ENHANCING RESILIENCE AGAINST ADVERSARIAL ATTACKS OF DEEP NEURAL NETWORKS USING EFFICIENT TWO-STEP ADVERSARIAL DEFENSE

A Thesis

by

TING-JUI CHANG

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Peng Li
Committee Members,	Paul Gratz
	Zhangyang Wang
Head of Department,	Miroslav M. Begovic

December 2018

Major Subject: Computer Engineering

ABSTRACT

In recent years, deep neural networks have demonstrated outstanding performance in many machine learning tasks. However, researchers have discovered that these state-of-the-art models are vulnerable to adversarial examples: legitimate examples added by small perturbations which are unnoticeable to human eyes. Adversarial training, which augments the training data with adversarial examples during the training process, is a well known defense to improve the robustness of the model against adversarial attacks. However, this robustness is only effective to the same attack method used for adversarial training. Madry has suggested that effectiveness of iterative multi-step adversarial attacks and particularly that projected gradient descent (PGD) may be considered the universal first order adversary and applying the adversarial training with PGD implies resistance against many other first order attacks. However, the computational cost of the adversarial training with PGD and other multi-step adversarial examples is much higher than that of the adversarial training with other simpler attack techniques. In this work, we show how strong adversarial examples can be generated only at a cost similar to that of two runs of the fast gradient sign method (FGSM), allowing defense against adversarial attacks with a robustness level comparable to that of the adversarial training with multi-step adversarial examples. We empirically demonstrate the effectiveness of the proposed two-step defense approach against different attack methods and its improvements over existing defense strategies.

DEDICATION

To my father Kuo-Ming Chang and my mother Chin-Yen Lin for their support during the past two years.

ACKNOWLEDGMENTS

I would like to thank my parents. They gave me strong support both financially and mentally in the period of pursuing this degree.

I would also like to thank my committee chair, Dr. Peng Li. He is always very professional in providing me with academic help. I would like to give my thanks to my committee members, Dr. Paul Gratz and Dr. Zhangyang Wang, for their guidance and support throughout the course of this research.

Thanks also go to my colleague Yukun He for spending lots of time together making this research work, and other friends and the department faculty and staff for making my time at Texas A&M University a great experience.

Finally, thanks to everyone in this university who gave me support.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supervised by Professor Peng Li and Professor Paul Gratz of the Department of Electrical & Computer Engineering and Professor Zhangyang Wang of the Department of Computer Science.

The computation resource used for data analysis in this research is supported by High Performance Research Computing at Texas A&M University.

Funding Sources

Graduate study was supported by an in-state tuition fellowship from Department of Electrical & Computer Engineering at Texas A&M University.

NOMENCLATURE

X	Set of Clean Inputs
Y	Set of Clean Outputs
X^*	Set of Perturbed Inputs
Y^*	Set of Perturbed Outputs
$F(X)$	Model Function
δ_X	Noise Added to Clean Inputs
R^d	Real Coordinate Space of d dimension
ℓ_∞	L-Infinity Norm
x	Clean Input
$x_a dv$	Adversarial Version of A Clean Input
δ	Perturbed noise
$J(\theta, x, y)$	Loss Function with Input x, Label y and Model Parameters θ
ε	Noise Level
α	Weighting Value for Basic Adversarial Training
$\mathcal{D}$	Underlying Data Distribution
E	Expectation
N	Number of Inputs In A Minibatch
CE	Cross Entropy
$f(x, \theta)$	Model Output Function
D	Dissimilarity Metric
k	Number of Steps for Iterative Adversarial Attack
λ	Weighting Value of The Dissimilarity Loss Term

g_1	Noise Direction of Adversarial Attack
g_2	Direction Orthogonal to Noise Direction

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

1. INTRODUCTION

1.1 Artificial Neural Network

With the great advance of computing resources and other technologies, we now can see many machine-learning techniques applied in different applications for improving the quality of life. Among all machine-learning techniques, deep learning perhaps is the most successful one which can achieve performance comparable to human beings. To dig out the essence of deep learning, we have to direct our attention to its core which is artificial neural network (ANN), a computational model inspired by the way how biological neural networks in the human brain process information.

1.1.1 History of ANN

In 1943, Warren McCulloch and Walter Pitts [24], first created a computational model for neural networks based on mathematics and algorithms called threshold logic, which is a linear classifier capable of differentiating inputs of two classes based on the signs of their outputs. From this work, the following neural network research split into two approaches. One focused on the biological processes in brains while the other one focused on applying neural networks on artificial intelligence. Then in 1949, Donald Olding Hebb [25], based on the mechanism of neural plasticity, proposed a new learning hypothesis called Hebbian Learning, which claimed that an increase in synaptic efficacy arises from a presynaptic cell's repeated and persistent stimulation of a postsynaptic cell and researchers started applying this idea to computational models with Turing's B-type machines. Based on Hebbian Learning, Farley and Clark [26] first used computational machines, then called "calculators", to simulate a Hebbian network. In 1958, Frank Rosenblatt [27] created an algorithm called perceptron, which is a type of linear classifier making its predictions based on a linear predictor function combining a set of weights with the feature vector. This algorithm was implemented in custom-built hardware as the "Mark 1 perceptron", which was designed for image recognition. The New York Times even reported the perceptron to be "the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence." However, Marvin Minsky and Seymour Papert [28] discovered two key issues with the implementation of perceptrons in their machine learning research in 1969. The first one was that even perceptrons of a single layer were able to handle some circuits like AND, OR, and NOT, they were incapable of processing the XOR circuit. The Second was that computers didn't have enough processing power to effectively handle the work required by large neural networks. These drawbacks, along with a variety of other factors, made researchers put less attention on neural networks until 1980s. A key trigger for renewed interest in neural networks and learning was Werbos's (1975) backpropagation algorithm [29] that effectively solved the exclusive-or problem by making the training of multi-layer networks feasible and efficient. Backpropagation distributed the error term back up through the layers, by modifying the weights at each node. And also in 1982, John Hopfield published his work known as Hopfield Network [30], which severs as content-addressable memory systems with binary threshold nodes. In 1986, David E. Rumelhard, Geoffrey E. Hinton and Ronald J. Williams [31] got good implementation results based on the backpropagation algorithm, which again drew peoples' attention back to multi-layer perceptrons. But this field again went downhill as researchers found it difficult to train networks with multi-layers especially due to the vanishing gradient problem. As errors propagate from layer to layer, they shrink exponentially with the number of layers, impeding the tuning of neuron weights that is based on those errors, particularly affecting deep networks. At the same time, support vector machines and other, much simpler methods such as linear classifiers gradually overtook neural networks in machine learning popularity. But later the advent of deep learning methods in the early 2000's allowed the training of networks with many hidden layers. Furthermore in 2006, Geoffrey E. Hinton [32] proposed learning a high-level representation using successive layers of binary or real-valued latent variables with a restricted Boltzmann machine to model each layer, which works well especially in areas like dimension reduction, classification, filtering, feature-extraction. This restricted Boltzmann machine works still very well when the multi-layer structure is applied, besides, it can also be combined with multi-layer perceptrons, resulting in the so called deep neural network. From that time, more newly proposed model like convolutional neural network, along with the use of GPUs and distributed computing, open the new era of neural networks today.

1.1.2 Multi Layer Perceptron

Multi Layer Perceptron is a simple but important type of Artificial Neural Network. By checking the structure and the training process of Multi Layer Perceptron, we can get the basic understanding about the background knowledge of other deep neural networks.

Structure of Perceptron

1. **A Single Neuron** The basic computation unit in a neural network is neuron, sometimes called node or unit. Each node receives the inputs from other node or external sources and then generates an output. Each input comes with an associated weight, which is assigned based on the input's relative importance. The node applies a function f to the weighted sum of its inputs. Figure1.1 shows how a neuron works.

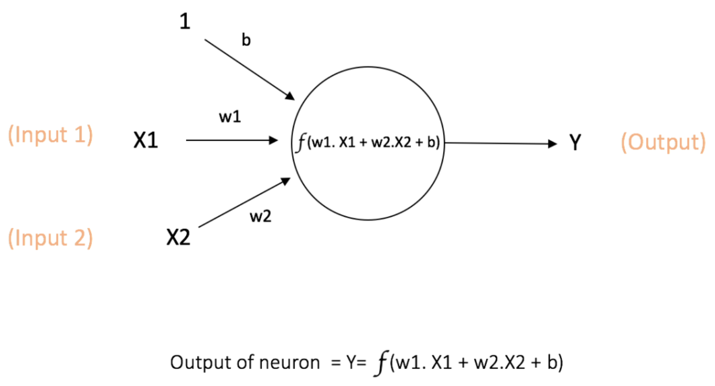


Figure 1.1: A single neuron, reprinted from [1]

The above network takes numerical inputs X_1 and X_2 associated with weights w_1 and w_2. Additionally, there is another input 1 with weight b (called the Bias) associated with it.

The output Y from the neuron is computed as shown in the Figure1.1. The function f is non-linear and is called the Activation Function. The purpose of the activation function is to introduce non-linearity into the output of a neuron in order to better learn the representations of real world data which is often non-linear. Figure1.2 shows some typically used activation functions.

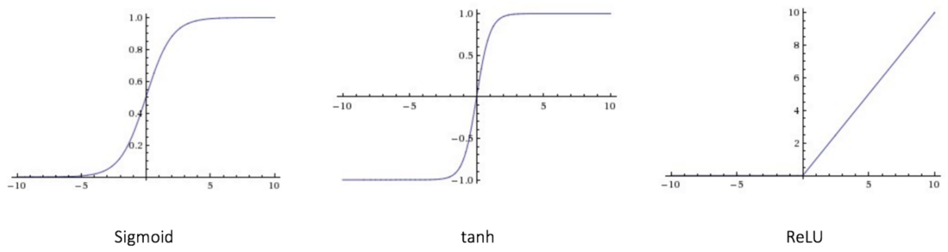


Figure 1.2: Different activation functions, reprinted from [2]

2. **Feedforward Neural Network** The feedforward neural network is the first and simplest type of artificial neural network. It consists of nodes arranged in layers. Nodes from adjacent layers have connections or edges between them and each edge has an associated weight. Figure1.3 shows the basic structure of a feedforward neural netowrk. A feedforward neural network has three different types of nodes:

   (a) **Input Nodes** − The input nodes are the ports of a network to receive the information from the outside world and the layer consisting of input nodes is referred to as the input layer. No computation is performed in any of the Input nodes − they just pass on the information to the hidden nodes.

   (b) **Hidden Nodes** − The hidden nodes have no direction connection with the outside world. They receive the information sent from the preceding layer, perform computations and transfer the processed information to the next layer. A collection of hidden nodes forms a "Hidden Layer".

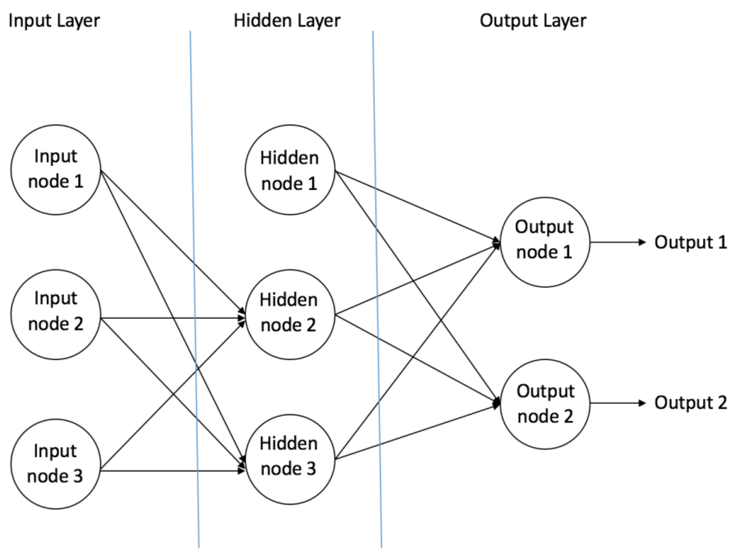


Figure 1.3: The structure of feedforward network, reprinted from [3]

(c) **Output Nodes** − The Output nodes are collectively referred to as the " Output Layer " and are responsible for computations and transferring information from the network to the outside world.

In a feedforward network, the information moves only in one direction − from the input nodes, through the hidden nodes and to the output nodes. Based on the number of hidden layers, a feedforward networks can be defined as two types:

(a) **Single Layer Perceptron** − The simplest feedforward networks which contains no hidden layers.

(b) **Multi Layer Perceptron** − The feedforward networks which contains one ore more hidden layers.

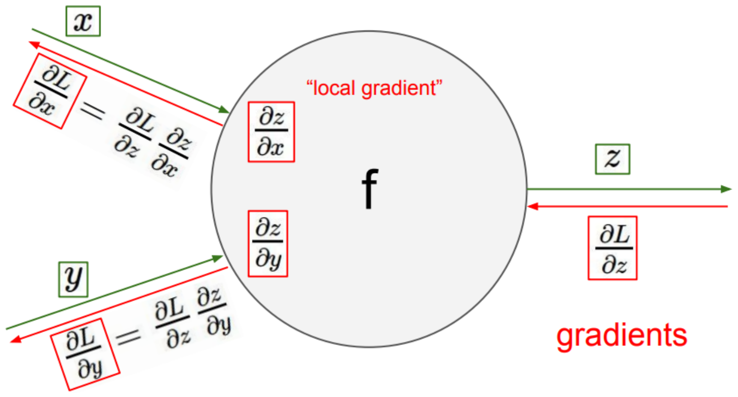
Learning Process of Multi Layer Perceptron

A multi layer perceptron can be thought as a function which transforms the input data from the input domain to the output domain and the learning process of the model is actually to craft out the function with some desired property by tuning the weights of the model.

1. **The Loss Function** – The loss function is used as an indicator which quantifies how well the current model matches the final goal. Take a supervised classification problem as an example, the output of the model for a certain input is a probability vector, where each element shows how likely that input belongs to the corresponding class. For the supervised case, trainers in fact have the ground truth label, which tells which class the input is exactly from, and the loss is defined as the dissimilarity between the output vector and the ground truth label based on some metrics such as mean-squared error or cross entropy.
2. **Learning Algorithm** – Once the loss is defined, the learning process of a model becomes solving an optimization problem which tries to minimize the loss value. By applying the gradient descent method, which tunes the weights (the parameters of the model) based on the gradients of the loss value w.r.t. those weight, the goal can be achieved. But how to compute the corresponding gradients for the weights especially for the structure of a feedforward neural network? The proposed way is called "Backpropagation" – For every input in the training dataset, the ANN is activated and its output is observed. This output is compared with the desired output that we already know to get the loss, and the gradient of the loss, or called the error is "propagated" back to the previous layer. This error is noted and the weights are "adjusted" accordingly. This training process is repeated until the loss value is below a predetermined threshold. The process of Backpropagation is exactly doing the chain-rule. Figure1.4 illustrates the process of Backpropagation for a node.

1.2 Deep Learning

Deep Learning which refers to networks with the structure of a cascade of multiple layers of nonlinear processing units, has been applied to fields including computer vision, speech recognition, natural language processing, audio recognition, etc, where it has produced results comparable to and in some cases superior to human experts. In this section, we briefly discussed two representative deep learning networks: convolutional neural network (CNN) and Long Short-Term Memory



x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients

Figure 1.4: Backpropagation of a single node, reprinted from [4]

(LSTM) which are widely used in computer vision and speech recognition respectively.

1.2.1 Convolutional Neural Network

Convolutional Neural Networks (CNNs) are a type of neural networks which have been proven effectively in areas such as image recognition and classification due to their great ability to extract high level features from the raw images. In this subsection, we would overview the basic knowledge of CNNs based on a simple CNN similar to LeNet which is the very first convolutional neural network.

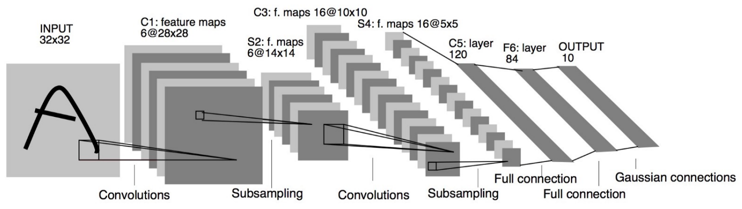


Figure 1.5: LeNet5, reprinted from [5] ©1998 IEEE

The LeNet5 Architecture (1990s)

This network is one of the very first convolutional neural networks which helped propelling the field of Deep Learning and was named as LeNet5 by Yann LeCun in 1994. At that time, it was applied by several banks to recognise hand−written numbers on checks digitized in 32x32 pixel images. (Figure1.5)

The convolutional neural network in Figure1.6 is similar to the architecture of the original LeNet and classifies the input image into four categories: dog, cat, boat and bird. In this basic CNN, there are four main operations: convolution, non-linearity (ReLU), pooling, and classification (the latter fully connected layers). There four operations are actually the building blocks for every convolutional neural network. The following would detail what each operation does.

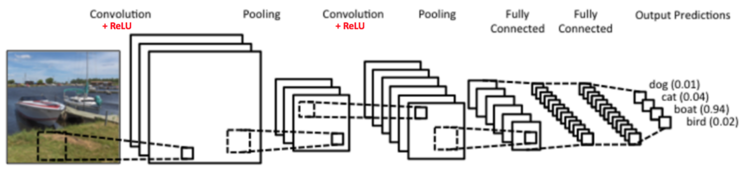
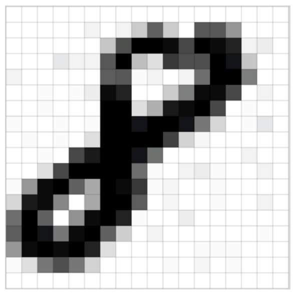


Figure 1.6: A simple CNN, reprinted from [6]

Here, we discuss the operations of a convolutional neural network based on the image classification problem, where every image is represented as a matrix of pixel values like what Figure1.7 shows.

The Convolution Step

CNNs get its name from the convolution operator used in the convolution step. For image classification, in order to extract features from the raw images, the convolution operator is applied to preserve the spatial relationship between pixels by using the so called "filter" or feature detector, which is also represented by a small square of pixel values. Figure1.8 shows a single step of 2D convolution computation, which is the summation of the results



```
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   1  12   0  11  39 137  37   0 152 147  84   0   0   0
  0   0   1   0   0   0  41 160 250 255 235 162 255 238 206  11  13   0
  0   0   0  16   9   9 150 251  45  21 184 159 154 255 233  40   0   0
 10   0   0   0   0   0 145 146   3  10   0  11 124 253 255 107   0   0
  0   0   3   0   4  15 236 216   0   0  38 109 247 240 169   0  11   0
  1   0   2   0   0   0 253 253  23  62 224 241 255 164   0   5   0   0
  6   0   0   4   0   3 252 250 228 255 255 234 112  28   0   2  17   0
  0   2   1   4   0  21 255 253 251 255 172  31   8   0   1   0   0   0
  0   0   4   0 163 225 251 255 229 120   0   0   0   0   0  11   0   0
  0   0  21 162 255 255 254 255 126   6   0  10  14   6   0   0   9   0
  3  79 242 255 141  66 255 245 189   7   8   0   0   5   0   0   0   0
 26 221 237  98   0  67 251 255 144   0   8   0   0   7   0   0  11   0
125 255 141   0  87 244 255 208   3   0   0  13   0   1   0   1   0   0
145 248 228 116 235 255 141  34   0  11   0   1   0   0   0   1   3   0
 85 237 253 246 255 210  21   1   0   1   0   0   6   2   4   0   0   0
  6  23 112 157 114  32   0   0   0   0   2   0   8   0   7   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

Figure 1.7: An image is represented as a matrix of pixel values, reprinted from [7]

from the element-wise multiplication between the filter and a small region of the raw image. Figure1.9 shows the result after the entire 2D convolution computation process.

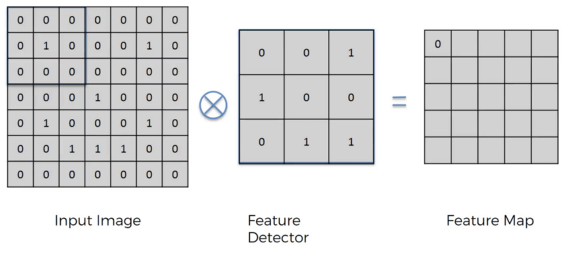


Figure 1.8: One step of the 2D convolutional computation, reprinted from [8]

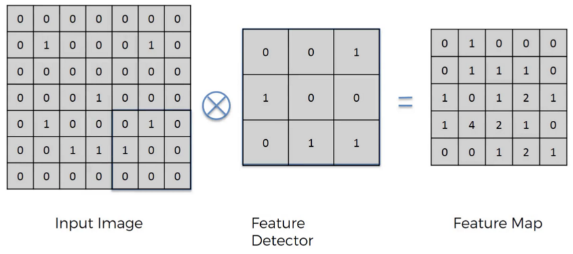


Figure 1.9: The result of the 2D convolutional computation, reprinted from [9]

A filter, or called "kernel" or "feature detector", typically represents an certain type of features such as edges. In the convolution computation, it plays the role of detecting if there is a sub-region of the raw image corresponding the property of the filter and then results in the "feature map", which shows the strength of the correspondence. Figure1.10 shows the process of how a filter extracts the feature information.

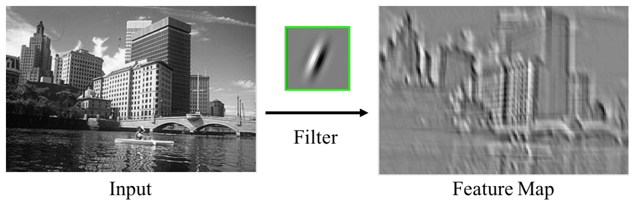


Figure 1.10: Example of the convolution operation, reprinted from [10]

Non Linearity Step

Like the case of multi layer perceptrons, in order to make the model better fit the non-linear property of the data from the outside world, non-linear functions such as sigmoid, tanh, ReLU are added as the activation functions. An example of ReLU operation is shown in Figure1.11.

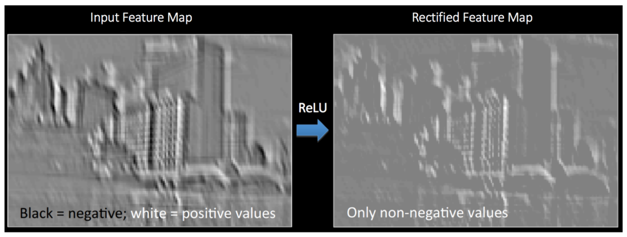


Figure 1.11: ReLU operation, reprinted from [11]

Pooling Step

Spatial pooling, or called as downsampling, reduces the dimensionality of each feature map but retains the most important information for the sake of accelerating the training process. Spatial pooling has several different types such as max, average, sum etc.

In case of Max pooling, a spatial neighborhood (for instance, a 2x2 window) is first defined and the largest value of each spatial neighborhood is selected from the incoming feature map. Figure1.12 illustrates the process of max pooling.

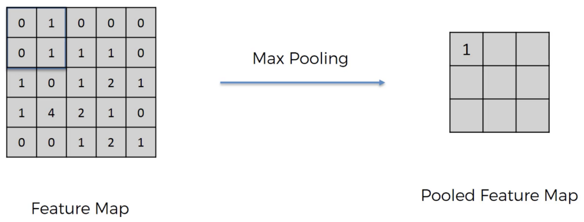


Figure 1.12: Max pooling, reprinted from [12]

Besides accelerating the training process, spatial pooling also offers other benefits

- ease the effect of overfitting by reducing the number of parameters in the model.
- makes the network invariant to small transformations, distortions and translations in the input image.
- makes the results invariant to small translation of few pixels.

Fully Connected Layer

The Fully Connected layer is a conventional multi layer perceptron that uses a softmax activation function in the output layer. The term "fully connected" means that every node in a layer is connected to every node in the adjacent layers. After getting the high level features of the raw image from the previous convolution and pooling layers, the fully connected layer is applied to learn the non-linear combination of these features in order to do the final classification job.

Compared to pure multi layer perceptrons, with the usage of convolution and pooling layers, CNNs can effectively extract more representative features from the raw image and therefore plays well at vision-based machine learning tasks.

1.2.2 Recurrent Neural Network

When it comes to jobs such as text generation or sentence translation where the output does not just depend on the current input (like text generation, the current output depends on the previous words in the same sentence), traditional neural networks do not work very well. Recurrent neural networks (RNNs), unlike traditional artificial neural networks, can solve this problem by processing sequences of inputs with their internal state memory. The introduction of the basic RNNs is as follows:

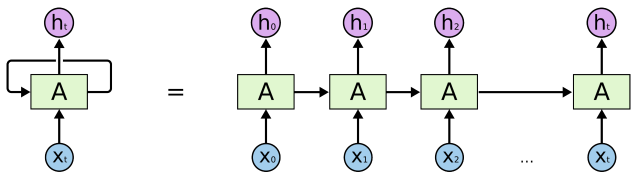


Figure 1.13: The loop structure of basic RNN, reprinted from [13]

- **The Loop Structure** Figure1.13 shows the basic structure of RNNs where A is a chunk of neural network, x_t is the input of tth moment and h_t is the corresponding output. A loop allows information to be passed from one step of the network to the next. By applying this loop-like structure, the current output of RNNs can take the previous inputs into consideration, which makes it more suitable for dealing with data sets which have dependency across time. It has been shown that RNNs work very well in applications like speech recognition, language modeling, translation, image captioning, etc.

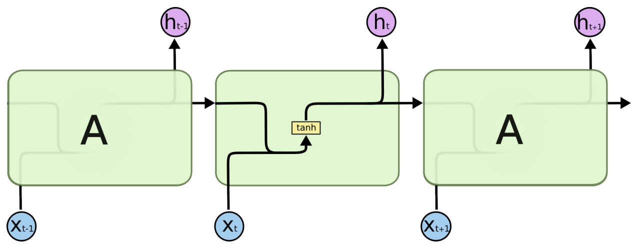


Figure 1.14: The repeating module in a standard RNN contains a single layer, reprinted from [14]

- **The Problem of Long-Term Dependencies** Like Figure1.14 shows, for the traditional RNNs, the inside of A, which is called RNN cell, has the structure as the following: the current input is concatenated with the previous one, then passed through some linear transformation (WX + b) and the activation function, which is the tanh here, then further passed to the output and the next operation. However, the this basic RNN has a shortcoming that each time the previous information passes the RNN cell, some information would be lost. If the current output depends on the information of a previous input which is from far back, then the basic RNN can not offer the needed message to make the correct decision.

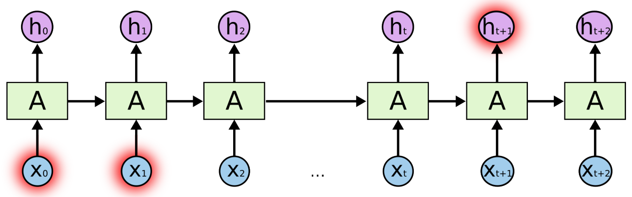


Figure 1.15: RNN can not maintain information from far way, reprinted from [15]

To solve this problem, a new structure called "Long Short Term Memory Network" (LSTM) was proposed.

1.2.3 Long Short Term Memory Network

LSTMs are a special kind of RNNs which are capable of learning long-term dependencies. They were introduced by Hochreiter Schmidhuber in 1997 and are now applied in many application areas and work tremendously well. LSTMs are explicitly designed to avoid the long-term dependency problem. With the designed structure shown in Figure1.16, LSTMs can decide which information to keep or to remove.

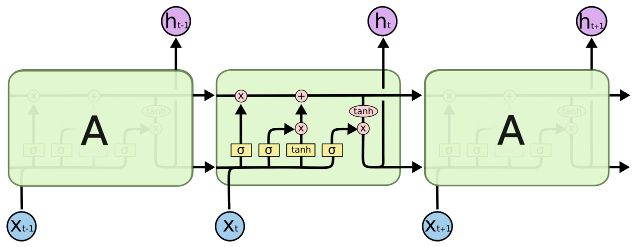


Figure 1.16: The structure of cell of the basic LSTM, reprinted from [16]

The key to LSTMs is the cell state, which is closely related to the current output and the future outputs. By applying the structures called gates, LSTMs can remove or add information to the cell state (Figure1.17). Gates refer to the way of optionally letting information through. Typically gates are composed of a sigmoid neural net layer and a pointwise multiplication operation.

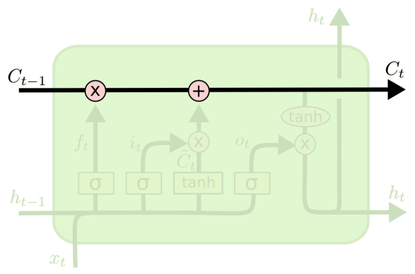
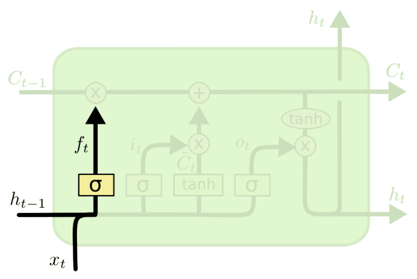


Figure 1.17: The cell state of the basic LSTM, reprinted from [17]

In the following, we would discuss the operation of the LSTM cell in three aspects: (1) removing the state information by the forget gate; (2) adding new state information; (3) the choice for the output.

- **Forget Gate** The first step in the cell operation of LSTMs is to decide which information to throw away from the previous cell state. This decision is made by a sigmoid neural layer called the "forget gate layer". This layer looks at the current input s_t and the previous output h_{t-1}, then outputs a number between 0 and 1 for each element in the cell state vector C_{t-1}. "1" means "completely pass through" and "0" means "completely being blocked away". The process is shown in Figure1.18.

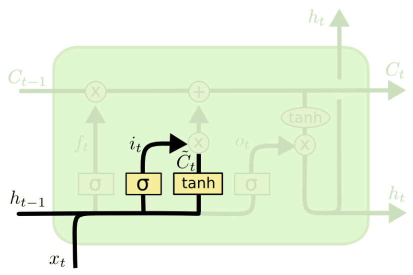


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure 1.18: The operation of forget gates, reprinted from [18]

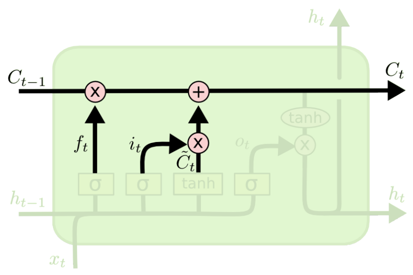
- **New State Information** The second step in the cell operation of LSTMs is to decide what new information to add to the cell state. This step consists of two parts. The first part is that a sigmoid neural layer called the "input gate layer" would decide which elements in the cell state vector will be updated. The second part is that a tanh layer would create a vector of new candidate values, $\tilde{C}_t$, that could be added to the state. The process is shown in Figure1.19.

  Combining the operations of these two step, LSTMs can remove some information from the previous state to get rid of the dependency and add the update information to some elements of the previous state to enhance the bonding. (Figure1.20)



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

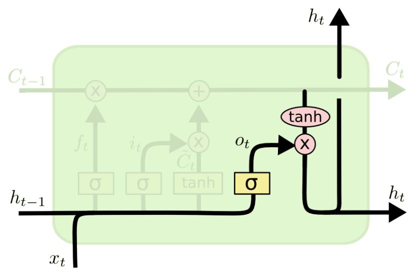
Figure 1.19: The operation of input gates, reprinted from [19]



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 1.20: The update of the cell state, reprinted from [20]

- **The Output Choice** The last step is to decide which parts of the cell state vector are going to be in the output. First a sigmoid layer would, based on the current input x_t and the previous output h_{t-1} to decide what parts of the cell state LSTMs are going to output. Then the cell state vector would be passed through the tanh activation function to increase non-linearity and multiplied with the output vector of the sigmoid layer. (Figure1.21)



$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 1.21: The decision for the output vector, reprinted from [21]

With the structural addition like forget gates and input gates, LSTMs successfully solve the long-term dependency problem and can even avoid the vanishing gradient problem when it comes to the training process. In practical usage, LSTMs have shown that they work very well especially in the tasks that require memories of events that happened thousands or even millions of discrete time steps earlier.

1.3 The Intrinsic Vulnerability of Deep Neural Network

Deep neural network(DNN) has altered the machine learning landscape and drawn lots of attention with outperforming other machine learning algorithms in many tasks due to the great improvement of the computational power and large datasets. There are many applications of DNN in different fields, such like speech recognition, visual object recognition and object detection and they all got better performance compared to the existing machine learning techniques. For instance, by taking advantage of large scale of dataset, DNN can now achieve the accuracy rates higher than many previous classification algorithms in image classification problem. The trend of applying DNN to solve many different kinds of missions in our daily lives actually motivates adversaries to manipulate DNN to attack the system, such as forcing DNN to misclassify the inputs. Especially, imperfections in the DNN make it vulnerable to the adversarial examples − inputs formed by applying small but intentionally worst-case perturbations to normal examples from the data set . Recent works in machine learning and security communities have shown that adversaries can force many machine learning models, including DNN, to produce adversary-selected outputs using carefully crafted inputs [23].

Specifically, adversaries can craft adversarial samples to make machine learning models to produce an output behavior of what they want, such as misclassification from one class to a certain class. This kind of inputs can be generated by adding a carefully chosen adversarial perturbation to a legitimate sample. Note that the adversarial samples are crafted during the test time after the DNN has been trained by the users, there is no requirement for adversaries to access and attack the training process. Algorithms crafting adversarial samples are designed to minimize the perturbation, thus making adversarial samples hard to distinguish from the legitimate samples ([23], [33]).

In addition, DNNs sharing similar structures may suffer from the same set of adversarial samples, which means that adversaries can find out from a relatively coarse neural network the adversarial samples which are functional to some more complex DNNs.

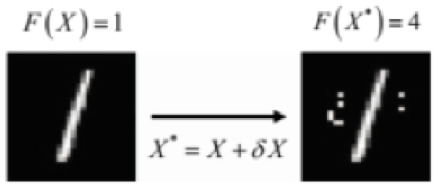


Figure 1.22: Image perturbed with adversarial noise, reprinted from [22] ©2016 IEEE

Figure 1.22 shows an example of how adversarial samples makes a system based on DNNs vulnerable. The left image is correctly classified by a trained DNN model F(X) as the digit one. By adding a carefully chosen adversarial perturbation δ_X to the legitimate sample, the adversarial sample which is the right image is generated and is misclassified by the DNN model F(X) as digit four.

To see why such misclassification is an important problem, consider DNN as a financial fraud detection system, an adversary able to craft misclassified inputs would profit from evading detection and cause a big chaos of the financial interaction. A more serious example is that, consider DNN as a commonly used system of autonomous cars. Systems based on DNN are used to recognize signs or other vehicles on the road. If perturbing the input of such systems, such as slightly altering the car's body, prevents DNN from classifying it as a moving vehicle, the car might not stop and be involved into a car accident with potentially disastrous consequence.

Thus, adversarial samples must be taken into account as designing security sensitive systems based on DNNs.

1.4 Adversarial Attack and Defense

Among many different applications using deep neural networks, my research mainly focuses on the problem of image classification and the convolutional neural network, one of the most promising neural network structures, is the main model applied in my research. Therefore, the following introduction of adversarial attack and defense would be explained in terms of the classification problem.

1.4.1 Adversarial Example

The general definition of an adversarial examples is like the following: Considering a DNN modeling a multidimensional function F: X $\rightarrow$ Y, where X is a raw feature vector and Y is an output vector. An adversarial sample X* can be constructed from a legitimate sample X by adding a perturbation vector via solving the following optimization problem:

$$\arg\min_{\delta_X} \|\delta_X\| \; s.t. F(X + \delta_X) = Y^*, \tag{1.1}$$

where X*=X + δ_X is the corresponding adversarial example of X, Y* is the desired adversarial output, and is a norm appropriate to measure the perturbation. The process of crafting a adversarial example is in fact solving a optimization problem and the final goal is that ideally, the adversary wants to make the resulting adversarial examples indistinguishable from the original legitimate examples under the human judgement.

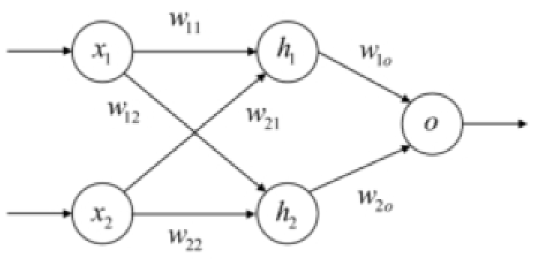


Figure 1.23: Simplified neural network of AND function, reprinted from [23] ©2016 IEEE

Taking an AND neural network in Figure 1.23 for example. Although this case is not a ConvNet, the underlying problem can still be demonstrated. The output space of this neural network is visualized in Figure 1.24. Figure 1.25 shows the derivative surface of the neural network function with respect to input neuron x_2

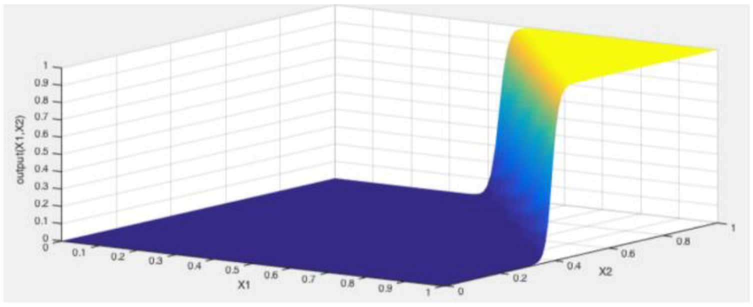


Figure 1.24: The output surface of the simplified neural network of AND function, reprinted from [23] ©2016 IEEE

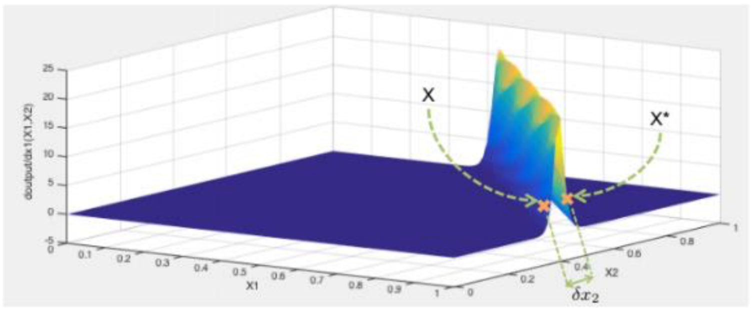


Figure 1.25: Forward derivative of the simplified AND neural network according to input neuron x_2, reprinted from [23] ©2016 IEEE

Consider a legitimate sample X = (1, 0.37) and an adversarial sample X* = (1, 0,43), which are both located near the spike in Figure 1.25. Although they only differ by a small amount (δx_2=0.05), they cause a significant change in the network's output, as F(X) = 0.11 and F(X*) = 0.95.

However, the solution of the optimization problem in Equation 1.1 is hard to solve due to the non-convexity and the non-linearity of DNNs. Instead of solving the optimization problem directly, some specific methods are designed for generating the adversarial examples based on the properties of DNNs. Take some representative attack methods or example, fast gradient sign method (FGSM) [33], which assumes the high linearity of DNNs and uses the gradient information of the cost function w.r.t. clean inputs to generates adversarial examples, Jacobian-based Saliency Map Algorithm (JSMA) [23], which builds a saliency map based on the gradient information of the original neural network and perturbs the legitimate input within limited amount of modified pixels, and Carlini and Wagner Attacks (CW) [34], which substitutes a set of suitable functions for the original goal function of the optimization problem and solves it with some existing optimization techniques.

The recent literature considered two types of threat models: black-box and white-box attacks. In black-box attacks, the attacker is assumed to have no access to the architecture and parameters of the model, whereas in white-box attacks, the attacker has complete access to such information.

1.4.2 Adversarial Defense

In response, several defenses have been proposed to mitigate the effect of adversarial attacks. These defenses were developed along three main directions: **(1)** expanding the training data to make the classifier more robustly learn the underlying function, e.g., by adversarial training which augments the training data set with adversarial examples generated by certain attack methods ([35]; [33]; [36]); **(2)** modifying the training procedure to reduce the gradients of the model w.r.t. the input such that the classifier becomes more robust to input perturbations, e.g., via input gradient regularization [37], or defensive distillation [22]; and **(3)** using external models as network add-ons when classifying unseen examples (feature squeezing [38], MagNet [39], and Defense-GAN) [40]).

1.5 Problems and Objectives

In this research, I keep exploring on the track of adversarial training. Adversarial training, a simple but effective method to improve the robustness of a deep neural net- work against white-box adversarial attacks, uses the same white-box attack mechanism to generate adversarial examples for augmenting the training data set. However, if the attacker applies a different attack strategy, adversarial training does not work well due to gradient masking [41].

Madry [42] has suggested the effectiveness of iterative multi-step adversarial attacks. In particular, it was suggested that projected gradient descent (PGD) may be considered as the strongest first-order attack so that the adversarial training with PGD can boost the resistance against many other first-order attacks. However, in the literature, a large number (e.g. 40) of steps of back propagation are typically used in the iterative attack method of PGD or its closely related variant iterative fast gradient (IFGSM) [36] to find strong adversarial examples to be used in each adversarial training step, incurring a prohibitively high computational complexity particularly for large DNNs or training datasets.

With the understanding of the advantage and disadvantage of iterative-attack-based adversarial training, there are three main objectives in this research as the following:

- Try to find a computationally efficient method to generate adversarial examples for each input example while effectively revealing the vulnerability of the learned classifier in the neighborhood of each clean data point.

- If the extra adversarial example is found, try to see by considering this kind of adversarial examples as part of a well-designed final loss function, whether the resulting model can be robust against both one-step and iterative white box attacks.

- Explore that by adopting other techniques such as soft labeling in the final loss function, whether the robustness of the model can be further improved.

1.6 Significance to Industry

Even though the application of deep neural networks achieves a great progress in many different machine learning tasks, the vulnerability of DNNs to adversarial attacks casts a shadow over user's confidence, especially when the usage of DNNs is closely related to person's life such as the autonomous cars. Some existing defense techniques do show the effectiveness against the adversarial attack, but since these techniques take a huge time cost during training, they are not applicable to models used in the real case which typically rely on structure of high complexity and large data sets. Therefore, finding an efficient defense technique is very import, especially to the real case usage.

1.7 Outline of Thesis

The rest of the thesis is organized as follows. We introduce the necessary background of some defense and attack method closely related to the work in this thesis in Chapter 2. The main work of this thesis, which is called e2SAD, is formally motivated and introduced in Chapter 3. Finally, experimental results are shown in Chapter 4, including the comparison with different defense models under both white-box and black box attack scenarios and some illustrations giving intuitive sense about this thesis work.

2. DETAILED BACKGROUND OF RELATED WORKS

In this chapter, some existing defense and attack methods closely related to my research would be introduced and part of them would be used as comparison with the new idea of this research.

Before going to the details, here is the recap of the general definition of adversarial attacks:

Attack Models and Algorithms. The goal of all attack models is to find a perturbation δ to be added to a clean input $x \in R^d$, resulting in an adversarial example $x_{adv} = x + \delta$ which may potentially lead to misclassification of the classifier. Typically, the noise level of the perturbation is constrained by the ℓ_∞ ball denoted by ε to make sure that the perturbation is sufficiently small. Based on the amount of information the attacker knows, there are two threat levels as follows:

1. White box: the attacker has full information about the model including its architecture and parameters such that it is possible craft adversarial examples using techniques such as gradient based attacks to specifically target the model;

2. Black box: the attacker has no knowledge about the architecture and parameters of the model. Neither is the attacker able to query the model. Adversarial examples can be generated using a substitute model which is a white-box to the attacker.

2.1 White Box Attacks

2.1.1 The Fast Gradient Sign Method (FGSM)

The cost function of a neural network is a good measure for the model convergence and a great indicator of the model accuracy. Higher loss value usually corresponds to poor performance of the classification job. The main goal of the adversary is to find the constrained perturbation which can maximize the loss value, and the local gradient information of the model with respect to the normal input is leveraged. Assume $J(\theta, x, y)$ is the cost function used to train the network, where θ is the networks's parameters, x is the input and y is the corresponding true label, the crafting process of the adversarial example is like the following:

$$x^{adv} = x + \varepsilon \cdot sign(\nabla_x J(\theta, x, y)) \tag{2.1}$$

where $\varepsilon \in [0, 1]$ is a constant value used to constrain the noise level of the perturbation. This attack method can work pretty well under the condition that the model is highly linear, which means the local gradient information can reflect the information in the larger scope.

2.1.2 Iterative Fast Gradient Sign Method (IFGSM) and PGD

This method is the enhanced version of FGSM. FGSM attack may not work very well when the model is not that linear since it just has the local gradient information. However, within the same noise level, IFGSM attack applies multiple steps of FGSM attack so that it can search the weakest point more precisely. Assume IFGSM attack generates adversarial examples by iteratively applying FGSM attack multiple, say N, times to a clean input x with a small constant

$$x_k^{adv} = clamp_{x,\varepsilon}\left(x_{k-1}^{adv} + a \cdot sign(\nabla_{x_{k-1}^{adv}} J(\theta, x_{k-1}^{adv}, y))\right),\ x_0^{adv} = x,\ k = [1, \cdots, N] \tag{2.2}$$

In the implementation of this research, a is set as $\frac{\varepsilon}{N}$. Typically, each component of the input vector, e.g. a pixel, is normalized to be within [0, 1]. The function $clamp_{x,\varepsilon}$ is an elementwise clipping function which clips each element x_i of input x into the range of $[max(0, x_i - \varepsilon), min(1, x_i + \varepsilon)]$.

Projected gradient descent (PGD) is a closely related variant of IFGSM. Typically, PGD first randomly picks a point within a confined small ball around each clean input and then applies the multi-step IFGSM to generate adversarial examples for that clean input. The idea behind PGD somehow corresponds the concept of data augmentation.

2.1.3 Jacobian-based Saliency Map Algorithm (JSMA)

Another perspective is to check the output of the model, then directly computing the sensitivity of the output of the model with respect to the input to determine the direction and the magnitude of the perturbation. JSMA tries to perturb the legitimate sample with a limited amount of modified pixels. For a specific legitimate input, a greedy selection is executed through multiple iterations to determine a set of pixels (m^*, n^*)which can lead to the misclassification most likely by adding perturbation to them. These chosen pixels will be set to either maximum or minimum of the value range according to the sign of sensitivity. This process is repeated until either (a) the desired output is obtained, or (b) the number of the modified pixels exceeds the threshold (The threshold is set as the distinguishable amount of modified pixels).

In order to pick up the optimal pair of pixels to modify in each searching iteration, JSMA uses the sensitivity of the output with respect to each input neuron to measure the importance of each input neuron. The following two values are computed.

$$\alpha_{mn} = \sum_{p \in \{m,n\}} \frac{\partial Z(X)_t}{\partial x_p} \tag{2.3}$$

$$\beta_{mn} = \left(\sum_{p \in \{m,n\}} \sum_{j=0}^{N} \frac{\partial Z(X)_j}{\partial x_p} \right) - \alpha_{mn} \tag{2.4}$$

where α_{mn} represents how likely the modification of pixel pairs (m, n) will make the output misclassified as class t, and β_{mn} represents how likely the modification of a pixel pair (m, n) will alter all the other outputs. Note that this attack uses the output of the final fully connected layer Z instead of the output of the softmax layer F to avoid some gradient problems. The algorithm picks the best pixel pairs using the following criteria.

$$(m^*, n^*) = \arg\max_{(m,n)} (-\alpha_{mn} \cdot \beta_{mn}) \cdot I(\alpha_{mn} > 0) \cdot I(\beta_{mn} < 0) \tag{2.5}$$

2.2 Representative Existing Defense Methods

Discussion in the first chapter has summarized that current adversarial defense methods can be separated along three main directions: expanding the training data to make the classifier more robustly learn the underlying function; (2)modifying the training procedure to reduce the gradients of the model w.r.t. the input such that the classifier becomes more robust to input perturbations; (3)using external models as network add-ons when classifying unseen examples. In this section, we briefly introduce one method for each direction.

2.2.1 Adversarial Training

This is a popular defense approach which augments the training dataset with adversarial examples ([33]; [36]). By taking the adversarial examples which indicate the vulnerability of the model under a certain attack method into consideration during the training process, the resulting model can defense the adversarial examples generated by that certain attack method effectively. In our implementation, we adopt the adversarial training equation proposed in [33] as the loss function

$$J_{adv(\theta,x,y)} = \alpha \cdot J(\theta, x, y) + (1-\alpha) \cdot J(\theta, x^{adv}, y), \tag{2.6}$$

where α is a constant specifying the relative importance of the adversarial examples. In our latter comparison, we choose two methods, FGSM and IFGSM, for generating the adversarial examples.

There exist defense methods ([43]; [42]) which view the process of training a robust model as solving a minmax optimization problem

$$\arg\min_{\theta} E_{(x,y)\sim\mathcal{D}} \left[\max_{\delta} J(\theta, x+\delta, y) \right], \tag{2.7}$$

where $\mathcal{D}$ is the underlying training data distribution, $J(\theta, x, y)$ is the loss function, and θ is the parameters of the model. In [43] and [42], the maximization with respect to δ is approximated by a specific attack method, for example, by PGD, which is suggested as the strongest first-order adversarial attack by [42].

Hamm [44] proposes a new approach which can instead of targeting the saddle points like previous methods (previous methods typically approximate the inner max optimization with a certain attack methods and solve the minmax problem iteratively), find the true optimal solution of the minmax optimization problem. [44] chooses FGSM to approximate the inner maximization step and for the outer minimization step, instead of plugging the adversarial version of the clean data directly and solving the optimization problem, changes the minimization objective function to

$$\arg\min_{\theta} \left[J(\theta, x^{adv}, y) + \frac{\varepsilon N}{2} \left\| \frac{\partial J(\theta, x^{adv}, y)}{\partial x^{adv}} \right\| \right], \tag{2.8}$$

where ε is a constant restricting the level of input perturbation, and N is the number of the training examples in each minibatch. The viewpoint of [44] is that the adversarial examples are the function of model parameter θ, so they should be treated as the function of θ rather than fixed inputs in the loss function for the min step.

2.2.2 Defensive Distillation

Another perspective considers that the cause of the DNN vulnerability to adversarial perturbations is due to the overfitting of the model to the training dataset. In practice, when training a supervised classification problem, one hot vector is usually used the training label, but the use of hot vectors may lead to a trained model with a more complex function surface since it attempts to force different inputs of the same class have the same extreme one hot vector output.

The authors of [22] roposed a defense method using the distillation technique, which is implemented by extracting class probability vectors produced by a first DNN or an ensemble of DNNs to train a second DNN of a reduced dimensionality without loss of accuracy. They consider that the knowledge extracted by distillation, in the form of probability vectors, can also be beneficial to improving generalization capabilities of DNNs outside of the training dataset.

Figure2.1 shows the procedure of the distillation defense method. Instead of using the original one hot vectors as the training labels, this method uses the so called soft labels, which are the probability output vectors of pre-trained neural network model to train the final neural network model. Note that the structure of the second neural network model is the same as that of the first

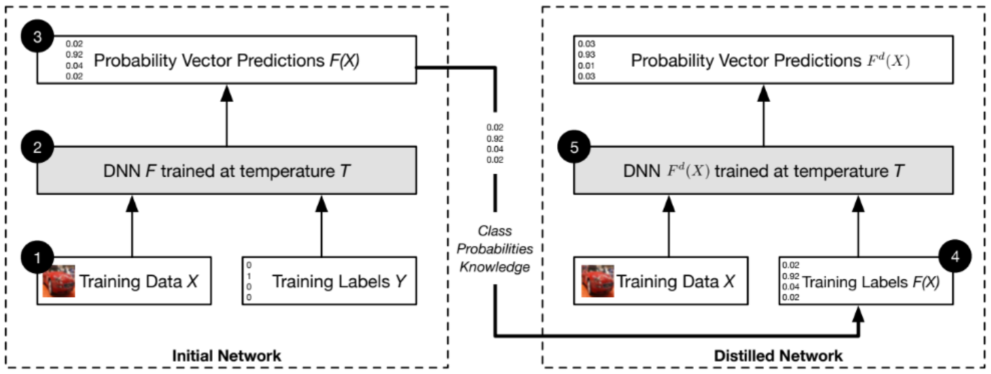


Figure 2.1: An overview of the defense mechanism based on a transfer of knowledge contained in probability vectors through distillation, reprinted from [22] ©2016 IEEE

model here. Then the goal becomes solving the following optimization problem:

$$\arg\min_{\theta} -\frac{1}{\|X\|}\sum_{x\in X}\sum_{i\in 0\ldots N} F_i(x)\log F_i^d(x), \tag{2.9}$$

where θ are the parameters of the neural network model, F(x) is the probability output vector of the input x based on the first model, and $F^d(x)$ is the output vector of the distillation model.

2.2.3 MagNet

Meng and Chen [39] proposed a framework that uses one or more external detectors to classify an input image as adversarial or clean. During the training process, this method trains a "reformer network" (which is an auto−encoder or a collection of auto−encoders) to learn the manifold of clean images. In the test phase, the network, based on the manifold learned by the reformer network, treats images which are far from the manifold as adversarial examples and reject them. By contrast, images that are close to the manifold (but not exactly on it) would be mapped back to the manifold and classified as clean inputs. When using a collection of auto−encoders, one reformer network is chosen at random at test time such that the strength of the defense is increased.

3. METHOD, EXPERIMENTAL RESULTS AND DISCUSSION

3.1 Adversarial Training

Adversarial training, which augments the training dataset with adversarial examples during the training process, has bee shown to effectively increase the robustness of the model against white box attacks when the attack method applied by the attacker is the same as the method used to generate adversarial examples during the training process. However, the attacker takes another attack strategy, which is quite different to the one used for the adversarial training, to apply the white box attack, the model resulted from adversarial training does not resist the attack very well. For instance, adversarial training using one- step FGSM can not improve the robustness of a model against the iterative version of FGSM.

Madry [42] suggest that PGD, one particular type of multi-step iterative adversarial attack, is the strongest universal first-order adversary, which implies that adversarial training using PGD can result in the model which is robust against all other first-order adversarial attacks. It is shown that the resulting model of PGD-based adversarial training is robust against not only PGD but other kinds of first-order attacks such as FGSM. However, compared to adversarial training using FGSM, adversarial training using IFGSM or PGD takes much more time for the training process since it takes multiple steps (e.g. 40) of back propagation to generate adversarial example during each training iteration.

3.2 Proposed Efficient Two-Step Adversarial Defense (e2SAD)

Motivated by the high training cost of adversarial training using iterative attack methods, the objective of this research is to develop a defense method with a cost similar to that of FGSM adversarial training while being robust to both FGSM and multiple-step attacks such as IFGSM.

A shown in Figure 3.1, the proposed efficient two-step adversarial defense (e2SAD) approach takes only two steps of back propagation to find adversarial examples.

First, for a given input the *categorical distribution* of the model is defined as the vector of

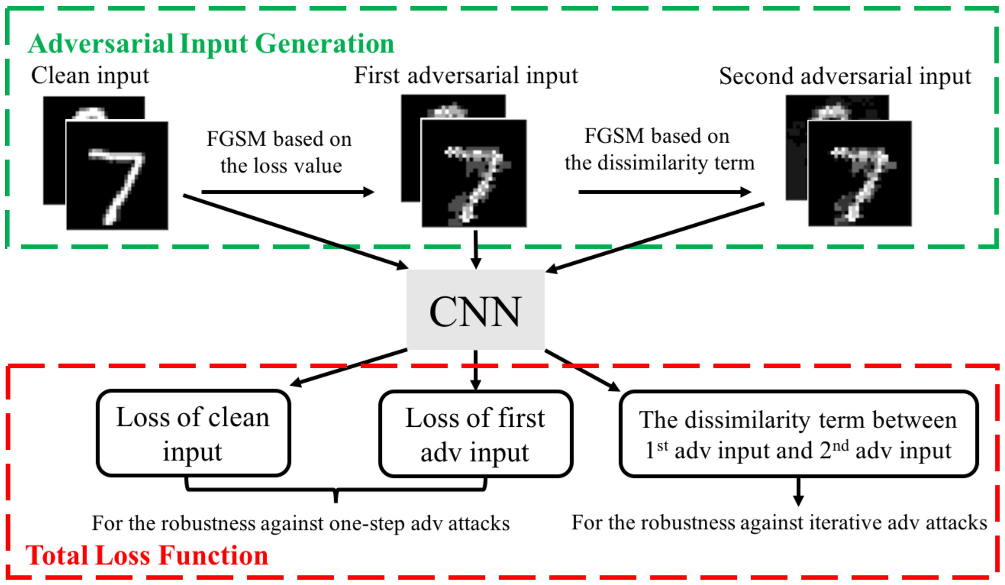

Adversarial Input Generation
Clean input
First adversarial input
Second adversarial input
FGSM based on the loss value
FGSM based on the dissimilarity term
CNN
Loss of clean input
Loss of first adv input
The dissimilarity term between 1st adv input and 2nd adv input
For the robustness against one-step adv attacks
For the robustness against iterative adv attacks
Total Loss Function

Figure 3.1: e2SAD: the proposed efficient two-step adversarial defense method.

probabilities the model outputs, where each component of the vector represents the probability for the input to be in the corresponding class. At the first step of e2SAD, a one-step attack method such as FGSM is applied to find the first adversarial example per each clean input. At the second step, within the neighborhood of the first adversarial example, the input point whose categorical distribution is most different from that of the first adversarial point is selected as the second adversarial example. For each clean input, these two generated adversarial examples are considered in the final loss function for training. The loss function consists of three terms: the loss of the original clean inputs, the loss of the adversarial examples generated at the first step, and the dissimilarity in categorical distribution of all pairs of the corresponding first and second adversarial examples. It is worth noting that the two-step e2SAD approach is structured in a particular way such that it may provide strong defense against both one-step and multiple-step attacks, as detailed below.

3.2.1 Robustness against One-Step Adversarial Attacks

The first two terms in the final loss are basically identical to the framework of the basic adversarial training where the main objective of this step is to find a highly vulnerable neighborhood around each clean training data point such that by including this step in the training process, the trained model can be made robust to one-step gradient-based attacks. In the implementation of this research, FGSM is applied as the one-step attack method to generate the corresponding adversarial version x_i^{adv} for each clean data point x_i.

$$x_i^{adv} = x_i + \varepsilon_1 \cdot sign(\nabla_{x_i} J(\theta, x_i, y)), \tag{3.1}$$

where ε_1 is a constant chosen step size. Essentially, by including the loss of this adversarial example in the final loss, the first two terms of Equation 3.3 guides the training process to reduce the loss value of both x_i and x_i^{adv}, acting as the mechanism of defending one-step adversarial attacks.

3.2.2 Robustness against Iterative Adversarial Attacks

As discussed earlier, compared to one-step adversarial attacks, iterative adversarial attacks typically have much stronger capability to attack as they search the neighborhood of a clean data point more exhaustively, which in turns makes the adversarial training using iterative adversarial attacks a stronger defense. At the second step of e2SAD, the goal is to efficiently find the second adversarial point which can give rise to stronger defenses against multi-step attack methods by using one extra step of searching computation. Therefore, the biggest is how to find a second adversarial example $\tilde{x}_i^{adv}$ which is close to x_i^{adv} and can effectively reveal the vulnerability of the model in a way similar to expensive multi-step attacks.

In an iterative attack method such as IFGSM or PGD, the adversarial example in each iterative attack step is typically found by perturbing the preceding adversarial example to maximize its loss value and the loss is usually described using the cross entropy based on either the hard or soft label. However, with the precondition that the number of searching steps is restricted, taking the loss value as the indicator for searching adversarial examples would not work. In this research, instead of constraining the loss value of the neighborhood around the clean data point, the objective is to constrain the training process in a way such that the level of the similarity in the prediction of the trained model is maintained in the neighborhood of each clean input x_i. It is important to note that *maintaining similarity of prediction* and *minimizing the loss* may be correlated but are not necessarily identical objectives; the latter attempts to ensure that predictions made in some neighborhood of the input individually have low loss without specifically constraining these predictions to be similar to each other. Therefore, it is likely to say that with the regularization on the similarity of the prediction, the training process takes not only the adversarial example but also the similarity of the prediction in the neighborhood of the clean data point into consideration and in turns leads to a well-regularized decision boundary around each x_i.

With the above understanding, at the second step of e2SAD, it attempt to find the second adversarial example $\tilde{x}_i^{adv}$ whose categorical distribution is maximally different from that of the first adversarial example x_i^{adv} in the neighborhood of x_i^{adv}. The level of the similarity in the prediction of the vulnerable region indicated by the first adversarial example is approximated by the dissimilarity in the categorical distributions of these two points. This dissimilarity term is measured by cross entropy (CE). To locate $\tilde{x}_i^{adv}$, FGSM is used as a one-step optimization method to maximize the CE-based dissimilarity measure

$$\tilde{x}_i^{adv} = x_i^{adv} + \varepsilon_2 \cdot sign\left(\nabla_{\tilde{x}_i^{adv}} CE(f(x_i^{adv}, \theta), f(\tilde{x}_i^{adv}, \theta))|_{\tilde{x}_i^{adv} = x_i^{adv}}\right), \tag{3.2}$$

where ε_2 is the step size, and the gradient of the CE-based dissimilarity is evaluated at x_i^{adv}.

The reason for using categorical distribution as the measure of dissimilarity to find the second adversarial point is as follows. First, note that the value of loss for a model prediction does not fully indicate whether the prediction is a misclassification or not. To see this, consider a simple classification task with three classes. Assume that the true class labels for two different inputs are both the first class, and the corresponding categorical distributions are $[0.45, 0.55, 0]$ and $[0.4, 0.3, 0.3]$, respectively. Knowing that one-hot encoding is conventionally used for the labels, in this case, the model misclassifies the first input while correctly classifies the second, but it can be found out that the loss of the first input is in fact lower than that of the second input.

Figure 3.2 shows how the choice of the optimization objective may influence the generation of the second adversarial example for an illustrative three-class classification problem. The prob-

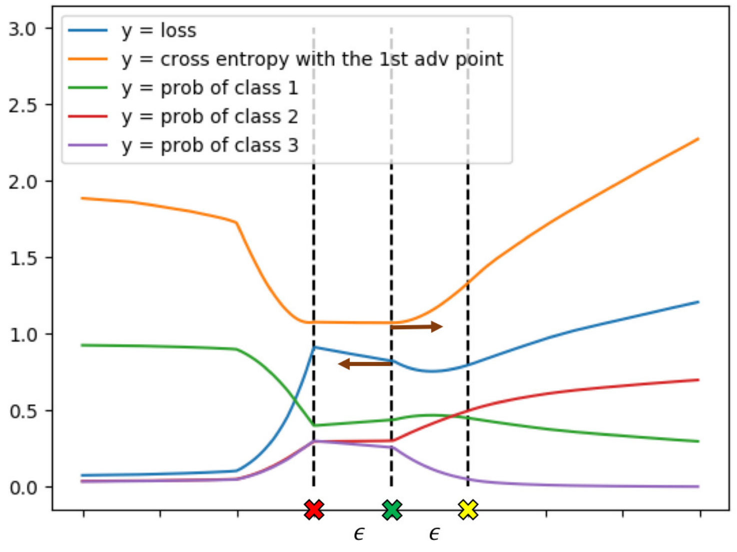


Figure 3.2: Using the loss and cross-entropy dissimilarity at the second step of e2SAD for a three-class classification problem. Both the clean input and first adversarial example belong to class 1. Green cross: the 1st adversarial point; Red cross: the 2nd adversarial point found by applying FGSM on the loss value; Yellow cross: the 2nd adversarial point found by applying FGSM on the cross entropy dissimilarity with the 1st adversarial point. Only the yellow cross corresponds to a misclassification.

abilities of three classes predicted by a trained model for a set of inputs are illustrated using the green, red, and purple curves, respectively. Accordingly, the one-hot encoding loss as a function of the input is shown by the blue curve. The cross entropy of categorical distribution between each input and the first adversarial example (green cross) is shown by the orange curve. Note that both the clean input and first adversarial example belong to class 1 in this setup. Starting from the green cross, maximizing the loss using FGSM produces the red cross as the second adversarial example. In comparison, using the CE dissimilarity measure as the objective function leads to the yellow cross. While having the highest loss, the red cross is correctly classified by the model. On the other hand, misclassification happens at the yellow cross which is found based on the CE dissimilarity measure, suggesting its effectiveness in finding stronger adversarial points.

3.2.3 The Final Loss Function

Based the two adversarial examples generated at the two steps of e2SAD, we design the loss function used for training the final model as follows. For a mini batch X of m clean examples $\{x_1, \cdots, x_m\}$ and the corresponding mini batch of the first set of adversarial examples $X^{adv} = \{x_1^{adv}, \cdots, x_m^{adv}\}$ generated at the first step of e2SAD, the total loss function is given by

$$\arg\min_{\theta} \left[\alpha \cdot J(\theta, X, Y), + (1-\alpha) \cdot J(\theta, X^{adv}, Y) + \lambda \frac{1}{m} \sum_{i=1}^{m} D(f(x_i^{adv}, \theta), f(\tilde{x}_i^{adv}, \theta)) \right], \quad (3.3)$$

where θ is the parameters of the model, each $\tilde{x}_i^{adv}$ is the second adversarial example which has the maximally different categorical distribution from the corresponding first adversarial example x_i^{adv}, $f(x, \theta)$ indicates the categorical distribution output function of the model for input x, and D is the cross-entropy dissimilarity measure.

3.2.4 The Training Process

We adopt label smoothing [35] for the training process. Here, instead of using hard labels (one-hot labels) for each cross-entropy loss, we employ the so-called soft labels which assign the correct class a target probability of δ and divide the remaining $(1 - \delta)$ probability mass uniformly among the incorrect classes. We have found that the use of label smoothing in e2SAD leads to better performance.

The overall training algorithm of the proposed e2SAD approach is summarized in Algorithm 1. The hyperparameters α and λ specify the weights for the losses of the clean and first set of adversarial inputs and those for the dissimilarity between each pair of the first and second adversarial inputs, respectively. ε_1 and ε_2 decide the scope of the neighborhood which is taken into consideration for the robustness for each clean data point. While the first two terms in the final loss function target the defense against one-step adversarial attacks, the last term mainly plays the role of defending multi-step attacks. In practice, α and λ shall be properly chosen to balance between these two different defense needs.

Algorithm 1 The proposed two-step adversarial defense: e2SAD

```
Input: training dataset $(X, Y)$; Initial model parameter $\theta$; batch size: $m$; hyperparameters
    $\alpha, \lambda, \varepsilon_1, \varepsilon_2$
Output: Trained model parameter $\theta$
1: for each minibatch t do
2:     for each $(x_i, y_i)$ in the current batch do
3:         $x_i^{adv} \leftarrow x_i + \varepsilon_1 \cdot sign(\nabla_{x_i} CE(y_i, f(x_i, \theta)))$
4:         $\tilde{x_i}^{adv} \leftarrow x_i^{adv} + \varepsilon_2 \cdot sign\left(\nabla_{\tilde{x_i}^{adv}} CE(f(x_i^{adv}, \theta), f(\tilde{x_i}^{adv}, \theta))|_{\tilde{x_i}^{adv} = x_i^{adv}}\right)$
5:     $L_{adv} = \alpha \cdot J(\theta, X_t, Y_t) + (1 - \alpha) \cdot J(\theta, X_t^{adv}, Y_t)$
6:     $L_d = \frac{1}{m} \lambda \sum_{i=1}^{m} D(f(x_i^{adv}, \theta), f(\tilde{x_i}^{adv}, \theta))$
7:     $L_{total} = L_{adv} + L_d$
8:     Update $\theta$ using backpropogation based on $L_{total}$
9: Return $\theta$
```

3.3 Experiment Results

We compare the proposed e2SAD method with two widely adopted techniques in the literature: adversarial training using single-step FGSM [33] and the adversarial training using multi-step IFGSM. We also report our experience on the minimax adversarial defense method proposed in [44]. We adopt the widely used the MNIST handwritten digits dataset [5] and the Street View House Numbers (SVHN) Dataset [45] as benchmarks.

3.3.1 Results on the MNIST Handwritten Digits Dataset

MNIST consists 60,000 training images and 10,000 testing images, where each pixel value is normalized to be within $[0, 1]$. The adversarial attacks considered are:

- White-box attacks with FGSM under different noise levels: $\varepsilon = 0.3, 0.4$.
- White-box attacks with IFGSM under the fixed noise level of $\varepsilon = 0.3$ with different numbers of steps: $k = 10, 30$.
- Black-box attacks from three substitute models: the naturally trained model (i.e. the one trained using only the clean inputs without any additional defense strategy), one trained with FGSM adversaries under the noise level of $\varepsilon = 0.3$, and one trained with IFGSM adversaries

under the total noise level of $\varepsilon = 0.3$ and step size of 0.01 $(k = 30)$. With respect to these substitute models, IFGSM with $\varepsilon = 0.3$ and $k = 30$ is used to generate adversarial examples, which are then employed to attack each of the targeted models.

All CNNs we use consist of two convolutional layers with 32 and 64 filters respectively, each of which is followed by a 2×2 max-pooling layer and ReLU activation function, and a fully connected layer of $1,024$ neurons. The configuration of the CNNs is summarized in Table A.1 in the Appendix. The results are as follows:

1. **Results on White-Box Attacks** For our proposed e2SAD method, we set the hyperparameters in the training Algorithm 1 as: $\alpha = 0.2$, $\lambda = 0.1$, $\varepsilon_1 = 0.3$, and $\varepsilon_2 = 0.1$. To increase the searching ability of the second step of e2SAD, we do not clamp the second adversarial point to be within a norm ball around the clean data point. All models are trained on MNIST for 30,000 iterations with the batch size of 256.

   (a) **Comparison with adversarial training** The performances of different models under various white-box attacks are shown in Table 3.1. It can be seen that each model reaches the accuracy of over 99% on the clean dataset. The baseline model trained naturally shows no defense ability towards both FGSM and IFGSM adversaries while other three models demonstrate different levels of defense. The model obtained via FGSM adversarial training maintains a very high accuracy under FGSM attacks with different noise levels. However, FGSM adversarial training can only defend FGSM attacks while shows no defense ability against IFGSM attacks of any step number. IFGSM adversarial training performs well under IFGSM adversaries and also shows robustness against FGSM attacks. However, the defense ability drops fast when the noise level ε increases in the case of FGSM attacks. Specifically, the accuracy can drop by almost 14% under the FGSM attacks when the noise level increases to $\varepsilon = 0.4$. Note that it makes 30 steps to generate IFGSM adversarial examples in each training iteration, leading to the high cost of the considered IFGSM adversarial training.

Table 3.1: Accuracy of different models under white-box adversarial attacks evaluated using MNIST. Rows are the attacks where “Clean Data“ in the first row means no attack. Columns report the accuracy of different defense solutions where ”Natural” means the baseline model without any additional defense strategy, and “FGSM Adv. Train" and “IFGSM Adv. Train" mean the adversarial training with FGSM and IFGSM, respectively. “H”(“S”) stands for use of one-hot hard (soft) labels when training the defense model.

Attack		Label	Natural	FGSM Adv. Train		IFGSM Adv. Train	e2SAD
				$\varepsilon = 0.3$	$\varepsilon = 0.4$	$\varepsilon = 0.3, k = 30$	
Clean Data		H	0.9942	0.9938	0.9921	0.9913	0.9932
		S	0.9952	0.9943	0.9941	0.9913	
FGSM	$\varepsilon = 0.3$	H	0.1741	0.9768	0.9658	0.9519	0.9641
		S	0.4256	0.9846	0.9919	0.9595	
	$\varepsilon = 0.4$	H	0.1027	0.9136	0.9732	0.8152	0.9499
		S	0.2146	0.9374	0.9914	0.9012	
IFGSM	k=10	H	0.0001	0.0856	0.2108	0.9336	0.8687
		S	0.1019	0.1166	0.0101	0.9422	
	k=30	H	0	0.082	0.1968	0.9325	0.8633
		S	0.093	0.0966	0.0065	0.9412	

Among all models considered, the proposed e2SAD method produces the highest accuracy for both the clean data and FGSM attacks at different noise levels. Under IFGSM attacks e2SAD significantly outperforms the FGSM adversarial training, demonstrating the effectiveness of the proposed two-step approach’s generalization capability with respect to defense against strong multi-step attacks. Compared with the adversarial training using IFGSM, e2SAD offers stronger defense against FGSM attacks while maintaining a good robustness against IFGSM attacks. Note that these are achieved using only two steps of gradient calculation in each training iteration, presenting a significant reduction of computational cost compared with the IFGSM adversarial training, which performs 30 steps of gradient computation.

Label smoothing is adopted in e2SAD and it is shown to be effective in helping the trained model generalize well. In our experiments, we set the probability for the correct label to 0.75 and the one for all other incorrect labels to 0.25. Table 1 shows that label smoothing also improves the performance of the traditional adversarial training under some circumstances, but not significantly.

(b) **Comparison with the minimax adversarial defense** We also implemented the minimax adversarial defense method proposed in Hamm [44] with a minor modification that the model is trained using a mixture of clean and adversarial examples to achieve better performance. Our results show that the trained model is very robust against FGSM attacks, however, shows no defense against IFGSM attacks.

2. **Results on Black-Box Attacks** In Table 3.2, we consider how adversarial examples generated by applying IFGSM to a substitute model may attack a different model. The rows of the table are the considered substitute models: “Natural model" is again the baseline model without any additional defense strategy; “FGSM $\varepsilon = 0.3$” is the model obtained via FGSM adversarial training with the setting $\varepsilon = 0.3$; “IFGSM $\varepsilon = 0.3, k = 30$" is the model obtained via IFGSM adversarial training with the setting $\varepsilon = 0.3, k = 30$. The substitution models are trained using hard labels (“H") and label smoothing (“S"), then *attacked by IFGSM* with the setting ($\varepsilon = 0.3, k = 30$) for generating adversarial examples. The adversarial examples generated from the substitute models are used to attack the four models shown in the columns of the table: “Natural" is the baseline model; “FGSM Adv. Train" and “IFGSM Adv. Train" are models trained by the FGSM and IFGSM adversarial training using the settings specified in the table, respectively; “e2SAD" is the proposed model. The models under attack are trained using both hard and label smoothing except for e2SAD which is based on label smoothing only. Note that in Table 3.2, white-box attacks are resulted when the substitute model and the one under attack are identical, and all other combinations correspond to black-box attacks.

Table 3.2 demonstrates that the proposed e2SAD approach delivers a well-balanced defense against black-box IFGSM attacks from all three substitute models with an accuracy of nearly 90% or higher. There are several cases under which the natural training (baseline)

Table 3.2: Accuracy of different models under black-box IFGSM attacks evaluated using MNIST.

Substitude Model	Label	Natural	FGSM Adv. Train		IFGSM Adv.Train	e2SAD
			$\varepsilon = 0.3$	$\varepsilon = 0.4$	$\varepsilon = 0.3, k = 30$	
Natural Model (H)	H	0	0.9083	0.9158	0.9668	0.868
	S	0.1376	0.7887	0.8963	0.9641	
FGSM (H) $\varepsilon = 0.3$	H	0.9127	0.082	0.8945	0.967	0.9422
	S	0.9083	0.7581	0.8214	0.9671	
IFGSM (H) $\varepsilon = 0.3, k = 30$	H	0.9163	0.9319	0.9148	0.9324	0.8886
	S	0.885	0.768	0.8482	0.9574	
Natural Model (S)	H	0.9024	0.9742	0.9674	0.9838	0.9719
	S	0.0929	0.8485	0.8963	0.9847	
FGSM (S) $\varepsilon = 0.3$	H	0.9777	0.9708	0.9543	0.9825	0.9698
	S	0.9745	0.0966	0.7658	0.9832	
IFGSM (S) $\varepsilon = 0.3, k = 30$	H	0.939	0.952	0.9464	0.9578	0.9525
	S	0.9435	0.9476	0.9481	0.9412	

and FGSM adversarial training have a poor performance. In all cases, e2SAD either noticeably outperforms both the natural training and FGSM adversarial training or produce a fairly close performance. Compared with the models trained with the 30-steps IFGSM adversarial training, e2SAD is still very competitive particularly given the fact that only two-steps of gradient computation are performed at each training iteration.

3.3.2 Results on the Street View House Numbers (SVHN) Dataset

The Street View House Numbers (SVHN) dataset [45] consists of a training set of 73,257 digits and a testing set of 26,032 digits obtained from house numbers in Google Street View images, representing a significantly harder real-world dataset compared to MNIST. We process the SVHN dataset by removing the mean and normalizing the pixel values with the standard deviation of all pixels in each image so that the normalized pixel values are within [-1, 1].

We train three different models with the CNN configuration summarized in Table A.2 in the Appendix and compare their performances under the scenario of white box attacks. All models are trained for 20 epochs with the following setup:

- FGSM-based adversarial training: {Batch size = 256, optimizer=AdamOptimizer with learning rate 0.001, $\alpha = 0.6$, $\varepsilon = 24/255$}
- IFGSM-based adversarial training: {Batch size = 256, optimizer=AdamOptimizer with learning rate 0.001, $\alpha = 0.6$, $\varepsilon = 24/255$, attack steps=10}
- e2SAD: {Batch size = 256, optimizer=AdamOptimizer with learning rate 0.001, $\alpha = 0.6$, $\lambda = 0.3$, $\varepsilon_1 = 24/255$, $\varepsilon_2 = 8/255$, label smoothing with correct class probability of 0.75}

The performances of the various models on this much harder SVHN dataset are summarized in Table 3.3. It turns out that e2SAD outperforms all other models in this case. More specifically, the baseline (natural) model shows no defense to any attack. e2SAD attains a significantly stronger robustness against the iterative IFGSM white-box attacks compared with the FGSM adversarial training, which shows no defense to such attacks. Furthermore, compared with the expensive IFGSM adversarial training, e2SAD offers a much stronger defense against the one-step FGSM attacks. This fact may be attributed to the particular two-step structure of e2SAD, which is geared towards defending both one-step and multi-step adversarial attacks.

Table 3.3: Test accuracy of different models under white-box adversarial attacks evaluated using SVHN. The e2SAD models are trained with soft labels while all other models are trained with hard labels. The definitions of all variables are identical to those used in Table 1.

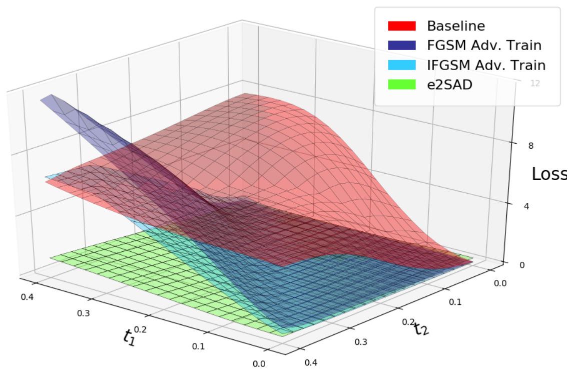
Attack		Natural	FGSM Adv. Train $\varepsilon = 24/255$	IFGSM Adv. Train $\varepsilon = 24/255, k = 10$	e2SAD
Clean data		0.9006	0.9119	0.9001	0.9236
FGSM	$\varepsilon = 24/255$	0.1962	0.7842	0.4289	0.7881
IFGSM	$\varepsilon = 24/255$, k=10	0.0628	0.0455	0.3228	0.3328
	$\varepsilon = 24/255$, k=20	0.0602	0.0402	0.3165	0.4020
	$\varepsilon = 24/255$, k=30	0.0593	0.0385	0.3146	0.3868

3.4 Visualization of the Loss Surfaces of Different Models for MNIST

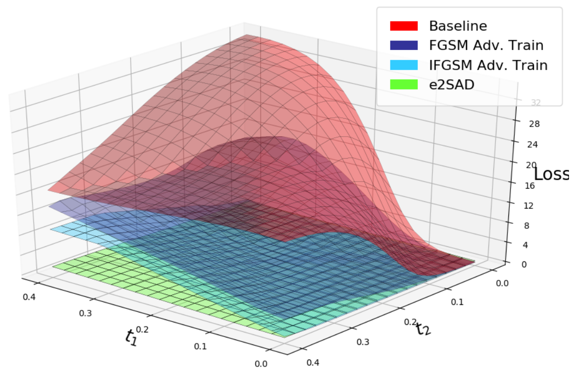
In this section, the visualization of the loss surfaces of different models is demonstrated in order to compare their potential capabilities of defensing against one-step FGSM attacks and iterative FGSM attacks. Here, four different models are chosen for the demonstration in Figure3.3a and Figure3.3b: the baseline model which is trained with the clean data and without any defense techniques; "FGSM Adv. Train" which is trained by adversarial training using FGSM with the hyperparameter setup ($\alpha = 0.5, \varepsilon = 0.3$); "IFGSM Adv. Train" which is trained by adversarial training using IFGSM with the hyperparameter setup ($\alpha = 0.5, \varepsilon = 0.3, k = 30$); And e2SAD is the proposed model trained with the hyperparameter setup ($\alpha = 0.2, \lambda = 0.1, \varepsilon_1 = 0.3, \varepsilon_2 = 0.1$). All models are trained using a total number of 30,000 mini-batches of 256 images each over the MNIST dataset.

Figure3.3a and Figure3.3b illustrate the loss surface of each model in the input space under FGSM attack and IFGSM attack respectively. To make the visualization possible in a reduced 2-dimensional input space, for each clean data x_i, a search direction g_1 is defined as $sign(x_i^{adv} - x_i)$, where x_i^{adv} is the corresponding adversarial examples of x_i found by the adversarial attack method. Another direction g_2 is defined as a direction orthogonal to g_1. Then around each x_i, a set of perturbed images is generated along directions g_1 and g_2: $X_p = \{x_i + t_1 \cdot g_1 + t_2 \cdot g_2\}, t_1, t_2 \in [0, 0.4]$. t_1 and t_2 are again chosen to be the two lateral axes. Here the loss function is defined as the cross entropy loss based on the hard labels. Consider a minibatch of 128 clean data inputs, the mesh loss surface shows the loss value of a model summed over the perturbed inputs for the entire minibatch as a function of t_1 and t_2. For examples, the value of the loss surface at (0,0) is the loss summed over the clean inputs of the minibatch.

Under FGSM attack in Figure3.3a and IFGSM attack in Figure3.3b, it can be observed that compared to other three models, the loss surface of the model trained by e2SAD of the flattest one with the lowest average value withing this 2-dimensional searching space. This find out corresponds to the empirically observed robustness of the e2SAD model against both white-box FGSM and IFGSM attacks.



(a)



(b)

Figure 3.3: Loss surfaces in a two-dimensional input space of different models: (a) surfaces seen by the FGSM attacks, (b) surfaces seen by the IFGSM attacks.

3.5 Summary

In this chapter, we first briefly introduce adversarial training and discuss the advantage and disadvantage of the currently strongest adversarial training, the one using projected gradient descent (PGD). In order to deal with the problem of time cost for the adversarial training using iterative attack methods, we propose e2SAD, which can efficiently reveal the vulnerability of the model with the searching computation of just two steps. We then explain the high level idea behind e2SAD and the corresponding function of the loss term from each searching step. In the experimental part, we empirically show that e2SAD can defend against both FGSM and IFGSM with different settings very effectively under the white box scenario. As for the black box attacks, the model trained with e2SAD also achieves relatively stable robustness against IFGSM attacks from different model substitutes. In the last, we visualize the loss surfaces of different models under both FGSM and IFGSM white box attacks and shows that the illustrations are consistent to the empirically observed results in the experiment.

4. CONCLUSIONS

It is found that based on the adversarial point generated by an one-step adversarial attack, by finding a point with the most different categorical distribution to that of the first adversarial point and combining these two points into the final loss function, it is likely to reveal the vulnerability of of the model around each clean data point more precisely. We exploited this and proposed a time-efficient training method: e2SAD, , which trains the network with an additional dissimilarity term which is defined by the dissimilarity in the prediction of the first and second adversarial points. We have demonstrated the effectiveness of e2SAD in terms of defense against while-box one-step FGSM and multi-step IFGSM attacks and black-box IFGSM attacks under various settings.

e2SAD provides a general mechanism for defending both one-step and multiple attacks and for balancing between these two defense needs, the latter of which can be achieved by properly tuning the corresponding weight hyperparameters in the training loss function. In the future work, we will explore hyperparameter tuning, different metrics used for measuring the dissimilarity and other new techniques to provide a more balanced or further improved defense quality for a wider range of white and black box attacks.

REFERENCES

[1] Ujjwal Karn, “A single neuron.” https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/, 2016. [Online; accessed October 10, 2018].

[2] Ujjwal Karn, “Different activation functions.” https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/, 2016. [Online; accessed October 10, 2018].

[3] Ujjwal Karn, “The structure of feedforward network.” https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/, 2016. [Online; accessed October 10, 2018].

[4] Fei-Fei Li, Justin Johnson, Serena Yeung, “Backpropagation of a single node.” http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture4.pdf, 2017. [Online; accessed October 10, 2018].

[5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[6] Ujjwal Karn, “A simple cnn structure.” https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/, 2016. [Online; accessed October 10, 2018].

[7] Adam Geitgey, “Pixel values of an image.” https://reurl.cc/NXkyk, 2016. [Online; accessed October 10, 2018].

[8] Yeh James, “One step of the 2d convolutional computation.” https://goo.gl/MDJbKt, 2017. [Online; accessed October 10, 2018].

[9] Yeh James, “The result of the 2d convolutional computation.” https://goo.gl/MDJbKt, 2017. [Online; accessed October 10, 2018].

[10] Rob Fergus (NYU), Honglak Lee (Michigan), Marc’Aurelio Ranzato (Google), Ruslan Salakhutdinov (Toronto), Graham Taylor (Guelph), Kai Yu (Baidu), “An illustration

of the image convoluation process.” https://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/, 2012. [Online; accessed October 10, 2018].

[11] Rob Fergus (NYU), Honglak Lee (Michigan), Marc’Aurelio Ranzato (Google), Ruslan Salakhutdinov (Toronto), Graham Taylor (Guelph), Kai Yu (Baidu), “An illustration of the relu activation function.” https://cs.nyu.edu/~fergus/tutorials/deep_learning_cvpr12/, 2012. [Online; accessed October 10, 2018].

[12] Yeh James, “Max pooling.” https://goo.gl/MDJbKt, 2017. [Online; accessed October 10, 2018].

[13] Christopher Olah, “Rnn unrolled structure.” http://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015. [Online; accessed October 10, 2018].

[14] Christopher Olah, “The cell structure of basic rnns.” http://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015. [Online; accessed October 10, 2018].

[15] Christopher Olah, “Missing long-term dependencies.” http://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015. [Online; accessed October 10, 2018].

[16] Christopher Olah, “The cell structure of basic lstms.” http://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015. [Online; accessed October 10, 2018].

[17] Christopher Olah, “The flow of cell states.” http://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015. [Online; accessed October 10, 2018].

[18] Christopher Olah, “The operation of forget gates.” http://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015. [Online; accessed October 10, 2018].

[19] Christopher Olah, "The operation of input gates." `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`, 2015. [Online; accessed October 10, 2018].

[20] Christopher Olah, "The update of the cell state." `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`, 2015. [Online; accessed October 10, 2018].

[21] Christopher Olah, "The output selection." `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`, 2015. [Online; accessed October 10, 2018].

[22] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *2016 IEEE Symposium on Security and Privacy (SP)*, pp. 582–597, IEEE, 2016.

[23] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pp. 372–387, IEEE, 2016.

[24] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[25] D. Hebb, *The Organization of Behavior: A Neuropsychological Theory*. Taylor & Francis, 2005.

[26] B. Farley and W. Clark, "Simulation of self-organizing systems by digital computer," *Transactions of the IRE Professional Group on Information Theory*, vol. 4, no. 4, pp. 76–84, 1954.

[27] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[28] M. Minsky and S. A. Papert, *Perceptrons: An introduction to computational geometry*. MIT press, 2017.

[29] P. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Harvard University, 1975.

[30] J. J. Hopfield, “Neural networks and physical systems with emergent collective computational abilities,” *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[31] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *nature*, vol. 323, no. 6088, p. 533, 1986.

[32] G. E. Hinton, S. Osindero, and Y.-W. Teh, “A fast learning algorithm for deep belief nets,” *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[33] I. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and harnessing adversarial examples,” in *International Conference on Learning Representations*, 2015.

[34] N. Carlini and D. Wagner, “Towards evaluating the robustness of neural networks,” *arXiv preprint arXiv:1608.04644*, 2016.

[35] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” 2014.

[36] A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial machine learning at scale,” *arXiv preprint arXiv:1611.01236*, 2016.

[37] A. S. Ross and F. Doshi-Velez, “Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients,” *arXiv preprint arXiv:1711.09404*, 2017.

[38] J. Wang, J. Sun, P. Zhang, and X. Wang, “Detecting adversarial samples for deep neural networks through mutation testing,” *arXiv preprint arXiv:1805.05010*, 2018.

[39] D. Meng and H. Chen, “Magnet: a two-pronged defense against adversarial examples,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 135–147, ACM, 2017.

[40] P. Samangouei, M. Kabkab, and R. Chellappa, “Defense-gan: Protecting classifiers against adversarial attacks using generative models,” *arXiv preprint arXiv:1805.06605*, 2018.

[41] N. Papernot, P. McDaniel, A. Sinha, and M. Wellman, "Towards the science of security and privacy in machine learning," *arXiv preprint arXiv:1611.03814*, 2016.

[42] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *arXiv preprint arXiv:1706.06083*, 2017.

[43] R. Huang, B. Xu, D. Schuurmans, and C. Szepesvári, "Learning with a strong adversary," *arXiv preprint arXiv:1511.03034*, 2015.

[44] J. Hamm, "Machine vs machine: Minimax-optimal defense against adversarial examples," 2018.

[45] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *NIPS workshop on deep learning and unsupervised feature learning*, vol. 2011, p. 5, 2011.

APPENDIX A

CNN MODEL CONFIGURATIONS

Table A.1: CNN model configuration for the MNIST dataset.

Layer Type	Details
ReLU Convolutional	32 filters (5*5, stride 1, padding 2)
Max Pooling	2*2
ReLU Convolutional	64 filters (5*5, stride 1, padding 2)
Max Pooling	2*2
ReLU Fully Connect	1024 units
Fully Connect	10 units
Softmax	10 units

Table A.2: Model configuration for the SVHN dataset.

Layer Type	Details
ReLU Convolutional	32 filters (5*5, stride 1, padding 0)
ReLU Convolutional	64 filters (5*5, stride 1, padding 0)
Max Pooling	2*2
ReLU Convolutional	128 filters (3*3, stride 1, padding 0)
Max Pooling	2*2
ReLU Fully Connect	512 units
Fully Connect	10 units
Softmax	10 units