Adversarial Reprogramming

Luigi Paolo Pileggi Alexandru Sarpe Mattia Nicolella

June 5, 2019

1 Introduction

In recent decades the use of neural networks in numerous fields has grown exponentially, due to their versatility and effectiveness. With the increase in sensitive tasks entrusted to neural networks, numerous types of attacks were born.

One of the most famous attacks on neural networks consists of adversarial examples. An adversarial example is crafted by taking a valid input element of the network and creating a small perturbation so as to bring the classification carried out by the network to an incorrect result. These type of attacks are subdivided in two categories:

- Untargeted: Which leads to the misclassification of the prediction of the model without aiming at a specific output.
- Targeted: Which produce a perturbation to the input of the model in order to produce a specific output of the classification.

In particular this last type of attack is very dangerous and can cause disastrous effects. For example it might allow the attacker to steal data or money, managing to trick an authentication system based on face or voice recognition, or might allow the attacker to perform an attack on medical equipments managed by neural networks, aiming to hurt or kill patients.

However, this approach can only produce previously configured outputs, not allowing to perform other tasks than those for which the model was created. Through their research Elsayed et al.[1] introduced a different type of attack called adversarial reprogramming. The goal of this method is to reprogram the model in order to perform a new task chosen by the attacker. The reprogramming of the model does not require the attacker to compute the desired output.

The methodology introduced by Elsayed et al.[1] is based on two adversarial reprogramming functions $h_f(*;\theta)$ and $h_g(*;\theta)$. The first function $h_f(*;\theta)$ is used to convert an input chosen by the attacker to a valid input of the model. The second function $h_g(*;\theta)$ is used to map the output of the model to the output desired by the attacker. The hyper parameter θ is trained in order to perform the correct task chosen by the attacker.

The adversarial reprogramming performed by Elsayed et al.[1] is applied on image classification, in particular on Networks pre-trained on the ImageNet dataset. The adversarial task consist in the classification of the images of the MNIST and the CIFAR10 datasets through the reprogramming of these pre-trained Networks.

In this work we will discuss our implementation of the Adversarial Reprogramming attack. In the section 2 we will discuss the methodology used, while in the section 3 we will show our result performed on the MNIST and the CIFAR10 datasets. Finally in the section 4 we will discuss our results.

2 Background Work

As presented in the original work, our adversarial reprogramming has several similarities with other works that we will shortly illustrate.

2.1 Adversarial Examples

Our work is an extension of what can be done with Adversarial Examples, where specific inputs are crafted to induct a mistake in the target machine learning model; which can be a specific mistake if the example is targeted or any mistake is the example is untargeted. Our work is aimed to produce a specific functionality rather than a specific result when one of our example is fed onto the target model. We use an adversarial program which will cause the model to process the given images given according to the adversarial program scope, instead of processing them as the model scope was intended.

2.2 Weird Machines

Weird machines are computational artifacts that when execute crafted code have a behaviour that is not intended by the developer. Mitigation of these problems usually require protecting the machine from memory errors, which can give the attackers means to inject crafted instructions and input validation, to verify that the instruction executed are the intended ones. Our work is closely related to weird machines because we use crafted inputs to make a target model execute an unintended operation, which in the experiments conducted is the classification of other objects not in the dataset. It must be taken in consideration that this method does not inject any code, it only leverages the communication protocol.

2.3 Parasitic Computing

Parasitic computing is a technique where in an interaction between two or more programs one manages to get the other program to perform complex computation. This idea[2] was presented with a pair of programs, one which was solving a 3-SAT problem and the other which was verifying TCP packets with their checksums. The first program had decomposed the problem in a way that a packet and a checksum value were equivalent to a 3-SAT clause, so by verifying the checksum the second program was contributing to solve the 3-SAT problem; without knowing anything about the general problem. Our work is very similar since our crafted input contains an image that must be classified and when given to the model we can induce the model to perform this classification, without doing any modification to the model itself.

3 Methods

We generate an adversarial program, which is the same for all the samples in the dataset, that is then passed along the network input. We define our adversarial program as:

$$P = \tanh(W \cdot M)$$

Where $W \in \mathbb{R}^{n \times n \times 3}$, n is the ImageNet width, and M is a mask to improve visualization of the action of the adversarial program.
Note that:

- We use the *tanh* function to bound the adversarial perturbation in the range of (-1, 1).
- The mask is not required.

Then for each image in the dataset we create the corresponding adversarial image by placing it in the area defined by the mask:

$$X_{adv} = h_f(\tilde{x}; W) = \widetilde{X} + P$$

We then define a map h_g which relates each class of our adversarial task to the one of the classes of the ImageNet dataset.
Let $P(y|X)$ be the probability that an ImageNet classifier gives to ImageNet label $y \in \{1, \ldots, 1000\}$, given an input image X. We want then to maximize the probability $P(h_g(y_{adv})|X_{adv})$ so we set up our optimization problem as:

$$\hat{W} = \underset{W}{\operatorname{argmin}}(-log(P(h_g(y_{adv})|X_{adv})) + \lambda||W||_F^2)$$

Here λ is the coefficient for a weight norm penalty, to reduce over fitting. We optimize this loss with Adam[3] while exponentially decaying the learning rate.

4 Results

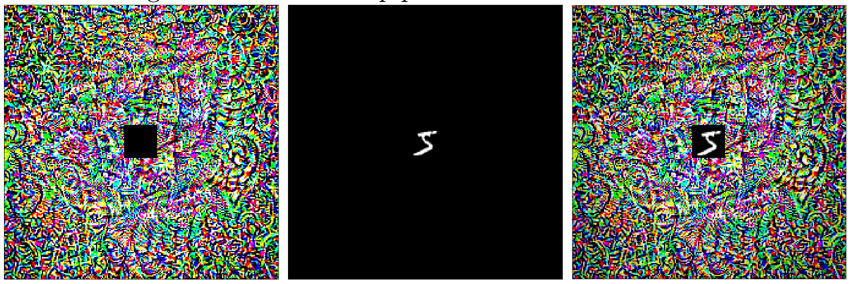
We tested the above described methodology on three network architectures trained on ImageNet. In each case, we successfully reprogrammed the network to perform three adversarial tasks:

- MNIST classification
- CIFAR-10 classification
- Concealed MNIST classification

4.1 MNIST classification

For this adversarial task we embedded MNIST[5] digits of size $28 \times 28 \times 3$ inside a frame representing the adversarial program, we then assigned the first 10 ImageNet labels to the MNIST digits, and then trained the adversarial program. We show an example in Figure 1.

Figure 1: Adversarial pipeline for MNIST classification



4.2 CIFAR-10 classification

For this task we trained our program to perform CIFAR-10[4] classification. Our achieved accuracy is near what is expected from typical fully connected networks [6] but with minimal computational cost from our side at inference time.

4.3 Concealed MNIST classification

Here, we add constrains on the size and scale of the adversarial data by concealing it behind a cover image taken from the ImageNet dataset. We achieved slightly lower performance from the first adversarial task but nonetheless the program successfully classified the MNIST images even after shuffling the pixels around. For this task we had to change the method we generate the adversarial program and input of the network:

$$P_X = \alpha \tanh\Big(\text{shuffle}_{ix}(\widetilde{X}) + (W \cdot \text{shuffle}_{ix}(M))\Big)$$
$$X_{adv} = clip(X_{ImageNet} + P_X, [0,1])$$

Where ix is the shuffling sequence (same for M and $\forall X$), α is a scalar used to limit the perturbation scale, and $X_{ImageNet}$ is an image chosen at random from ImageNet, which is the same for all MNIST samples.

Figure 2: Adversarial pipeline for concealed MNIST classification

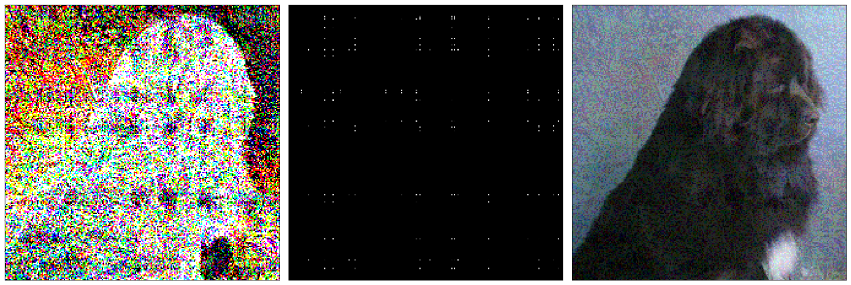


Figure 3: Training accuracy MNIST, CIFAR-10, Shuffled MNIST

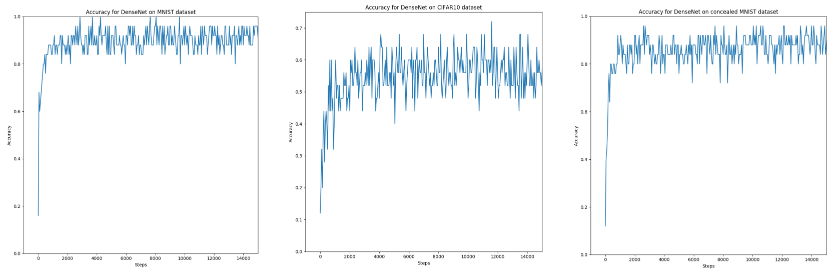


Figure 4: Training loss MNIST, CIFAR-10, Shuffled MNIST

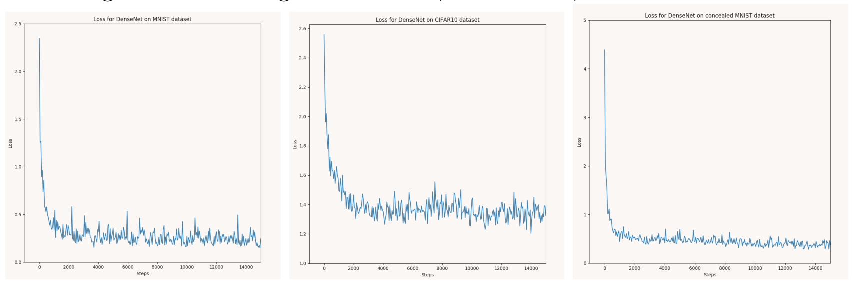


Table 1: Top-1 accuracy for the three tasks on the test set

	MNIST	CIFAR-10	Shuffled MNIST
DenseNet121	0.9714	0.6947	0.9684
Inception V3	0.9651	0.6883	0.9517
InceptionResNetV2	0.9637	0.6792	0.9578

5 Conclusion

In this work we have successfully reached results which are comparable to the work done by Elsayed et al.[1] in reprogramming a pre-trained Neural Network to perform a classification for which the model was not trained for. This show that these networks can behave like weird machines if we conceal the input.

References

[1] Elsayed, G. F., Goodfellow, I., Sohl-Dickstein, J. 2018. Adversarial Reprogramming of Neural Networks. arXiv e-prints arXiv:1806.11146.

[2] Albert-Laszlo Barabasi, Vincent W. Freeh, Hawoong Jeong and Jay B. Brockman. Parasitic Computing *Nature. 412. 894-7. 10.1038/35091039.*

[3] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, 2014; *arXiv preprint arXiv:1412.6980.*

[4] Krizhevsky A. Learning Multiple Layers of Features from Tiny Images; 2009. `http://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`

[5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition; *Proceedings of the IEEE, 86(11):2278-2324*, November 1998. `http://yann.lecun.com/exdb/publis/index.html#lecun-98`

[6] Lin, Y.-C., Hong, Z.-W., Liao, Y.-H., Shih, M.-L., Liu, M.-Y., Sun, M. 2017. Tactics of Adversarial Attack on Deep Reinforcement Learning Agents. arXiv e-prints arXiv:1703.06748.