Adversarial examples, adversarial models and deep learning based security

Adrien Chan-Hon-Tong

Adversarial examples, adversarial models and deep learning based security

Adrien CHAN-HON-TONG
ONERA, the french aerospace lab
adrien.chan_hon_tong@onera.fr

March 28, 2018

Abstract

Cyber security system could benefit from the deep learning breakthrough. However, there are, today, critical issues with deep learning. The major issue is the lack of robustness of classical deep networks. This lack of robustness allows to compute well documented adversarial examples: confusing a network with just the addition of a low amplitude perturbation.

Here, I show that this lack of robustness also allows adversarial model: possibility to change the global model behaviour with just the addition of a low amplitude perturbation to the train set. This property could be a real security faults for continuously trained system.

Evaluation of the efficiency of adversarial model is provided on image classification datasets.

1 Introduction

Today, it seems that deep learning (which appears in computer vision with [1] - see[2] for a review) may lead to a major industrial revolution. Applications already go much further than social network [3] but include autonomous driving [4], health care [5], plagiarism detection [6] and security (e.g. [7, 8]).

However, classical deep network still exhibits a strong lack of robustness. The typical example of this lack of robustness is that these networks admit adversarial examples (first introduced in *arxiv.org/abs/1312.6199*, examples in security context include [8, 9], see also [10, 11] adversarial examples in computer vision context). For almost all input, it is possible to design a specific imperceptible (additive) perturbation that will make the network to predict an output opposite to that which he would have produced on the original input.

Currently, on static problem, adversarial examples are not critical because in this case, the goal is more to generalize than to be robust. Off course, both properties are correlated. Yet, generalizing is more a matter of affinity between the algorithm and the targeted data distribution than a property of

the algorithm itself (see [12] for illustration). Generally, deep network aims to capture a distribution which are very sparse. As the adversarial examples do not belong to this targeted distribution, it is not too surprising/problematic that the network may badly handle such outliers.

Now, adversarial example is a critical issue for real life application of deep learning: indeed, if a deep network is deployed in real word, people will interact with it. And so, the network may have to deal with a dynamic distribution instead of a static one. And, this dynamic distribution may include hacking attack which can take the form of adversarial examples. For artificial perception, this is a problem since [13] shows that these adversarial examples can be produced in real physical word.

In numerical context, this is worse as nothing can prevent to design adversarial attack especially since [14] breaks all proposed patches including gradient masking and distillation both for continuous or discrete domain. As typical example [8] highlights this issue of adversarial examples in cyber security context. A module is designed in order to detect malwares from large app repository (like android app). Static features are extracted from the code and dynamic features are extracted from run into sandbox. Then, deep network is trained to recognize normal software and malware. This network offer very high detection performance (reaching state of the art). Yet, on modified malwares, detection performances drop to only 66%. And, both distillation or retraining does not highly increase robustness to adversarial examples.

Now, in this paper, I introduce a (never documented) related issue. I show that some targeted properties can be transferred to the network during a training on a database containing modified examples. I call this issue adversarial models in reference to well documented adversarial examples. The two phenomenons are illustrated in fig 1.

In other word, adversarial examples are perturbations that modify network output during testing. While, I introduce here perturbations performed during training that modify the global network behaviour.

The main purpose of this paper is to evaluate the amplitude of this fault. This is done on image classification datasets: this allows to deal with very classical deep networks, and, thus, providing highly reproducible code (see *achanhon/AdversarialModel* github).

Yet, I also state that this adversarial model property should be considered in context of cyber security. Indeed, this theoretic contribution could at least allow:

- to insert back door in a continuous learning module
- to pervert a continuous learning module
- to falsify the observed quality of an algorithm on a public test set (this last fault is not directly related to cyber security attack but is still somehow a matter of truthfulness and/or security)

In the next section, I present a quick overview of mechanism allowing adversarial examples. Then, I present the possibility to transfer desired property into

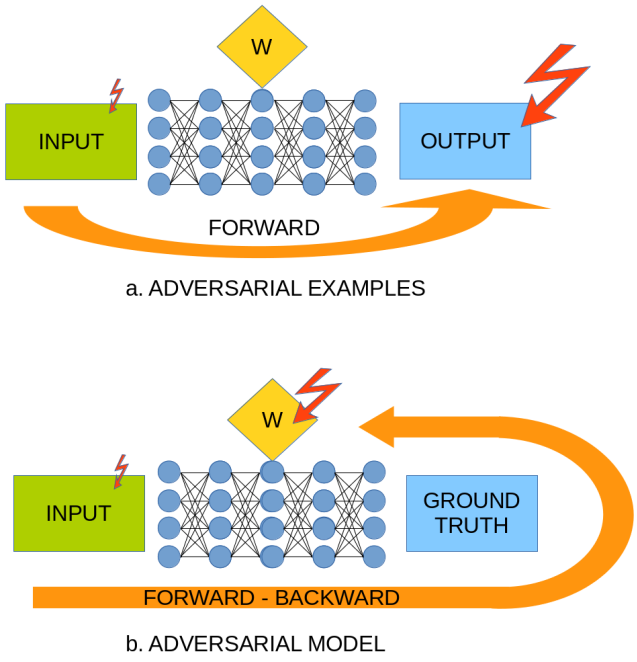


Figure 1: Illustrations of adversarial examples and adversarial model. The light symbolise the perturbation. Adversarial example is a perturbation of the input which leads to a output perturbation. Adversarial model is a perturbation of the training data which leads to a model perturbation.

a model by hacking the training set. Three possible security faults related to this phenomenon are presented. Then, in section 4, I present empirical experiments evaluating the criticality of this adversarial model behaviour, before conclusion and perspective in section 5. These experiments are not directly related to the presented faults, but, rather designed to highlights to adversarial model.

2 Why such robustness fault

A classical deep network is a sequence of simple tensor operations done from an input. As stressed by *arxiv.org/abs/1312.6199*, classical deep networks are Lipschitz function, and so, expected to be smooth.

Let *input* be an input vector in $\mathbb{R}^D$, a simple neural network (here a multi layer perceptron: *MLP*) is defined by an activation function h (let consider *relu* function i.e. $h(u) = \max(0, u)$), and, by a set of matrix $(M_1, M_2, ..., M_K) \in M_{n_1,m_1} \times ... \times M_{n_K,m_K}$ with $m_1 = D$, $n_k = m_{k+1}$ and n_K be the desired output size of the network (typically m_K is the number of classes - the binary classification is a special case with $m_K = 2$ when using a cross entropy loss but $m_K = 1$ with hinge loss). Then, the output of the network on an input is just $M_K\widehat{h}\left(M_{K-1}\widehat{h}\left(...M2\widehat{h}\left(M_1 input\right)\right)\right)$ with $\widehat{h}$ the operator applying h on each cell of the input matrix (i.e. $\forall I, J, \forall Q \in M_{I,J}, \widehat{h}(Q) \in M_{I,J}$ and $\forall i, j\ \widehat{h}(Q)_{i,j} = h(Q_{i,j})$). So, in the case $n_K = 1$, such deep network is just a function from $\mathbb{R}^D$ to $\mathbb{R}$. And, such network is at most $\prod_k^K \lambda_k$ Lipschitz with λ_k be the operator norm of M_k (It is trivial that h is trivially 1 Lipschitz and that composition of Lipschitz function is Lipschitz with coefficient be lower than the product of the sub function coeffs). Let recall that this function is even a continuous piecewise linear function.

Now, the problem is that $\prod_k^K \lambda_k$ could be very large for classical deep networks. Frobenius norm of Alexnet [1] (the first deep network) is 5908006809. The one of the 10 first layers of VGG is 7662531036 (VGG is somehow the second most classical deep network - first introduced in *arxiv.org/abs/1409.1556* and at the root of [15, 16]). Off course, these bound are well not tight. However, for Alexnet, the maximal ratio of each layer has been measured in *arxiv.org/abs/1312.6199* on IMAGENET database [17] (so including the empirical behaviour of *relu*). From these measures, Alexnet can be expected to be a 792793 Lipschitz function. This is much less than 5908006809 but still very large. Such no smoothness is very clear in computer vision but still a reality in cyber security [8].

So, these results highlight that even if a deep network is just a sequence of simple operations, the number of parameters (and the amplitude of these parameters) coupled with the large number of layers lead to potential very unstable output. This is clear for computer vision deep network (like Alexnet and VGG) but is still a reality for any deep network, even in cyber security context [8].

So compare to other machine learning algorithm (e.g. support vector machine *SVM* [18]), deep learning may be more unstable. But, worse, compare to others, deep learning is more derivable, so there is trivial way to exploit this potential for instability. The easy way to generate adversarial examples is to look into the root algorithm to train networks: the back propagation of the gradient[19]. Back propagation allows to compute derivative according to each weight given a loss from the last layer. But, this relies on the computation of derivative according to each *neuron*. Thus, by a nature, back propagation allows to compute derivative according to the input data itself (this can be done easily with PYTORCH *pytorch.org/* just asking the internal variable corresponding to the input to store gradient). If these derivatives are high for a given sample, it means that performing very little changes to on this sample will lead to very different behaviour of the network. And, for classical deep learning network, this is the case for almost all samples.

An other way to look this back propagation fault is that the same process which allows to update the weights in order to process an input that can be used to update the input in order to get a specific behaviour. Typically, one can done gradient descent from a specific input to minimize the probability of the truth class, eventually producing an adversarial example. This is why adversarial examples are mainly a deep learning issue (rather than a machine learning one, even if technically adversarial examples may also exist for other machine learning algorithm like SVM).

As the simple mechanism to produce adversarial example is to use the internal gradient, there was a hope that keeping the gradient private and/or masking the gradient at the first layer may protect against adversarial attack. Typically, it is possible to add some layers at the bottom of a network behaving like the function *round* (i.e. having locally a constant value around each integer from two bound). With these layers the gradient reaching the integer input is 0, forbidding to compute easily adversarial examples.

Yet, [14] shows that using only the pairs of input output, one can tune a copy network to be an approximation of the targeted network. Then, it is possible to use the gradient of the copy network to design adversarial examples for the targeted network. This algorithm currently bypass all known defence. Typically, in face of a network protected by a round function, the copy network will essentially approximate the round function per the identity which is exactly what we need to catch the masked gradient. This brief review of adversarial examples is well documented in the literature.

Now, the question of how properties can be transferred to the model via perturbed training has not been explored.

Let note that training on adversarial has been explored to try to reduce the adversarial examples fault (e.g. [8]) not to create new one. And, beside it is just training, computing adversarial examples and then training again. While, I design specific samples for disturbing the training.

3 Adversarial model

3.1 Theoretic presentation

Let consider a MLP defined by $(M_1, M_2, ..., M_K) \in M_{n_1,m_1} \times ... \times M_{n_K,m_K}$ with $m_1 = D$, $n_k = m_{k+1}$ and $n_K = 1$. Let introduce the notation $F(input) = \widehat{h}\left(M_{K-1}\widehat{h}\left(...M2\widehat{h}\left(M_1\ input\right)\right)\right)$ (I will need this non standard notation after). Let remark that the output of the network on the input $input$ is $MLP(input) = M_K F(input)$ (as presented in previous section).

During the training, the matrix $M_1, M_2, ..., M_K$ are updated incrementally according to the following process. Each $input$ comes with a target value, the weight are updated according to the *proximity* of $MLP(input)$ to this target. This proximity is measured by a derivable loss function. Forward Backward algorithm can be used to compute the partial derivative of the loss according to each cell of each matrix $M_1, M_2, ..., M_K$ (each cell is called *a neuron*). Then, matrix are updated according to these derivatives.

Now, if $input$ is transformed into $input'$ (with the same target value), then instead of updating the weight according to the proximity of $MLP(input)$ and the target, the update will be done according to the proximity of $MLP(input')$ and the target. Yet, as recalled $MLP(input') - MLP(input)$ can be large even when $input' - input$ is small. So, by tuning the perturbation $input' - input$, the weight will be updated very differently.

Let consider the training of the last layer only (i.e. M_K). Let train this layer with SVM framework (this is easier for reasoning as the training output is deterministic). Let $x_1, ..., x_N$ be some vectors in $\mathbb{R}^D$ and $y1, ..., y_N$ the label (-1/+1 to simplify). Given a model w in $\mathbb{R}^Q$, the classical smoothed error of w on the dataset x, y is $e(w, x, y) = w^T w + C\sum_n relu\left(1 - y_n x_n^T w\right)$ (there is usually a bias term b omitted for faster understanding). Let write w^* for the model which minimizes the smoothed error is $w^*(x, y) = \arg\min_w e(w, x, y)$. Let recall that if R is an orthonormal matrix (e.g. a rotation) then $e(Rw, Rx, y) = e(w, x, y)$, and thus, $w^*(Rx, y) = Rw^*(x, y)$.

In our context, w is M_K (with size n_{k-1}), and, $x_{train,n}$ are the output of the network just before the last layer i.e. $x_{train,n} = F(input_n)$. So the training will lead to $w_{train} = w^*((F(input_1), ..., F(input_N)), (target_1, ...target_N))$. Now, let assume to **want** a specific and known model $w_{desired}$. This is crucial here to be able to compute the desired model $w_{desired}$ which traduces the desired behaviour of the system in the weight space.

Let fix a value δ and let consider the auxiliary loss function l_n which measures the squared distance between $F(input)$ and $x_{train,n} + \delta(x_{train,n}^T w_{train}) w_{desired} - \delta(x_{train,n}^T w_{desired}) w_{train}$. A critical point is to see that this network F plus l_n is just a classical regression network with a typical square loss. So, using the forward backward algorithm to compute the derivative of the weight of F relatively to a modification of $F(input)$ is completely straightforward. So, here, the only difference is that I use the forward backward algorithm to compute the deriva-

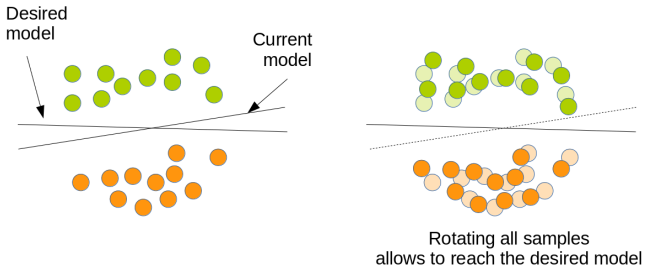


Figure 2: Scheme of adversarial model based on *rotation*.

tive of $input_n$ (this can be done with a small PYTORCH script). So, from these derivatives, I form ∇_{input_n} and compute $x_{hack,n} = F\left(input_n + \lambda_n \nabla_{input_n}\right)$ (for a specific λ_n value).

Let note x_n for $x_{train,n}$, and let assume that derivative are that high on $input_n$ that one can get l_n loss to be 0 with only one gradient step i.e. $x_{hack,n} = F\left(input_n + \lambda_n \nabla_{input_n}\right) = x_n + \delta(x_n^T w_{train}) w_{desired} - \delta(x_n^T w_{desired}) w_{train}$ (of course this is highly not probable, yet it can be accepted in first approximation). Training on these modified vectors, this will lead to $w_{hack} = w^*\left(x_{hack}, y_{train}\right) = w^*\left((I + \delta w_{desired} w_{train}^T - \delta w_{train} w_{desired}^T)x, y\right)$. But, at first order in δ, $(I + \delta w_{desired} w_{train}^T - \delta w_{train} w_{desired}^T)$ is orthonormal. So, I can write the approximation, $w_{hack} = w_{train} + \delta(w_{train}^T w_{train}) w_{desired} - \delta(w_{train}^T w_{desired}) w_{train}$.

So, by hacking the training data, I make the hacked model to rotated toward the desired model. Precisely, one can trivially show that scalar product between w_{hack} and $w_{desired}$ is higher than between w_{train} and $w_{desired}$. Indeed, this scalar product increases even more quickly as w_{train} and $w_{desired}$ have a low scalar product which means that this hacking can be very efficient (this can be see in the equality $w_{hack}^T w_{desired} - w_{train}^T w_{hack} = \delta(w_{train}^T w_{train})(w_{desired}^T w_{desired}) - \delta(w_{train}^T w_{desired})^2$).

Off course, this demonstration mix two approximations: first, it assumes that $F\left(input_n + \lambda_n \nabla_{input_n}\right)$ is exactly $x_{hack,n}$, and then, that, $(I + \delta w_{desired} w_{train}^T - \delta w_{train} w_{desired}^T)$ is orthonormal. Yet, both approximation are somehow acceptable in first order. So, I just give theoretic hint that small perturbation of training data can be computed to produce a specific and desired change in the model behaviour (see fig 2).

Now, could producing such a rotation be interesting to give the network specific behaviour ? I give in the next subsections examples of security faults one can create with this method.

3.2 creating a back door

Let consider a continuous trained security module. The module is put in place, then each time the module predicts a security thread, the input is considered to be added to the training dataset. Then with a specific regularity, the module is

trained on the new database.

Now, let assume one has designed an new attack (or a set of attack). There are two way to increase the *probability* for the attack to bypass such module: the first is to add a perturbation on the target attack (classical adversarial example [8]), the second is to flood the module with modified version of previous attacks in order to make that next training will make the module becoming particularly bad on the desired attack. This second way is less direct, but it will work even in context where designing adversarial versions of the new attacks is impossible or decrease too much the efficiency of these new attacks. In this case, it is much more relevant to create benign modified version of previous attacks to create a back door in the model, than to create benign modified version of the new one.

In other words, the old attacks are a training set, and, the strategy is to pollute the training set to make the resulting model ineffective on a particular new attack or a set of attack (see fig 3).

So this is exactly the previous theoretic presentation with $w_{desired}$ being the opposite of feature vector corresponding to the new attack.

3.3 perverting a module

Let again consider a continuous trained security module. This module can be perverted to incorrectly handle specific attacks (i.e. to create a back door). But, this long run strategy assumes that the specific new attack is already designed (it can be multiple new attacks).

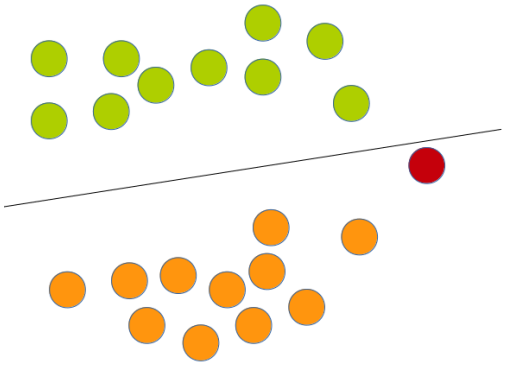
An other attack will be to pervert this module in anticipation. Of course, making the module becoming inefficient may be somehow detectable by the developers of the module, while the previous back door is much more difficult to detect. Yet, here, the interest is to have no need for a target set. And, making the module inefficient could by a way to discard the confidence into the module and not only to hope to make a new attack bypass it.

So, in other word, the strategy is to flood the module with modified version of previous attacks to make the module becoming globally inefficient after the next training. The fault corresponds to polluting the training dataset in order to decrease global classification performance. One way is to use $w_{desired}$ to be a random vector (or the vector produced when learning on the same data with random labelling).

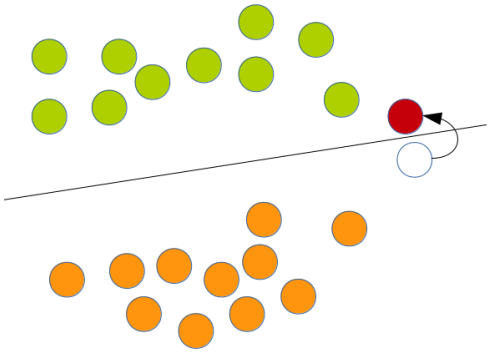
3.4 evaluation policy

The last fault that may be created with this paper contribution is not directly related to cyber security but to the confidence given to deep learning based module.

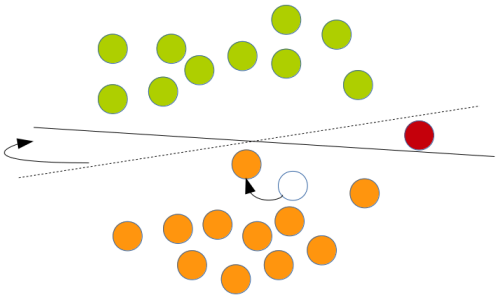
It is well known that evaluating an algorithm is truthful only when performed on private data [20]. Indeed, machine learning results (especially deep ones) can be distorted even by unconscious and common bad practices like to tune few parameters on the test set, or more generally, to use little feed back from test



a. Initial situation, training is performed on green/orange samples. The goal of the hacker is to make the red sample to be classified as green.



b. Adversarial example attack: a way is to perturb the red sample to make it to go across the boundary



c. Adversarial model attack: the other way is to flood the module (which is trained again with a regular frequency) with modified samples, in order to make the boundary to go across the red sample

Figure 3: Illustrations of adversarial examples and adversarial model.

evaluations [20]. So, one can imagine how it could be distorted by voluntary misconduct. This is why the strongest academic results in deep learning are those evaluated on guidance benchmark. In such competition like IMAGENET [17] or MSCOCO, a leader team publishes only training data and provides a strict evaluation process (see *www.image-net.org/challenges/LSVRC/announcement-June-2-2015*). This way, participants can not tune too much the algorithms on the test set, allowing a quite fair evaluation of the algorithms.

In other words, the safe way to evaluate a deep learning system should follow these steps. First the system should be completely designed. And, then, new data should be collected for the evaluation. Finally, the system is tested on these new data. Now, let consider a company A who wants to buy such system to the company B. A would collect only few samples just to illustrate the problem. Then, B would design the system and collect very large set of samples to train the system. Then, A would collect test samples (let say less than 5% of what is needed to train) and evaluate the system produced by B on the test samples. On the paper, it is relevant from business point of view and it allows a quite safe evaluations. However, the problem is that legislation is not clear about how such scheme would end up if the system produced by B is not sufficient on the test set from A. This, problem is even more critic if A is a public organization required to make public tender to buy anything.

This juridical question is not new but worth to be recall: private evaluation is safe but raises economic issues. So, in some context, even if it is known to be theoretically unsafe, there will be case where algorithm will be evaluated on public dataset. Yet, one could though that these theoretic fault can be detected by an expert reviewer. Indeed simple falsifications are very simple to detect. Learning directly on testing data can be detected by trying to reproduce the train/test. Adding testing samples on training dataset can be detected by looking at the training dataset. Changing the testing data can be detected by comparing with a copy of the testing data (which are public here).

Now, unfortunately, with this paper contribution, one can produce an undetectable falsification by changing training data. Reviewing the train/test process can not detect anything as this process is fair: falsifications have been done before by adding undetectable perturbation within the training data. Even if it is not directly related to cyber security, this last fault highlights importance to remember that public evaluation is untruthful, and, should not be considered for critical deep learning module (diagnosis, autonomous driving and/or security).

Formally, in this case $w_{desired} = w_{test}$ i.e. the result of the training on the public test data. Such falsification is not possible on private data as $w_{desired}$ can not be computed. But, inversely on public data, one can compute $w_{desired}$, and so, use my adversarial model hack.

4 Experiments

To this point, I have presented two contributions of this paper:

	adversarial examples	adversarial model
pros	simple way to bypass a defence	work without modification of the target
cons	perturbation should not decrease attack efficiency + assume that target is coded by the hacker	need to have access to training (e.g. flooding a continuously trained system)

Table 1: Adversarial examples vs model from hacker point of view
Adversarial examples is a more direct hack. Yet, adversarial model allows a larger set of insidious strategies. For example, flooding a continuously trained module can make the module to block the last fashionable app (without any control on this app), and thus, making users to disconnect the defence module.

- I have presented a theoretic presentation of how it is possible to design a specific noise in order to bias a training in a specific desired way
- I have presented different scenarios where such property could be used in cyber security context (some where simple adversarial examples could not see table 1)

In this section, I present experiments on the amplitude of this adversarial model phenomenon. Indeed, the possibility of such adversarial model is now clear but the amplitude is in question: if the phenomenon is as perturbing that adversarial examples than this is a real issue, otherwise, this is not.

For this experiment, I have chosen to rely on experiments in computer vision context. Thus, the experiment are not directly related to the cyber security faults. Yet, computer vision is one of the main area of deep learning development. So, presenting computer vision experiment allows to use the material developed in this context.

4.1 Experimental setting

In order to provide reproducible artefact, I select a 0 parameter convex pipeline deep pipeline. The pipeline inspired from [21] is composed of a convolutional neural network (*CNN*) trained on IMAGENET [17] used as feature extraction plus a SVM. This way the pipeline match the theoretic part. The CNN used is some of the first layers of a VGG. IMAGENET weights of the network are available (for example here: *github.com/jcjohnson/pytorch-vgg*). The SVM implementation is LIBLINEAR [22] with bias and default parameter.

Features are extracted from images by the network producing one vector per image. Weights are IMAGENET ones and are never updated in all experiments (even if gradient is eventually computed to get derivative relatively to the input). So this feature extraction has no parameter, and is, thus, completely reproducible (only possible changes should be due to hardware setting, PYTORCH version and/or NVIDIA driver). Then, SVM is learnt on training

vectors and used on testing vectors. As SVM learning is a convex problem, multiple runs should converge to the same solution. And, there is also 0 parameter as only default parameters are used. Typically $C = 1$ as default parameter of LIBLINEAR.

4.2 Datasets

For these experiment, I consider CIFAR10 and CIFAR100 datasets, two very classical image datasets [23] (first introduce in 2009 in a technical report *learning-features-2009-TR.pdf* available from *www.cs.toronto.edu/ kriz/*). Both datasets can be downloaded (see the page *cifar.html* from the same website).

4.3 Accuracy gap

Here, I consider CIFAR10 and CIFAR100. First, vectors are extracted using the CNN from original images (both from train and test set). On this honest vector SVM reaches 75% of accuracy on CIFAR10 and 53% of accuracy on CIFAR100 (this performances are quite normal for a CNN learnt on IMAGENET and not on CIFAR).

Then, $w_{desired}$ is computed as the SVM learnt directly on the test. Here, this is a multiclass classification problem, not just a binary one. Then, I consider the auxiliary network consisting of the CNN + a last layer corresponding to $w_{desired}$. Then, for each training image, I perform the forward backward with this auxiliary network to collect the derivative corresponding to the image pixel. Then, I update each pixel value (ranging from 0 to 255) with the sign of the corresponding derivative. This way, I create modified training realistic image (still integer images) with each pixel having a distance at most 1 from the original value. This is not completely the same transformation than in the theoretic argument. Here, the transformation of x_n into x'_n aims to verify $w^T_{desired}x'_n = w^T_{desired}x_n + \lambda_n y_n$ (pseudo code is summarized in 2). This is not just a rotation which is not possible in multi class setting, not possible with integer images and assumes that all samples will follow the same transformation.

Learning with the original pipeline on this derived training dataset leads to 82% of accuracy on CIFAR10 and 66% of accuracy on CIFAR100!

Worse, repeating 3 times the kind of gradient descent on the pixel values, the performances goes to 92% of accuracy on CIFAR10 and 78% of accuracy on CIFAR100!

This accuracy gap alone shows that adversarial model is as critical as adversarial examples: by allowing pixel values to be modified by at most 3, performances of the pipeline are modified by more than 24%. Let remark that 78% of accuracy on CIFAR100 is a never reached accuracy. But, here, with a small hack, the completely not adapted pipeline seems to jump to this level of performances.

Code of this experiment (CIFAR100) is available (for research purpose only) on the github *github.com/achanhon/AdversarialModel*. Code of the other ex-

periment can be requested but are currently not on the github to make easy the use of the main code (all codes are close to the one provided anyway).

This implementation of adversarial model is relevant from functional point of view as modifying only some sample allows to bias the resulting model while the rotation based implementation assumes all the training dataset is perturbed simultaneously. Also, the relevancy of this implementation from performance point of view is strongly established by experimental results. Unfortunately, I do not manage to demonstrate formally that this implementation behaves as observed. Currently, one can prove easily that if each x_n is transformed into $x'_n = x_n + \lambda_n y_n w_{desired} + \mu_n$ with $\mu_n^T w_{desired}$ then as $relu$ is an increasing function, it holds that $e\left(w_{desired}, x', y\right) \leq e\left(w_{desired}, x, y\right)$ (see definition of e in the section 3.1) because $relu\left(1 - y_n x_n^T w_{desired} - \lambda_n y_n^2 w_{desired}^T w_{desired}\right) \leq relu\left(1 - y_n x_n^T w_{desired}\right)$. Thus, the energy of the desired model is lower with the new dataset than with the original. Yet, this does not prove that resulting model after training will be closer to the desired one. The interesting point in the rotation based implementation is that rotation lets the norm of the model invariant. Here, it seems this is not true. So, the perturbation breaks the previous tradeoff between regularity $w^T w$ and data attachment sum $\sum_n relu\left(1 - y_n x_n^T w\right)$. This introduces difficulties in proving properties on the distance of the global minimal after perturbation, and the desired model.

4.4 Adversarial model vs training on adversarial examples

Let consider again the objective to make a deep learning module globally inefficient. The use case is that one want to bypass a security module, the (long run) strategy is to flood the module with modified samples, so next time module is retrained on last samples it will become inefficient.

Such strategy can not trivially be implemented with classical adversarial examples. Off course, adding strong outliers will indeed make the training harder, and may be, break the resulting performance, but, adding lowly modified adversarial example can even increased the performance because it will force the classifier to be robust [8] (see fig 4) And adding inverse adversarial example (changing the samples to make then classified even more easily by the algorithm) may have no effect at all (it makes easier to reach the same point).

Inversely, optimizing the perturbation of the training samples with the goal of breaking a security module is exactly a use case of adversarial model. A simple way to implement this goal is to use the adversarial model algorithm of previous subsection with $w_{desired}$ resulting from a training with random labelling. So the idea is to make the current model rotates toward a random one.

I evaluated both strategies on CIFAR100. Training is modified without any knowledge from the test set either by classical adding adversarial or by biasing the training toward a random model. Then, to evaluated the degradation, accuracy is measured on test set.

Results are not available yet.

```
adversarialModel(CNN,wDesired,X,Y)
   auxiliaryCNN = CNN::wDesired
   X' = []
   for x,y in X,Y:
      gradient = forwardBackward(auxiliaryCNN,x,y)
      x' = x
      for i in x:
         x'[i] += sign(gradient[i])
      X'.append(x')
   return X'
```

Table 2: Pseudo code of the selected adversarial model implementation

Applying a fair training process on the dataset `X',Y` leads to a model biased toward `wDesired`. This function can be applied multiple times leading to a more and more distant dataset but biasing more and more the resulting model. As an example, a fair training could be implemented as:

```
fairTraining(CNN,X,Y,lr)
   w = randomVect()
   pipeline = CNN::w
   for epoch in nbEpoch:
      shuffle(X,Y)
      for x,y in X,Y:
         gradient = forwardBackward(pipeline,x,y)
         for i in w:
            w[i] -= lr*gradient[i]
   return w
```

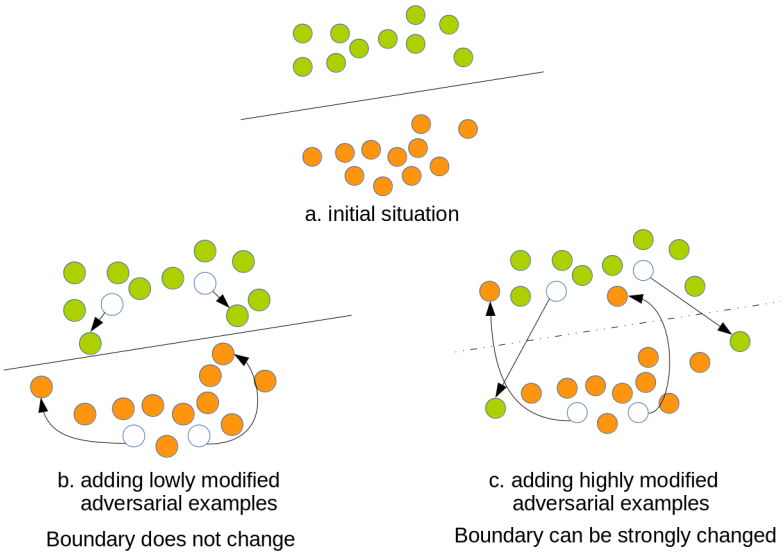


Figure 4: Polluting a system by adding adversarial examples into the training has an ambiguous effect.

5 Conclusion

I offer in this paper a theoretic contribution applied to cyber security. The theoretic contribution is to show that it is possible to design specific perturbation of training examples which will make the standard/fair training to produced a desired model (i.e. system behaviour).

Experimental validation of this theoretic contribution is provided on image dataset with classical deep networks. Code allowing to reproduce the main experiment is disclosed. These experiments show that adversarial model as perturbing than adversarial examples.

I state that this phenomenon could allows to hack a continuously trained security module (e.g. malware detection module). The interest of this long run hacking strategy is to work even when directly creating efficient adversarial attacks is not possible.

Thus, this new issue for deep learning base cyber security module should be considered just behind adversarial example issue.

References

[1] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. (2012) 1097–1105

[2] LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553) (2015) 436–444

[3] Taigman, Y., Yang, M., Ranzato, M., Wolf, L.: Deepface: Closing the gap to human-level performance in face verification. In: Proceedings of the IEEE conference on computer vision and pattern recognition. (2014) 1701–1708

[4] Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., Schiele, B.: The cityscapes dataset for semantic urban scene understanding. In: Conference on Computer Vision and Pattern Recognition. (2016)

[5] Greenspan, H., van Ginneken, B., Summers, R.M.: Guest editorial deep learning in medical imaging: Overview and future promise of an exciting new technique. IEEE Transactions on Medical Imaging **35**(5) (2016) 1153–1159

[6] Alsulami, B., Dauber, E., Harang, R., Mancoridis, S., Greenstadt, R.: Source code authorship attribution using long short-term memory based networks. In: European Symposium on Research in Computer Security, Springer (2017) 65–82

[7] Javaid, A., Niyaz, Q., Sun, W., Alam, M.: A deep learning approach for network intrusion detection system. In: Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS), ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering) (2016) 21–26

[8] Grosse, K., Papernot, N., Manoharan, P., Backes, M., McDaniel, P.: Adversarial examples for malware detection. In: European Symposium on Research in Computer Security, Springer (2017) 62–79

[9] Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z.B., Swami, A.: The limitations of deep learning in adversarial settings. In: Security and Privacy (EuroS&P), 2016 IEEE European Symposium on, IEEE (2016) 372–387

[10] Moosavi Dezfooli, S.M., Fawzi, A., Fawzi, O., Frossard, P.: Universal adversarial perturbations. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR). (July 2017)

[11] Xie, C., Wang, J., Zhang, Z., Zhou, Y., Xie, L., Yuille, A.: Adversarial examples for semantic segmentation and object detection. In: The IEEE International Conference on Computer Vision (ICCV). (Oct 2017)

[12] Zhang, C., Bengio, S., Hardt, M., Recht, B., Vinyals, O.: Understanding deep learning requires rethinking generalization. In: International Conference on Learning Representations (ICLR). (2017)

[13] Kurakin, A., Goodfellow, I.J., Bengio, S.: Adversarial examples in the physical world. In: International Conference on Learning Representations (ICLR). (2017)

[14] Narodytska, N., Kasiviswanathan, S.: Simple black-box adversarial attacks on deep neural networks. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). (July 2017) 1310–1318

[15] Ronneberger, O., Fischer, P., Brox, T. In: U-Net: Convolutional Networks for Biomedical Image Segmentation. Springer International Publishing (2015)

[16] Badrinarayanan, V., Kendall, A., Cipolla, R.: Segnet: A deep convolutional encoder-decoder architecture for image segmentation. IEEE transactions on pattern analysis and machine intelligence **39**(12) (2017) 2481–2495

[17] Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, IEEE (2009) 248–255

[18] Vapnik, V.N., Vapnik, V.: Statistical learning theory. Volume 1. Wiley New York (1998)

[19] LeCun, Y., Boser, B.E., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W.E., Jackel, L.D.: Handwritten digit recognition with a back-propagation network. In: Advances in neural information processing systems. (1990) 396–404

[20] Dwork, C., Feldman, V., Hardt, M., Pitassi, T., Reingold, O., Roth, A.: The reusable holdout: Preserving validity in adaptive data analysis. Science **349**(6248) (2015) 636–638

[21] Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR). (June 2014)

[22] Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: LIBLINEAR: A library for large linear classification. Journal of Machine Learning Research **9** (2008) 1871–1874

[23] Krizhevsky, A., Hinton, G.E.: Using very deep autoencoders for content-based image retrieval. In: ESANN. (2011)