Fooling neural networks by adding noises
CS289 final project

Ruonan Hao
3032131935
ruonan_hao@berkeley.edu

Mingjia Chen
3032130297
mingjia_chen@berkeley.edu

Shuhui Huang
3032129712
shuhui@berkeley.edu

Abstract

This paper uses two methods to generate adversarial examples, which is Gaussian blur and fast gradient sign method. Both trained with and without adversarial training respectively,and compare their performances on the adversarial examples. The dataset we mainly train on is MINST, and we also compare the results of adding different noises as well as using different neural network.

1. Introduction

Adversarial examples are generated by applying small but intentionally worst-case perturbations to examples from the dataset, such that the perturbed input results in the model outputting an incorrect answer with high confidence. In other words, adversarial examples are inputs to machine learning models that an attacker has intentionally designed to cause the model to make a mistake; they're like optical illusions for machines. Neural networks are vulnerable to adversarial examples. That is, these machine learning models misclassify examples that are only slightly different from correctly classified examples drawn from the data distribution.

To generate adversarial examples, we use two methods to add noises to the original images. One is adding Gaussian noise, while the other is fast gradient sign method. Then we compare the accuracy before and after adding noises. In addition, we use Back-propagation Network and Convolutional Network to train the models, both trained with and without adversarial training respectively,and compare their performances on the adversarial examples.

Adversarial examples are a good aspect of security to work on because they represent a concrete safety problem that can be addressed in the short term, and because fixing them is difficult enough that it requires a serious research effort. Traditional techniques for making machine learning models more robust, such as weight decay and dropout, do not work well on adversarial examples. Therefore, how to generate adversarial examples, and how to provide a significant defense against them, is what we are going to deal with in this paper.

CS289 final project, Berkeley, CA, USA, 2017

2. Data

We will use the mnist data set. The set of images in the mnist database is a combination of two of NIST's databases: Special Database1 and Special Database3. Special Database1 and Special Database3 consist of digits written by high school students and employees of the United States Census Bureau, respectively (LeCun et al., 2013).

The mnist dataset is divided in 60,000 examples for the training set and 10,000 examples for testing. The training set of 60,000 is divided into an actual training set of 50,000 training examples and 10,000 validation examples randomly. All digit images have been size-normalized and centered in a fixed size image of 28 x 28 pixels. In the original data set each pixel of the image is represented by a value between 0 and 255, where 0 is black, 255 is white and anything in between is a different shade of grey. Here are some examples of mnist digits:

Figure 1. examples of mnist digits

For convenience, we pickled the dataset to make it easier to use in python he pickled file represents a tuple of 3 lists: the training set, the validation set and the testing set. Each of the three lists is a pair formed from a list of images and a list of class labels for each of the images. An image is

represented as numpy 1-dimensional array of 784 (28 x 28) float values between 0 and 1 (0 stands for black, 1 for white). The labels are numbers between 0 and 9 indicating which digit the image represents. The transformation can be shown in figure2.

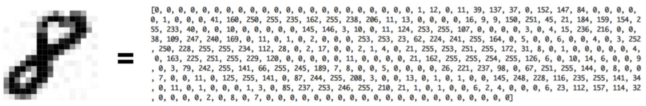

Figure 2. how do computers transform a digital image into vector

And we will make classification about labels according to the features.

3. Related work

Adversarial learning first gained popularity in the realm of image processing, where the heavy use of neural networks improved performance across different corpora. In image processing, where automatically generating variants of data points is quite common, Goodfellow made the distinction between adversarial learning and common data augmentation techniques such as transformations: adversarial examples are not naturally occurring examples that need to realistically happen in the world. Adversarial examples have been shown to exist for a variety of deep learning architectures. They are small perturbations of the original inputs, often barely visible to a human observer, but carefully crafted to misguide the network into producing incorrect outputs. Seminal work by (Christian Szegedy, 2013) and (Ian J Goodfellow & Szegedy, 2014), as well as much recent work, has shown that adversarial examples are abundant and finding them is easy.

Adversarial attacks can be untargeted, where the adversarys goal is to cause any misclassification, or the least likely misclassification (Alexey Kurakin & Bengio, 2016); or they can be targeted, where the attacker desires a specific misclassification. (Seyed-Mohsen Moosavi-Dezfooli & Frossard, 2016) gives a recent example of a strong targeted adversarial attack.

This time, we will focus on untargeted adversarial examples. Generating untargeted adversarial examples can be approximated with single-step gradient-based techniques like fast gradient sign (Ian J Goodfellow & Szegedy, 2014), fast gradient L2 (Ruitong Huang & Szepesvari, 2015), or fast least likely class (Alexey Kurakin & Bengio, 2016); or they can be approximated with iterative variants of those and other gradient-based techniques (Alexey Kurakin & Bengio, 2016); (Seyed-Mohsen Moosavi-Dezfooli & Frossard, 2016).

Adversarial examples add noise to the training set that is smaller than the precision of the initial training image. Such noise is undetectable by human eyes and also cannot be obtained by the image capturing equipment. Goodfellow argues that training with such perturbations will help neural networks truly capture concepts and gain accuracy beyond perceived data. (Jonghoon Jin & Culurciello, 2015) pointed out the adversarial examples generated in Goodfellows paper are sampled near a decision boundary, thus making network models particularly vulnerable. (Takeru Miyato & Ishii, 2015) further demonstrated the theoretical implication that adversarial training examples will help models to better find non-linear relationships in data.

As for models that can be robust to adversarial examples, (Gu & Rigazio, 2014) and (Chalupka & Eberhardt, December, 2014) have already begun the first steps toward designing models that resist adversarial perturbation, though no model has yet successfully done so while maintaining state of the art accuracy on clean inputs.

4. Methods & Models

4.1. Generating adversarial examples

4.1.1. Gaussian noise

The Gaussian noise is a type of image-blurring filter that uses a Gaussian function (which also expresses the normal distribution in statistics) for calculating the transformation to apply to each pixel in the image. In other words, the values that the noise can take on are Gaussian-distributed.The equation of a Gaussian function in one dimension is:

$$p_G(z) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(z-\mu)^2}{2\sigma^2}} \tag{1}$$

where z represents the grey level, μ the mean value and σ the standard deviation. Figure 3 shows the results of adding gaussian noise.

We use CNN and back-propagation neural network to train the model separately before and after adding Gaussian noises to mnist dataset.

4.1.2. Fast gradient sign method

As introduced by Goodfellow (Ian J Goodfellow & Szegedy, 2014), we trained with an adversarial objective function based on the fast gradient sign method was an effective regularizer:

$$\tilde{J}(\theta, x, y) = \alpha J(\theta, x, y) + (1-\alpha) J(\theta, x + \epsilon sign(\bigtriangledown_x J(\theta, x, y))) \tag{2}$$

with J the cost used to train the neural network, θ is the model parameters, and y is the label of x.

This approach means that we continually update our supply of adversarial examples, to make them resist the cur-

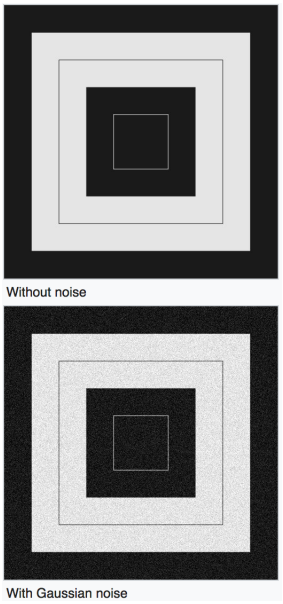

Figure 3. the influence of adding Gaussian noise to the image

rent version of the model. Because the derivative of the sign function is zero or undefined everywhere, gradient descent on the adversarial objective function based on the fast gradient sign method does not allow the model to anticipate how the adversary will react to changes in the parameters.

One of the example of adding noise using fast gradient sign method is shown in the figure4.

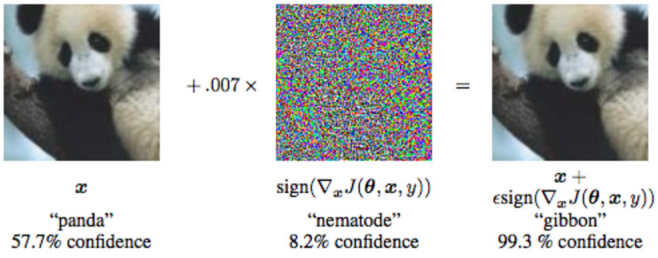

Figure 4. adding noise using fast Gradient Sign Method

In order to use adversarial examples to train a robust network, we train it both on natural images and constructed adversarial images.

4.2. BP and Conv Network

Why are neural networks easily fooled?

As (Nguyen A, 2015) demonstrated, that discriminative Deep neural network models are easily fooled in that they classify many unrecognizable images with near-certainty as members of a recognizable class. Two different ways of encoding evolutionary algorithms produce two qualitatively different types of unrecognizable fooling images, and gradient ascent produces a third.

To get a deeper understanding of neural network, we will use Back-propagation Network and Convolutional network to train mnist dataset, and compare their difference.

4.2.1. Back-propagation Network

Consider a network with a single real input x and network function F. The derivative $F'(x)$ is computed in two phases:

Feed-forward: the input x is fed into the network. The primitive functions at the nodes and their derivatives are evaluated at each node. The derivatives are stored.

Back-propagation: the constant 1 is fed into the output unit and the network is run backwards. Incoming information to a node is added and the result is multiplied by the value stored in the left part of the unit. The result is transmitted to the left of the unit. The result collected at the input unit is the derivative of the network function with respect to x.

Algorithm 1 Back-propagation network

Input: vectors X_i, size m
repeat
  Feed-forward computation
  Back propagation to the output layer
  Back propagation to the hidden layer
  Weight updates
until the value of the error function has become sufficiently small

After choosing the weights of the network randomly, the back propagation algorithm is used to compute the necessary corrections. The algorithm can be decomposed in the four steps as the algorithm1 shows, and it will be stopped when the value of the error function has become sufficiently small.

4.2.2. Convolutional Network

The other neural network we use is Convolutional Network.

Convolutional Neural Networks are very similar to ordinary Neural Networks : they are made up of neurons that have learnable weights and biases. The difference is ConvNet architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network.

A simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. We use three main types of layers to build ConvNet architectures: Convolutional Layer, Pooling Layer, and Fully-Connected Layer (exactly as seen in regular Neural Networks). We will stack

these layers to form a full ConvNet architecture.

The architecture of a convolutional network looks like this:

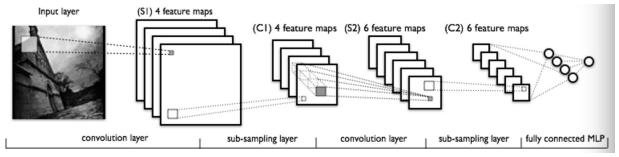

Figure 5. architecture of a simple convolutional network

- *INPUT layer:* will hold the raw pixel values of the image
- *CONV layer:* will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume.
- *RELU layer:* will apply an element-wise activation function, such as the max(0,x) thresholding at zero. This leaves the size of the volume unchanged.
- *POOL layer:* will perform a down-sampling operation along the spatial dimensions (width, height).
- *FC (i.e. fully-connected) layer:* will compute the class scores. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

4.2.3. ADVERSARIAL TRAINING

What is adversarial training?

Different from training the model with both the original data and adversarially modified data (with correct labels) as (Ian J Goodfellow & Szegedy, 2014) said. In our model, adversarial training isn't acutely trained on adversarial examples. It works by adding an adversarial loss to our original cross entropy loss and count two losses equally towards our final loss function. For each batch, compute the cross entropy loss as usual, use fast gradient sign method to generate adversarial examples, and compute the cross entropy loss of the adversarial examples. Take the mean of the two cross entropy losses as the final loss function we try to minimize.

5. Implement with MNIST dataset

We constructed two neural networks to train the adversarial example. One is convolutional neural nets and the other is ordinary back-propagation neutral nets.

For back-propagation neural nets, we used two hidden layers; we used RELU function as activation function for all layers. Each hidden layer contains 256 hidden units. The input layer contains 784 units whereas the output layer has 10 neuron units with softmax cross entropy function, which is minimized by Adam Optimizer.

For convolutional neural nets, we used two CONV layers, two RELU layers, two POOL layers, one fully-connected layer and one dense layer. The first CONV layer computes 32 feature maps with 5×5 filters, which is followed by a RELU layer. Then is the first POOL layer, who performs max pooling with a 2×2 filter and stride of 2. The next three layers are the same as before, except for the CONV layer has 64 feature maps this time. The fully-connected layer has 1024 neuron units with a RELU activation function. The dense layer performs a dropout regularization with rate 0.4. The output later (logits layer) has 10 neuron units with softmax function. Cross entropy loss is minimized by Adam Optimizer.

After first training, back-propagation neural nets reached an accuracy of 0.94 after 5 epochs on training dataset and 0.91 on validation dataset. CNN neural nets reached an accuracy of 0.986 after 5 epochs on training dataset and 0.981 on validation dataset. The results are shown below as in table 1:

Table 1. classifier accuracy for ordinary training

	BP	CNN
TRAINING	0.946	0.986
VALIDATION	0.918	0.981

Then we used fast gradient sign method to add noise on our training dataset and validation dataset. We also added random noise with the same mean and standard deviation as the noise we extracted from fast gradient sign method on our dataset and compared the performance between two classifiers.

By using 5000 training dataset and 5000 validation dataset to test the accuracy, we got the following results on back-propagation neural nets after adding noise, which is shown in table2:

Table 2. classifier accuracy after adding noise for BP neural nets.

	FGSM NOISE	GAUSSIAN NOISE
TRAINING	0.1868	0.563
VALIDATION	0.1728	0.565

From the table2, we can see that we heavily break down the neural networks if we use fast gradient sign method to add noise. The accuracy has also been reduced if adding

Gaussian noise with the same mean and standard deviation, but not as much as fast gradient sign method.

The results in table 3 shows accuracy for CNN neural nets after adding noise, by using 5000 training dataset and 5000 validation dataset to test the accuracy.

Table 3. classifier accuracy after adding noise for CNN neural nets.

	FGSM NOISE	GAUSSIAN NOISE
TRAINING	0.295	0.365
VALIDATION	0.299	0.370

The result for CNN is a little bit surprise to us, since it does not have a significant difference between fast gradient sign method and Gaussian blur. Fast gradient sign method still works better (in terms of breaking down the classifier) than ordinary Gaussian noise, but the Gaussian random noise is good enough to break down the classifier if taking the difficulty to extract fast gradient noise into consideration. The accuracy for CNN on Gaussian noise is lower than back-propagation. However, these two accuracies are not comparable, because the standard deviation of noise added to CNN is much higher than the standard deviation of noise added to back-propagation neural nets. Generally speaking, validation accuracy of CNN on Gaussian noise is much higher than back-propagation neural nets. The comparison can be seen in Figure 6.

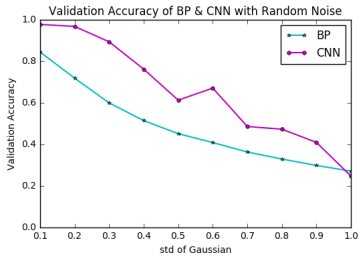

Figure 6. Random noise training for CNN and BP

One of the amazing properties of CNN lies in that we can construct a much more complicated neural nets with less parameters compared to ordinary back-propagation neural nets. Thus, we expected that this much more complicated neural nets can perform better in adversarial training. However, one of the amazing properties of CNN lies in that we can construct a much more complicated neural nets with less parameters compared to ordinary back-propagation neural nets. We expected that this much more complicated neural nets can perform better in adversarial training.

We used fast gradient sign noise to conduct adversarial training, which means adding an adversarial loss to our original cross entropy loss and count two losses equally towards our final loss function. After 5 epochs adversarial training, we evaluate the performance of our classifier by training dataset with adversarial examples and examples with random noise. Table 4 shows results for BP and CNN on adversarial examples.

Table 4. Adversarial Training Results for BP and CNN on Adversarial Examples

	ORD_{bp}	ADV_{bp}	ORD_{CNN}	ADV_{CNN}
TRAINING	0.1868	0.603	0.295	0.339
VALIDATION	0.1728	0.606	0.299	0.337

For back-propagation neural nets, the training and validation accuracy improve a lot: from around 0.18 to 0.6. For CNN, the accuracy only improves 0.03. It seems like CNN did not perform well on adversarial training. However, it has a higher accuracy with random noise after adversarial training.

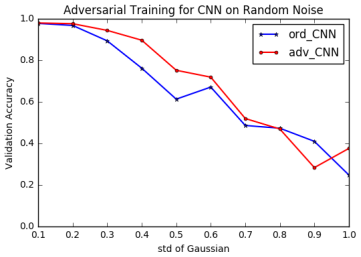

Figure 7. Adversarial Training Results for CNN on Random Noise

Figure 7 and figure 8 compare the CNN and BPs resistance against Gaussian noise before and after adversarial training.

The blue line in figure 7 stands for the accuracy for the original CNN classifier when the standard deviation of Gaussian distribution increases whereas the red line stands for the accuracy for CNN classifier after adversarial training. Generally speaking, CNN classifier after adversarial training has a higher accuracy than the original one, which means CNN has become more robust on random noise after adversarial training. However, on the other hand, the back-propagation neural nets did not become more robust on random noise or even perform worse than the original back-propagation neural nets.

6. Conclusions and Future Work

6.1. Conclusions

There is an interesting cross connection between two classifiers and two noises. Initially, we expect CNN would per-

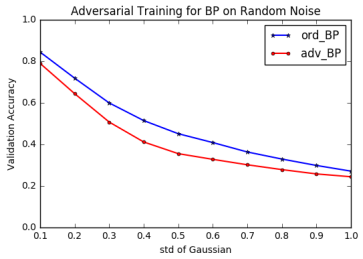

Figure 8. Adversarial Training Results for BP on Random Noise

form better in both situations meaning it will improve its accuracy a lot after adversarial training on adversarial examples and on examples with Gaussian blur. However, the result only satisfy the later. CNN still has a higher accuracy on adversarial examples but not improve that much. To sum up the result, consider the table5:

Table 5. Adversarial Training Performance for BP and CNN

	FGSM	GAUSSIAN BLUR
BP	0.1728 →0.606 IMPROVE A LOT	0.565→ 0.507(STD = 0.3) WORSE THAN BEFORE
CNN	0.299→0.337 IMPROVE A LITTLE	0.365→0.469(STD = 0.8) BETTER THAN BEFORE

Even though the results have a little bit divergence with our initial construction, it makes sense to some extent. Adversarial training is a brand new topic in deep learning, it is called adversarial examples means it is not as easy to train as other datasets. CNN is only the simple neural networks architectures in deep learning algorithms. There are tons of complicated neural networks structures that may be more robust than CNN and could perform even better in this situation. For example, as Ian J. Goodfellow mentioned in his paper, RBF neural nets are resistant to adversarial example. Neither CNN nor BP might not resistant enough to adversarial example.

Nonetheless, CNN is still worthwhile for adversarial training, since it indeed has a better performance on random noise. What we usually will encountered in daily life is more likely the random noise rather than the adversarial examples created intentionally by fast gradient sign method. If we can make CNN more robust on random noise by conducting adversarial training, it is still worth to make effort on it. We only try one architecture of CNN, there might be other architecture of CNN works, but we do not have time to figure it out.

6.2. Future work

There are still a lot of things to do to deal with adversarial examples.

Our experiment showed that neural networks suffer from vulnerability to adversarial examples. As Goodfellow (Ian J Goodfellow & Szegedy, 2014) have already proved, models with local units (RBF networks and SVMs with RBF kernels) are quite resistant to such behaviour. Several hypothesises and tests have been done, while none of them is satisfying. So one direction is to find the reason of robustness towards adversarial examples.

Another direction is to find an effective model which is able to defend against adversarial examples. In this paper, we use adversarial training that simply generate a lot of adversarial examples and explicitly train the model not to be fooled by each of them. Although we do reduce error rate in this way, our machine learning models still work on a very small amount of all the many possible inputs they might encounter. Another way called "Defensive distillation", which was originally introduced in (Geoffrey Hinton, 2015), is useful to some extent. However, it can easily be broken by giving more computational firepower to the attacker. Designing a defense that can protect against a powerful, adaptive attacker is an important research area.

References

Alexey Kurakin, Ian J. Goodfellow and Bengio, Samy. Adversarial examples in the physical world. *CoRR, abs/1607.02533*, 2016.

Chalupka, K., Perona P. and Eberhardt, F. . deep neural networks are easily fooled: High confidence predictions for unrecognizable images. *ArXiv e-prints*, December, 2014.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever Joan Bruna Dumitru Erhan Ian Goodfellow Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

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

Gu, Shixiang and Rigazio, Luca. Towards deep neural network architectures robust to adversarial examples. In NIPS Workshop on Deep Learning and Representation Learning, 2014.

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

Jonghoon Jin, Aysegul Dundar and Culurciello, Eugenio. Robust convolutional neural networks under adversarial noise. *arXiv preprint arXiv:1511.06306*, 2015.

LeCun, Yann, Cortes, Corinna, and Burges, Christopher J.C. Mnist handwritten digit database, yann lecun, corinna cortes and chris burges, 2013.

Nguyen A, Yosinski J, Clune J. Visual causal feature learning. *In Computer Vision and Pattern Recognition (CVPR 15), IEEE*, 2015.

Ruitong Huang, Bing Xu, Dale Schuurmans and Szepesvari, Csaba. Learning with a strong adversary. *CoRR, abs/1511.03034*, 2015.

Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi and Frossard, Pascal. Deepfool: a simple and accurate method to fool deep neural networks. 2016.

Takeru Miyato, Shin-ichi Maeda, Masanori Koyama Ken Nakae and Ishii, Shin. Distributional smoothing with virtual adversarial training. *stat, 1050:13*, 2015.