Multi-Target Perturbations

Submitted by: Elior Nehemya

Abstract:

Universal perturbations showed unique generalization properties and raise many questions on the security of machine learning. Unlike targeted attacks, universal perturbations send each data point to the easiest class without any control of the adversary. In this work we suggest a new algorithm to create targeted universal perturbations that allows the attacker to direct its universal perturbation into a specific class. In addition we make a big step towords multi-target perturbations that let the attacker the ability to pull different source-target pairs with a single perturbation. To do so we evaluate three methods to combine source-target pairs constraints into one multi-targeted perturbation.

Introduction:

At the early stages of adversarial learning, researchers started to notice strange behavior of their learning models. They saw images that are easily classified by humans, but misclassified by their models, and even with high confidence. This phenomenon raises many questions about the reliability of learning models.

Szegedy et al. [1] at 2013 were the first to state the term adversarial example and even formulate an algorithm to craft those malicious samples. In a white-box scenario, for a given trained classifier $f(\cdot)$ an adversary seeks to find a valid perturbation v, when added to input x implies $f(x+v) \neq c$ i.e misclassified as the real class c. The way adversaries solve this optimization problem varies from different adversarial attack methods, yet most of them rely on the back-propagation process of the learning model.

The research on adversarial examples gain a lot of attention and now there are many adversarial attacks in white box case e.g FGSM, BIM, C&W [2],[3],[4] and even in black box case e.g. where papernot et al. [5] was the first to produce an interesting attack with a surrogate model instead of the original network.

What takes our attention is a revolutionary work by Moosavi et al. [6] the Universal Adversarial Perturbation. This work presents for the first time a unique perturbation that can fool an image classifier on most of the data samples which give the attack its universality once and twice where those perturbations can transfer to different networks. This is very unique since until then the perturbation was always directed to a specific data sample. It allows the attacker to perturb unseen data samples which can be crucial to time series especially.

Although [6] showes remarkable capabilities, it is only an untargeted attack, which means that the attacker has no control over its perturbation. In that matter, it makes the attack less practical for cases where the attacker needs to gain control of its perturbation and not just create a DOS attack.

In this work, we will start a journey to create a Multi-Targeted Universal Perturbation, a powerful attack in which the attacker can force a model to predict specific labels for specific source classes which gives the attacker the top capabilities as an adversary. We start by producing a targeted variant universal perturbation algorithm, and later we use our attack to create and combine two class movment perturbations. We show 3 methods to combine those perturbations into one multi-target universal perturbation. Our best attack obtains a 44% targeted fooling rate with a 56% leak. Later we suggest further steps to extend this work.

Research Question:

In this research, we will try to understand how to create a multi-target universal perturbation that moves most of the samples to the desired target where the target can be different for each class?

Hypothesis:

The untargeted attack allows the perturbation to pull for any other class when the targeted version is much more restricted. To make the targeted attack work we must utilize inner knowledge of the decision boundaries.

Experiment:

In order to test our hypothesis, we made three major steps. First, we created a targeted universal perturbation attack, which enhances the original [6] algorithm to a targeted variant. Second, we evaluate our attack on different targets (classes). Third, we comper three different candidates for our final multi-target universal perturbation attack. Next, we will describe the settings for our experiments.

Dataset: we chose to evaluate our attack on CIFAR-10 [7], a well-known image classification dataset. CIFAR-10 consists of 60000 color images in 10 classes, each image has 32x32 pixels and each pixel has 3 channels (RGB). There are 50000 images for training, and 10000 images for testing. Each class has 6000 images and the train and test sets are balanced (same amount of images per class).

Preprocessing: color pixel has 3 channels and each one can get a natural value between 0-255. In order to help the model to converge faster and more accurately, we used a min-max normalization to a 0-1 continuous scale. The input shape of the image after preprocessing is a three-dimensional matrix with shape 32x32x3 where each value is between 0-1.

Detailed description: as far as we know there are no previous successful targeted universal perturbation attacks and especially no multi-target universal attacks. Therefore, we first suggest our targeted variant for a universal perturbation attack. Given a set of data points X and a classifier $\hat{f}$ we iteratively craft our targeted universal perturbation v. At each iteration, we compute the minimal perturbation v_i such that $Acc(X_B + v + r\ , y) > \delta$, than we update our universal perturbation v by projecting $v + v_i$ into epsilon ball. We denote $Acc(x, y)$ as the targeted accuracy i.e. the percent of samples from x that the model predicted the desired target y as specified by the attacker. Later we use a projection step to controls the magnitude of v to a maximal ϵ size.

Computation of targeted universal perturbation:

```
1. Input: Data points $X$, target class $y$, classifier $\hat{f}$, maximal $L_2$ norm of perturbation $\epsilon$, desired
   accuracy $\delta$ on perturbed data.
2. Output: universal perturbation vector $v$
3. While $Acc(X+v,y) < \delta$ :
4.     For each batch $X_B \subset X$ :
5.         if $Acc(X_B + v\ ,y) < \delta$ :
6.             Compute the minimal perturbation as follows:
                   $v_i \leftarrow argmin_r \|r\|_2 \;\; s.t. \;\; Acc(X_B + v + r\ ,y) > \delta$
7.             Update the perturbation $v$:
                   $v \leftarrow Projection_{2,\epsilon}(v + v_i)$
```

As we can see the algorithm halts when we reach the desired accuracy on the malicious targets. Unlike the original algorithm, we propose to optimize our goal in batches of images and not to use a single image at a time, this proved to be more efficient in the tests. Also, we chose to use a targeted BIM attack to calculate the “local” perturbation in line 6 in our implementation although it is not mandatory and we can use any other gradient-based attack.

What makes this algorithm so versatile is the fact that the attacker can choose the target y and more interestingly he can choose the composition of the data points X which has a great influence on the resulted targeted universal perturbation. The exact composition of X and the target label y will define the direction of the resulted perturbation and they are the key to gain a powerful perturbation that can fool most of the samples in the test set.

We start our experiment by testing our proposed targeted variant and to set a baseline for the later experiment. To do so we create ten targeted universal perturbation (TUAP), one for each target class. The training set X for each TUAP will contain twenty images from each class, to a total of 200 images. We will use our attack on our pretraind CNN model, with $\varepsilon = 1.6$ and a desired targeted accuracy $\delta = 0.95$. We later show our targeted and untargeted fooling rate for each TUAP.

In the entire research, we use a simple CNN model as the attacked model that was build and used after training on CIFAR-10. Our CNN model contains 12 convolution layers and a softmax layer at the end, a full structure can be found in the appendix. The model obtains 83% accuracy on the CIFAR-10 test set. The model was built with Keras API and hence we can obtain its gradient for later attacks.

The second experiment makes a big step towards our desired multi-target universal attack. In general, our goal here is to create two TUAPs where each seeks to pull different source classes to different target classes, and then we compare between three methods to combine those two TUAPs into one multi-targeted universal perturbation. We start by creating two TUAPs but now we use a special composition for our train set. The first TUAP will train on X which contains 200 images that belong to the ‘deer’ class and the target class will be ‘horse’, we will denote this TUAP as TUAP1. The second TUAP will train on X which contain 200 images that belong to the ‘bird’ class and the target class will be ‘airplane’, we will denote this TUAP as TUAP2.

Now we propose three methods for combining the two TUAPs. First, we suggest combining them with a random grid masking, which means that half of the resulted combination input will belong to TUAP1 and half will belong to TUAP2. The grid masks will be chosen randomly without overlapping. Second, we suggest a region masking method that will take half of each TUAP and combine them to one perturbation. The right part will belong to TUAP1 the left part will belong to TUAP2. Third, we suggest to let the gradient to compose its combination. In this method, we use our attack again but the training set X will have 200 images from each source and the target labels will be equal to the targets for each TUAP (deer -> horse, bird -> airplane).

After using our combining methods we will compare the results for each method for the untargeted/targeted fooling rate, and we will report the leak effect as well. Now we observe two constrains (two directions of UAP) in one TUAP therefore we will evaluate the leak effect witch is the untargeted fooling rate for the other classes (not the source class). To farther analyze the effect of our combining methods we will compute a confusion matrix for each method. This experiment is a small yet mandatory step to get into the complex idea of multi-targeted universal perturbations.

<u>Evaluation:</u> moving from an untargeted UAP attack to a multi-targeted attack variant raises many questions on how to evaluate this attack?. In our opinion, the answer depends on the attacker's needs. Since we are just at the beginning of opening this Pandora box we will use both untargeted and targeted fooling rates to feel this movement. In addition, we will use the confusion matrix to farther understand the UAPs effect on the model.

<u>Results:</u>

Now we will show the results for our two major experiments. We start by evaluating the untargeted fooling rate and the targeted fooling rate for each TUAP.

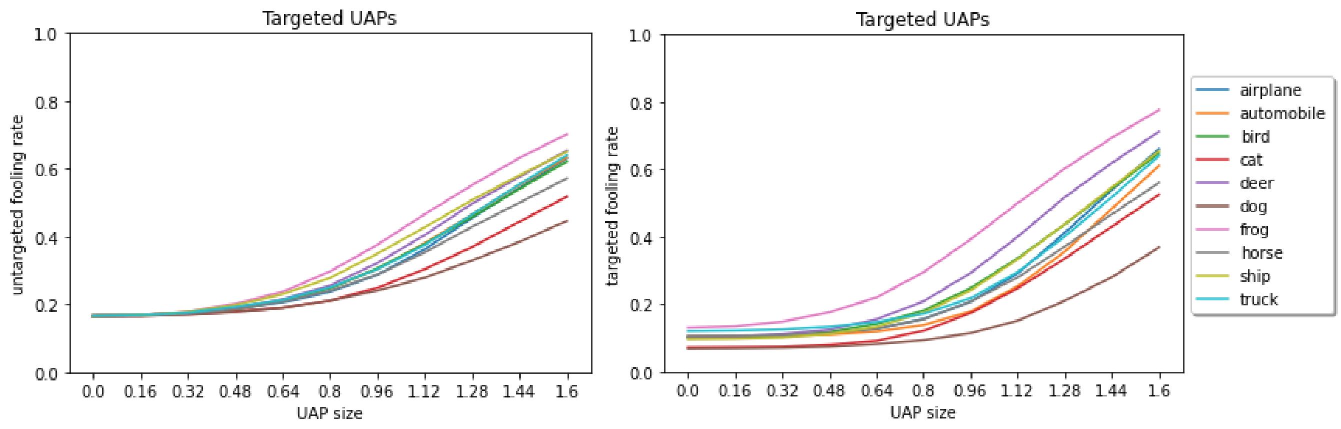


Figure 1: Targeted UAP for each class

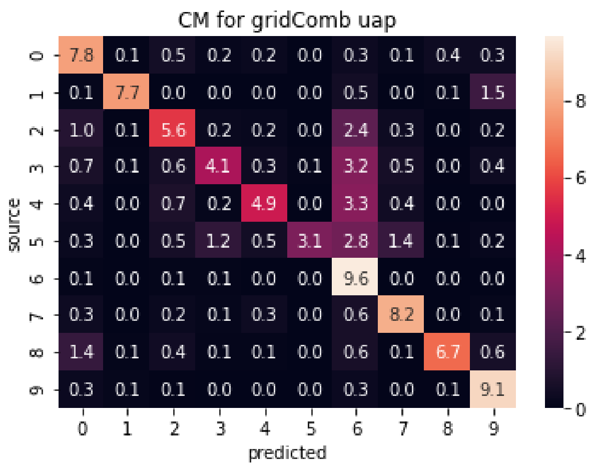
The X-axis shows the UAP size from 0 to 1.6 as the maximum size. The results show that as the size grows the fooling rates rise as well, and we can see that several classes more likely to fooled into than others.

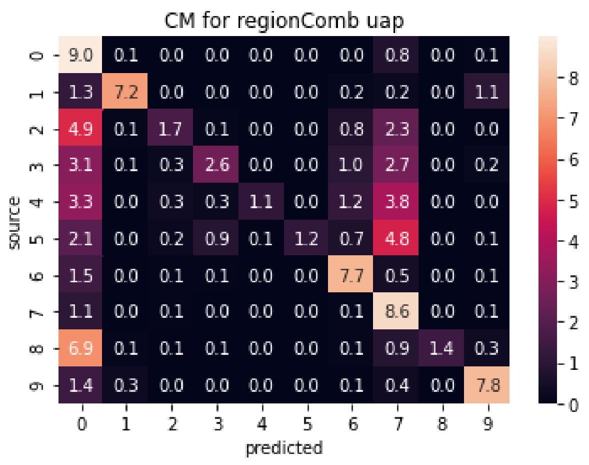
The second experiment compares three methods to combine two TUAPs, and later we visualize them through the confusion matrix.

Perturbation	**Untargeted fooling rate**	**Targeted fooling rate**	**Leak effect**
TUAP1	0.87	0.76	0.52
TUAP2	0.91	0.87	0.54
gridComb	0.4	0.064	0.29
regionComb	0.84	0.42	0.43
gradComb	0.88	0.44	0.56

Table 1: Comparison between the combining methods

In table 1 we can see for each perturbation the untargeted\targeted fooling rate and the leak effect. TUAP1, TUAP2 are the original perturbations that pull 'deer'->'horse', 'bird'->airplane' accordingly. The gridComb,regionComb,gradComb shows the three methods used grid, region, gradient accordingly. We can see that the first two perturbations have higher fooling rates than the combined methods, and we can see that there are differences between the methods which we later discuss.





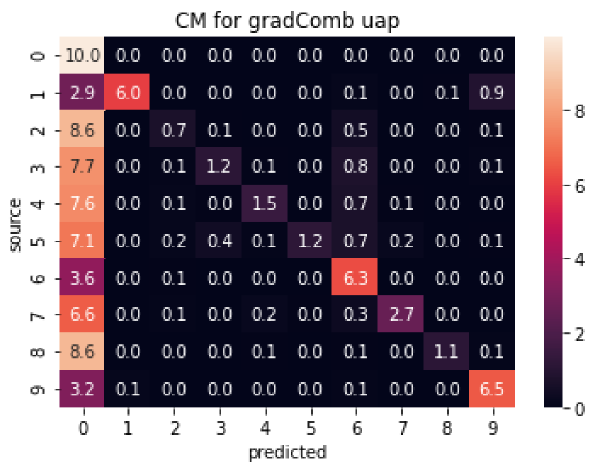


Table 1 gives us limited knowledge of the attack and therefore we visualize the confusion matrices (CM) of the three methods. Each CM shows for each source class what the model predicted in percentages. This provides much more information on the effect of the perturbation that a raw fooling rate.

Conclusions:

As we can see from figure 1 our targeted variant can fool the model to predict for around 60% of the test images for any target class the attackers choose. This is a very optimistic start for our journey to multi-target perturbations which has higher complexity and constraints. We can see that not all the target labels are equally fooled (80% for frog, 40% for dog). This property may strengthen the thinking that some labels are more dominant than others and have a bigger area in the decision hyperplane.

In the second experiment, our goal was to determine which among the three methods would be the best technique to combine two (or more) requirements into one perturbation. Regardless of the method used we see that the attack fooling rates (untargeted&targeted) suffer a decline, which is very understandable since we are satisfying a more complex requirement. The combined perturbation has to pull for two different targets instead of one. The first method appeared to be less efficient to the targeted task since we get only 6.4% targeted fooling rate while the other two methods obtain more than 40%. In our opinion this is highly related to the way the CNN model scans the images, the convolutional layers extract features, not by one pixel, but rather a group of pixels. With that in mind, a random grid creates a more noisy pattern, where close pixels can relate to different directions. Unlike the grid, the region method achieves 42% targeted fooling rate. This approves for us that combining two constraints is possible and more interesting is that close pixels that pull together are very crucial for fooling the model. In that matter, it validates why the random grid cannot work as the network intercept it as a noise only. At last, our gradient method appeared to has the upper hand but not so upper as we thought. The gradComb perturbation gets only 2% higher targeted fooling rate, but in the leak effect, we see 56% as oppose to 43%.

To farther understand which method is better and why we will look at the confusion matrices. We can see in the CM of the gridComb that most of the samples that were moved out of the source class got into class 6 which is the 'frog' class. It's being told in previous talks that this dataset has a biased into the frog class, and it's pretty clear that the strict TUAPs become a random noise that pushed the samples into the highly biased 'frog' class. When we observe the CM of the regionComb, we can see that the target class 'horse'\'airplane' (7\0) becomes dominant classes which is what we intended to do. But in the CM of gradComb we observe a strange behavior, most of the samples moved into the 'airplane' class while the 'horse' class faded away. Unlike what we struct the attack to do the gradient chose to maximize its capability by pooling to only one class instead of two. In our opinion, the gradient did what best for the optimization problem and in fact, the attack was stronger. This could be overcome by constructing a different optimization problem such that gives more weight to balance the formula. In fact, this optimization problem can be extended very easily to satisfy even more than two constraints. We believe that that region combining method and the gradient optimization method can both lead to multi-targeted universal attacks with further research.

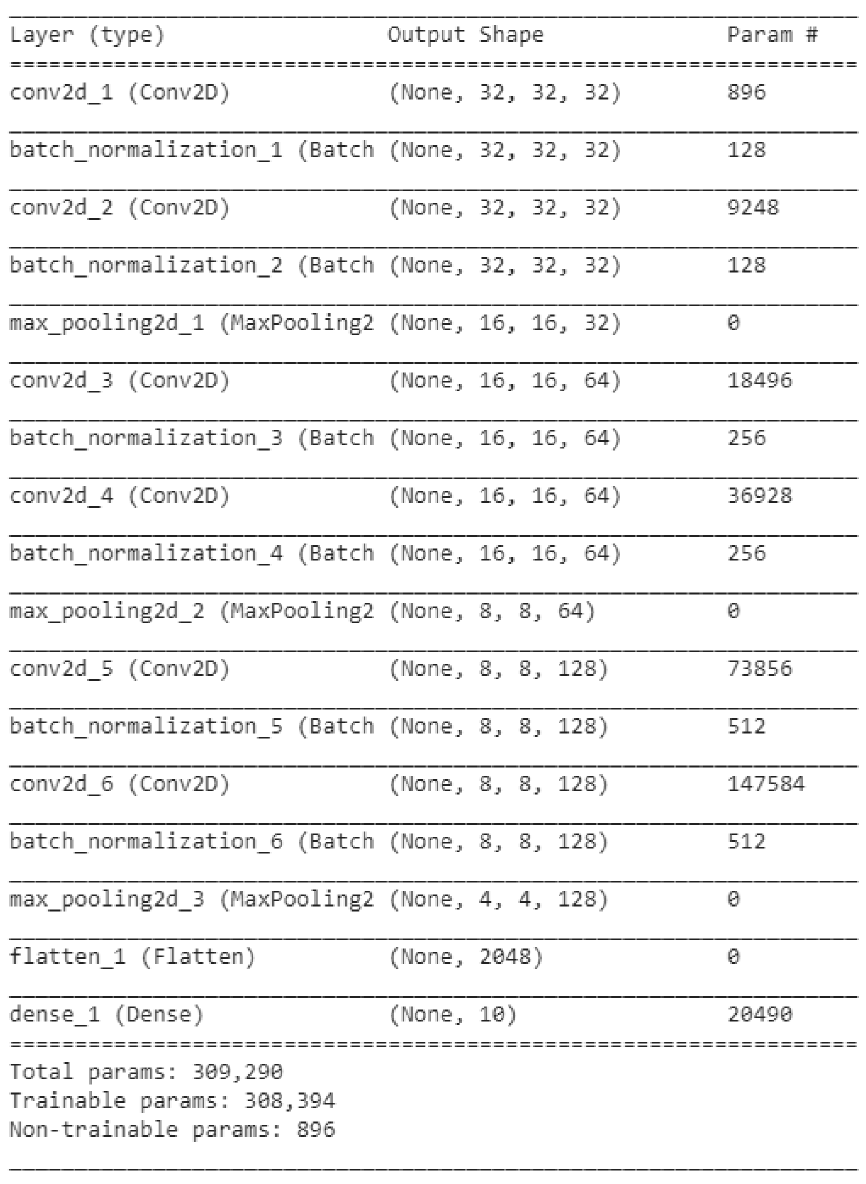
Follow up steps:

In this work, we made the first step towards multi-targeted universal perturbations. We created a targeted attack algorithm and we suggested two promising methods to combine attack constraints. To further extend this work we suggest:

- Evaluation of this work on larger and more realistic dataset like IMEGNET
- Use decision boundary visualization to farther analyze the decision surface
- Test our work on different domains besides computer vision
- Formulate a new optimization problem to handle several source-target constrains with priority\ Weights.

Appendix:

CNN model summary:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896       
_________________________________________________________________
batch_normalization_1 (Batch (None, 32, 32, 32)        128       
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 32, 32, 32)        9248      
_________________________________________________________________
batch_normalization_2 (Batch (None, 32, 32, 32)        128       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 16, 16, 32)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 16, 16, 64)        18496     
_________________________________________________________________
batch_normalization_3 (Batch (None, 16, 16, 64)        256       
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 16, 16, 64)        36928     
_________________________________________________________________
batch_normalization_4 (Batch (None, 16, 16, 64)        256       
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 8, 8, 64)          0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 8, 8, 128)         73856     
_________________________________________________________________
batch_normalization_5 (Batch (None, 8, 8, 128)         512       
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 8, 8, 128)         147584    
_________________________________________________________________
batch_normalization_6 (Batch (None, 8, 8, 128)         512       
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 128)         0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 10)                20490     
=================================================================
Total params: 309,290
Trainable params: 308,394
Non-trainable params: 896

_________________________________________________________________
```

References:

[1] C. Szegedy *et al.*, “Intriguing properties of neural networks,” in *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, 2014.

[2] I. J. Goodfellow, J. Shlens, and C. Szegedy, “Explaining and Harnessing Adversarial Examples,” pp. 1–11, 2014, doi: 10.1136/pgmj.2005.034868.

[3] N. Carlini and D. Wagner, “Towards Evaluating the Robustness of Neural Networks,” *Proc. - IEEE Symp. Secur. Priv.*, pp. 39–57, 2017, doi: 10.1109/SP.2017.49.

[4] A. Kurakin, I. Goodfellow, and S. Bengio, “Adversarial examples in the physical world,” no. c, pp. 1–14, 2016, doi: 10.1109/ICCV.2017.56.

[5] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, “Practical Black-Box Attacks against Machine Learning,” pp. 506–519, 2016, doi: 10.1145/3052973.3053009.

[6] S. M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, “Universal adversarial perturbations,” *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-Janua, no. 5, pp. 86–94, 2017, doi: 10.1109/CVPR.2017.17.

[7] G. Krizhevsky, A., Nair, V., & Hinton, “The cifar-10 dataset.” 2014.