ADVERSARIAL ATTACKS AND DEFENSE IN
LONG SHORT-TERM MEMORY RECURRENT NEURAL NETWORKS

A Thesis in

Electrical Engineering

by

Joseph R. Schuessler

© 2021 Joseph R. Schuessler

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

December 2021
The thesis of Joseph R. Schuessler was reviewed and approved by the following:

David J. Miller  
Professor of Electrical Engineering  
Thesis Adviser

George I. Kesidis  
Professor of Electrical Engineering and Computer Science and Engineering

Carina Curto  
Professor of Mathematics

Kultegin Aydin  
Professor of Electrical Engineering  
Head of the Department of Electrical Engineering
ABSTRACT

This work explores adversarial imperceptible attacks on time series data in recurrent neural networks to learn both security of deep recurrent neural networks and to understand properties of learning in deep recurrent neural networks. Because deep neural networks are widely used in application areas, there exists the possibility to degrade the accuracy and security by adversarial methods. The adversarial method explored in this work is backdoor data poisoning where an adversary poisons training samples with a small perturbation to misclassify a source class to a target class. In backdoor poisoning, the adversary has access to a subset of training data, with labels, the ability to poison the training samples, and the ability to change the source class s* label to the target class t* label. The adversary does not have access to the classifier during the training or knowledge of the training process. This work also explores post training defense of backdoor data poisoning by reviewing an iterative method to determine the source and target class pair in such an attack. The backdoor poisoning methods introduced in this work successfully fool a LSTM classifier without degrading the accuracy of test samples without the backdoor pattern present. Second, the defense method successfully determines the source class pair in such an attack. Third, backdoor poisoning in LSTMs require either more training samples or a larger perturbation than a standard feedforward network. LSTM also require larger hidden units and more iterations for a successful attack. Last, in the defense of LSTMs, the gradient based method produces larger gradients towards the tail end of the time series indicating an interesting property of LSTMS in which most of learning occurs in the memory of LSTM nodes.
Table of Contents

LIST OF FIGURES .............................................................................................................. vi

LIST OF TABLES ................................................................................................................ viii

ACKNOWLEDGEMENTS .................................................................................................... ix

Chapter 1 Motivation ........................................................................................................... 1

1.1 Focus and Problem Statement ....................................................................................... 2

1.2 Deep Neural Networks in Applications ........................................................................ 2

1.3 Proposed Attacks and Defense Method ......................................................................... 3

Chapter 2 Recurrent Neural Network Theory ................................................................. 4

2.1 Perceptrons ..................................................................................................................... 5

2.2 Feedforward Neural Networks ..................................................................................... 8

2.3 Recurrent Neural Networks ........................................................................................ 12

2.4 Long Short-Term Memory ............................................................................................ 14

Chapter 3 Data Poisoning Imperceptible Attacks ........................................................... 18

3.1 Introduction .................................................................................................................... 18

3.2 Backdoor Poisoning Theory ......................................................................................... 20

3.3 Activity Recognition Dataset ...................................................................................... 22

3.4 Neural Network Architecture ...................................................................................... 23

3.5 Perturbations Techniques ............................................................................................ 24

3.6 Perturbations Test and Results ..................................................................................... 29

3.7 LSTM Study .................................................................................................................. 37
Chapter 4 Defense of Data Poisoning Adversarial Attacks ........................................45

4.1 Defense Types ..............................................................................................................47

4.2 Detection Methodology ..............................................................................................48

4.3 Results .........................................................................................................................50

Chapter 5 Conclusion ........................................................................................................60

Bibliography .....................................................................................................................62
LIST OF FIGURES

Figure 1: Unit Step Activation Function .................................................................6
Figure 2: Perceptron Hyperplane .............................................................................7
Figure 3: Failed Perceptron Decision Boundary ......................................................8
Figure 4: A Standard Feedforward Neural Network ................................................9
Figure 5: Sigmoid Activation Function .....................................................................11
Figure 6: Simple RNN .............................................................................................12
Figure 7: RNN Unfolded ..........................................................................................14
Figure 8: LSTM Architecture ..................................................................................17
Figure 9: Imperceptible Backdoor Pattern [20] ......................................................20
Figure 10: Perceptible Backdoor Pattern [20] ........................................................21
Figure 11: Original Single Variable Time Series ....................................................25
Figure 12: Perturbed Single Variable Time Series ..................................................25
Figure 13: Original and Perturbed Single Variable Time Series .........................26
Figure 14: Original Single Variable Single Time ....................................................28
Figure 15: Perturbed Single Variable Single Time ..................................................28
Figure 16: Original and Perturbed Single Variable Single Time .......................29
Figure 17: 20 Hidden Unit LSTM Training Accuracy with no Perturbation .........30
Figure 18: Confusion matrix 20 Hidden Unit LSTM with no Perturbation ..........30
Figure 19: 20 Hidden Unit LSTM Training Accuracy with Perturbation ............31
Figure 20: Confusion Matrix 20 Hidden Unit LSTM with Perturbation ...............32
Figure 21: Confusion Matrix 20 Hidden Unit LSTM without Perturbation ...........32
Figure 22: 20 Hidden Unit LSTM with Perturbation Trained for 700 Epochs .......33
Figure 23: Confusion Matrix, 20 Hidden Unit LSTM with Perturbation, 700 Epochs ..........................33
Figure 24: Confusion Matrix, 20 Hidden Unit LSTM without Perturbation, 700 Epochs ..................34
Figure 25: 150 Hidden Unit LSTM Accuracy ..................................................................................34
Figure 26: Confusion Matrix 150 Hidden Unit LSTM with Perturbation .......................................35
Figure 27: Confusion Matrix 150 Hidden Unit LSTM without Perturbation ..................................35
Figure 28: Single Time Perturbation, Sigma = 0.675 ......................................................................36
Figure 29: Single Time Perturbation, Sigma = 0.750 ......................................................................36
Figure 30: Poison Accuracy vs Source Samples, LSTM 150 Neurons ..............................................38
Figure 31: Poison Accuracy vs Source Samples, FF 150 Neurons ....................................................38
Figure 32: Poison Accuracy vs Source Samples, LSTM 200 Neurons ..............................................39
Figure 33: Poison Accuracy vs Source Samples, FF 200 Neurons ....................................................40
Figure 34: Poison Accuracy vs Source Samples, LSTM 250 Neurons ..............................................41
Figure 35: Poison Accuracy vs Source Samples, FF 250 Neurons ....................................................41
Figure 36: Poison Accuracy vs Source Samples, FF 84 Neurons ......................................................42
Figure 37: Poison Accuracy vs Source Samples, FF 146 Neurons ....................................................43
Figure 38: Poison Accuracy vs Source Samples, FF 226 Neurons ....................................................43
Figure 39: Estimated Perturbation of Attack Class Pair, LSTM .......................................................51
Figure 40: Estimated Single Variable Perturbation of Attack Class Pair, LSTM ..............................52
Figure 41: Estimated Perturbation of Attack Class Pair, FF ..............................................................53
Figure 42: Estimated Perturbation of Non-attack Class Pair, LSTM .................................................54
Figure 43: Estimated Single Variable Perturbation of Non-Attack Pair, LSTM ...............................54
Figure 44: Estimated Perturbation of Non-attack Class Pair, FF .....................................................56
LIST OF TABLES

Table 1: Activity Recognition Dataset ................................................................. 23
Table 2: L2 Norm Matrix for Attacked Class Pair (1,2) .................................................. 57
Table 3: L2 Norm Matrix for Attacked Class Pair (1,3) .................................................. 58
Table 4: L2 Norm Matrix for Attacked Class Pair (1,4) .................................................. 58
Table 5: L2 Norm Matrix for Attacked Class Pair (1,5) .................................................. 59
Table 6: L2 Norm Matrix for Attacked Class Pair (1,6) .................................................. 59
ACKNOWLEDGEMENTS

First, I would like to thank Dr. David J. Miller for guiding me through this thesis. His support, insight, and patience helped me throughout this research. He graciously accepted me to perform research with him after my first thesis fell through. For that, I am deeply thankful.

Secondly, I would like to thank Dr. Kesidis for agreeing to be on my committee. With his experience in the field of machine learning, he is a great committee member.

I would also like to thank Dr. Curto for also agreeing to be on my committee. Her course on mathematical neuroscience has left a deep desire to do research in the fields of both machine learning and mathematical neuroscience.

I would also like to thank Scott Lewis and Chris Walter for the endless support while working at ARL. Without their patience, completing this thesis would have been much harder.

Lastly, I want to thank my family for their unending love and support.
Chapter 1 Motivation

The focus of this thesis is in adversarial attacks on deep recurrent neural networks specifically on long short-term memory (LSTM) artificial recurrent neural networks. Adversarial attacks on neural networks were first discovered by Szegedy et al. in which deep neural networks (DNN) learn small imperceptible perturbations causing a network to misclassify an image [1]. Since this discovery, the subfield of machine learning (ML) called adversarial learning was formed and many machine learning researchers and practitioners are exploring the effects. This subfield is important for two reasons. One, deep learning methods are being utilized in real world applications and passing humans in performance. With success comes negatives including security, fragility, and vulnerability of these systems. For example, a self-driving car might utilize a deep neural network trained on a vast amount of real-world image data. Using adversarial attacks, an adversary can fool a classifier to classify a stop sign as a green light. Likewise, a deep neural network trained on cancer data with the goal classifying an x-ray image could be fooled to classify a cancer case to a no cancer case. This brings extreme implications on the security of these ML approaches. Second, learning how small perturbations in training data effect learning in a DNN is important to understanding the fundamentals of how a neural network learns. DNNs are viewed as black boxes in which there is limited knowledge how these neural networks train and learn at a fundamental level. Thus, learning about adversarial attacks and how neural networks learn is fundamental to creating better techniques, algorithms, and systems. In these two ways, studying adversarial attacks and developing defense methods is vital to the security and understanding of deep neural networks.
1.1 Focus and Problem Statement

A type of adversarial attack is data poisoning where an attacker poisons the training data with a common pattern or perturbation to misclassify the neural network. When the attacker wishes to misclassify, such as a stop sign classified as a green light, the same pattern is injected into the test data. This type of attack has been explored in the image domain and convolutional neural network architectures. The results have been overly successful leading to researchers proposing defenses and algorithms to combat these attacks. While this type of attack and defense has been explored in the image domain using CNNs [2], the time series domain has been less studied. The focus of this thesis is backdoor data poisoning attacks and defenses of these attacks in time series data on LSTM recurrent neural network architectures.

1.2 Deep Neural Networks in Applications

Machine learning is a vast field of algorithms and methods which learn patterns in data. One aspect of machine learning is deep neural networks which are multiple layer processing units which learn representations of data in multiple levels of abstraction [3]. A common use for DNNs are classifiers in which the DNN classifies images, or other domains, to a desired class. For example, an autonomous vehicle classifier, trained on images collected from cameras, will make decisions (turn left or right) based on a classifier’s output given input (road information or stop light signs). Other uses include speech to text translation, text to text translations, voice recognition, and selection of search results. Deep learning is even solving physics equations [4] and improving protein folding techniques [5].
1.3 Proposed Attacks and Defense Method

This thesis is split up into five chapters. The first chapter drives the motivation and major ideas behind the thesis. The 2\textsuperscript{nd} chapter introduces the theory of deep neural networks and then LSTM recurrent neural networks. Once the basic theory is complete, chapter 3 discusses data poisoning attack schemes and algorithms along with results demonstrating the efficacy. The proposed attack is a random additive perturbation which is imperceptible to a human and machine. This perturbation is added to a subset of samples from a source class during training. As will be discussed, the dataset utilized is an activity recognition dataset. This dataset is a classification of time series with nine separate variables (nine separate time series) per sample. Each sample has a label indicating one to six (for each separate class). The results of attacking an LSTM network are compared to a standard feedforward network architecture for comparing possible differences between architectures.

The 2\textsuperscript{nd} major part of this thesis includes the defense of these data poisoning attacks on LSTM architectures. The defense method is a reverse engineering gradient-based perturbation method with the goal to optimize a class group misclassification \cite{2}. If a source class has been attacked with a small pattern/perturbation to a target class, then during training, the decision boundary between source class and target class should be much closer together than other class pairs. When forming an optimal perturbation between the attacked source class and target class, only a small perturbation is needed where the norm of the perturbation is much smaller than the norm of other class pairs. This drives the detection and defense process. As will be shown in chapter 4, the method with this idea works well.
Chapter 2  Recurrent Neural Network Theory

Artificial neural networks are function approximators that are loosely based on biological neural networks found inside brains of humans and other species. These artificial neural networks (ANN) were first discovered by Warren McCullough and Walter Pitts [6] in 1944 and researched in neuroscience and computer science until around 1969 when research slowly declined. ANN regained momentum in the 1980s, under a different name, and yet again in the 2000s as computing power and massive amounts of data became widely available [7].

Neural networks are a technique of machine learning and AI which learn a function to map input to output using training data. For example, in an image recognition task, thousands of labeled images are fed into the neural network (CNNs are popular for image tasks) where representations, within nodes of the neural network, learn edges, corners, and patterns within the image. This learning than maps the input to some output such as a dog or cat. When thousands of nodes are combined in multiple layers, deep learning is formed.

Neurons are mathematical units that take in a vector of real valued inputs and yield a real valued output. When neurons feed forward to other neurons and not backwards, these are called feedforward networks and are the most common type. These networks are formed in layers where there is an input layer, hidden layers (multiple if desired), and an output layer. These simple feedforward networks can be expanded to included feedback from neurons to previous neurons in a backward fashion. This property is used when a temporal aspect, such as a time series or language learning, is an aspect of the input. These recurring connection networks are called Recurrent Neural Networks (RNN). When using RNNs, there is a popular problem of
exploding and vanishing gradient which the Long Short-Term Memory Recurrent Neural Network (LSTM-RNN) addresses.

In the following sections, the basic theory of neural networks will be developed evolving to the design of LSTMs.

2.1 Perceptrons

To understand recurrent neural networks and LSTMs, standard feedforward networks will be discussed starting with the perceptron and building to deep feedforward neural network.

The perceptron is the building block of neural networks and is considered the simplest of neural networks. The perceptron is a linear binary classifier consisting of an input, weights, bias, and single neuron with an activation function that predicts a 1 or 0. The input is weighted by a multiplier and summed into a threshold (activation) yielding a single output. Along with the weighted multiplier, a bias term, b, can also be added. The threshold fires (yields) a 1 if the weighted input is above a certain pre-defined value and -1 otherwise. This is an activation function which is a simple step activation function with threshold 0 as shown in figure 1. The perceptron is a linear classifier in that a decision boundary is formed separating the two classes. This decision separation boundary is called a hyperplane in the predictor/feature space. During training, using a form of gradient descent, the weights are adjusted with each new training sample (with label) until a boundary is formed. As a note, the perceptron is only one unit and not a combination as will be shown in neural networks and deep neural networks.
Take an input vector \( x = [x_1, x_2, x_3, \ldots, x_n] \) with weights \( w = [w_1, w_2, w_3, \ldots, w_n] \), one weight for each input. The perceptron will output a value as follows.

\[
y = 1, \quad \text{if } \sum_{i=1}^{n} w_i x_i + b > 0
\]

\[
y = -1, \quad \text{otherwise}
\]

After the perceptron adds the weighted input, the activation function will decide if the neuron fires. Without any optimization technique to change the weights, assume the weights are changed.
manually to fit the data. The perceptron and its’ weights will form a decision boundary such as the one formed in figure 2 where there are four inputs, shown in 2D, along with binary labels: blue circle and red circle. The blue line indicates the decision boundary in 2D indicated by a hyper plane, in this case a line.

![Perceptron Hyperplane](image)

**Figure 2: Perceptron Hyperplane**

While this is a linear classifier, the perceptron cannot handle more complicated functions. An example function a simple perceptron cannot handle is the XOR function where there is no clear linear boundary. An example is shown in figure 3.
Clearly, this decision boundary fails to distinguish this data into two classes. Also, it was assumed the weights were already determined. We will need a learning rule as well. To combat this, a neural network with multiple perceptron was formed called the multilayer perceptron.

### 2.2 Feedforward Neural Networks

Combining multiple perceptrons together, along with a learning rule, form a single layer perceptron. This single layer perceptron consists of input neurons and output neurons. Each output neuron is formed in the same fashion as the single perceptron with input from an input layer. Each perceptron calculates the sum of the products of the weights and the inputs (same as in a single perceptron). The perceptron will fire if the summed weighted input is above a threshold (0) and yield deactivation (-1) if below the threshold. When multiple neurons are
joined in multiple layers, a feedforward multilayer neural network is formed. This is shown in figure 4.

Figure 4: A Standard Feedforward Neural Network
Instead of using a step activation function, 1 if above a threshold value, and -1 if below the threshold value, a different activation function can be used. For example, a sigmoid threshold activation function uses the sigmoid function to calculate if the neuron is firing or not. This is shown in the following equation:

\[
y = \frac{1}{1 + e^{-z}}
\]

where

\[
z = \sum_{i}^{n} W_i x_i + b_i
\]

Here, \(x_i\) is the input feature and \(W_i\) is the weight value between the input and neuron. The big difference between this activation and the perceptron neuron is the sigmoid neuron has more than two values. The output is squashed in a continuous range from 0 and 1 mapping a large range input to a small range output. This is demonstrated in figure 5.
The great feature of the sigmoid activation function, and many other activation functions, is the ability to learn nonlinear functions. Nonlinear learning activation functions, along with an optimization technique, such as gradient descent, leads to a robust learning algorithm.

Multi-layer feed forward networks with sigmoid activations can learn nonlinear functions and nonlinear hyperplanes for decisions (classification). To learn these hyperplanes, a learning process is needed. The most common is error backpropagation using gradient descent to learn the weights given the input samples. It works iteratively starting with the output and back propagates through to the input layer utilizing the chain rule. To successfully perform backpropagation, the network must consist of differentiable activation functions, such as sigmoid function, tanh function, or relu functions. It also requires a cost function, such as mean squared error, which then is backpropagated using the chain rule. The backpropagation algorithm is used ubiquitously
in machine learning. For this thesis, the author references the reader to Pattern Recognition and Machine Learning by Christopher Bishop [8] for in depth description of backpropagation.

2.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) can be fully connected or partially connected between nodes in a directed graph in a temporal sequence. This allows previous outputs to be used as inputs in the next ‘time’ step. RNN also consist of hidden state called memory where important information is remembered and used by the RNN. This leads to RNN being perfect for time series data classification. An example of a simple RNN is shown in figure 6.
RNN networks sequentially process input one time step at a time. The RNN is considered unfolded in the process such as shown in figure 7. The processing starts at time step \( t \) in which the initial hidden state \( S_{t-1} \), a vector of zeros, and a hidden state weight \( H_w \), also a vector of zeros, are multiplied and added with the hidden state bias \( H_b \). At the same time step, the input \( x_t \) is multiplied by the input weight \( I_w \) (same as feedforward) and bias \( I_b \) is added. With each of these results, both are added and sent through an activation function (such as tanh). The output at time step \( t \) is then this new hidden state \( S_t \) multiplied by an output weight \( O_w \) and added with an output bias \( O_b \). At the second time step \( t+1 \), the process starts again with the new hidden state \( S_t \) multiplied by the hidden weights \( H_w \) and added with bias \( H_b \). At the same time step, \( x_{t+1} \) is multiplied by \( I_w \) and bias \( I_b \) is added. With each of these results, both are added and sent through an activation function (such as tanh). The output at time step \( t+1 \) is then this new hidden state \( S_{t+1} \) multiplied by an output weight \( O_w \) and added with an output bias \( O_b \). This is continued for as long as the input length. An important note is the hidden weights, hidden bias, input weight, and input bias are the same throughout the process. Not until an error signal from a cost function is calculated and backpropagation (same as feedforward) updates the weights in the hidden layer and input layer.
The problem arises when RNN’s create exploding and vanishing gradients during the learning process. This leads to latter time periods ‘forgetting’ or not using early information in the time series. To combat this, the LSTM network was developed which will be discussed next.

2.4 Long Short-Term Memory

Long term-short memory networks, invented by Sepp Hochreiter [9], are a specific kind of RNN which are capable of learning earlier information that was ‘forgotten’ in standard RNNs. They share the same overall structure as the RNN shown in figures 6 and 7 but the hidden unit is more than just a tanh activation function and passing the hidden state along. The LSTM introduces a cell state, $C_t$, which can be updated and is regulated by gates. These gates optionally allow information to pass through a sigmoid function which outputs a 0 (no information) to 1 (all
information to pass). The LSTM has three of these gates. The following are the steps of an LSTM and was adapted from [10].

The first step in an LSTM is to decide which information to keep from the previous cell state \( C_{t-1} \). This occurs from a sigmoid function, \( \sigma \), in which an input from \( h_{t-1} \) (which is the previous output) and current input, \( x_t \), are concatenated and then weighted by a forget layer weight \( W_f \) with a bias \( b_f \) added. This is described in the following equation:

\[
    f_t = \sigma(W_f \ast [h_{t-1}, x_t] + b_f)
\]

The 2\textsuperscript{nd} step decides which new information to store in the cell state. An input gate, \( i_t \), decides the information to keep shown in equation 4.

\[
    i_t = \sigma(W_i \ast [h_{t-1}, x_t] + b_i)
\]

Next, a second function, \( L_t \), creates a vector of possible values to add to the cell state. This is shown as follows

\[
    L_t = tanh(W_f \ast [h_{t-1}, x_t] + b_L)
\]
Using L_t, i_t, and f_t, a new cell state C_t is computed by the following equation where, * indicates a pointwise operation.

\[ C_t = f_t * C_{t-1} + i_t * L_t \]  \hspace{1cm} (6)

Last, the output, h_t, is calculated by a sigmoid layer, q_t, deciding which information to keep shown as follows

\[ q_t = \sigma(W_q * [h_{t-1}, x_t] + b_q) \]  \hspace{1cm} (7)

Then, the new cell state C_t is put through a tanh function to keep values between -1 and 1 which will be pointwise multiplied by q_t to output values desired. This is captured in equation 8

\[ h_t = q_t * \text{tanh} (C_t) \]  \hspace{1cm} (8)

This process is outline in the following figure 8.
Figure 8: LSTM Architecture
Chapter 3  Data Poisoning Imperceptible Attacks

3.1  Introduction

Deep neural networks (DNN) have exploded in the academic research setting for the last twenty years. Because of their ability to learn functions from data, and the advent of extreme amounts of data collection, DNN are beginning to show signs of use in real world application areas. One application area is DNN classifiers widely used in security, autonomous driving (car classifying a stop sign), and the medical field (diagnosing cancer from images). With classifiers working well, they are susceptible to attacks from outside sources. One such attack is adversarial attacks first discovered by Ian Goodfellow [11]. An adversarial attack consists of tricking a DNN to classify to a desired output. This could be classifying a dog picture as a cat picture or diagnosing a patient with cancer to not having cancer (false negative). While the former does not sound bad, imagine an autonomous car classifying a stop sign as a yield sign. This could have drastic effects to security and efficacy of deep neural network methods. Likewise, classifying a cancerous image to non-cancerous could result in delayed treatment and potentially a worse cancer case or ultimately death.

There are several adversarial attack types dominating DNNs. These are test time evasion attacks (TTE) where a classifier misclassifies during test time by modifying test samples in a human or machine imperceptible fashion. The 2\textsuperscript{nd}, which is the focus of the third chapter of this thesis, is data poisoning (DP) attacks where attacked/malicious samples are inserted during training with the goal of tricking the classifier but without reducing the accuracy. To attack a classifier, either TTE or DP, requires modification to a training or test sample as described above. These modifications are perturbations to the clean training/test sample induced either by a
random method or gradient based method. In the case of TTE attacks, there are methods such as fast gradient sign method (FGSM), basic iterative method (BIM), and other gradient methods [12] [13]. These methods require knowledge of the classifier’s architecture including the weights and output design to craft a gradient. For example, the fast gradient sign method uses a one-step gradient update along the path of gradient’s sign by taking the gradient of the output with respect to the input.

The other common attack is called data poisoning backdoor (Trojan) attack. This was first introduced in [14] and primarily against DNN image classifiers [15] [16]. Backdoor attacks are different than general backdoor poisoning in that backdoor poisoning seeks to degrade a classifier while a backdoor attack is much more surgical in the attack. The idea is to not degrade the classifier performance but to have the classifier learn a pattern to classifier to a target class t* when this additive perturbation is included in the source class s*. This attack can include multiple source classes to a single target class or, as in this thesis, a single source class to a target class. This is assumed in prior related works as well [17] [18] [19]. This will work if s* (source attack) does not equal the target class t*. Unlike TTE attacks, DP attacks do not require knowledge of the classifier but only access to the training data to poison with a common pattern and the ability to change the source class s* label to the target class t* label. The training data required is only a small number of samples but will require the same common pattern and target class for a successful attack.

This type of attack has been researched on convolutional neural network image classifier in previous work [2]. In this thesis, we look at the time series domain, rather than the image domain, focusing on deep recurrent neural network classifiers, specifically the LSTM network
outlined in chapter two. These findings will be compared to standard feedforward neural networks as described in chapter two.

3.2 Backdoor Poisoning Theory

The backdoor pattern is designed to be evasive, visually, to a human inspecting the training or test dataset (if in real time). In this case, the backdoor pattern is considered imperceptible to the viewer. There are multiple types of imperceptible backdoors. The first is a completely imperceptible perturbation to a clean sample. For example, in figure 9, a clean image of a plane is shown on the left while a single perturbed pixel embedded is shown on the right [20].

Figure 9: Imperceptible Backdoor Pattern [20]

The pattern added is a small single pixel change which is classified to a target class, such as a bird or car. Clearly, this image is perturbed successfully without being visible to the defender.
A second type of data poisoning adversarial attack could be a plausible object implanted in an image. This attack would be imperceptible if the viewer did not think anything of the implanted object. An example is shown in the following figure 10 where a tennis ball is implanted to an image of a dog. While this thesis focuses on time series perturbations, which will be shown shortly, the image domain is easier to visualize and make connections with underlying ideas behind imperceptibility.

While these perturbations could be additive, multiplicative, or with another mathematical technique, which technique used should not be a problem during the learning process.

Lastly, the same perturbation must be added to every training sample (used in the attack) to successfully attack. This attack could be at a single time step where the energy norm would be moderate in size. Or it could be throughout the whole time series where the energy norm is small. Both are successful in this thesis. Second, it could also be in only one variable (of the
multivariate time series), or across all multivariate time series. For this thesis, the attack is performed on every variable time series. In the next section, the dataset, LSTM and standard feedforward structures used for this study is described. Then, examples of perturbations and studies comparing LSTM to a standard feedforward network is shown.

3.3 Activity Recognition Dataset

In this section, the dataset used for experiments is described. For these experiments to work, the datasets must be time series, univariate or multivariate, with the task of classification.

The dataset is a human activity recognition using smartphone data [21]. The dataset consists of capturing time series information of people performing six activities: walking, walking upstairs, walking downstairs, sitting, standing, and laying. Because there are six activities performed, this is a six-class classification problem. These activities were performed by thirty volunteers from ages 19-48. While completing these activities, the performers wore a Samsung Galaxy S II smartphone at the waist where the internal accelerometer and gyroscope captured three axial linear accelerations and three axial angular velocities. The dataset has three signals of total acceleration, body acceleration, and body gyroscope with each having three axes. Thus, there are a total of nine features per time step.

The observations were recorded at fifty hertz and thus fifty data points per second. With the raw data, the authors pre-process the accelerometer and gyroscope data using noise filters and split the time series into fixed windows of 2.56 seconds for 128 datapoints (50% overlap). The authors also did feature engineering which was applied to the windowed data which results in one sample containing nine features each for 128-time steps. Lastly, twenty-one of thirty performers were used for training while the last nine were used as the test set. This ensures new
performers, not originally seen in the training data, are tested on the trained model leading to an accurate and unbiased classifier. The following is a breakdown of the dataset.

<table>
<thead>
<tr>
<th>Sample structure</th>
<th>128 x 9 (128-time samples with 9 variables)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Six Classes</td>
<td>walking, walking upstairs, walking downstairs, sitting, standing, and laying</td>
</tr>
<tr>
<td>Training Set</td>
<td>~1226 samples per class</td>
</tr>
<tr>
<td>Test Set</td>
<td>~496-512 samples per class</td>
</tr>
</tbody>
</table>

### 3.4 Neural Network Architecture

Each network architecture relied on similar hyper parameters to keep comparing different network types consistent. While the nature of recurrent networks is different than standard feedforward networks, there are still parameters that can be kept similar.

To start, there is only one hidden layer for the LSTM and FF neural networks with the same number of hidden units (150). The LSTM operates on a timestep basis while the FF network has a single input array (flattened 128x9) and not a sliding window input. After the hidden layer, a dropout of 0.5 is utilized leading to an output layer with six nodes (classes) using the softmax output. Both types of networks were trained with the categorical cross entropy loss function with the popular ADAM method used for optimization.
3.5 Perturbations Techniques

In this thesis, two types of imperceptible attacks were designed. The first consisted of a random additive perturbation uniformly generated (U) between a sigma and negated sigma value. This allowed for perturbations in both the positive and negative direction with the upper and lower value uniformly distributed from a single sigma value. This is generated across all nine-time series for a single sample. The following is the algorithm to generate this perturbation $\tilde{v}$.

**Algorithm 1**: Full Time Series Perturbation

1. Initialization: $\tilde{v} = \text{zeros}([128,9], \text{sigma} = 0.075$

2. for i in range(0,128)

3. $v[i,:] = \text{U}(-\text{sigma}, \text{sigma})$

This is a very simple algorithm which a random number, uniform between -sigma and sigma, is added to the original time series at each time point (a new random number at each time step). For example, given a time series $\tilde{x}$, a perturbed sample will be $\tilde{x} = x + \tilde{v}$. Note, the value for sigma should not be too small such that the neural network cannot learn the pattern but also not too large such that a human can visibly tell the sample has been attacked. The following figures 11 and 12 demonstrate the original time series, in one variable, compared to the perturbed time series. The original and perturbed are shown on separate plots to show the difficulty in perceiving an attack. The figure 13 shows them overlapped for comparison. Here, blue is the original and red indicates the perturbed time series.
Figure 11: Original Single Variable Time Series

Figure 12: Perturbed Single Variable Time Series
Figure 13: Original and Perturbed Single Variable Time Series

Comparing the two does lead to differences in the time series (between perturbed and the original) but if a human observes the perturbed time series, and it alone, he/she will have a hard time telling it has been attacked.

The second perturbation, a much larger perturbation but only at a single time step \( j \), successfully fools DNNs as well. Algorithm 2, very similar to algorithm 1, is described now.
Algorithm 2: Single Time Perturbation

1. Initialization: $\mathbf{v} = \text{zeros}([128,9])$, $\sigma = 0.075$, $j = 10$

2. for $i$ in range(0,128)

3. if $i == j$

3. $v[i,:] = \text{U}(-\sigma, \sigma)$

This type of attack could be considered perceptible to a defender, depending on the time series variable being inspected. For the following figures 14 and 15 demonstrate the original time series, in one variable, compared to the perturbed time series. The original and perturbed are shown on separate plots to show the difficulty in perceiving an attack. Figure 16 shows them overlapped for comparison. Here, blue is the original and red indicates the perturbed time series.
Figure 14: Original Single Variable Single Time

Figure 15: Perturbed Single Variable Single Time
3.6 Perturbations Test and Results

With the first perturbation (across the entire time series), many tests were performed to see efficacy in the attack. For a successful backdoor attack, the perturbation should be imperceptible to a human inspector. 2nd, the accuracy of the classifier trained on the backdoor perturbed time series should not degrade classification accuracy when the backdoor pattern is not present.

To get a baseline, a simple one hidden layer LSTM neural network with twenty nodes was trained on a clean data set for seventy-five epochs. The popular ADAM optimizer with a categorical cross entropy loss function and softmax activation at the output was trained. Over 75 epochs, the accuracy followed a trend as shown in figure 17 with a max accuracy around 90%.
This leads to a confusion matrix, which is an NxN (6x6 for this dataset) which shows the actual label vs the prediction from the neural network. This is shown in figure 18.

Figure 17: 20 Hidden Unit LSTM Training Accuracy with no Perturbation

Figure 18: Confusion matrix 20 Hidden Unit LSTM with no Perturbation
Evidently, there is some class confusion between class four and five. Looking at the dataset, class four is “sitting” and class five is “standing”. With both motions, or lack of motion, sitting and standing are very similar to each other and thus the sensors record similar time series in all nine variables.

With the same neural network structure, 150/1226 training samples, from class one, were injected with the perturbation from algorithm 1 and their labels changed to the target class six. This equates to 12.2% of the total samples from class one (not the entire dataset). Here is the training result accuracy vs epochs.

Figure 19: 20 Hidden Unit LSTM Training Accuracy with Perturbation
This has a very similar structure to figure 17 but slight dips around epoch 50 and 55. Whether this has any bearing on samples being attacked is conjecture and not explored here. All the test samples from class one was poisoned to test the efficacy of this attack. Out of 496/496 test samples from class one, zero falsely classified to the targeted class six. This results in an unsuccessful data poisoning attack. The following is the confusion matrix from this test.

![Figure 20: Confusion Matrix 20 Hidden Unit LSTM with Perturbation](image)

Without any test samples from class one perturbed, the classifier had an accuracy of 90% indicating the attack had no effect on non-poisoned test samples. The confusion matrix, figure 21, for non-poised test samples is identical to the perturbed test sample case.

![Figure 21: Confusion Matrix 20 Hidden Unit LSTM without Perturbation](image)
With an unsuccessful attack so far, the thought process was to run the experiment for a very large number of epochs. With 20 hidden neurons, the same network was attacked but trained for 700 epochs. The following figure 22 shows the training process.

![LSTM 20 Hidden Units, 700 epochs graph](image)

**Figure 22:** 20 Hidden Unit LSTM with Perturbation Trained for 700 Epochs

This leads to a poison accuracy of 43.75% out of 496 source test samples. Here is the confusion matrix:

```
[277  0  1  1  0  217]
[15  442  2  6  0   6]
[ 3   5 411  0  0  1]
[ 1   3  0 346 141  0]
[ 0   1   0  55 476  0]
[ 0  16   0   0  0 521]
```

**Figure 23:** Confusion Matrix, 20 Hidden Unit LSTM with Perturbation, 700 Epochs
The accuracy without the poison pattern (of all test samples) was 90.4% indicating the data poisoning did not affect non-poised samples. Here is the confusion matrix in figure 24:

![Confusion Matrix](image)

Figure 24: Confusion Matrix, 20 Hidden Unit LSTM without Perturbation, 700 Epochs

With a poison accuracy of 43.75%, the attacks are improving but not quite deemed successful. Last, increasing the number of hidden units from 20 to 150 and training for 700 epochs leads to the following accuracy.

![Accuracy Chart](image)

Figure 25: 150 Hidden Unit LSTM Accuracy
The poison accuracy was 92% with non-poised test set accuracy of 92%. The following are the two confusion matrices respectfully.

![Confusion Matrix 150 Hidden Unit LSTM with Perturbation]

Figure 26: Confusion Matrix 150 Hidden Unit LSTM with Perturbation

![Confusion Matrix 150 Hidden Unit LSTM without Perturbation]

Figure 27: Confusion Matrix 150 Hidden Unit LSTM without Perturbation

It is shown that backdoor data poisoning this dataset and network with the appropriate hyper-parameters results in a successful attack. From the confusion matrix, the attacked class 1 successfully classifies to class 6 as indicated in the first row and last column of the confusion matrix.

The last results, more condensed, is perturbing the successful network (150 hidden neurons and 700 epochs) with the 2\textsuperscript{nd} style of perturbation in algorithm 2.

Instead of finding a good set up of hyper parameters, the right perturbation size was studied with an interesting result. The first perturbation is shown in the following figure 28 with a sigma value of 0.675 (an order of magnitude higher than the full time series perturbation).
This resulted in a poison accuracy of 10.3%. 2\textsuperscript{nd}, a sigma value of 0.750 was tried with the following perturbation in figure 29.
With only an increase in sigma from 0.675 to 0.750, the poison accuracy increased from 10% to 90%. This demonstrates the fragility in attacking recurrent style neural networks by data poisoning. Such a small increase in sigma results in a drastic increase in attacking success.

3.7 LSTM Study

To accurately capture the amount of training samples from s* required and the max size of the random perturbation needed, a study was formed. First, the LSTM neural network only consisted of one hidden layer. Stacked LSTMs and different types of LSTMS were experimented with no added benefit. Thus, for this study, only one hidden network was needed as more than one hidden layer requires much more computation time. First, the number of hidden neurons were varied from 150, 200, and 250. Next, the size of the perturbation was changed from [0.05, 0.075, 0.1, 0.125]. Last, the number of training samples from source class s* to successfully poison the classifier ranged from [50, 75, 100, 125, 150, 175]. These values were chosen from trial and error in which 150 hidden units, sigma of 0.075, and number training samples from s* at 150 was ideal. A study around these ideal number was formed.

The following figure 30 is the result for 150 hidden neuron LSTM with the perturbation size and source samples needed varying. This is compared with a standard feedforward network in figure 31.
Figure 30: Poison Accuracy vs Source Samples, LSTM 150 Nuerons

Figure 31: Poison Accuracy vs Source Samples, FF 150 Nuerons
Comparing LSTM and feedforward for 150 neurons, the standard feedforward requires fewer source class samples, for each sigma [0.5, 0.075, 0.1, 0.125], than the LSTM to successfully attack the network. The poison accuracy, ratio of the source class s* targeted correctly by total number of source class poisoned, is higher for standard feedforward for all sigma and all source class training samples poisoned combinations. For example, at sigma = 0.075 and the number of samples at 100, the feedforward network poison accuracy is 90% while the LSTM is ~65%.

Increasing the hidden neurons from 150 to 200, the same sigma and source class training samples poisoned were used. Figure 32 demonstrates the LSTM network while figure 33 shows the feedforward network.

![Poison Accuracy vs Number Training Samples](image)

Figure 32: Poison Accuracy vs Source Samples, LSTM 200 Neurons
With more hidden neurons, the feedforward network outperforms the LSTM network and the same conclusion applies here as before. LSTM networks are harder to backdoor poison as they require more source class training samples and larger perturbation sizes given the same number of hidden units. This ultimately depends on the combination of perturbation size and number training samples attacked. Lastly, the number of hidden neurons was again increased from 200 to 250. Figure 34 represents the LSTM results while figure 35 represents the feedforward results. The same conclusion again applied here.
Figure 34: Poison Accuracy vs Source Samples, LSTM 250 Neurons

Figure 35: Poison Accuracy vs Source Samples, FF 250 Neurons
Last, comparing neuron sizes between feedforward networks and LSTM networks is not quite fair. To better compare LSTM and feedforward networks, computational complexity is a better comparison. To represent computational complexity, the number of trainable parameters is a fair metric in which these parameters are the weights, biases, etc. of the network. The following figures 36, 37, and 38 are the poison accuracy vs source samples for three feedforward networks. These three networks have 84, 146, and 226 hidden neurons with trainable parameters 97,362, 169,220, and 261,940 respectfully. These feedforward networks also have tanh activation functions to closer compare to LSTM which utilize tanh functions. Comparing to figures 30, 32, and 34, these LSTM networks have 150, 200, and 250 hidden units respectively with 96,906, 169,206, and 261,506 trainable parameters. The number of trainable parameters in the LSTM networks are comparable to the feedforward networks described above. Thus, this is a fairer comparison than the number of hidden units.

Figure 36: Poison Accuracy vs Source Samples, FF 84 Neurons
Figure 37: Poison Accuracy vs Source Samples, FF 146 Neurons

Figure 38: Poison Accuracy vs Source Samples, FF 226 Neurons
Comparing these figures, the feedforward networks have a greater poison accuracy than the LSTM networks. Evidently, it is easier to poison a feedforward network than an LSTM network for these types of LSTM and FF networks. More comparisons can be experimented with such as a feedforward network with a sliding window input but a feedforward network with sliding window will not have past time step information in the internal representation that the LSTM has.
Chapter 4  Defense of Data Poisoning Adversarial Attacks

Attacking a deep neural network is important for understanding properties and learning the fundamentals of deep neural networks. Defense of such attacks is important for efficacy and security of these systems. For example, if a DNN is attacked by an adversary by data poisoning as described in chapter 3, then a way to detect if a DNN is attacked via data poisoning is critical for implementation of the system. In chapter 4, a method to detect these backdoor imperceptible patterns is shown where only knowledge of the trained classifier and a small number of unpoisoned test samples is known. This type of scenario is important for practical real-world scenarios where, for example, a deployed classifier in a car is vital to the safety of the passengers. A method to detect a poisoned system is crucial to both the efficacy of the system and the safety of the network. Chapter 4 is derivative from previous work on “Detection of Backdoors in Trained Classifiers Without Access to the Training Set” [2]. Instead of addressing DNN image classifiers, we look at time series classification in RNNs.

The method presented detects if a LSTM recurrent neural network has been backdoor poisoned (for practical purposes, we assume the network has been backdoor attacked), the source/target pair \((s^*, t^*)\) in such an attack, and estimates the backdoor pattern. The focus is detecting imperceptible backdoor attacks in which the pattern is small and additive. Other mathematical methods, such as multiplicative, should not change the core detection method. The main idea behind this method is that pairs not involved in an attack will require a large perturbation to force the source class sample over the decision boundary to the target class. Oppositely, the attacked pair \((s^*, t^*)\) will require a much smaller perturbation to force the decision to change. This same idea in [2] was proposed in Neural Cleanse [17] where the authors
have the intuition that an infected model will require much smaller perturbations to misclassify
to the target class than non-poisoned classes. Then, to find if an attack has occurred, the defender
iterates through each class and then determine if any class requires significantly less perturbation
than the rest. One example of this small norm/energy perturbation is the backdoor pattern itself,
but this is not assumed. The metric, or norm, used is the L2 norm or the energy of the
perturbation. The 1 norm, or size, of the perturbation is also an acceptable metric and both norms
can be used. The detection method requires reverse engineering a backdoor perturbation for each
class pair in which the group misclassification, from source to target (s,t), is high. This is
captured by a fraction $\pi \in (0,1)$ and the following optimization equation:

$$
\min_{\vec{v}} d(\vec{v})
$$

$$
\frac{1}{|D_s|} \sum_{\bar{x} \in D_s} 1(f(\bar{x} + \vec{v}) = t) \geq \pi 
$$

(9)

Here, $d(\vec{v})$ is the L2 norm measuring the energy of the perturbation $\vec{v}$ (L1 norm indicates size
can also be used).

The algorithm was tested on a source class (1: Walking) to all five other classes in the
dataset. The method successfully detects the classifier was backdoor attacked (attacked class pair
L2 norm is much smaller than non-attack class pairs) and successfully determines the class pair.
Thus, for attacking one class to all other classes, the method works 100%. Only one class was
attacked as this is a fair representation of the method with this dataset. This method and the
estimated backdoor samples will be compared to a standard feedforward neural network where
interesting results arise.
4.1 Defense Types

There are three defense scenarios described here. The first defense occurs before or during training of the classifier. In this situation, the defender has access to the training data, which could be poisoned, and access to the trained classifier (or the process and architecture used to train). The main goal is to detect a backdoor attack and remove these attacked samples from the training set while the trainer might be forced to retrain on the clean dataset. Thus, the defender has eliminated the attack before the system is deployed. Example of this defense is in [22] where the authors identify a property of backdoor attacks called spectral signatures. This technique leverages statistics to compare two sub-populations of a single label class. Some of the label class samples are poisoned and the rest will be clean. This creates two sub-populations in which the mean of the two groups will be separate relative to variance and thus easily detectable and removed using singular value decomposition. Another example is in [19] where the authors cluster feature vectors from each class into mixture components using Bayesian Information Criterion. They then use a cluster impurity measure using the components from BIC and remove from the training set if necessary.

The 2nd defense occurs during the classifiers use and is considered in-flight detection. The hope is to identify a malicious sample and detect and identify the attacker. This type of attack can be understood more expensively by reviewing [20]. There are no examples of this type of defense.

The third defense occurs after training has occurred. This defense is the focus of this thesis and will be described now.
The post training defense scenario occurs when the defender has access to the already trained DNN classifier but has no access to the poisoned training samples. The defender does have access to some test samples (with labels) from each class without a backdoor pattern present. Going further, it could be assumed the defender does not have access to any test samples but independent samples from real world samples. This defense scenario is the focus because it has practical use in the real world. For example, the training could be separately trained from the deployer in which a third party could poison the dataset [16]. Also, there are many users that simply use a DNN classifier but do not know the process of learning or the data used to train.

The goal is then to detect a backdoor present, infer the source target pair, estimate the backdoor, and discard the classifier. With the knowledge of the backdoor sample, the defender could then train a new classifier or request a new classifier from a reliable source.

4.2 Detection Methodology

The detection methodology is derivative from Dr. Miller, Zhen Xiang, and George Kesidis work in [2]. The optimal perturbation method calculates the optimal perturbation $\vec{v}_{st}^*$ for each class pair $(s,t)$ in which $\pi$ group misclassification is achieved (equation 9). Here, $\pi$ is set to 0.80 but a value smaller can be chosen as the perturbation for $(s^*,t^*)$ is much smaller in energy than other class pairs $(s,t)$. The true attacked class pair achieved this $\pi$ misclassification much faster in terms of computation, iterations, and L2 norm size. In this thesis, we assume that an optimal class pair $(s^*, t^*)$ exists. This is a weak assumption and more work on a detection statistic should be completed to determine if a network has been attacked at all.
The optimal perturbation is formed by gradient descent on the following objective function until group misclassification is achieved in (9).

\[ J_{st}(\tilde{v}) = -\frac{1}{|D_s|} \sum_{x \in D_s} p(t|(x + \tilde{v}) \right) \]

This objective function reads taking the sum over samples from source class s of the output of the classifier at the target class node. The perturbation optimization is shown in algorithm 3 below. As a note, because the DNNs used in this thesis utilized the softmax activation function, the output prior to this activation is used instead of the output of the softmax. This is required as the output of softmax layer is saturated (the network is very sure it is correct). Taking the gradient after the softmax activation leads to poor results. Taking the output prior to the softmax activation leads to a well-formed perturbation.

**Algorithm 3: Perturbation Optimization**

1. Initialization: \( \tilde{v} = 0, p = \frac{1}{|D_s|} \sum_{x \in D_s} 1(f(x) = t) \)

2. while \( p < \pi \)

3. \( \tilde{v} = \tilde{v} - \alpha \times J_{st}(\tilde{v}) \)

4. \( p = \frac{1}{|D_s|} \sum_{x \in D_s} 1(f(x + \tilde{v}) = t) \)
With algorithm 3, there are a few notes and hyper parameters that need explained. The misclassification fraction, $p$, is updated on each iteration with the current perturbation $v$. 2nd, the gradient of objective function in 10 is a sum of gradients on each individual sample from the source class. 3rd, $\alpha$ is the size to weight the gradient of all samples in source class and $\alpha$ should be chosen very small. As will be shown in the results section, $\alpha$ is set to 0.00001 and is shown to have success. The parameter $\alpha$, should be carefully chosen as if $\alpha$ is too small than the computation time to calculate $p$ and the perturbation will be quite long. If $\alpha$ is too large, then the perturbation for the actual attacked class pair $(s^*, t^*)$ will be much larger than is required. When taking the norm of this perturbation, it could be larger than a none attacked pair $(s, t)$ resulting in a failed detection. Lastly, an upper limit on the norm of the perturbation for each class pair is chosen as some class pairs will require a very large perturbation causing the algorithm to run for quite a long time. This is assumed both through experimentation and the hypothesis that the actual class pair in the attack $(s^*, t^*)$ will have a much smaller norm than other class pairs.

Because we assume a class pair is attacked, the lowest L2 norm in all class pairs will be the class pair chosen in the attack.

4.3 Results

With the perturbation method used in chapter 3, the detection method described in chapter 4 was tested on both an LSTM and standard feedforward neural network as described in chapter 2. The results are as follows.

1st, the LSTM network was tested. The hyper parameters used are $\alpha = 0.00001$, $\pi = 0.8$, and a L2 norm upper limit of 1.2. The following optimal perturbation for the attacked class pair
$(s^*, t^*)$ for all nine variable time series is shown in figure 39. Figure 40 demonstrates only one variable for emphasis.

![Estimated Perturbation of Attack Class Pair](image)

**Figure 39: Estimated Perturbation of Attack Class Pair, LSTM**
Examining figure 39 and 40, the optimal perturbation has extreme perturbation values toward the end of the time series. A hypothesis is that the LSTM has memory units and thus the last few LSTM time steps (associates with later time points) are the most important time steps for classification of this dataset. This hypothesis is born from the gradient being much larger at the end of the time series input resulting in the end of time series and the end of the LSTM nodes being most deterministic for classification. Comparing to a standard feedforward network, in figure 41, the artifact from the LSTM perturbation is not shown in the standard feedforward network. The standard feedforward optimal perturbation for \((s^*, t^*)\) is similar to the perturbation designed in chapter 3. The optimal perturbation from the LSTM network does not create a perturbation similar to the perturbation designed in chapter 3.
To compare \((s^*, t^*)\) perturbation, the estimated perturbation for a non-attack class pair \((s, t)\) is shown in figure 42 and 43.
Figure 42: Estimated Perturbation of Non-attack Class Pair, LSTM

Figure 43: Estimated Single Variable Perturbation of Non-Attack Pair, LSTM
Comparing the perturbation of attack class pair \((s^*, t^*)\) from figure 39 with a non-attack class pair \((s, t)\) from figure 42, the perturbation for non-attack class pair, by visual inspection and L2 norm metric, is much greater than in the attacked class pair. The L2 norm of the attacked class pair is \(~0.40\) while the L2 norm for non-attacked class pair is much greater \(>1.2\) in which the algorithm would normally limit to 1.2. This perturbation has the similarly large energy towards the end of the time series. This is probably occurring from the memory of the LSTM recurrent neural network as discussed before. To really understand why, more analysis and experiments should be performed.

Figure 44 demonstrates a non-attack class pair for a standard feedforward network. Comparing to the attacked pair in figure 41, the visual size and L2 norm are much higher than the attacked pair. Again, the feedforward network has a similar perturbation to the form in chapter 3.
Last, the optimal perturbation technique in algorithm 3 was tested on source class 1 to all other classes (2 through 6). The method successfully determined the attacked source and target class pair for each attack. This required training five separate networks with the appropriate attack class pair. As can be seen from the results in tables 2-6, the attack class pair \((s^*, t^*)\) all had much smaller L2 norm values than the rest of class pairs. With the assumption each network was attacked, the method in algorithm 3 was successful.

An example of the L2 norm matrix is shown in table 2 for attack class pair \((1,2)\). The table is formatted as follows. The horizontal class row indicates source class, and the vertical class column indicates the target class. For this experiment, the L2 norm upper limit is 1.2 to
keep computation time reasonable ~1/2 day per test. The method successfully has the smallest L2 norm for the class pair (1,2) at 0.401 highlighted in green. Note, the class pair (5,2) has a relatively small L2 norm as well which could indicate inherent class similarity between 5 and 2 or the actual perturbation in (1,2) is similar to (5,2). The following five tables shows the L2 norm for each class pair, accuracy on a non-poised test set, and target success percent.

Accuracy on non-poisoned test set: 91%.
Target success from poisoning source class 1 to target class 2: 92%.

Table 2: L2 Norm Matrix for Attacked Class Pair (1,2)

<table>
<thead>
<tr>
<th>Class</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100000</td>
<td><strong>0.401</strong></td>
<td>1.201</td>
<td>1.204</td>
<td>1.214</td>
<td>1.21</td>
</tr>
<tr>
<td>2</td>
<td>1.209</td>
<td>100000</td>
<td>1.201</td>
<td>1.211</td>
<td>1.215</td>
<td>1.202</td>
</tr>
<tr>
<td>3</td>
<td>1.206</td>
<td>1.209</td>
<td>100000</td>
<td>1.207</td>
<td>1.204</td>
<td>1.201</td>
</tr>
<tr>
<td>4</td>
<td>1.219</td>
<td>1.216</td>
<td>1.222</td>
<td>100000</td>
<td>1.213</td>
<td>1.221</td>
</tr>
<tr>
<td>5</td>
<td>1.236</td>
<td><strong>0.882</strong></td>
<td>1.212</td>
<td>1.207</td>
<td>100000</td>
<td>1.205</td>
</tr>
<tr>
<td>6</td>
<td>1.222</td>
<td>1.205</td>
<td>1.220</td>
<td>1.206</td>
<td>1.212</td>
<td>100000</td>
</tr>
</tbody>
</table>
Accuracy on non-poisoned test set: 93.5%.
Target success from poisoning source class 1 to target class 3: 99%.

Table 3: L2 Norm Matrix for Attacked Class Pair (1,3)

<table>
<thead>
<tr>
<th>Class</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0000</td>
<td>1.200</td>
<td><strong>0.384</strong></td>
<td>1.215</td>
<td>1.203</td>
<td>1.202</td>
</tr>
<tr>
<td>2</td>
<td>1.202</td>
<td>1.0000</td>
<td>1.206</td>
<td>1.220</td>
<td>1.203</td>
<td>1.201</td>
</tr>
<tr>
<td>3</td>
<td>1.201</td>
<td>1.203</td>
<td>1.0000</td>
<td>1.214</td>
<td>1.209</td>
<td>1.208</td>
</tr>
<tr>
<td>4</td>
<td>1.206</td>
<td>1.209</td>
<td>1.28</td>
<td>1.0000</td>
<td>1.217</td>
<td>1.200</td>
</tr>
<tr>
<td>5</td>
<td>1.206</td>
<td>1.206</td>
<td>1.218</td>
<td>1.213</td>
<td>1.0000</td>
<td>1.202</td>
</tr>
<tr>
<td>6</td>
<td>1.208</td>
<td>1.214</td>
<td>1.223</td>
<td>1.217</td>
<td>1.206</td>
<td>1.0000</td>
</tr>
</tbody>
</table>

Accuracy on non-poisoned test set: 93.5%.
Target success from poisoning source class 1 to target class 4: 96%.

Table 4: L2 Norm Matrix for Attacked Class Pair (1,4)

<table>
<thead>
<tr>
<th>Class</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0000</td>
<td>1.212</td>
<td>1.201</td>
<td><strong>0.379</strong></td>
<td>1.203</td>
<td>1.203</td>
</tr>
<tr>
<td>2</td>
<td>1.219</td>
<td>1.0000</td>
<td>1.203</td>
<td>1.216</td>
<td>1.214</td>
<td>1.201</td>
</tr>
<tr>
<td>3</td>
<td>1.205</td>
<td>1.203</td>
<td>1.0000</td>
<td>1.209</td>
<td>1.213</td>
<td>1.203</td>
</tr>
<tr>
<td>4</td>
<td>1.205</td>
<td>1.206</td>
<td>1.230</td>
<td>1.0000</td>
<td>1.215</td>
<td>1.200</td>
</tr>
<tr>
<td>5</td>
<td>1.212</td>
<td>1.207</td>
<td>1.223</td>
<td>1.205</td>
<td>1.0000</td>
<td>1.203</td>
</tr>
<tr>
<td>6</td>
<td>1.214</td>
<td>1.239</td>
<td>1.202</td>
<td>1.208</td>
<td>1.209</td>
<td>1.0000</td>
</tr>
</tbody>
</table>
Accuracy on non-poisoned test set: 92.2%.

Target success from poisoning source class 1 to target class 5: 94%.

Table 5: L2 Norm Matrix for Attacked Class Pair (1,5)

<table>
<thead>
<tr>
<th>Class</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.000000</td>
<td>1.204</td>
<td>1.220</td>
<td>1.216</td>
<td>0.586</td>
<td>1.205</td>
</tr>
<tr>
<td>2</td>
<td>1.207</td>
<td>1.000000</td>
<td>1.203</td>
<td>1.211</td>
<td>1.212</td>
<td>1.207</td>
</tr>
<tr>
<td>3</td>
<td>1.206</td>
<td>1.207</td>
<td>1.000000</td>
<td>1.204</td>
<td>1.206</td>
<td>1.233</td>
</tr>
<tr>
<td>4</td>
<td>1.230</td>
<td>1.209</td>
<td>1.204</td>
<td>1.000000</td>
<td>0.9935</td>
<td>1.202</td>
</tr>
<tr>
<td>5</td>
<td>0.7834</td>
<td>1.207</td>
<td>1.201</td>
<td>1.173</td>
<td>1.000000</td>
<td>1.205</td>
</tr>
<tr>
<td>6</td>
<td>1.205</td>
<td>1.203</td>
<td>1.207</td>
<td>1.200</td>
<td>1.232</td>
<td>1.000000</td>
</tr>
</tbody>
</table>

Accuracy on non-poisoned test set: 92.7%.

Target success from poisoning source class 1 to target class 6: 94%.

Table 6: L2 Norm Matrix for Attacked Class Pair (1,6)

<table>
<thead>
<tr>
<th>Class</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.000000</td>
<td>1.201</td>
<td>1.208</td>
<td>1.200</td>
<td>1.204</td>
<td>0.454</td>
</tr>
<tr>
<td>2</td>
<td>1.211</td>
<td>1.000000</td>
<td>1.209</td>
<td>1.209</td>
<td>1.207</td>
<td>1.213</td>
</tr>
<tr>
<td>3</td>
<td>1.200</td>
<td>1.202</td>
<td>1.000000</td>
<td>1.208</td>
<td>1.209</td>
<td>1.210</td>
</tr>
<tr>
<td>4</td>
<td>1.201</td>
<td>1.205</td>
<td>1.206</td>
<td>1.000000</td>
<td>1.219</td>
<td>1.200</td>
</tr>
<tr>
<td>5</td>
<td>1.212</td>
<td>1.214</td>
<td>1.206</td>
<td>0.870</td>
<td>1.000000</td>
<td>0.997</td>
</tr>
<tr>
<td>6</td>
<td>1.221</td>
<td>1.209</td>
<td>1.229</td>
<td>1.212</td>
<td>1.200</td>
<td>1.000000</td>
</tr>
</tbody>
</table>
Chapter 5 Conclusion

This thesis focused on imperceptible data poisoning adversarial attacks in the LSTM deep recurrent neural network and the defense of such attacks. The idea behind this attack is to fool a classifier into classifying a source class $s^*$ to a target class $t^*$. The two methods, described in chapter 3, are a small norm additive perturbation stretching across the entire time series and a rather large additive perturbation at a single time step. As shown in the results, these types of attacks successfully trick an LSTM classifier on learning a backdoor pattern. With this success interesting properties emerge.

In chapter 3, the LSTM deep neural network architecture is much harder to fool than a standard feedforward network. The LSTM network requires larger perturbations and more training samples to successfully fool the network. This naturally arises from the LSTM’s ability to pass relevant past information to the present input time step. In addition, the size of the hidden layer and number of epochs needed to successfully learn the backdoor pattern during training is much higher in a LSTM than a standard feedforward network. Likewise, with similar computational complexity, feedforward networks have a higher poison accuracy than LSTM networks given perturbation size and source sample poisoned. Evidently, the way a LSTM trains and its recurrent architecture led to a natural defense against backdoor attacks. For example, a smaller size LSTM with only a few epochs could successfully classify a non-poisoned dataset to an accuracy of 90%. When a backdoor pattern is injected, with a reasonable perturbation size and source training class size, into this same architecture, the LSTM is not fooled. It requires both increasing the number of hidden layer LSTM units and epochs to successfully attack.
In chapter 4, a defense method was introduced in which a reverse engineering gradient-based approach was taken. Assuming the network was attacked, the defense method was successful in determining the attacked class pair \((s^*, t^*)\). Again, an interesting difference arises between LSTM and a standard feedforward network. When taking the gradient of the output with respect to the input, a standard feedforward designed an optimal perturbation similar to the perturbation used in the attack. Oppositely, the LSTM crafted perturbations which have high additive values near the end of the time series input. This is only possible if the gradient of the output with respect to the input is much higher at the end of the time series. Again, this probably occurs because of the nature in which LSTMs train and their ability to pass relevant information to future time steps. Even though the perturbation is extreme in the tail end of the time series, the defense method still correctly distinguishes the source target pair \((s^*, t^*)\).

For future work, more experiments should be performed to fully understand why the LSTM produces higher gradients in the latter part of a time series (when taking the gradient of the output with respect to the input). In addition to the optimal perturbation, a defense method should be developed without the assumption the network was attacked. This might require a detection inference procedure in combination with the optimal perturbation technique shown in this work. An example order-statistic p-value detection inference is found in [2].
Bibliography


