Midterm Project

MATH 540 Statistical Learning
Adversarial Attacks

Artur Pak

Fall 2023

Project overview

Objectives:

1. Learn about theory and application of adversarial attacks in machine learning.
2. Gain understanding of some practical tools in machine learning e.g. `pytorch`, `tensorflow`, etc.

Tasks:

1. Read literature, papers, watch videos or use other means to gain knowledge relevant to the topic.
2. Write a program which can create adversarial examples.

Outcomes:

1. A comprehensive overview of the topic with theory and discusssion.
2. Pet project: Generation of adversarial examples via FGSM.

1 Introduction

Adversarial machine learning is a branch of ML that is concerned with various methods of messing with inputs to make ML algorithms to make wrong predictions - such methods are called adversarial attacks. In divide attacks into 3 categories:

- Poisoning Attacks - attackers introduce artificially modified data into training set for the model, aka "poison" the set, which in turn disrupts predicting ability of the algorithm.
- Evasion Attacks - are attempts to bypass or fool filtering algorithms, like spam detection or barometric authentication.

- Model Extraction - as the name suggests, the goal is to either reverse-engineer the model under the attack, or to extract the data it was trained on. There is a lot of concern around this issue, as it is connected to privacy and safety.

Adversarial examples are inputs to the algorithm, which are modified in a way that is not detected (or barely so) by a human, but make the algorithm to make a wrong prediction. There are two types of such techniques: while-box - where the attackers have an access to the architecture of targeted model, and black-box - where they don't.

2 FGSM attack

Among numerous methods to create adversarial examples, **Fast Gradient Sign method** (FGSM in short) is one of the most intuitive ones. It is a white-box method, which can be used to attack gradient-based algorithms. Though access to the model is a hard condition to satisfy, once it is granted, FGSM attack is considered to be relatively fast and efficient.

2.1 Theory

The basic idea behind creating adversarial examples via FGSM is described in by Goodfellow et al.[1]. In hindsight it seems surprisingly simple and logical. For simplification, consider a linear model which maps x to $w^T x$. Now, we want to tweak the some input x_0 by adding a perturbation η, but we want η to be "barely noticeable", so lets say that $||\eta||_\infty < \epsilon$. This is equivalent to changing each pixel of the image by at-most some value, so that the result is not visible. Then, the prediction on the perturbed image $\hat{x_0}$ will be:

$$w^T \hat{x_0} = w^T x_0 + w^T \eta \tag{1}$$

Now, if the average of elements of the weight vector is w, then the activation will grow roughly by ϵmn, where n is the dimension of x. So, if the input has high dimensionality, as in case with pictures, then even perturbations with small ϵ may lead do drastic changes in activation. Moreover, our goal is to make the algorithm to make a wrong prediction, i.e. have a high loss. So, it seems logical to take η in the direction of the gradient of the loss with *respect to the input.* This is why we take $\eta = sign(w)$ in this example.

Now the question is the following: neural networks are specifically designed to fit complex non-linear functions, so why does it still work? The answer lies in our choice of *activation functions*: we choose functions with simple-to calculate gradient with *almost linear behaviour.* Consider ReLU networks. We rely on the fact that most of the time the activation behaves linearly, otherwise the gradient will be mostly 0 and will give no information. Other activation functions have regions were they display almost linear behaviour, so the mechanism stays the same.

2.2 Practical example

I decided to implement FGSM attack by following a tutorial from tensorflow: Adversarial example using FGSM. However, to preserve and intensify the learning experience, I changed the library to pyTorch and used MNIST dataset for better visualization; also, I created and trained a simple fully connected neural network with ReLU activation functions, instead of using a pre-trained model. See code in attachments 4.3.

For the demonstration the network has 2 hidden layers with 64 nodes each; it was trained with Adam optimizer and Cross Entropy loss as a loss function. Over 3 epochs it achieved accuracy 0.9714 on the test set.

The FGSM attack takes the model, image-label pair and ϵ as input. It calculates signs of the gradient of the loss w.r.t. the image, scales by $\epsilon = 0.1$ and adds to the image. The results are shown on Figure 1.

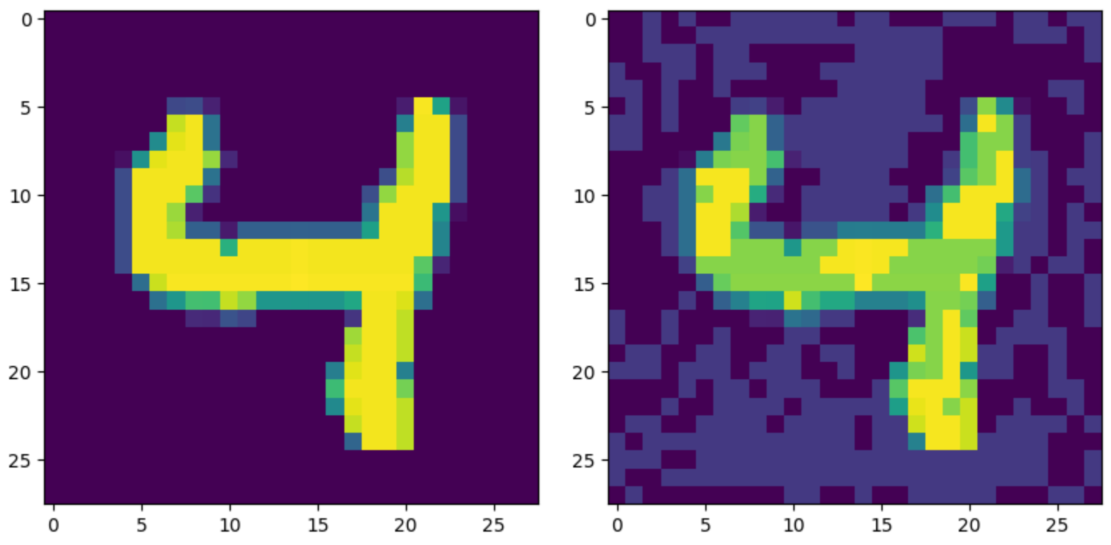

Figure 1: Left: Adversarial example on a MINST dataset. The original image is on the left. The network predicts 4 with 0.999 confidence. On the left if the perturbed image after FGSM attack. The network predicts 9 with 0.689 confidence

3 My hypotheses and experiments

3.1 Naive thoughts

Explanation of the principle behind FGSM attack led me to a couple of thoughts:

1. Since most of the connections are linear, then outputs up to the last one (argmax) should also depend roughly linearly on the input. This may imply that the average **loss should grow linearly with** ϵ, at least for shallow neural networks.

2. Deeper neural networks supposedly create more complicated, i.e. less linear patterns. Hence, I suppose, **the deeper the network, the higher ϵ is required to achieve the same loss**.

I am pretty sure these propositions were addressed in some form in serious papers, but 1) I am too lazy to google it right now; 2) need some practice with tensors anyway.

3.2 Naive tests

In order to test my hypotheses, I created 3 architectures: a shallow net described it section 2.2; a deep net with 3 hidden layers 128 nodes each; a very deep net with 5 hidden layers 256 nodes each.

I created 3 instances of each network type, and trained on a test set, having a total of 9 models. Then I perturbed the test set with various values of ϵ for each of the model and calculated average loss and accuracy. See the code in the attachments 4.3.

3.3 Kinda results

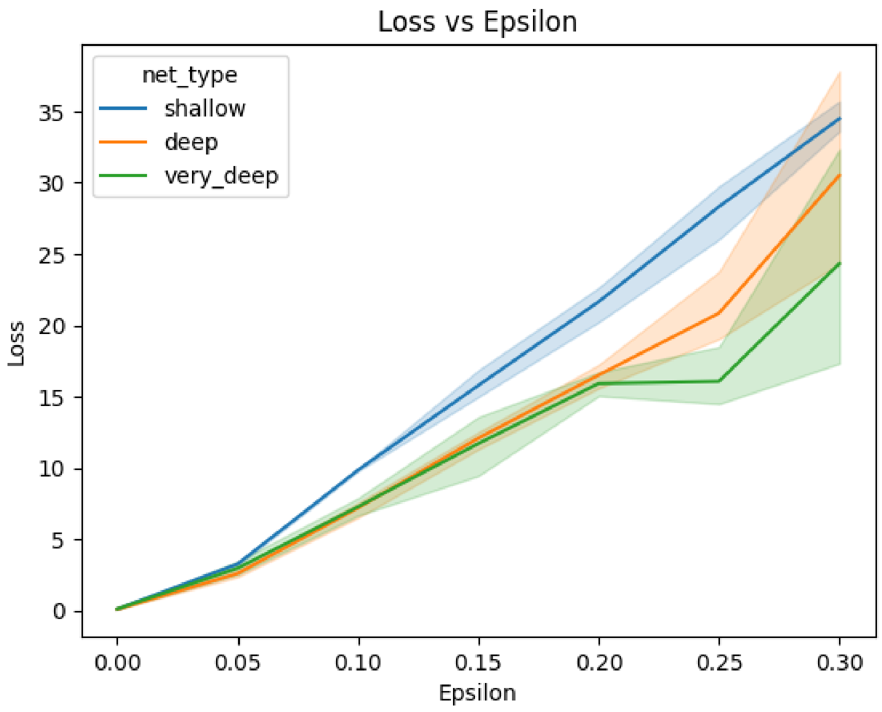

Figure 2: Shaded regions indicate the 90% confidence interval

The results shown on Figure 2 suggest that both hypotheses are true.

3.4 Future work

First and foremost, the test in section 3 should be improved: more model types and a higher number of models per type is needed to get a more reliable conclusion.

Second, it would be interesting to explore, how the choice of the activation function will affect the behaviour of the loss with respect to ϵ. For example, if we used *tanh* instead of ReLU, how would the graph Loss vs. Epsilon look like?

Finally, similar question as in 2, but for other types of architectures, such as CNN.

Actually I have lots more questions regarding the adversarial attacks, but for now three points above are the closest to the current point of progress to be focused on.

4 Miscellaneous

4.1 Things I learned

Apart from the things I read and reported, I was able to get some practical experience. In particular, how to handle tensors in some cases. For example, tensors usually track the operations which were done to them, so that in the future we can use back-propagation and find the gradient with respect to the leaf nodes. However, it takes up time and resources to memorize them, so during predictions we disable tracking to speed up the process. Moreover, with the tracking enabled some other functions, such as visualizing the tensor, can not be performed.

Actually, I encountered more problems and learned from them during coding, but I was too busy troubleshooting them and did not remember any but the one I mentioned above.

4.2 Interesting thoughts

Once you understand the principles, *magic* tricks become *just* tricks, so is with the Machine Learning.

Goodfellow noted, that an existence of such methods to fool the algorithm suggest that these models, despite their fantastic performance, lack the understanding of concepts we possess. So, all there is, is just a gradient and optimization over it.

Since artificial neural networks are attempts to simulate the functioning of our brains, I wonder if there is a method to fool our minds in a similar fashion.

4.3 Attachments

The git link: https://github.com/usaginoki/pytorch_mnist_fgsm_pet_project

References

[1] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and Harnessing Adversarial Examples.* 2015. arXiv: `1412.6572 [stat.ML]`.