Final Project- Convolutional Neural Networks

TAU University

Eldad Peretz, i.d. 323820225

Towards Deep Learning Models Resistant to Adversarial Attacks

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, Adrian Vladu

https://arxiv.org/pdf/1706.06083.pdf

Content

Mail - eldadperetz@mail.tau.ac.il.

GitHub Repository - https://github.com/eldadp100/cnn_course_final

Theoretical

Introduction and Discussion

Adversarial attack for a classification model M is a procedure that given an input x produces a new input x_adv (the adversarial example) such that M classifies x_adv incorrectly. The definition can be extended to other ML tasks as regression easily. The restriction on x_adv is $x\_adv \in B(x, \epsilon)$ for $\epsilon > 0$. In computer vision, for small ϵ it produces almost indistinguishable difference (example below).

This definition is suitable when there is a defined norm on the input, in computer vision and signals processing (speech) there are possible norms. It is less intuitive in NLP. The authors used $l_{inf} - norm$ (also called uniform). The paper shows a method to train models that are resistant to adversarial attacks (in some manner that we discuss next), such as models are called robust models.

Breakthroughs in computer vision, speech recognition, natural language processing and deep learning in general, lead trained machine learning models to the center of security critical systems (see "motivation examples"). Therefore, resistant to adversarial attacks should be an important measurement of machine learning models. The following figure is an example to adversarial attack:

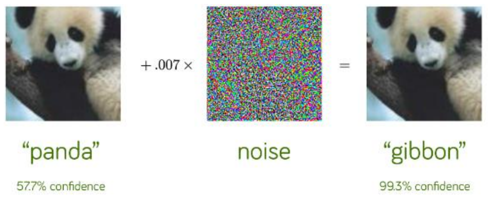


Figure: In the figure we see that by adding a bit of noise to a panda image we get a new image that still looks like a panda but classified as a gibbon in very high confidence. The right-side image is the adversarial example to the left-side. The addition of small amount of noise, that we as humans can't distinguish between the two images, causes the model to incorrect classification.

A known variant of adversarial attacks allows to cause an incorrect classification with restrictions on the output predicted label. For example, specify we want an adversarial example that classified as a specific person. This called targeted attack.

<u>motivation examples:</u>

- In autonomous cars adversarial examples can cause incorrect traffic signs classification and therefore traffic violations. In [1] the authors practically created adversarial example to a real stop sign by placing inconspicuous spray paint or a sticker on the sign. It causes the stop sign to be classified incorrectly. It shows that adversarial examples can be applied in the physical world (without digital manipulations).
- In face recognition authentication systems, we can break into the system by fooling it with an adversarial example that classified as a person with access to the system. It becomes extremely critical in government and military organizations. The same applies to voice / fingerprint authentication.
- In speech to text we can use adversarial attacks to cause errors in the translation. Deaf people might use this technology to understand speech so defending it from adversarial attacks is highly important. Same applies to blind people with visual assistants (that are used in online shops. This allows the seller to cheat the users by uploading an item image that classified incorrectly – an adversarial example).

Another motivation to protect from adversarial attacks is to increase generalization of the model. If the model is vulnerable to adversarial attacks it implies the model does not learn the underlying concepts as supposed to. Therefore, resistance to adversarial attacks is a critical property in order to build highly generalized ML models. Surprisingly, the method used in the paper (adversarial training) does the opposite (hurt generalization) [2].

<u>Attacks settings:</u> To describe the paper contribution, we need to introduce the following terms:

- White box setting: the adversary has full access to the trained model. Hence it can calculate accurate gradients of the loss w.r.t. the input and use it to generate adversarial attacks as we show next.
- First order setting: the adversary can use only first-order derivatives of the loss w.r.t. the input.
- Black box setting: the adversary has no knowledge or access to the model. A common approach is to use approximate gradient instead the accurate gradient. Another approach uses genetic search [3].

The Attacks: given input data x (e.g. input image) an adversarial attack (as defined before) outputs x_adv. We are going to discuss only on attacks that produce x_adv in the x's ball with radios ϵ $in\ l_{inf} - norm$ (we denote this ball as $B(x, \epsilon)$).

FGSM (Fast Gradient Sign Attack): a one-step gradient based method. Acts similarly to signed gradient accent to maximize the loss on x but for 1-step only. To ensure $x\_adv \in B(x, \epsilon)$ we add the sign of the gradient multiplied by ϵ instead the gradient itself. The algorithm:

```
$x\_adv = x + \epsilon\, sign(\nabla x\, L(\Theta, x, y)$
$return\ x\_adv$
```

PGD (Projected Gradient Decent): is a generalization of FGSM to multi-step. To force PGD result stay in $B(x, \epsilon)$ we project it after each step. We denote the input by x, step size by α and number of steps by N. S is the perturbations set ($S = B(x, \epsilon)$). Here the output can be inside $B(x, \epsilon)$. The algorithm:

```
$x^{(1)} = x.$
$x^{(t+1)} = \Pi_S\ (x\_adv + \alpha\, sign(\nabla x\, L(\Theta, x^{(t)}, y))$
$calcualte\ and\ return\ x^{(N)}$
```

$(\Pi_S\ t)\ is\ the\ projection\ of\ t\ on\ the\ set\ S\ in\ l_{inf} - norm.$ In our case $\Pi_S\ x = \mathrm{x} + \mathrm{clip}(\mathrm{y} - \mathrm{x}, -\epsilon, \epsilon))$ when clip(z, -a, a)=min(max(x, -a), a) (applies on each dimension of z) because this is projection on $l_{inf}\ - norm$ ball.

Transferred attacks: Related papers discuss the phenomenon that adversarial examples of similar architecture models that trained independent fool each other. Hence, black-box attacks can first apply a white-box attack on a similar network and then use its output as an adversarial example to the target model. This raises that protecting from first-order attacks only is not enough to protect against any black box attack but also transferred attacks should be considered. We see next that model capacity effects the resistance to transferred attacks.

Paper contributions:

- The paper shows a procedure to create models that are resistant to first order adversarial attacks. The procedure in the paper is universal – what means that it defenses from any first order attack unlike other papers that defense from a specific attack. To prove that, the authors show that PGD is universal attack (discussed later).
- The relation between model capacity and robustness (also to transferred attacks). Much more about this topic will be discussed in its dedicated section.

The Method - procedure to build robust models and analysis

Formalizing the optimization problem:

One of the most important contributions of the paper is defining and formalizing both attacking and defensing paradigms. They observe that defensing paradigms are solving a saddle point problem (min-max). In simple words, the formulation for robust model is a min-max problem where the max comes from the need to be resistant to **any** adversary and the min because we want to find a model with minimal loss on any adversary.

Therefore, the optimization problem is: $min_{\Theta} E_{(x,y)\sim D}[max_{\delta} L(\Theta, x + \delta, y)]$ where Θ is the model parameters, δ is a valid perturbation and $x + \delta$ an adversarial example, L is the loss function and D is the data distribution. The adversary can choose a perturbation δ to add the input, so the inner problem formalizes the best (highest) loss that an adversary can achieve for a given Θ. The outer problem searches for Θ that minimizes this. The inner problem is applying adversarial attack on the current Θ (a max problem).

PGD is universal attack:

The authors observed (verified empirically) that PGD is a "universal" adversary in the first-order setting – means that resistance to PGD attack implies resistant to any first order attack. To show that we need to prove that PGD solves the inner problem better than any other first-order gradient based method. To prove that (empirically), they start PGD from many random points and showed that the results distribution is with low variance and no outliers. Moreover, the local maximums are with same expected distance as the start points distances.

Any uniform first-order method (i.e. that works for any input and not data specific) must be stuck at a local maximum and assuming that PGD can reach any local maximum we conclude that it doesn't matter in which of the local maximum points we finished the first-order method search, we get a local maximum of PGD. Therefore, as analyzed empirically before, this local maximum is with same value as a PGD execution result.

It's important to mention that if we protect only from $1-\delta$ of PGD examples ($0 < \delta < 1$), the guarantee is for any adversary we protect from at least $1-\delta$ of the examples. In practice we solve with error.

Danskin's Theorem – and how it helps us to solve the optimization problem:

Denote $f(x) = max_{\delta} L(\Theta, x+\delta, y)$. To solve the outer problem, it is enough to find a decent direction of f and use SGD variant that subtracts a descent direction at each step. In this section we show how to calculate descent directions of f using Danskin's theorem.

Denote $\Delta = argmax_{\delta} L(\Theta, x+\delta, y)$. Danskin's theorem states that $D_h f(x) = sup_{\delta\in\Delta}\, h^T * \nabla_x L(\Theta, x+z, y)$. $D_h f(x)$ is the directional derivative of f in point x and direction h.
Corollary: For any $\forall\hat{\delta} \in \Delta$: $-\nabla_x L(\Theta, x+\hat{\delta}, y) \neq 0 \;\rightarrow$ is a descent direction.
Proof: denote $h = \nabla_x L(\Theta, x+\hat{\delta}, y)$ then $D_h f(x) = sup_{\delta\in\Delta}\, h^T * \nabla_x L(\Theta, x+\delta, y) \geq h^T h \geq 0$.

The first equality is Danskin's theorem. The second is because we can choose $\delta = \hat{\delta}$ - If the gradient is non-zero then $h^T h > 0$ and we get a descent direction.

Note that using previous section we can calculate find $\delta \in \Delta$ and therefore to solve the outer problem using the SGD variant - find $\delta \in \Delta$ and subtract the descent direction (derives from δ in Danskin's theorem) each step. This is exactly the procedure to build a robust network as we show in the next section.

The procedure (called adversarial training):
Train the model in batches and in each batch (x, y) construct adversarial examples to x, denoted as x_adv ,using PGD attack on the current network parameters. We train the network on (x_adv, y) as the batch in a gradient based optimization method (e.g. SGD or Adam) as usual.

Note that the only change from a regular training is that we train on the solution to the inner problem (i.e. the adversarial examples – x_adv) instead x. In practice mostly train on both adversarial and natural data. The procedure illustrations are under "experiments" section.

Analysis: Here we define Δ as PGD adversarial examples. Using the section on Danskin's theorem, the procedure solves the min-max formulation (because $\exists_{\hat{\delta}\in\Delta} x\_adv = x + \hat{\delta}$ and therefore $-\nabla_x L(\Theta, x\_adv, y)$ is a descent direction). The resulting model is the solution to the min-max problem on PGD attack (i.e. when solving the inner problem using PGD). Applying "Note" we conclude that if the model is resistant to PGD attacks than it resistance to any other first order attack. In case that the saddle point is resistant to PGD we find it and we done. (we also might stop in local minimum because we use SGD).

Capacity and Adversarial Resistance

Another major result of the paper is the relation between capacity and adversarial robustness. The equation $min_{\Theta} E_{(x,y)\sim D}[max_{\delta} L(\Theta, x+\delta, y)]$ value (the final loss we achieve) is highly depended on the model architecture (i.e. not every architecture is able to solve this min-max problem with low value). The authors found that in order to solve that equation we need a stronger model (i.e. with higher capacity) than to solve $min_{\Theta} E_{(x,y)\sim D}[L(\Theta, x, y)]$ (i.e. with no robustness considerations) since the presence of adversarial examples changes the decision boundary to a more complicated one. The following figure illustrate that phenomena

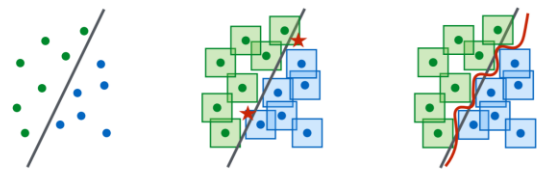


Figure: In the left side we see a set of points that can be separated with linear decision boundary (small capacity). In the middle we see that this linear decision boundary does not separate the $l_{inf} - norm$ (the squares) around the points. To protect against adversarial attacks, each point in the squares should be classified the same as the center of the square (the original point). Therefore, each point that not classified like that (i.e. marked by red star) is an adversarial attack. Moreover, we see in the right-side figure that a higher capacity classifier defines a decision boundary which is protected to adversarial attacks (i.e. classifies the points and their surrounding squares correctly).

The authors observed the following phenomena in the context of capacity and robustness relation:

1) Capacity alone helps. increasing the capacity of the model also increases the robustness to **one step** adversarial attacks while training only with natural examples (i.e. with the original data only). The authors note that the effect is greater when ϵ parameter is small.

2) Low capacity models may underfit while training with PGD adversary. For low capacity network training against a strong adversary (PGD) causes the network to underfit (i.e. does not learn any meaningful insights). This problem is equivalence to the saddle point of the optimization problem being high. A possible explanation is because there is a tradeoff between the accuracy on natural examples and adversarial examples which low capacity model can't be solved by low capacity models.
3) The value of the saddle point problem decreases as we increase the capacity. Continuing the previous section, in order to get both high resistance to adversarial example and accuracy we need more complicated classifiers (i.e. with higher capacity). Therefore, as long we increase the capacity the model it is more suitable to solve the problem. The figure in the previous page illustrates this observation.
4) More capacity and stronger adversaries decrease transferability. The observation is that using strong adversaries as PGD and increasing the capacity reduces effectiveness of transferred adversarial examples. The intuition behind this observation is that for high capacity networks, the optimization mostly converges to one of many local minimums of the loss. While in low capacity, the optimization mostly converges to optimal[4]. Therefore, in low capacity networks we can re-train the network (that called surrogate model) and to hope that the parameters will be close. Then as we illustrate in the figure, we can easily transfer the adversarial examples.

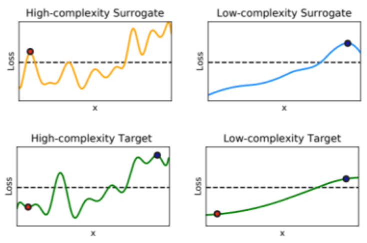


Figure: In reference [4] they showed that the loss function of the attack objective as a function of a single feature x. At the top row there are the surrogate models (i.e. the models that are with same architecture as the target but trained independently) and at the bottom the target models. The right side of the image contains the example on high complexity target model and as we see the adversarial example in red at the surrogate is not an adversarial example in the surrogate. But for the left side we see the opposite – the surrogate and the target behavior are almost the same. Therefore, an adversarial example of the surrogate most probably be an adversarial example also to the target. That illustrates the point that low capacity networks are more vulnerable to adversarial attacks transferring.

These phenomena illustrations are in "experiments" section.

Additional Content

Label Leaking: when training with one-step adversarial attack a known phenomenon is that the model test error on the generated examples is lower than on the natural examples. What happens called label leaking. It happens because when we construct adversarial examples we use the truth label to construct it, therefore it has extra information about the truth label that the original input doesn't have. That was an introduction to the next observation which is that in FGSM with large ϵ the outcome model isn't robust. Because that the network parameters are fixed and for any two images with same label the advanced layers values on them should be very close and therefore the gradient of the loss w.r.t. the input will be very correlated. We add this highly correlated terms multiplied by ϵ that assumed to be a high scalar. Therefore, the generated adversarial examples with same label are correlated and is enough to classify a few of them to classify all of them. For small ϵ they are much less correlated, and the output is closer to the PGD constructed examples. A possible solution I thought can be useful for that is adding also original data to the adversarial training. This should encourage the model to ignore the extra information because it doesn't exist in the original examples. In experiment 5 I show that this actually solves the problem.

MNIST Inspection (Appendix C):

The MNIST robust network is small enough that it can be visually inspected as the authors did and understand the following behaviors of robust networks:

- By examine the first convolutional layer. Only 3 filters of total 32 where used and for each one only one weight is non-zero. Because we use RELU activations the first layer applies a threshold on the input image $Relu(\alpha x - \beta)$ while β is the bias and α is the non-zero weight.
- We also see that in output layer the robust networks utilize bias more than the standard network (it means the bias classes vector is not uniform as it is in the standard network). An interpretation to this phenomenon is that some classes are more vulnerable than others so on these classes the bias smaller.

The authors tried to manually apply those modifications to the standard network but with no success. [Appendix D in the paper for examples]

Related work

We have already talked about some related work as generated as targeted attacks. Moreover, we talked about transferring adversarial attacks and the art of practically apply adversarial attack in real life. The following points shortly review some more related topics.

- Query Limited Setting – In this setting the attacker can ask for a classification result (in black box) or the input gradient / any other knowledge (in white box) for limited number of queries. Each time that we ask for information from the classifier is a query. This setting might be useful for example when each query costs money. An example for such situation in black-box scenario is The Clarifai NSFW where after the first 2500 prediction each prediction costs money. Therefore, the study of attacking and defensing in this setting is highly important. [5]
- Label only setting – a black box setting in which the classifier outputs only the predicted and not the accuracy. Therefore, we get a 0-1 loss and usual black-box PGD doesn't work well (both approximating derivatives and applying optimization on 0-1 loss – for this we usually use a surrogate loss).
  For an example to adversarial attack in this setting see RayS [6].
- The writers established a competition to fool the robust network they build on MNIST and CIFAR on both black-box and white-box. We see that during the last 2-3 years that the competition exists, no one succeed to decrease the accuracy on black box to more than 92% and in white box to more than 88%. See [7], [8], [9] for attacks that participated in this competition.
- Related papers show that adversarial examples issues are not specific to computer vision or signal processing systems (continues inputs) but also to natural language processing applications with more limited perturbations set [10].
- Fine-Tuning approach enables training neural networks faster and training with less available data. Recently (Rezaei & Liu, 2020 [11]) demonstrated that fine-tuned models are vulnerable to adversarial examples crafted solely based on the pre-trained model. Knowing that, the attacker can construct adversarial examples based on the models available online (such as in torchvision) in white-box approaches. "Improving the Adversarial Robustness of Transfer Learning via Noisy Feature Distillation" [12] shows a method to improve the robustness of transferred network against the attacks that design specifically for the pre-trained model.

Practical

Overview

In the paper the authors applied the method of adversarial training exactly as shown on MNIST dataset and achieved both high accuracy and resistant to adversarial attacks. This might lead us believe that the problem of training robust models on a general dataset is solved. Unfortunately, on large-scale problems we are nowhere close to build robust models that match the standard models in terms of performance. For example, even on CIFAR10 the authors couldn't train a robust model with high performance. For that reason, they restricted the attacks perturbations set (decreased ϵ from 0.3 on MNIST to $\frac{8}{255} \approx 0.03$ which is far from real attacks scenario). The best-known resistant model on CIFAR10 has around 70% accuracy and 55% of resistant to PGD based attack.

To increase the adversarial training accuracy people often add also natural examples to the adversarial generated batch in training phase (as explained before and discussed later). Moreover, in adversarial training, the choices of optimizers parameters, architecture, regularization, etc, significantly influence the adversarial training performance (for example see "capacity and adversarial resistance" section of this file). All of this makes adversarial training on real world case studies as face recognition or autonomous vehicles components very hard and maybe not possible yet.

Saying that, there are some benefit of robust models right now. It allows to get better inputs representations [13] and extract meaningful features [14]. Some more examples can be found in the paper "Adversarial Examples Are Not Bugs, They Are Features" [14].

Experiments Environment

In the rest of the project, I train and analyze the adversarial training on a real-world application, traffic sign classification, and encountered the difficulty of doing so. I encountered the problems mentioned before as label leaking, underfitted models and searching the right architectures and training methods.

The Traffic Signs Classification case study is highly important because is a component of autonomous vehicles - In the introduction we discussed more about this topic and the dangerous of adversarial attacks there. The experiments verified on MNIST dataset to ensure correctness of the experiments.

About the dataset: I decided to work with the GTSRB (German Traffic Sign Recognition) dataset. I took the data from - https://github.com/tomlawrenceuk/GTSRB-Dataloader (please download it from my repo). I applied normalization transformer to scale the data to $[0,1]$ scale. The normalization is on each channel separately. The project is on github, I recommend downloading the datasets and code from there. If there is any problem please send me an email to eldadperetz@mail.tau.ac.il. The link: https://github.com/eldadp100/cnn_course_final)

The models: In the experiments I applied the attacks on a CNN on both GTSRB and MNIST datasets. To test the capacity and robustness relation I used increasing capacity of CNNs (i.e. increased channels amount and number of layers). I noticed that the kernel size and other standard networks components (as dropout and pooling layers) influences the networks performance very match and differently than in standard training. For example, I found that using dropout and pooling layers decrease the performance and therefore I decided not to use them. In the paper we see that the network should be not low capacity in order to apply adversarial training which causes to computational performance and performance trade-off. The specific networks I chose are in models.py file. In short, are CNNs 3-4 conv layers and another 2 linear (fully connected) layers. I used 5x5 kernels and dropout as regularization with $p = 0.05$. I found that higher values of p decreases the performance significantly.

Setting: To apply adversarial training, various of possible configurations are considered for both attacks and training parameters. All combinations of parameter are tested in grid search method. I apply adversarial training for each and then we choose the model with highest accuracy on validation (also resistance measurements could be considered, but the bottleneck in all experiments on GTSRB is the accuracy on natural samples. Therefore, I used accuracy as the only measurement to consider when choosing a model in the hyperparameter search. In MNIST the results were easy to achieve even without hyperparameter search).

I tested the robustness of the adversarial trained networks with a various of attacks and chose the worst case as the measurement. In experiment 3 are the rest of the attacks scores. The set of attacks to train and test is not the same and treated separately in the hyperparameter search.

On MNIST: I tested with the following sets: on FGSM ϵ=0.3. For PGD same ϵ value and #steps$\in \{20, 7, 40, 100\}$ and α=0.01. Those are both very strong attacks (as long ϵ is higher the attack should perform better but adversarial examples difference will be more distinguishable). We get very good accuracy and robustness results and therefore I used strong attacks in contrast to GTSRB that I used weaker attacks. The paper does the same.

<u>On GTSRB</u>: I needed to find a suitable ϵ. On CIFAR10 the authors used $\epsilon = \frac{8}{255}$ which is very small. I saw that I get good results also on higher ϵ, and finally I decided to test on $\epsilon = 0.22$ for two reasons:

1. With $\epsilon = 0.22$ the normally trained network is very not resistant to the attacks (i.e. its accuracy on PGD and FGSM generated examples is low). It gives us the opportunity to show that using adversarial training we do better in terms of resistance.
2. We successfully achieved not bad resistance results on the dataset. The results are in the same scale as in the paper.

I also used #steps ∈ {7, 20, 30} and $\alpha = 0.01$ exactly as tested in the paper on CIFAR10. I found that doing the adversarial training on both natural and adversarial generated data is more effective on test measurements and used in practice, it allowed to increase the ϵ value. I used this tip only on GTSRB – on MNIST I applied the same method that done in the paper. train the robust classifier. I used the same values to attacks training sets hyperparameters.

More about the project technical details is discussed next in “Appendix 1 - project technical details”.

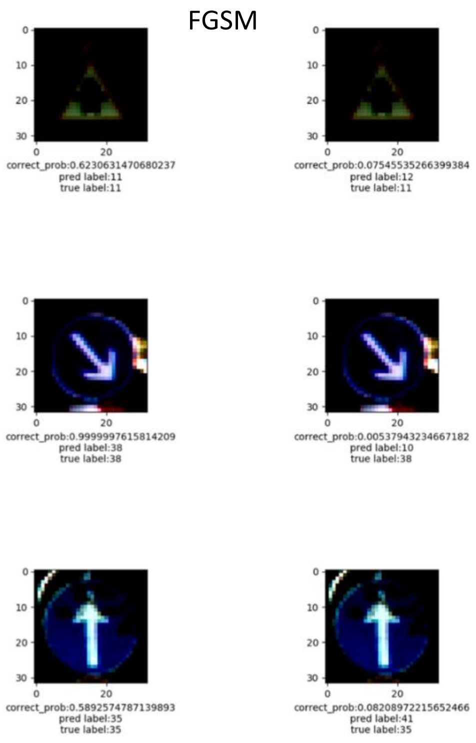
Note: all experiments measurements are on test dataset.

Experiments

Experiment 1: In this experiment we will attack a network using PGD and FGSM attacks. The experiment illustrates that PGD and FGSM attacks works also on GTSRB dataset. We train and attack a Spatial Transformer Network which is invariant to geometrical transformations (rotations and scaling). The results verify the paper.

CNN default training (measured on Test)			
	Accuracy on Natural examples	Accuracy on PGD adversarial examples	Accuracy on FGSM adversarial examples
GTSRB	93%	5%	17%
MNIST	99%	2%	17%

The results show the attacks works perfectly also on GTSRB (as on MNIST). Unfortunately, the adversarial training on GTSRB will not work well as on MNIST (This problem occurs also on CIFAR10 in the paper). Some adversarial examples for GTSRB with normal training (more figures in appendix D):



FGSM
correct_prob:0.6230631470680237
pred label:11
true label:11
correct_prob:0.07545535266399384
pred label:12
true label:11
correct_prob:0.9999997615814209
pred label:38
true label:38
correct_prob:0.00537943234667182
pred label:10
true label:38
correct_prob:0.5892574787139893
pred label:35
true label:35
correct_prob:0.08208972215652466
pred label:41
true label:35

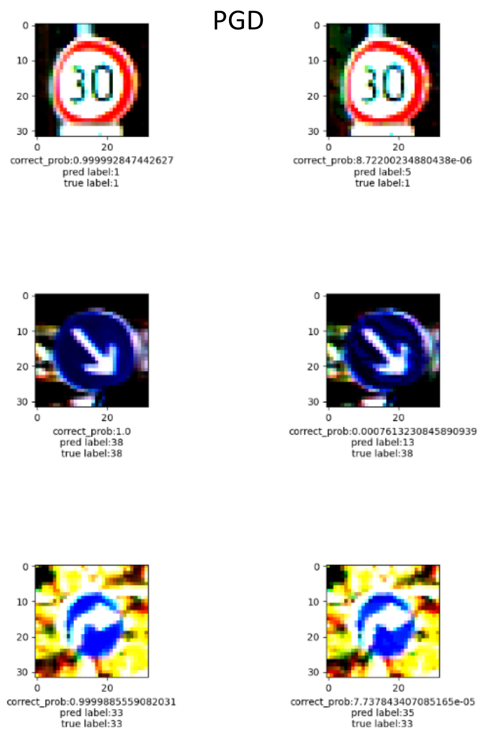


Figure: On the left side we see the FGSM constructed examples. On almost the same pictures (up to small amount of noise) the trained model classifies the natural example taken from the dataset correctly with high confidence and the adversarial example that looks almost the same incorrectly with high confidence that is not the true label. On the right side we see the PGD constructed examples. The main difference is that in the PGD examples the model is much more confident that the sample is not the truth label – what means it has a negligible probability to classify correctly (specifically for PGD around e^{-8} while for FGSM around e^{-3}).

Experiment 2: In this experiment we will use adversarial training (the paper procedure) in order to make the network from experiment 1 resistant to FGSM and PGD attacks separately. Denote the network that trained in adversarial training using FGSM as Net_1 and the network trained with PGD as Net_2. Then we test Net_1, Net_2 robustness as we did in experiment 1 and show the following:

1. Net_1 is resistant to FGSM attack but not to PGD attack. What means resistant against FGSM doesn't yields resistance against PGD.
2. Net_2 is resistance to both FGSM and PGD attacks. This is a motivation to experiment 3 that shows PGD is a universal attack. (i.e. that resistance against PGD yields resistance to any other first order attack).

as explained in the method section in adversarial training we want to solve $min_\Theta E_{(x,y)\sim D}[max_\delta L(\Theta, x+\delta, y)]$. Where in Net_1 the inner problem solver is FGSM and for Net_2 is PGD.

Results on FGSM adversarial training:

- Resistant against FGSM doesn't yields resistance against PGD.

CNN adversarial training with FGSM (measured on Test)			
	Accuracy on Natural examples	Accuracy on PGD adversarial examples	Accuracy on FGSM adversarial examples
GTSRB	92%	14%	54%
MNIST	99%	4%	97%

We see that the resulting models are highly not resistant to PGD attacks on both GTSRB and MNIST. Therefore, we verified that FGSM adversarial training is not resistant against PGD as we claimed. I trained with FGSM on not high ϵ as the paper suggests (used $\epsilon = 0.15$).

Results on PGD adversarial training:

- The method produces high accuracy models which are resistant to FGSM and PGD adversarial attacks on testing samples.
- We see that on MNIST the resulted model is much more resistant to adversarial attacks than in GTSRB.
- Resistant to PGD attacks yields resistant to FGSM attack as we see.

CNN adversarial training with PGD (measured on Test)			
	Accuracy on Natural examples	Accuracy on PGD adversarial examples	Accuracy on FGSM adversarial examples
GTSRB	93%	39%	52%
MNIST	98%	95%	97%

The models we trained in this experiment are resistant to FGSM and PGD. In MNIST, the adversarial training is fast and produces a very high accuracy model that is almost fully resistant (i.e. resistant more than 95% of the testing samples) on FGSM and PGD. Because PGD is universal, the robust network on MNIST is resistant to any first order attack with error of 5%. In GTSRB the training was more difficult, therefore, I used an in-practice method to apply adversarial training which is adding also natural examples to the adversarial generated as explained before. This method yields the same robustness as the original adversarial training but with higher accuracy on natural examples.

Both datasets were trained with the same system. This phenomenon occurs also on the paper and therefore I assume is not a mistake in my experiments.

Experiment 3:

In this experiment we will attack Net_2 from experiment 2 using a various of adversarial attacks with different parameters. We want to show that Net_2 is resistance to all of them. This is an evidence that we can practically use adversarial training with PGD as done in Net_2 to defend through stronger attacks (PGD with more steps or higher ϵ. We also show the effect of increasing ϵ, we will see that the robustness results are lower as long we increase ϵ and show an example of constructed adversarial examples with $\epsilon = 0.3$ which considered high.

On MNIST:

CNN default training (not adversarial) – MNIST	
	Accuracy on constructed examples (from test dataset)
PGD $\epsilon = \frac{10}{255}$ # steps=7 $\alpha = 0.01$	0.68
PGD $\epsilon = \frac{30}{255} - actualy\ 0.07$ # steps=7 $\alpha = 0.01$	0.64
FGSM $\epsilon = 0.3$	0.18
PGD $\epsilon = 0.2$ # steps=20 $\alpha = 0.01$	0.13
PGD $\epsilon = 0.3$ # steps=40 $\alpha = 0.01$	0.13
PGD $\epsilon = 0.3$ # steps=100 $\alpha = 0.01$	0

CNN adversarial training with PGD – MNIST	
	Accuracy on constructed examples (from test dataset)
PGD $\epsilon = \frac{10}{255}$ # steps=7 $\alpha = 0.01$	0.97
PGD $\epsilon = \frac{30}{255}$ # steps=7 $\alpha = 0.01$	0.96
FGSM $\epsilon = 0.3$	0.98
PGD $\epsilon = 0.2$ # steps=20 $\alpha = 0.01$	0.98
PGD $\epsilon = 0.3$ # steps=40 $\alpha = 0.01$	0.95
PGD $\epsilon = 0.3$ # steps=100 $\alpha = 0.01$	0.9

On GTSRB:

CNN FGSM training (not adversarial) – GTSRB	
	Accuracy on constructed examples (from test dataset)
PGD $\epsilon = \frac{10}{255}$ # steps=7 $\alpha = 0.01$	0.35
PGD $\epsilon = \frac{30}{255}$ # steps=7 $\alpha = 0.01$	0.23
FGSM $\epsilon = 0.2$	0.2
FGSM $\epsilon = 0.3$	0.16
PGD $\epsilon = 0.2$ # steps=20 $\alpha = 0.01$	0.06
PGD $\epsilon = 0.3$ # steps=30 $\alpha = 0.01$	0.03
PGD $\epsilon = 0.3$ # steps=100 $\alpha = 0.01$	0.02

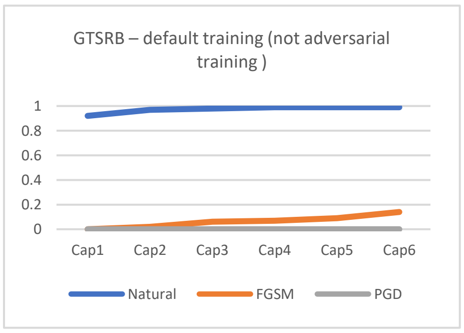
CNN adversarial training with PGD – GTSRB	
	Accuracy on constructed examples (from test dataset)
PGD $\epsilon = \frac{10}{255}$ # steps=7 $\alpha = 0.01$	0.66
PGD $\epsilon = \frac{30}{255}$ # steps=7 $\alpha = 0.01$	0.58
FGSM $\epsilon = 0.22$	0.55
FGSM $\epsilon = 0.3$	0.49
PGD $\epsilon = 0.2$ # steps=20 $\alpha = 0.01$	0.48
PGD $\epsilon = 0.3$ # steps=30 $\alpha = 0.01$	0.38
PGD $\epsilon = 0.3$ # steps=100 $\alpha = 0.01$	0.28

As we see, the adversarial trained networks are more robust on any of those attacks than the default (not adversarial) training. We also see that using adversarial training on specific $\epsilon, \#steps, \alpha$ we achieve robustness for more stronger adversaries (i.e. higher $\epsilon\ or\ \#steps$) but much less robustness that we were achieve using adversarial training on the stronger adversaries. The results are very similar to those of the paper in terms of robustness.

Experiment 4 (Capacity and Adversarial Robustness): This experiment will examine the 3 following statements:

1. Capacity alone helps: High capacity models are more robust to adversarial attacks than low capacity models.
2. Weak models may fail to learn non-trivial classifiers, : We show that we might fail to apply the paper procedure to create a robust model if the model is with too low capacity. Specifically, we will take networks in increasing capacity and show that after adversarial training with PGD on the low capacity models we get extremely underfitted models. While for high capacity we succeed to create robust model.
3. The experiment of section 2 also shows the value of the saddle point problem decreases as we increase the capacity.

To show 1, I applied training (without adversary) and testing as explained in "setting" (page ___) on exponentially increased width convolutional neural networks. We see that the paper results are verified also in this experiment.



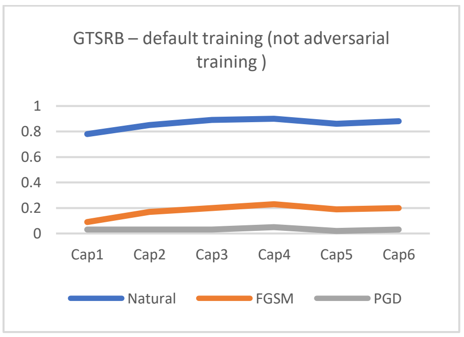
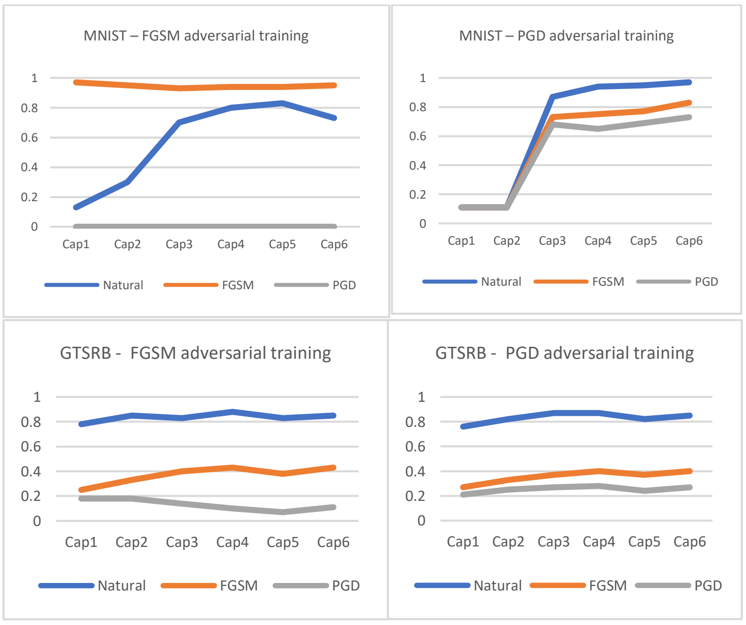


Figure: The exact values are in the tables below. In the figure are the testing results on natural / FGSM / PGD constructed examples (from test dataset) on natural trained **(not adversarial training**) exponentially increased width (explained next) CNNs. On both MNIST and GTSRB the results verify that "capacity alone helps" – increasing the model capacity and training normally increasing the robustness to FGSM attacks as we see. It also increases the robustness to PGD.

About the models' architectures: From capacity 1 (lowest capacity) to capacity 6 (highest) I increased exponentially the width of a 2-layers convolution and 2 of fully connected. I also applied a 1x1 convolutional after the convolutional layers with linear increased width. In contrast to experiment 2, I didn't used dropout as it hurt the performance of some models.

To show 2+3, I applied robust training on both PGD and FGSM adversaries on the CNNs increased capacity we tested in 1. The results plots:



On MNIST where I applied the paper method for adversarial training (i.e. trained only on adversarial generated examples), low capacity networks can't train against PGD, that because PGD is a strong adversary that increase the saddle point value of the optimization problem. We also see that training against FGSM is possible for low capacity networks. On GTSRB because the method produces low accuracy on natural examples (which is a known phenomenon in not trivial datasets) I added natural examples to the adversarial generated examples and the results was better also to low capacity networks (Adversarial training is done in practice as I did in GRSRB case study). We conclude 2 as we see that low capacity models the accuracy on natural examples in lower after adversarial training than for higher capacity models. We also conclude that if we add natural examples to the adversarial inputs we succeed to train also in low capacity models. We also see that as long we increase the capacity, the adversarial training with PGD adversary is more robust and with higher accuracy.

Results Tables:

MNIST - Original training	**Cap1**	**Cap2**	**Cap3**	**Cap4**	**Cap5**	**Cap6**
Natural	0.92	0.97	0.98	0.99	0.99	0.99
FGSM	0	0.02	0.06	0.07	0.09	0.14
PGD	0	0	0	0	0	0

MNIST – FGSM adversarial training	**Cap1**	**Cap2**	**Cap3**	**Cap4**	**Cap5**	**Cap6**
Natural	0.13	0.3	0.7	0.8	0.83	0.73
FGSM	0.97	0.95	0.93	0.94	0.94	0.95
PGD	0	0	0	0	0.00	0.00

MNIST – PGD adversarial training	**Cap1**	**Cap2**	**Cap3**	**Cap4**	**Cap5**	**Cap6**
Natural	0.11	0.11	0.87	0.94	0.95	0.97
FGSM	0.11	0.11	0.73	0.75	0.77	0.83
PGD	0.11	0.11	0.68	0.65	0.69	0.73

GTSRB – Original training	**Cap1**	**Cap2**	**Cap3**	**Cap4**	**Cap5**	**Cap6**
Natural	0.78	0.85	0.89	0.9	0.86	0.88
FGSM	0.09	0.17	0.20	0.23	0.19	0.2
PGD	0.03	0.03	0.03	0.05	0.02	0.03

GTSRB - FGSM adversarial training	**Cap1**	**Cap2**	**Cap3**	**Cap4**	**Cap5**	**Cap6**
Natural	0.78	0.85	0.83	0.88	0.83	0.85
FGSM	0.25	0.33	0.40	0.43	0.38	0.43
PGD	0.18	0.18	0.14	0.1	0.07	0.11

GTSRB - PGD adversarial training	**Cap1**	**Cap2**	**Cap3**	**Cap4**	**Cap5**	**Cap6**
Natural	0.76	0.82	0.87	0.87	0.82	0.85
FGSM	0.27	0.33	0.37	0.4	0.37	0.4
PGD	0.21	0.25	0.27	0.28	0.24	0.27

In GTSRB, we add also the natural examples to the adversarial training (as explained before) and therefore the low capacities models can achieve not trivial results (also much more robust! - compared to MNIST that done with adversarial examples only).

The cells that marked in yellow are a result of **label leaking.** Explained in the theoretical part. We also see that as long the network is with higher capacity then it's more resistant to the label leaking problem. This happened only on MNIST, a possible explanation to this phenomenon is that on GTRSB there are more labels and therefore more options to construct an adversarial example and the correlation of the added perturbation is smaller. In the next experiment I show that with adding natural examples in adversarial training label leaking doesn't happen.

Another phenomenon we see (marked in green) is that FGSM adversarial trained networks are more robust to PGD for low capacity than in higher capacity. It can be explained as for low capacity networks FGSM and PGD attacks produces more similar examples than for higher capacity attacks.

Experiment 5: (My addition)

In this experiment I will examine the technique of adding natural examples to the adversarial generated batches in adversarial training. We used this technique to improve the performance on GTSRB but also on MNIST there this method improves the performance significantly. I will illustrate the following:

1. With adding natural examples, we can apply adversarial training also on low capacity networks.
2. Adversarial training with FGSM adversary (on large ϵ=0.3) doesn't suffer from label leaking on MNIST.

1. CNN adversarial training with PGD for Low capacity network (cap 2)			
	Accuracy on Natural examples	Accuracy on PGD adversarial examples	Accuracy on FGSM adversarial examples
MNIST with standard adversarial training	11%	11%	11%
MNIST with also natural examples in the adversarial training	93%	38%	50%

We see that the network that trained with standard adversarial training is underfitted – performs as random classifier. Applying the adversarial training with adding the natural examples improves the performances.

To show 2, I repeated experiment 4 for adversarial training with natural examples. The results in the next 2 tables.

MNIST – FGSM standard adversarial training	**Cap1**	**Cap2**	**Cap3**	**Cap4**	**Cap5**	**Cap6**
Natural	0.13	0.3	0.7	0.8	0.83	0.72
FGSM	0.97	0.95	0.9	0.92	0.93	0.95
PGD	0	0	0	0	0.00	0.00

MNIST – FGSM adversarial training with natural examples	**Cap1**	**Cap2**	**Cap3**	**Cap4**	**Cap5**	**Cap6**
Natural	0.93	0.96	0.97	0.98	0.98	0.99
FGSM	0.7	0.8	0.89	0.93	0.95	0.97
PGD	0	0	0	0	0	0

We see that the network that suffered from label leaking in experiment 4 is resistant to label leaking problem when training it with also natural examples as explained. The intuition behind is discussed in the theoretical part in "Additional Content" section, in short the idea is it encourage the training to succeed also on the natural examples that doesn't contain the extra information and therefore the optimized network will not use it as it leads to incorrect classification for the natural examples. For high capacity networks it might find a complex decision boundary that separates FGSM and natural examples and therefore will suffer from label leaking as we see in the table – if we apply early stopping the results are better in this case. In standard adversarial training, for high capacities the training suffers from label leaking again, it doesn't in adversarial training with added natural examples.

Note: In configs.py adding natural examples in adversarial training is configurable (see "add_natural_examples" in "MNIST_experiments_configs" dictionary.)

Appendix 1 - project technical details

In this section I review the project code and discuss the implementation details.

<u>The configurations:</u> (configs.py)

- Random seed: used to reproduce results.
- Paths to save log (see logger.py), results plots and trained models.
- Train / validation sets ratio
- Turn on/off prints of optional information as validation accuracy, plot figures, save figures and save / load checkpoints.

<u>Dataset specific configurations:</u> (configs.py)

- Data transformers if needed.
- Training and adversarial training stopping criteria. (see trainer.py StoppingCriteria class – I implemented constant stopping after pre-defined number of iterations or pre-defined time duration in seconds. I also implemented early stopping).
- Loss function – I used CE loss in both projects.
- Add natural examples: this is a technique for adversarial training that discussed both in the practical and theoretical sections of this file. It's adding also the natural examples to the batch of adversarial generated examples. It's mostly used in non-trivial datasets. I used it in GTSRB dataset and not used in MNIST.

<u>The models:</u> (models.py)

- The CNN networks for both GTSRB and MNIST datasets are in models.py.
    - MNIST - CNNMNISTNet
    - GTSRB – CNNTrafficSignNet

  They are both 3-4 convolutional layers followed by 2 fully connected layers. I used ReLU activations in all models. I also used dropout and maxpool layers. For more details see these classes implemetations.
- In experiment 4 there are convolutional neural networks in different capacities. To create these networks efficiently, I implemented a CNNs generator (ConvNN class).

<u>Hyperparameters:</u> (helper.py - see HyperparamsGen class)

I implemented hyperparameter generators for grid search method. I decided to implement it as a generator to save memory in case there are many possible combinations. In the project there are 5 hyperparameters sets:

- Neural networks training hyperparameters: batch size, learning rate and learning rate decay parameter.
- PGD Attack hyperparameters set: N (number of steps), α (step size), ϵ (projection parameter)
- PGD Defense hyperparameters set: N , α , ϵ
- FGSM Attack hyperparameters set: ϵ
- FGSM Defense hyperparameters set: ϵ

Each adversarial attack procedure has attack and defense set of parameters. The defense set is used in order to adversarial train the network. We stack it to the training parameters and execute on every combination of them. Then choosing the best result hyperparameter (as explained in the experiments section). The attack set is used to measure the resistance to this attack (i.e. we calculate resistance measure on test for each combination and choose the on with highest loss – which is the strongest adversary in the set of parameters.

The method full_train_of_nn_with_hps and full_attack_of_trained_nn_with_hps (implemented in helper.py) are responsible to do the hyperparameter searches. The generators classes also implanted in helper.py.

<u>Data Processing:</u> This section is relevant only for GTSRB dataset. I loaded and scaled the data from the folders using torchvision transformers. The dataset scaled to [0,1] on each of the 3 channels separately using subtract the mean and dividing by reasonable variance (which is the root of the highest pixel value).

<u>Losses:</u> I used the cross-entropy loss and therefore didn't apply softmax at the last layers of the used networks for better numerical stability (pytorch CrossEntropyLoss already does this).

References

1. Robust Physical-World Attacks on Deep Learning Visual Classification: https://arxiv.org/pdf/1707.08945.pdf
2. Adversarial Training Can Hurt Generalization: https://arxiv.org/pdf/1906.06032.pdf
3. Automatically Evading Classifiers: https://evademl.org/docs/evademl.pdf
4. Why Do Adversarial Attacks Transfer? Explaining Transferability of Evasion and Poisoning Attack https://arxiv.org/pdf/1809.02861.pdf
5. Black-box Adversarial Attacks with Limited Queries and Information https://arxiv.org/pdf/1804.08598.pdf
6. RayS- https://arxiv.org/pdf/2006.12792.pdf
7. Distributionally Adversarial Attack - https://arxiv.org/pdf/1808.05537.pdf
8. Diversity can be Transferred: Output Diversification for White- and Black-box Attacks - https://arxiv.org/pdf/2003.06878.pdf
9. Generating Adversarial Examples with Adversarial Networks - https://arxiv.org/pdf/1801.02610.pdf?
10. TextAttack: A Framework for Adversarial Attacks, Data Augmentation, and Adversarial Training in NLP - https://arxiv.org/pdf/2005.05909.pdf
11. A TARGET-AGNOSTIC ATTACK ON DEEP MODELS: EXPLOITING SECURITY VULNERABILITIES OF TRANSFER LEARNING - https://openreview.net/pdf?id=BylVcTNtDS
12. Improving the Adversarial Robustness of Transfer Learning via Noisy Feature Distillation - https://arxiv.org/pdf/2002.02998.pdf
13. Adversarial Robustness as a Prior for Learned Representations- https://arxiv.org/pdf/1906.00945.pdf
14. Adversarial Examples Are Not Bugs, They Are Features- https://arxiv.org/pdf/1905.02175.pdf