Adversarial examples

Pablo Álvarez Cabrera
Universitat de Barcelona
Facultat de Matemàtiques i Informàtica
palvarca22@alumnes.ub.edu

Àlex Padrós Zamora
Universitat de Barcelona
Facultat de Matemàtiques i Informàtica
apadroza8@alumnes.ub.edu

Abstract

This paper is focused on studying a recently discovered issue affecting deep learning models: *adversarial examples*. Concretely, it has been found that by applying small but intentionally crafted perturbations to the inputs it is possible to drastically change the outputs. This is a serious problem in which a huge effort has been done in the last years in order to find a solution, but stills unsolved. We will focus on image classifiers based on deep neural networks, describing and analyzing two techniques for generating adversarial examples. We will perform some experiments with these methods, considering different situations.

1 Introduction

Recently, deep learning has become one of the most powerful tools in machine learning. Deep neural networks allow us to achieve great results in many kind of problems, such as image classification, speech recognition or natural language processing. This high performance is achieved because of the computation of several nonlinear steps. Even though, these computations are difficult to interpret, and the current algorithms for training these models have some issues. One of those issues, which affects the stability of neural networks, is the one that we will analyze in this paper: *adversarial examples*.

1.1 Problem description

Deep neural networks have demonstrated to achieve a high performance. This fact leads us to expect them to be robust to small perturbations in the input. That is, we expect the model to predict the same result for an slightly modified input and for the original one. However, it has been revealed [1] (Àlex) that by adding a small but intentionally crafted perturbation to an input, it is possible to change the network's output.

If we take a look at this from the real world perspective, it can be a major threat, since it reveals that many models are vulnerable to adversarial attacks. For example, think about a deep neural network that is used for detecting street signs, which makes a self-driving car stop when it detects the stop sign. If the input is perturbed, the car can ignore the stop and cause an accident (see Fig.1).

In this work we are focusing only on image classifiers based on deep neural networks. Thus, the adversarial examples that we are going to consider are images (*adversarial images*) with intentionally perturbed pixels, whose aim is to deceive the classifier to assign them incorrect labels.

2 Related work

To develop this paper, we have reviewed a collection of sources which introduce adversarial examples and explain some techniques to generate them. The papers that we mention below are the ones that have been essential.

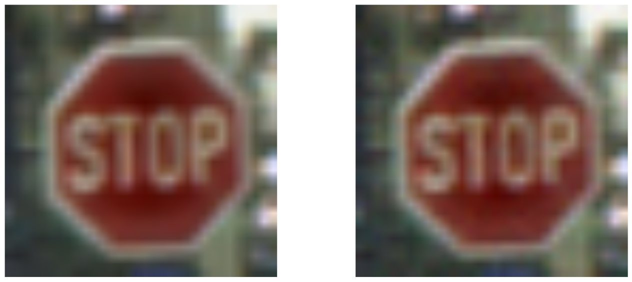

Figure 1: The image on the left is an ordinary image of a stop sign, while the image on the right is designed to force a particular deep neural network to classify it as a yield sign [2] (Pablo). To humans, both images appear to be the same.

- *Intriguing properties of neural networks* [1] (Àlex), by C. Szegedy et al., states the problem and provides a first approach to generate adversarial examples, based on the L-BFGS optimization algorithm.
- *Explaining and Harnessing Adversarial Examples* [3] (Pablo), by Ian J. Goodfellow et al., provides a widely used technique to obtain adversarial examples, known as the Fast Gradient Sign method, which we are going to discuss in the following section.
- *DeepFool: a simple and accurate method to fool deep neural networks* [4] (Pablo), by S.M. Moosavi-Dezfooli et al., details a very effective method for this type of attacks, which is based on geometric properties of high-dimensional spaces. We are also considering this approach to generate adversarial examples in the following sections.

The articles below contain information that, although it has not been crucial for this paper, we believe it is interesting to read them in order to go a little deeper.

- *Adversarial examples in the physical world* [5] (Àlex) by Ian Goodfellow et al., gives an introduction to adversarial examples, some real world examples of that phenomenon and it explains some existing methods for generating adversarial images, giving an overall comparison. In particular, one of them is FGSM.
- *Universal adversarial perturbations* [6] (Àlex), by S.M. Moosavi-Dezfooli et al., explains an useful method to generate universal perturbations, valid for several images in a dataset.
- *Shortcut Learning in Deep Neural Networks* [7] (Pablo), by Robert Geirhos et al., seek to link many of the current problems in deep learning as different symptoms of the same underlying problem: shortcut learning.

3 Problem statement and adversarial methods

This section covers the main concepts that will be needed throughout this paper. Concretely, we formalize the idea of adversarial examples and describe two of the most relevant methods for generating them.

3.1 Adversarial examples

Let $f : \mathbb{R}^m \to [0,1]^c$ be a classifier that maps image pixel value vectors to vectors, denoting the probability of the image of belonging to each of the c classes considered in a certain problem. The classification is done by the following mapping, which picks the label that best fits the input (the one with the maximum probability):

$$\begin{aligned} \hat{k}_f : \mathbb{R}^m &\longrightarrow \{1, \dots, c\} \\ x &\longmapsto \hat{k}_f(x) = \operatorname{argmax}_{i=1\dots c} f_i(x), \end{aligned}$$

We will refer to a classifier either by f or its corresponding discrete mapping $\hat{k}_f$, writing only $\hat{k}$ when the classifier is clear from the context.

We also denote

$$\text{loss}_f : \mathbb{R}^m \times \{1, \ldots, c\} \to \mathbb{R}^+$$

as the associated continuous loss function of the classifier f.

Given an image $x \in \mathbb{R}^m$ and a classifier f, the ℓ_p-norm *adversarial perturbation* is defined as the *minimal* perturbation r that is sufficient to change the estimated label of the image x:

$$\text{argmin}_r \, \|r\|_p \quad \text{s.t.} \quad \hat{k}(x+r) \neq \hat{k}(x). \tag{1}$$

Hence, to get an adversarial perturbation, we have to solve the former optimization problem, which is not an easy task.

Once we compute an adversarial perturbation r, the associated *adversarial example* will be the vector $x + r$. The solution r of the problem is not unique in general, but we denote one such $x + r$ for an arbitrarily chosen minimizer.

3.2 Adversarial methods

In this paper, we will focus on a couple of methods for generating adversarial examples: *FGSM* and *DeepFool*. Both methods are explained below.

3.2.1 Fast gradient sign method (FGSM)

This method is introduced in [3] (Pablo). It is a simple, analytical way to generate adversarial examples.

Given a classifier f, let $x \in \mathbb{R}^m$ be an input and $y = \hat{k}(x)$ the associated label. The idea is to linearize the loss function so as to obtain a constrained (depending on $\epsilon > 0$) adversarial perturbation given by

$$r = \epsilon \cdot \text{sign}\left(\nabla_x \, \text{loss}_f(x, y)\right).$$

The gradient of the loss function with respect to the input image indicates the direction in which the loss function increases its value faster and it can be efficiently computed using backpropagation.

This method does not fit the usual expression (1) for generating adversarial examples. In fact, it does not involve solving any optimization problem, but adding a small vector whose elements are proportional to the sign of the gradient of the cost function with respect to the input. This is actually interesting, because FGSM provides an efficient way to obtain adversarial examples.

Despite its efficiency, this method provides only a coarse approximation of the optimal perturbation vectors. In fact, it performs a unique gradient step, which often leads to sub-optimals solutions.

3.2.2 DeepFool

DeepFool [4] (Pablo) is introduced as a simple and accurate method to find adversarial examples. The basic idea behind it is to obtain adversarial perturbations by iteratively linearizing the classifier around the data point and finding the orthogonal projection onto a class-separating affine hyperplane.

In order to clarify the former idea, we are going to analyze first the case where f is an affine classifier, that is, $f(x) = w^\top x + b$, where $w \in \mathbb{R}^m$ and $b \in \mathbb{R}$. Here, the ℓ_2-norm adversarial perturbation of a point $x_0 \in \mathbb{R}^m$ corresponds to its orthogonal projection onto the hyperplane $\mathcal{F} = \{x : w^\top x + b = 0\}$. Therefore, the adversarial example is obtained by moving the data point $\frac{f(x_0)}{\|w\|_2}$ units (the distance between the point and the hyperplane) in the sense given by the unitary vector $\frac{-w}{\|w\|_2}$, so the adversarial perturbation can be computed as

$$r_*(x_0) := -\frac{f(x_0)}{\|w\|_2^2} w. \tag{2}$$

Assuming now that f is a general binary differentiable classifier, one can estimate the adversarial perturbation by using an iterative procedure. Concretely, at each iteration i, f is linearized around the current data point x_i by the function $x \mapsto f(x_i) + \nabla f(x_i)^\top (x - x_i)$, which is tangent to the classifier function. The class-separating hyperplane is then given by $\mathcal{F} = \{x : f(x_i) + \nabla f(x_i)^\top (x - x_i) = 0\}$, so the perturbation r_i can be computed as

$$\operatorname{argmin}_{r_i} \|r_i\|_2 \text{ s.t. } f(x_i) + \nabla f(x_i)^\top r_i = 0,$$

using the solution given in (2), with $w = \nabla f(x_i)$. The datapoint is then updated as $x_{i+1} = x_i + r_i$, repeating this until the datapoint changes the sign of the classifier. The complete algorithm is shown below in Algorithm 1.

Algorithm 1: DeepFool for binary classifiers.

```
Input  : Image $x$, classifier $f$
Output : Perturbation $r$
Initialize $x_0 \leftarrow x$, $i \leftarrow 0$
while $\operatorname{sign}(f(x_i)) = \operatorname{sign}(f(x_0))$ do
    $r_i \leftarrow -\frac{f(x_i)}{\|\nabla f(x_i)\|_2^2} \nabla f(x_i)$
    $x_{i+1} \leftarrow x_i + r_i$
    $i \leftarrow i + 1$
return $r = \sum_i r_i$
```

The former algorithm can be extended for the multi-class case, following the same idea. We are not going to explain the details, because the process is somewhat technical and is not relevant for our purpose. The corresponding algorithm for the multi-class case is detailed in Algorithm 2.

Algorithm 2: DeepFool.

```
Input  : Image $x$, classifier $f$
Output : Perturbation $r$
Initialize $x_0 \leftarrow x$, $i \leftarrow 0$
while $\hat{k}(x_i) = \hat{k}(x_0)$ do
    for $k \neq \hat{k}(x_0)$ do
        $w'_k \leftarrow \nabla f_k(x_i) - \nabla f_{\hat{k}(x_0)}(x_i)$
        $f'_k \leftarrow f_k(x_i) - f_{\hat{k}(x_0)}(x_i)$
    $\hat{l} \leftarrow \operatorname{argmin}_{k \neq \hat{k}(x_0)} \frac{|f'_k|}{\|w'_k\|_2}$
    $r_i \leftarrow \frac{|f'_{\hat{l}}|}{\|w'_{\hat{l}}\|_2^2} w'_{\hat{l}}$
    $x_{i+1} \leftarrow x_i + r_i$
    $i \leftarrow i + 1$
return $r = \sum_i r_i$
```

In practice, the final adversarial perturbation r is multiplied by a constant $1 + \eta$, with $\eta \ll 1$, in order to effectively reach the other side of the classification boundary.

In the previous algorithm we have considered the 2-norm, but any other norm can be used (see details in [4]) in order to obtain different results from a geometric point of view.

The algorithm may not converge to the optimal perturbation given by (1), since it operates in a greedy way. However, it provides good approximations and it is an efficient way to evaluate the robustness of classifiers.

4 Experiments and results

Now, we will perform some experiments using both FGSM and DeepFool. We have selected a state-of-the-art convolutional neural network, *InceptionV3*, as the model to be attacked. In particular, we are considering two different experiments for each method:

- First, we use the CIFAR-10 test set, composed of 10.000 32x32 colour images, divided in 10 classes. We will test both methods in some test images, as well as perform a complete attack (trying to generate an adversarial image for each one of the images on the test set), attempting to measure the effectiveness of the method and the robustness of the classifier.

  For this experiment, we use a CIFAR-10 pretrained version of the model that we found in `https://github.com/huyvnphan/PyTorch_CIFAR10`.

- Finally, we are going to use the ImageNet pretrained version of the model (included in `PyTorch`) to generate adversarial perturbations for natural images obtained from the internet, in order to test the methods in more realistic cases.

The objective is to analyse the performance of both methods and to gain a better insight into them.

You can find the code of these experiments in `https://github.com/pabloac31/ML-workshop`. You can also check the code for the adversarial methods, which have been developed in `PyTorch` based on the original implementations [3, 4] (Pablo).

4.1 Experiments using FGSM

Remember that FGSM uses the expression $r = \epsilon \cdot \text{sign}\left(\nabla_x \text{loss}_f(x, y)\right)$ to generate an adversarial perturbation for an image x with label y.

In order to make the perturbation both effective and barely visible in the image, we need to find a good value for ϵ. Using a small ϵ may lead the generated image not to be effective, but using large values instead make the resulting images be too noticeable.

In Figure 2 we can see some adversarial images generated with this method. For the dog image, we have taken $\epsilon = 0.01$, which is enough to deceive the classifier. It predicts a horse, despite the difference between both images is imperceptible for humans. In the second image, which is a bird, it is not enough to take $\epsilon = 0.01$. We had to increase the value up to $\epsilon = 0.15$ for getting an adversarial example. The resulting image is clearly modified.

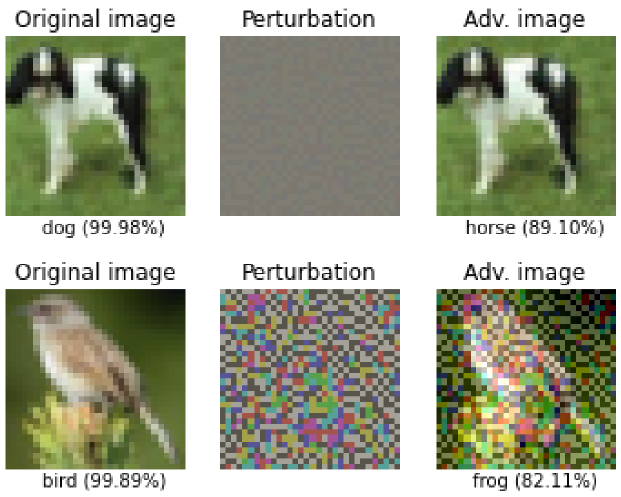

Figure 2: Adversarial examples generated using FGSM. Original images are on the left while adversarial images are on the right. The actual labels and confidences assigned by the model are shown in each case.

We have also performed an experiment using all the images in the test set. We consider the *fooling rate* as the percentage of modified images that actually fool the classifier, as a measure for the effectiveness of the method.

The results obtained are the following:

- *InceptionV3* obtains an initial accuracy of 95.41% on the CIFAR10 test set.
- Setting $\epsilon = 0.01$, we obtain a fooling rate of 44.25%.
- Setting $\epsilon = 0.1$, we obtain a fooling rate of 71.73%.

- The average time required to generate an adversarial perturbation has been 0.02 seconds.

We can see that higher values of ϵ lead to successful adversarial images, paying the price of having a clearly modified image. If we want the perturbation to be unnoticeable (i.e., a small value of ϵ), the adversarial images will fail in most than half of the cases.

Finally, Figure 3 shows an adversarial image generated using a high resolution image (obtained from the internet) using the ImageNet-pretrained version of the model, as explained before.

We can see that FGSM is able to generate a proper adversarial image in this case.

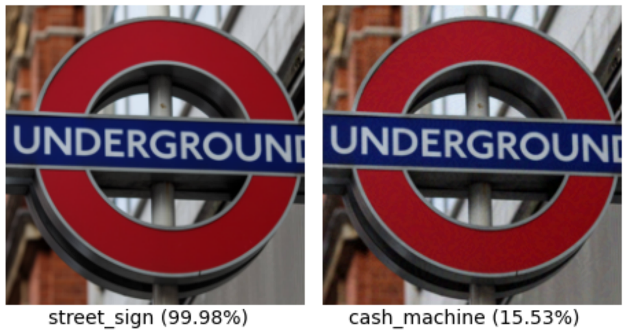

Figure 3: Adversarial example generated using FGSM on a natural image.

4.2 Experiments using DeepFool

As explained before, the final adversarial perturbation r generated with DeepFool is multiplied by $1 + \eta$ to ensure that it crosses the classification boundary. For this experiment, we set $\eta = 0.01$.

First of all, we show a couple of adversarial examples generated using DeepFool for some images from CIFAR-10 test set (see Figure 4). We can see that both generated images fool the classifier and the perturbations are barely perceptible on the nake eye (especially in the second image).

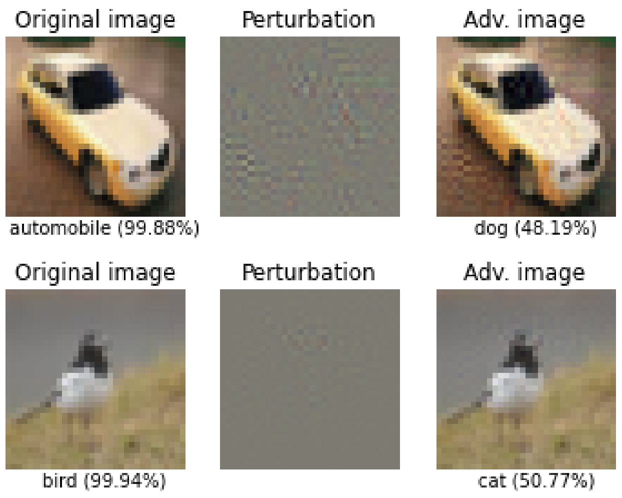

Figure 4: Adversarial examples generated using DeepFool in CIFAR10 images.

We have performed an attack to the model using the DeepFool method, obtaining the following results:

- As before, the initial accuracy of InceptionV3 on the CIFAR10 test set is 95.41%.
- The fooling rate is 92.33%.

- The average time needed to generate an adversarial perturbation has been 0.5 seconds.

In this case, the adversarial images generated with DeepFool are surprisingly effective, leading to a very high fooling rate, i.e., the model fails when classifying most of the modified images.

Finally, Figure 5 shows an adversarial image obtained using DeepFool on the previous high-resolution image.

We can see that, as before when using FGSM, the adversarial image is labelled as a *cash machine*, but this time, the model assigns 50.32% of confidence to its prediction, which is a larger value.

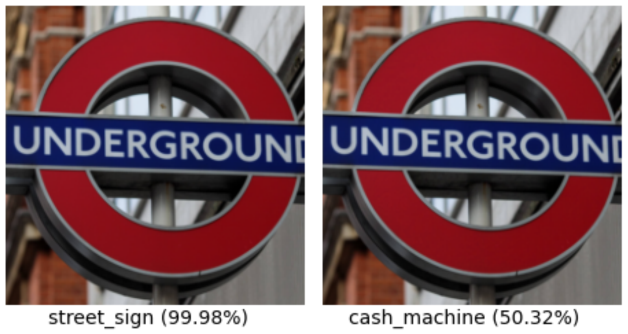

Figure 5: Adversarial example generated using DeepFool on a natural image.

5 Conclusions

After performing some experiments in the previous section, we can draw some conclusions regarding the adversarial methods we have tested.

We can state that FGSM is an acceptable method (more importantly, quick, since it performs a single gradient step) to generate adversarial examples. However, it has an important drawback, which is the dependence on the parameter ϵ. In fact, if we take a look at the expression $r = \epsilon \cdot \text{sign}\left(\nabla_x \, \text{loss}_f(x, y)\right)$, we clearly see that small values of ϵ will result in small modifications of the image, which will not be effective if the approximation given by the gradient of the loss function is not good enough. On the other side, higher values of ϵ will allow us to obtain an adversarial example easily, but the modifications in the image will be very brazen, and the results won't be applicable to real world.

On the other hand, DeepFool successfully fools the classifier in more than 90% of the images, so it is really a very effective method. In general, DeepFool computes small perturbations, which are difficult to distinguish with the naked eye.

We have also observed that DeepFool requires more time than FGSM, due to the iterative procedure and the operations needed by this method, but it is worth if we just consider the high fooling rates obtained.

References

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *International Conference on Learning Representations*, 2014. [Online]. Available: http://arxiv.org/abs/1312.6199

[2] N. Papernot, P. D. McDaniel, I. J. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against deep learning systems using adversarial examples," *CoRR*, vol. abs/1602.02697, 2016. [Online]. Available: http://arxiv.org/abs/1602.02697

[3] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2014. [Online]. Available: https://arxiv.org/abs/1412.6572

[4] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," 2015. [Online]. Available: https://arxiv.org/abs/1511.04599

[5] A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial examples in the physical world,” 2017. [Online]. Available: https://arxiv.org/abs/1607.02533

[6] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, “Universal adversarial perturbations,” 2016. [Online]. Available: https://arxiv.org/abs/1610.08401

[7] R. Geirhos, J.-H. Jacobsen, C. Michaelis, R. Zemel, W. Brendel, M. Bethge, and F. A. Wichmann, “Shortcut learning in deep neural networks,” 2020. [Online]. Available: https://arxiv.org/abs/2004.07780