Adversarial attack and training - Discussion

Sander Zeeman, Thijs Havinga & Jasper van Thuijl

November 2021

Discussion

Adversarial examples are malicious inputs with the specific purpose of confusing machine learning models. In this project we have implemented a variety of adversarial attack methods that can be used to generate such examples. The adversarial attack methods that we implemented are:

- Fast Gradient Sign Method (FGSM) [1] & related methods [2]
- JSMA [3] & M-JSMA [6]
- Pixel based attack [5]

Besides implementing Adversarial attack methods, we have evaluated several ways to defend against these attacks. The evaluated defence mechanismes include: Adversarial training and adding noise to the input images before classification. In this document, we aim to show our personal opinions on adversarial attacks and mitigating these attacks. Furthermore, we will give insight into our experiences in creating the implementations.

Fast Gradient Sign Method (FGSM) & Related methods (Implemented by Jasper)

This subsection corresponds to the notebooks `fsgm.ipynb` and `fgsm-mnist.ipynb` located in `/code/FGSM/`.

Regarding the problem and 'solution'

The main advantage of the Fast Gradient Sign Method (FGSM) is that it is simple to understand and relatively easy to implement. Its variation, the One-stop Least Likely Class method, is also relatively easy to implement. The iterative methods such as Basic Iterative Method (BIM) and Iterative Least-Likely Class are slightly more complex. The effectiveness of these attacks are quite worrying however. Although [2] state that this method has the lower success rate compared to other more complex methods, we were clearly able to demonstrate its effect on the example image. Even more worrying are the more complex methods such as BIM, which results in a even higher classification probability for the incorrect labels.

Adversarial training seems to be effective against FGSM attacks, since the error rate of the Adversarial examples is much lower after training. However, we have only trained our robust model with a examples generated using a single epsilon value. This results in the model only being robust against attacks of that exact episilon value. As demonstrated in the notebook, examples generated using other epsilon values still result in a high loss values, and are thus still able to fool the robust model. A solution to this is to choose the epsilon values randomly for each training example, however this will also result in an increased training time. For relatively simple datasets and models, this might be okay, however this might become a bigger issue for larger and more complex datasets.

My experience & Reflection

Although the mathematical notation of FGSM and related methods seems simple, implementing the methods was harder than expected. This might in part be due to the fact that I personally do not have much experience with pytorch, and other machine learning frameworks. Another aspect that proved to be difficult was the choice of dataset. I first chose to use ImageNet, since this leads to the most interesting and relatable results. However, for the adversarial training part this was not a good choice, since retraining would require a lot of time. For this reason I chose to MNIST instead (similar to Sander). However, the CIFAR-10 dataset would also be a good (if not better) choice. Unfortunately, due to time reasons I was unable to try this dataset. Furthermore, the ResNet-50 model is quite complex. For prototyping, choosing a more simple model would have been better. Regarding the attack methods, this project has sparked my interest in machine learning security research. Before this project I would not have thought that it would be so easy to circumvent correct predictions of a machine learning model.

JSMA & M-JSMA
(Implemented by Sander)

This subsection corresponds to the notebooks `/code/JSMA/attack.ipynb` *and* `/code/JSMA-M/attack.ipynb`.

Regarding the problem and 'solution'

Jacobian-based Saliency Map Attack (JSMA) JSMA (introduced in [3]) showed very promising results, obtaining a 100% success rate, as described in the notebook. However, these results were not as good as one would expect. While they certainly did fool the model, the images were often so broken from the attack, that humans would be tricked by the images as well. This is not the goal of these attacks, therefore, we conclude JSMA is most effective when a human can guide it, by giving it a fitting target and a fitting saliency map type (positive/negative).

When it comes to time complexity, JSMA is not too bad. A single JSMA will run in approximately 3 seconds, which is acceptable.

Finally, let us discuss the results of adversarial training. As also mentioned in the notebook, adversarial training managed to stop 10% of the attempted attacks. Since the adversarially trained model was based on the FGSM attack, rather than JSMA, this is to be expected. The fact that this model managed to prevent 10% of attacks gives us great hope that a model that was properly trained on jacobian-based methods would be very effective at preventing these types of attacks.

Maximal Jacobian-based Saliency Map Attack (M-JSMA) M-JSMA (introduced in [6]) seemed to improve on JSMA in most ways. M-JSMA obtained adversarial examples that the classifier wrongly classified, while still looking like the original to any human. Additionally, this attack was non-targeted, which means no human interaction was required to make this attack return a strong adversarial example. The only downside to this attack is the time complexity. While JSMA ran in approximately 3 seconds, M-JSMA took 4 minutes on average. However, I do not consider this a problem in the attack. It caused some set backs in the adversarial training, however, in general, I believe a complex attack is not necessarily bad. GPUs are getting stronger every year, so attacks that may seem slow today, may be very viable in the near future, or even today, with the right resources.

The adversarial training once again used the adversarially trained model from FGSM, because of the reasons mentioned above. While this model did not manage to stop an attack, it did cause one attack to change its approach, which makes me believe that, if we took the time to properly train a model on jabobian-based attacks, this model would have the potential to defend against these jacobian-based attacks.

My experiences

This project got off to rough start for me. By simply following the provided tutorial, I decided on using the pre-trained resnet50 model. Combining this with the fact that I had little experience with training neural networks, meant that I tried to make this model work for a bit too long, until I accepted that resnet50 was too large for a non-trivial attack such as (M-)JSMA. Each run took an average of 12 minutes, which wasted a lot of time. However, after getting more familiar with the PyTorch environment and switching to an MNIST classifier (by the PyTorch repository[1]), implementing the pseudocode that was given in the corresponding papers, as referenced above, was not that difficult. The mathematical notation in the papers seemed very clear. Overall, even though I got off to a rough start, the project ended up being very interesting and taught me a lot about Neural Networks (and ways to attack them) as a whole.

Reflection

As mentioned in the previous section, a lot of time went into trying things that did not work, only to replace them by more things that did not work, before figuring out the correct way of doing

[1] `https://github.com/pytorch/examples/tree/master/mnist`

something. In hindsight, perhaps doing some research before jumping into the implementation would have been a good idea. Additionally, thinking about the dataset and its implications would have saved a lot of time, which could have been used to build a better model that was adversarially trained on (M-)JSMA.

Overall, the effectiveness of these attacks is very worrying. It has become very clear how powerful attacks on machine learning models have become, and for this to not become a large issue in the nearby future, strong defensive measures are required.

Pixel-based attack & Basic iterative methods (Implemented by Thijs)

The following corresponds to the notebook `pixel_attack.ipynb` *located in* `/code/pixel_attack/`.
In [5] a method based on differential evolution [4] is used to find adversarial examples where only a fixed number (typically one or two) of pixels are changed. Unable to reproduce this exact method, we simply applied an exhaustive search to find a first and second pixel to change with maximum effect in the correct class loss. The results of this approach were underwhelming.

The following corresponds to the notebooks `basic_iterative.ipynb` *and* `basic_iterative_2.ipynb` *located in* `/code/pixel_attack/`.
In the first notebook, a basic iterative method is used to generate adversary images for a ResNet model classifying the ImageNet data set. It is shown that creating such examples is easy and undetectable. We then show that by adding random noise to the adversary example, the problem can be mitigated.
In the second notebook we train a ResNet model from scratch to classify images from the CIFAR-10 data set. We then create an adversary example using FGSM. We see again that it is easy to create an adversary example that is not recognizable as such. We now use adversarial regularization [1] however to train the model specifically against such attacks.

Discussion

The experiments show that creating adversary images that look exactly like the original is very easy. Training a machine learning model to mitigate this thus becomes interesting and possibly important.

Conclusion

From the results of our implementations, it becomes clear that neural networks still have significant security issues, that could be easily misused if they are not taken into account. With this project, we hope to raise awareness on how easy it is to generate adversarial examples. Furthermore, we provide insight in several mechanisms to defend against these attacks.

Bibliography

[1] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. URL: http://arxiv.org/abs/1412.6572, arXiv:1412.6572.

[2] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. URL: http://arxiv.org/abs/1611.01236, arXiv:1611.01236.

[3] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. 2015. URL: https://arxiv.org/abs/1511.07528, arXiv:1511.07528.

[4] Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, Dec 1997. doi:10.1023/A:1008202821328.

[5] Jiawei Su, Danilo Vasconcellos Vargas, and Sakurai Kouichi. One pixel attack for fooling deep neural networks. 23(5):828–841. URL: http://arxiv.org/abs/1710.08864, arXiv:1710.08864, doi:10.1109/TEVC.2019.2890858.

[6] Rey Wiyatno and Anqi Xu. Maximal jacobian-based saliency map attack. URL: http://arxiv.org/abs/1808.07945, arXiv:1808.07945.