Rehana Mahfuz (rmahfuz@purdue.edu)
BME 595 Project Report

Mitigating the effect of Adversarial Attacks on Neural Networks

Introduction

Neural networks are vulnerable to adversarial attacks [1]. There are two kinds of attacks. Poisoning attacks [2] corrupt training data so that the neural network is incorrectly trained. Evasion attacks corrupt test data, so that the output of the neural network for that test sample is incorrect. Among the many types of evasion attacks, one type is gradient-based attacks, which add to the data some noise which is proportional to the sign of the gradient of the cost function. The Fast Gradient Sign (FGS) attack [1] adds a perturbation which is l_∞ bounded:

$$\tilde{x}(\epsilon) = x + \epsilon * sign\left(\nabla_x J(\theta, x, y)\right)$$

$\tilde{x}(\epsilon)$ is the perturbed version of the data sample x, the perturbation being bounded by ϵ. θ is the trained weights, y is the label corresponding to x, and $J(\theta, x, y)$ is the loss function.

A variation of FGS is the Fast Gradient (FG) attack, which adds a perturbation bounded by the l_2 norm:

$$\tilde{x}(\epsilon) \;=\; x \;+\; \epsilon * \frac{\nabla_x\, J(\theta, x, y)}{||\nabla_x\, J(\theta, x, y)||}$$

In an attempt to make the perturbation imperceptible, the value of ϵ is kept low by the adversary.

There are different scenarios based on how much information the adversary has about the neural network classifier. If the adversary has access to the trained classifier and has knowledge of the defense strategy, that is the white-box scenario. The adversary having so much knowledge is very unrealistic, which is why I have not considered this scenario. If the adversary has access to the trained classifier only, that is a semi-white box scenario. The adversary uses this trained classifier to generate perturbations. If the adversary has access to the training dataset but not to the trained classifier, that is a black box scenario. In this case, since the adversary does not even know the classifier architecture, it trains its own classifier which does a good job of classifying the training data, and generates perturbations based on this classifier.

Procedure

This project is an attempt to devise a strategy to combat the effect of these adversarial attacks in the semi-white box and black box scenarios. Since we do not know which bound the adversary will attack our data with, we cannot just calculate the perturbation and then subtract it from the test data. Also, we need to find a strategy which would not hurt the classification accuracy if the data is not attacked. That leads us to think that we need to make the classifier learn the structure of the data, which would stay approximately the same even if a small perturbation was added to it.

My strategy is to reduce the dimension of the data before passing it through the classifier, so that the manifold of the data is retained. I do this dimensionality reduction using an autoencoder. An autoencoder is a neural network which tries to find an output approximately the same as the input. The hidden layers are of a different dimension than the input, which is why the autoencoder cannot simply copy the input to the output. If the hidden layers have lesser neurons that the input layer, the autoencoder will be forced to learn a compressed representation of the data. The output of the hidden layer of a trained autoencoder gives us this compressed representation.

Rehana Mahfuz (rmahfuz@purdue.edu)
BME 595 Project Report

My defense strategy involves first training an autoencoder with the training data. Then the compressed representation of the training data is obtained, which is used to train the classifier. An important point to note is that the classifier architecture changes since the number of input dimensions changes. While testing, the test data sample is passed through the autoencoder to obtain the compressed representation, which is input into the classifier.

Implementation details

I used the MNIST dataset which consists of 60,000 training images are 10,000 testing images of 28 x 28 pixel grayscale handwritten digits, each image labelled by a digit between 0 and 9. The classifier I used is FC-784-100-100-10, which is a fully connected neural network with an input layer of size 784, followed by two layers of 100 neurons each, followed by an output layer of 10 neurons. The output is one-hot encoded. In a black box scenario, the adversary uses an FC-784-200-200-100-10 architecture to generate the perturbations. The naming convention is similar to that of the former classifier. The reason I chose a more complicated architecture for the adversary on purpose, because I wanted to see how well my defense does against an attack crafted by a more complex architecture.

The autoencoder that I use has an architecture 784-*k*-784, where k is the number of dimensions of the hidden layer. I consider *k* values of 331, 100, 80, 60, 40 and 20. For FGS attack, I test my defense against attacks of ϵ values from 0 to 0.5, with a step size of 0.05. The FG attack is weaker, and it requires a larger bound value to create a proportional effect, which is why for FG, I consider ϵ values from 0 to 2.5, with a step size of 0.25.

Results

The dimensionality reduction technique using autoencoders significantly reduced the decrease in accuracy caused due to the adversarial perturbations. Figure 1 shows the results for all four cases: semi-white box FGS, semi-white box FG, black box FGS, black box FG. In general, 80, 40, and 60 dimensions seem to cause the greatest hike in accuracy. More specifically, 80 dimensions does the best. It is interesting to see that not only does this defense not hurt the accuracy in the absence of an attack, this defense improves the accuracy in the absence of an attack, where ϵ is 0.0 in the plots.

Discussion

In the past, Principal Components Analysis (PCA) has been used as a defense [3]. The classifier is trained with the compressed representation found using PCA, and the test data is projected onto the same subspace before being classified. I also implemented this PCA defense for comparison. The performance of my defense is similar to that of PCA, which is not surprising because I only used a single hidden layer for my autoencoder. Using deeper autoencoder architectures will probably create a better defense, and I plan to explore this further.

Something I noticed is that for the adversary to generate perturbations in the semi-white box scenario, it is important for the adversary to have access to the trained classifier, and not just know the classifier architecture. If the adversary trains its own classifier with the same architecture and generates perturbations based on that, the perturbations are not as strong. In fact, the perturbations are similar in strength to those in a black box scenario, where the adversary uses a completely different architecture to generate perturbations.

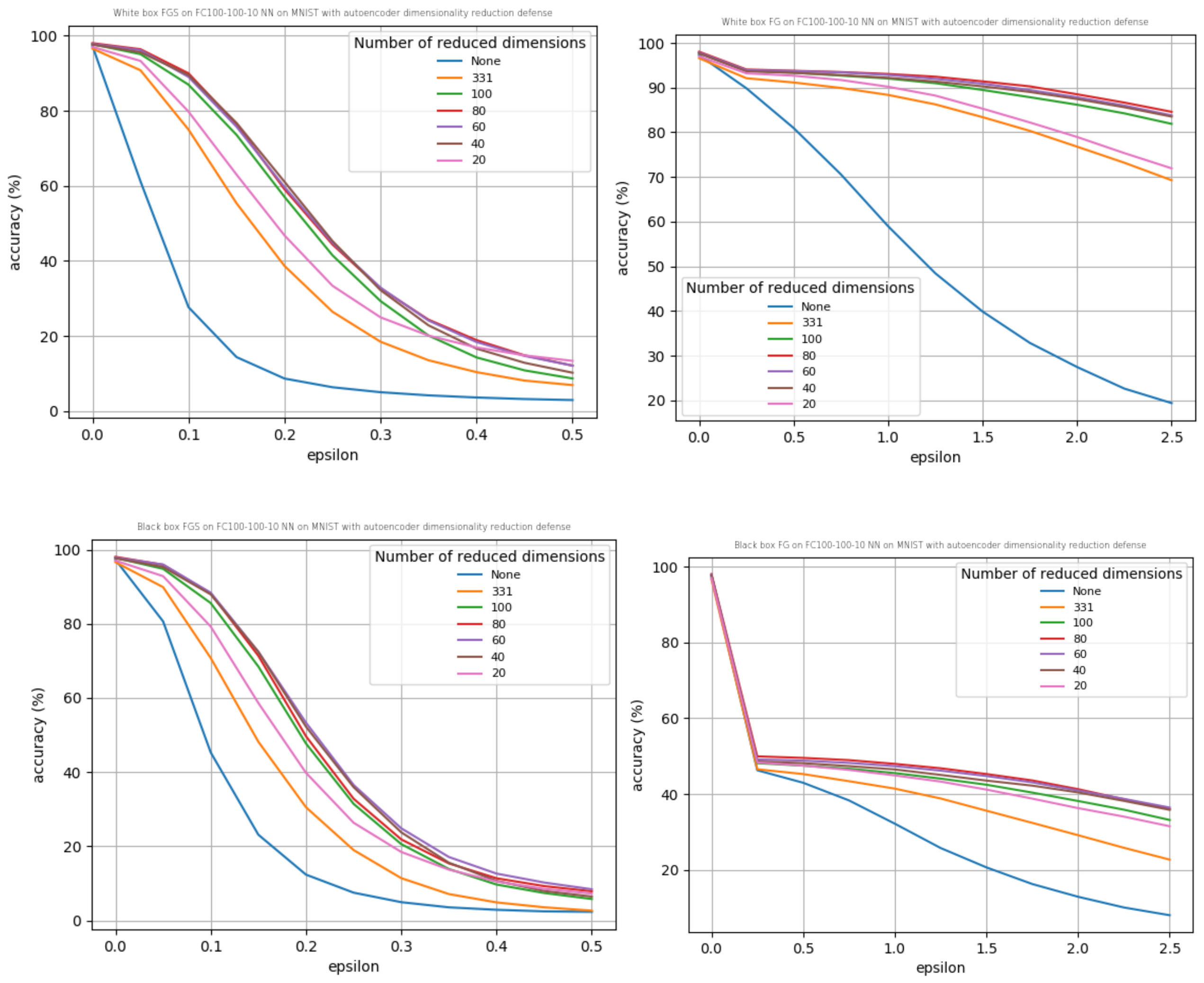

Figure 1: Improvement in accuracies with varying value of perturbation bound for different number of reduced dimensions in all four scenarios.

References

[1] Goodfellow, I., Shlens, J., & Szegedy, C. (2015). Explaining and Harnessing Adversarial Examples. ArXiv.org, ArXiv.org, Mar 20, 2015.

[2] Rubinstein, B., Nelson, L., Joseph, A., Lau, S., Rao, N., Tygar, J., . . . Taft. (2009). Antidote: Understanding and defending against poisoning of anomaly detectors. Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC, 1-14.

[3] Arjun, N., Cullina, D., Sitawarin, C., & Mittal, P. (2017). Enhancing Robustness of Machine Learning Systems via Data Transformations. ArXiv.org, ArXiv.org, Nov 29, 2017.

Rehana Mahfuz (rmahfuz@purdue.edu)
BME 595 Project Report

Appendix: Software details

I used the Keras library in Python to work with neural networks. To generate adversarial examples, I used the Cleverhans library. All neural networks are trained with 100 epochs.

The main function has multiple invocations of the experiment function, each invocation to illustrate one of the four cases. The invocations are in this order: semi-white box FGS, semi-white box FG, black box FGS, black box FG. There are four more invocations to generate results using PCA defense.

The experiment function has a nested loop. The outer loop loops through different numbers of layers used as defense, while the inner loop loops through different values of the perturbation bound. After one run of the outer loop, resulting accuracies are printed. Inside the inner loop, the experiment function calls either the function train_test_red_classifier or the function train_test_pca_classifier.

The function train_test_red_classifier tests the robustness of autoencoder dimensionality reduction defense for a specified number of layers for a specified scenario and perturbation bound. First, an autoencoder is trained with the training data, if such a trained model is not already saved. The output of the hidden layer of this trained autoencoder is used to train the classifier, if such a trained classifier is not already saved. Then the test data is perturbed using the cleverhans library, in the perturb or perturb_arch_mismatch function. This perturbed data is passed through the autoencoder to obtain the compressed representation, which is then forwarded through the classifier to obtain the classification accuracy. The function train_test_pca_classifier follows a similar structure to obtain the accuracy using PCA defense, except that it does not have to train an autoencoder, but generates the feature subspace based on the training data, onto which it projects the test data.

The functions perturb and perturb_arch_mismatch generate perturbations for the semi-white box and black box scenarios respectively, given the order of the perturbation bound. For the semi-white box scenario, the same classifier that has already been trained by the non-adversary is used to generate perturbations. For the black box scenario, a different architecture: FC-784-200-200-100-10 is trained and used.

All saved models are stored in subdirectories of the saved_models directory. The directory autoencoders has the trained autoencoders, and the directory classifiers has the classifiers trained with autoencoder-reduced dimensions. The directory pca_classifiers has classifiers trained with PCA-reduced dimensions. The directory adv_models has models generated by the adversary for the black box case. The results can be found in text files in the four subdirectories of scenario: wb_fgs, wb_fg, bb_fgs, bb_fg.

This software implementation has been designed to be scalable and easily modifiable for further work. Only the data loading function get_data would have to change to use a different dataset. The gen_model function would have to be modified to use a different classifier. One can easily experiment with a larger number of epochs by changing the epochs parameter that is passed to any function that trains a neural network.