DL autonomous lab: Defending and attacking convolutional image classifiers with generative adversarial training

Johannes Heidecke

November 6th 2017

1 Introduction

This autonomous lab is aimed at exploring different aspects of feed-forward networks (FFN) and convolutional neural networks (CNN). In particular, the vulnerability of CNNs exposed to *adversarial attacks* has been investigated with different experiment settings.

In the 2013 paper "Intriguing properties of neural networks" [4], it was shown that deep neural networks with high test accuracy could easily be fooled to incorrectly classify input data that had been 'attacked' with small perturbations. These perturbations are usually not perceptible for the human eye, but completely change the behaviour of the neural network. Adversarial attack perturbations can be generated by building a model which tries to maximize the targeted network's prediction error. Interestingly, attacks generated for one particular network often also work on other networks - the attacks are transferable.

Several methods have been proposed to both generate and defend against adversarial attacks, among them the fast gradient sign method (FGSM) [2] and projected gradient descent (PGD) [3]. For this lab, it was decided to use a generative adversarial network (GAN) [1], both to find adversarial attacks and to train the CNN to defend against them. As in a typical GAN setting, we have two networks 'pitted' against each other: a classifier CNN which classifies images and an attacker CNN which generates perturbation for an image it is fed. The specific architecture and functioning will be explained in a later section.

In section 2, the performed experiments are described in detail, including the overall set-up, details about the models used, the data set and the evaluation metric. In section 3 we discuss the results and give a conclusive outlook of potential future experiments.

2 Methods

2.1 Experiment set-up

The goal of the performed experiments is to evaluate how well the attacker network **A** is able to create adversarial examples that minimize the classification performance of the classifier **C1**. We perform two different experiment modes. In the first mode, **C1** is trained first and then **A** is trained on the loss of **C1** when classifying attacked input without adapting **C1** to the attacks (consecutive training). In the second experiment mode, **C1** is trained for a few epochs first and then **A** and **C1** are trained alternating with attacked input for one training batch each (alternating training). **C1** can adapt its weights to the attacks.

In order to compare the transferability of **A**'s attacks when used on another network we train two classifier networks, **C1** and **C2**, with equal architecture but different random weight initialization. **A** is always only trained on the loss of **C1**. **C2** is never provided with any attacked training data. In this sense, **A** and **C2** do not have any information about each other and we can investigate how well the attacks of **A** transfer to reduce the performance of **C2**. The two classifiers are always trained for the same number of epochs to ensure fair training conditions and comparability.

The experiment was implemented in Python, using the PyTorch framework. The experiment code can be found in `code/adversarialTraining.py`.

2.1.1 Consecutive training

This experiment mode aims to evaluate if **A** can successfully generate attacks for a classifier which has finished training. There are two phases of this experiment mode:

- **phase 1, classifier training:** **C1** and **C2** are trained for 20 epochs each on real (unattacked) training data.
- **phase 2, attacker training:** The weights of **C1** are frozen and cannot change anymore. **A** is trained for 20 epochs, minimizing the negative cross-entropy between real labels and outputs of **C1** when it is fed attacked input data.

2.1.2 Alternating training

This experiment mode aims to evaluate if **C1** is able to adapt its own weights in a way that make it more robust against the attacks generated by **A**. There are two phases of this experiment mode:

- **phase 1, classifier training:** **C1** and **C2** are trained for 5 epochs each on real (unattacked) training data.

- **phase 2, adversarial training: C1** and **A** are trained alternatingly in an adversarial setting for 20 epochs: they respectively try to minimize/maximize the cross-entropy between real labels and outputs of **C1** when it is fed attacked input data. While training **A**, the weights of **C1** are frozen and vice versa. At the same time, **C2** is trained on unattacked training data for 20 more epochs to ensure comparability.

2.2 Data set

The experiment was performed on the **MNIST** data set. 60,000 examples were used for training and 10,000 for obtaining generalization accuracy. A description of this data set can be found here: `http://yann.lecun.com/exdb/mnist/`

2.3 Models

2.3.1 Classifier model C

The classifier models **C1** and **C2** are basic convolutional neural networks with two convolutional layers and one fully connected layer.

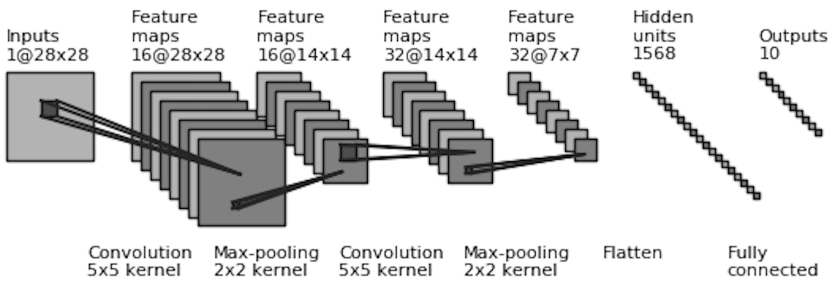


Figure 1: Architecture of classifier models **C1** and **C2**. There are two convolutional layers with batch normalization, kernels of size 5x5 and ReLU activation functions, followed by max-pooling layers with kernel size 2x2. Generated using `https://github.com/gwding/draw_convnet`

The output of the classifier is fed to a softmax layer. This architecture is relatively small but proved to be efficient at classifying MNIST data.

2.3.2 Attacker model A

The attacker model is based on the paper "pixel-level domain transfer" [5] which takes some image and transforms it into a new image of the same size (in this case the perturbation which will be added to the input in order to create an attack). This architecture is too large to be visualized with the tool that was used for the classifier, so here is a description of the layers as a list:

- **Input:** `(28x28x1)` MNIST data
- **Conv1:** convolution with kernel size 5x5 and stride 2. LeakyReLU activation function. `(14x14x64)` feature map.
- **Conv2:** convolution with kernel size 5x5 and stride 2. Batch normalization. LeakyReLU activation function. `(7x7x128)` feature map.
- **Conv3:** convolution with kernel size 5x5 and stride 2. Batch normalization. LeakyReLU activation function. `(4x4x256)` feature map.
- **Conv4:** convolution with kernel size 5x5 and stride 2. Batch normalization. LeakyReLU activation function. `(2x2x512)` feature map.
- **FractStridedConv1:** fractionally strided convolution with kernel size 5x5, stride 2 and output padding of 1. Batch normalization. ReLU activation function. `(4x4x256)` feature map.
- **FractStridedConv2:** fractionally strided convolution with kernel size 5x5 and stride 2. Batch normalization. ReLU activation function. `(7x7x128)` feature map.
- **FractStridedConv3:** fractionally strided convolution with kernel size 5x5, stride 2 and output padding of 1. Batch normalization. ReLU activation function. `(14x14x64)` feature map.
- **FractStridedConv4:** fractionally strided convolution with kernel size 5x5, stride 2 and output padding of 1. Batch normalization. Tanh activation function. `(28x28x256)` feature map.

The output of the network is either clamped to lie in a range of $[-t, t]$ with t being the maximum intensity of the perturbation, or the output of the tanh activation in the last layer is simply multiplied by t.

2.4 Evaluation

The models are evaluated based on their accuracy of classifying the 10,000 examples of the test set. There are two different metrics to observe: attacked and unattacked accuracy. The former is the accuracy obtained when the test set is first perturbed by the attacker, the latter is the accuracy on the unchanged test set. To calculate the attacked accuracy, **A** is fed each example of the test set, generates a perturbation for each, and these perturbations are added to the original data.

We compare the results for different choices of intensity t: $0.01, 0.02$, 0.04, 0.08, 0.16, 0.32, and 0.64.

3 Results

3.1 Perturbations and attacks

First of all, let's have a closer look at the perturbations and attacks that are created with **A**. The following figure 2 shows some examples for attacks that have been created for two random digits of the MNIST data set. In the first column counting from the left, **A** is only allowed to create perturbations with a maximum intensity of 0.01 (the image values lie between 0 and 1). In this case, there is almost no visible change between the original and the attack. The perturbation becomes slightly visible at $t = 0.04$ and clearly visible at $t = 0.16$. Interestingly, the perturbations created for low intensities look structurally very diferent than perturbations for high intensities. This indicates that **A** changes its 'strategy' depending on how much it is allowed to manipulate the original image. It is also notable that the generated perturbation seems to depend on the original that was fed to **A**. We can see in the figure that the perturbations for the two different digits differ along all intensities.

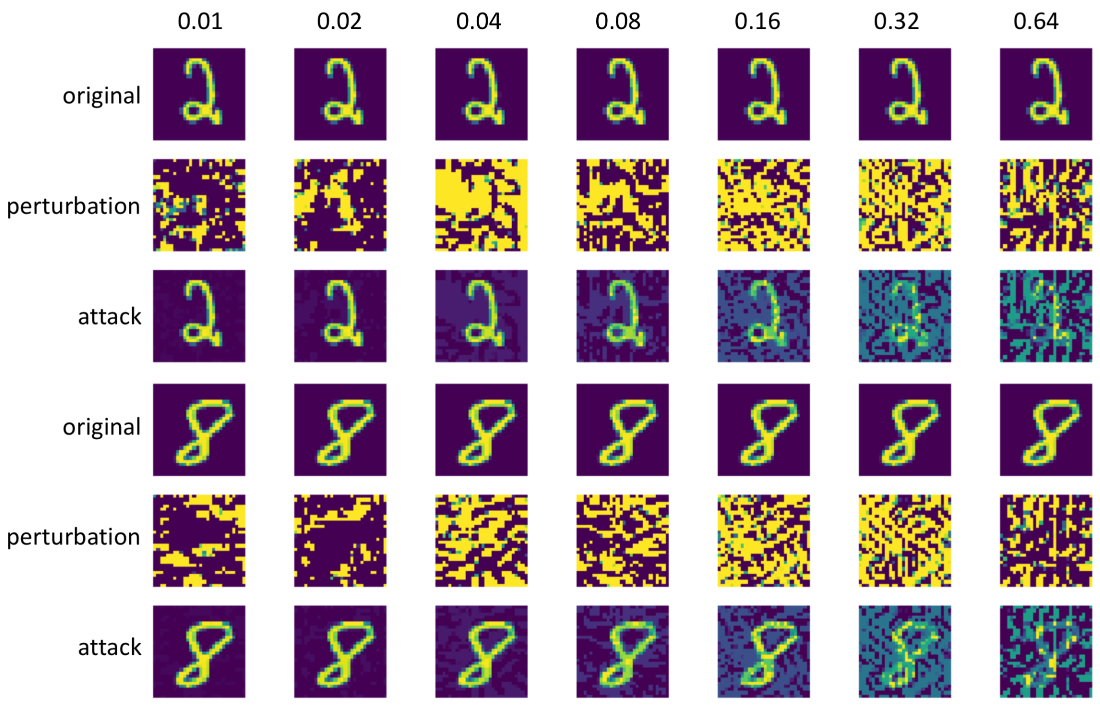
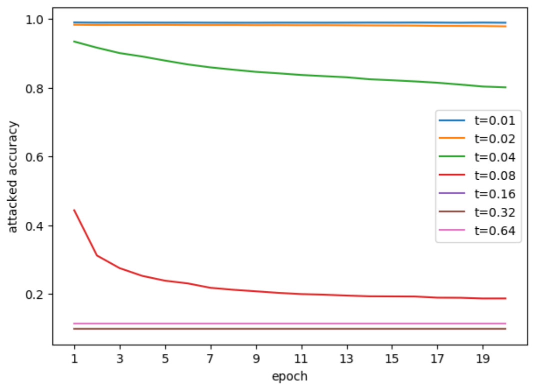


Figure 2: Generated perturbations and attacks for two different, randomly chosen digits. The first and fourth row show the original images. Second and fifth row show the perturbations generated by **A** (values normalized to lie between 0 and 1 for better visibility). Third and sixth row depict the generated attacks. The columns vary in intensity, the first column corresponds $t = 0.01$, the second $t = 0.02$, then 0.04, 0.08, 0.16, 0.32, and 0.64.

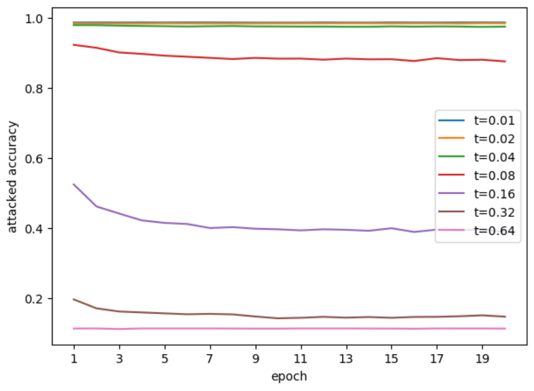
3.2 Consecutive training mode

In the first of the two experiment modes, the classifiers **C1** and **C2** are trained first, followed by **A** being trained using the loss of **C1**. In the following figure 3 we can see how the attacker improved during its training time. First of all, we can see in subfigure 3a that attackers with a very low maximal intensity t cannot successfully decrease **C1**'s accuracy. For $t = 0.01, 0.02$ we could not observe any effect of the attacks on the classifier. The first notable decrease of attacked accuracy is observable for an intensity of $t = 0.04$. After training **A** for 20 epochs, **C1** can only achieve an attacked accuracy of about 80% - the error rate increases by a factor of almost 20. As one would expect, **A** performs better the more intense perturbations it is allowed to generate. For $t = 0.08$, **A** manages to decrease the accuracy significantly to about 20% after 20 epochs. For the higher intensities, **A**'s performance degrades to being hardly better than guessing randomly.

Using the results depicted in subfigure 3b we can evaluate if the perturbations of **A** are also able to decrease the performance of a classifier it has not targeted directly during training. We can see here that the attacker with intensity $t = 0.04$ does not seem to be sufficient to create transferable attacks. Instead of decreasing accuracy to 80% as for the targeted network **C1**, on the untargeted network the accuracy stays almost unchanged. This is different for higher intensities, with $t = 0.08$ the untargeted attacked accuracy decreases to about 90%. In general we can say that attacks with a sufficient intensity do work on similar but untargeted networks, but their effectiveness is lower than on the targeted networks.



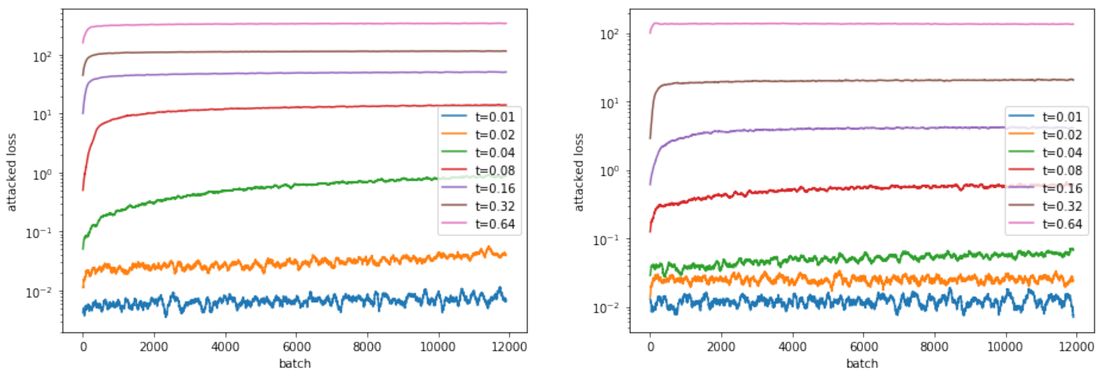
(a) targeted network **C1**



(b) untargeted network **C2**

Figure 3: Attacked accuracy for different levels of intensity t

Figure 4 shows the attacked loss of both **C1** and **C2** over the 20 epochs of training the attacker. Note that the loss is depicted on a log-scale here. Since **A** is being trained to directly maximize the loss of **C1**, it conforms with expectations to observe that the loss of the targeted network increases substantially more than for the untargeted network. Most of the training's success seems to happen in a few epochs at the beginning. In the last 75% of training time there is no strong improvement of the attacker.

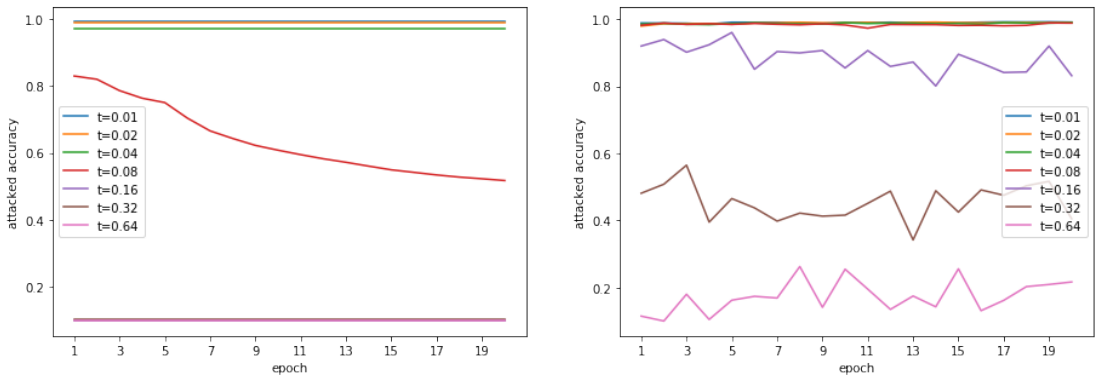


(a) targeted network **C1** (b) untargeted network **C2**

Figure 4: Attacked loss for different levels of intensity t

3.3 Alternating training mode

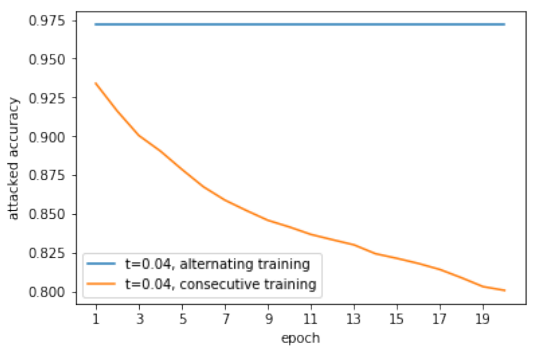
In figure 5a we can see how results change when not training the classifier and attacker consecutively but instead training them in an alternating way. The first important observation is that the first significant decrease of attacked accuracy occurs for a intensity of $t = 0.08$, not already with $t = 0.04$ as before. If the maximally allowed intensity is low enough, alternating training seems to be sufficient to prevent the attacks from becoming succesful. However, if the intensity is too large (0.08 and higher) the attacker is able to to degrade **C1**'s attacked accuracy.



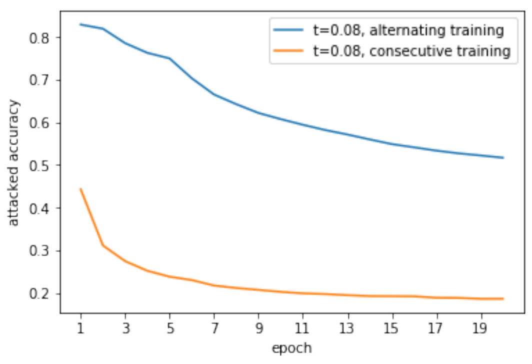
(a) targeted network **C1** (b) untargeted network **C2**

Figure 5: Attacked accuracy for different levels of intensity t with alternating training

In figure 6 we can direclty compare the two different experiment modes. The blue line shows the attacked accuracy of **C1** in alternating training mode where the classifier is allowed to adapt to **A**'s attacks. The orange line shows the attacked accuracy in consecutive training. In subfigure 6a we can see that for an intensity of $t = 0.04$, alternating training prevents the attacked accuracy from decreasing. For an intensity of $t = 0.08$ in subfigure 6b the attacked accuracy decreases in both training modes, but significantly more in consecutive training.



(a) intensity $t = 0.04$



(b) intensity $t = 0.08$

Figure 6: Comparing attacked accuracy of **C1** with consecutive and alternating training

3.4 Conclusion and Outlook

The performed experiments have both shown that CNNs as image classifier can be vulnerable to both targeted and untargeted attacks using small perturbations of the input data. It seems to be possible to alleviate the effects of these attacks when using alternating adversarial training.

One direct improvement to the proposed mechanism would be to train the attacker on the average loss of several classifiers in order to generate more generalized attacks that are not too specifically tuned to specific weights and architecture properties of a single model. It should also be tested if training **C1** alternatingly with the attacker also makes it more robust to other attackers which are trained with other classifiers. In fact the experiment can be extended in many directions, e.g. testing the transferability of attacks to models with very different size and architecture.

References

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[2] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[3] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. 2017.

[4] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013.

[5] D. Yoo, N. Kim, S. Park, A. Paek, and S. Kweon. Pixel-level domain transfer. In *European Conference on Computer Vision*, pages 517–532. 2016.