INF573 Final Project Report

Daniil Smoliakov, Markus Chardonnet

December 2020

Introduction

Increase of computation power and widespread use of GPU allowed to achieve new results in deep learning. Development of powerful convolutional neural networks made it possible to solve the problem of image classification with high accuracy. With an efficiency greater than that of a human. And now, these networks are being integrating as solution into existing tasks. However security in machine learning has received far less attention. And in this report we are describing concept of adversarial examples.

Szegedy el al. (2014) described an idea that ML models can miss-classify data which is only slightly different from the data that model has seen. We will call this data adversarial example. And this data(adversarial example) is not specific to some architecture. Moreover such examples can be wrongly classified by different algorithms trained with different data sets. And this examples can be used to disrupt work of neural network.

Robustness, to accident and to malevolent attacks is one of the crucial determinant of the success of machine learning system, that applied in the real world. There are variety of possible negative impacts from poorly secured systems in sensitive regions like medical and transportation systems. Learning how to attack and defend such neural networks improves the quality, security and efficiency of models.

In this report we focus on implementation of different algorithms for creation of adversarial examples and further comparison of results given by those algorithms on our own created data set of images, showing how neural network vulnerable to such examples.

1 Implemented algorithms

For each algorithm we used Python programming language and open source machine learning framework Torch. These tools were chosen as the most common, versatile and convenient for prototyping and fast development.

All attacks can be split in two huge groups: white box and black box attacks.

In the first case we have direct and full access to our neural network and can get all information from it like gradient and weights. This allows us to directly find changes leading to the desired result. Implementation of this attacks are Fast gradient sign method and deep fool

In the second case we have only ability to retrieve probability of classes. It's more complex and challenging scenario, since we can't finding direction to adversarial changes guided by gradient information. At the same time it's more common and close to the real-world settings where CNN can be given as API with ability to do requests to it.

All attacks implemented during the course working in not targeted way and can not perturb image in a way to get desired target class.

1.1 Fast Gradient Sign Method (FSGM)

Described by Goodfellow et. al (2014) fast gradient sign method is a way to quickly generate a perturbation direction for input image such that loss function of the model will increase, reducing confidence of prediction and increasing probability of wrong classification. It is one of the first algorithm for generation of adversarial examples and there is no guarantee that increasing loss by some amount will result in miss-classification, however it is a sensible direction to take.

At the first step we calculate the loss function according to our index or label. After that using we are finding how much each value of input image changes loss value. In the case of convolution neural networks this can be calculated through the back-propagation algorithm. After that we just add this value to the image multiplied by the learning rate, in order to reduce the loss function. Each iteration can be described as follows:

$$x' = x + \alpha \cdot sign(\nabla_x J(\theta, x, y))$$

where x - base image, J - loss function, y - correct index or label, α - small multiplier to control speed of changes, θ - model parameters.

This process is repeated until an adversarial example is found.

This can be updated to basic iterative method. It is one of the many extensions for fast gradient sign method. The principle is just to fix an ϵ and clip x' into a ball of radius ϵ around x. Of course, we have to be sure that the resulting x' stays in the RGB values boundaries (when converted to integers, it has to fit the [0,255] bounds) but this problem rarely occurs in practice. [3]

1.2 Deep Fool

Deepfool is a elaborated and very successful white box attack, one of the state-of-the-art solutions. The principle is to approximate the distance from and input to the closest decision boundary of a classifier. Deepfool linearizes the the model's decision boundaries around the current batch and moves this batch to the closest boundary. As in FGSM, This process it repeated until the batch is miss-classified. This algorithm can be adapted in order to get the best results for a specific L_p distance metric.

The algorithm takes as an input image x, its truth label y and desired L_p norm. The distance (which depends on p) to the decision boundary of every class is computed according to linearized model. Then the image is perturbed along the according direction so that it would became miss-classified in a linear model. This process while the classifier f keeps predicting y. It outputs the perturbation (eg. the difference between the batch found and the original batch). [2]

The distance in L_p norm from an input batch x to a targeted class c with respect to the the linearized model is :

$$\frac{|f_c|}{\|\omega_c\|_q} \tag{1}$$

where q = $\frac{p}{p-1}$ and :

$$f_c = f(x)_{(c)} - f(x)_{(y)} \tag{2}$$

$$\omega_c = \nabla_x f(x)_{(c)} - \nabla_x f(x)_{(y)} \tag{3}$$

1.3 Simple Black-box Adversarial Attack (SIMBA)

Described by Guo et. al (2019) is simple and straightforward black-box attacks, that proposes to use output of probabilities as a strong proxy guide for creation of adversarial images by changing as minimal elements as possible. With simple intuition we are changing our image x in any direction q with some step size ε and one of values: $x + \varepsilon q$ or $x - \varepsilon q$ will likely make our probability of correct label smaller.

Algorithm receive base image, its class or label, step-size and some orthonormal vectors Q. At first step we are creating permutation of this vectors. By this we prevent secondary usage of the same vector to guarantee maximum query efficiency, so that no two directions cancel each other out and diminish progress. After that, we iterate through this permutation applying each vector perturbation to the image in purposed way:

$$x' = x \pm \varepsilon q$$

where x - base image, ε - step size, q - chosen vector. As result one of the changed images should cause changes in probability of base label.

Choice of this orthonormal vectors is a debatable topic, but for this implementation we are taking most natural choice. Our image is our basis, so we

perform algorithm directly in the pixel space. In each iteration we are increasing or decreasing one color of the a single chosen pixel. But the algorithm should work with any general orthonormal basis.

By design, this algorithm is very easy to upgrade in order to be able to handle targeted attack, just by controlling that every change is not only minimizing probability of base class, but also maximizing probability of target class. [5]

1.4 Simultaneous Perturbation Stochastic Approximation (SPSA)

This algorithm one of the huge group with similar idea. The principle of this technique is to approximate the gradient of the classifier f with respect to an input x and then apply gradient descend to modify the input. While the Finite Difference method approximates the derivative of the classifier for each input coordinates, SPSA does a more global approximation (and thus less accurate). The reason for this is to lower the number of requests to the classifier and thus reduce the computation time.

The idea is to take a vector v which has the same shape as x and which components are elements randomly chosen in $\{1, -1\}$. Then, approximate the gradient using only to queries :

$$\frac{\partial f}{\partial x} \approx g(v) = \frac{f(x + \delta v) - f(x - \delta v)}{2\delta} \tag{4}$$

Where δ is a small constant. In practice, we compute several such vectors v : $\{v_1, .., v_n\}$ and then state the following :

$$\frac{\partial f}{\partial x} \approx \frac{1}{n} \sum_{i=1}^{n} g(v_i) \tag{5}$$

We update x as in the gradient descend algorithm : $x' = x - \frac{\alpha}{n} \sum_{i=1}^{n} g(v_i)$ where α is the step size.

This process is repeated until we find and adversarial example x'. One can also add a a bound error value ϵ and then clip x' into a ball of radius ϵ around x. This can enforce the solution to be close to the initial input. [4]

2 Results and Experiments

In this section we evaluate our implemented algorithms on a set of images in multiple directions: How much our image changed, how many requests were done to the network and how strong is the miss-classification.

Globally we want to show and achieve expected result, where black-box attacks much more complicated from a computational point of view, but able to give acceptable adversarial examples and white-box methods are capable of producing results that are maximally indistinguishable from the original image in shorter time comparing to black-box methods.

2.1 Setup

All our algorithms were tested on pre-trained convolution network resnet50 provided by PyTorch. It is one of the most famous and popular CNN, that network used as a backbone for many computer vision tasks. That network required image pre-processing pipeline with cropping to size $224x224x3$ (width, height, RGB channel) and normalizing that values. As result this network provide one of the 1000 predefined classes. Notice that these algorithms should be able to work on any network, but they could lead to different results depending on the strength of the network.

We evaluate our methods with 10 different pre-processed images of different classes after having normalized them. The new images given by the algorithms are then scaled back according to the original image in order to visualize the results.

2.2 Numerical results

In this section, we show numerical results to compare the different methods we used. However it is important to remark, that our control data set contains only 10 images and for more reliable and accurate evaluation it is important to run more tests to collect more information

Difference norm : A first step is to measure how much our images have changed. Obviously, the goal would be to have the minimal difference between the outputted image and the original image. The perception of this difference can be easy to see in some cases, usually when the new image is very freezed. In other cases the images look very similar. In order to quantify this, we calculated the euclidean norm of the difference.

Computation constraints : Computation effectiveness is important to evaluate the strength of a method and its applicability in the real world and existing tasks and processes. Here we test the previous explained algorithms on two linked aspects : the computation time and the number of requests to the model (or the number of backwards on the model for white box attacks). As one can expect, black box methods tend to run longer and to make more

requests to to the classifier as white box methods, since they don't have access to internal information of the model. Under access we assume prediction and back-propagation to retrieve information for image correction. All calculation were made on Intel Core i7 CPU, under Linux and Python 3.9.

FGSM: According to table 4 Fast Gradient Sign Method significantly surpasses all other presented algorithms in terms of execution speed and the number of requests to the neural network. However despite the fact of fast and successful miss-classification this methods produce very noisy and low quality results as can be seen on every image in section 6. Even difference in norm, that however do not represent human perception shows huge distinction between base and resulted image.

The results shown bellow are obtained using a step size of 0.3. One of the possible ways to reduce noise is to iterate on step size until finding the most suitable one. However, reducing this value does not give significantly better results in our experience. Overall it is great, simple and cheap baseline algorithm that work fast and doing it job almost always within one iteration, that can be used to speed up adversarial training or just to analyse of trained networks.

FGSM	**Time (seconds)**	**Queries number (or backwards)**	**Difference norm**
bullet train	1	1	119
car wheel	2	3	156
electric locomotive	1	1	82
guacamole	1	1	111
guacamole	1	1	72
power drill	1	1	83
red wine	1	1	93
samoyed	1	1	75
space shuttle	3	3	116
wombat	1	1	81

Table 1: FGSM result on images

Deep Fool: According to table 4 DeepFool is much slower than FGSM but achieves far better results as white-box attack as shown as table and images comparison tables. There is a longer computation time and significantly more requests to the neural network, because we propagate each loss value in labels however it produce clear image that difficult to distinguish from base image and achieved successful miss-classification. As a result, DeepFool can be used as a valuable tool to accurately assess the robustness of classifiers.

The results shown bellow are obtained using a step size of 0.3. Overall algorithm provides an efficient and accurate way to compute minimal adversarial perturbation direction. And by initial paper this size of the resulting perturbation can be interpreted as a measure of the model's robustness to adversarial attacks and further adversarial training to improve quality of neural network.

DeepFool	Time (seconds)	Queries number (or backwards)	Difference norm
bullet train	240	1001	0.26
car wheel	654	2002	1.03
electric locomotive	643	2002	1.83
guacamole	243	1001	3.91
guacamole	233	1001	0.83
power drill	238	1001	0.14
red wine	226	1001	2.62
samoyed	215	1001	1.24
space shuttle	210	1001	1.42
wombat	234	1001	1.36

Table 2: DeepFool result on images

Simba: All previous attacks was based on the fact that we have direct access to the network internal data. SIMBA overcome this to be more close to the real world problem, when we usually have only access to the output value. This should leads to raise in computational time and amount of requests to the network. This corresponds to the results of the experiment on the dataset and shown in Table 4.

Simba achieves very satisfactory results bot stay far from the performance of DeepFool. Some Simba images are sharp and have bright pixels, but still better at human perception if comparing with FSGM.

Visible difference leads us to possible improvements in comparison. To make it more similar to human perception we can switch from RGB to HSV as it is closer to natural color system. Other point to mention: computation time differs significantly from image to image from couple of minutes to half of the hour. The same applies to the number of requests to the network.

The results shown bellow are obtained using a step size of 0.3. Reducing this value lead to more soft and indistinguishable result, but significantly increasing amount of computations. But on other hand SIMBA can be easily parallelized by thanks to the design of the algorithm as far as our vectors are orthonormal. Other way to improve computation speed is to use discrete cosine basis, because recent work has discovered that and random noise in low frequency space is more likely to be adversarial.

Simba	**Time (seconds)**	**Queries number**	**Difference norm**
bullet train	1353	247	84.8
car wheel	11344	2203	104.1
electric locomotive	14466	3084	62.4
guacamole	8009	1811	94.5
guacamole	1355	352	69.4
power drill	355	90	72.8
red wine	5709	1583	82.3
samoyed	4271	1157	71.8
space shuttle	2161	400	90.9
wombat	4419	833	47.0

Table 3: Simba result on images

SPSA: For Simultaneous Perturbation Stochastic Approximation method, we were unable to achieve any satisfactory results. In the current state, the code calculates the approximation of the gradient, but this does not lead to the expected change in the class of the original image. We believe that our algorithm could work but with an undetermined amount of time.

Conclusion

We implemented several algorithms that make it possible to create adversarial examples in difference settings. These examples clearly show that existing machine learning solutions are vulnerable to creating examples that lead to missclassification. In the results area, the implemented algorithms demonstrated the expected results.

White-box algorithms work more efficiently in terms of speed and number of requests to the network. However, the fast gradient sign method creates images of very low quality that look questionable even to the human eye. However, deep fool, being a state-of-the-art solution for an acceptable number of requests, provides a result that is indistinguishable from the original image. To compare such images, we used numerical methods to compare.

The black-box method we implemented showed that for a large number of requests and without access to the internal content of the network, you can get a decent result, but not as qualitative as DeepFool and not as fast as gradient sign method. These methods can be directly used to exploit network vulnerabilities.

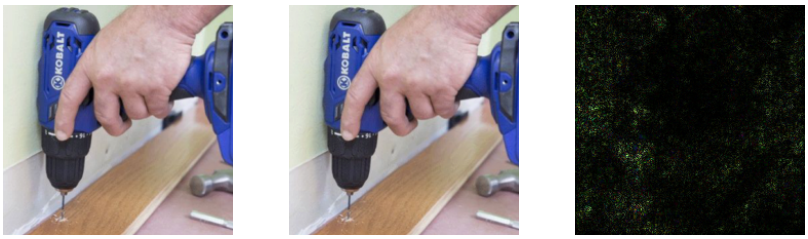
Each of the algorithms implemented by us can be improved in one way or another, be it an engineering or mathematical approach. But within the time of this project, we do not have the opportunity to deal with such modifications, we leave them for further work.

However, the results obtained make it possible to understand that the current neural networks are quite vulnerable for the described adversarial examples and it is necessary to conduct adversarial training and engage in attempts to interpret the internal content of the networks.

Attack	Average Queries	Average execution time	Average Image Difference
FGSM	1.4	1.3	98.8
DeepFool	1201.2	313.6	1.32
Simba	5342.2	1176.0	78.0

Table 4: Comparison of different methods

Images



Base/Power Drill | DeepFool/Plane | Scaled Difference

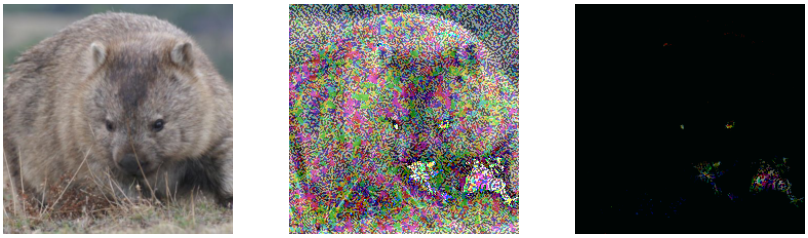
Base/Car Wheel | DeepFool/Disk Brake | Scaled Difference

Base/Wombat | DeepFool/Beaver | Scaled Difference

Figure 1: Some of DeepFool result

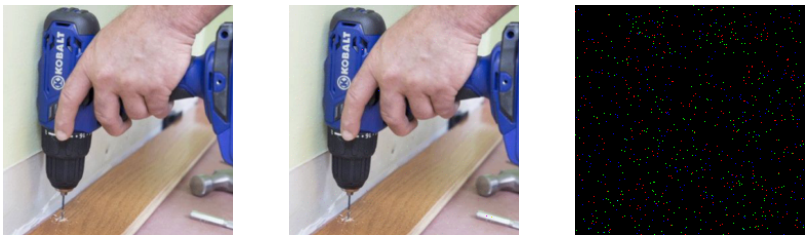
Base/Power Drill FGSM/Bow Tie Scaled Difference

Base/Car Wheel FGSM/Strainer Scaled Difference



Base/Wombat FGSM/bubble Scaled Difference

Figure 2: Some of FGSM result



Base/Power Drill SIMBA/Plane Scaled Difference

Base/Car Wheel SIMBA/Puck Scaled Difference

Base/Wombat SIMBA/Beaver Scaled Difference

Figure 3: Some of DeepFool result

References

[1] R. R. Wiyatno, A. Xu, O. Dia and A. de Berker, *Adversarial Examples in Modern Machine Learning: A Review*, Available at : link1

[2] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, *Deepfool: a simple and accurate method to fool deep neural networks*, Available at : link2

[3] I. Goodfellow, J. Shlens, and C. Szegedy, *Explaining and harnessing adversarial examples*, in International Conference on Learning Representations, Available at : link3

[4] J. C. Spall, *Multivariate stochastic approximation using a simultaneous perturbation gradient approximation*, IEEE Transactions on Automatic Control

[5] Chuan Guo, Jacob R. Gardner, Yurong You, Andrew Gordon Wilson, Kilian Q. Weinberger, *Simple Black-box Adversarial Attacks*, link5