Fooling a neural network with common adversarial noise

Nikola Popović*, Marko Mihajlović*

Abstract - these days deep Neural Networks (NN) show exceptional performance on speech and visual recognition tasks. These systems are still considered a black box without deep understanding why they perform in such a manner. This lack of understanding makes NNs vulnerable to specially crafted adversarial examples - inputs with small perturbations that make the model misclassify. In this paper, we generated adversarial examples that will fool a NN used for classifying handwritten digits. We start by generating additive adversarial noise for each image, then we craft a single adversarial noise that misclassifies different members of the same class.

I. Introduction

Artificial intelligence (AI), especially its subfield of machine learning (ML) became very popular over the last few years. A lot of experts today are saying that the next big breakthrough in AI will mark the new industrial revolution. There is more and more application of AI in many different industries as time goes by, which indicates that we should be very concerned about safety.

In the 2014, Szegedy et al. [1] discovered that you could make Adversarial Examples (AE) by adding carefully crafted small perturbations to the input and fool several ML models, including NN that have high accuracy on previously unseen data. Goodfellow et al. [2] explained this as a result of the models being too linear. With that kind of attack you could make an autonomous vehicle misclassify a stop sign, fool an intrusion-detection system, fool an identification system and so on. Most popular approaches for crafting AE are: the fast gradient sign method [2, 3] and Papernot et al. method [6]. On the other side, best strategies that are currently known for making models more robust to adversarial examples are: adversarial training [1, 2] and defensive distillation [7]. It turns out that none of these are fully successful and that an attacker can always find a new way to synthesize AE to overcome these defenses. One interesting type of attack that overcomes these defenses without knowing the models internals is the black-box attack [4]. It is also shown that it is possible to make AE that are robust to rescaling, translation and rotation [9]. Generating AE and defending against them is still an open research topic.

Goals of attackers are diverse. Some simply want the model to make a mistake while others aim to achieve class-targeted misclassification. There are relevant scenarios where an attacker can get a time limited access to the model internals. In those scenarios, he can create a common adversarial noise that can later be used to make the model misclassify any member of a class. Moreover, it is shown that you can train your own neural network to mirror another neural network. Then you can use your neural network to generate adversarial images that often fool the original network. This idea is explored by Papernot, et al. [4].

In section 2, we describe the NN that we fool with AEs. In section 3, we present algorithms for generating adversarial noise and show their results. In the last section, we conclude the paper with a few of our thoughts.

The code, which was used for all experiments, can be found at: https://github.com/Maki94/cnn_adv_examples.

II. Neural network

In recent years Neural Networks (NN) have become widely popular due to their universal character (used for supervised, unsupervised and reinforcement learning) and increasing affordability of powerful distributed systems capable of processing large amount of data. A NN is composed of layers which contain units called neurons. Each neuron is parameterized by its weight vector w_{ij}. The NN also has a nonnegative cost function $J(W)$ assigned to it. During the learning phase, a network is given training data, input-output pairs (x, y), and the goal is to adjust the set of weights W in a way that minimizes the cost function $J(W)$ by using some optimization algorithm.

First, we need a classifier that we want to fool. We use a specific type of NN - Convolutional Neural Network (CNN) to classify images of handwritten digits [10]. We used the famous MNIST dataset [11] which contains 70,000 28x28 pixel images of digits. Each pixel has one of 256 intensity values evenly distributed between 0 and 1. We divide the data into the training set (55,000 images), validation set (5,000 images) and the test set (10,000 images).

N. Popović is with the Department of Signals and Systems, Faculty of Electrical Engineering, University of Belgrade, Bulevar Kralja Aleksandara 74, 11000 Belgrade, Serbia, E-mail: pn173042m@student.etf.bg.ac.rs

M. Mihajlović is with the Department of Computer Science, Faculty of Electronic Engineering, University of Niš, Aleksandra Medvedeva 14, 18000 Niš, Serbia, E-mail: marko.mihajlovic@elfak.rs

* Equal contribution. Author ordering determined by coin tossing at https://www.random.org/coins/.

We trained our CNN on the training set and evaluated it on the test set. The validation set will be used later for crafting the adversarial noise shared amongst the whole class. The network has the architecture described in Table I.

The network was implemented in Python 3.5, using the *tensorflow* library. For the cost function we used cross-entropy. We minimized it using the Adam optimizer [5]. The training lasted 20,000 iterations and the weights were updated using a random batch of size 50 in each iteration. After the training, the NN had an accuracy of 99.19% on the test set.

TABLE I
Architecture of the CNN classifier

Layer type	Layer parameters
Input	28x28 image
Convolutional	32 filters(5x5); ReLU activation
Max pooling	2x2
Convolutional	64 filters(5x5); ReLU activation
Max pooling	2x2
Fully connected	1204 neurons; ReLU activation
Dropout	$p = 0.5$
Fully connected	10 neurons; Softmax activation

III. ADVERSARIAL NOISE

In the first hidden layer each neuron computes the dot product between the input x and its weight vector w_{ij} and then applies an activation function to the product. If we take the input, add a small perturbation and align its signs with the signs of w_{ij} along each dimension, we get an increase in the value of the dot product which is proportional to the dimensionality of the data. If the dimensionality is big enough, many small changes can add up to one big change. This can make the wrong neurons fire or stop firing, which can result in misclassification. The bigger the dimension of the data, the smaller the perturbation needed to make the model misclassify.

While generating adversarial examples we used the following iterative algorithm:

$$x_{adv}^{i+1} = Clip_{x,\varepsilon}\{x_{adv}^{i} + \alpha \nabla_x J(x_{adv}^{i}, y_{true})\}$$
$$x_{adv}^{0} = x \tag{1}$$

The algorithm stops when x_{adv}^{i+1} achieves a misclassification, or when it reaches the maximal number of iterations. Function $Clip_{x,\varepsilon}\{\cdot\}$ is a pixel-wise clipping of the image that assures the AE stays in the ε-neighborhood of the original image:

$$Clip_{x_j,\varepsilon}\{x_j^i\} = \begin{cases} x_j + \varepsilon\,, & x_j^i > x_j + \varepsilon \\ x_j - \varepsilon\,, & x_j^i < x_j - \varepsilon \\ x_j^i\,, & otherwise \end{cases} \tag{2}$$

In other words, we want to maximize the cost function over the input image, but we don't want the adversarial image to differ too much from the original one.

On the other side, to achieve target-label misclassification, the following iterative algorithm can be used:

$$x_{adv}^{i+1} = Clip_{x,\varepsilon}\{x_{adv}^{i} - \alpha \nabla_x J(x_{adv}^{i}, y_{target})\}$$
$$x_{adv}^{0} = x \tag{3}$$

Now, we are minimizing the adversarial cost function over the input image, which pushes the model to classify the AE as y_{target}. Also, the algorithm could be modified to stop when a specified classification confidence is reached, instead of stopping as soon as the model outputs y_{target}.

We didn't use one of the popular gradient sign methods that are presented in [3], because besides the information about the direction in which to move that is contained in the gradients sign, its values inform us how relevant each pixel is for making the model misclassify. This means that the intensity values of pixels that are not too relevant won't change as much as the intensity of the relevant ones. We think that this will result in a less damaged image while paying the price of more iterations. Analyzing the difference in their performance can be a topic for further investigation.

To measure how strong the noise e is, we will use the Root Mean Square (RMS) metric:

$$RMS = \sqrt{\frac{1}{n}\sum_{i=1}^{n} e_i^2} \tag{4}$$

$$e = x_{adv} - x \tag{5}$$

We made AE's for the whole test set using the algorithm described in (1). We used the clipping value of $\varepsilon = 0.15$ and the learning rate of $\frac{10}{255}\frac{1}{m}$, where m is the maximum absolute value in the gradient vector. We generated the noise using 200 iterations for the maximum iteration parameter. In the 98.6% of the cases, the model makes a mistake, while the average RMS of the noise was 0.063. An example of a misclassified image is shown in Fig. 1.

Fig. 1. Left: original image. Right: AE.

As we can see, an average human would still label the image as a seven, but the NN, which has a very high accuracy on previously unseen data, made a mistake.

Let's now imagine an attacker gets a time limited access to a protected model. He can use that time to learn a common adversarial noise that can fool the model when applied to different members of the same class. We propose an algorithm for crafting such a noise on a set of N members from the same class:

$e = \vec{0}$
for $j = 1$ **to** *epochs*:
  for $i = 1$ **to** N:
    $e := Clip_{0,\varepsilon}\{e + \alpha \nabla_x J(x^{(i)} + e, y_{true})\}$
  end
end

Alg. 1: Algorithm for finding the common adversarial noise

This is similar to stochastic gradient descent. When we craft an adversarial noise e for a single image, we are updating the noise using only that image until we end up with a misclassification. Here, we update the noise by a little using each image and repeat that process until we find a good enough noise. By updating it a little on each image, we are forcing it to find a weak spot that is common for all the data points in that class.

We generate common adversarial noise for the class of 5's on the data in the validation set over 25 epochs with the learning rate of $\frac{1}{255}\frac{1}{m}$. We generate the noise for different values of the clipping value ε. For each generated noise, we apply it to all of the data points in the test set from the class of 5's and measure its success rate of fooling the network and the RMS of the noise signal. Results are shown in Figures 2 and 3.

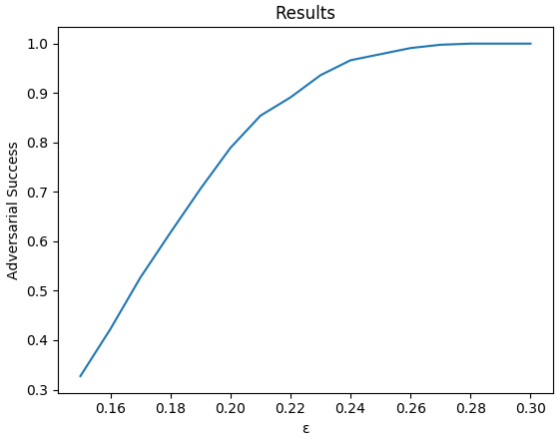

Fig. 2. Adversarial success for different clipping values

Fig. 2 depicts that for $\varepsilon = 0.24$ and above the model is fooled most of the time. As we can see from Fig. 3. the RMS of the noise rises almost linearly with ε. This makes sense because as we loosen the box constraint, the perturbation get bigger in order to achieve a larger adversarial success.

We then generate common adversarial noise for all the classes, on the validation set, over 25 epochs with the clipping value of $\varepsilon = 0.24$, the learning rate of $\frac{1}{255}\frac{1}{m}$ and evaluate it on the test set. Results are shown in Fig. 4 and Fig. 5.

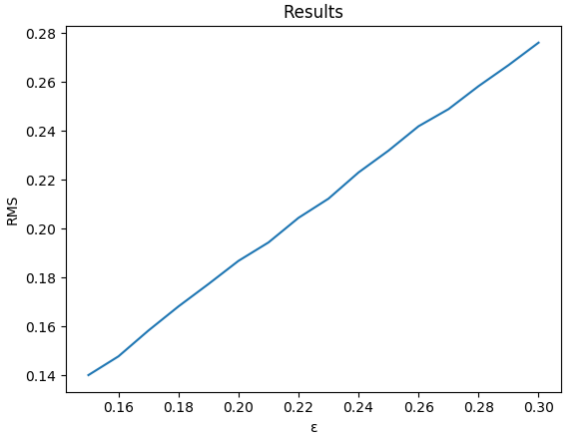

Fig. 3. RMS for different clipping values

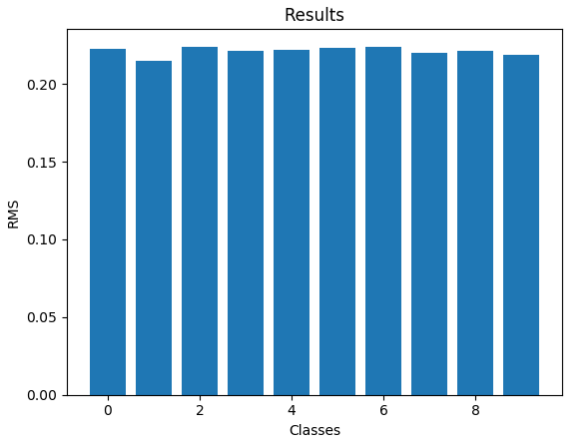

Fig. 4. RMS for different classes with the clipping value of $\varepsilon = 0.24$

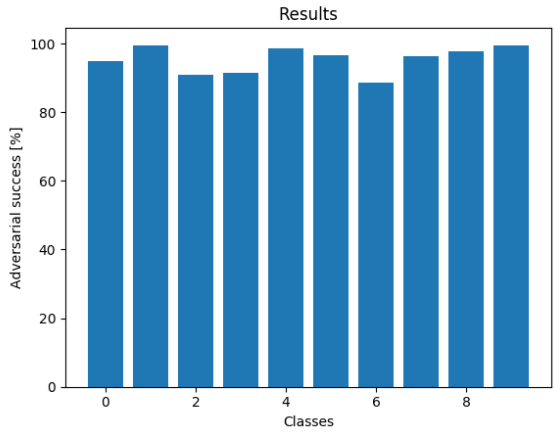

Fig. 5. Adversarial success for different classes with the clipping value of $\varepsilon = 0.24$

We infer that the algorithm can make a single noise that we can apply to different inputs from the same class, which were not involved in its making, and still fool the model most of the time. As Fig. 4. and Fig. 5. depict it is possible to create a successful common noise for each class. Also, adversarial success and RMS do not vary too much for all the classes. This tells us that making common noise for different classes is a problem of approximately the similar difficulty for digit classification.

This is a serious safety problem. The noise has a larger RMS than the noise crafted for one specific input, because crafting a more robust noise requires larger perturbations. A few AE with the generated common noise are shown in Fig. 6.

As we can see in Fig. 6, now AE are much more damaged than in the case of generating an AE on a single digit. However, a person would still classify them as their correct label or maybe discard them with an explanation that they are too damaged and do not resemble any digit. We think that this is because MNIST digits have a low resolution and that the perturbations would be much smaller on images with higher resolution. This could be a topic for future investigation.

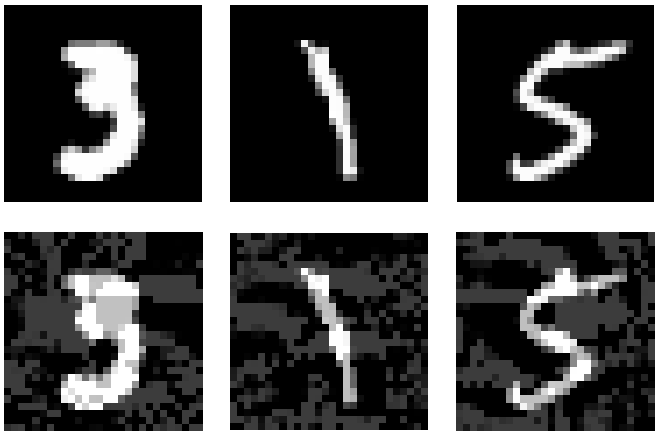

Fig. 6. Above: original images. Below: AEs

IV. Conclusion and Future Work

In this work, we have generated AE similar to the original data that are able to fool a CNN without a problem. Afterwards, we proved that it is possible to create a common adversarial noise that can be used to fool a CNN with great success. The purpose of this paper is to increase awareness about the drawbacks of modern ML algorithms that are used increasingly in all walks of life.

While working on this paper, we stumbled upon a few questions that we think are worthy of investigation. Maybe this can inspire some other authors to work on those problems, which are:

- Analyzing the difference in the performance of algorithms (1, 3) from the section 2 and the gradient sign algorithms.
- Crafting common adversarial noise on images that have a larger resolution and multiple color channels.
- Trying to craft common adversarial noise in the purpose of target-label misclassification.
- Trying to make common adversarial noise that is robust to rescaling, rotation and translation

Acknowledgement

The authors would like to thank Goran Dubajić and Bruno Gavranović for introducing us to the topic of AE and providing advice throughout experiments.

References

[1] Christian Szegedy, et al., Intriguing properties of neural networks. In *Proc. Proceedings of the International Conference on Learning Representations*, 2014.

[2] Ian J Goodfellow, et al. Explaining and harnessing adversarial examples. *In Proceedings of the International Conference on Learning Representations*, 2015.

[3] Alexey Kurakin, et al. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*, 2016.

[4] N. Papernot, et al. Practical black-box attacks against deep learning systems using adversarial examples. *arXiv preprint arXiv: 1602.02697*, 2016.

[5] Diederik P. Kingma, et al. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference for Learning Representations*, San Diego, 2015.

[6] N. Papernot, et al. The limitations of deep learning in adversarial settings. In *Proceedings of the 1st IEEE European Symposium on Security and Privacy*. IEEE, 2016.

[7] Nicolas Papernot, et al. Distillation as a defense to adversarial perturbations against deep neural networks. In *Proceedings of the 37th IEEE Symposium on Security and Privacy.*

[8] N. Papernot, et al. Towards the Science of Security and Privacy in Machine Learning. arXiv preprint arXiv: 1611.03814, 2016.

[9] A. Athalye, et al. Synthesizing Robust Adversarial Examples. arXiv preprint arXiv: 1707.07397, 2017.

[10] A. Krizhevsky, et al. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[11] Y. LeCun et al. The mnist database of handwritten digits. 1998.