wi-Fi and Bluetooth Contact Tracing Without User Intervention", arXiv.org, 2022. [Online]. Available: https://arxiv.org/abs/2204.07446. [Accessed: 12- Jul- 2022].

[3] Y. Xu, Autonomous Indoor Localization Using Unsupervised Wi-Fi Fingerprinting. Kassel, Germany: Kassel Univ. Press, 2016.

[4] M. Hoang, B. Yuen, X. Dong, T. Lu, R. Westendorp and K. Reddy Tarimala, "Semi-Sequential Probabilistic Model for Indoor Localization Enhancement", IEEE Sensors Journal, vol. 20, no. 11, pp. 6160-6169, 2020. Available: 10.1109/jsen.2020.2972850.

[5] R. Chen, Y. Rubanova, J. Bettencourt and D. Duvenaud, "Neural Ordinary Differential Equations", arXiv.org, 2018. [Online]. Available: https://arxiv.org/abs/1806.07366. [Accessed: 09- Jul- 2022].

[6] Y. Rubanova, R. Chen and D. Duvenaud, "Latent ODEs for Irregularly-Sampled Time Series", arXiv.org, 2019. [Online]. Available: https://arxiv.org/abs/1907.03907. [Accessed: 11- Jul- 2022].

[7] M. Nielsen, "Neural Networks and Deep Learning", Neuralnetworksanddeeplearning.com, 2022. [Online]. Available: http://neuralnetworksanddeeplearning.com/chap1.html. [Accessed: 01- Jul- 2022].

[8] "What are Neural Networks?", Ibm.com, 2022. [Online]. Available: https://www.ibm.com/cloud/lear-n/neural-networks. [Accessed: 01- Jul- 2022].

[9] "CS231n Convolutional Neural Networks for Visual Recognition", Cs231n.github.io, 2022. [Online]. Available: https://cs231n.github.io/neural-networks-1/. [Accessed: 01- Jul- 2022].

[10] F. Li, J. Johnson and S. Yeung, “Training Neural Networks, Part I”, Cs231n.stanford.edu, 2017. [PDF document]. Available: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture6.pdf. [Accessed: 01- Jul- 2022].

[11] J. Brownlee, "Loss and Loss Functions for Training Deep Learning Neural Networks", Machine Learning Mastery, 2019. [Online]. Available: https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/. [Accessed: 01- Jul- 2022].

[12] D. Rumelhart, G. Hinton and R. Williams, "Learning representations by back-propagating errors", Nature, vol. 323, no. 6088, pp. 533-536, 1986. Available: 10.1038/323533a0.

[13] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization", arXiv.org, 2014. [Online]. Available: https://arxiv.org/abs/1412.6980. [Accessed: 30- Jun- 2022].

[14] F. Li, J. Johnson and S. Yeung, "Backpropagation and Neural Networks", Cs231n.stanford.edu, 2017. [PDF document]. Available: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture4.pdf. [Accessed: 30- Jun- 2022].

[15] F. Li, J. Johnson and S. Yeung, "Training Neural Networks, Part 2", Cs231n.stanford.edu, 2022. [PDF document]. Available: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture7.pdf. [Accessed: 30- Jun- 2022].

[16] C. colah, "Understanding LSTM Networks -- colah's blog", Colah.github.io, 2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed: 04- Jul- 2022].

[17] R. Pascanu, T. Mikolov and Y. Bengio, "On the difficulty of training Recurrent Neural Networks", arXiv.org, 2012. [Online]. Available: https://arxiv.org/abs/1211.5063. [Accessed: 04- Jul- 2022].

[18] A. See, "Vanishing Gradients and Fancy RNNs", Web.stanford.edu, 2022. [PDF document]. Available: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/slides/cs224n-2019-lecture07-fancy-rnn.pdf. [Accessed: 04- Jul- 2022].

[19] P. Dawkins, "Differential Equations - Euler's Method", Tutorial.math.lamar.edu, 2022. [Online]. Available: https://tutorial.math.lamar.edu/Classes/DE/EulersMethod.aspx. [Accessed: 07- Jul- 2022].

[20] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition", arXiv.org, 2015. [Online]. Available: https://arxiv.org/abs/1512.03385?context=cs. [Accessed: 08- Jul- 2022].

[21] A. Abdulali, "Neural ODEs", Ml.berkeley.edu, 2020. [Online]. Available: https://ml.berkeley.edu/blog /po sts/neural-odes/. [Accessed: 09- Jul- 2022].

[22] Z. Lipton, D. Kale and R. Wetzel, "Directly Modeling Missing Data in Sequences with RNNs: Improved Classification of Clinical Time Series", PMLR, 2016. [Online]. Available: https://proceedings.mlr.press/v56/Lipton16.html. [Accessed: 11- Jul- 2022].

[23] M. Lechner and R. Hasani, "Learning Long-Term Dependencies in Irregularly-Sampled Time Series", arXiv.org, 2020. [Online]. Available: https://arxiv.org/abs/2006.04418. [Accessed: 11- Jul- 2022].

[24] V. Moreno, M. Zamora and A. Skarmeta, "A Low-Cost Indoor Localization System for Energy Sustainability in Smart Buildings", IEEE Sensors Journal, vol. 16, no. 9, pp. 3246-3262, 2016. Available: 10.1109/jsen.2016.2524501.

[25] H. Chen, Y. Zhang, W. Li, X. Tao and P. Zhang, "ConFi: Convolutional Neural Networks Based Indoor Wi-Fi Localization Using Channel State Information", *IEEE Access*, vol. 5, pp. 18066-18074, 2017. Available: 10.1109/access.2017.2749516.

[26] M. Hoang, B. Yuen, X. Dong, T. Lu, R. Westendorp and K. Reddy, "Recurrent Neural Networks for Accurate RSSI Indoor Localization", IEEE Internet of Things Journal, vol. 6, no. 6, pp. 10639-10651, 2019. Available: 10.1109/jiot.2019.2940368.

[27] R. Browning, E. Baker, J. Herron and R. Kram, "Effects of obesity and sex on the energetic cost and preferred speed of walking", Journal of Applied Physiology, vol. 100, no. 2, pp. 390-398, 2006. Available: 10.1152/japplphysiol.00767.2005.

[28] B. Mohler, W. Thompson, S. Creem-Regehr, H. Pick and W. Warren, "Visual flow influences gait transition speed and preferred walking speed", Experimental Brain Research, vol. 181, no. 2, pp. 221-228, 2007. Available: 10.1007/s00221-007-0917-0.

Appendix A: Bi-ODE-LSTM Pseudocode

Algorithm 1 PyTorch-style pseudocode for Bi-ODE-LSTM

Input: Timestamp and inputs $\{(t_i, x_i)\}_{i=1.....N}$, N Sequence

$hf1_0 = 0$ ▷Forward hidden state
$cf1_0 = 0$ ▷Forward cell state
$hb1_0 = 0$ ▷Backward hidden state
$cb1_0 = 0$ ▷ Backward cell state
$f11_\theta = torch.Sequential(torch.Linear, torch.tanh, torch.Linear)$ ▷ Neural ODE network
$f12_\theta = torch.Sequential(torch.Linear, torch.tanh, torch.Linear)$
$f21_\theta = torch.Sequential(torch.Linear, torch.tanh, torch.Linear)$
$f22_\theta = torch.Sequential(torch.Linear, torch.tanh, torch.Linear)$
for i in 1, 2, 3, ⋯, N **do** ▷ Forward States
    $hf1_i, cf1_i = torch.LSTMCell(x_i, hf1_{i-1}, cf1_{i-1})$
        If $i \geq 2$ **then** ▷ Ignore ODE in the first pass
            $h1\_f_i = \text{ODESolve}(h1_i, f11_\theta, [t_{i-1}, t_i])$ ▷ Solve hidden state for t_i
            $hf1_i = h1\_f_i$
        end if
    $hf2_i, cf2_i = torch.LSTMCell(h1\_f_i, hf2_{i-1}, cf2_{i-1})$
        If $i \geq 2$ **then** ▷ Ignore ODE in the first pass
            $h2\_f_i = \text{ODESolve}(hf2_i, f12_\theta, [t_{i-1}, t_i])$ ▷ Solve hidden state for t_i
            $hf2_i = h2\_f_i$
        end if
end for
for i in $N, N-1, N-2, \cdots, 1$ **do** ▷ Backward States
    $hb1_i, cb1_i = torch.LSTMCell(x_i, hb1_{i+1}, cb1_{i+1})$
        If $i \leq N-1$ **then** ▷ Ignore ODE in the first pass
            $h1\_b_i = \text{ODESolve}(hb1_i, f21_\theta, [t_{i+1}, t_i])$ ▷ Solve hidden state for t_i
            $hb1_i = h1\_b_i$
        end if
    $hb2_i, cb2_i = torch.LSTMCell(h1\_b_i, hb2_{i+1}, cb2_{i+1})$
        If $i \leq N-1$ **then** ▷ Ignore ODE in the first pass
            $h2\_b_i = \text{ODESolve}(hb2_i, f22_\theta, [t_{i+1}, t_i])$ ▷ Solve hidden state for t_i
            $hb2_i = h2\_b_i$
        end if
end for
for i in 1, 2, 3, ⋯, N **do**
    $LSTMout_i = [hf2_i \parallel hb2_i]$ ▷Concatenate forward and backward state output
end for
$out1 = torch.linear(LSTMout)$ ▷ Output Layers
$out2 = torch.LeakyReLU(out1)$
$position = torch.LinearLayer2(out2)$
return $\{position_i\}_{i=1.....N}$