LEARNING STOCHASTIC WEIGHT MASKING TO RESIST ADVERSARIAL ATTACKS

by

Yoshimasa Kubo

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

at

Dalhousie University
Halifax, Nova Scotia
November 2019

To my parents, who supported to pursue my dreams,

SAYOKO and KATSUTOSHI

To my brother, who always encourage me,

TOSHIHISA

To my great grand parents

Table of Contents

List of Tables

List of Figures

Abstract

Adding small perturbations to test images can drastically change the classification accuracy of machine learning models. These perturbed examples are called *adversarial examples* [108]. Studying these examples may shed light on the learned structure in the network, as well as on the potential security threat that they pose for practical machine learning applications [63]. Furthermore, since human observers can be fooled by adversarial examples [25], this study may aid in preventing the manipulation of human observers' reactions.

In this thesis, at first, we focus on gaining an understanding of the cause of adversarial examples. We argue, adding to the view of Galloway et al., that overfitting is a factor of adversarial examples, while the other researchers found the cause of adversarial examples is not related to overfitting. To make this argument, we include two directions in our study, the first is to evaluate several standard regularization techniques with adversarial attacks, and the second is to evaluate stochastic binarized neural networks on adversarial examples. We report that strong regularizations including stochastic binarized neural networks do not only improve overfitting but also help the networks in fighting against adversarial attacks.

Furthermore, we introduce a model called the *Stochastic-Gated Partially Binarized Network* (SGBN), which incorporates binarization and input-dependent stochasticity. In particular, a gate module learns the probability that individual weights in corresponding convolutional filters should be masked (turned on or off). The gate module itself consists of a shallow convolutional neural network, and its sigmoid outputs are stochastically binarized and pointwise multiplied with corresponding filters in the convolutional layer of the main network. We test and compare our model with several related approaches on both white- and black-box attacks, and to try to gain an understanding of our model, we visualize activations of some of the gating network outputs and their corresponding filters. Moreover, we apply a simple version of SGBN to a toy experiment to gain an understanding of how changeable the activations of the gate modules may be.

List of Abbreviations Used

AD Adaptive Dropout.

BIM Basic Iterative Method.

BNN Binary Neural Network.

BP Backpropagation.

CNN Convolutional Neural Network.

CNN-BIN Convolutional Neural Network with Binarized Dense Layer.

DBN Deterministic Binarized Network.

DC DropConnect.

FGSM Fast Gradient Sign Method.

FNN Feedforward Neural Network.

GM_DT Deterministic Non-binarized Activation Gate Module with the Weight Inputs.

GM_ST Stochastic Binarized Activation Gate Module with the Weight Inputs.

GMWOW_DT Deterministic Non-binarized Activation Gate Module without the Weight Inputs (GMWOW_DT).

GMWOW_ST Stochastic Bbinarized Activation Gate module without the Weight Inputs.

ILL Iterative Least-likely Class Method.

JSMA Jacobian-based Saliency Map Attack.

MLP	Multilayer Perceptron.
NN	Neural Network.
PGD	Projected Gradient Descent.
RM	Random Masking.
SAP	Stochastic Activation Pruning.
SGBN	Stochastic-Gated Partially Binarized Network.
STE	Straight-through Estimator.
WOGM	MLP without gate module.

Acknowledgements

I would like to thank my parents and brother, Sayoko Kubo, Katsutoshi Kubo, and Dr. Toshihisa Kubo for encouraging me to finish my PhD program. I would also like to acknowledge my supervisor and co-supervisor, Dr.Thomas Trappenberg and Dr.Sageev Oore who always gave me excellent suggestions to finish my PhD. Furthermore, I thank my colleague Michael Traynor. I always discussed my ideas with him, and he gave me many suggestions. Without his help, I could not publish any papers for the conferences. I would thank my friend and colleague, Junliang Luo. When I felt so stressed about my research, he always listened to me, and I relaxed a lot.

Beside my advisors, I would like to thank the rest of my thesis committee: Dr. Pawan Lingras, Dr. Malcolm Heywood, and Dr. Fernando Paulovich for their comments and encouragement.

Chapter 1

Introduction

1.1 Overview

Machine learning was defined by Arthur L. Samuel [97] as a *"field of study that gives computers the ability to learn without being explicitly programmed."* More formally, Tom M. Mitchell [72] defined machine learning as *"a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."* A machine learning model learns from training examples to improve predictions on unseen examples without hard-coded solutions.

Machine Learning algorithms tend to be divided into three categories: Supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, there are inputs and labels of the inputs. The supervised machine learning model takes the inputs and makes predictions. The errors calculated between the predictions and the labels are used to train the model. The goal of supervised learning is to learn a function that maps inputs to the desired outputs (labels). In unsupervised learning, there is input data, but no accompanying labels. An unsupervised learning model learns the patterns of the inputs or structure within the inputs. In reinforcement learning, an agent learns how to take an action to maximize the cumulative rewards in the environment.

There have been many successful attempts to apply supervised learning models, especially neural networks, to problems such as camera relocalization [55, 54, 17], object detection [91, 92], video classification [53, 112, 122, 71], language translation [106, 8], and speech recognition [45, 38]. One well-studied and advanced application of supervised learning is image processing with neural networks [59, 101, 107, 41, 42]. For instance, VGG [101] and GoogleNet [107], which represent very deep neural networks, have been applied to solve image classification problems. These models successfully improved the classification accuracies on large image datasets such as ImageNet [20].

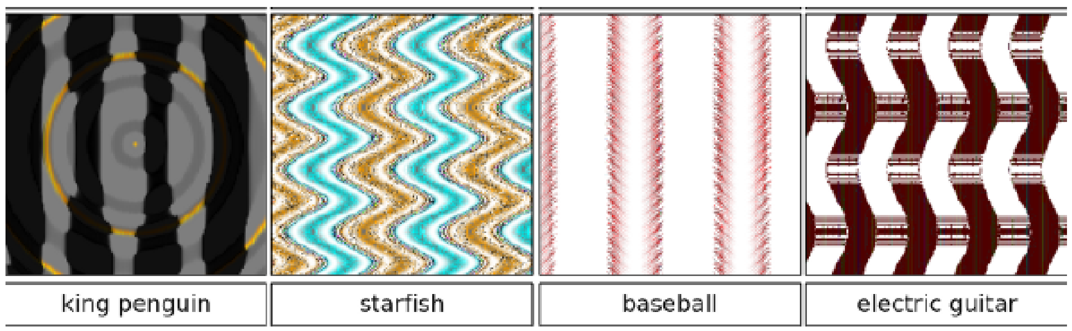


Figure 1.1: Adversarial examples crafted by evolutionary algorithms. Figure from [79]. Deep learning model classified these images as king penguin, starfish, baseball, and electric guitar.

However, these successful models for the image classification problem are limited when small perturbations are added to the images. In 2014, Szegedy et al. [108] and Biggio et al. [10] found that adding small, well-crafted perturbations to images, creating what they call *adversarial examples*, cause the images to be misclassified by machine learning classifiers. Adversarial examples are mostly imperceptible to humans, but they are significantly harmful to machine learning classifiers [121]. In addition, Nguyen et al. [79] proposed adversarial attacks that craft adversarial examples by using evolutionary algorithms, which humans cannot recognize (figure 1.1). These examples fool deep neural networks with a high confidence level. Even though well-trained deep neural networks achieve high levels of accuracy on testing datasets, it is still difficult for the models to defend against adversarial examples. Recently, researchers found that the adversarial attack is not only a problem for image classification, but also for object detection [118], voice recognition [12, 102, 123], and question answering [52]. Furthermore, this problem is not limited to machines - it can also negatively impact human behaviour. Recently, Elsayed et al. [25] showed that time-limited human observers can be fooled by adversarial examples.

The cause of this problem has been debated, and several hypotheses have been suggested. A well-known explanation for adversarial examples is the linear hypothesis [37]. According to this hypothesis, the reason for adversarial examples in neural networks, even deep neural networks, is that the models are “too linear” (a small perturbation can grow linearly with the dimension of the parameter size) including activation units that are piecewise linear such as rectifier units or sigmoidal units.

Another hypothesis is that the cause of adversarial examples is related to *overfitting*. Overfitting is defined as a model that learns a function which is too closely fit to a training dataset causing the model to possibly fail to classify unseen or test data correctly. Although some studies show that the detrimental effect of adversarial examples is *not* related to overfitting [108, 37], Galloway et al. [35] have shown recently that limiting overfitting does increase the accuracy of machine learning models presented with adversarial examples. Thus, it is not clear that overfitting is unrelated to the cause of adversarial examples.

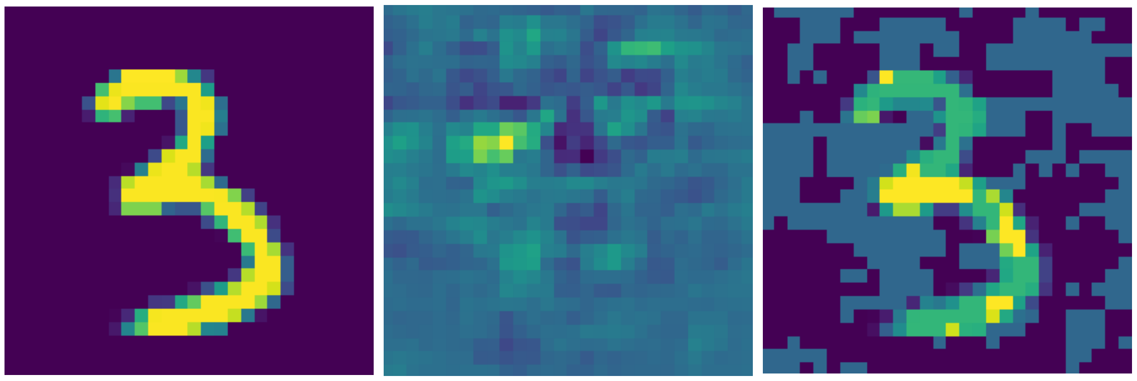
Another hint that overfitting might indeed be an important element of successful adversarial attacks is that binarized neural networks improve their accuracies in the face of adversarial examples as well. Binarized neurons are well studied as a form of regularization [44, 9, 18]. Galloway et al. [34] demonstrated the effectiveness of binary neural networks (BNN), whose weights and activations are constrained to 1 or -1, against adversarial examples. BNN improve the robustness against some adversarial examples when using the projected gradient descent (PGD) technique of Madry et al. [70].

Moreover, Dhillon et al. [21] recently showed that activation nodes can be dropped out using stochastic masking at each layer of the network during forward propagation to improve accuracy against adversarial examples. Stochastic masking of activations is another form of regularization that can help achieve some robustness against adversarial examples [114].

1.2 Focus of thesis

Adversarial attacks are categorized into two classes, *white-box attacks* and *black-box attacks*. White-box attacks [108, 37, 86, 78, 13, 77] use a model's own gradient to craft adversarial examples. This means that the attacker needs to know information about the model. On the other hand, black-box attacks [84, 85, 14, 23, 105] are attacker models that can only access the inputs and outputs of a defender model, but the attacker models do not know the architecture of the defender model.

In this dissertation, the focus of our study is on defending neural networks against both white-box and black-box attacks, and we introduce a learning model to increase the robustness of the neural networks. Previous studies [34, 21] show that binarization



(a) Original image (b) Perturbation (c) Perturbed image

Figure 1.2: An example of adversarial examples. One layer convolutional neural network classifies this original image (a) as 3 with 100% confidence. However, after adding the well-crafted perturbation (the fast gradient sign method) to the image, the network classifies the perturbed image (c) as 5 with 100% confidence.

and stochasticity help to defend the networks against adversarial examples. We combine the strength of binarization and stochasticity to our model to create an extremely robust mechanism against white-box attacks. In addition, we check our model's robustness against black-box attacks and find that our model is more resilient than the other binarized models and stochastic models. Our model is called *Stochastic-Gated Partially Binarized Network* (SGBN). SGBN model has one or more gate modules learn which weights of the main network should be turned on or off. This turning on and off of the weights depends on the input and can be considered a regularizer. We visualize the gate module activations that turn the weights of the main networks on and off in order to gain an understanding of our model. Furthermore, we discuss what the gate module learns. Besides comparing our model to the other binarized and stochastic models in standard datasets, we apply a simple version of SGBN to a toy experiment to gain a basic understanding of the mechanism. In this experiment, we compare a conventional SGBN without the weight inputs and one that has the weight inputs for the gate modules. The model with the weight inputs is the original model of SGBN. The result of this experiment shows that the performances of SGBN with and without the weight inputs are very similar, but SGBN without the weight learns the targets faster than SGBN with the weight inputs.

1.3 Overview of adversarial methods

There are many studies on adversarial attacks with different methods. Here, we discuss, several areas of methods to outline. First, we discuss different adversarial attacks and defense methods. Then, we briefly review relations with the physical world and the cause of adversarial examples.

1.3.1 Adversarial attacks

The first study showing adversarial examples is that of Szegedy et al., published in 2014 [108]. The authors generate small perturbations to images for an image classification task, and the classifiers are fooled by the perturbed images with high probability. One of the attacks in the study, L-BFGS, is found by minimizing its ℓ_2 norm distortion. L-BFGS aims to find the perturbation r that minimize:

$$c|r| + \mathrm{loss}(\mathrm{x}+\mathrm{r},\mathrm{t}) \quad \text{subject to} \quad \mathrm{x}+\mathrm{r} \in [0,1]^{\mathrm{m}}, \tag{1.1}$$

where c is a minimum constant and t is a target class. Linear search is used to find the minimum constant c where $c > 0$ until an adversary is found. This method is time-consuming and impractical. Instead of this method, subsequent research uses the fast gradient sign method (FGSM) [37] as a baseline. This method uses the sign of the gradient of the loss with respect to an input to craft a perturbation, and this perturbation is added to the input with a small step size. This method is a step away (with the small step size) from the direction of the gradient which classifies the input x as a target y. This method is straightforward to compute, and strong perturbation levels are possible. Papernot et al. [86] also implement an adversarial attack called Jacobian-based saliency map attack (JSMA). L-BFGS and FGSM that are methods to add perturbations to a whole input, but the goal of JSMA attack is to modify a few pixels in an input for a model to misclassify the input. With JSMA, it is computationally expensive to calculate a saliency map that gives an indication of which features have more effects on misclassification if perturbed [93].

The above-mentioned attacks are single gradient-step attacks. The single gradient-step attacks might not be strong enough to fool machine learning models. Instead of adding small perturbations to an image in a single step, iterative methods add perturbations to an image over a loop. These methods can craft stronger attacks than the single-step attacks. One iterative method, the basic iterative method (BIM) [63], iteratively applies FGSM multiple times with a smaller step size to an image. Another method is to iteratively apply FGSM multiple times with a smaller step size to an image, but using the least-likely class predicted by the network as a target class. This is called the iterative least-likely class method (ILL) [63]. This method iteratively adds small perturbations to an image, and the perturbed images are classified as the class which is the lowest confidence level in a prediction of a model on the clean image. All of the above methods are also used for black-box attacks. One model crafts adversarial examples using any of these methods, and these adversarial examples often fool other models [108, 111].

1.3.2 Defending against adversarial examples

There have been studies on how to defend neural networks against adversarial examples. One of the early studies on defending against adversarial examples is on adversarial training [37, 62]. Adversarial training is a method to train a model on a training dataset with adversarial examples. One of the popular adversarial training methods is the projected gradient method (PGD) introduced by Madry et al. [70]. The PGD uses an iterative method to craft adversarial examples to train a model, and the authors show that PGD is robust against both white- and black-box attacks. Tramer et al. [111] introduce another type of adversarial training called the ensemble adversarial training. The idea is to train a pre-trained model on not only adversarial examples crafted by the pre-trained model, but also adversarial examples crafted by other models, which are pre-trained separately from the main network in order to avoid overfitting to the main network's specific perturbations. This training improves the network performances on both white- and black-box attacks.

Another approach of defending methods is defensive distillation [87, 82, 83]. Distillation is originally studied for reducing a size of neural networks by transferring the knowledge from a large network to a small network [46]. Defensive distillation reduces a model's sensitivity against small adversarial perturbations, making it difficult for attackers to craft effective adversarial perturbations.

Binarized neurons improve the resilience of neural networks against adversarial examples. For example, Galloway et al. [34] show that binary neural networks (weights and activations are constrained to -1 or 1) with PGD are very robust against adversarial examples. These defensive distillation and binary neural networks are known as gradient masking models [85]. The gradient masking models do not have useful gradients to craft gradient-based adversarial attacks. For example, binary neural networks constrain weights and activations to +1 or -1 stochastically or deterministically. These binarized functions are not differentiable, and the gradients of these functions are estimated by the gradient estimator [9]. Thus, they do not provide useful gradients for crafting adversarial examples. Another example of gradient masking is applying dropout [104] during training and testing time [114]. When crafting adversarial examples, the gradient of the dropout model will be returned, but some of the nodes are stochastically dropped out.

Athalye et al. [6] define obfuscated gradients as a special case of gradient masking. There are three types of obfuscated gradients. One is shattered gradients that do not exist or are incorrect because models use defenses which are non-differentiable or cause gradients to be non-existing or incorrect such as the input transformation [40]. Another type of obfuscated gradients is stochastic gradients depending on testing time randomness [21] and randomization layers [117]. One of the stochastic gradient methods, stochastic activation pruning (SAP) [21], prunes a subset of the activations stochastically. This stochasticity improves an accuracy on adversarial examples. The third type is vanishing/exploding gradients that are often caused by defenses that consist of multiple iterations of neural network evaluation. For example, a classifier $f(g(x))$ where $g(\cdot)$ is a optimization loop to transform a input x to a new input $\hat{x}$. Differentiation through this optimization loop yields exploding or vanishing gradients. This happens in very deep computation such as Defense-GAN [96].

1.3.3 Adversarial examples in the real world

Adversarial examples can be applied to the physical world [121]. For most of the studies of adversarial examples, adversarial examples are *directly* fed into models to misclassify them. However, in the real world, we often use cameras or sensors to collect the data, as opposed to directly feeding the images to the models. Kurakin et al. [63] study whether adversarial images, obtained from a cell phone, could fool a neural network. They find that adversarial examples are misclassified by the neural network. In addition, a study by Evtimov et al. shows that perturbed traffic sign images such as a stop sign can fool neural networks [27].

1.3.4 The cause of adversarial examples

The cause of adversarial examples has been debated. Szegedy et al. [108] show that adversarial examples crafted by one model are misclassified by other models trained from scratch with different hyper-parameters, and the other models trained from scratch on a disjoint training dataset. From these observations, the authors suggest that adversarial sensitivity is not a problem of overfitting. Instead, Goodfellow et al. [37] introduce the linear hypothesis as the cause of adversarial examples. Their explanations is, even a linear model, does not overfit, misclassify adversarial examples. Instead of overfitting, the linear hypothesis is explained by the dot product of a weight vector and adversarial example:

$$w^T\tilde{x} = w^T x + w^T\gamma, \tag{1.2}$$

$$\tilde{x} = x + \gamma, \tag{1.3}$$

where γ is an adversarial perturbation, and $\tilde{x}$ is an adversarial example. This perturbation γ is increased linearly by $w^T\gamma$. When x and γ are very high dimensional, the summation of small changes of the γ will lead to a huge change of the input.

The other hypothesis should be seen when the network is poorly generalized or overfitting [109, 35, 98, 33]. Tanay et al. [109] point out that the linear hypothesis is not sufficient to explain the cause of adversarial examples. The authors explain that the ratio of the perturbation respective to the input does not change even though the dimension of the input increases. The authors introduce a new hypothesis that is related to tilting a classification boundary. When a model learns a classification boundary which lies close to a sub-manifold of a class and is tilted with respect to the manifold, adversarial examples are obtained by adding perturbations to the manifold until the perturbed data crosses the boundary. Moreover, when the classification boundary is slightly tilted, the distance between the data and the boundary is very close, and this can lead to strong adversarial examples that are almost not perceptible by humans because the data point is very close to the original data point. The authors argue that this occurs along directions of low variance in the data, and this can be considered an effect of overfitting which is mitigated by varying regularization methods. Galloway et al. [35] find that mitigating overfitting of neural networks improves the robustness on adversarial examples. Another study by Galloway et al. [33] applied two neural networks, one with adversarial training on FGSM, while the other was with L2 weight decay, to FGSM and non-FGSM attacks on a logistic regression task. The authors find that the adversarially trained network on FGSM is robust against FGSM but not the non-FGSM attack. On the other hand, the network with L2 weight decay is reasonably robust against both of the attacks. Galloway et al. show that overfitting to one type of perturbation might lead the network to be weak against other attacks, while the weight decay has a beneficial effect in defending against both of FGSM and the non-FGSM attacks. Schmidt et al. [98] find that a classification model must generalize, and the number of samples is needed to achieve the robustness against adversarial examples. Furthermore, Elsayed et al. [24] propose a regularization technique, specifically, a new form of the loss function, to enforce a large margin that is a large distance from the decision boundary. The authors discuss that benefits of the large margin classifiers are better generalization and robustness to input perturbations. This means that even if small perturbations are added to the inputs, these perturbations cannot easily flip the predicted labels of the models. Applying the proposal of Elsayed et al. of regularization to deep neural networks improves the robustness of the networks against both white- and black-box attacks compared with the standard cross-entropy loss.

1.4 Contributions of studying adversarial examples and this thesis

Adversarial examples crafted by one machine learning model often mimic other models. However, a recent study by Elsayed et al. [25] shows that time-limited human observers can be deceived by adversarial examples crafted by machine learning models. Studying adversarial examples may shed light on the learned structure in the network, as well as on the potential security threats that they pose for practical machine learning applications [63]. Since human observers can be fooled by these examples, the study of adversarial examples is also important for avoiding the manipulation of a human observer's reactions to, for example, a picture of a politician, which could be manipulated by machine learning models in an untrustworthy way [25].

In this thesis, our contribution is twofold. First, motivated by the study of Galloway et al. [35] into whether or not the cause of adversarial examples is related to overfitting, we investigate the use of regularization methods such as dropout, weight decay, and binarized neurons to mitigate overfitting for each neural network and to see how the regularization changes network performance against adversarial examples. We find that regularizations, especially strong ones, drastically improve accuracies on adversarial examples.

The second contribution we make is to introduce a new gradient masking approach that is more robust against both white- and black-box attacks compared with the other gradient masking approaches. This is based on combining two forms of regularization, binarization and stochasticity. We argue that robustness is due to two reasons: The first reason for robustness against white-box attacks is that our model's gradient masking approach creates weak perturbations that are easily classified by even an undefended model. The second reason is that the stochasticity introduced by turning the weights on and off results in the ability to ignore some perturbations on the weight level, and this helps to defend our model against black-box attacks. This specific form of turning the weights on and off is not purely random but instead learned. Note that our model does not use adversarial training to improve robustness.

1.5 Stochastic-gated partially binarized network - A general overview

We investigate a partially binarized neural network whose weights are stochastically masked depending on each input. This model's weights are masked using the gate module. A general view of this model is depicted in figure 1.3. In this figure, the model without the gate module(s) is a conventional neural network with a hidden layer(s). What the gate module does is to mask the weights of the main network depending on the input. This gate module's activations are stochastically binarized, and the activations are used to mask the weights of the main network. The gate module is

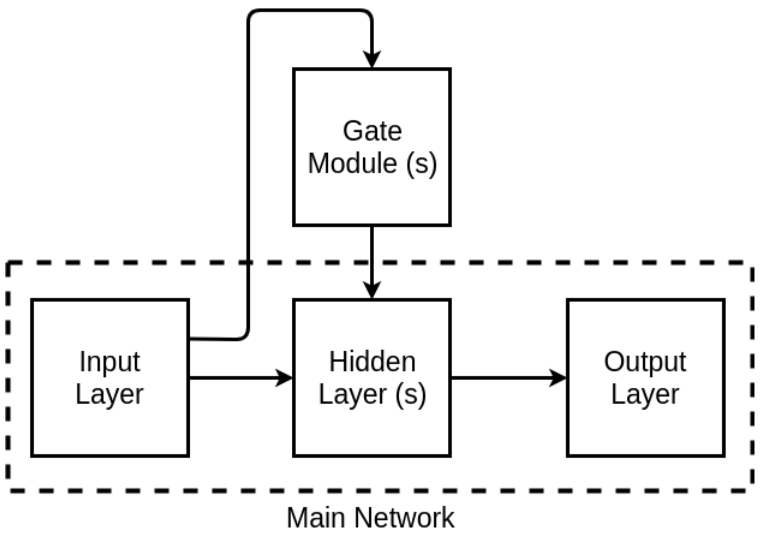


Figure 1.3: A general view of a stochastic-gated partially binarized network model.

jointly trained with the main network on a training dataset. The stochastic part of the gate module is not differentiable. For this reason, one type of gradient estimators, the straight-through estimator (STE) [9], is used for calculating the gradient. Since the stochastic part of the gate module is non-differentiable, SGBN is one type of gradient masking approaches. One of the benefits for SGBN is that it does not greatly reduce the performance on a clean testing dataset.

The difference between our model and the other binarized neural networks is the types of binarization. For instance, the weights *and* or *or* activations for binary neural networks such as [18, 19, 50, 90, 124, 110, 5] are binarized to +1 and -1. On the other hand, for our model, SGBN, the activations of the gate module are binarized to 1 to 0 to mask the main network's weights which are constrained to real values and zeros.

In our model, each input for the gate module changes the activations of the gate module. This means the weight of the main network will be changed by each of the inputs. This weight level masking or gating is analogous to activity in the human brain. Synapses on a single neuron often display widely varying neurotransmitter release probabilities [43, 94, 48, 22, 26].

1.6 Thesis outline

In this work, we aim to introduce a model of what we have applied to various types of adversarial attacks. First, we will discuss some basic concepts very briefly in three sections: Neural Networks, Binarized Neural Networks and Estimating Gradients, and Adversarial Examples and Training. In the section on binarized neural networks and estimating gradients, we explain binarized neurons with the straight-through estimations. In the section on adversarial examples and training, three adversarial attacks, two adversary distortion measurements, and adversarial training are explained. Thereafter, in Chapter 3, we applied basic regularizations to shallow networks to mitigate the overfitting. This study connects to the next chapter (Chapter 4) to propose a new model, stochastic-gated partially binarized network for adversarial examples. In this chapter, we report the performance of SGBN compared with the other gradient masking approaches on the FGSM, two iterative methods on the MNIST, and the FGSM on CIFAR-10. The black-box attack (FGSM) on MNIST are also examined. The parameter setting and environment of experiments will be explained in the chapter. Finally, we conclude with an outlook to future works in Chapter 5.

Chapter 2

Basic concepts

In this chapter, we first talk about the basic concept of supervised learning. Then neural networks, binarized neural networks with gradient estimators, adversarial examples and adversarial training, and two adversary distortion measurements will be discussed.

2.1 Supervised Learning

The goal of supervised learning is to learn a function that maps an input to a desired output. After training, a supervised learning model is used to predict the output of unseen data. There are two tasks in supervised learning. One is classification, and the other one is regression. The classification task is to learn the output of a discrete label. For the regression task, the model learns the output, which is a numerical continuous value. One method to train supervised learning models is to use gradient descent to minimize a loss function. To minimize the loss function, the partial derivative of the loss function with respect to each parameter or weight will be calculated. This derivative is used for updating each weight w as follows:

$$w_n = w_n - \alpha \frac{\partial \text{Loss}}{\partial w_n}, \tag{2.1}$$

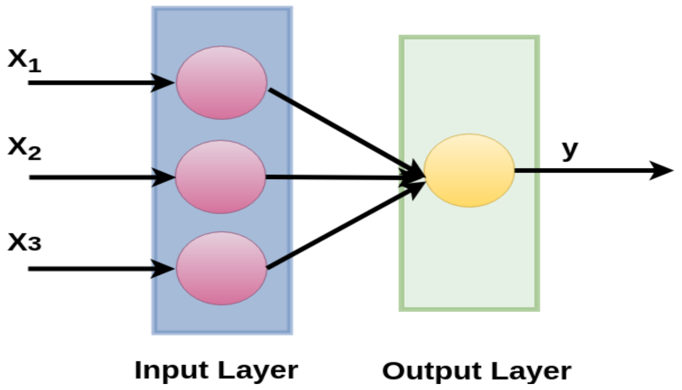
where α is a step size or learning rate, and n is an index for the weights.

2.2 Neural networks

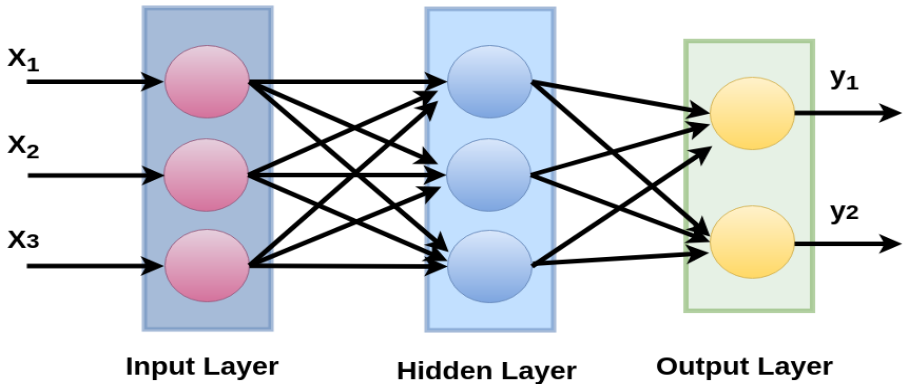
A neural network (NN) is a biologically inspired algorithm [120] used to solve many problems. There are several types of NNs, including feedforward neural networks and convolutional neural networks. We will discuss here feedforward neural networks and convolutional neural networks briefly.

2.2.1 Feedfoward neural networks

A feedforward neural network (FNN) is frequently used to solve various problems. We discuss two of the simplest models: The perceptron and the multilayer perceptron.



(a) A perceptron: An example of Feedforward Neural networks. Multiplication of inputs and weights will be the output of the network.



(b) A multilayer perceptron: An example of Feedforward Neural networks. Between the input and output layers, there is a hidden layer(s) to learn a more complicated (non-linear) function than a simple (linear) function.

The simplest FNN is a perceptron network, as pictured in figure 2.1a. In this case, the sum of the products of the weights and inputs is directly calculated to produce the output, and the information moves from the input layer to the output layer. The weights and output node model the synapse and neurons in a biological brain, and these weights establish the strength of the connection between nodes. This is described with a weight w_i and an input x_i by

$$y = \sum_i x_i w_i, \tag{2.2}$$

where y is the output of the perceptron. The perceptron can learn only a linear function ($y = ax + b$). This means the network cannot learn a non-linear function (such as $y = ax+cx^2+b$) to solve more complicated functions than the linear function.

A multilayer perceptron (MLP) (figure 2.1b) can solve this problem, that consists of at least two layers; one or more hidden layers and an output layer. Similar to a perceptron, the information in the network moves forward to the output layer through the hidden layer from the input. This hidden layer is helpful to learn the non-linear function to solve the problem which the perceptron has. The outputs in each layer are also produced by equation 2.2 and applied to an activation function which transforms the input to the non-linear output. An activation function can be sigmoid, tanh, rectified linear unit, or others.

There are several proposals for how to train the FNN. One of them is the gradient descent algorithm. To compute the gradient of the network outputs with respect to all weights in the network, there is a method called backpropagation (BP) [95]. A loss function, for example, mean square error or cross-entropy, is used to calculate the error between the desired output and the output of the network. Then, this error back-propagates through the networks to update the weights using BP.

Training NNs on a small dataset may cause overfitting [100]. Overfitting is defined as a model which is too closely fit to training dataset. There are reasons why overfitting happens to the model. First one is the training dataset that does not have enough examples. Second one is a model has many parameters. The last one is a model is too complex. The model is only good on the training dataset but not on an unseen or a testing dataset. Also, underfitting may happen when a model is too simple and not well-suited to both the training dataset and the testing dataset. These are depicted in Figure 2.2. A good model performs well not only on the training dataset but also on the testing dataset.

2.2.2 Neural networks in image classification

One well-studied and advanced application of neural networks is image classification. Image classification is the processing of taking an image input and predicting the target class of the image [1]. Recently, most of applications for image classification is with the convolutional neural network (CNN), so why is CNN more often applied to

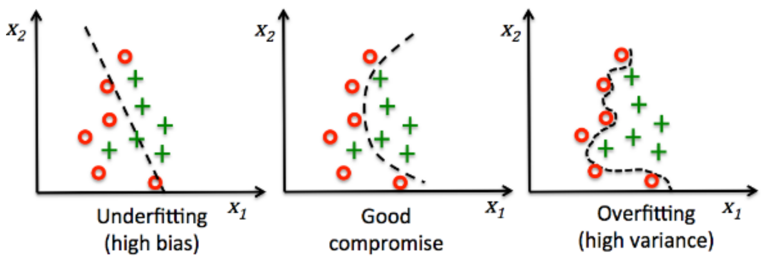


Figure 2.2: Examples of underfitting, generalized well, and overfitting. The figure is from [89]. Bias means the error on the training dataset, and variance means the error on the testing dataset.

image classification than a multilayer perceptron (MLP)?

MLP is very useful for tabular datasets such as CSV and spreed sheet files with fixed features in specific cells of a table. However, there are two problems for applying MLP to image classification. One of the problems of MLP is weight sizes are depending on dimensionality of images. For example, if an image of MNIST dataset [65] is 28 × 28 × 1 (greyscale) = 784 pixels, and there are 10 nodes in a hidden layer, total number of weights from an input to a hidden layer will be 7840. This is still manageable. However, if an image higher dimensions than MNIST such as 128 × 128 × 3 (RGB) = 49152 pixels, and there are 10 nodes in a hidden layer, total number of weights from an input to a hidden layer will be 491520. This means a huge number of weights are required for training neural networks. The other problem of MLP is not translation-invariant. MLP reacts differently to images when the images are shifted. For instance, after we train a simple MLP on MNIST and shift an image of number to the top-left (Figures in 2.3), the MLP misclassifies the image.

CNN solves these problems inherent in MLPs. CNN has an ability to extract internal representations of two-dimensional images using smaller weight sizes (or kernel sizes) [4]. The size of the weights for CNN is M × N squares, and these M and N are smaller than the images. These weights are shared by all nodes in a feature map. This is called weight sharing. Furthermore, CNN is translation-invariant through a sliding kernel that builds in shift-invariance of higher-level features such as edges anywhere in images.

In our study, we implemented our proposed model based on CNN for image classification, specifically adversarial examples. However, our model might be also useful to natural language processing and tabular datasets since our model use stochastic

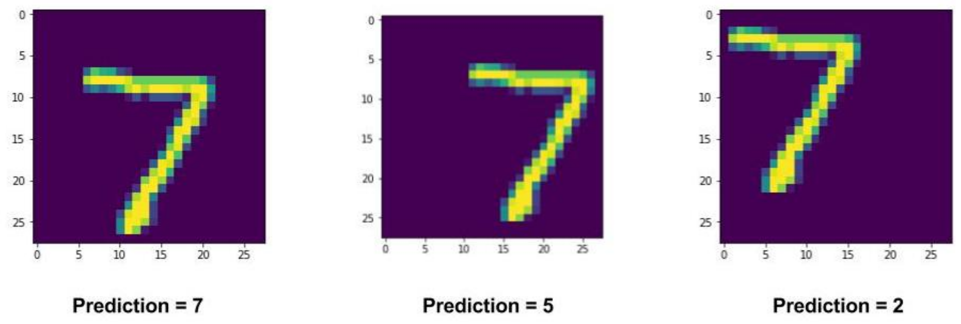


Figure 2.3: Examples of shifted images on MNIST and predictions of MLP in bottom of the images. Left: centered image (Original), middle: shifted to right, right: shifted to the top-left. MLP cannot classify the shifted images anymore. Figures from [3]

masking for the weights. This stochastic masking might ignore some unrelated features of inputs to improve the performance for models. We will discuss this stochastic masking that ignores some perturbations on adversarial examples, and it helps our model's robustness.

In the next subsection, we will discuss convolutional neural networks in detail.

2.2.3 Convolutional neural networks

One variant of a FNN is the convolutional neural network (CNN) [64, 59, 2]. The traditional FNN receives inputs and transforms them through a series of hidden layers. These hidden layers are sets of neurons, where each neuron is fully connected to all of the neurons in the previous layer. However, these connections in each layer are not shared. A convolutional layer replaces the dense to sparse interactions (or sparse connectivity of weights) using smaller filters than inputs. The shared filters look at only a small region in the inputs and move from the left and right spatially in the inputs. Usually, these filters are M $\times$ N squares, and these M and N are smaller than the inputs' width and height. These small filters are used to detect specific features or patterns of inputs such as edges and are used to convolve across the widths and heights of the inputs and compute dot products between the filters and inputs at any positions. This could be described as follows:

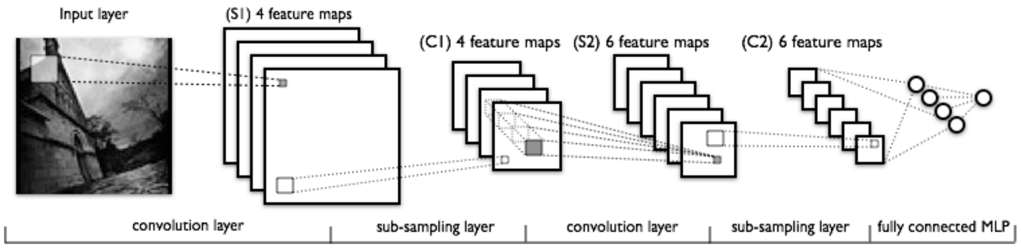


Figure 2.4: An example of convolutional neural network architecture, Lenet. Figure from [64]. In this case, there are two convolutional layers, two max-pooling layers after each convolutional layers, and one fully connected layer.

$$y_{i,m,n} = f(\sum_j \sum_q \sum_r x_{j,m+q,n+r} w_{i,j,q,r} + b_i), \tag{2.3}$$

where i and j are indices for the channels of the outputs and inputs, respectively. Also, m and n are the locations of images in 2-dimensional coordinate, and q and r are indices for the locations for the filters in 2-dimensional coordinate. These filters can be much smaller than the inputs, which reduces the number of parameters compared with an MLP. The convolutional layer produces 2-dimensional activation maps, which are passed to a next layer in CNN. A CNN is usually comprised of one or more convolutional layers and followed by one or more fully connected layers. Also, there are filters in each convolutional layer to extract many kinds of features at spatial locations. These architectures help CNN to detect local features and global features of objects. For the local features, the filters detect many local features in positions with a large variety of poses. For the global features, many layers achieve high-level representation using each layer that detects the local features.

Pooling

Another key feature of a CNN which is used to help the detection of features is pooling. Pooling operators subsample the representation. Pooling layers are usually added after each convolutional layer. These pooling layers replace the outputs of the convolutional layers at certain locations with a summary of the nearby outputs. For example, max-pooling layers take maximum values of certain locations of convolved images. This means the images are subsampled by only considering the maximum feature represented by the filters in the locations. The locations span outputs of the convolutional layers, and in the simplest case are non-overlapping. The most common pooling layers are max-pooling with filters of size 2 $\times$ 2 applied with a stride of 2 downsamples. This reduces the widths and heights of the inputs to the half. This reduction is crucial for object recognition, especially large image recognition. The reduction reduces the parameter sizes and computation time.

Batch normalization

One of the techniques used to accelerate training a CNN is batch normalization [51]. The normalization layer is usually inserted after an activation function to make each dimension of activation outputs unit Gaussian [66]. To apply batch normalization to each layer, mini-batch mean μ_B and variance σ_B are calculated as follows:

$$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i, \tag{2.4}$$

$$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B), \tag{2.5}$$

where m is a mini-batch size and i is an index for the mini-batch. These μ_B and σ_B are used to normalize the input of this normalization layer and to make each dimension of the input unit Gaussian:

$$\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}, \tag{2.6}$$

where ε is a small float number to avoid dividing by zero. There are learnable parameters γ and β of batch normalization. These parameters are used to scale and shift the input if the network does not want zero-mean and unit-variance, and allows the network to recover the raw input or the identity mapping if the raw input scale is useful. These γ and β are used as follows:

$$y_i \leftarrow \gamma\hat{x}_i + \beta, \tag{2.7}$$

where y is an output in the batch normalization layer. Usually, a higher learning rate causes exploding or vanishing the gradients. The gradients for outlier activations with the higher learning rate may explode or vanish. Since batch normalization reduces these outlier activations, the higher learning rate can be used for faster convergence.

Dropout

When there is not enough training data for a model, overfitting might happen. To avoid this problem, dropout [104] is often used for models, especially dense layers for deep convolutional neural networks. This technique drops out nodes on each layer with uniform probability, and this is mostly applied to the networks during training but not testing.

Dropout probability is most often set to a constant, but this dropout probability can be learned and changed depending on inputs, which has some relevance with respect to our work. Ba et al. [7] propose this learnable and changeable dropout called *adaptive dropout*. This adaptive dropout jointly trains a neural network with a binary belief network that learns the probability. This learnable probability is described as:

$$P(m_j = 1 | a_i : i < j) = f(\sum_{i:i<j} w_{j,i} a_i), \tag{2.8}$$

where a is an input for a layer, m is a binary variable to be used for dropping out or not, and w is weights for the layer. In this case, i and j are layer indices. For the standard dropout, this m is set to 0.5. This probability is used for Bernoulli distribution to dropout the activations stochastically, or for multiplying the activations by this probability similar to a gaussian dropout [103], however, as mentioned, this probability is learnable with sampling.

L1 and L2 weight decay

L1 and L2 weight decays are regularizers for the networks in the form of weight decay. They can be derived from adding a penalty term to the loss function that represents a prior. A common choice is to encourage small weights with a Gaussian or Laplacian prior around zero. The corresponding L1 and L2 weight decays are described by adding the penalty to the loss function $\mathrm{Loss(w; x, y)}$ as follows:

$$\widetilde{\mathrm{Loss}}(w; x, y) = \mathrm{Loss(w; x, y)} + \lambda \mathrm{||w||^n}, \tag{2.9}$$

where n is the norm size, λ is the regularization parameter that determines the regularization strength, and w are the weights of the networks.

2.3 Binarized neural networks and estimating gradients

A binarized neural network is a neural network with binarized activations, weights, or both. These binarizations are implemented in various ways. We will introduce several ways to implement the binarized neural networks here.

2.3.1 Binarized neurons and gradient estimators

Neural networks used in deep learning often have real-valued activation functions. However, binarized neural networks use binarized neurons where each activation h has a value of 0 or 1. These values are chosen probabilistically depending on the neuron's net activation, as follows:

$$p(h = 1|a; w) = \sigma(a; w), \tag{2.10}$$

where a is the neuron's input vector, and σ is the logistic activation function computed with weights w. This can be implemented by simply comparing the activation probability with a uniform random variable d:

$$h = 1_{d<\sigma(a;w)}, \tag{2.11}$$

where 1_d is the indicator function, which compares the samples p with the uniform distribution $d \sim U[0, 1]$, and returns 1 or 0 .

This equation is not differentiable, thus preventing classic backpropagation. To overcome this problem, there have been proposals for how to estimate and propagate gradients through stochastic neurons [44, 9]. These methods are called straight-through estimators (STEs). One suggestion [9] which we call hard STE, is to back-propagate through the hard threshold function by using a gradient value of 1 even if the argument is positive or negative. Another approach [9], which we call soft STE, is to use the gradient of the sigmoid for the backpropagation, and the soft STE can be further enhanced with a slope-annealing trick [15] that gradually increases the slope of the logistic function. This slope-annealing reduces the discrepancy between the two threshold function and the sigmoidal approximation during the forward and backward pass. This means that for the forward pass, the activations will be constrained to 1 or 0 stochastically (equation 2.12). If the gradient of the logistic function is used for the backpropagation, there is a gap between the forward and backward passes.

To bridge the gap, the slope-annealing trick can be used. The slope-annealing trick modifies h by multiplying the input a by the slope m as follows:

$$h = 1_{d<\sigma(ma;w)}. \tag{2.12}$$

The slope is gradually increased during training, and computation of the gradient of the output with respect to the input will be:

$$\frac{\partial h}{\partial a} = \frac{\partial \sigma(ma)}{\partial a}. \tag{2.13}$$

The other suggestion of estimation of the gradient, introduced by [9]. The algorithm estimates the expectation of the gradient of a loss function with respect to the inputs with a baseline as

$$\mathbb{E}[\frac{\partial \mathrm{Loss(w;x,y)}}{\partial a}] = \mathbb{E}[(h-\sigma(a))(\mathrm{Loss(w;x,y)} - \overline{\mathrm{Loss}}\mathrm{(w;x,y)})], \tag{2.14}$$

where $\overline{\mathrm{Loss}}(w;x,y)$ is a baseline as:

$$\overline{\mathrm{Loss}}(w;x,y) = \frac{\mathbb{E}[(h-\sigma(a))^2\mathrm{Loss(w;x,y)}]}{\mathbb{E}[(h-\sigma(a))^2]}. \tag{2.15}$$

This baseline is used to minimize the variance of the estimation. For this estimator, the estimator requires broadcasting $\mathrm{Loss(w;x,y)}$ throughout the networks [9]. This estimator is a special case of the REINFORCE algorithm [116] which uses a sampling technique to train a neural network in reinforcement learning without any target labels. In the study of [116], the author shows that when the stochastic action h is sampled with probability $p_w(h)$ and yields a reward R then,

$$\mathbb{E}_h\left[\frac{\partial logp_w(h)}{\partial w}(R-b)\right], \tag{2.16}$$

where b is a baseline. The REINFORCE estimator computes the differentiation between the binarized activation h and the probability $\sigma(a)$ to backpropagate it with the loss. Bengio et al. consider that this is equivalent to the original REINFORCE algorithm, which backpropagates the derivative of the log probability of h with respect to the weight with the reward.

2.3.2 Binary neural networks

Courbariaux et al. [18] introduce the first study of binary neural networks (BNN) whose weights are binarized. The authors consider that most of the network consists of real-valued weights, and these real values make the computation time higher because of matrix-multiplications of these values. Also, large amounts of memory are needed for these parameters. The authors propose binary neural networks to eliminate the need for these multiplications by forcing the weights to binarize (+1 or -1). There are two ways to binarize the weights. The first is a deterministic way to binarize the weights based on the sign function as follows:

$$w_b = \begin{cases} +1, & \text{if w} \geq 0, \\ -1, & \text{otherwise,} \end{cases} \tag{2.17}$$

where w_b is the binarized weight, and w is the real-valued weight. The second is a stochastic way to binarize the weights:

$$w_b = \begin{cases} +1, & \text{with probability p} = \sigma(\text{w}), \\ -1, & \text{with probability 1- p,} \end{cases} \tag{2.18}$$

where σ is the hard sigmoid function:

$$\sigma = \text{clip}(\frac{\text{x}+1}{2}, 0, 1) = \max(0, \min(1, \frac{\text{x}+1}{2})). \tag{2.19}$$

These binarizations make the weights smaller than precision weights. These binarizations are known as strong regularizers [18]. The authors discuss that they only binarize the weights during the forward and backward propagations, and not during the parameter update, in order to keep good precision weights which are necessary for stochastic gradient descent to work at all during the update. However, the derivative of the sign function is zero almost everywhere since the gradient of the cost with respect to the sign function quantization before discretization is zero, and this problem does not update the weights at all [19]. To solve this problem, another study by Courbariaux et al. [19] apply the straight-through estimators [44, 9] to BNN. In that study, the authors consider the sign function quantization q :

$$q = \text{sign(r)}, \tag{2.20}$$

where r can be weights, and *sign* is the sign function. To obtain the gradient of the cost function with respect to r, STE is used:

$$g_r = g_q \cdot 1_{|r| \leq 1}, \tag{2.21}$$

where g_q is the estimator of the gradient $\frac{\partial \text{Loss}}{\partial q}$, and g_r is the estimator of the gradient $\frac{\partial \text{Loss}}{\partial r}$. The derivative $|r| \leq 1$ can be seen as propagating through the hard tanh *htanh* as follows:

$$\text{htanh(x)} = \text{clip(x}, -1, 1). \tag{2.22}$$

Courbariaux et al. [19] use these methods for activations to binarized to +1 or -1 in order to more greatly reduce the computation time and memory usage.

BNN are one of the successful models against adversarial examples with one of adversarial training, the projected gradient descent (PGD)[70]. Galloway et al. [34] apply BNN with PGD to both white- and black-box attacks, and BNN are very robust against these attacks because of combining PGD and binarization, which is one of the gradient masking approaches. In this study, BNN weights and activations are quantized to +1 and -1 stochastically or deterministically using Bernoulli samples (the authors show that implementation in TensorFlow in the paper like Bernoulli(probs=tf.clip by value((x + 1.)/ 2., 0., 1.))).sample() -1) and the sign function, respectively. They find that the stochastic quantization for BNN is helpful to defend against an iterative attack.

2.3.3 Stochastic activation pruning networks

Stochastic activation pruning (SAP) applies randomness to a neural network to defend against adversarial examples [21]. SAP stochastically masks the activation in each layer. The probability of masking activations, p is defined as

$$p_j^i = \frac{|h_j^i|}{\sum_{k=1}^{a^i} |h_k^i|}, \tag{2.23}$$

where h is the activation output for each layer. i, j, a are the indices of the layer, individual activation, and the total number of activations, respectively. These probabilities are used to drop out the activations using the reweighting factor,

$$q_j^i = \frac{\mathbb{1}(h_j^i)}{1 - (1 - p_j^i)^{r_p^i}}, \tag{2.24}$$

where r is the number of outputs sampled, $\mathbb{1}$ is the indicator function that returns 1 if h^i_j is sampled at least once and 0 otherwise. This q will be used;

$$M_p(h^i) = h^i \odot q^i_j, \tag{2.25}$$

where $M_p(h^i)$ is a new activation with dropping out the activations. The authors apply the SAP technique to a pre-trained deep network (ResNet [42]). Applying SAP to the networks reduces the performance of classification on a clean dataset slightly, but increases the robustness against adversarial examples [6]. The SAP model is very similar to dropout [104]. The difference between SAP and dropout is that SAP is likely to sample activations with weighted probabilities in absolute values which are changeable depending on inputs, whereas dropout sample probabilities are always constant.

2.4 Adversarial examples and training

In this section, at first, we will give an overview of crafting adversarial examples, and discuss three white-box adversarial attacks. One is the fast gradient method (FGSM), another one is an iterative method, the basic iterative method (BIM), and the last one is an iterative and targeted, the iterative least-likely class method (ILL).

2.4.1 Overview of crafting adversarial examples

For crafting an adversarial example, the purpose is to maximize $\mathrm{Loss(w; x + \gamma, y)}$ for each clean example x, where w are the weights of a model, and y is the target, and γ is an adversarial perturbation. This $x + \gamma$ is replaced by $\tilde{x}$ as an adversarial example. The $\tilde{x}$ should be similar to the clean image x or perceptible by humans, and the prediction $\tilde{y}_{pred}$ of the model for $\tilde{x}$ and y should be $\tilde{y}_{pred} \neq y$.

2.4.2 Fast gradient sign method

The fast gradient sign method is a common method for crafting adversarial examples [37]. After training a model on a training dataset, the model parameters are frozen. The derivative of the cost function with respect to the inputs, $\mathrm{Loss(w; x, y)}$, is used to calculate adversarial examples $\tilde{x}$ with a sign function as:

$$\tilde{x} = x + \epsilon \cdot \mathrm{sign}(\nabla_{\mathrm{x}} \mathrm{Loss}(\mathrm{w}; \mathrm{x}, \mathrm{y})), \tag{2.26}$$

where x is the input, y is the target, and ϵ is the hyperparameter to change the perturbation level or magnitude. Intuitively, this method attempts to increase the cost function by adding the calculated perturbation to the inputs.

2.4.3 Basic iterative method

The basic iterative method (BIM) [62] iteratively applies the FGSM to images with a small step size. In this method, the pixel values are clipped after each step. The equation for crafting perturbed images using this method is given by

$$x^t = x^{t-1} + \epsilon \cdot \mathrm{sign}(\nabla_{\mathrm{x}^{\mathrm{t}-1}} \mathrm{Loss}(\mathrm{w}; \mathrm{x}^{\mathrm{t}-1}, \mathrm{y})), \tag{2.27}$$

where $t = 1, ..., T$ describes the number of iterations, x^0 is the original image, and $\tilde{x} = x^T$ is the perturbed image after the iterations.

2.4.4 Iterative least-likely class method

The iterative least-likely class method also applies perturbations to images iteratively with a small step [62]. However, this method crafts a perturbation which moves the image toward the class with the lowest confidence probability in the original prediction. The equation of this method is given by

$$x^t = x^{t-1} - \epsilon \cdot \mathrm{sign}(\nabla_{\mathrm{x}^{\mathrm{t}-1}} \mathrm{Loss}(\mathrm{w}; \mathrm{x}^{\mathrm{t}-1}, \mathrm{y}_{\mathrm{llc}})), \tag{2.28}$$

where $t = 1, ..., T$ is the number of iterations, x^0 is the original image, $\tilde{x} = x^T$ is the perturbed image after the iterations, and y_{llc} is the least-likely class for a machine learning classifier.

2.4.5 Adversarial training

Overview of adversarial training

The first study of adversarial training was conducted by Goodfellow et al. [37]. This study shows that training a model on a training dataset and adversarial examples simultaneously improve accuracy on an adversarial attack. Most of the studies replace half of mini-batch with adversarial examples to train the models. For adversarial training, a loss is described in [37] as:

$$\widehat{\mathrm{Loss}}(\mathrm{w};\mathrm{x},\mathrm{y}) = \tau\mathrm{Loss}(\mathrm{w};\mathrm{x},\mathrm{y}) + (1-\tau)\mathrm{Loss}(\mathrm{w};\mathrm{x}+\gamma,\mathrm{y}), \tag{2.29}$$

where τ is the constant weight for the loss (when τ is 1, the adversarial loss ($\mathrm{Loss}(\mathrm{w};\mathrm{x}+\gamma,\mathrm{y})$) will completely be ignored, and the conventional loss ($\mathrm{Loss}(\mathrm{w};\mathrm{x},\mathrm{y})$) will be used). $\tau = 0.5$ is used for replacing the half of mini-batch with adversarial examples to train the models. However, one of the studies for adversarial training replaced the entire mini-batch with adversarial examples [49], and their method improves the robustness of neural networks against adversarial examples. Kurakin et al. [62] mention that adversarial training acts as a regularizer. Also, Kurakin et al. find that a deeper model benefits more than a shallower model from adversarial training, and does not reduce accuracy much on clean examples compared with the shallower model. However, Galloway et al. [33] point out that adversarial training can be the cause of overfitting to specific perturbations. The authors show that an adversarially trained network on FGSM is strong against FGSM but are weak against an unseen attack.

Projected gradient descent

Projected gradient descent (PGD) is introduced by Madry et al. [70]. Their defense uses the basic iterative method to train a model. It consists of min-max optimization to defend the model against adversarial examples as follows:

$$\underset{\theta}{\mathrm{argmin}} \left[E_{(x,y)\sim D} \left[\max_{\gamma\in\mathrm{G}} \mathrm{Loss}(\mathrm{w};\mathrm{x}+\gamma,\mathrm{y}) \right] \right], \tag{2.30}$$

where D is a data distribution, and G is a set of allowed perturbations. Equation 2.30 describes that the inner maximization aims to find the adversarial perturbation for x that reaches a high loss. The outer minimization finds model parameters that minimize the “adversarial loss”. Madry et al. point out that PGD is equivalent to BIM. This means that when applying PGD to a network, adversarial examples crafted by BIM are used for adversarial training. PGD is well used for defending models against both white- and black-box attacks, but Carlini et al. [11] question whether these strong attacks will increase the robustness against all of the attacks or not similar to the discussion put forth by Galloway et al. [33].

2.4.6 Adversary distortion measurement

In our study, we check the adversary distortion between perturbed and clean images using a method, ℓ_1. We briefly discuss them in this subsection.

ℓ_1 and ℓ_∞ measurement

There is a commonly used method to measure the closeness between perturbed and clean images [121]. ℓ_m measures the magnitude of perturbation by m-norm distance:

$$||x||_m = \left(\sum_i^n ||x_i||^m \right)^{\frac{1}{m}}. \tag{2.31}$$

ℓ_1 measure the mean absolute distance between the perturbed and clean images; ℓ_∞ denote the maximum absolute change for all pixels in adversarial examples. If these distortion measurements or distances are very small, the perturbed and clean images are very similar. In our experiment, ℓ_1 will be used to measure the distortion.

Chapter 3

Mitigating Overfitting Using Regularization to Defend Networks Against Adversarial Examples [60]

3.1 Motivation

The cause of adversarial examples is still debated. Goodfellow et al. [37] argued that adversarial examples generated by one model are often also misclassified by other models which consist of different architectures or are trained on disjoint training datasets. Because of these behaviors, the authors argue that overfitting is *not* related to the cause of adversarial examples. Instead of overfitting, their explanation for the cause of adversarial examples in neural networks, even deep neural networks, is that they are "too linear" (a small perturbation can grow linearly with the dimension of parameter size) including the activation units that are piecewise linear functions such as rectified linear units or sigmoid. Moreover, in their paper, the authors applied a modest L1 weight decay to the first layer of a maxout network that did not resolve the sensitivity to adversarial examples. They concluded that smaller weight decay permitted successful training, but conferred no regularization benefit. Furthermore, Papernot et al. [87] mention that L1 and L2 weight decay will cause underfitting and therefore should be discarded as a solution to avoid decreasing classification accuracy on a clean dataset.

On the other hand, Galloway et al. [35] address how researchers have falsely concluded that overfitting is not related to the cause of adversarial examples. The authors show that pruning overfitting using strong L2 weight decay could be a way to improve the network accuracy on adversarial examples. In their paper, the authors suggest that starting with smaller models that can handle strong weight decay is worth exploring as a natural defense against adversarial examples.

However, the legitimacy of the argument presented by Goodfellow et al. [37] is still unclear. Even if the models are trained with different architectures or on the

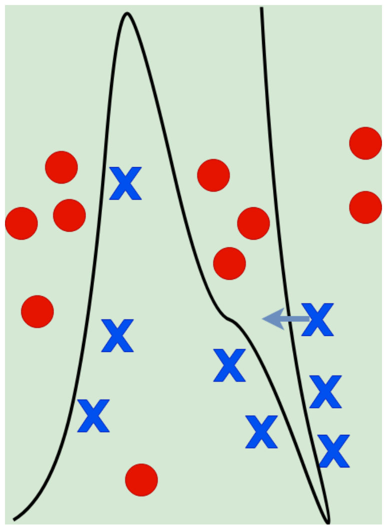


Figure 3.1: An example of the overfitting decision boundary. The overfitted decision boundary can easily create adversarial examples since examples are very close to the boundary.

disjoint datasets, these models might learn similar decision boundaries because these datasets are not completely different.

In this chapter, we add to these arguments that overfitting contributes at least somewhat to adversarial sensitivity. If overfitting is not problematic, the accuracy performances of well-generalized networks and overfitted networks on adversarial examples would not be considerably changed. We consider that harmful adversarial examples would be hard to create if a decision boundary is well-generalized, but an overfitting decision boundary would easily flip an example to another class by adding perturbations. An example of overfitting decision boundary is shown in figure 3.1.

Hence, we study the performances of neural networks with various forms of regularization on adversarial attacks compared with a neural network without any regularizations. In addition to fairly standard methods of regularization such as dropout and weight decay, stochastic binarized neurons are also regularizers [44]. We therefore also study the performance of stochastic binarized neural networks on adversarial examples.

More specifically, in this chapter, we apply dropout [104] to a shallow convolutional neural network (CNN) with low to high dropout probabilities. In [37], the authors applied dropout to a maxout network, but they did not check whether or not changing the strength regularization affected the network performance on adversarial examples.

We check the relationship between the overfitting rates and accuracies by changing the dropout probability on adversarial examples. Similar to Wang et al. [114], we study dropout as a regularizer with adversarial examples. However, unlike Wang et al., we do not use dropout during testing. Rather, we specifically check if overfitting is a problem. Furthermore, we apply L1 and L2 weight decay to the shallow CNN. Although Galloway et al. [35] have already shown that L2 weight decay improves the network performance on adversarial examples, here we add a wider range of λ from weak to strong in response to Papernot et al. [87], and we show that accuracy is improved when λ is well-tuned.

Finally, we apply stochastic binarized neural networks to adversarial examples. Galloway et al. [34] have already shown that the binary neural networks (-1 or 1 on weights and activation) with the straight-through estimator (STE, in this case, soft STE with the slope-annealing trick) [9] improved robustness against adversarial examples. In our study, we investigate a stochastic neural network with a special case of the REINFORCE estimator [9] and compare it with STE and a deterministic binarized neural network. In our case, our binarizations are only for activations constrained to 0 or 1. We conduct these experiments on the perturbed MNIST dataset [65].

To generate adversarial examples, we use a white-box attack with the fast gradient sign method (FGSM) [37]. This method is fast and straightforward to compute, and strong perturbation levels are possible. Furthermore, we do not use adversarial training, but we train the networks on a clean dataset with the regularizations.

These results have been published and presented at the Canadian Conference on Artificial Intelligence (CanAI 2019).

3.2 Experiments

3.2.1 Parameter setting

We start by applying a shallow convolutional neural network (CNN) with changing dropout probabilities to the MNIST dataset. This network consists of one convolutional layer with 32 filters whose size is 5×5 and stride is 2. This is followed by a 100-node dense layer and a 10-node softmax output layer. The activations for the convolutional and dense layers are the rectified linear unit except for the output layer. We use dropout after the first dense layer with the dropout probability 10, 30, 50, 70, 90%, and also compare this to CNN without any regularizations.

The second experiment is the same as the first, but we use L1 and L2 weight decay instead of dropout. These regularizers are added to each layer including the convolutional layer(s). The network condition is the same as the first one. The hyperparameter λ are 1e-5, 1e-4, 1e-3, 1e-2, and 1e-1. As before, we compare the results to CNN without any regularizations as a baseline for this experiment too.

Finally, we apply three versions of a binarized MLP, one with the soft STE and the slope-annealing trick [15] (STE-MLP), one based on the REINFORCE estimator (REINFORCE-MLP), and one deterministic binarized MLP (DBN-MLP). The details of architectures are given in table 3.1. In addition, we add one or two convolutional layers to the MLPs to examine if the convolutional layer(s) improve the robustness against adversarial examples. The details of the architectures for one convolutional (1CNN) and two convolutional layers (2CNN) with binarized activations are given in tables 3.2 and 3.3. For the slope techniques in the soft STE, we increased the slope m with the following schedule $m = min(5, 1 + 0.04 * N_{epoch})$ where N_{epoch} is the number of epochs. For the binarized dense layer of DBN, we round the output of logistic activation function for the output $h = round(\sigma(a))$.

Table 3.1: Architecture of multilayer perceptron

Layer	Size	stride	activation
dense	100		stochastic binarized activation
dense	10		softmax

Table 3.2: Architecture of adding one convolutional layer

Layer	Size	stride	activation
conv	32×5×5	2	relu
dense	100		stochastic binarized activation
dense	10		softmax

The networks are trained on the softmax cross-entropy loss function, and we use Adam optimizer [56] and Xavier initialization [36] for all of the networks. The learning rate for the two first experiments is 1e-4. For the binarized neuron experiment, the

Table 3.3: Architecture of adding two convolutional layers

Layer	Size	stride	activation
conv	32×5×5	2	relu
conv	64×5×5	2	relu
dense	100		stochastic binarized activation
dense	10		softmax

learning rates for MLPs, 1CNNs, and 2CNNs are 1e-2, 1e-3, and 1e-3, respectively. The training epoch sizes for the 1st and 2nd experiments are 20 epochs, and for the stochastic neuron experiments, the epoch sizes are 20, 30, and 30 for MLPs, 1CNNs, and 2CNNs, respectively. All of the accuracies are averaged over 10 runs for the models trained from scratch. Experiments are conducted on the MNIST dataset [65] with the regular 60000 28×28 grayscale images as the training dataset and 10000 for the testing dataset.

3.2.2 Results for changing dropout probability

The network without dropout was the least accurate at classifying adversarial examples in this experiment (Figure 3.2). This type of network is more vulnerable to adversarial examples even at small perturbation levels compared with the networks with dropout. When using a dropout technique, it is common to use a dropout probability of 50%, but in this study, a dropout probability of 90% was optimal for the network to mediate adversarial examples. Compared with the network without dropout, the network with a dropout probability of 90% is more robust, especially at the medium perturbation level (at $\epsilon = 0.1$, the difference is around 40%). Furthermore, after training the models, we observed the errors, $error = (100\text{-}accuracy)$, for each model on the clean training and testing datasets to check if overfitting had occurred for each model. In addition, we checked the confidence levels of the target classes (the softmax probability only for target classes) on FGSM when we changed the perturbation level (in this case, $\epsilon = 0.01$, 0.13, and 0.25 for the dropout probabilities of 10 and 90 %) as seen in figure 3.3. In this figure, at $\epsilon = 0.13$, it is clear that most of the target confidence levels for the network with a dropout probability of 10 % are already zero compared with the confidence levels for the network with a dropout probability of 90% that gradually changes from high to low. High dropout probabilities suppress rapid changes from high to low confidence. Additional confidence levels for the networks with and without dropout are available in Appendix A.1.

3.2.3 Results for changing L1 and L2 λ parameters

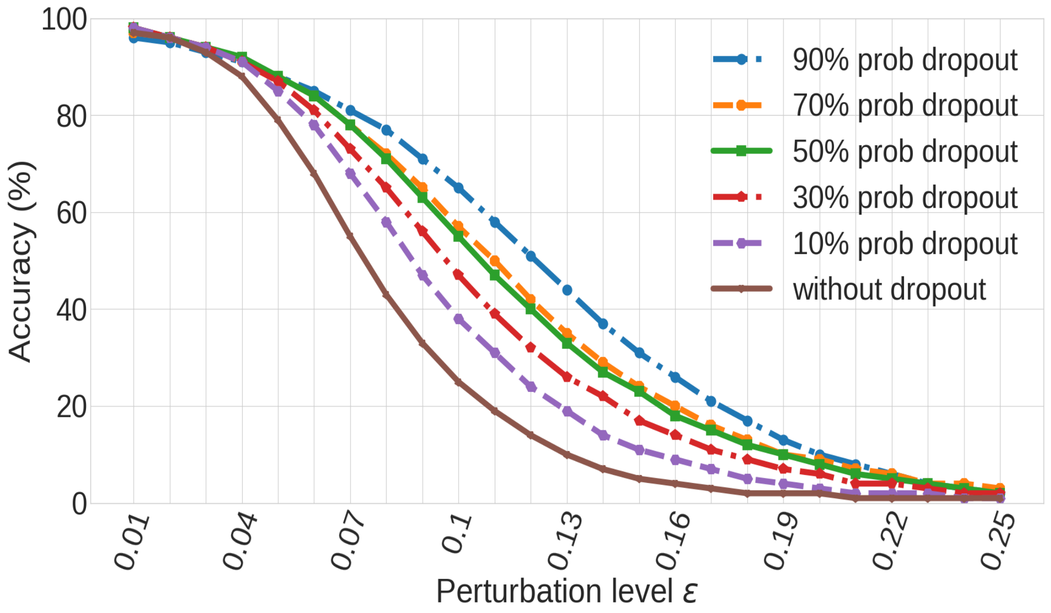
As seen in figure 3.4, strong L1 weight decay helps the networks to defend against adversarial examples. The network with the strongest λ underfits the clean dataset. However, interestingly, this underfitted network was *better* than the overfitted networks at classifying adversarial examples. The overfitted networks are more vulnerable to adversarial examples even at small perturbation levels. Furthermore, as seen in figure 3.5, the results for the networks with L2 weight decay were similar to the L1 results. The accuracies of the networks with $\lambda <$ 1e-2 were not affected by the weight decay at any perturbation level, but the networks with $\lambda \geq$ 1e-2 were more accurate than the networks with $\lambda <$ 1e-2 at medium to higher perturbation levels. We also found that pruning overfitting increases the accuracies of the networks on adversarial examples in this experiment.

The confidence levels of the target classes on adversarial examples were also checked for L1 and L2 as seen in figures 3.6 and 3.7. When both weight decays are overly strong, they might cause the networks to underfit the datasets, reducing the accuracies of the networks. However, if the λ is well-tuned, it helps the networks to improve accuracy on adversarial examples.

Additional confidence levels for the networks both with and without L1 and L2 weight decay are available in Appendix A.2.

3.2.4 Results for binarized neural network

The previous experiments have already demonstrated that strong regularizations help the networks to defend against adversarial examples. We conducted similar experiments with stochastic binarized networks (STE and REINFORCE) and deterministic binarized networks (DBN). Figures 3.8, 3.9, and 3.10 show the accuracies on adversarial examples for MLP, 1CNN, and 2CNN with both stochastic and deterministic binarized neural networks. The figures show that both STE and REINFORCE always perform better than DBN. The graphs at the bottom show the differences in accuracy



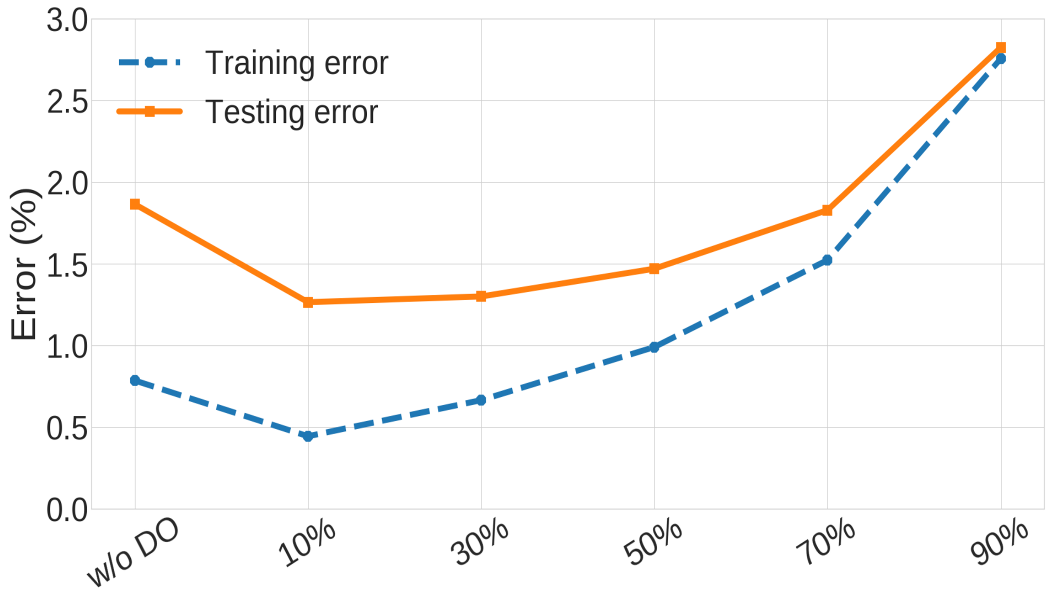


Figure 3.2: Results of the dropout experiments Left: The accuracies for the network with dropout on the perturbed testing dataset. Right: The errors for the networks with dropout and without dropout (w/o DO) on the training and testing dataset after the training for checking the overfitting levels on each model.

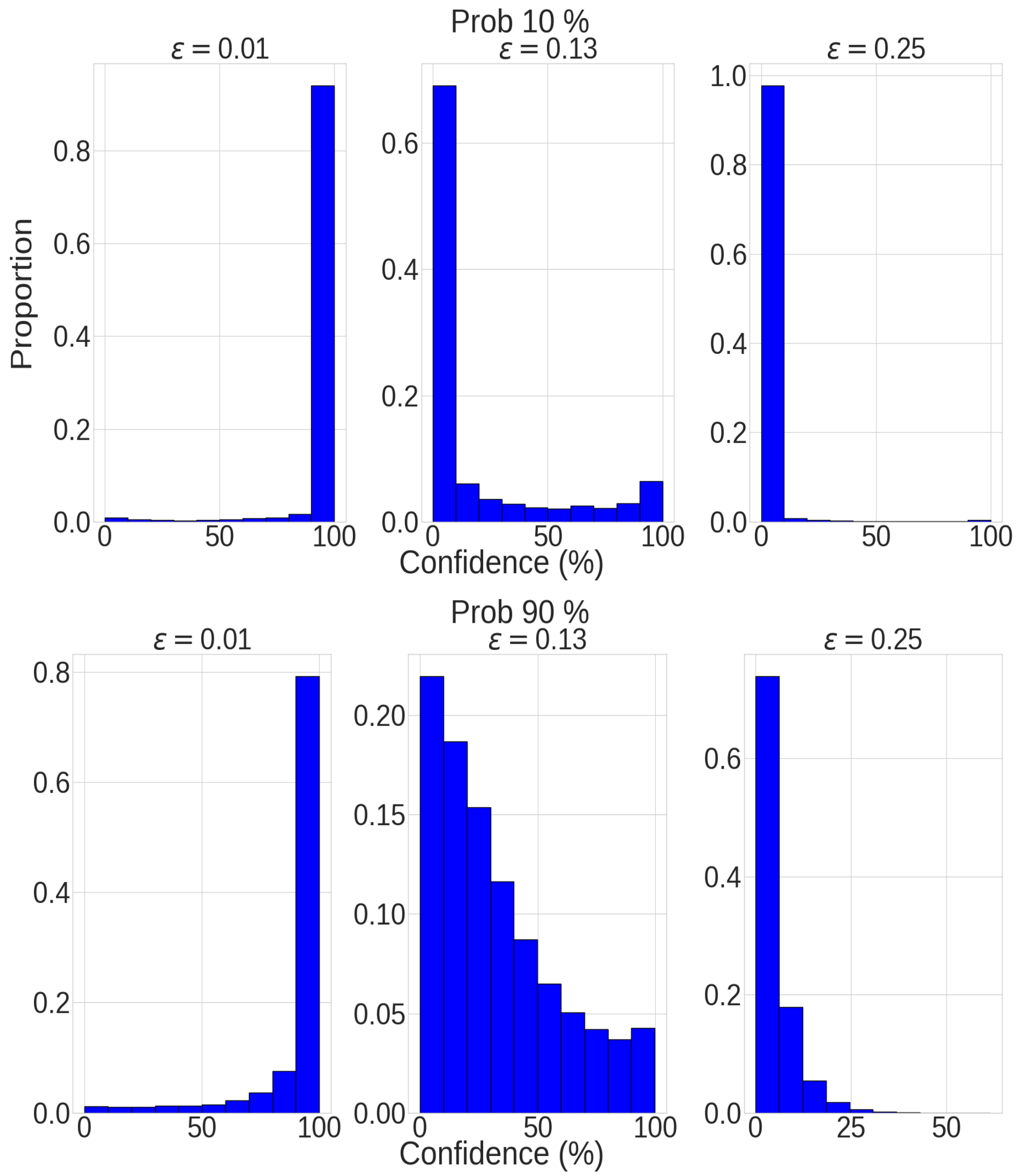


Figure 3.3: Proportion (0 to 1) vs confidence level (0-100) for target classes of adversarial examples on changing dropout probability experiments (10 and 90 % probabilities). ϵ means the perturbation level (in this case, 0.01, 0.13, and 0.25). Dropout probability of 90 % gradually changes confidence level from high to low while increasing the perturbation level. However, obviously, the confidence level for the probability of 10 % changes from high to low compared with the confidence level for probability 90 %.

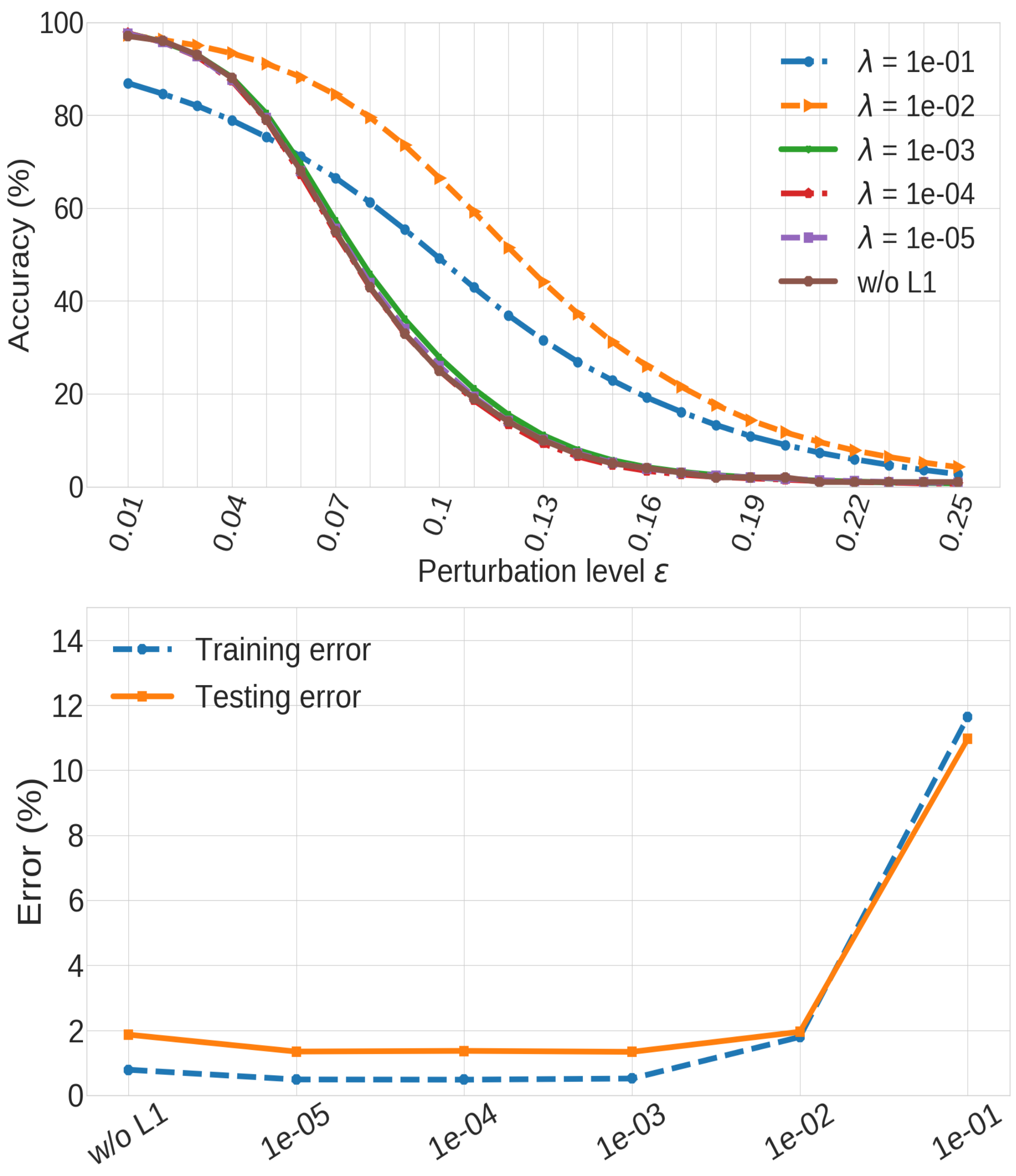


Figure 3.4: Results of the L1 weight decay experiments: The accuracies for the network with L1 weight decay on the perturbed testing dataset. L1 weight decay - strong λ is helpful to defend the networks against adversarial example, but excessive λ cannot be helpful.

between the training and testing datasets to check the overfitting levels for each network. Furthermore, we checked the accuracies for all of the networks on the training and testing datasets as seen in tables 3.4 (MLP), 3.5 (1CNN), and 3.6 (2CNN). Even though the accuracies of REINFORCE networks on the training and testing were not

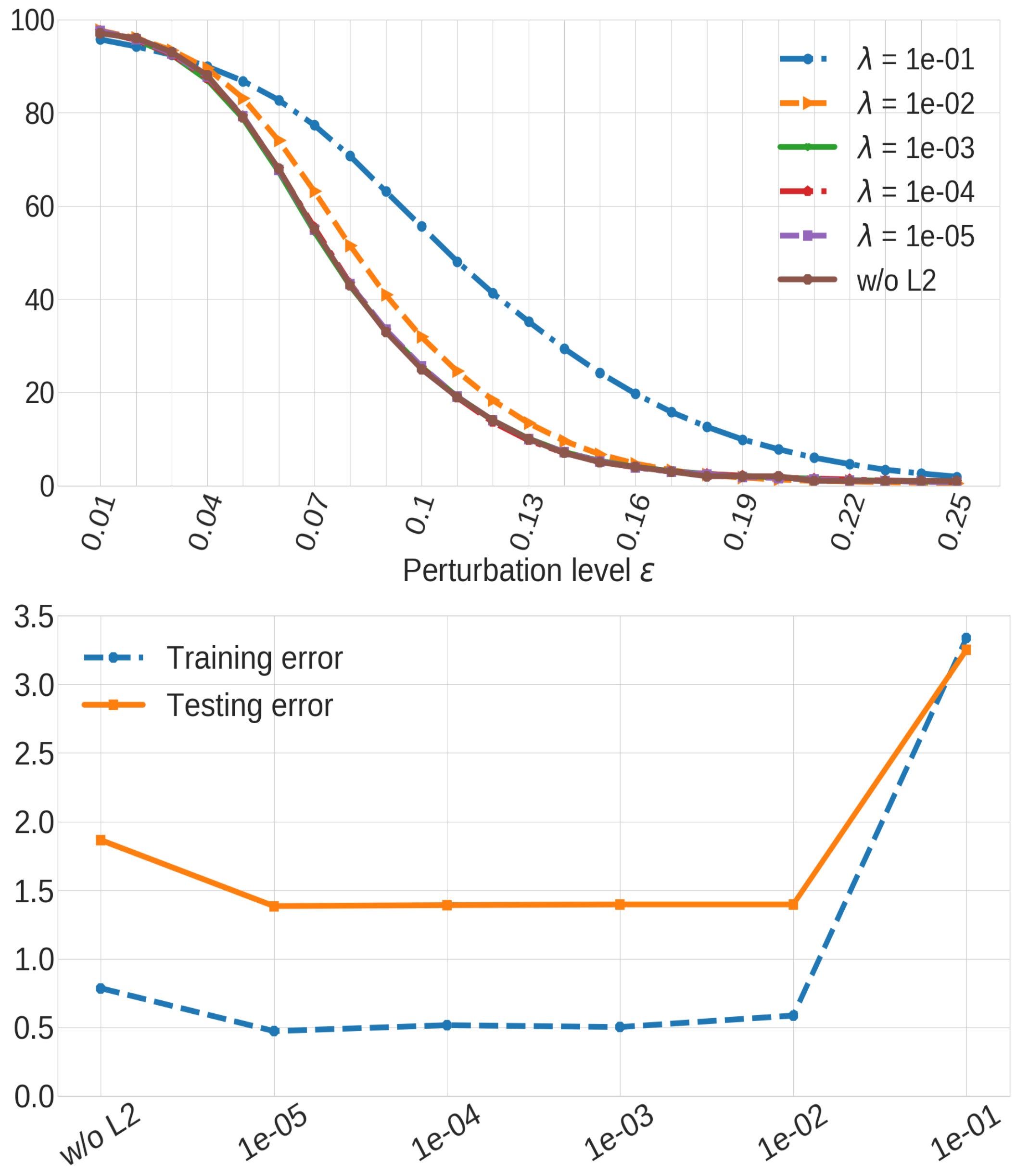


Figure 3.5: Results of the L2 weight decay experiments: The accuracies for the network with L2 weight decay on the perturbed testing dataset. L2 weight decay - strong λ is helpful to defend the networks against adversarial example.

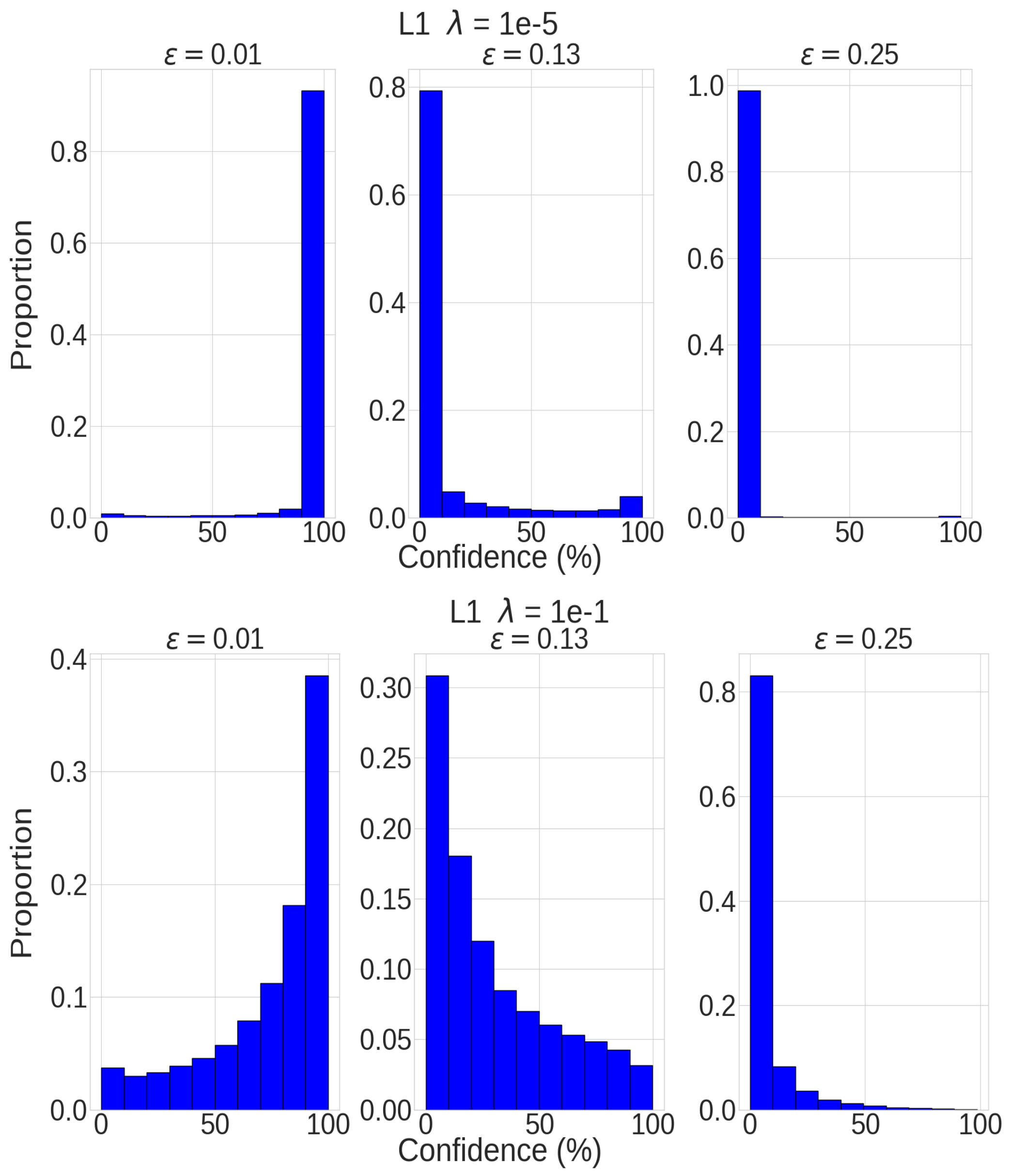


Figure 3.6: Proportion (0 to 1) vs confidence level (0-100) for target classes of adversarial examples on changing L1 λ experiments (λ = 1e-5 and 1e-1). At $\epsilon = 0.13$, most of the confidence levels the network with λ = 1e-5 are zeros. On the other hand, confidence levels for λ = 1e-1 gradually change from high to low.

better than the others, REINFORCE networks always generalized well and were more robust at higher perturbation levels than the others. REINFORCE-1CNN achieved an accuracy of 78.2±2.4% at $\epsilon = 0.25$. STE and DBN had very similar differences in accuracy between the training and testing datasets, but STE was slightly more accurate than DBN against adversarial examples over all types of networks.

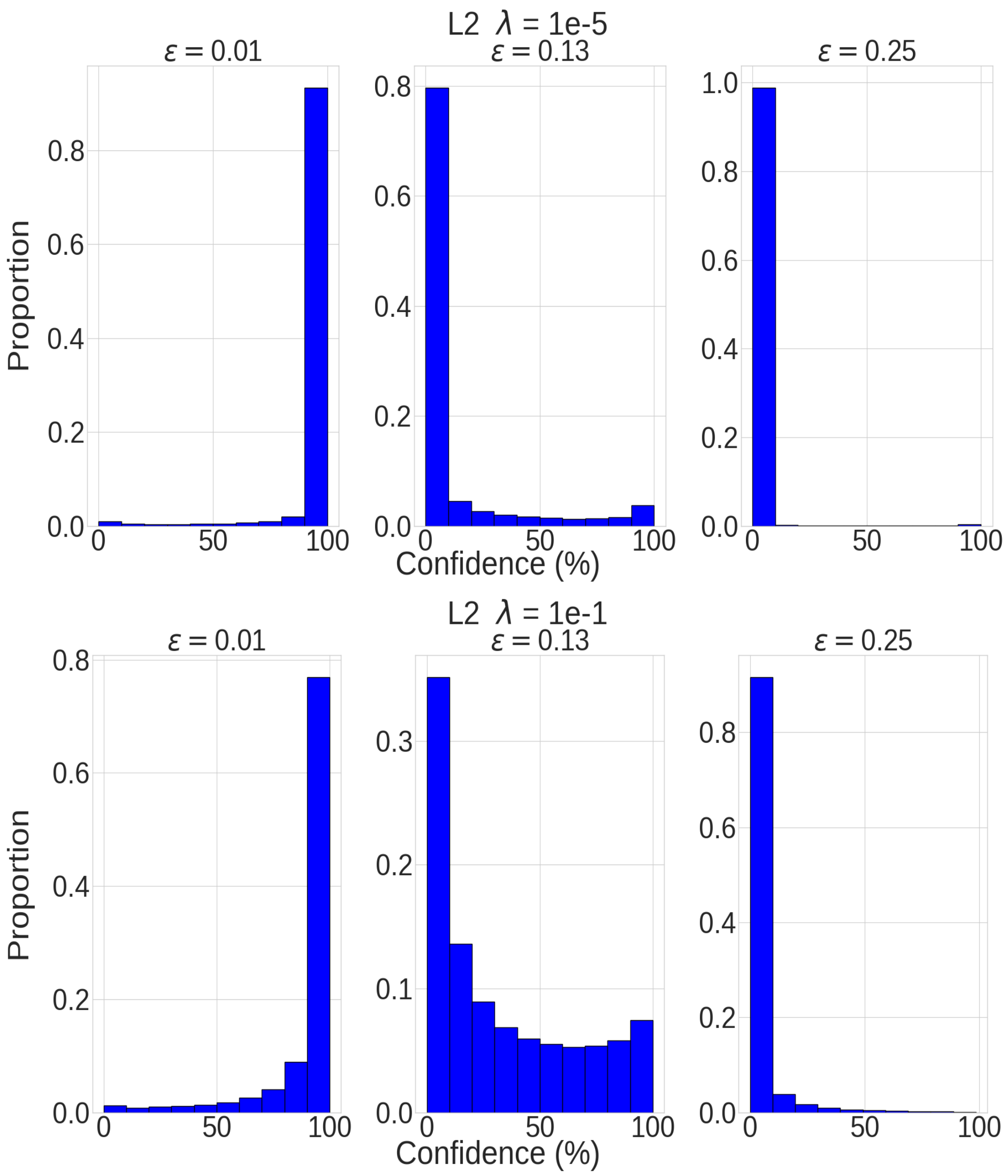


Figure 3.7: Proportion (0 to 1) vs confidence level (0-100) for target classes of adversarial examples on changing L2 λ experiments (λ = 1e-5 and 1e-1). At $\epsilon = 0.13$, most of the confidence levels the network with λ = 1e-5 are zeros. On the other hand, confidence levels for λ = 1e-1 gradually change from high to low.

The confidence levels of the target classes on FGSM for all types of 1CNN were checked in figure 3.11. REINFORCE-1CNN had more robust confidence levels than the other networks on the confidence levels as well. Even at the highest perturbation level, 0.25, most of the confidence levels for the target classes were quite high.

Additional confidence levels for the binarized neural networks are available in

Appendix A.3.

Table 3.4: Training and testing accuracies for each MLP

Data type	DBN-MLP	STE-MLP	REINFORCE-MLP
Training	98.7±0.2%	98.7±0.1%	91.7±0.04%
Testing	96.7±0.1%	96.9±0.1%	91.7±0.05%

Table 3.5: Training and testing accuracies for each 1CNN

Data type	DBN-1CNN	STE-1CNN	REINFORCE-1CNN
Training	99.9±0.02%	99.9±0.1%	96.9±1.5%
Testing	98.7±0.08%	98.7±0.1%	96.7±1.4%

Table 3.6: Training and testing accuracies for each 2CNN

Data type	DBN-2CNN	STE-2CNN	REINFORCE-2CNN
Training	99.9±0.02%	99.9±0.01%	95.6±0.4%
Testing	98.9±0.07%	99.0±0.07%	95.9±0.5%

3.3 Discussion

Regularizations drastically improved the network accuracies on adversarial examples. We have shown the performances here with several regularization techniques, including dropout, weight decay, and binarized neurons. These clearly demonstrated that overfitting contributes to the cause of adversarial sensitivity. Such regularization techniques usually help to smooth decisions, thereby moving them further from training examples.

Regularizers could not completely defend against adversarial examples with strong perturbation levels. The deterioration in accuracy of the recognition networks should, therefore, be evaluated in light of some criteria of picture integrity, such as whether a human would perceive the examples as problematic. Another possible way to catch adversarial examples with strong perturbations is to include an adversarial class in the training dataset and hence learn the characteristics of such attacks.

We checked how different the perturbations between CNN without any regularizations and CNN with L1 weight decay (λ=1e-2) were. These are shown in figure 3.12.

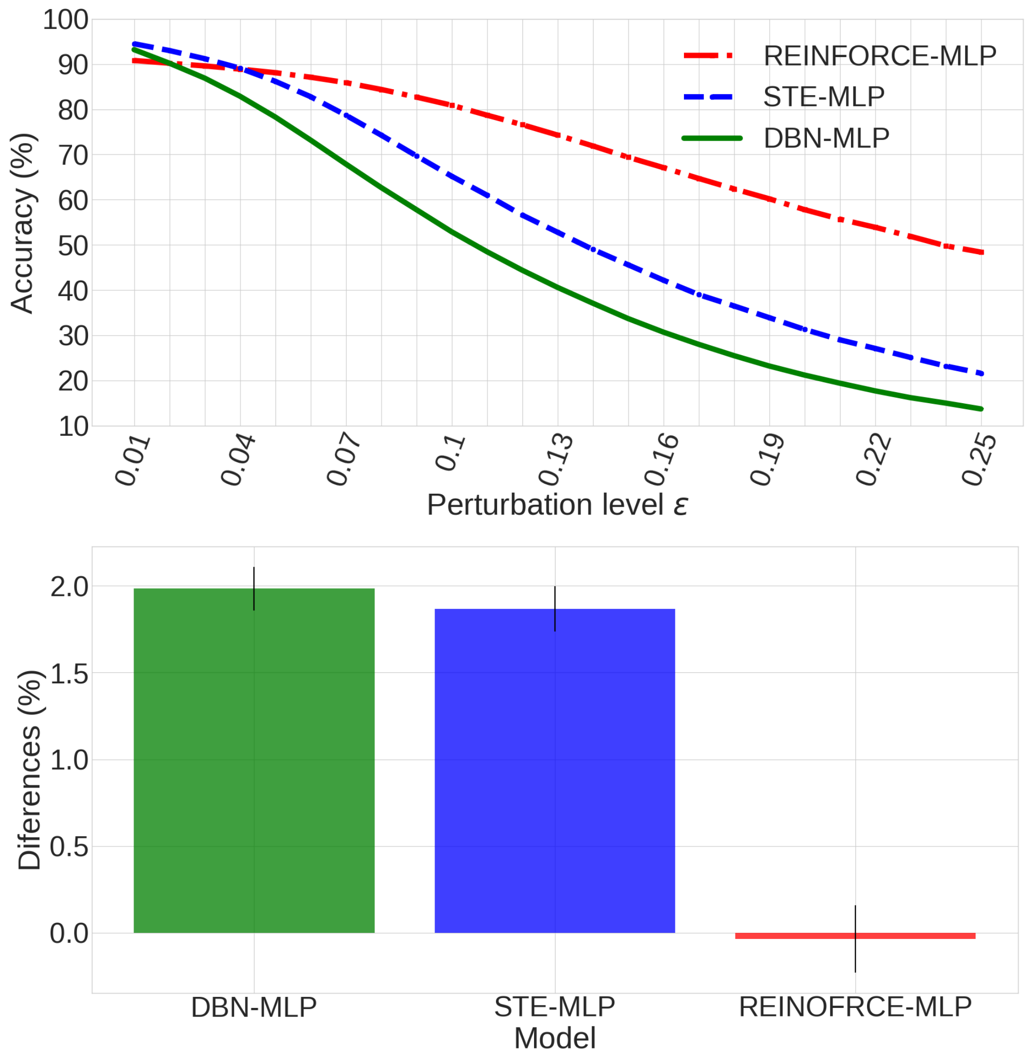


Figure 3.8: Stochastic and deterministic binarized MLP accuracies (top) and the differences in accuracy (bottom) between the training and testing datasets (clean datasets)

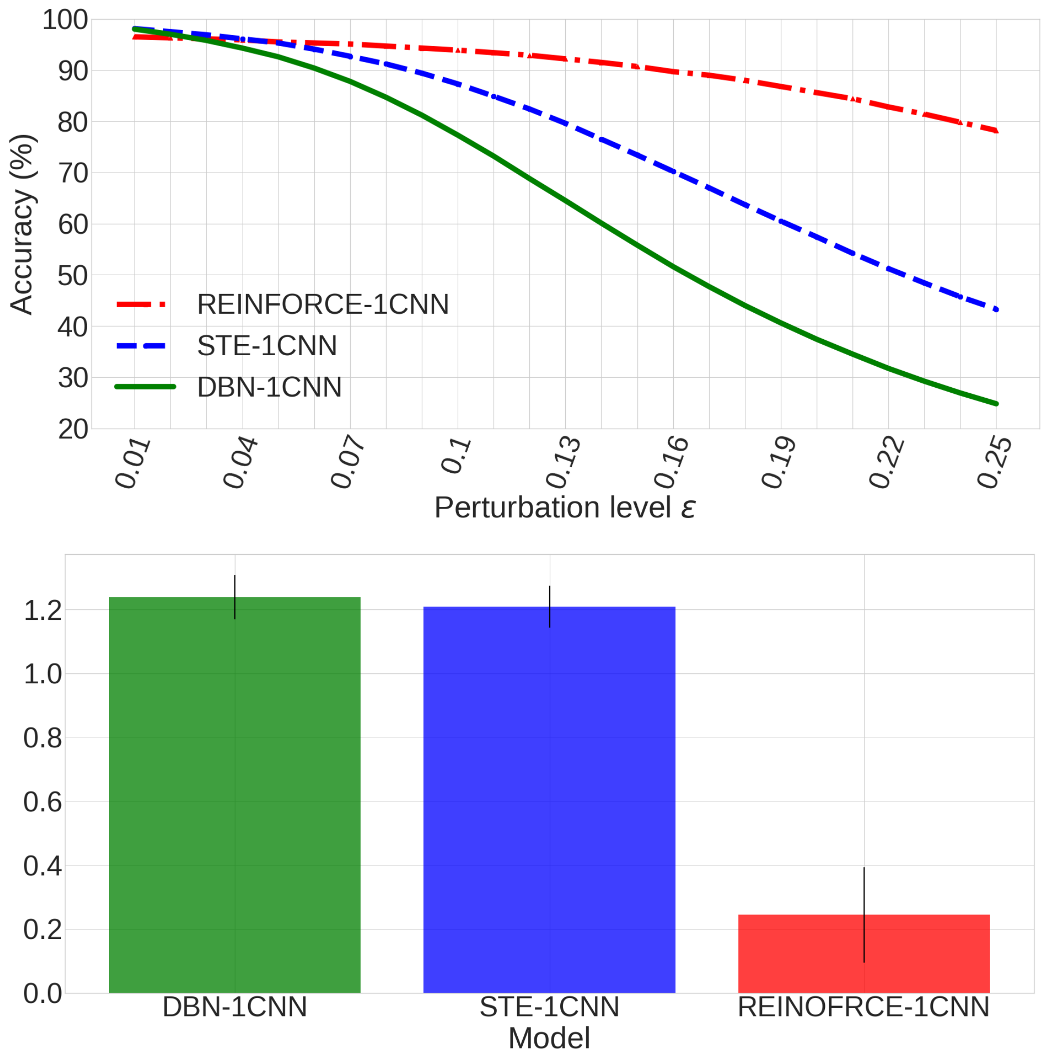


Figure 3.9: Stochastic and deterministic binarized 1CNN accuracies (top) and the differences in accuracy (bottom) between the training and testing datasets (clean datasets)

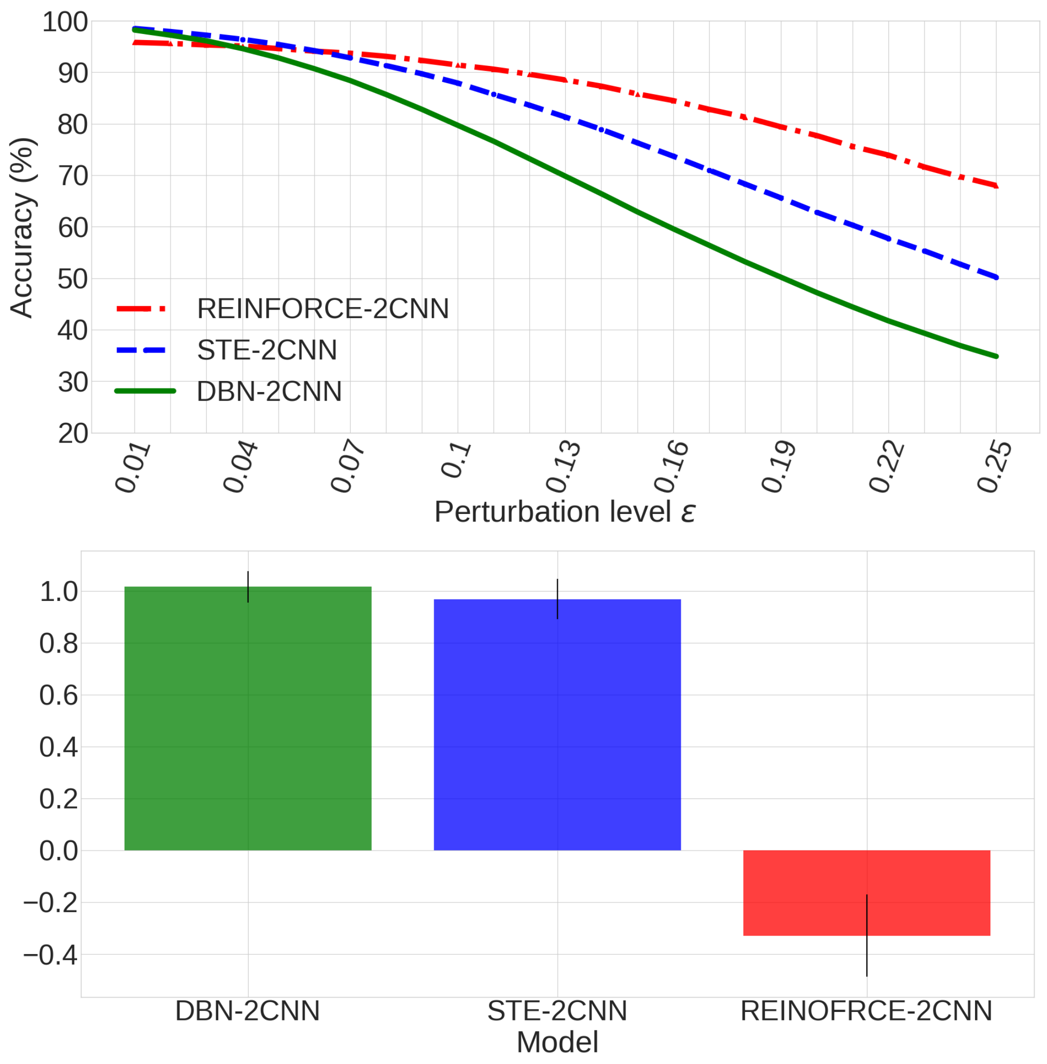


Figure 3.10: Stochastic and deterministic binarized 2CNN accuracies (top) and the differences in accuracy (bottom) between the training and testing datasets (clean datasets)

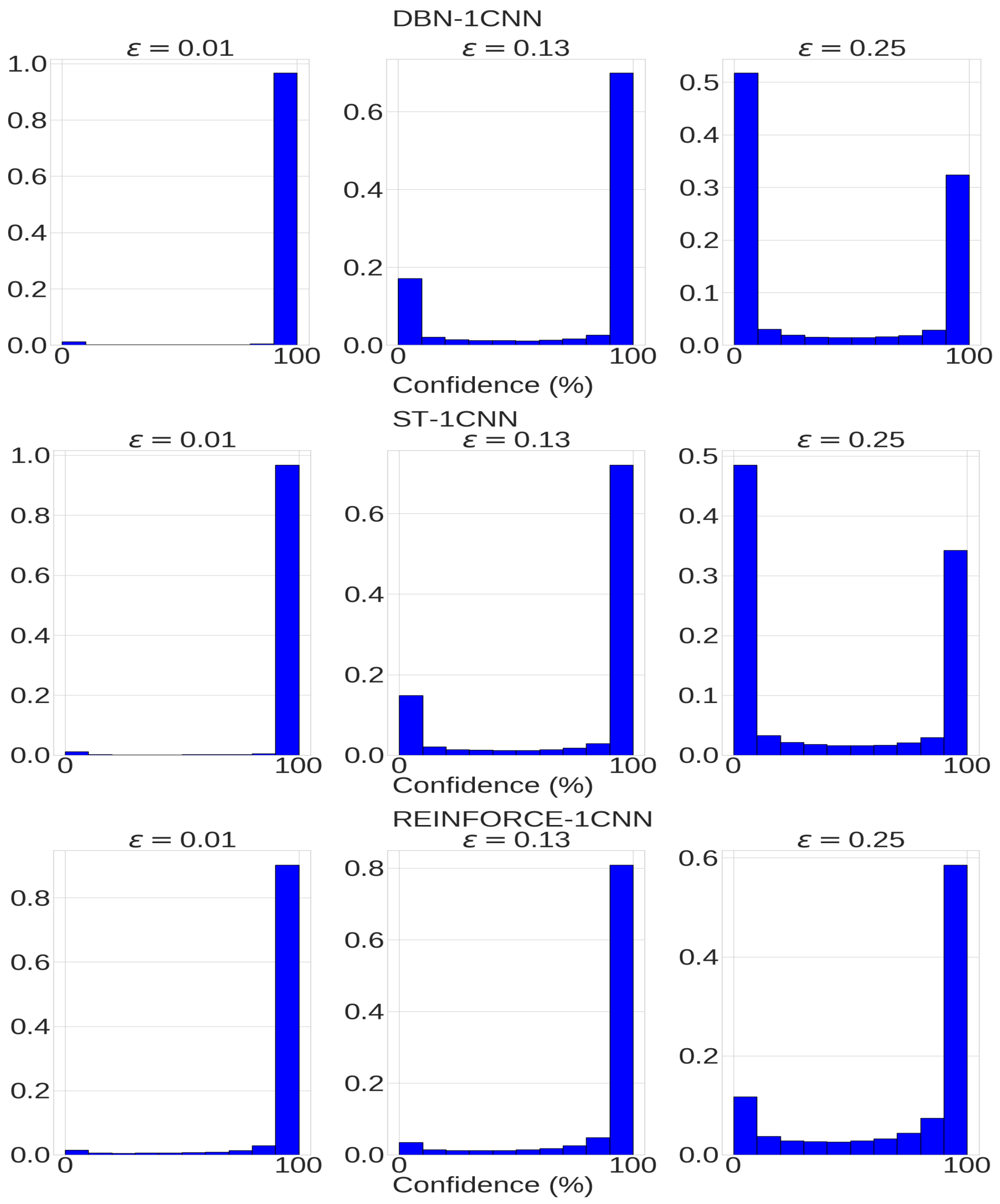
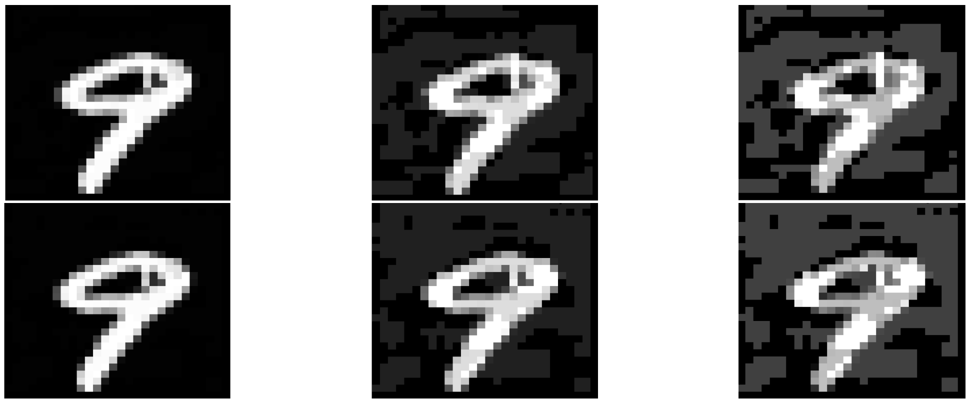


Figure 3.11: Proportion (0 to 1) vs confidence level (0-100) for target classes on stochastic binarized and deterministic experiments for 1CNN. DBN and STE 1CNN have very similar behaviors. Most of the confidence levels at a lower ϵ are 100 %. while increasing ϵ, the proportions for DBN and STE are divided into two, close to 0 or 100%, even at $\epsilon = 0.25$. The proportions of the middle of confidence levels are less. On the other hand, REINFORCE-1CNN are most robust among all of the models. The proportions of high confidence levels for REINFORCE gradually decrease but not too much.



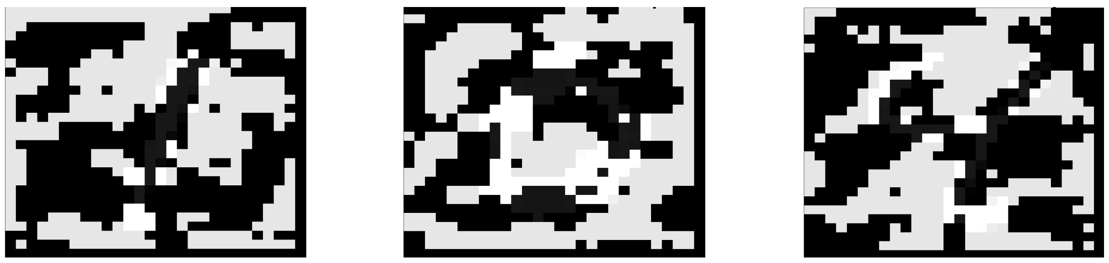
(a) $\epsilon = 0.01$ (b) $\epsilon = 0.13$ (c) $\epsilon = 0.25$

Figure 3.12: Comparison of FGSM for CNN without any regularizations (upper) and CNN with L1 weight decay (λ = 1e-2) (lower). The perturbations for CNN without any regularizations are well spread on images. In contrast, the perturbations for CNN with L1 are centered.

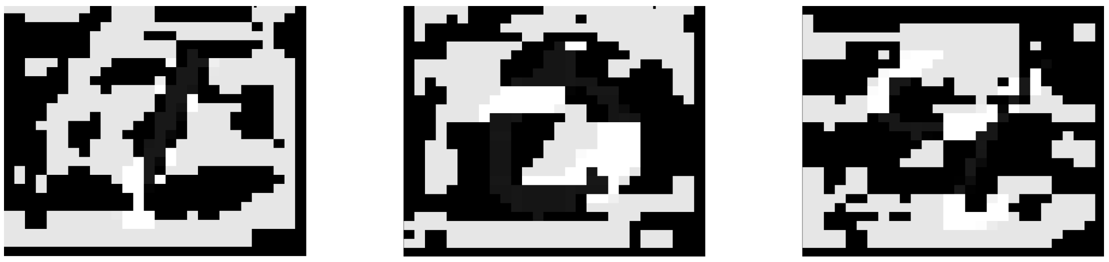
Even though the upper parts of figures for both with and without L1 are slightly different, humans can see that the perturbations with L1 weight decay and without any regularizations are very similar. Furthermore, we checked if humans can still classify adversarial examples crafted by these CNNs when the perturbation level is extremely high. Figure 3.13 shows adversarial examples (FGSM of $\epsilon = 0.9$) crafted by the models. The perturbed images crafted by both CNN with L1 weight decay and without any regularizations are hard for even humans to recognize the targets as 1, 0, and 4.

Overly strong regularizations could be the cause of underfitting such as L1 weight decay. However, we have seen that in our L1 weight decay experiment, the accuracy with overfitting on adversarial examples was *worse* than it was with underfitting. This is another indication that overfitting is related to the cause of adversarial examples. While weight decay and dropout helped with protecting against adversarial attacks, we would suggest using stochastic binarized neural networks for such protection because of their high performances and confidence levels for the correct classes even at high perturbation levels. This would especially be the case with the REINFORCE training of binarized neurons.

A recent study using binary representations of weights and activations with values -1 or 1 showed the robust results on some adversarial examples [34]. This is another



(a) Without any reguralizations



(b) L1 weight decay ((λ = 1e-2))

Figure 3.13: Comparison of FGSM (ϵ = 0.9) for CNN without any reguralizations (upper) and with L1 weight decay (λ = 1e-2) (lower). Left: The figure is 1. Center: The figure is 0. Right: The figure is 4.

example of a regularizer based on binarized neurons [18]. This result also supports the assertion that overfitting is related to the cause of adversarial examples.

Although the network with these regularizations could not protect themselves at high perturbation levels, this study reported that the improvement of accuracies at medium perturbation levels on adversarial examples could be evidence of the relation to the cause of adversarial examples.

Chapter 4

Learning Adaptive Weight Masking Networks for Adversarial Examples [61]

4.1 Motivation

In the previous chapter, we have shown that regularizers are helpful in defending the neural networks against the adversarial examples, especially stochastic binarized neurons. Also, we have seen that stochastic masking approaches are helpful to defend the networks [114, 21], which are also forms of regularization.

We consider here combining these two forms of regularization, i.e. binarization and stochasticity. We thus investigate a partially binarized neural network whose weights are stochastically masking dependence of each input. We call this model a *stochastic-gated partially binarized network* (SGBN). Our model is a type of gradient masking approach since it estimates the gradient of a non-differentiable function (a binarized activation) using the hard STE [9]. Also, we consider a stochastic synapse as a regularizer and implement it with a gate module. The gate module consists of a shallow convolutional neural network, and it learns the probability with which certain weights in the main network should be masked. This is similar to DropConnect [113], but our model tries to *learn* which connections should be dropped depending on the current input. This means that training the network with the module in some senses involves learning to guess the best sub-network for each input and using such guesses during the testing time. This is related to other examples where the dropout probability is learned, such as adaptive dropout [7], variational dropout [57], and excitation dropout [125]. For example, adaptive dropout [7] jointly trains a neural network with a binary belief network that learns the probability, which depends on the input of masking the activation nodes of the neural network. Here, we also jointly train a main network together with a single or multiple gate modules, but in this case, we learn the probability for masking *weights* of the main network rather than activations, where the probability varies as a function of the input. To gain some understanding of our model, we visualize the activations of the gate module and the probabilities of the masking weights.

Wen et al. [115] discuss that stochastic weight methods such as DropConnect [113] have a serious drawback compared with the stochastic activation methods. Networks typically have more weights than units, and these stochastic weight methods are done typically with a single sample per mini-batch because of computational expenses. In our study, all samples per mini-batch are used to learn the probability of dropping the connection.

Our models are similar in parts to gated recurrent units (GRU) [16] and long short-term memory cells (LSTM) [47]. GRUs and LSTMs use gating to determine how much of the past information stored in the previous state should be preserved, and how much can be overwritten by the current input. These gates are implemented at the unit/node level. In our case, the gate module is used to learn the probability with which weights should be used (turned on or off) to classify a given example.

The binarized neural networks for adversarial examples show significant improvement compared with conventional neural networks. Our model, stochastic-gated partially binarized network (SGBN) will show that improvement and robustness against some adversarial examples, compared with other binarized neural networks: A stochastic binarized neural network, a stochastic activation pruning network, and a binary neural network. We set up five experiments; the first three experiments, FGSM, BIM, and ILL are on MNIST, the fourth one is FGSM on CIFAR-10. The last one is a black-box attack with FGSM on MNIST.

Materials and parts of results for these experiments have been published and presented at the International Joint Conference on Neural Networks (IJCNN 2019).

4.2 Proposed model

In this section, we introduce our network. We give an outline of the network before discussing the individual models in more detail.

4.2.1 Stochastic-gated partially binarized network

We use binarization in different parts of our network. The base network that we call CNN-BIN is shown in figure 4.1a. In this network, we use two convolutional layers followed by a dense layer and a softmax output layer. While our convolutional layers use conventional real-valued filters and ReLU activations, the following dense layer uses binarized neurons and is trained with the soft STE with the slope-annealing trick as described in Chapter 2, “Basic concepts”.

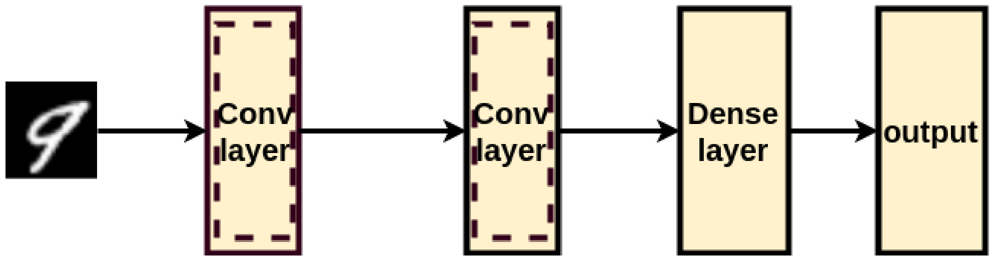
While binarizing the dense layer potentially improves resilience against adversarial attacks, we hope to further improve robustness by incorporating stochasticity in the preceding convolutional layers. We do this by adding a gating network modules alongside each convolutional layer, as shown in figure 4.1b. The same features that are input to the convolutional layer are also passed to the gate module. The output of the gate module, however, is used to modulate the stochastic activity of the convolutional filters, as described next.

4.2.2 Gate module

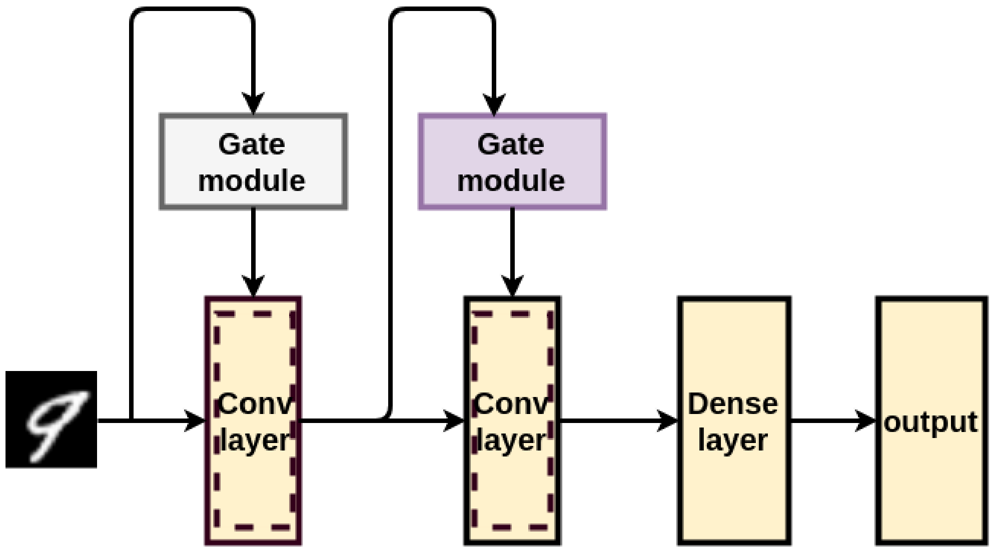
Figure 4.2 depicts the relationship between a gate module and its associated convolutional layer in more detail. The gate module is itself a shallow CNN, with a sigmoid output layer. These output values are then stochastically binarized (this stochastic part of the gate module is not differentiable. Thus, the hard STE is used for calculating the gradient. We found our system worked best when using soft STE with slope-annealing for the dense layer, while using hard STE for the output of the gate modules) and multiplied element-wise with the convolutional layer’s weights. Thus, the gate module looks at the image and uses this information to determine which filter weights should be active when classifying that particular input. That is:

$$\begin{aligned} h &= g(x), \\ w' &= w \odot h, \end{aligned} \tag{4.1}$$

where g is a gate module, and x is the input (both to the shallow CNN of the gate module and to its associated convolutional layer). The output of the gate module has the same shape as the filter weights of its associated convolutional layer in the



(a) The architecture of a 2 layer CNN-BIN. This dense layer uses stochastic binarized activations. The output is a softmax layer.



(b) A 2 layer SGBN framework. An image is used as input both for the convolutional layer and for the gate module on the first layer. The output from the gate module is used to modulate the stochastic activity of the convolutional layer's weights, as described in more detail in figure 4.2.

Figure 4.1: Two architecture: CNN-BIN (a) and SGBN (b).

main network, so the stochastically masked synapses are computed by the pointwise product of w and h. The masked weights w' are then convolved with the inputs of that layer:

$$o = f(x; w'), \tag{4.2}$$

where f is the convolutional layer and o is its output.

The stochastic synapses can be thought of as a learnable version of DropConnect, where the stochasticity is applied during both training and testing. There are other architectures that use gates, such as Wavenet [80], which is a network for generating raw audio wave and Sparsenet [103], which uses gate variables to learn the sparsity for

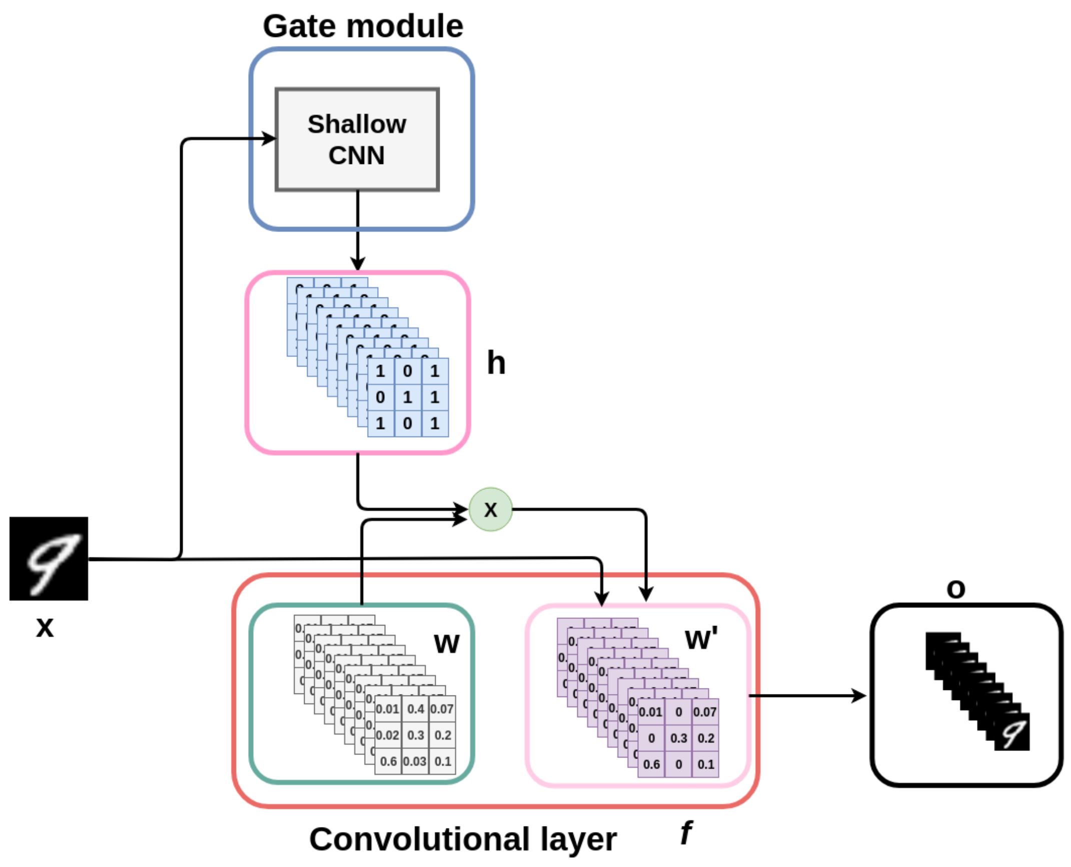


Figure 4.2: This illustrates the relationship between a gate module and a convolutional layer. The activations of the gate module h are converted stochastically into binary masks h (using the stochastic binarized activation (1 or 0)). For this stochasticity, the hard STE is used for calculating the gradient. These masks, in turn, are multiplied pointwise with the weights w of a convolutional layer f in the main network to produce new, masked filters w'. The convolutional layer f then use w' to convolve the image (or incoming feature maps) to output o.

the weights on each layer. However, they are unit/node-based gating (e.g. Wavenet), or the gating is not adaptable for each input (e.g. Sparsenet).

4.3 Experimental evaluation

We conduct five experiments to compare the performance of several different stochastic and/or binary models. In the first three, we use the MNIST dataset and apply three different techniques to generate the adversarial examples: FGSM, BIM, and ILL. In the fourth experiment, we use the CIFAR-10 dataset and test the networks against an FGSM-based adversarial attack. In the last experiment, we use the MNIST dataset and FGSM to check if adversarial examples transfer between models.

4.3.1 Network parameters

For the first three experiments (FGSM, BIM, ILL on MNIST) our model's overall architecture is comprised of two parts: the *main network*, and the *gate modules*:
(1) The *main network* consists of two convolutional layers, one dense layer, and softmax output layer. The two convolutional layers have 32 and 64 filters, respectively, with rectified linear unit activation. All filter sizes are 5×5. After the convolutional layers, there are a 100-node dense layer and a 10-node softmax output layer.
(2) Each of the *gate modules* consists of a shallow CNN. Each shallow CNN consists of one convolutional layer and three dense layers. The details of the two gate modules are given in tables 4.1 and 4.2.

We compared the SGBN's performance against 4 models. The first one consists of two conventional convolutional layers followed by one dense layer with the rectified linear unit activation and a softmax output layer. In the second one, we changed the dense layer's activation to the stochastic binarized activation (1 or 0) (CNN-BIN, figure 4.1a). In the third one, we applied stochastic activation pruning method [21] to the convolutional layers of the CNN-BIN model (SAP). Finally, the fourth model consists of one conventional convolutional layer and one convolutional layer with binarized weight (1 or -1) followed by a dense layer with binarized (1 or -1) weight and the softmax output layer (BNN) with PGD.

All of the networks for the three experiments are configured the same as SGBN (32 and 64 filters whose size is 5×5 for the 1st and 2nd layer respectively, and 100 hidden nodes and finally a 10-node softmax output layer). For the slope-annealing techniques, we increased the slope m with the schedule $m = min(5, 1 + 0.04 * N_{epoch})$ where N_{epoch} means the number of epochs. As our model contains more parameters than the other models because of the gate module, in the discussion section (Section 4.7) we also describe results when re-training the other models with a comparable number of total parameters.

For PGD (only for BNN), the iteration size is set to 40, and the constant weight τ is 0.5. The mini-batch and epoch sizes for all of the models are 50 and 25 with early stopping to avoid overfitting, respectively. These three experiments are conducted on the MNIST dataset [65].

For FGSM, we checked the results on a perturbation level ϵ from 0.01 to 0.3, and for each model, the accuracies are calculated using the entire perturbed testing dataset of 10,000 samples. For BIM and ILL, the number of iterations (T) in our case is 10. We tested the small step size (perturbation level ϵ) from 0.01 to 0.1. After 10 iterations, we clipped the color channel values on perturbed images between 0 and 1.

For the fourth experiment (FGSM on CIFAR-10), we changed the parameters of the networks to accommodate the more complex nature of the images. SGBN consists of 64, 128, and 128 filters whose sizes are 8×8, 6×6, and 5×5 for the 1st, 2nd, and 3rd layers, respectively (the same filter sizes are used for each CNN on the gate modules, with other parameters and activation functions for the gate modules the same as for the other three experiments, except the hidden layer node sizes of dense layers in the gate modules are 512, 512, and each filter size of 2nd and 3rd convolutional layers of the main network), and followed by one dense layer [1]. Also, BNN, SAP, and conventional CNN consist of the same filter and dense layer sizes as SGBN's main network. In this experiment, the 1st layers of all the models are full precision. For PGD (only for BNN), the iteration size is set to 20, and the constant weight τ is 0.5. Furthermore, the mini-batch and epoch sizes for all of the models are 128 and 40 with early stopping to avoid overfitting, respectively. This experiment is conducted on the CIFAR-10 dataset [58]. It consists of 60000 32×32 RGB natural images for 10 classes, with 6000 images per class. These are separated into 50000 for training and 10000 for testing dataset. The pixel values of the images are also normalized to real numbers in the range [0,1] for each color channel in our experiment.

For the last experiment (black-box attack on FGSM), we used the same parameters as the MNIST experiment. We computed adversarial examples on one model and transferred these examples to the other models. This is tested on the highest perturbation level of $\epsilon = 0.3$.

For all of our experiments, we use the Adam optimizer [56] and batch normalization [51] for the convolutional layers (including those in the gate module). The learning rates for all of the models are 1e-3. PGD for BNN is applied for the last 5 epochs. All of the accuracies are averaged over 10 runs for the models trained from

[1]This architecture comes from [34]

scratch. All of the source code for the adversarial attacks are based on Cleverhans [81]. For BNN with PGD implementation, we used the author's source code on [32].

Table 4.1: First gate module parameters on MNIST

Layer	Size	activation	Note
conv	32×5×5	relu	
dense	512	relu + batch normalization	
dense	512	asinh	
dense	800	binarized	reshaped to 5×5×32×1

Table 4.2: Second gate module parameters on MNIST

Layer	Size	activation	Note
conv	64×5×5	relu	
dense	512	relu + batch normalization	
dense	512	tanh	
dense	51200	binarized	reshaped to 5×5×64×32

Table 4.3: First gate module parameters on CIFAR

Layer	Size	activation	Note
conv	128×6×6	relu	
dense	512	relu + batch normalization	
dense	512	asinh	
dense	6912	binarized	reshaped to 6×6×128×64

4.4 Results

Figure 4.3 shows the results for FGSM on the MNIST testing dataset. We can see that the accuracy gets better when binarizations are added to the network. Note that even for these relatively small networks, adding binarization (i.e. from CNN to CNN-BIN) improves robustness, and adding stochastic activation pruning on top of that adds further robustness. In these particular experiments, weight and activation binarizations for both convolutional and dense layers (i.e. BNNs) helped get results comparable to SAP. Adding stochastic masking (i.e. from CNN-BIN to SGBN) appears to further help robustness against this white-box attacks on this dataset. In

Table 4.4: Second gate module parameters on CIFAR

Layer	Size	activation	Note
conv	128×5×5	relu	
dense	512	relu + batch normalization	
dense	512	tanh	
dense	409600	binarized	reshaped to 5×5×128×128

particular, SGBN achieves an accuracy of 93.1±1.2% at $\epsilon = 0.3$ and shows clear improvements over other models.

Figures 4.4a and 4.4b show the accuracies of each network on BIM and ILL. In these methods, for BNN, stochastic and deterministic quantized weights at the 2nd layer are applied during testing since Galloway et al. [34] suggest that the stochastic layers improve the performance at test time and can be a possible defense against iterative attacks. This stochastic binarization is explained in [34]. The accuracy of this small CNN at $\epsilon = 0.02$ is already close to 0 on BIM. Compared with the CNN, both deterministic and stochastic BNNs and CNN-BIN accuracies are better, but at a moderate perturbation level of 0.06, the accuracies are less than 10%: 6.3±1.8, 4.6±0.7, and 7.9±1.0% for deterministic BNN, stochastic BNN, and CNN-BIN respectively. SAP applied to the CNN-BIN is resistant at the lower perturbation level but not at higher perturbation levels. SGBN seems relatively robust against strong iterative attacks on this dataset even at the highest perturbation level (77.7±5.5% at $\epsilon = 0.1$). On both ILL and BIM, the performances for the stochastic binarization networks are better than those of deterministic networks.

Results for FGSM on CIFAR-10 are in figure 4.5. Our relatively small CNN is naturally most vulnerable to these examples. Even though SAP helps robustness on MNIST, it still leaves this particular network vulnerable on CIFAR-10. SGBN achieves 17.3±0.8% on the highest perturbation level. BNN with PGD achieves the accuracy of 13.6±1.2%. These experiments need to be interpreted with caution because it is not clear how these results scale as the networks become larger. For example, when SAP is applied to a deep ResNet architecture as shown in Dhillon et al [21], it achieved an accuracy of 80% at a perturbation level of roughly 0.015.

Black-box results for FGSM on MNIST are in figure 4.6. The white-box attacks are also shown when the sources and targets are the same. Among the models, SGBN

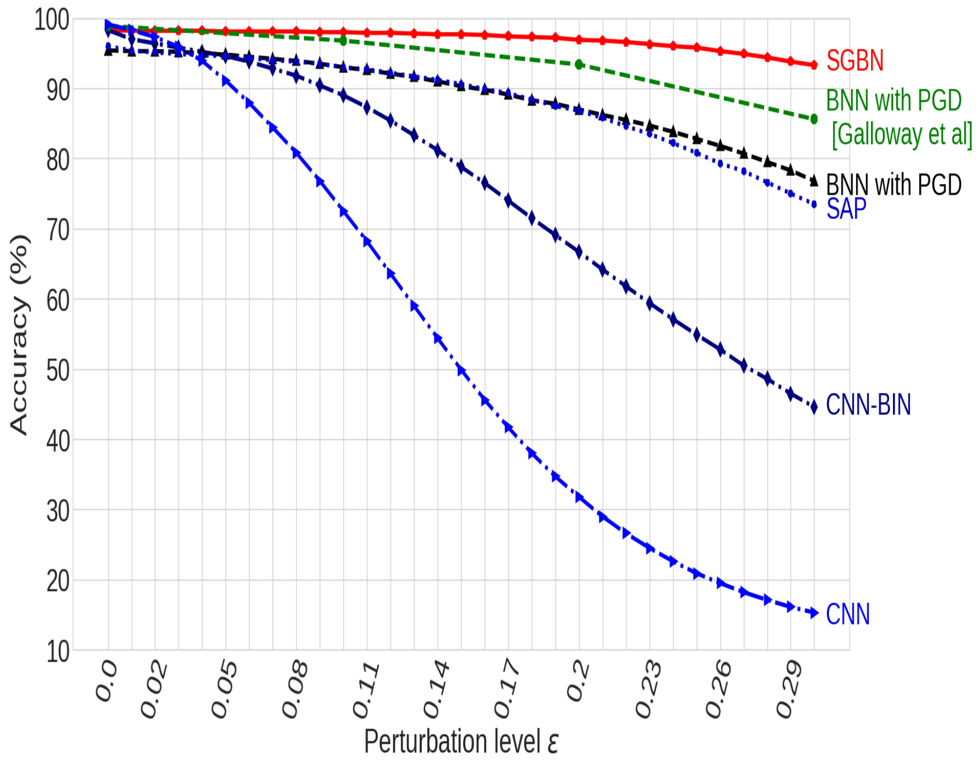
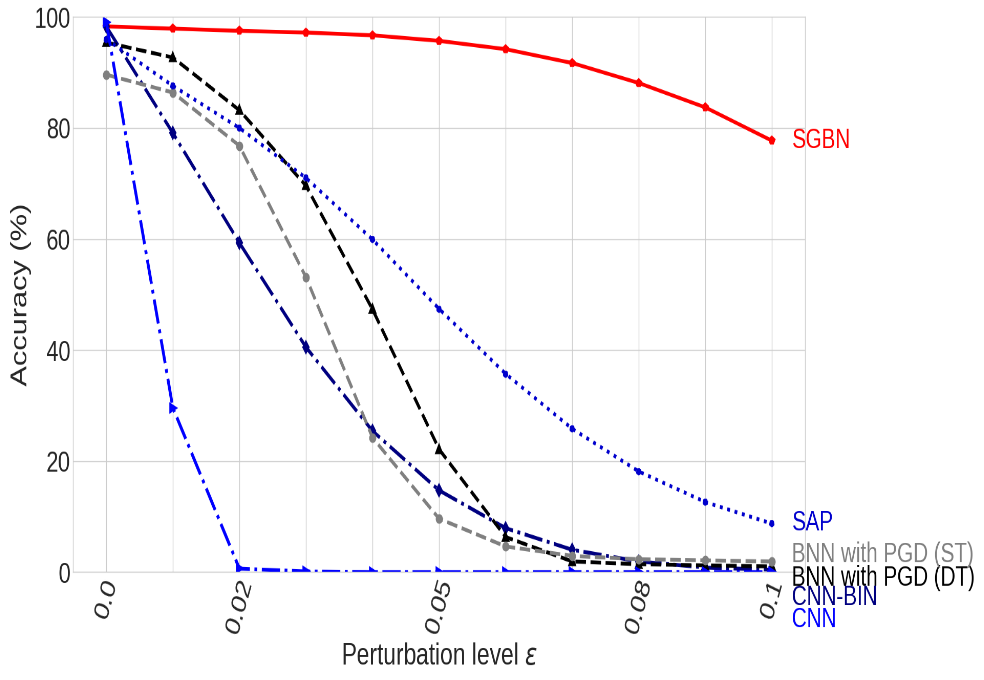
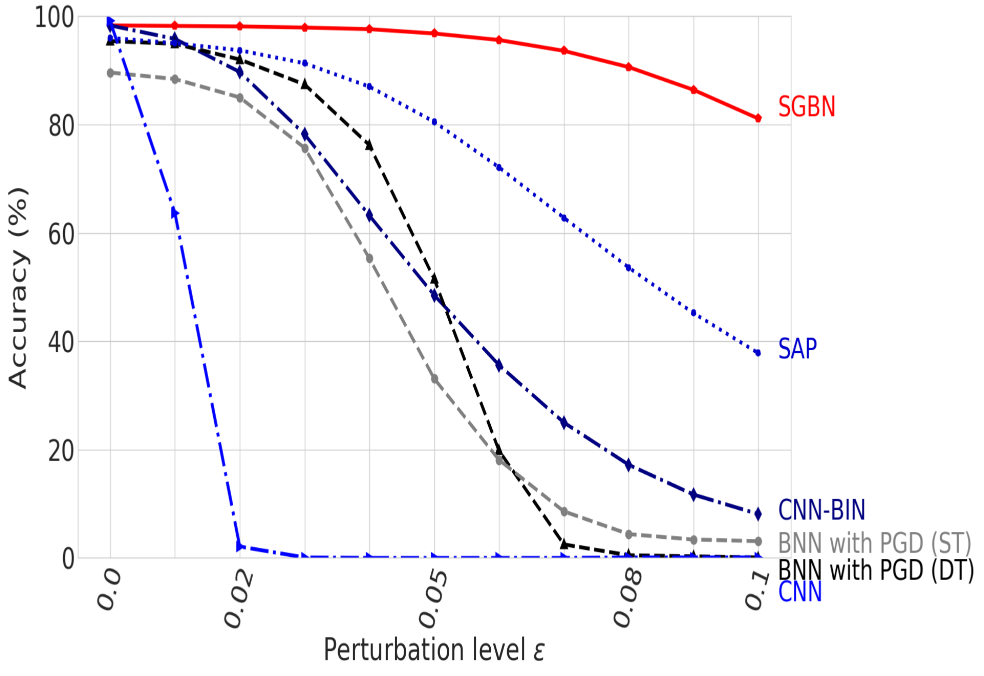


Figure 4.3: The accuracies of six models on FGSM for MNIST. Also, the best accuracies of BNN with PGD [34] are plotted on the perturbation level $\epsilon = 0.1$, 0.2, 0.3. Another model, BNN, is the 2 convolutional layer version of BNN (scaled, the first layer is a full precision, and the second layer is binarized) and binarized dense layers without PGD. The other model, SAP, is 2 layer convolutional with stochastic activation pruning (SAP) and dense layers (with stochastic binarized activation) [21] without any adversarial training.



(a) The accuracies on BIM.



(b) The accuracies on ILL.

Figure 4.4: (a): The accuracies of SGBN, CNN-BIN, BNNs (deterministic (DT) and stochastic (ST)), and SAP on the basic iterative method (BIM) for MNIST. (b): The accuracies of SGBN, CNN-BIN, BNN (DT and ST), and SAP on the iterative least-likely class method (ILL) for the MNIST. Stochasticity (in weights, activation, or both) seems to be particularly helpful against the ILL-based examples.

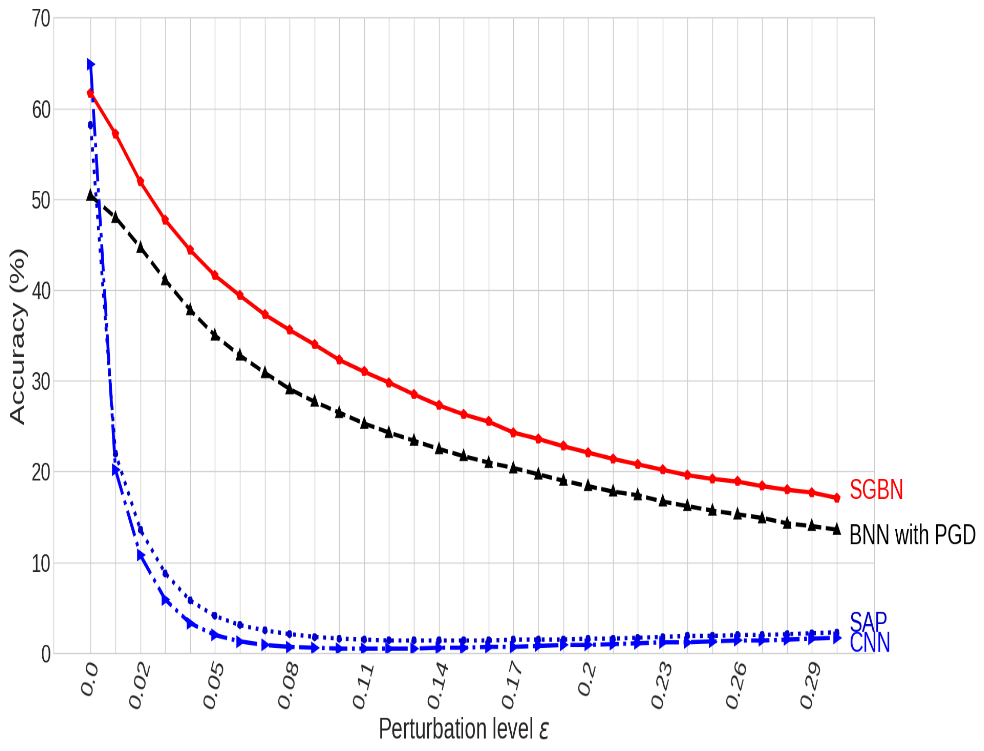


Figure 4.5: The accuracies of four models on FGSM for the CIFAR-10. Note that some approaches, such as SAP, for example, were originally applied on much larger networks than the ones used for testing in this experiment. The accuracies here are 1.7±0.4, 2.3±0.3, 13.6±1.2, and 17.3 % at the highest perturbation level for CNN, SAP, BNN, and SGBN, respectively.)

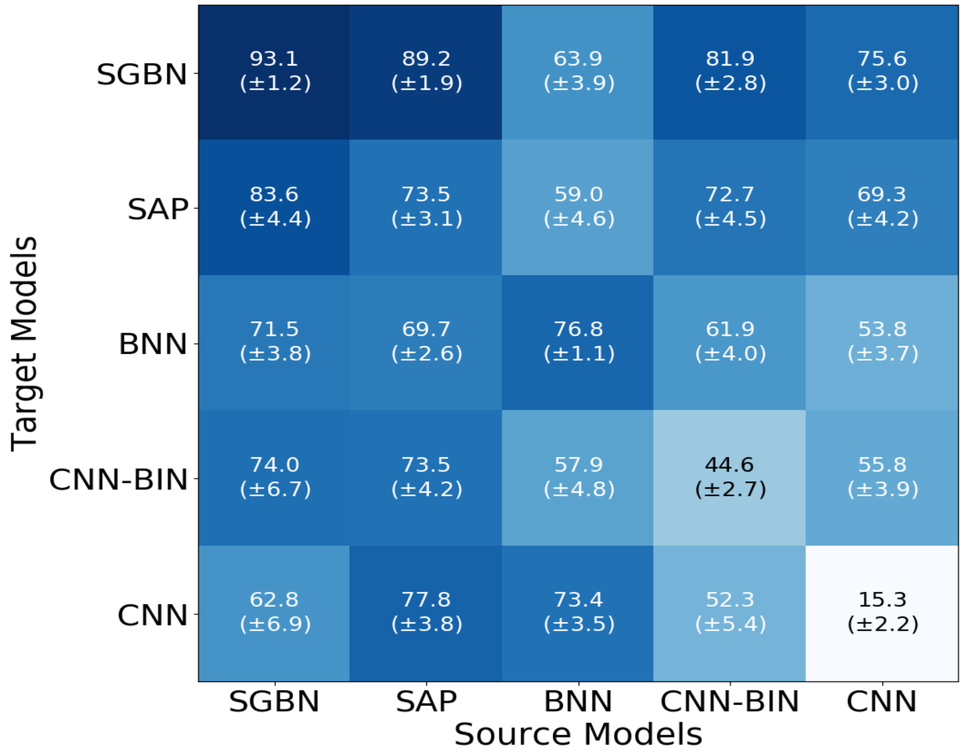


Figure 4.6: The accuracies of FGSM ($\epsilon = 0.3$) transferred between models.

is the most resilient with the accuracies from 63.9% to 93.1%, while CNN is the most vulnerable to the attacks with the accuracies from 15.3% to 77.8%. Furthermore, SGBN creates the weakest perturbations among the models. The perturbations of SGBN are easy to classify not only for SGBN but also for the other models even the undefended model of CNN (average of the accuracies over all of the targets for SGBN is 77.0% compared with 76.7%, 66.2%, 62.7%, and 54.0% for SAP, BNN, BIN, CNN, respectively).

4.5 Visualization of SGBN

4.5.1 Visualization of activations of the gate module and main network

How does the masking of a binarized synapse layer change as the input changes? In this section, we visualize activations of the gate module and associated synapse layer using the clean MNIST dataset. Figure 4.7 shows activations of the 1st layer's gate module along with the corresponding input image after gated CNN filters have been applied. The gate module turned on or off the specific weights of the main network.

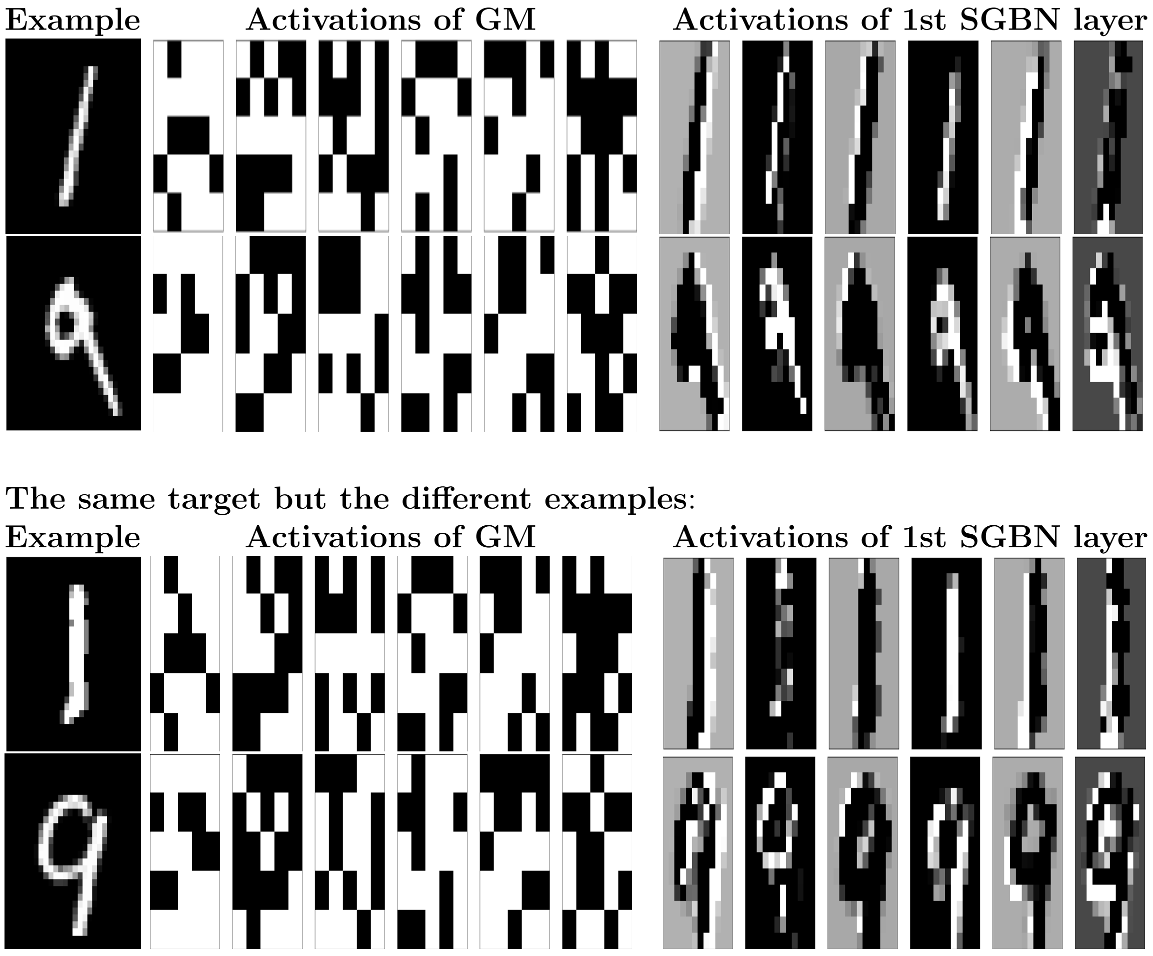


Figure 4.7: Examples of activations of the gate modules on the 1st layer of SGBN. In the first column, there are images for inputs for the networks. The 2nd through 7th column are outputs from the gate modules on the clean MNIST (zeros are in black, and ones are in white). The rest of the columns are activations of the 1st layer of the SGBN. Upper: We picked random 5 filters, in this case, filter number 7 to 12 of 32 filters. Bottom: We also picked the same filters as the previous one. In this case, we feed the same target but different examples to the model. Some of the activations are similar to the previous ones, but the rest of the activations have completely changed.

Thus, this gating clearly depends on the input image. The activations of the gate module are indeed adaptive and change in response to changes in the input image. We also visualize the activation of the same target, but different examples to check if the activation does or does not change on the examples. The bottom panels of figure 4.7 show the results. When the examples for the same target is fed into the model, it changes the activation of the gate module to mask the main network.

Furthermore, figure 4.8 visualizes the activations ($\sigma(a)$) of the gate module (the 1st layer) before binarized (probabilities of masking) on the testing dataset. The gate module masks specific weight depending on images. Around 60 % of the weights in the filter are turned on, and around 40 % of the weights are turned off (figure 4.9). In addition, we plot the histogram of probabilities of activations ($\sigma(a)$) from the gate module for the second layer before they are binarized in Appendix B. Even though the gate module uses the stochastic binarized activation function, the activations change depending on each image in a nearly deterministic way.

The changing of the weights depending on each input may contribute to the ability of these models to resist adversarial examples in addition to the gradient masking, even at the high perturbation levels.

4.5.2 Visualization of adversarial examples and perturbation measurement

We visualize and measure distance metrics of adversarial examples for each model on a testing dataset of MNIST to check how perturbations are added to images. We measure the distance, ℓ_1 between clean and perturbed images which are crafted by each model. The measurements of the distance metrics are averaged over 10 runs for each model trained from scratch. We also use one of the dimension reduction algorithms, t-SNE [69] to compare the clean and perturbed images of SGBN on the data representations.

First, we chose a clean image, perturbation, and perturbed image on FGSM ($\epsilon = 0.3$) for each model as seen in figure 4.10. The perturbations for CNN and CNN-BIN are spread over the entire image except for the right and bottom sides. Compared with these models, the perturbations for BNN with PGD and SAP are very unique. The shapes of these perturbations are similar to the shape of the target, 4. For

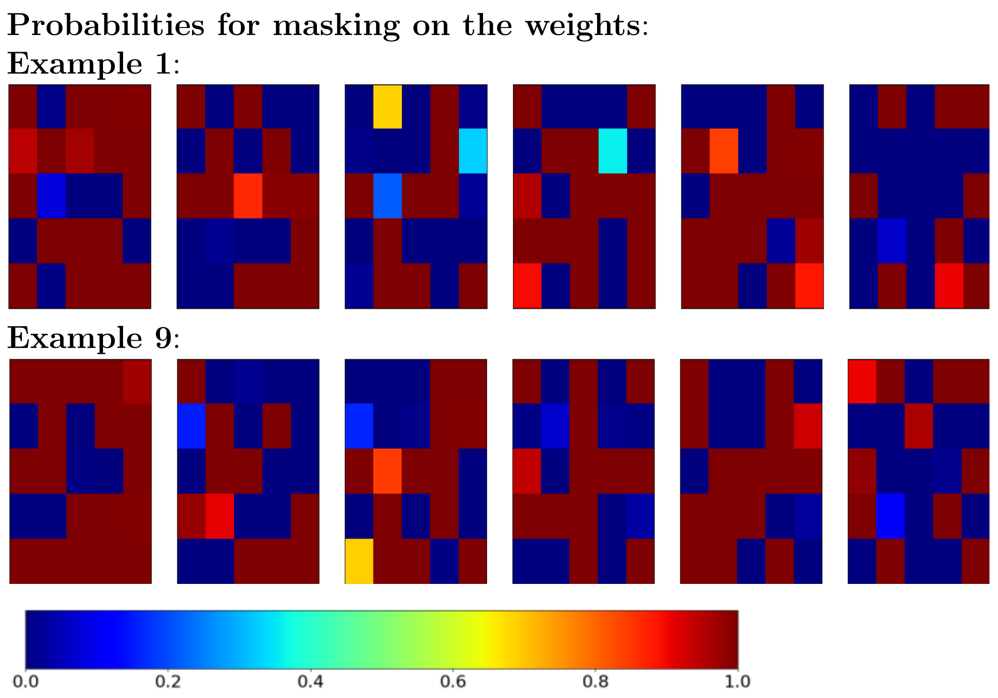


Figure 4.8: Examples of probabilities of activations ($\sigma(a)$) from the gate module (1st layer) before they are binarized (scale is 0 to 1, and the colors are blue to red). Examples are 1 and 9 from upper panels of figure 4.7.

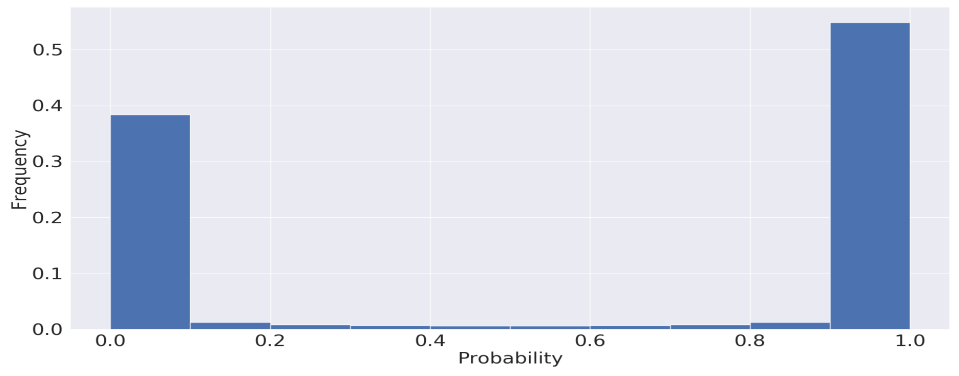


Figure 4.9: Histogram of probabilities of activations ($\sigma(a)$) from the gate module (1st layer) before they are binarized (scale is 0 to 1). Around 60% of the activations are close to 1, and around 40 % are close to 0.

SGBN, the perturbations are spread over the entire image. We can see this difference between clean and perturbed images on the ℓ_1 distortion measure in figure 4.11.

In figure 4.11, all of the models had lower ℓ_1 distances at lower perturbation levels, but SGBN, CNN, and CNN-BIN ℓ_1 distances kept increasing steadily. On the other hand, SAP and BNN ℓ_1 distances did not increase as steadily for these models.

Finally, at ϵ=0.3, the distortion for SGBN was the highest among the models. On the other hand, the distortions for SAP and BNN with PGD were very stable even as the perturbation level increased. There was not a significant difference between CNN and CNN-BIN on the ℓ_1, although the accuracies of these models on the FGSM were very different.

Figure 4.12b shows the representations of the perturbed dataset using t-SNE for SGBN. The representations tell us that the digits were still clustered well even though the distortion was very high for the perturbed images at $\epsilon = 0.3$ compared with the representations of the clean dataset (Figure 4.12a). Furthermore, even though the distortion for BNN with PGD was the lowest, the digits were not clustered well.

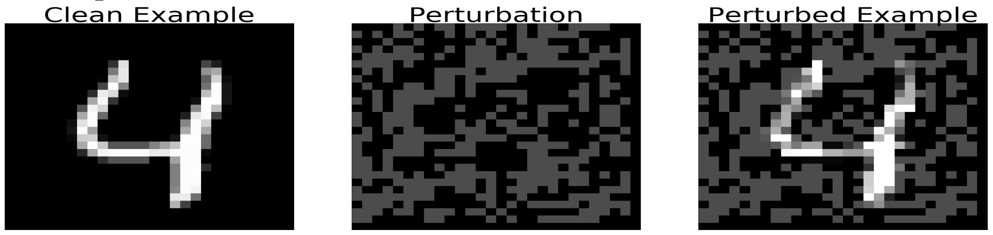
4.6 Gaining an understanding of SGBN and the gate module

In this section, to gain an understanding of SGBN and the gate module, we conduct several experiments. We start by comparing a conventional SGBN and one has a deterministic non-binarized activation function in the gate modules to discover how much stochasticity and binarization of the gate modules affect the results of white-box attacks. After that, we retrain the gate module on the clean MNIST to check if the gate module learns the proper representations to classify the images. We then check whether or not SGBN needs double-sized filters, and why SGBN is more robust against black-box attacks than the other gradient masking models. Finally, we conduct a simple toy experiment to gain an understanding of how changeable the gate module activations are. All of the experiments are conducted on the MNIST. All of the parameter and iteration sizes are the same as in the previous experiment on white-box attacks for FGSM on MNIST. When we set additional parameters, we will specifically mention them in each subsection.

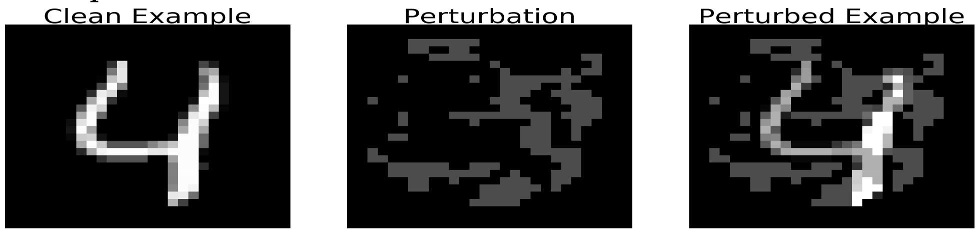
4.6.1 Stochastic binarized versus deterministic non-binarized gate modules

We have already seen that SGBN is robust against adversarial examples. Here, we would like to check how much stochasticity and binarization of the gate modules affect its performance on a white-box attack. The stochasticity and binarization are gradient masking approaches. Since gradient masking models do not have useful

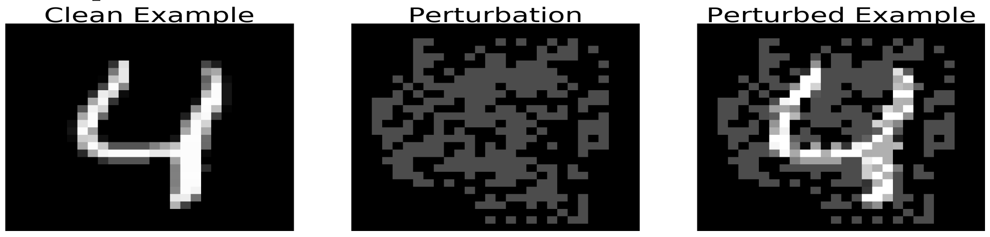
Example 4 on SGBN :



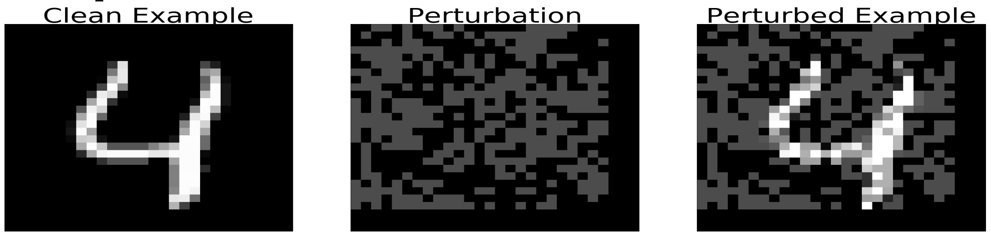
Example 4 on BNN with PGD:



Example 4 on SAP:



Example 4 on CNN-BIN:



Example 4 on CNN:

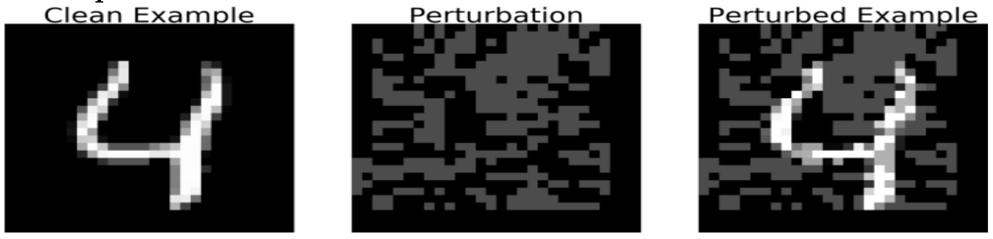


Figure 4.10: Examples of clean examples, perturbation, and perturbed examples on FGSM ($\epsilon = 0.3$) for SGBN, BNN with PGD, SAP, CNN-BIN, and CNN.

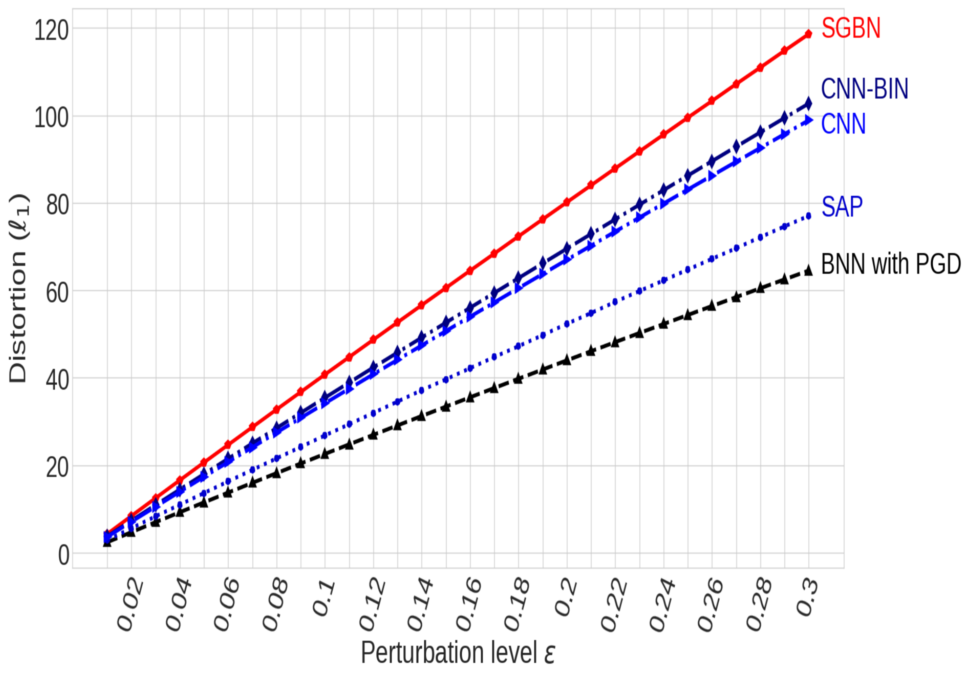
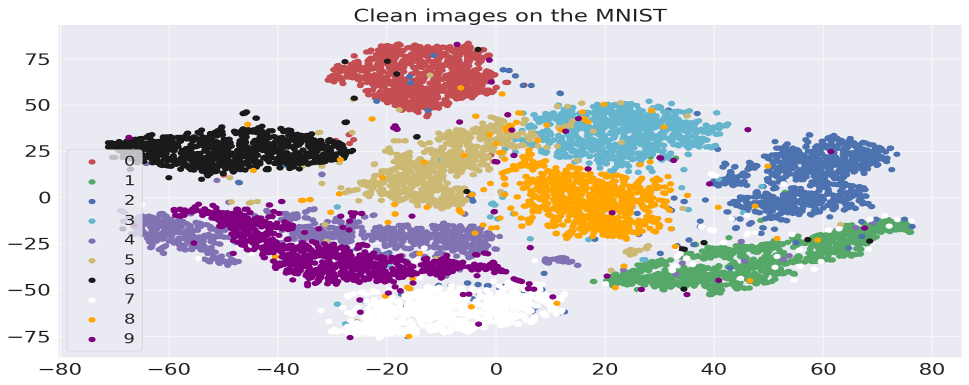


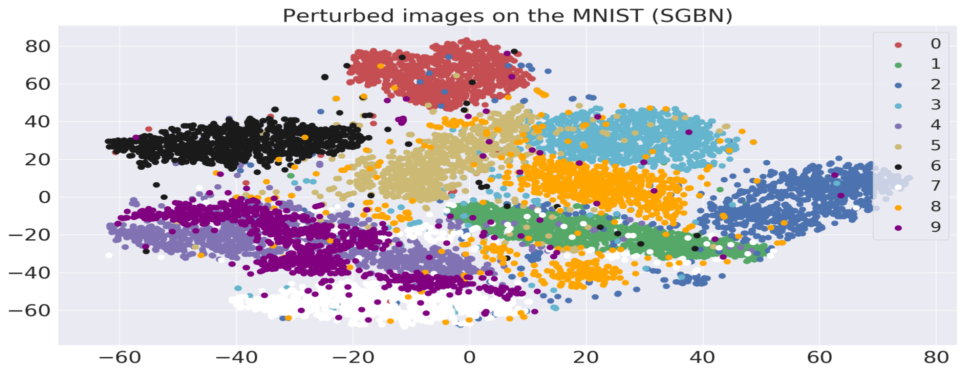
Figure 4.11: ℓ_1 distance between the clean and perturbed images on each perturbation level for each model.

gradients to craft adversarial attacks, we hypothesize that these gradient masking approaches create weak perturbations which can be classified easily even by an undefended model. To explain the weakness of these perturbations, we conduct two experiments. First, we compare the performances of SGBN (SGBN (ST)) and SGBN with a deterministic non-binarized activation in the gate modules (SGBN (DT)) on a white-box attack. Second, we compare the performances of other models than SGBNs on adversarial examples crafted by SGBN (DT) compared with SGBN (ST) to see if the other models can easily classify the adversarial examples. Additionally, we check the overfitting rates for SGBN (ST), SGBN (DT), and CNN-BIN to see if both stochastic binarized and deterministic non-binarized versions of the gate module can be regularizers. Furthermore, we check the gradients of the cost function with respect to inputs ($\nabla_x \mathrm{Loss}(\mathrm{w}; \mathrm{x}, \mathrm{y})$) for each model to see if both binarized and deterministic non-binarized versions of the gate module affect the gradients.

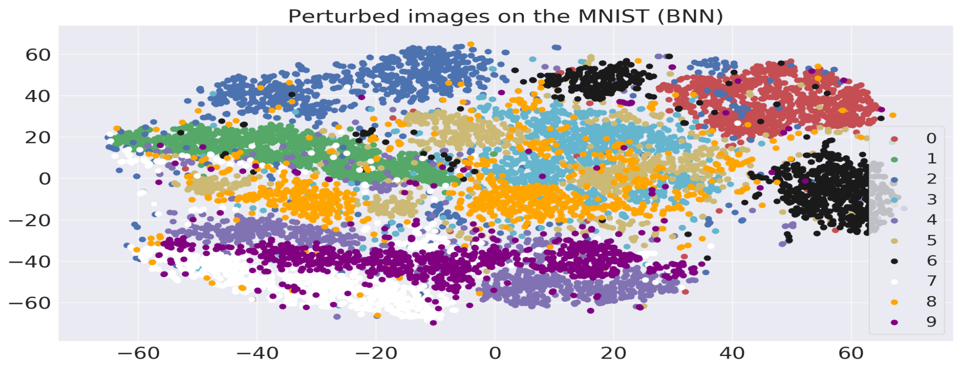
For SGBN (DT), we replaced the stochastic binarized activation function for the gate modules with the logistic function. In this case, the synapses w' were computed from the pointwise product of w and h, which were the outputs of the logistic function



(a) The clean 10000 images of the MNIST testing dataset are reduced to two dimensions by t-SNE.



(b) The perturbed 10000 images (FGSM with $\epsilon = 0.3$) of the MNIST testing dataset for SGBN reduced to two dimensions by t-SNE.



(c) The perturbed 10000 images (FGSM with $\epsilon = 0.3$) of the MNIST testing dataset for BNN with PGD reduced to two dimensions by t-SNE.

Figure 4.12: Comparing two t-SNE results: clean dataset (a), adversarial examples crafted by SGBN (b) and BNN with PGD (c). We can see that adversarial examples crafted by BNN with PGD changed a lot compared to adversarial examples crafted by SGBN.

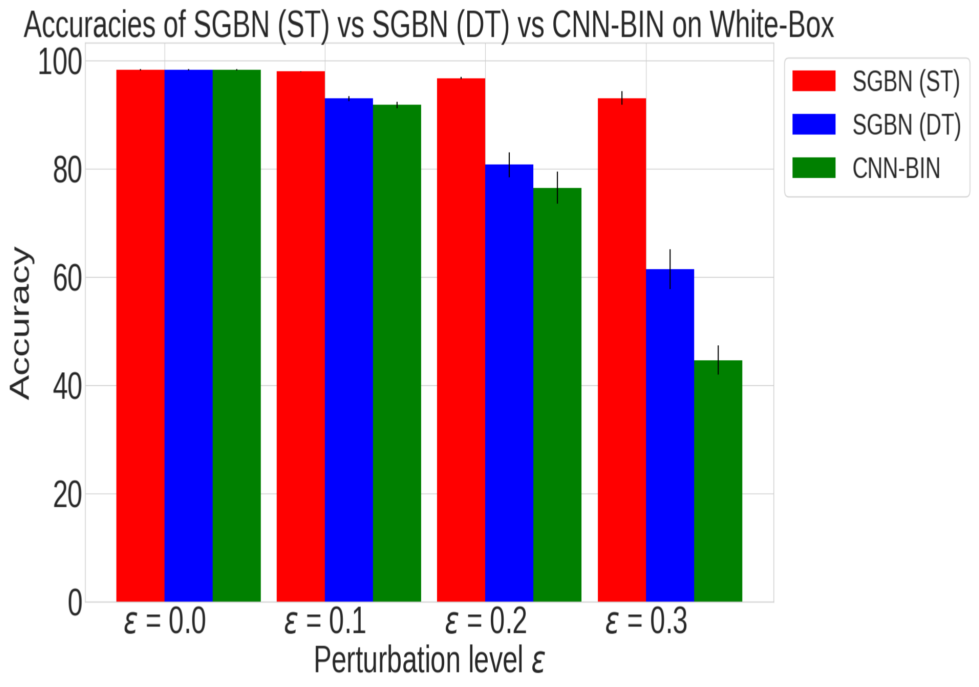


Figure 4.13: The accuracies of SGBN (ST), the deterministic non-binarized version of SGBN (DT), and CNN-BIN on the white-box attack.

for the gate module (without stochastic binarization). The model was trained on the MNIST from scratch.

First, we checked the accuracies of the stochastic binarized and deterministic non-binarized version of SGBN on a white-box attack. Figure 4.13 shows the accuracies of SGBN (ST), SGBN (DT), and CNN-BIN as a baseline on the white-box attack of FGSM. Although there is not much difference between CNN-BIN and SGBN (DT) on at $\epsilon = 0.0$, 0.1, and 0.2, SGBN (DT) achieves an accuracy 61.5±3.7% at $\epsilon = 0.3$ compared with CNN-BIN (44.6 ±2.7%), but SGBN (ST) is more robust against FGSM than SGBN (DT). On the clean dataset, we cannot see much difference among the models. Furthermore, figure 4.14 shows the differences in accuracy between the training and testing datasets to check the overfitting levels for each network. The difference in accuracy for SGBN (ST) is smaller than SGBN (DT) and CNN-BIN, and SGBN (ST) shows better generalization behaviors than SGBN (DT) and CNN-BIN. Even though the architectures of SGBN (ST) and SGBN (DT) are very similar, SGBN (DT) is weaker against the white-box attack than SGBN (ST). Of course, the gate modules in SGBN (DT) are differentiable, so they do not have a gradient

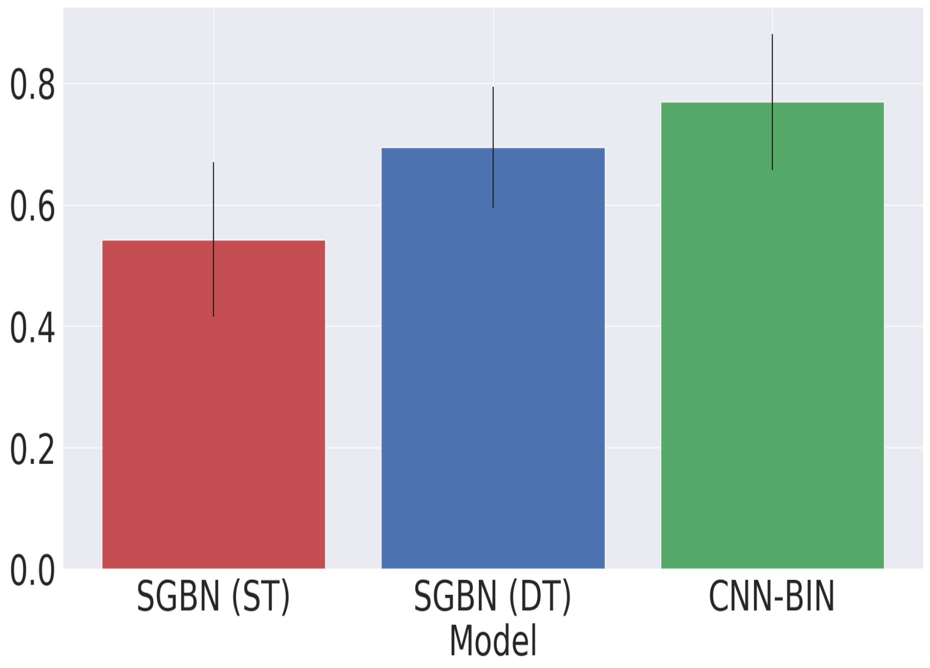


Figure 4.14: The differences in accuracy between the training and testing datasets (clean datasets) for SGBN (ST), the deterministic version of SGBN (DT), and CNN-BIN.

masking approach.

We also checked how easy it was to classify adversarial examples crafted by SGBN (ST) compared with SGBN (DT). We applied the examples to SAP, BNN, CNN-BIN, and CNN.

The accuracies for each model against adversarial examples crafted by SGBN (ST) and SGBN (DT) are shown in figure 4.15. Obviously, adversarial examples crafted by SGBN (ST) are easier than the examples crafted by SGBN (DT). Specifically, SGBN (ST)'s examples are at least 8% easier than SGBN (DT) (stochastic binarized: 62.8% and deterministic non-binarized: 54.1% on CNN).

We plot histograms for the gradients of the cost function with respect to inputs as seen in figures 4.16. The gradient variance for SGBN (ST) is larger than SGBN (DT) and CNN-BIN. This means even if the perturbation level ϵ is 0.3 (the highest), the level might not be high enough to flip an example to another class. For the larger variances of the gradients, larger changes in an input are needed to flip the input to another class. This may be one of the reasons why SGBN (ST) cannot create strong perturbations. Furthermore, the reason why the gradient variance for SGBN

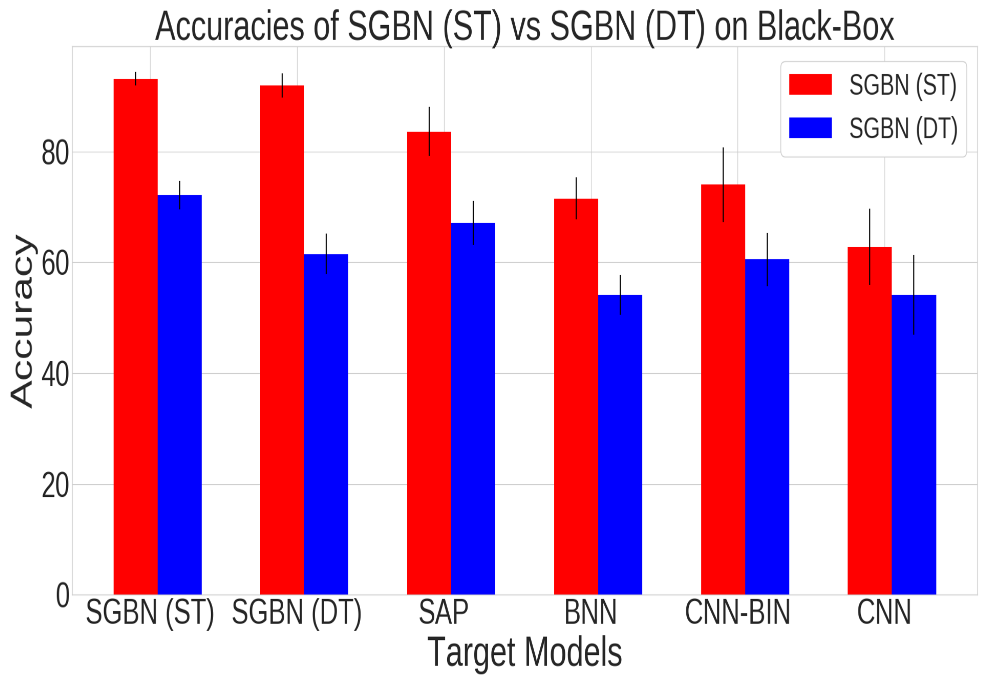
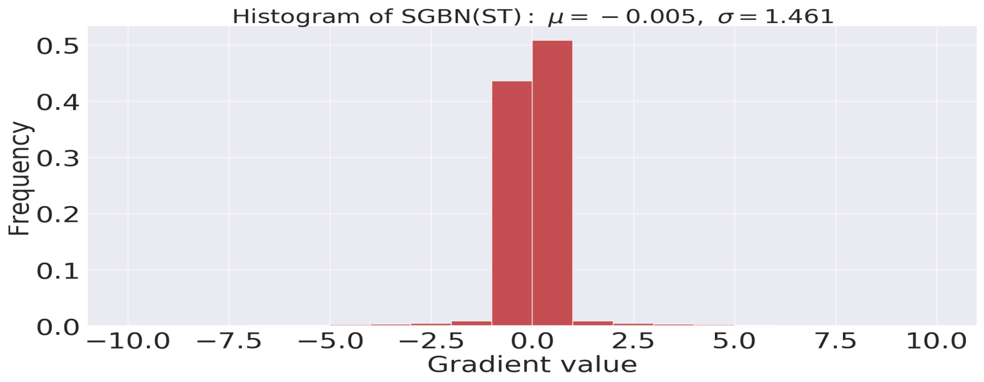


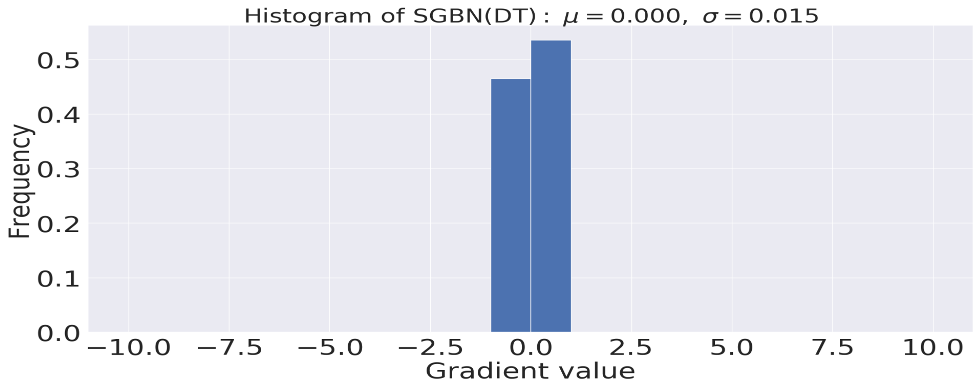
Figure 4.15: The accuracies of each model on FGSM attack crafted by SGBN (ST) and SGBN (DT). When target and source models are the same, it is the white-box attack.

(ST) is larger is the gate module uses hard STE, which returns 1 for the gradients of the stochastic binarized activations in the gate module. This gradient of 1 might make the gradients larger than the other models. Wang et al. [114] also discuss that the larger variances of the gradients are more difficult for the attacker to generate effective adversarial examples.

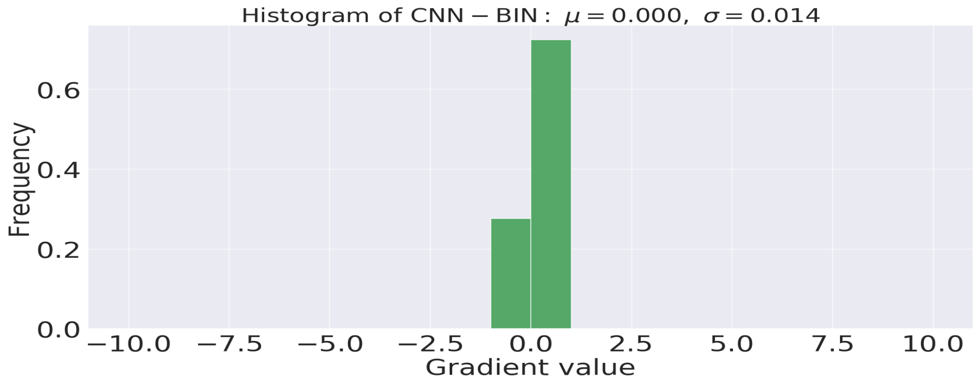
Both the stochastic binarized and deterministic non-binarized gate modules, especially the stochastic binarized gate modules help not only to improve generalization of the networks but also to protect the networks against adversarial examples. The stochastic binarized model is more robust against the white-box attack, but the deterministic non-binarized model might be useful for more sensitive tasks such as a toy experiment as in subsection 4.6.5. The stochasticity and binarization *do* help the robustness of SGBN against the white-box attack.



(a) Histogram of the gradients of the cost function with respect to inputs for SGBN (ST).



(b) Histogram of the gradients of the cost function with respect to inputs for SGBN (DT).



(c) Histogram of the gradients of the cost function with respect to inputs for CNN-BIN.

Figure 4.16: The gradient variance for SGBN (a) are larger than SGBN (DT) (b) and CNN-BIN (c).

Table 4.5: Training a dense layer setting for the first gate module

Parameters	dense layer
learning rate	0.001
Iteration size	5000
Optimization function	Adam optimizer

Table 4.6: Training a dense layer setting for the second gate module

Parameters	dense layer
learning rate	0.0001
Iteration size	5000
Optimization function	Adam optimizer

4.6.2 Learning representation and attacking the gate modules

The higher accuracy of SGBN during white- and black-box attacks compared with the other models is almost certainly caused by adding the gate module(s) to the convolutional layer(s). We have already seen that the stochasticity and binarization of the gate module help to defend SGBN during attacks. However, we have not discovered yet what the gate module has learned. We hypothesize that the gate module learns the proper representations to classify the images, and it decides which weights should be turned on or off to improve the SGBN accuracy even on adversarial attacks. Furthermore, we would like to examine if each gate module is still resilient to a white-box attack. To test these hypotheses, we designed a simple experiment.

In this experiment, we added one dense output layer to each of the gate modules. The dense layers were trained on the MNIST dataset with all of the parameters other than the dense layers were frozen. Figure 4.17 illustrates which part of the SGBN parameters were trained and frozen (the first gate module with the dense layer: 1st-GM, and the second gate module with the dense layer: 2nd-GM). After training the dense layers, the accuracies of each model on a clean dataset and FGSM were examined. FGSM was crafted by each gate module with the dense layer. The parameter settings for this experiment are in tables 4.5, 4.6, and 4.7.

The results of this experiment are shown in figure 4.18. In this figure, we also add the accuracies of the original (2-layer) version of SGBN (2-layer-SGBN) and a 1 layer version of SGBN (one convolutional layer with one gate module, 1-layer-SGBN) to the results in order to check how these SGBN performances are related to the gate

Table 4.7: 1-layer-SGBN parameters

Layer	Size	activation
conv	32×5×5	relu
dense	100	the stochastic binarized activation
dense	10	softmax

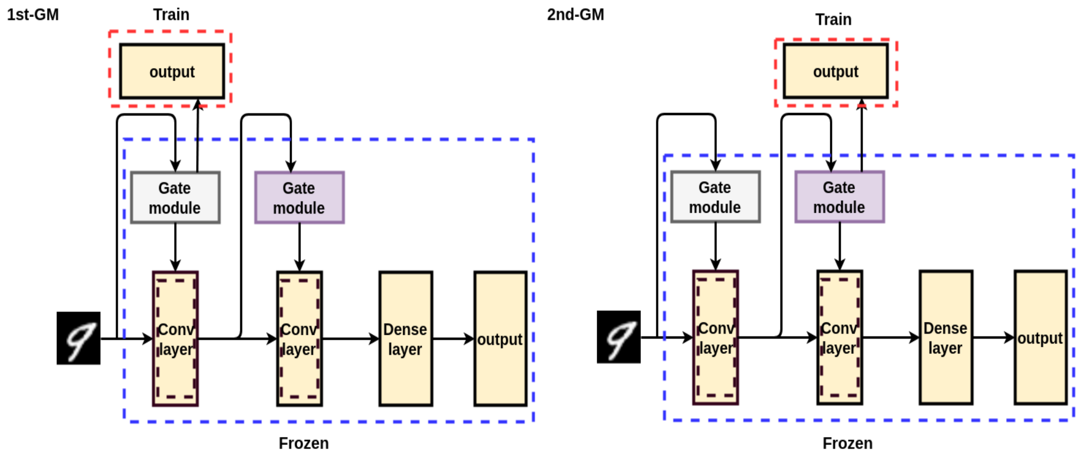


Figure 4.17: Left: Training the dense layer which is added on the first gate module. Right: Training the dense layer which is added on the second gate module.

modules' performances. The figure shows that the curve pattern of 2-layer-SGBN was very similar to 2nd-GM while the curve pattern of 1-layer-SGBN was similar to 1st-GM. Furthermore, 1st-GM and 2nd-GM performances on the clean testing dataset were 84.2% and 95.8%, respectively. These results clearly illustrate that both the gate modules learned how to classify the images. Moreover, the models with the two gate modules are more consistently accurate than the ones with the one gate module.

Adversarial examples crafted by the gate modules

The gate modules create weak perturbations because stochasticity and binarization are gradient masking approaches. We visualize the perturbations crafted by 1st-GM and 2nd-GM in Appendix D. These are very similar to the perturbations crafted by SGBN. Because of these similarities, we test whether these examples can still be easily classified by an undefended model, conventional CNN. If CNN can achieve a similar accuracy to SGBN during adversarial attacks crafted by the gate modules, this might show correlations between the perturbations from the gate modules and those

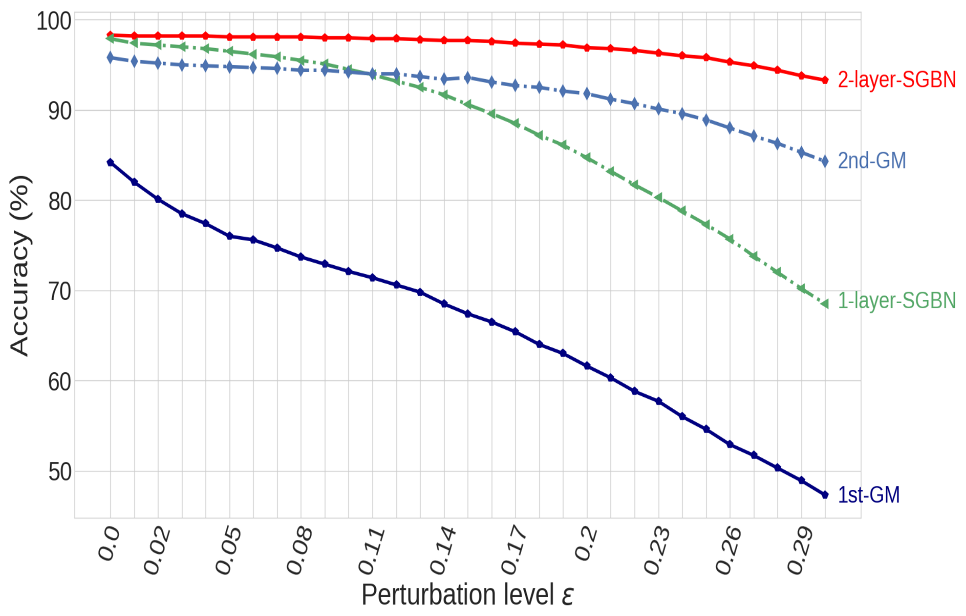


Figure 4.18: The accuracies of the first and second gate modules, 2 layer binarized synapses of SGBN, and 1 binarized synapse of SGBN.

from SGBN. Additionally, we check the gradients of the cost function with respect to inputs ($\nabla_x \mathrm{Loss}(\mathrm{w}; \mathrm{x}, \mathrm{y})$) for the gate modules to see if the gradients from the gate modules and those from SGBN are correlated.

The results are shown in figure 4.19 along with the accuracies of SGBN. These results show that adversarial examples crafted by SGBN and 2nd-GM were correlated since their accuracies are very similar. Furthermore, we applied the other models, CNN-BIN, SAP, BNN with PGD to the examples (Please see Appendix D). The results also showed the correlation of adversarial examples were crafted by SGBN and 2nd-GM.

We plot histograms for the gradients of the cost function with respect to inputs are plotted as seen in figures 4.20. The gradient variances of SGBN are very similar to the gradient of 2nd-GM, and these are larger than the variance of 1st-GM. The results also showed the correlation of the gradients of SGBN and 2nd-GM.

Necessity of the binarized dense layer

From these results, we hypothesize that the two gate modules might be enough to defend SGBN against white-box attacks. SGBN might not need to have a stochastic binarized dense layer, which is also one of the gradient masking approaches. If SGBN

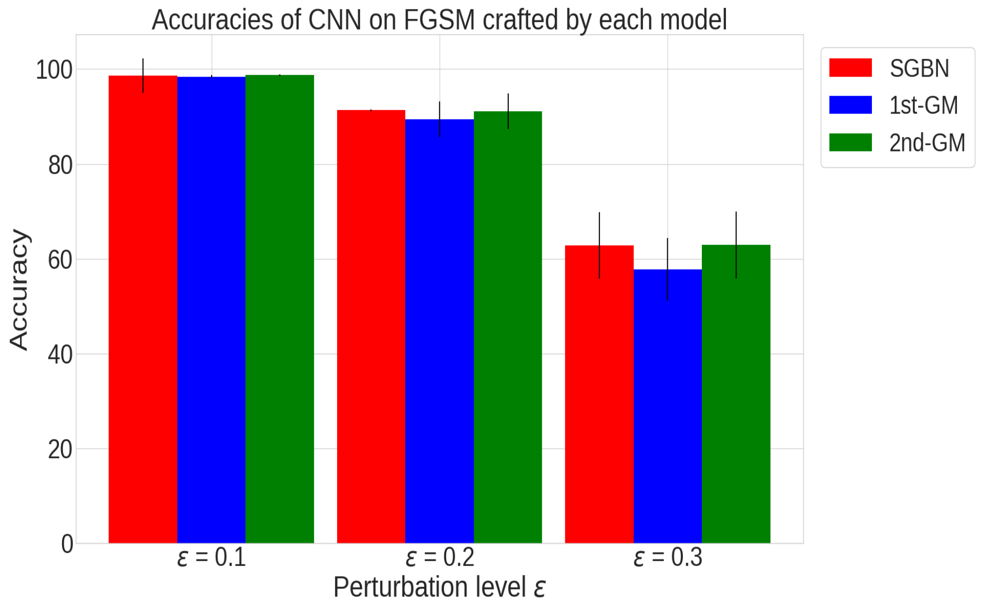
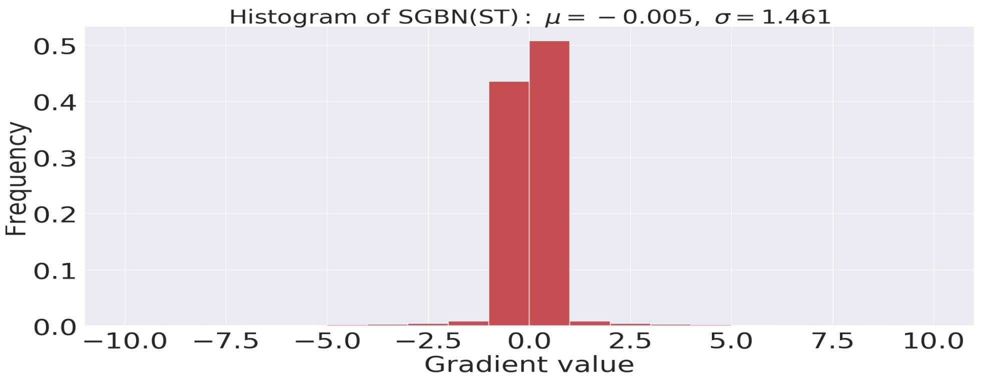


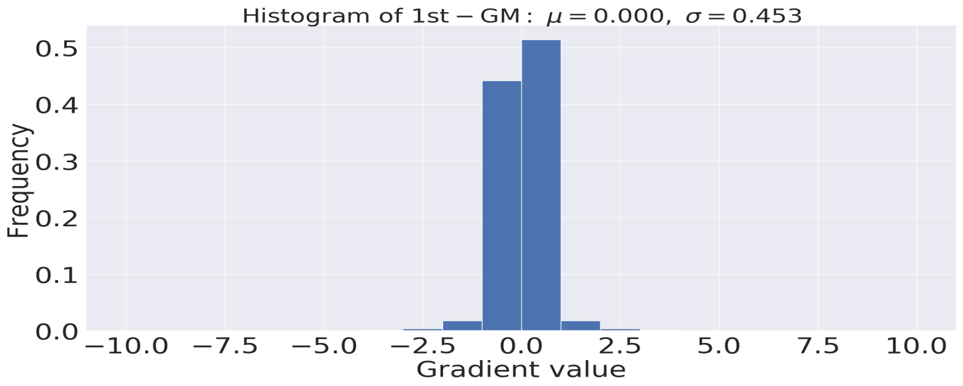
Figure 4.19: The accuracies of CNN on FGSM crafted by SGBN (red), first (blue), and second gate modules (green).

does not need the stochastic binarized dense layer, we might be able to seek a better activation function to improve the performances of a clean testing dataset and against adversarial attacks. To test this hypothesis, we trained SGBN without the stochastic binarized activation but with the rectified linear unit activation for the dense layer on the MNIST, and we tested these models on FGSM. This result is shown in figure 4.21. The figure shows that even though we changed the activation, SGBN achieved an accuracy of 91.1±1.1% at the highest perturbation level ($\epsilon = 0.3$). SGBN with a stochastic binarized activation function is slightly better, but we cannot see much difference between these models. For this experiment, the stochastic binarized activations for the dense layer are not needed for significant improvement on the white-box attack. For the first consideration, based on the regularization perspective, we still recommend using the stochastic binarized activation function for the dense layer when applying SGBN to the other adversarial attacks (The differences in accuracy between the training and testing MNIST datasets for SGBN and SGBN with relu are available in Appendix C). Then, the other activation functions should be tested to find the best activation function for classifying a clean dataset and defending a model against attacks.

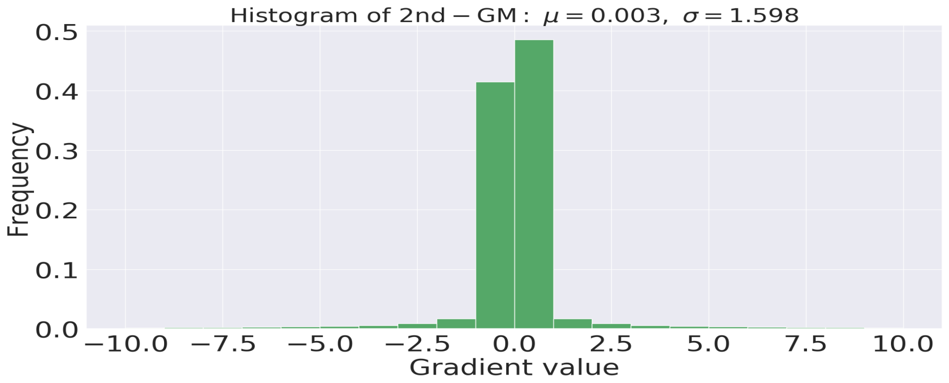
These results provide strong evidence that the two gate modules are the main



(a) Histogram of the gradients of the cost function with respect to inputs for SGBN (ST).



(b) Histogram of the gradients of the cost function with respect to inputs for the first gate module.



(c) Histogram of the gradients of the cost function with respect to inputs for the second gate module.

Figure 4.20: The gradient variances for SGBN (a) and 2nd-GM (c) are similar and larger than 1st-GM (b).

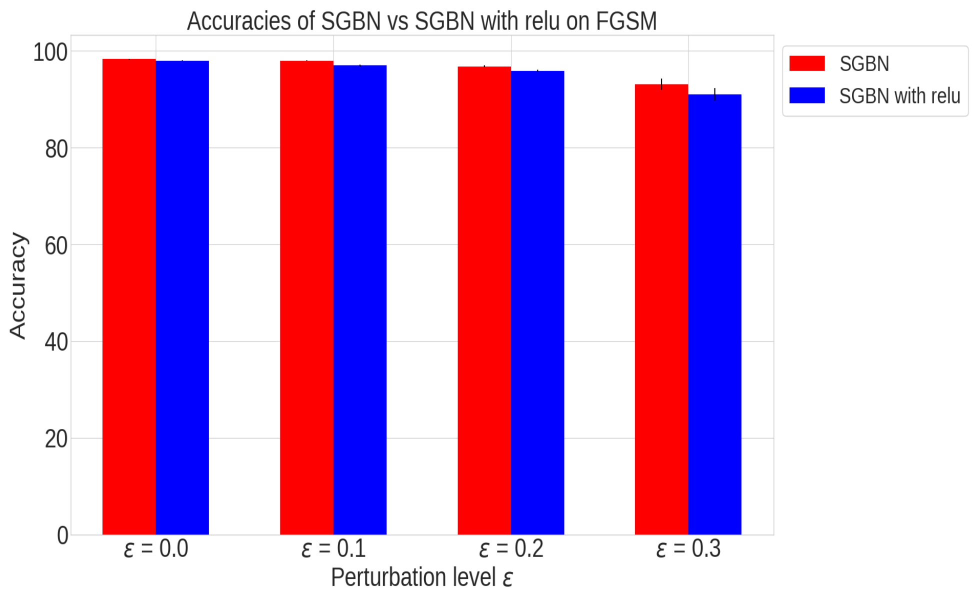


Figure 4.21: Each accuracy for SGBN with the stochastic binarized activation (red) vs SGBN with the rectified linear unit activation (blue) on FGSM.

causes of creating the weak perturbations for SGBN.

4.6.3 Double-sized filters for the gate module and the main network

Since the gate module learns the proper representation to classify the inputs, we arrive at another question: "Does SGBN need double-sized filters, filters for the gate module of the convolutional layer and filters for the main network of the convolutional layer, to protect itself against adversarial examples?" If SGBN does not need to have double-sized filters, we can reduce the parameter size, and this might help to make our model converge more quickly.

To answer this question, we compared the accuracies of a conventional SGBN on FGSM and one has shared weights. In this case, the shared weights mean convolutional filters in the main network are reused or shared with the gate modules.

The result is shown in figure 4.22. The accuracies of SGBN with shared weights (1W-SGBN) are very similar to the original model (2W-SGBN). Even though 2W-SGBN are slightly more accurate than 1W-SGBN at lower perturbation levels, and 1W-SGBN are slightly more accurate than 2W-SGBN at the perturbation levels higher than $\epsilon = 0.23$, there is not much difference between these models. Sharing the weights is very similar to adaptive dropout as described in [7]. In this study,

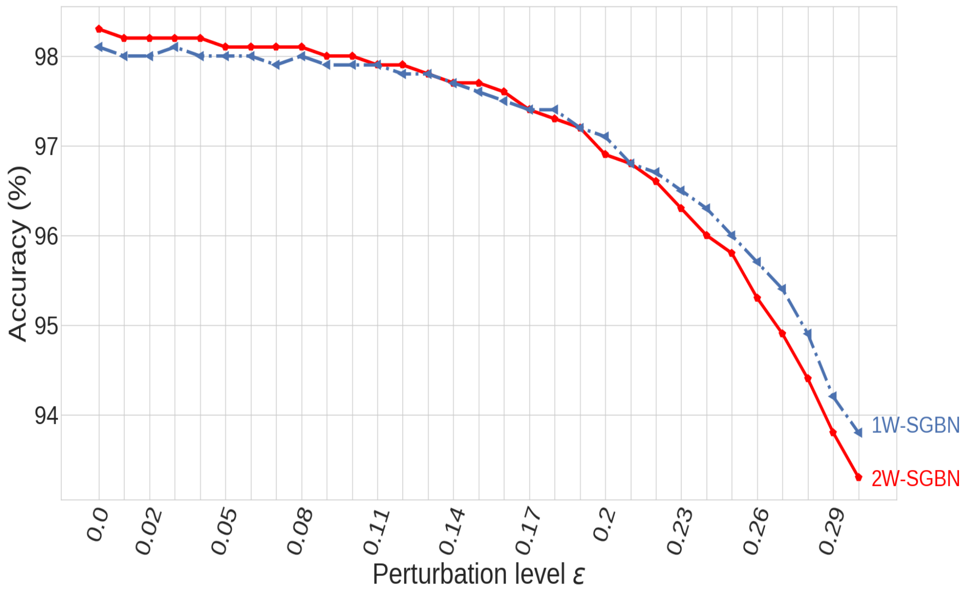


Figure 4.22: The accuracies of the original model (2W-SGBN) and SGBN with the shared weights (1W-SGBN) on FGSM.)

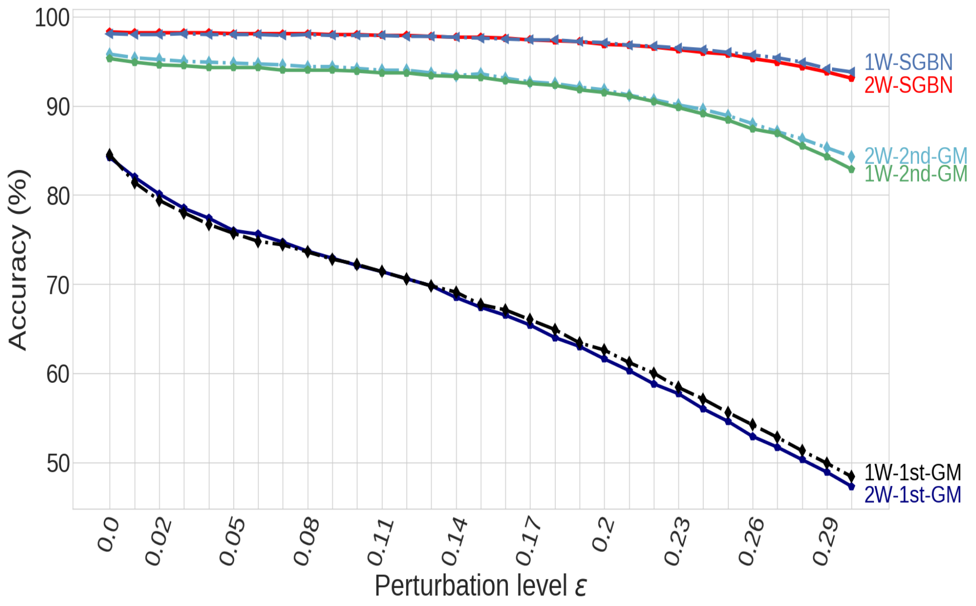


Figure 4.23: The accuracies of each model, 2 layer of SGBN (2W-SGBN), 2 layer of SGBN with the shared weights (1W-SGBN), the second gate module (2W-2nd-GM), the second gate module with the shared weights (1W-2nd-GM), the first gate module (2W-1st-GM), and the first gate module with the shared weights (1W-1st-GM).

the authors at first jointly trained a neural network with a binary belief network that learned the probability to drop out the activation, but later the authors also shared the weights of the main network and the belief network and trained them together. Because the convolutional layers in the gate modules and the main network learned similar features, the weights of the main network could be shared with the gate modules.

Classifying images by the gate modules with shared weights

We have already discussed how the two SGBN gate modules learn how to classify images. To confirm that the of 1W-SGBN gate modules learn how to classify images too, the first and second gate modules were trained by adding one dense output layer to the top of the gate modules similar to the last section. This is shown in figure 4.23. In this figure, we compare the gate modules with shared weights (1W-1st-GM and 1W-2nd-GM for the first and second gate module, respectively) and the gate modules for the original model (2W-1st-GM and 2W-2nd-GM for the first and second gate module, respectively). The results of 1W-1st-GM and 2W-1st-GM are very similar. The results of 1W-2nd-GM and 2W-2nd-GM are also similar even though 2W-2nd-GM is slightly more accurate at the highest perturbation level than 1W-2nd-GM. We have not seen much difference between the original model and shared weights version, especially at the lower perturbation levels. Furthermore, 1W-2nd-GM is less accurate on the clean dataset than 1W-SGBN even though 1W-2nd-GM uses shared weights.

4.6.4 Robustness on the black-box attack

So far, we have discussed our model's robustness against white-box attacks. In this section, we will also discuss the robustness of SGBN against black-box attacks compared with other gradient masking approaches.

We hypothesize that the robustness is caused by binarization constrained to real values or 0 (turned on or off) on the weights of the convolutional layers because it makes SGBN possible to ignore some perturbations.

To check this hypothesis, we replaced the stochastic 1 or 0 binarization with the stochastic +1 or -1 binarization activations (the same stochastic binarization implementation as Galloway et al. [34]) in the gate modules for SGBN. Furthermore, we

Table 4.8: The accuracies of the black-box attacks on FGSM with $\epsilon = 0.3$.

Target Models \ Source Models	SGBN	SAP	BNN	CNN-BIN	CNN
CNN-BIN	74.0±6.7%	73.5±4.2%	57.9±4.8%	44.6±2.7%	55.8±3.9%
SGBN with +1 or -1	74.9±0.9%	78.9±1.2%	44.0±2.0%	66.2±1.1%	64.7±1.4%
DropConnect	88.3±2.4%	81.6±2.1%	54.8±3.9%	74.7±3.0%	70.2±2.0%
Random Masking	88.7±3.1%	82.7±2.9%	55.6±4.4%	75.4±3.9%	71.6±3.8%
SGBN with randomly shuffled activations	89.9±1.6%	84.8±2.1%	58.3±3.7%	77.5±2.5%	73.0±3.1%
SGBN	93.1±1.2%	89.2±1.9%	63.9±3.9%	81.9±2.8%	75.6±3.0%

applied DropConnect [113] to the convolutional layers with the dropconnect probabilities set to 40 % for the first layer and 50 % for the second layer during both training and testing time. This is because around 40 % of the weights of SGBN for the first layer and 50 % of the weights for the second layer are turned off. Also, we replaced the gate modules with random masking using a Bernoulli distribution (p-value = 0.6 for the first layer and 0.5 for the second layer). We trained these models from scratch 10 times. This is to examine if random masking and dropconnect help the robustness against black-box attacks by allowing the models to ignore random pixels. The applications of random masking and dropconnect can also help us to understand if the gate modules learn useful masks or just random ones. Additionally, we applied one more model in this experiment. This model was based on SGBN, but when we applied SGBN to adversarial examples after training, we saved the activations of the gate modules. Then, we reused the activations to mask the weights, but the activations were randomly shuffled. This is depicted in figure 4.24. For instance, in the figure, one of the adversarial examples, in this case, an image of 0 with perturbations, is fed into SGBN, and the activations of the gate modules are saved. However, the activations of the gate modules for the image are not used for masking the weights. Rather, other activations (e.g. activations of the gate module for 1, 2, or another 0) are used[2]. We added this model in this experiment to check whether or not the performance changes when the activations of the gate modules are not proper for each input. We tested these models on FGSM of the MNIST for black-box attacks, which were crafted by SGBN, SAP, BNN with PGD, CNN-BIN, and CNN.

In table 4.8, we also show the performances of CNN-BIN and SGBN. We included

[2]More specifically, for example, in figure 4.7, the activation of GM for example "1" might be used for examples "9" to mask the weights.

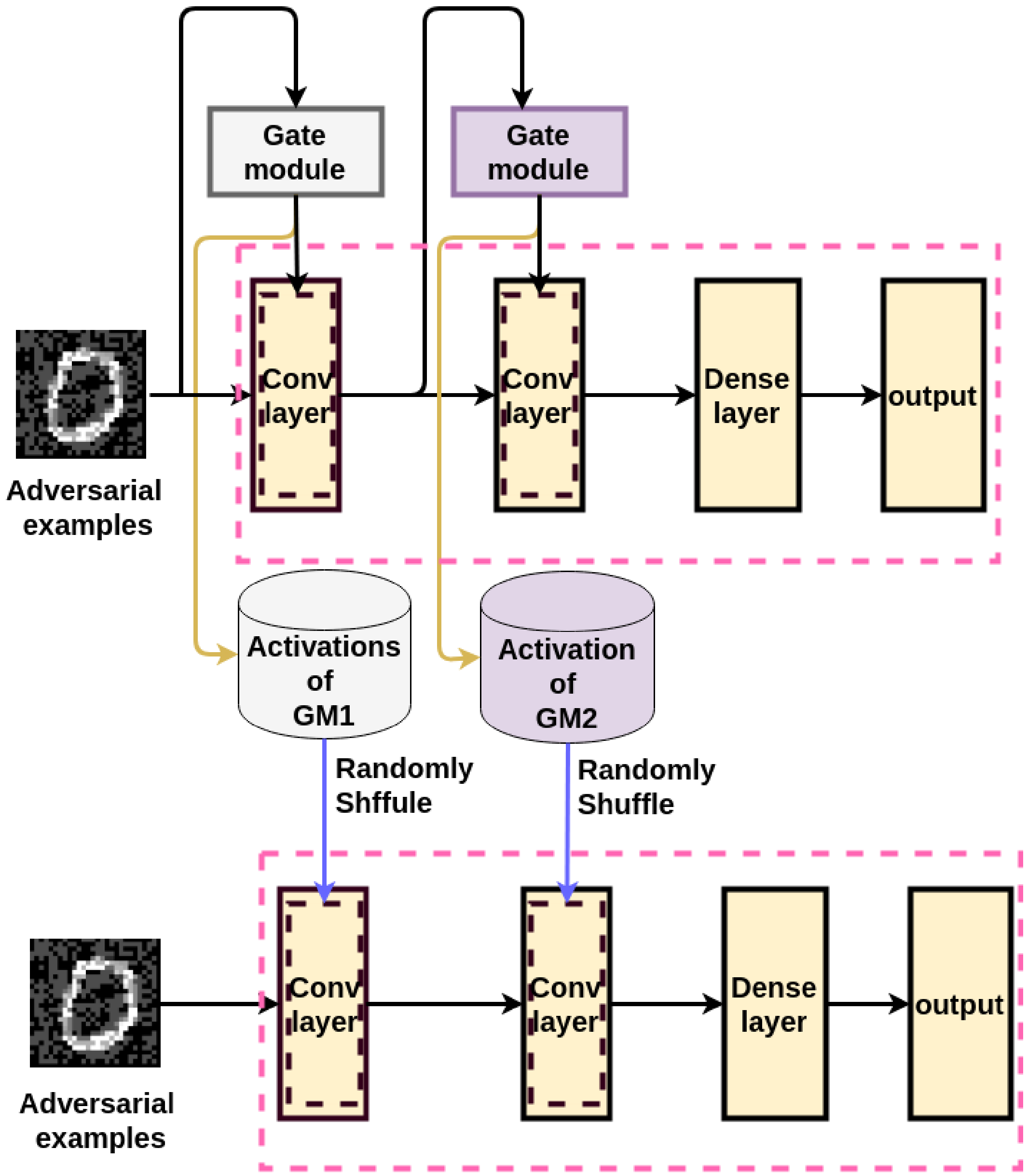


Figure 4.24: The activations of the gate modules are saved and randomly shuffled for testing the model on the black-box attacks.

CNN-BIN as a baseline for this experiment since we added some masking techniques (random masking and dropconnect) or gate modules with binarized activation (-1 or 1 and 0 or 1) to the convolutional layers of CNN-BIN. Random masking and dropconnect models significantly improve the robustness more than SGBN with +1 or -1 activations, especially against adversarial examples crafted by SGBN and SAP. However, we cannot see any improvements for these models against BNN. SGBN with shuffled activations diminish the performances compared with SGBN. The shuffled model performance is very similar to the performances of the random masking and dropconnect models.

The performance improvements gained by applying the random masking and dropconnect models to the black-box attacks might show us why our model is robust against these attacks. One of the reasons might be our hypothesis, the turning off the weights can make the model possible to ignore some perturbations. The other reason why SGBN improves the robustness against the attack is the gate module learns non-random masking since we have seen that the random masking, dropconnect, and the randomly shuffled activations models are not competitive with SGBN. This non-random masking the weights helps SGBN to defend itself against the black-box attacks.

4.6.5 Toy experiment

In previous subsections, some experiments were examined to gain a deeper understanding of the gate module and SGBN. However, we still do not know how the gate module affects the weights. We hypothesize that the activations of the gate module change to fit the circumstances. For example, if the output values of a layer need to be small, then the gate module will turn off many of the weights. To gain an understanding of this dynamics, we conduct a small experiment with SGBN. In this experiment, we also apply our original model that is implemented with weight inputs. We compare SGBN with one has the weight inputs to check how different these are.

Experiment setting

The experiment is a very simple regression task. Inputs are real values of [-1,1], and targets are absolute values of the inputs. We created 300 inputs using the numpy

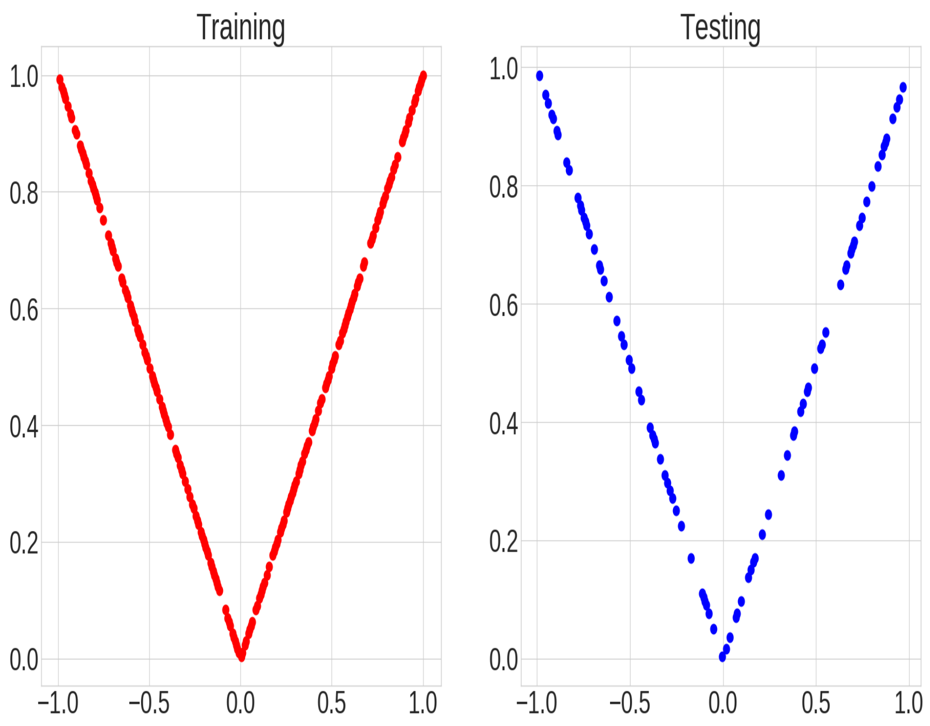


Figure 4.25: Plotting the training and testing targets on the toy experiment.

linspace function (linspace(-1.0, 1.0, 300)) and randomly separated them into a training dataset with 210 and a testing dataset with 90. Figure 4.25 is plotted for the training and testing targets.

The parameter settings for this experiment are as follows: The main network of SGBN consists of 2 nodes in a hidden layer and 1 node in an output layer. The gate module of SGBN has a perceptron to mask the weights between the hidden layer and the output layer in the main network. The activation function for the hidden layer is the tanh activation function. We also apply the original model of SGBN with weight inputs. This model uses the weights of the main network as inputs to the gate module. This is depicted in figure 4.26. For example, if there are 2 weights between the hidden layer and the output layer in the main network, then these 2 weights are used as inputs for the perceptron in the gate module along with other inputs. We expect that looking at the weights and input at the same time will accelerate the network convergence. In this experiment, we do not use convolutional layers for the gate module and the main network, and the output of the main network is without any activation function. All of the parameter settings are summarized in tables 4.9

Table 4.9: Network parameters and variables for the main network on the toy experiment

Parameters	all models
learning rate	0.001
Weights for the input to hidden	1×2
Activation func. in the hidden layer	tanh
Weights for the hidden to output	2×1

Table 4.10: Gate module variables for the second layer on the toy experiment

Parameters	Gate module
Weights for the gate module with the weight input	4×2
Weights for the gate module without the weight input	2×2
Activation func. in the output of both of the gate module	binarized

and 4.10.

MLP is applied as a baseline in this task. MLP and the main network of SGBN architecture are the same. Furthermore, the deterministic non-binarized versions of SGBN, with and without the weight inputs, are applied in this task. To create the deterministic non-binarized version of SGBN, we replaced the stochastic binarized activation function for the gate modules with the logistic function.

For this experiment, we use the Adam optimizer, and the learning rates for all of the models are 1e-3. The iterations of this experiment are 5000. We use batch, not mini-batch, for this experiment.

Result

Each result is shown in figure 4.27. Figure 4.28 shows the learning curves for each model in this experiment. First of all, conventional MLP (WOGM) could not solve this task properly. Because the tanh activation function does not have a sharp corner, the input values close to 0 are really hard to output the targets for WOGM. Compared with WOGM, deterministic non-binarized SGBN both with and without the weight inputs (GM_DT and GMWOW_DT, respectively) are better. The activations of the deterministic non-binarized gate modules are very dynamic compared with stochastic binarized models because the deterministic non-binarized gate modules use logistic activation values. On the other hand, stochastic binarized models (with the weight inputs: GM_ST, without the weight input:GMWOW_ST) might not be suitable for

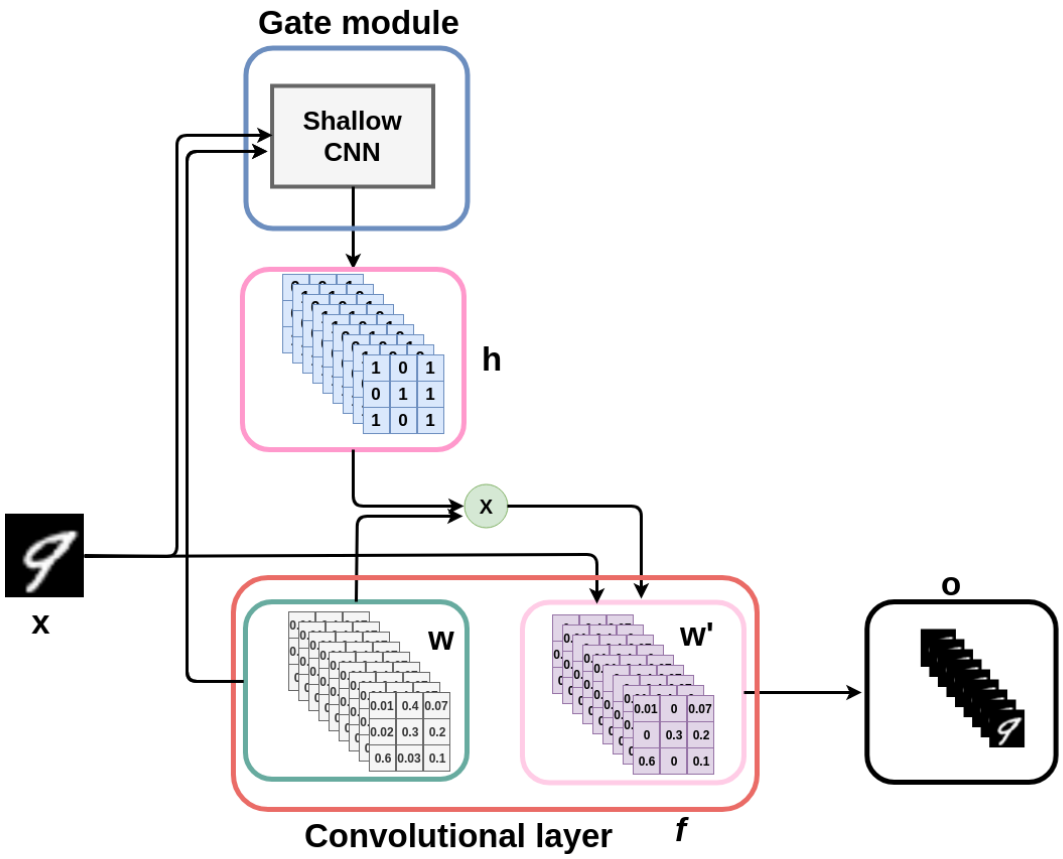


Figure 4.26: The original SGBN. A gate module learns which specific weights on the filter should be turned on or off by the image and weights. The weights from the convolutional layer will be used as input for the gate module. The weight inputs will be concatenated on the original input.

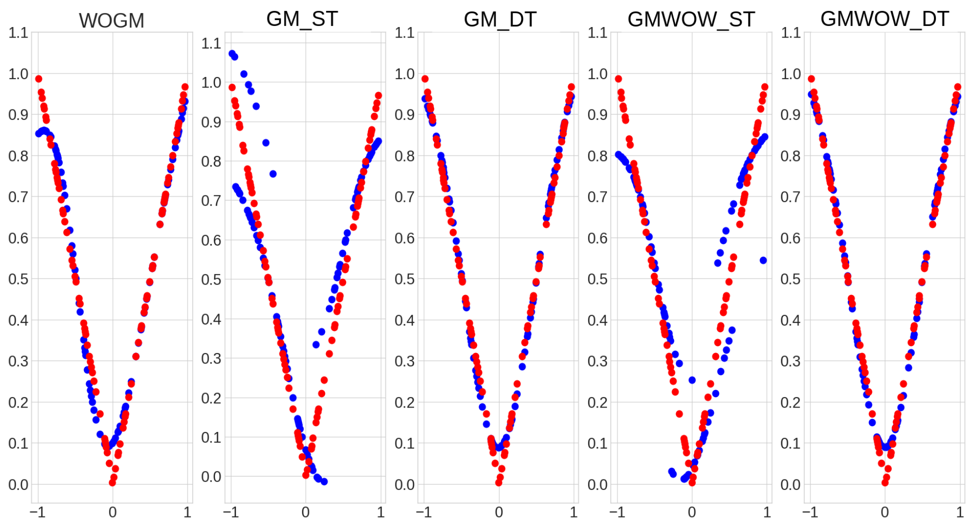


Figure 4.27: Experimental results on testing data for MLP (WOWGM), SGBN With the weight input for stochastic binarized and deterministic non-binarized activations (GM_ST and GM_DT), and SGBN without the weight input for stochastic binarized and deterministic non-binarized activations (GMWOW_ST and GMWOW_DT)). These results are the best results among 30 training from scratch. Blue dots are predictions of each model, and red dots are targets.

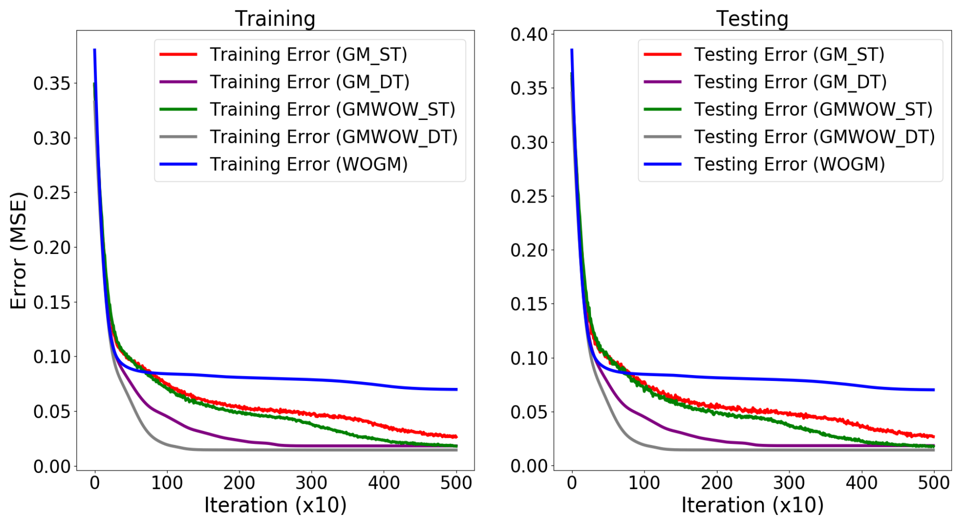


Figure 4.28: The learning curves for each model, MLP without gate module (WOGM), stochastic binarized activation gate module with the weight inputs (GM_ST), deterministic non-binarized activation gate module with the weight inputs (GM_DT), stochastic binarized activation gate module without the weight inputs (GMWOW_ST), and deterministic non-binarized activation gate module without the weight inputs (GMWOW_DT) on the toy experiment. These results are averaged over 30 runs for the models trained from scratch.

this problem because 1 or 0 masking does not change outputs dynamically. However, around the sharp curve (around input=0.0), the outputs of GM_ST and GMWOW_ST are very close to the targets.

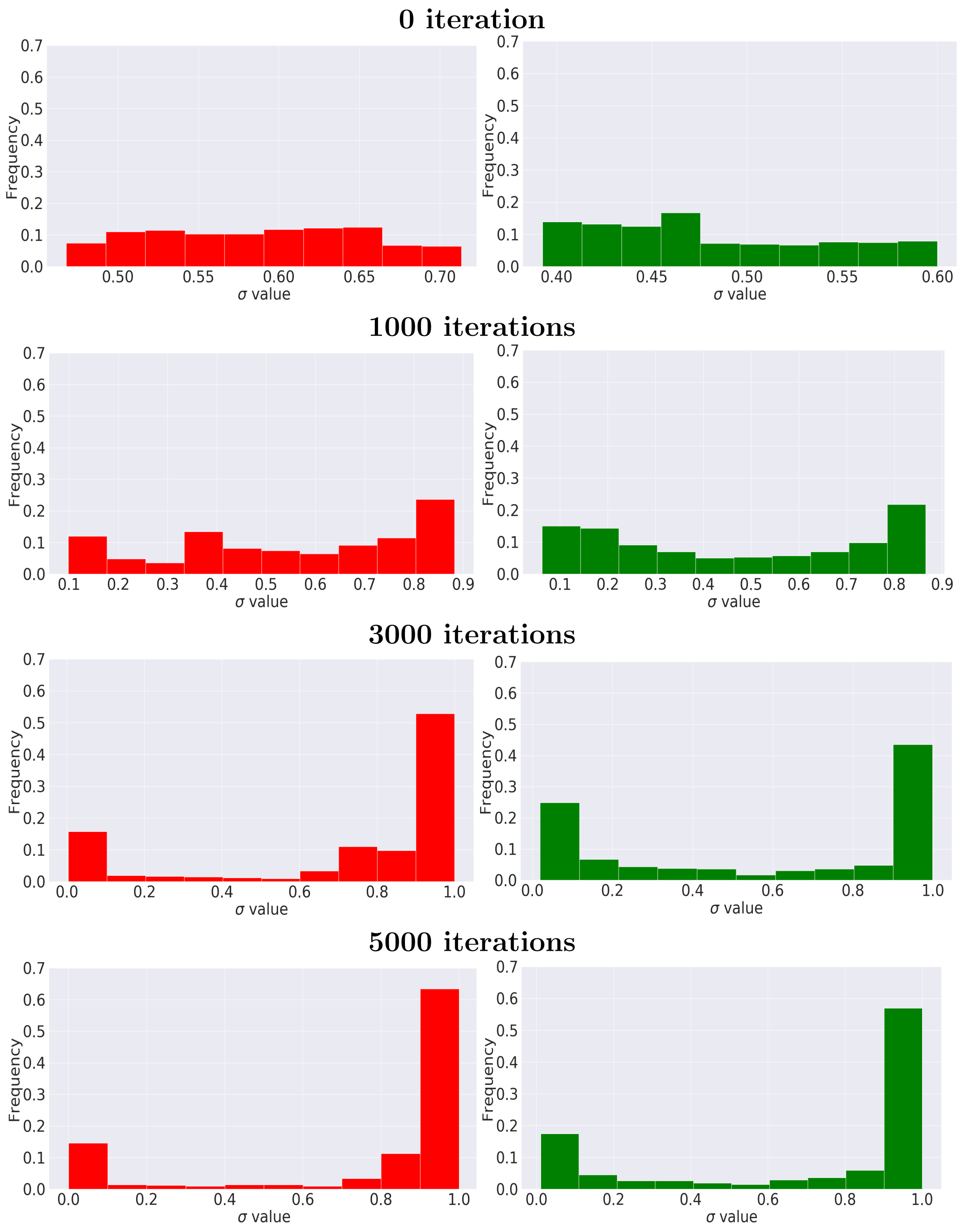
We also checked the activations of the gate modules with and without the weight inputs for this experiment in figures 4.30 and 4.31. All of SGBNs change the activation of the gate module depending on the input. For the stochastic binarized models, we have not seen much difference between GM_ST and GMWOW_ST. Both of the models change the activations at some points. For example, GMWOW_ST change activations around under -0.3 or over 0.5. Compared with these stochastic binarized models, GM_DT and GMWOW_DT are more dynamic to change the activations. Even though GM_DT and GMWOW_DT are better in this experiment, against adversarial examples, we need stochasticity and more 0 activations of the gate module (turning off) to protect SGBN against both of the white- and black-box attack.

In the learning curve, our models without the weight inputs (both stochastic and deterministic) converged faster than with the weight inputs. The parameter sizes and the bias of the weight inputs might cause these results.

Furthermore, we plot histograms in which the vertical axis represents the frequency and the horizontal axis represents the logistic activation values of the gate modules in both GM_ST and GMWOW_ST (before the activations are binarized) for 0, 1000, 3000, and 5000 iterations during the training. These are found in figure 4.29. We can see that the activations for the gate modules in both stochastic binarized models gradually become nearly deterministic as the number of iteration increased.

4.7 Discussion

In this chapter's "Motivation" section, we discussed learning the probability for Dropout, and that it is very similar to SGBN. To see if learning dropout probability is also helpful to defend a neural network against adversarial examples, we replace the gate module with the adaptive dropout [7] on 2 layer CNN with the binarized dense layer on the MNIST dataset (The dropout is applied to the convolutional layers). We tested the network in deterministic non-binarized and stochastic binarized ways. These are also tested on $\epsilon = 0.3$ of FGSM for the MNIST testing dataset, and the results are



(a) GM_ST (b) GMWOW_ST

Figure 4.29: Comparison of the logistic activation function values for GM_ST and GMWOW_ST. The first row is the histograms for 0 iteration, the second row is the histograms for 1000 iterations, the third is the histograms for 3000 iterations, and the last one is the histograms for 5000 iterations.

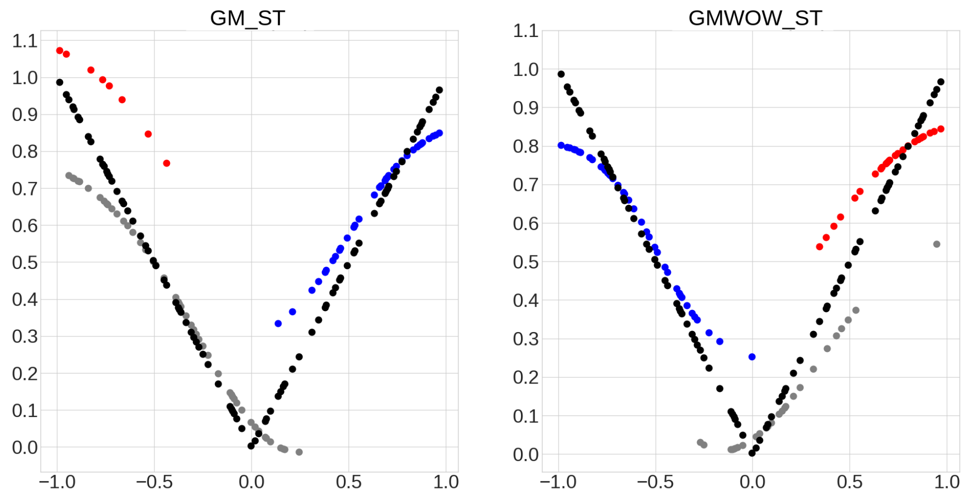


Figure 4.30: Results with the output of the stochastic binarized gate modules with and without the weight inputs. These results are corresponding to GM_ST and GMWOW_ST) on figure 4.27. Blue and red dots describe one of the weights for hidden to output layer will be used, and grey dots describe both weights for hidden to output layer are used.

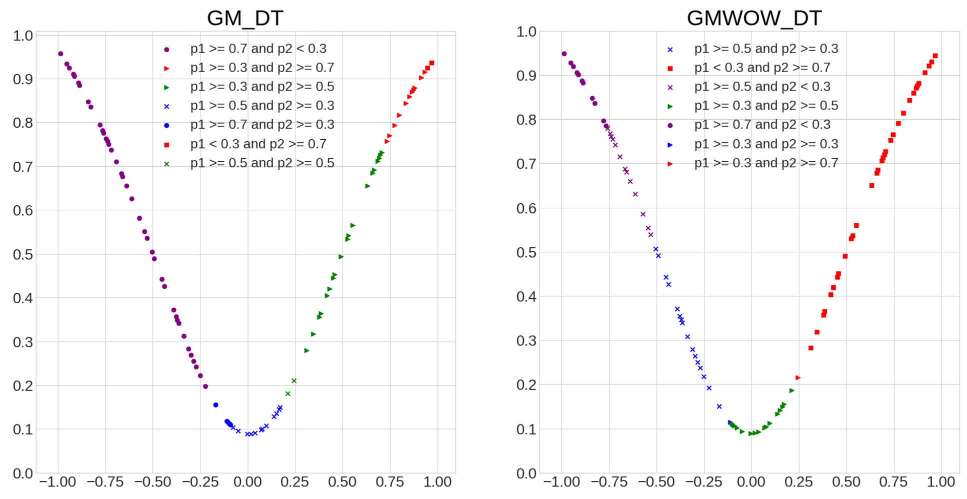


Figure 4.31: Results for the output of the deterministic non-binarized gate modules with and without the weight inputs. These results are corresponding to GM_DT and GMWOW_DT on figure 4.27. p1 is outputs of the logistic function on the gate module for one of the two weights, p2 is for the other one. Both deterministic gate modules are very dynamic to the inputs.

- Adaptive dropout (deterministic): 54.8±4.7%
- Adaptive dropout (stochastic): 62.7±1.6%

These accuracies are again worse than SGBN, but this adaptive dropout does improve the accuracy (without this dropout, again, CNN-BIN, 44.6±2.7%), especially stochastic one. We find that the stochastic dropout method drops out mostly non-centered nodes. The reason why this might occur with this method is that the MNIST dataset is a centered-image dataset. Dropping out the non-centered nodes with perturbations might help to defend against the attack, but is not robust enough compared with SGBN.

We mentioned SGBN contains more parameters than the other binarized models in the network parameter section. To test whether or not the additional parameters are the main reason for our improvement, we trained conventional CNN, CNN-BIN, SAP, and BNN with the additional parameters (the same or similar size as SGBN's total parameter size) on the MNIST and CIFAR-10 testing dataset (parameter settings are in tables 4.12 and 4.13), and tested them on $\epsilon = 0.3$ of FGSM (percentages in parentheses are without the additional parameters):

- MNIST
  - CNN: 7.2±0.9% (15.3±2.2%)
  - BNN with PGD: 70.3±2.9% (76.8±1.1%)
  - CNN-BIN: 61.7% (44.6±2.7%)
  - SAP: 63.7±11.9% (73.5±3.1%)
- CIFAR-10
  - CNN: 4.3±0.5% (1.7±0.4%)
  - BNN with PGD: 14.0±1.1% (13.6±1.2%)
  - SAP: 6.3±0.4% (2.3±0.3%)

On MNIST, most of the models are worse than without the additional parameters except for CNN-BIN. Even though CNN-BIN improves the accuracy with the additional parameters, it does not achieve a comparable accuracy with SGBN (93.1±1.2%). On

Table 4.11: Summary of each accuracy for each model and each dataset on FGSM

Network	Dataset	$\epsilon = 0.0$	$\epsilon = 0.1$	$\epsilon = 0.2$	$\epsilon = 0.3$
SGBN	MNIST	98.3±0.1%	98.0±0.1%	96.8±0.2%	93.1±1.2%
	CIFAR10	61.6±1.1%	32.3±1.5%	22.1±0.7%	17.3±0.8%
1W-SGBN	MNIST	98.1±0.1%	97.9±0.1%	97.1±0.1%	93.8±0.3%
SGBN (DT)	MNIST	98.3±0.1%	93.0±0.5%	80.8±2.3%	61.5±3.7%
BNN with PGD	MNIST [34]	98.9±0.03%	96.8±0.3%	93.4±0.3%	85.6±0.6%
	MNIST	95.4±0.4%	92.6±0.5%	86.2±1.0%	76.8±1.1%
	CIFAR10	50.4±2.9%	26.5±1.3%	16±1.3%	13.6±1.2%
BNN with PGD (add params)	MNIST	98.1±0.2%	91.6±2.0%	83.4±2.7%	70.3±2.9%
	CIFAR10	46.0±3.5%	29.8±1.7%	20.1±1.7%	14.0±1.1%
SAP	MNIST	96±0.3%	95.4±0.4%	93.0±1.6%	73.5±3.1%
	CIFAR10	58.2±1.1%	1.6±0.3%	1.6±0.3%	2.3±0.3%
SAP (add params)	MNIST	89.1±2.9%	84.7±5.9%	76.7±7.3%	63.7±6.8%
	CIFAR10	57.6±0.6%	6.6±0.5%	5.8±0.5%	6.3±0.4%
CNN-BIN	MNIST	98.3±0.1%	91.8±0.6%	76.5±3.0%	44.6±2.7%
	CIFAR10	N/A	N/A	N/A	N/A
CNN-BIN (add params)	MNIST	94.3±6.8%	78.3±15.4%	68.4±13.0%	61.7±11.9%
	CIFAR10	N/A	N/A	N/A	N/A
CNN	MNIST	99.0±0.1%	72.5±2.3%	31.8±3.1%	15.3±2.2%
	CIFAR10	64.9±1.9%	0.5±0.1%	0.9±0.3%	1.0±0.4%
CNN (add params)	MNIST	99.1±0.1%	69.8±2.8%	18.6±2.1%	7.2±0.9%
	CIFAR10	66.2±1.4%	3.1±0.4%	3.3±0.4%	4.3±0.5%
AD	MNIST(ST)	98.3±0.1%	92.8±0.2%	82.1±0.6%	62.7±1.6%
	MNIST(DT)	98.3±0.1%	91.5±0.5%	76.9±3.6%	54.8±4.7%
AD (add params)	MNIST(ST)	96.2±3.7%	88.4±2.9%	74.2±3.3%	59.3±3.2%
	MNIST(DT)	97.2±3.0%	87.9±2.6%	72.8±3.4%	57.0±3.3%
RM p=0.6	MNIST	97.5±0.1%	88.6±0.6%	68.6±2.5%	44.6±2.8%
RM p=0.6 (add params)	MNIST	93.2±7.5%	83.2±6.4%	66.8±5.4%	50.4±3.5%
DC p = 40%	MNIST	97.8±0.1%	89.8±0.7%	70.0±2.3%	46.4±2.0%
DC p = 40% (add params)	MNIST	97.3±0.2%	92.0±0.6%	79.8±1.7%	61.5±1.9%

*AD = Adaptive Dropout, *RM = Random Mask, *DC = DropConnect

CIFAR-10, the additional parameters improve all of the model accuracies, indeed suggesting that doing further testing in the future on larger models would be worthwhile.

The gate module is one of the gradient masking approaches and a regularizer. These two methods protect SGBN itself against adversarial examples because the gradient masking approach creates weak perturbations, and the regularizer smooth decisions, thereby moving them further from training examples.

We summarize all of the network performances on the white-box attacks for MNIST and CIFAR-10 in table 4.11.

Table 4.12: Network parameters and variables for the additional parameters experiments on MNIST

Parameters	NON-SGBN models
learning rate	0.001
Weights in the first convolutional layer	64×5×5
Weights in the second convolutional layer	128×5×5
Weights in the fully connected layer	6952
Weights in the last fully connected layer	10

Table 4.13: Network parameters and variables for the additional parameters experiments on CIFAR

Parameters	NON-SGBN models
learning rate	0.001
Weights in the first convolutional layer	64×5×5
Weights in the second convolutional layer	256×5×5
Weights in the third convolutional layer	256×5×5
Weights in the fully connected layer	1176
Weights in the last fully connected layer	10

Chapter 5

Conclusions and Future Work

5.1 Conclusion

In this dissertation, we first considered that the cause of adversarial examples is related to overfitting, referring to the work of Galloway et al. [35]. To examine this statement, we have shown that the cause of adversarial examples is related to overfitting by applying some regularization to a shallow network and adding adversarial sensitivity. The overly strong regularizations led the network to underfit, but this underfitting was better than overfitting for adversarial examples. This result is very important in determining the cause of adversarial examples. In addition, we applied stochastic binarized neural networks with the straight-through estimator and REINFORCE to adversarial examples. Among them, the strongest regularization, REINFORCE, had the best performance against adversarial examples. These stochastic binarized neural networks are, of course, gradient masking approaches which do not have a useful gradient to craft adversarial examples, but we showed that the strongly regularized model was more robust against adversarial examples among gradient masking approach models.

A model, SGBN, was presented in this dissertation, as a new model of a recognition network that could defend itself better against adversarial examples than other state-of-the-art proposals. This is based on stochastic binarized neurons. We implemented SGBN with the gate module because we hypothesized that combining stochasticity and binarization is helpful to defend the networks. SGBN is a regularizer and one of the gradient masking approaches. We hypothesized that adding a combination of stochasticity and binarization to the stochastic binarized model that was implemented for the first study of this dissertation would make it more robust against adversarial attacks. In the experiments, we tested this model on white- and black-box attacks, and our model was more robust against adversarial examples than the other gradient masking models. The cause of this strength against adversarial examples for SGBN was, indeed, one of the gradient masking approaches, the larger variances of the gradient of the cost function with respect to inputs, a regularizer, and 0 or 1 maskings which ignored some of the perturbed pixels.

SGBN has double-sized filters for the gate module and the convolutional layer in the main networks. We assumed that our model might not need to have these double-sized filters for both of the gate modules and convolutional layers in the main network, and found that the main network's filters can be shared with the convolutional layers in the gate module.

The benefits of SGBN are 1) the gate module is a strong regularizer; 2) SGBN is strong against some adversarial examples including both white- and black-box attacks compared with the other gradient masking approaches; 3) it is easy to implement or to understand; 4) our model's binarization does not greatly harm the accuracy on a clean dataset.

We did not apply adversarial training to our model in this dissertation to avoid computational expense and overfitting to specific perturbations. Again, Galloway et al. [33] showed that adversarial training models overfit specific perturbations, and these models were weak against other types of attacks. We would like to avoid this overfitting. Furthermore, for our model, presumably adversarial training would not work since our model generates weaker perturbations than the other gradient masking models. We have already seen that perturbations at the highest perturbation level were classifiable for even a conventional convolutional neural network. This means adversarial examples crafted by SGBN would be similar to original examples, and these examples would not be useful to train SGBN adversarially.

Even if a deep neural network model can achieve a high accuracy on the clean dataset, it is still vulnerable to adversarial examples. From our results and Galloway's studies [35, 34, 32], we suggest that some regularization forms, especially the gate module, should be applied to a model to protect it against adversarial examples instead of adversarial training, which overfits specific perturbations. In Galloway et al. [32], the authors applied L2 weight decay to a model, and the model lost small percentages of accuracy on the clean testing dataset. However, L2 weight decay drastically improved the performance of the model on adversarial attacks. The authors believe that this loss in clean testing accuracy in exchange for a huge gain during the attack is a worthwhile trade-off. We also agree with their belief. The gate module, again, reduces the performance on the clean dataset by a few percentages but can protect the model against both white- and black-box attacks. This is a good trade-off for the model with a small price.

5.2 Future work

We did not apply non-FGSM based attack, such as JSMA [86], to our model. JSMA has very high computation costs due to the need for computing the Jacobian matrix [68]. We avoided these high costs, but we would like to apply our model to JSMA in future work to see if SGBN can still defend itself against it. Presumably, SGBN would be resilient to JSMA, in which perturbations are added to a few pixels because turning off the weights might ignore the perturbations.

Another thing we are interested in is applying the gate module to parts of deep network architectures, such as a residual neural network (ResNet) [41]. Many researchers have already shown that ResNet is very robust against adversarial examples. Combining ResNet and the gate module could be very beneficial for defending against adversarial examples. Since the gate modules improved our shallow model, we believe that applying the gate modules to a deeper net such as ResNet would be beneficial in improving its robustness against adversarial examples. Furthermore, stochastic activation pruning is applied to ResNet in [21], and SAP reduces the performance on a clean dataset compared with a conventional ResNet. We assume that the gate modules would not greatly reduce the performance of ResNet on a clean dataset compared with SAP since we showed that the gate module did not greatly reduce the performance of our model in this dissertation.

Since we could see that SGBN was still resilient to the adversarial attack when we changed the dense layer with the stochastic activation to the relu activation function, we can replace a dense layer with the stochastic activation function with a dense layer with another activation function depending on each task. Seeking the best replacement for the activations might help a model to improve its performance on both the clean testing dataset and adversarial attack.

Furthermore, a deterministic non-binarized gate module's activation would be still changeable depending on the task. We have used the logistic activation function for the deterministic non-binarized gate module in this dissertation, but this does not need to be the activation function. We would replace it with tanh, asinh, rectified linear unit, and another one depending on each task.

SGBN can be used as an attention model [74, 119, 39]. Typically, attention is focused on an input, but in our case, attention is focused on the weights. We would like to apply both stochastic binarized and deterministic non-binarized versions of SGBN to non-centered images such as Cluttered Translated MNIST [74]. Conventional attention models work well on the dataset because the attention is focused on the specific part of the image (target image). We would like to compare the performances of the attention focused on the input and the attention focused on the weights on the dataset to see if the attention focused on the weights would still be beneficial for these cluttered images.

Finally, our model would also be useful for reinforcement learning, especially deep reinforcement learning [75, 76, 67, 73, 28]. Stochasticity for a reinforcement learning agent has the potential to improve exploration and handling of uncertainty [31, 29, 30, 88, 99]. In our research, the gate modules used sampling to learn which weights should be turned on or off stochastically during the training, depending on the input. We hypothesize that this sampling would work in a reinforcement learning environment. After the training, the activations of the gate modules would be nearly deterministic similar to our experiment, but during the training, they would still be a stochastic way to decide which weights should be turned on or off. We would like to examine if SGBN improves the performance of an agent in a reinforcement learning environment.

Bibliography

[1] The complete beginner's guide to deep learning: Convolutional neural networks and image classification. `https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb`. Accessed: 2019-11-21.

[2] Convolutional neural networks (cnns / convnets). `http:http://cs231n.github.io/convolutional-networks/`. Accessed: 2019-07-15.

[3] Image classification using convolutional neural networks in keras. `https://www.learnopencv.com/image-classification-using-convolutional-neural-networks-in-keras/`. Accessed: 2019-11-20.

[4] When to use mlp, cnn, and rnn neural networks. `https://machinelearningmastery.com/when-to-use-mlp-cnn-and-rnn-neural-networks/`. Accessed: 2019-11-19.

[5] Alexander G Anderson and Cory P Berg. The high-dimensional geometry of binary neural networks. *arXiv preprint arXiv:1705.07199*, 2017.

[6] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018.

[7] Jimmy Ba and Brendan Frey. Adaptive dropout for training deep neural networks. In *Advances in Neural Information Processing Systems*, pages 3084–3092, 2013.

[8] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[9] Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[10] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 387–402. Springer, 2013.

[11] Nicholas Carlini, Guy Katz, Clark Barrett, and David L Dill. Provably minimally-distorted adversarial examples. *arXiv preprint arXiv:1709.10207*, 2017.

[12] Nicholas Carlini, Pratyush Mishra, Tavish Vaidya, Yuankai Zhang, Micah Sherr, Clay Shields, David Wagner, and Wenchao Zhou. Hidden voice commands. In *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16, pages 513–530, Berkeley, CA, USA, 2016. USENIX Association.

[13] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. IEEE, 2017.

[14] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 15–26. ACM, 2017.

[15] Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704*, 2016.

[16] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS Deep Learning Workshop*, 2014.

[17] Ronald Clark, Sen Wang, Andrew Markham, Niki Trigoni, and Hongkai Wen. Vidloc: A deep spatio-temporal model for 6-dof video-clip relocalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6856–6864, 2017.

[18] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In Corinna Cortes, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama, and Roman Garnett, editors, *NIPS*, pages 3123–3131, 2015.

[19] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to + 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[20] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[21] Guneet S. Dhillon, Kamyar Azizzadenesheli, Zachary C. Lipton, Jeremy Bernstein, Jean Kossaifi, Aran Khanna, and Anima Anandkumar. Stochastic activation pruning for robust adversarial defense. *arXiv preprint arXiv:1803.01442*, 2018.

[22] Lynn E Dobrunz and Charles F Stevens. Heterogeneity of release probability, facilitation, and depletion at central synapses. *Neuron*, 18(6):995–1008, 1997.

[23] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting adversarial attacks with momentum. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9185–9193, 2018.

[24] Gamaleldin Elsayed, Dilip Krishnan, Hossein Mobahi, Kevin Regan, and Samy Bengio. Large margin deep networks for classification. In *Advances in Neural Information Processing Systems*, pages 842–852, 2018.

[25] Gamaleldin Elsayed, Shreya Shankar, Brian Cheung, Nicolas Papernot, Alexey Kurakin, Ian Goodfellow, and Jascha Sohl-Dickstein. Adversarial examples that fool both computer vision and time-limited humans. In *Advances in Neural Information Processing Systems*, pages 3910–3920, 2018.

[26] Ryosuke Enoki, Yi-ling Hu, David Hamilton, and Alan Fine. Expression of long-term plasticity at individual synapses in hippocampus is graded, bidirectional, and mainly presynaptic: optical quantal analysis. *Neuron*, 62(2):242–253, 2009.

[27] Ivan Evtimov, Kevin Eykholt, Earlence Fernandes, Tadayoshi Kohno, Bo Li, Atul Prakash, Amir Rahmati, and Dawn Song. Robust physical-world attacks on deep learning models. *arXiv preprint arXiv:1707.08945*, 1, 2017.

[28] Farzaneh S Fard and Thomas P Trappenberg. A novel model for arbitration between planning and habitual control systems. *arXiv preprint arXiv:1712.02441*, 2017.

[29] Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1704.03012*, 2017.

[30] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

[31] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[32] Angus Galloway. Source for paper attacking binarized neural networks. https://github.com/AngusG/cleverhans-attacking-bnns, 2018.

[33] Angus Galloway, Thomas Tanay, and Graham W Taylor. Adversarial training versus weight decay. *arXiv preprint arXiv:1804.03308*, 2018.

[34] Angus Galloway, Graham W. Taylor, and Medhat Moussa. Attacking binarized neural networks. *International Conference on Learning Representation*, abs/1711.00449, 2018.

[35] Angus Galloway, Graham W. Taylor, and Medhat Moussa. Predicting adversarial examples with high confidence. *arXiv preprint arXiv:1802.04457*, 2018.

[36] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*, 2010.

[37] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[38] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[39] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

[40] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens van der Maaten. Countering adversarial images using input transformations. *arXiv preprint arXiv:1711.00117*, 2017.

[41] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[42] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[43] Neal A Hessler, Aneil M Shirke, and Roberto Malinow. The probability of transmitter release at a mammalian central synapse. *Nature*, 366(6455):569, 1993.

[44] Geoffrey Hinton. Lecture 9.3 – Using noise as a regularizer. COURSERA: Neural Networks for Machine Learning, 2012.

[45] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 29, 2012.

[46] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[47] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[48] Emily P Huang and Charles F Stevens. Estimating the distribution of synaptic reliabilities. *Journal of neurophysiology*, 78(6):2870–2880, 1997.

[49] Ruitong Huang, Bing Xu, Dale Schuurmans, and Csaba Szepesvári. Learning with a strong adversary. *arXiv preprint arXiv:1511.03034*, 2015.

[50] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in neural information processing systems*, pages 4107–4115, 2016.

[51] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis R. Bach and David M. Blei, editors, *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456, 2015.

[52] Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. *arXiv preprint arXiv:1707.07328*, 2017.

[53] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[54] Alex Kendall and Roberto Cipolla. Modelling uncertainty in deep learning for camera relocalization. In *2016 IEEE international conference on Robotics and Automation (ICRA)*, pages 4762–4769. IEEE, 2016.

[55] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE international conference on computer vision*, pages 2938–2946, 2015.

[56] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[57] Diederik P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pages 2575–2583, 2015.

[58] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[59] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[60] Yoshimasa Kubo and Thomas Trappenberg. Mitigating overfitting using regularization to defend networks against adversarial examples. In *Canadian AI 2019: Advances in Artificial Intelligence*, 2019.

[61] Yoshimasa Kubo, Michael Traynor, Thomas Trappenberg, and Sageev Oore. Learning adaptive weight masking networks for adversarial examples. In *International Joint Conference on Neural Networks*, 2019.

[62] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[63] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[64] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[65] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 1998.

[66] Feifei Li, Justin Johnson, and Serena Yeung. Lecture 6 – Training Neural Networks I. Lecture collection: Convoluitional neural networks for visual recognition (Spring 2017), 2017.

[67] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[68] Shiqing Ma, Yingqi Liu, Guanhong Tao, Wen-Chuan Lee, and Xiangyu Zhang. Nic: Detecting adversarial samples with neural network invariant checking. In *NDSS*, 2019.

[69] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[70] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, abs/1706.06083, 2017.

[71] Stuart McIlroy, Yoshimasa Kubo, Thomas Trappenberg, James Toguri, and Christian Lehmann. In vivo classification of inflammation in blood vessels with convolutional neural networks. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 3022–3027. IEEE, 2017.

[72] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

[73] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[74] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014.

[75] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[76] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[77] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 86–94. Ieee, 2017.

[78] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: A simple and accurate method to fool deep neural networks. In *CVPR*, pages 2574–2582. IEEE Computer Society, 2016.

[79] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 427–436, 2015.

[80] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[81] Nicolas Papernot, Ian Goodfellow, Ryan Sheatsley, Reuben Feinman, and Patrick McDaniel. cleverhans v1. 0.0: an adversarial machine learning library. *arXiv preprint arXiv:1610.00768*, 10, 2016.

[82] Nicolas Papernot and Patrick McDaniel. On the effectiveness of defensive distillation. *arXiv preprint arXiv:1607.05113*, 2016.

[83] Nicolas Papernot and Patrick McDaniel. Extending defensive distillation. *arXiv preprint arXiv:1705.05264*, 2017.

[84] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv preprint arXiv:1605.07277*, 2016.

[85] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 506–519. ACM, 2017.

[86] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. pages 372–387. IEEE, 2016.

[87] Nicolas Papernot, Patrick Drew McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *IEEE Symposium on Security and Privacy*, pages 582–597. IEEE Computer Society, 2016.

[88] Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*, 2017.

[89] Sebastian Raschka. *Python machine learning*. Packt Publishing Ltd, 2015.

[90] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[91] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[92] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[93] Maria Rigaki. Adversarial deep learning against intrusion detection classifiers, 2017.

[94] Christian Rosenmund, John D Clements, and Gary L Westbrook. Nonuniform probability of glutamate release at a hippocampal synapse. *Science*, 262(5134):754–757, 1993.

[95] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[96] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. Defense-gan: Protecting classifiers against adversarial attacks using generative models. *arXiv preprint arXiv:1805.06605*, 2018.

[97] Arthur L Samuel. Some studies in machine learning using the game of checkers. ii—recent progress. In *Computer Games I*, pages 366–400. Springer, 1988.

[98] Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Aleksander Madry. Adversarially robust generalization requires more data. In *Advances in Neural Information Processing Systems*, pages 5014–5026, 2018.

[99] Wenling Shang, Douwe van der Wal Herke, Herke Van Hoof, and Max Welling. Stochastic activation actor-critic methods. 2018.

[100] Farzaneh Sheikhnezhad Fard. Modelling human target reaching using a novel predictive deep reinforcement learning technique. 2018.

[101] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[102] Liwei Song and Prateek Mittal. Inaudible voice commands. *arXiv preprint arXiv:1708.07238*, 2017.

[103] Suraj Srinivas, Akshayvarun Subramanya, and R Venkatesh Babu. Training sparse neural networks. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*, pages 455–462. IEEE, 2017.

[104] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[105] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 2019.

[106] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[107] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[108] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *International Conference on Learning Representation*, 2014.

[109] Thomas Tanay and Lewis Griffin. A boundary tilting persepective on the phenomenon of adversarial examples. *arXiv preprint arXiv:1608.07690*, 2016.

[110] Wei Tang, Gang Hua, and Liang Wang. How to train a compact binary neural network with high accuracy? In *AAAI*, pages 2625–2631, 2017.

[111] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

[112] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015.

[113] Li Wan, Matthew D. Zeiler, Sixin Zhang, Yann LeCun, and Rob Fergus. Regularization of neural networks using dropconnect. In *ICML (3)*, volume 28 of *JMLR Proceedings*, pages 1058–1066, 2013.

[114] Siyue Wang, Xiao Wang, Pu Zhao, Wujie Wen, David Kaeli, Peter Chin, and Xue Lin. Defensive dropout for hardening deep neural networks under adversarial attacks. *arXiv preprint arXiv:1809.05165*, 2018.

[115] Yeming Wen, Paul Vicol, Jimmy Ba, Dustin Tran, and Roger Grosse. Flipout: Efficient pseudo-independent weight perturbations on mini-batches. *arXiv preprint arXiv:1803.04386*, 2018.

[116] Ronald. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[117] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. Mitigating adversarial effects through randomization. *arXiv preprint arXiv:1711.01991*, 2017.

[118] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Yuyin Zhou, Lingxi Xie, and Alan Yuille. Adversarial examples for semantic segmentation and object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1369–1378, 2017.

[119] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

[120] Neha Yadav, Anupam Yadav, and Manoj Kumar. *An introduction to neural network methods for differential equations.* Springer, 2015.

[121] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems*, 2019.

[122] Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4694–4702, 2015.

[123] Guoming Zhang, Chen Yan, Xiaoyu Ji, Tianchen Zhang, Taimin Zhang, and Wenyuan Xu. Dolphinattack: Inaudible voice commands. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 103–117. ACM, 2017.

[124] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[125] Andrea Zunino, Sarah Adel Bargal, Pietro Morerio, Jianming Zhang, Stan Sclaroff, and Vittorio Murino. Excitation dropout: Encouraging plasticity in deep neural networks. *arXiv preprint arXiv:1805.09092*, 2018.

Appendices

Appendix A

Confidence level

In this thesis, confidence levels for each model are checked in Chapter 3. The specific perturbation levels are shown in the chapter, so here, the results at all of the perturbation levels are shown. At first, the confidence levels for the dropout are shown, L1 and L2, and binarized neural networks models are shown.

A.1 Confidence level on changing dropout probability

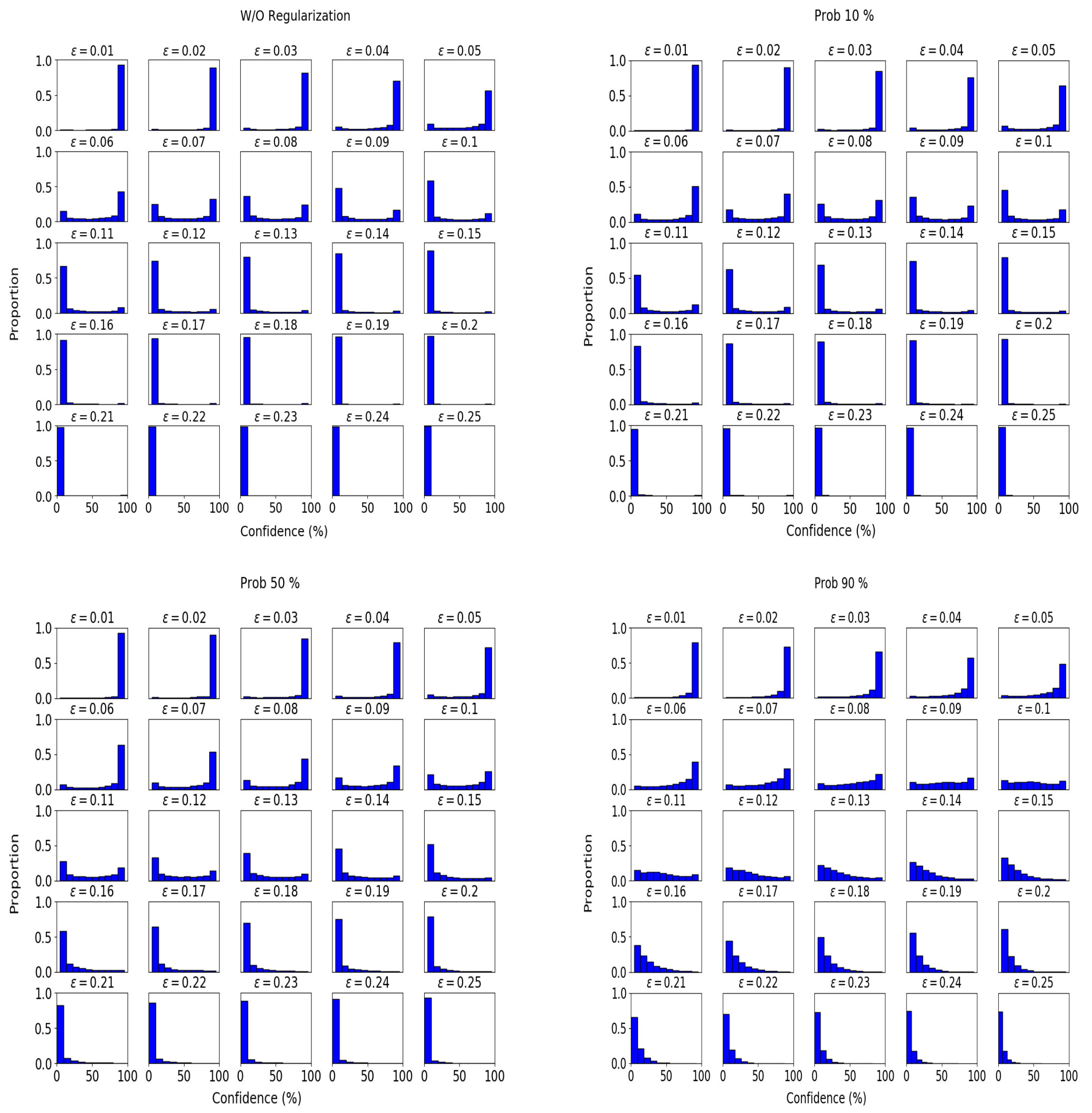


Figure A.1: Proportion (0 to 1) vs confidence level (0-100) for target classes of adversarial examples on changing dropout probability experiments (without any regularizations, 10, 50, and 90 % probabilities). ϵ means perturbation level (0.01 to 0.25). Proportion 0 means none of the examples, and 1 means all of the examples. Also, Confidence level 0 means 0 % confidence for the target class, and 100 means 100 % confidence for the target class.

A.2 Confidence level on changing L1 and L2 λ parameters

A.2.1 L1 weight decay

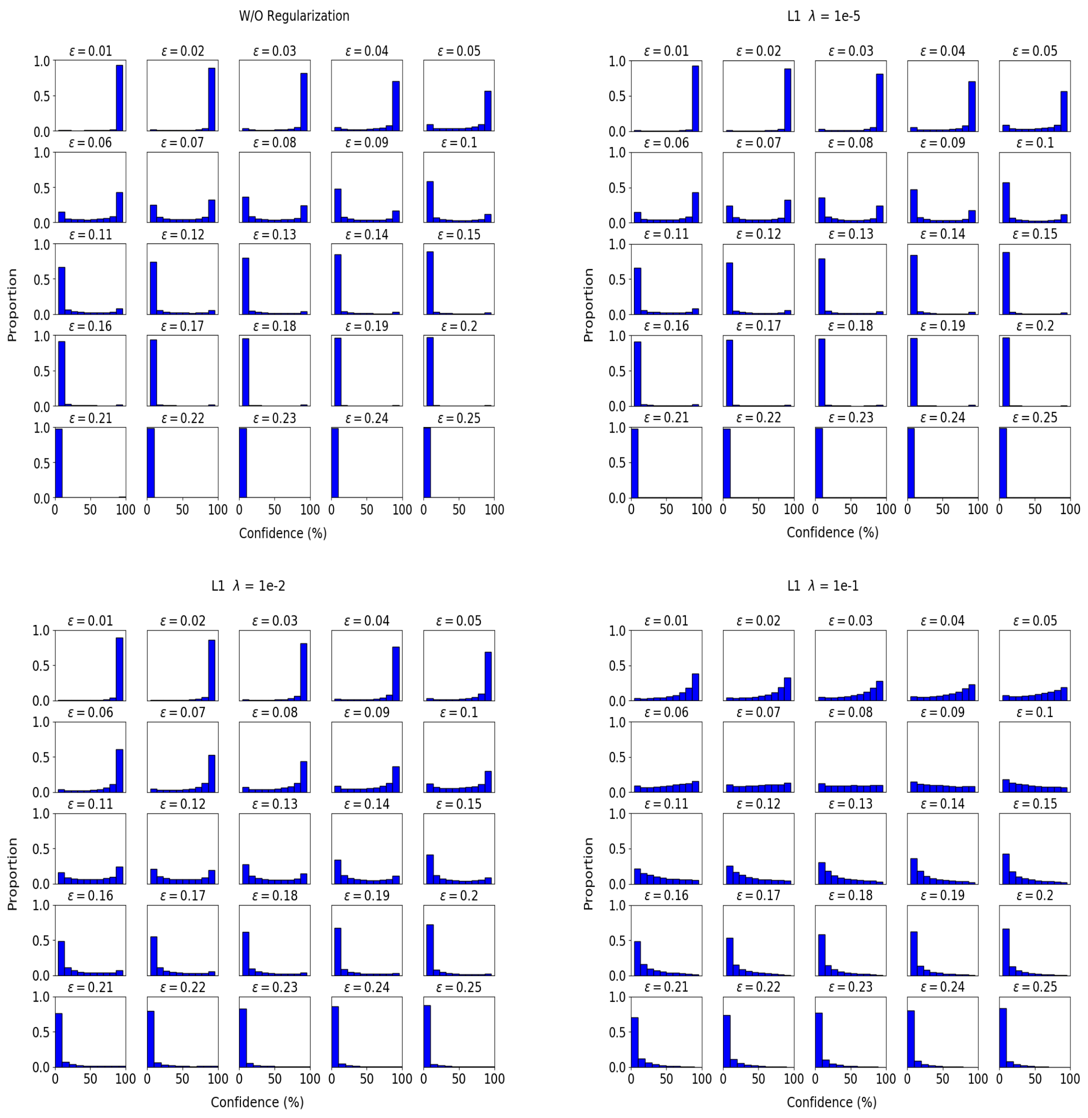


Figure A.2: Proportion (0 to 1) vs confidence level (0-100) for target classes of adversarial examples on changing L1 λ experiments (without any regularizations, $\lambda = 1e-5, 1e-2, 1e-1$). The confidence levels for W/O any regularizations and $\lambda = 1e-5$ change very similarly. At $\epsilon = 0.15$, most of both confidence levels are zeros. On the other hand, confidence levels for $\lambda = 1e-2, 1e-1$ gradually change from high to low.

A.2.2 L2 weight decay

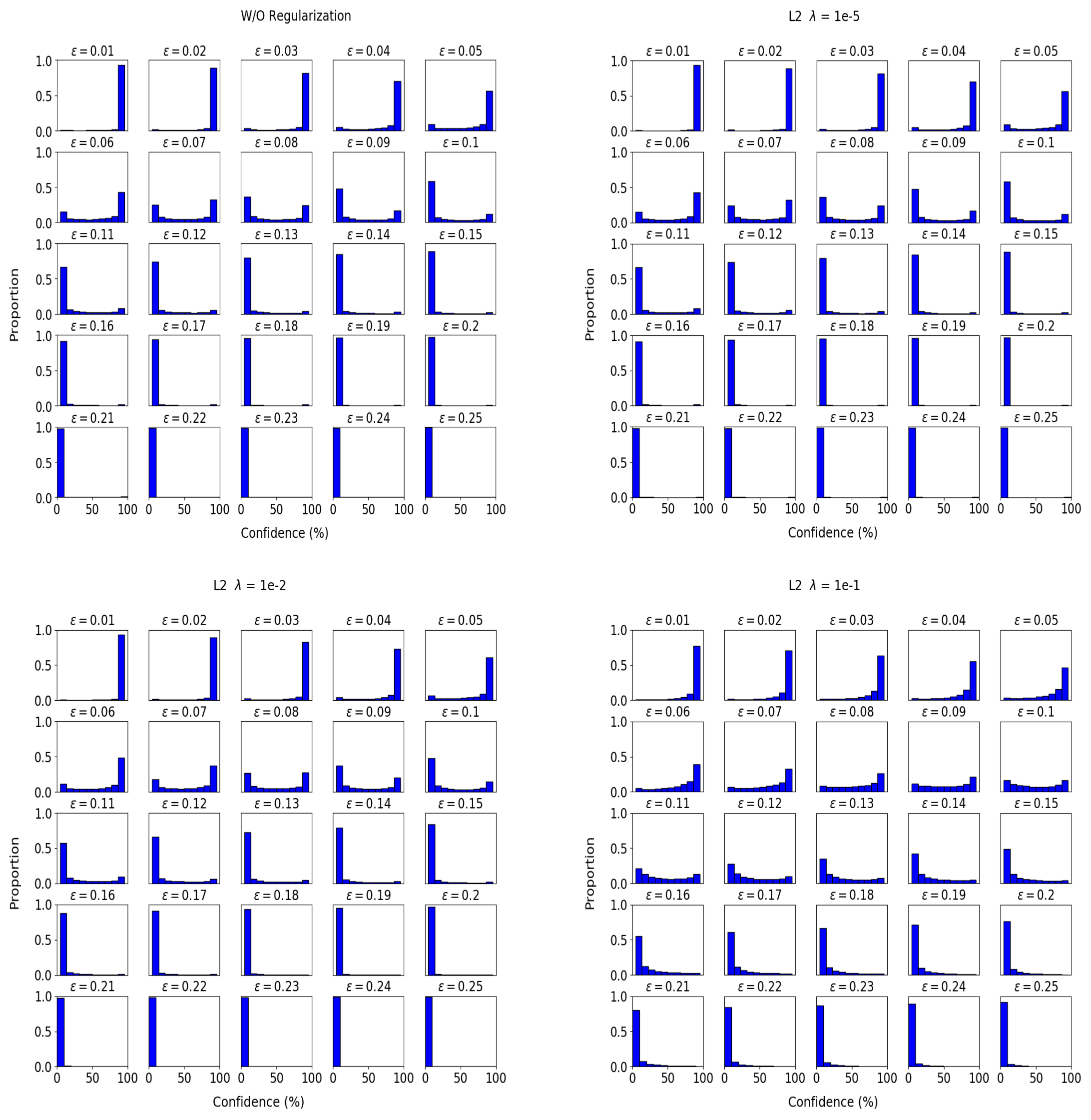


Figure A.3: Proportion (0 to 1) vs confidence level (0-100) for target classes of adversarial examples on changing L2 λ experiments (without any regularizations, $\lambda = 1e-5, 1e-2, 1e-1$). These results are similar to L1. The high λ gradually change the confidence levels from high to lower, but the lower change the confidence levels rapidly.

A.3 Confidence levels on binarized neural networks

A.3.1 MLP

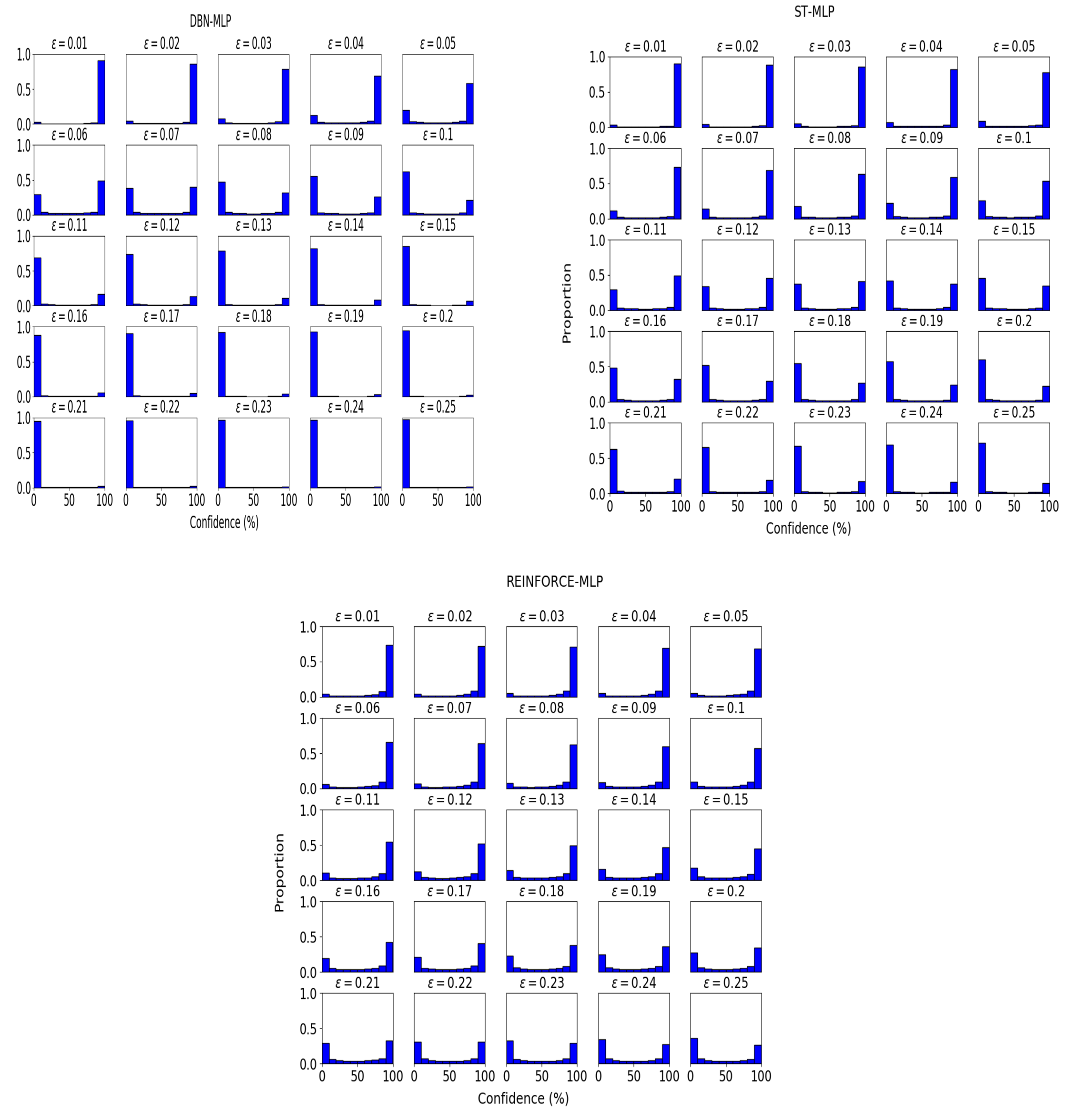


Figure A.4: Proportion (0 to 1) vs confidence level (0-100) for target classes on stochastic and deterministic binarized neuron experiments for MLP.

A.3.2 1CNN

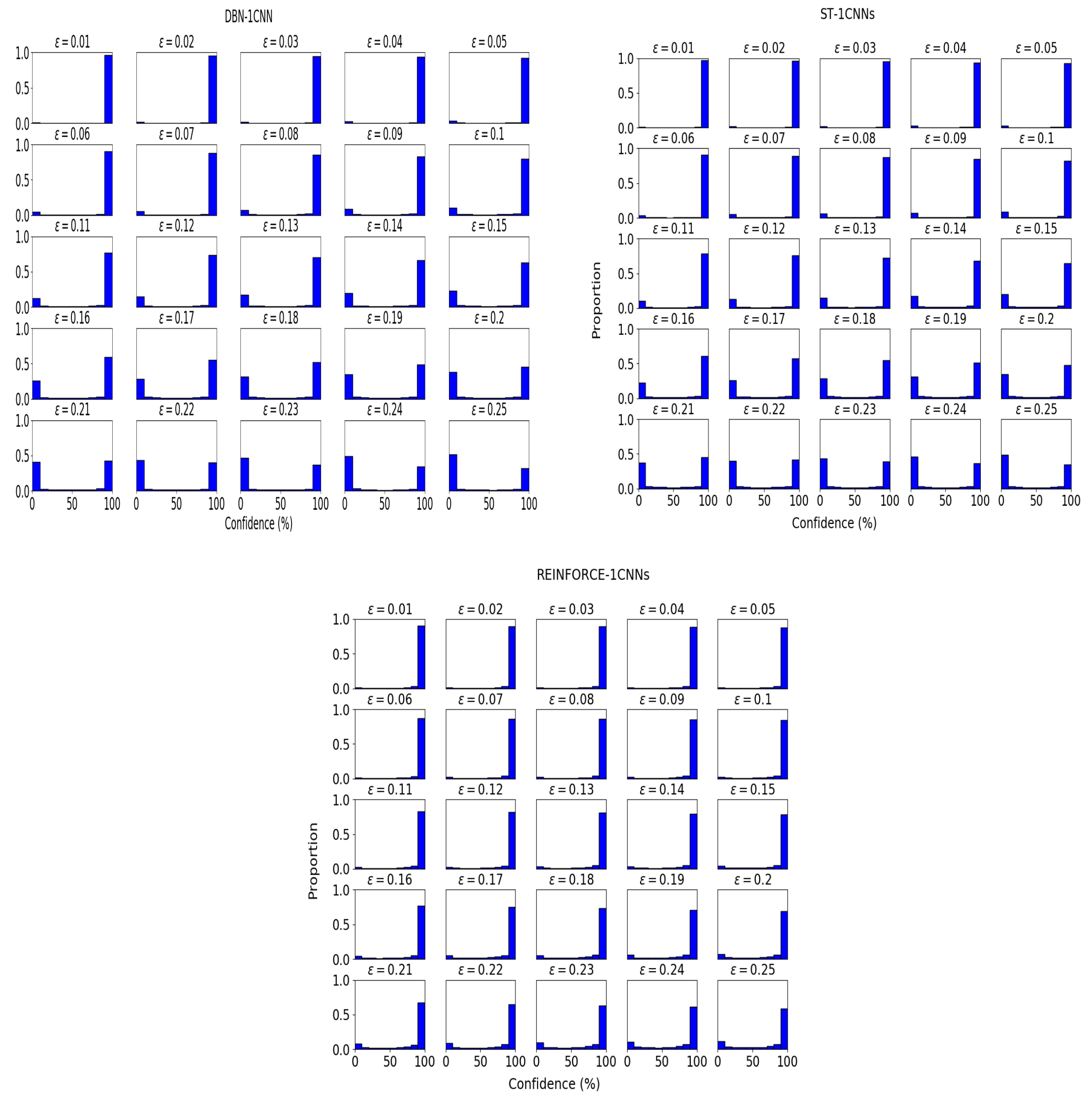


Figure A.5: Proportion (0 to 1) vs confidence level (0-100) for target classes on stochastic and deterministic binarized neuron experiments for 1CNN. DBN and ST 1CNNs have very similar behaviors. Most of the confidence levels at a lower ϵ are 100 %. while increasing ϵ, the proportions for DBN and ST are divided into two, close to 0 or 100%, even at $\epsilon = 0.25$. The proportions of the middle of confidence levels are less. On the other hand, REINFORCE-1CNN are most robust among the models. The proportions of high confidence levels for REINFORCE gradually decrease but not too much.

A.3.3 2CNN

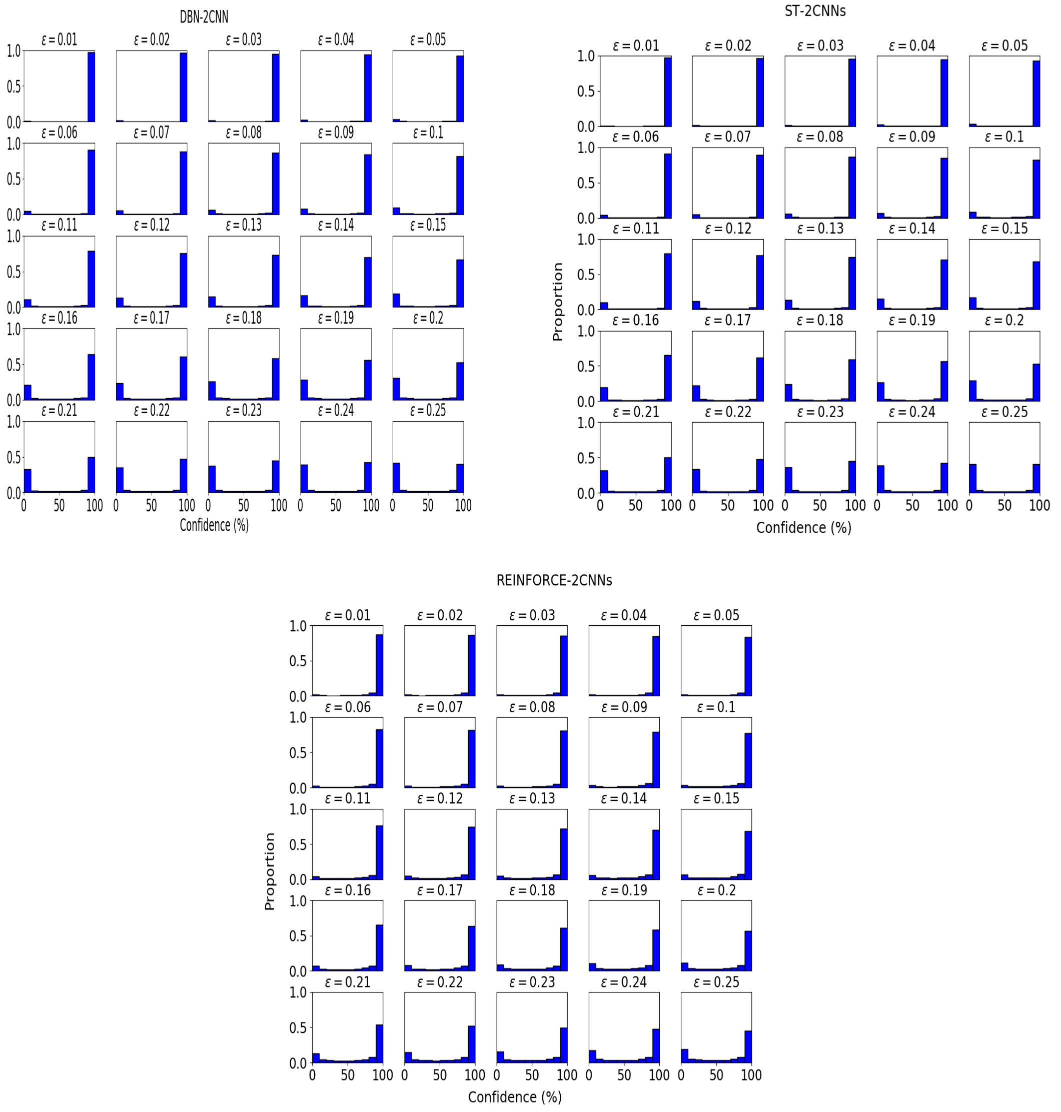


Figure A.6: Proportion (0 to 1) vs confidence level (0-100) for target classes on stochastic and deterministic binarized neuron experiments for 2CNN. Similar to 1CNN result, DBN and ST 2CNNs are very similar behaviors. The proportions for REINFORCE at high confidence levels are more robust than the others in this experiment too.

Appendix B

The activations ($\sigma(a)$) of the gate module for the 2nd layer before binarized (probabilities of masking) on the testing dataset.

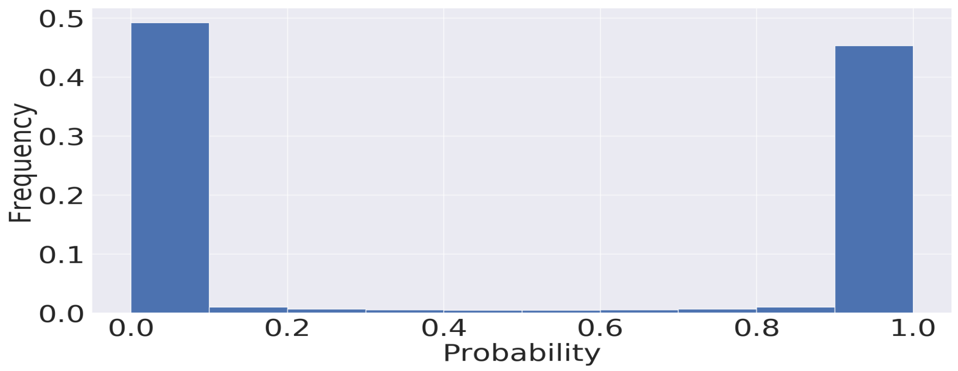


Figure B.1: Histogram of probabilities of activations ($\sigma(a)$) from the gate module (2nd layer) before they are binarized (scale is 0 to 1). Around 50% of the activations are close to 1, and around 50 % are close to 0.

Appendix C

The differences in accuracy between the training and testing MNIST datasets (clean datasets) for SGBN and SGBN with relu activation

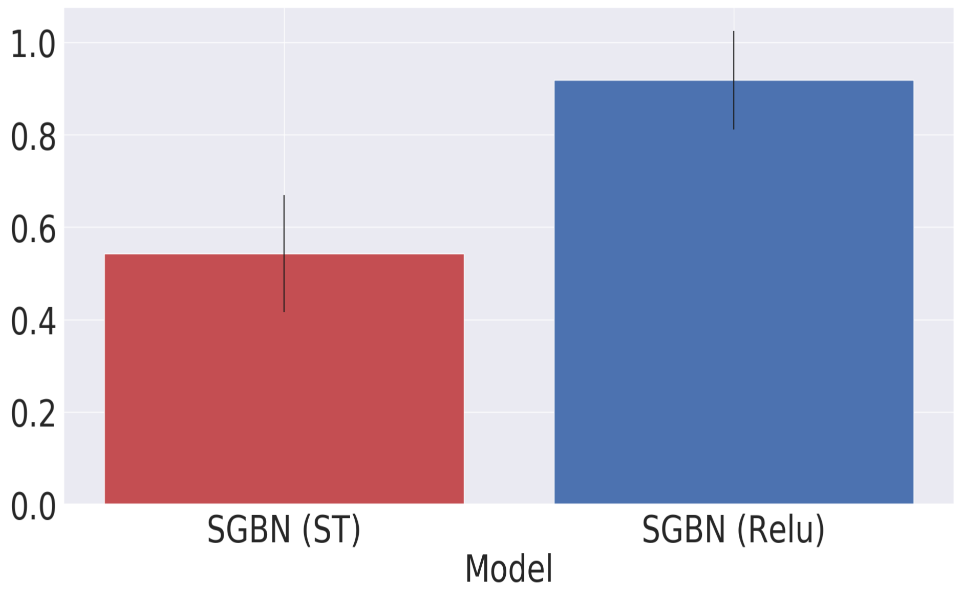


Figure C.1: The differences in accuracy between the training and testing MNIST datasets (clean datasets) for SGBN and SGBN with relu activation

Appendix D

Adversarial examples crafted by the gate module and accuracies of each model on the examples

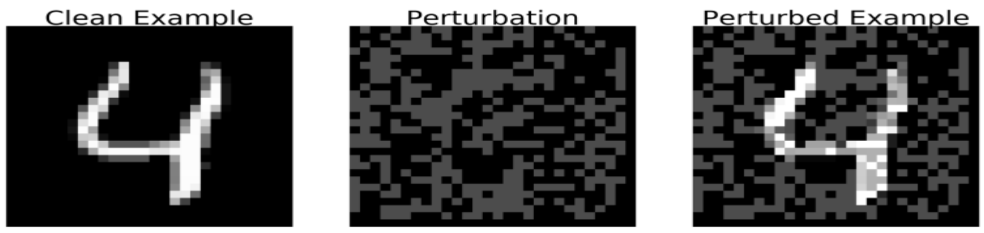


Figure D.1: An adversarial example crafted by the first gate module.

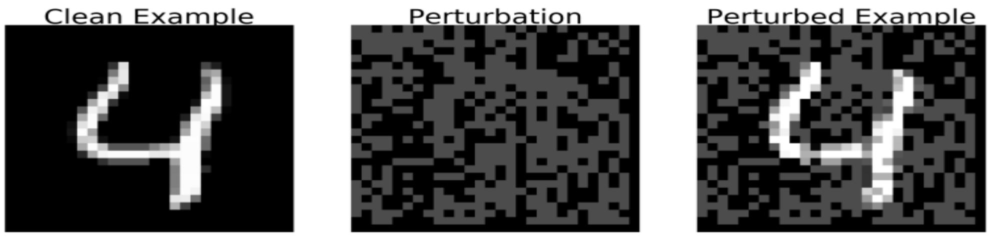


Figure D.2: An adversarial example crafted by the second gate module.

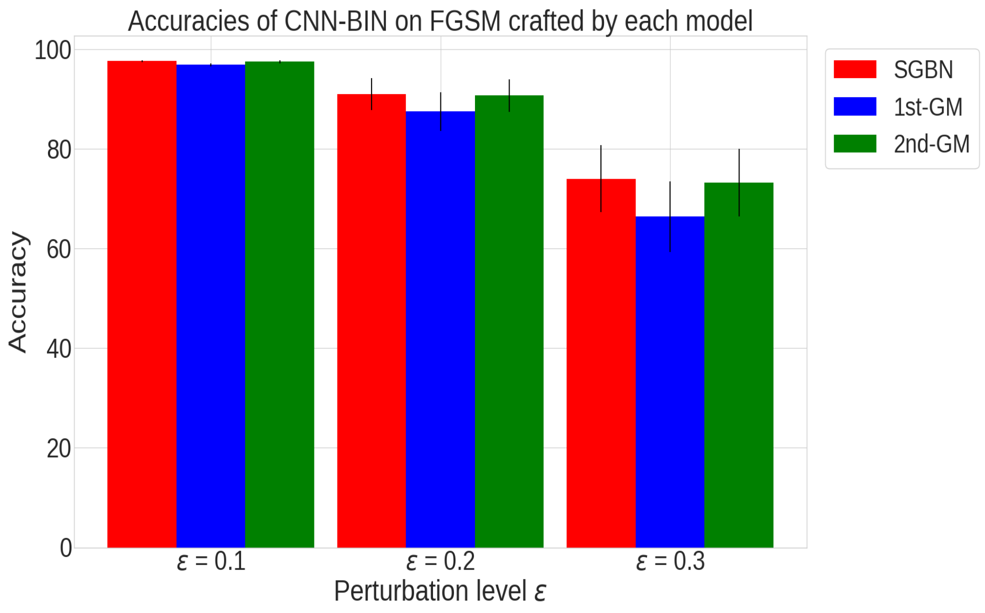


Figure D.3: The accuracies of CNN-BIN on FGSM crafted by SGBN (red), the first (blue), and the second gate modules (green).

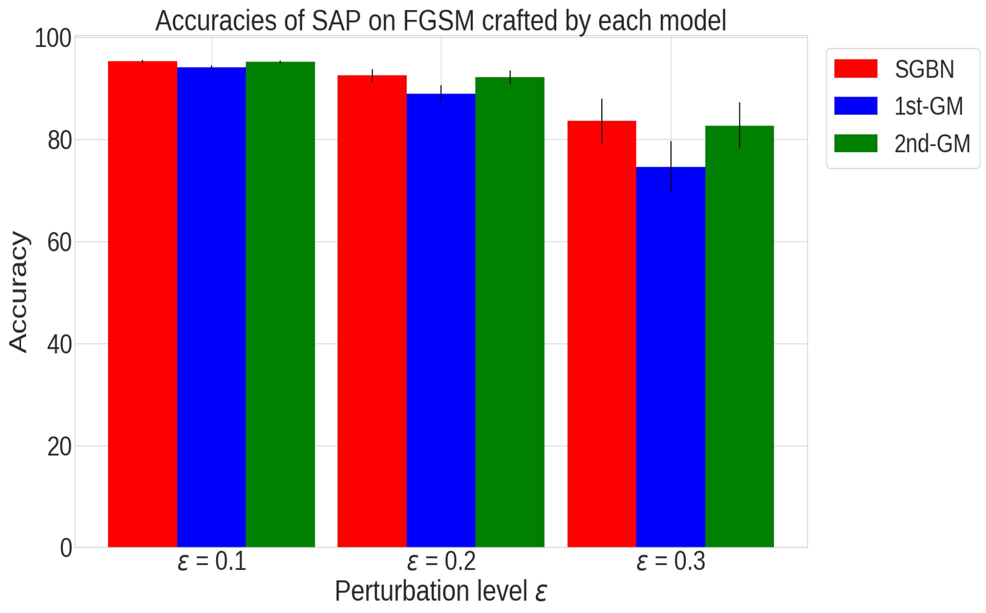


Figure D.4: The accuracies of SAP on FGSM crafted by SGBN (red), the first (blue), and the second gate modules (green).

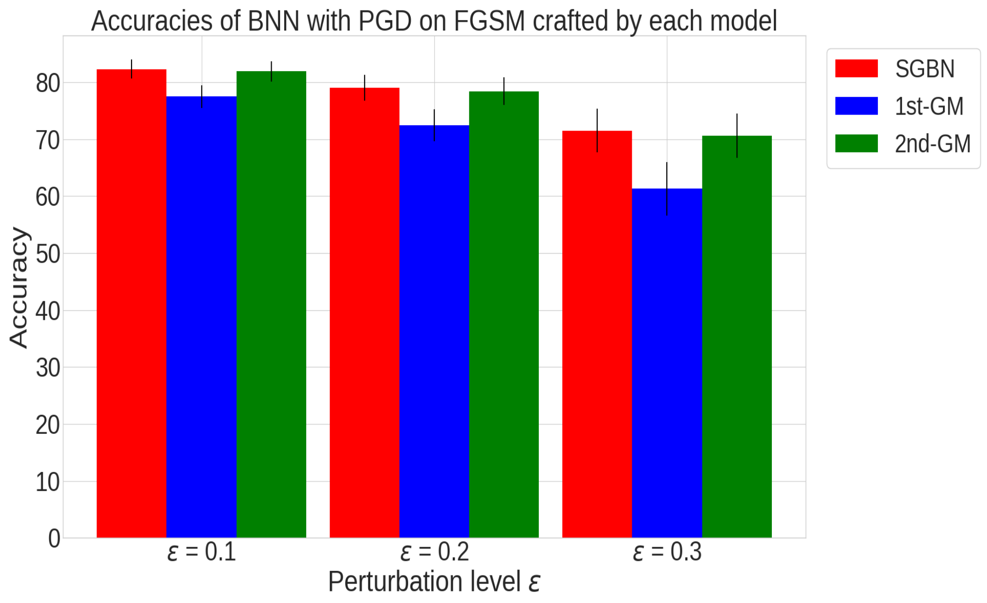


Figure D.5: The accuracies of BNN with PGD on FGSM crafted by SGBN (red), the first (blue), and the second gate modules (green).

Appendix E

Notices of permission to use excerpts from author's publications

In this thesis, large and small excerpts were taken verbatim from two of the author's own published papers [60, 61]. A form of the student's contribution to the manuscript is signed and submitted to the graduate studies office.

Both Springer (publishers of the Canadian Conference on Artificial Intelligence) and IEEE (publisher of the proceedings of the International Joint Conference on Neural Networks) state in the documents reproduced on the pages linked below that use of the author's work in their own dissertation is allowed.

Please see the copyright forms for permissions:

Consent to Publish

Lecture Notes in Computer Science

Title of the Book or Conference Name: 32nd Canadian Conference on Artificial Intelligence

Volume Editor(s) Name(s): Frank Rudzicz and Marie-Jean Meurs

Title of the Contribution: Mitigating Overfitting Using Regularization to Defend Networks Against Adversarial Examples

Author(s) Full Name(s): Yoshimasa Kubo and Thomas Trappenberg

Corresponding Author's Name, Affiliation Address, and Email:
Yoshimasa Kubo, Dalhousie University, Halifax Canada, yoshi.with.ai@gmail.com

When Author is more than one person the expression "Author" as used in this agreement will apply collectively unless otherwise indicated.

The Publisher intends to publish the Work under the imprint **Springer**. The Work may be published in the book series **Lecture Notes in Computer Science (LNCS, LNAI or LNBI)**.

§ 1 Rights Granted

Author hereby grants and assigns to **Springer Nature Switzerland AG, Gewerbestrasse 11, 6330 Cham, Switzerland** (hereinafter called **Publisher**) the exclusive, sole, permanent, world-wide, transferable, sub-licensable and unlimited right to reproduce, publish, distribute, transmit, make available or otherwise communicate to the public, translate, publicly perform, archive, store, lease or lend and sell the Contribution or parts thereof individually or together with other works in any language, in all revisions and versions (including soft cover, book club and collected editions, anthologies, advance printing, reprints or print to order, microfilm editions, audiograms and videograms), in all forms and media of expression including in electronic form (including offline and online use, push or pull technologies, use in databases and data networks (e.g. the Internet) for display, print and storing on any and all stationary or portable end-user devices, e.g. text readers, audio, video or interactive devices, and for use in multimedia or interactive versions as well as for the display or transmission of the Contribution or parts thereof in data networks or search engines, and posting the Contribution on social media accounts closely related to the Work), in whole, in part or in abridged form, in each case as now known or developed in the future, including the right to grant further time-limited or permanent rights. Publisher especially has the right to permit others to use individual illustrations, tables or text quotations and may use the Contribution for advertising purposes. For the purposes of use in electronic forms, Publisher may adjust the Contribution to the respective form of use and include links (e.g. frames or inline-links) or otherwise combine it with other works and/or remove links or combinations with other works provided in the Contribution. For the avoidance of doubt, all provisions of this contract apply regardless of whether the Contribution and/or the Work itself constitutes a database under applicable copyright laws or not.
The copyright in the Contribution shall be vested in the name of Publisher. Author has asserted his/her right(s) to be identified as the originator of this Contribution in all editions and versions of the Work and parts thereof, published in all forms and media. Publisher may take, either in its own name or in that of Author, any necessary steps to protect the rights granted under this Agreement against infringement by third parties. It will have a copyright notice inserted into all editions of the Work according to the provisions of the Universal Copyright Convention (UCC).
The parties acknowledge that there may be no basis for claim of copyright in the United States to a Contribution prepared by an officer or employee of the United States government as part of that person's official duties. If the Contribution was performed under a United States government contract, but Author is not a United States government employee, Publisher grants the United States government royalty-free permission to reproduce all or part of the Contribution and to authorise others to do so for United States government purposes. If the Contribution was prepared or published by or under the direction or control of the Crown (i.e., the constitutional monarch of the Commonwealth realm) or any Crown government department, the copyright in the Contribution shall, subject to any agreement with Author, belong to the Crown. If Author is an officer or employee of the United States government or of the Crown, reference will be made to this status on the signature page.

§ 2 Rights Retained by Author

Author retains, in addition to uses permitted by law, the right to communicate the content of the Contribution to other research colleagues, to share the Contribution with them in manuscript form, to perform or present the Contribution or to use the content for non-commercial internal and educational purposes, provided the original source of publication is cited according to the current citation standards in any printed or electronic materials. Author retains the right to republish the Contribution in any collection consisting solely of Author's own works without charge, subject to ensuring that the publication of the Publisher is properly credited and that the relevant copyright notice is repeated verbatim. Author may self-archive an author-created version of his/her Contribution on his/her own website and/or the repository of Author's department or faculty. Author may also deposit this version on his/her funder's or funder's designated repository at the funder's request or as a result of a legal obligation. He/she may not use the Publisher's PDF version, which is posted on the Publisher's platforms, for the purpose of self-archiving or deposit. Furthermore, Author may only post his/her own version, provided acknowledgment is given to the original source of publication and a link is inserted to the published article on the Publisher's website. The link must be provided by inserting the DOI number of the article in the following sentence: "The final authenticated version is available online at https://doi.org/[insert DOI]." The DOI (Digital Object Identifier) can be found at the bottom of the first page of the published paper.
Prior versions of the Contribution published on non-commercial pre-print servers like ArXiv/CoRR and HAL can remain on these servers and/or can be updated with Author's accepted version. The final published version (in pdf or html/xml format) cannot be used for this purpose. Acknowledgment needs to be given to the final publication and a link must be inserted to the published Contribution on the Publisher's website, by inserting the DOI number of the article in the following sentence: "The final authenticated publication is available online at https://doi.org/[insert DOI]".
Author retains the right to use his/her Contribution for his/her further scientific career by including the final published paper in his/her dissertation or doctoral thesis provided acknowledgment is given to the original source of publication. Author also retains the right to use, without having to pay a fee and without having to inform the Publisher, parts of the Contribution (e.g. illustrations) for inclusion in future work. Authors may publish an extended version of their proceedings paper as a journal article provided the following principles are adhered to: a) the extended version includes at least 30% new material, b) the original publication is cited, and c) it includes an explicit statement about the increment (e.g., new results, better description of materials, etc.).

§ 3 Warranties

Author agrees, at the request of Publisher, to execute all documents and do all things reasonably required by Publisher in order to confer to Publisher all rights intended to be granted under this Agreement. Author warrants that the Contribution is original except for such excerpts from copyrighted works (including illustrations, tables, animations and text quotations) as may be included with the permission of the copyright holder thereof, in which case(s) Author is required to obtain written permission to the extent necessary and to indicate the precise sources of the excerpts in the manuscript. Author is also requested to store the signed permission forms and to make them available to Publisher if required.
Author warrants that Author is entitled to grant the rights in accordance with Clause 1 "Rights Granted", that Author has not assigned such rights to third parties, that the Contribution has not heretofore been published in whole or in part, that the Contribution contains no libellous or defamatory statements and does not infringe on any copyright, trademark, patent, statutory right or proprietary right of others, including rights obtained through licences; and that Author will indemnify Publisher against any costs, expenses or damages for which Publisher may become liable as a result of any claim which, if true, would constitute a breach by Author of any of Author's representations or warranties in this Agreement.
Author agrees to amend the Contribution to remove any potential obscenity, defamation, libel, malicious falsehood or otherwise unlawful part(s) identified at any time. Any such removal or alteration shall not affect the warranty and indemnity given by Author in this Agreement.

§ 4 Delivery of Contribution and Publication

Author agrees to deliver to the responsible Volume Editor (for conferences, usually one of the Program Chairs), on a date to be agreed upon, the manuscript created according to the Publisher's Instructions for Authors. Publisher will undertake the reproduction and distribution of the Contribution at its own expense and risk. After submission of the Consent to Publish form signed by the Corresponding Author, changes of authorship, or in the order of the authors listed, will not be accepted by the Publisher.

§ 5 Author's Discount for Books

Author is entitled to purchase for his/her personal use (if ordered directly from Publisher) the Work or other books published by Publisher at a discount of 40% off the list price for as long as there is a contractual arrangement between Author and Publisher and subject to applicable book price regulation.
Resale of such copies is not permitted.

§ 6 Governing Law and Jurisdiction

If any difference shall arise between Author and Publisher concerning the meaning of this Agreement or the rights and liabilities of the parties, the parties shall engage in good faith discussions to attempt to seek a mutually satisfactory resolution of the dispute. This agreement shall be governed by, and shall be construed in accordance with, the laws of Switzerland. The courts of Zug, Switzerland shall have the exclusive jurisdiction.

Corresponding Author signs for and accepts responsibility for releasing this material on behalf of any and all Co-Authors.

Signature of Corresponding Author: **Date:**

.... 03/08/2019 ....

☐ I'm an employee of the US Government and transfer the rights to the extent transferable (Title 17 §105 U.S.C. applies)

☐ I'm an employee of the Crown and copyright on the Contribution belongs to the Crown

For internal use only:
Legal Entity Number: 1128 Springer Nature Switzerland AG
Springer-C-CTP-01/2018

IEEE COPYRIGHT AND CONSENT FORM

To ensure uniformity of treatment among all contributors, other forms may not be substituted for this form, nor may any wording of the form be changed. This form is intended for original material submitted to the IEEE and must accompany any such material in order to be published by the IEEE. Please read the form carefully and keep a copy for your files.

Learning Adaptive Weight Masking for Adversarial Examples
Yoshimasa Kubo, Michael Traynor, Thomas Trappenberg and Sageev Oore
2019 International Joint Conference on Neural Networks (IJCNN)

COPYRIGHT TRANSFER

The undersigned hereby assigns to The Institute of Electrical and Electronics Engineers, Incorporated (the "IEEE") all rights under copyright that may exist in and to: (a) the Work, including any revised or expanded derivative works submitted to the IEEE by the undersigned based on the Work; and (b) any associated written or multimedia components or other enhancements accompanying the Work.

GENERAL TERMS

1. The undersigned represents that he/she has the power and authority to make and execute this form.
2. The undersigned agrees to indemnify and hold harmless the IEEE from any damage or expense that may arise in the event of a breach of any of the warranties set forth above.
3. The undersigned agrees that publication with IEEE is subject to the policies and procedures of the IEEE PSPB Operations Manual.
4. In the event the above work is not accepted and published by the IEEE or is withdrawn by the author(s) before acceptance by the IEEE, the foregoing copyright transfer shall be null and void. In this case, IEEE will retain a copy of the manuscript for internal administrative/record-keeping purposes.
5. For jointly authored Works, all joint authors should sign, or one of the authors should sign as authorized agent for the others.
6. The author hereby warrants that the Work and Presentation (collectively, the "Materials") are original and that he/she is the author of the Materials. To the extent the Materials incorporate text passages, figures, data or other material from the works of others, the author has obtained any necessary permissions. Where necessary, the author has obtained all third party permissions and consents to grant the license above and has provided copies of such permissions and consents to IEEE

You have indicated that you DO wish to have video/audio recordings made of your conference presentation under terms and conditions set forth in "Consent and Release."

CONSENT AND RELEASE

1. In the event the author makes a presentation based upon the Work at a conference hosted or sponsored in whole or in part by the IEEE, the author, in consideration for his/her participation in the conference, hereby grants the IEEE the unlimited, worldwide, irrevocable permission to use, distribute, publish, license, exhibit, record, digitize, broadcast, reproduce and archive, in any format or medium, whether now known or hereafter developed: (a) his/her presentation and comments at the conference; (b) any written materials or multimedia files used in connection with his/her presentation; and (c) any recorded interviews of him/her (collectively, the "Presentation"). The permission granted includes the transcription and reproduction of the Presentation for inclusion in products sold or distributed by IEEE and live or recorded broadcast of the Presentation during or after the conference.
2. In connection with the permission granted in Section 1, the author hereby grants IEEE the unlimited, worldwide, irrevocable right to use his/her name, picture, likeness, voice and biographical information as part of the advertisement, distribution and sale of products incorporating the Work or Presentation, and releases IEEE from any claim based on right of privacy or publicity.

BY TYPING IN YOUR FULL NAME BELOW AND CLICKING THE SUBMIT BUTTON, YOU CERTIFY THAT SUCH ACTION CONSTITUTES YOUR ELECTRONIC SIGNATURE TO THIS FORM IN ACCORDANCE WITH UNITED STATES LAW, WHICH AUTHORIZES ELECTRONIC SIGNATURE BY AUTHENTICATED REQUEST FROM A USER OVER THE INTERNET AS A VALID SUBSTITUTE FOR A WRITTEN SIGNATURE.

	31-03-2019
Signature	**Date (dd-mm-yyyy)**

Information for Authors

AUTHOR RESPONSIBILITIES

The IEEE distributes its technical publications throughout the world and wants to ensure that the material submitted to its publications is properly available to the readership of those publications. Authors must ensure that their Work meets the requirements as stated in section 8.2.1 of the IEEE PSPB Operations Manual, including provisions covering originality, authorship, author responsibilities and author misconduct. More information on IEEE's publishing policies may be found at http://www.ieee.org/publications_standards/publications/rights/authorrightsresponsibilities.html Authors are advised especially of IEEE PSPB Operations Manual section 8.2.1.B12: "It is the responsibility of the authors, not the IEEE, to determine whether disclosure of their material requires the prior consent of other parties and, if so, to obtain it." Authors are also advised of IEEE PSPB Operations Manual section 8.1.1B: "Statements and opinions given in work published by the IEEE are the expression of the authors."

RETAINED RIGHTS/TERMS AND CONDITIONS

- Authors/employers retain all proprietary rights in any process, procedure, or article of manufacture described in the Work.
- Authors/employers may reproduce or authorize others to reproduce the Work, material extracted verbatim from the Work, or derivative works for the author's personal use or for company use, provided that the source and the IEEE copyright notice are indicated, the copies are not used in any way that implies IEEE endorsement of a product or service of any employer, and the copies themselves are not offered for sale.
- Although authors are permitted to re-use all or portions of the Work in other works, this does not include granting third-party requests for reprinting, republishing, or other types of re-use.The IEEE Intellectual Property Rights office must handle all such third-party requests.
- Authors whose work was performed under a grant from a government funding agency are free to fulfill any deposit mandates from that funding agency.

AUTHOR ONLINE USE

- **Personal Servers**. Authors and/or their employers shall have the right to post the accepted version of IEEE-copyrighted articles on their own personal servers or the servers of their institutions or employers without permission from IEEE, provided that the posted version includes a prominently displayed IEEE copyright notice and, when published, a full citation to the original IEEE publication, including a link to the article abstract in IEEE Xplore. Authors shall not post the final, published versions of their papers.
- **Classroom or Internal Training Use.** An author is expressly permitted to post any portion of the accepted version of his/her own IEEE-copyrighted articles on the author's personal web site or the servers of the author's institution or company in connection with the author's teaching, training, or work responsibilities, provided that the appropriate copyright, credit, and reuse notices appear prominently with the posted material. Examples of permitted uses are lecture materials, course packs, e-reserves, conference presentations, or in-house training courses.
- **Electronic Preprints.** Before submitting an article to an IEEE publication, authors frequently post their manuscripts to their own web site, their employer's site, or to another server that invites constructive comment from colleagues. Upon submission of an article to IEEE, an author is required to transfer copyright in the article to IEEE, and the author must update any previously posted version of the article with a prominently displayed IEEE copyright notice. Upon publication of an article by the IEEE, the author must replace any previously posted electronic versions of the article with either (1) the full citation to the