




Is attacking machine learning easier than defending it?

Feb 15, 2017

by Ian Goodfellow and Nicolas Papernot

In our [first post](#), we presented a few ways that an attacker can break current machine learning systems, such as by poisoning the data used by the learning algorithm [BNL12], or crafting adversarial examples to directly force models to make erroneous predictions [SZS13]. In this post, we'll take adversarial examples as an illustration of why attacking machine learning seems easier than defending it. In other words, we cover in detail some of the reasons why we do not yet have completely effective defenses against adversarial examples, and we speculate about whether we can ever expect such a defense.

An adversarial example is an input to a machine learning model that is intentionally designed by an attacker to fool the model into producing an incorrect output. For example, we might start with an image of a panda and add a small perturbation that has been calculated to make the image be recognized as a gibbon with high confidence [GSS14]:

| | | | | |
|---|-----------------|---|-----|---|
|  | $+ .007 \times$ |  | $=$ |  |
| x | | $\text{sign}(\nabla_x J(\theta, x, y))$ | | $x + \epsilon \text{sign}(\nabla_x J(\theta, x, y))$ |
| “panda” | | “nematode” | | “gibbon” |
| 57.7% confidence | | 8.2% confidence | | 99.3 % confidence |

So far, it is much easier to design tricks like this that fool a model than it is to design models that can't be fooled.

How have we tried to make ML models more robust to adversarial examples?

Let's take a look at two defense techniques, adversarial training and defensive distillation, as examples of how a defender can attempt to make a machine learning model more robust and mitigate adversarial examples.

Adversarial training seeks to improve the generalization of a model when presented with adversarial examples at test time by proactively generating adversarial examples as part of the training procedure. This idea was first introduced by Szegedy et al. [SZS13] but was not yet practical because of the high computation cost of generating adversarial examples. Goodfellow et al. showed how to generate adversarial examples inexpensively with the fast gradient sign method and made it computationally efficient to generate large batches of adversarial examples during the training process [GSS14]. The model is then trained to assign the same label to the adversarial example as to the original example—for example, we might take a picture of a cat, and adversarially perturb it to fool the model into thinking it is a vulture, then tell the model it should learn that this picture is still a cat. An open-source implementation of adversarial training is available in the [cleverhans](#) library and its use illustrated in the following [tutorial](#).

Defensive distillation smooths the model's decision surface in adversarial directions exploited by the adversary. Distillation is a training procedure where one model is trained to predict the probabilities output by another model that was trained earlier. Distillation was first introduced by Hinton et al. in [HVD15], where the goal was for a small model to mimic a large, computationally expensive model. [Defensive distillation](#) has a different goal of simply making the final model's responses more smooth, so it works even if both models are the same size. It may seem counterintuitive to train one model to predict the output of another model that has the same architecture. The reason it works is that the first model is trained with “hard” labels (100% probability that an image is a dog rather than a cat) and then provides “soft” labels (95% probability that an image is a dog rather than a cat) used to train the second model. The second *distilled* model is more robust to attacks such as the fast gradient sign method [PM16] or the Jacobian-based saliency map approach [PMW16]. Implementations of these two attacks are also available on [cleverhans](#), respectively [here](#) and [here](#).

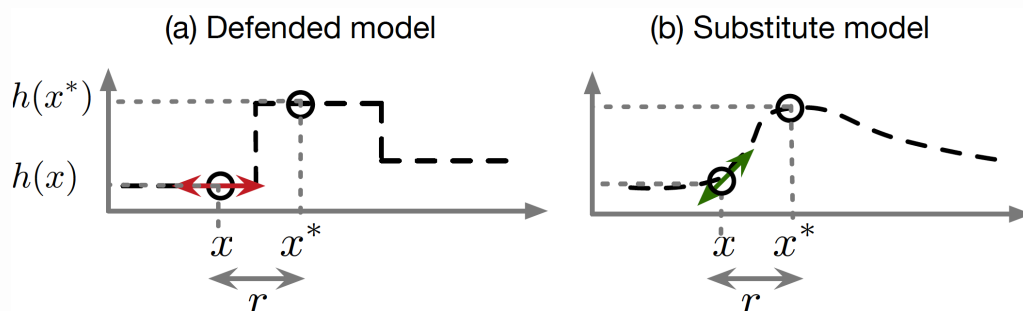
A failed defense: “gradient masking”

Most adversarial example construction techniques use the gradient of the model to make an attack. In other words, they look at a picture of an airplane, they test which direction in picture space makes the probability of the “cat” class increase, and then they give a little push (in other words, they perturb the input) in that direction. The new, modified image is mis-recognized as a cat.

But what if there were no gradient—what if an infinitesimal modification to the image caused no change in the output of the model? This seems to provide some defense because the attacker does not know which way to “push” the image.

We can easily imagine some very trivial ways to get rid of the gradient. For example, most image classification models can be run in two modes: one mode where they output just the identity of the most likely class, and one mode where they output probabilities. If the model's output is "99.9% airplane, 0.1% cat", then a little tiny change to the input gives a little tiny change to the output, and the gradient tells us which changes will increase the probability of the "cat" class. If we run the model in a mode where the output is just "airplane", then a little tiny change to the input will not change the output at all, and the gradient does not tell us anything. Let's run a thought experiment to see how well we could defend our model against adversarial examples by running it in "most likely class" mode instead of "probability mode." The attacker no longer knows where to go to find inputs that will be classified as cats, so we might have some defense. Unfortunately, every image that was classified as a cat before is still classified as a cat now. If the attacker can guess which points are adversarial examples, those points will still be misclassified. We haven't made the model more robust; we have just given the attacker fewer clues to figure out where the holes in the model's defense are. Even more unfortunately, it turns out that the attacker has a very good strategy for guessing where the holes in the defense are. The attacker can train their own model, a smooth model that has a gradient, make adversarial examples for their model, and then deploy those adversarial examples against our non-smooth model. Very often, our model will misclassify these examples too. In the end, our thought experiment reveals that hiding the gradient didn't get us anywhere.

Hence, we call this flawed defense strategy **gradient masking**, a term introduced in [PMG16]. The defense strategies that perform gradient masking typically result in a model that is very smooth in specific directions and neighborhoods of training points, which makes it harder for the adversary to find gradients indicating good candidate directions to perturb the input in a damaging way for the model. However, the adversary can train a *substitute* model: a copy that imitates the defended model by observing the labels that the defended model assigns to inputs chosen carefully by the adversary. A procedure for performing such a model extraction attack was introduced in [PMG16]. The adversary can then use the substitute model's gradients to find adversarial examples that are misclassified by the defended model as well. In the figure below, reproduced from the discussion of gradient masking found in [PMS16], we illustrate this attack strategy with a one-dimensional ML problem. The gradient masking phenomenon would be exacerbated for higher dimensionality problems, but harder to depict.



Surprisingly, we find that both adversarial training and defensive distillation accidentally perform a kind of gradient masking. If we transfer adversarial examples from one model to another model trained with one of these defenses, the attack often succeeds, even when a direct attack on the second model would fail [PMG16]. This suggests that both training techniques do more to flatten out the model and remove the gradient than to make sure it classifies more points correctly.

Playing a game of “whack-a-mole”

In the game of “hide the gradient,” we saw that gradient masking was not a very good defense. It defends against an attacker who uses the gradient, but if the attacker knows we are using that defense, they can just switch to a transferability attack. In security terminology, this means that gradient masking is not an **adaptive** defense.

Most defenses against adversarial examples that have been proposed so far just do not work very well at all, but the ones that do work are not adaptive. This means it is like they are playing a game of whack-a-mole: they close some vulnerabilities, but leave others open.

Adversarial training requires choosing an algorithm to generate adversarial examples. Usually, the model is trained to resist cheap adversarial examples that are generated in a single step, like with the fast gradient sign method. After training to resist these cheap adversarial examples, the model is usually successfully able to resist new instances of the same kind of cheap adversarial example. If we then use expensive, iterative adversarial examples, like those in [SZS13], the model is usually fooled.

Guaranteeing adaptiveness is challenging. Inspiration may be taken from the framework of [differential privacy](#), which offers a formal guarantee that a randomized algorithm does not expose individual users’ privacy. This guarantee holds without making assumptions about the adversary’s knowledge or capabilities, and as such holds in the face of future hypothetical attacks engineered by adversaries.

Why is it hard to defend against adversarial examples?

Adversarial examples are hard to defend against because it is hard to construct a theoretical model of the adversarial example crafting process. Adversarial examples are solutions to an optimization problem that is non-linear and non-convex for many ML models, including neural networks. Because we don’t have good theoretical tools for describing the solutions to these complicated optimization problems, it is very hard to make any kind of theoretical argument that a defense will rule out a set of adversarial examples.

From another point of view, adversarial examples are hard to defend against because they require machine learning models to produce good outputs *for every possible input*. Most of the

time, machine learning models work very well but only work on a very small amount of all the many possible inputs they might encounter.

Because of the massive amount of possible inputs, it is very hard to design a defense that is truly adaptive.

Other attack and defense scenarios

Several other kinds of attacks against machine learning are difficult to defend against. In this post, we have focused on test-time inputs intended to confuse a machine learning model, but many other kinds of attacks are possible, such as attacks based on surreptitiously modifying the training data to cause the model to learn to behave the way the attacker wishes it to behave.

One bright spot in adversarial machine learning is differential privacy, where we actually have theoretical arguments that certain training algorithms can prevent attackers from recovering sensitive information about the training set from a trained model.

It is interesting to compare machine learning to other scenarios where attacks and defenses are both possible.

In cryptography, the defender seems to have the advantage. Given a set of reasonable assumptions, such as that the cryptographic algorithm is implemented correctly, the defender can reliably send a message that the attacker cannot decrypt.

In physical conflict, attackers seem to have the advantage. It is much easier to build a nuclear bomb than to build a city that is able to withstand a nuclear explosion. The second law of thermodynamics seems to imply that, if defending requires maintaining entropy below some threshold, then the defender must eventually lose as entropy increases over time, even if there is no explicit adversary intentionally causing this increase in entropy.

The no free lunch theorem for supervised learning [W96] says that, averaged over all possible datasets, no machine learning algorithm does better on new points at test time than any other algorithm. At first glance, this seems to suggest that all algorithms are equally vulnerable to adversarial examples. However, the no free lunch theorem applies only when we make no assumption about the structure of the problem. When we study adversarial examples, we assume that small perturbations of the input should not change the output class, so the no free lunch theorem in its typical form does not apply.

Formally exposing the tension between robustness to adversaries and model performance on clean data remains an active research question. In [PMS16], a first no free lunch theorem for adversarial examples in machine learning shows that such a tension exists when learning from limited data. The result shows that defenders can thwart adversarial examples by moving to

richer hypothesis classes. However, the tension stems from challenges faced when the appropriate data and learning algorithms are not available to learn models with high fidelity.

Conclusion

The study of adversarial examples is exciting because many of the most important problems remain open, both in terms of theory and in terms of applications. On the theoretical side, no one yet knows whether defending against adversarial examples is a theoretically hopeless endeavor (like trying to find a universal machine learning algorithm) or if an optimal strategy would give the defender the upper ground (like in cryptography and differential privacy). On the applied side, no one has yet designed a truly powerful defense algorithm that can resist a wide variety of adversarial example attack algorithms. We hope our readers will be inspired to solve some of these problems.

References

- [BNL12] Biggio, B., Nelson, B., & Laskov, P. (2012). Poisoning attacks against support vector machines. arXiv preprint arXiv:1206.6389.
- [GSS14] Goodfellow, I. J., Shlens, J., & Szegedy, C. (2014). Explaining and harnessing adversarial examples. arXiv preprint arXiv:1412.6572.
- [HVD15] Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean. "Distilling the knowledge in a neural network." arXiv preprint arXiv:1503.02531 (2015).
- [PM16] Papernot, N., & McDaniel, P. (2016). On the effectiveness of defensive distillation. arXiv preprint arXiv:1607.05113.
- [PMG16] Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Berkay Celik, Z., & Swami, A. (2016). Practical Black-Box Attacks against Deep Learning Systems using Adversarial Examples. arXiv preprint arXiv:1602.02697.
- [PMS16] Papernot, N., McDaniel, P., Sinha, A., & Wellman, M. (2016). Towards the Science of Security and Privacy in Machine Learning. arXiv preprint arXiv:1611.03814.
- [PMW16] Papernot, N., McDaniel, P., Wu, X., Jha, S., & Swami, A. (2016, May). Distillation as a defense to adversarial perturbations against deep neural networks. In the 2016 IEEE Symposium on Security and Privacy (pp. 582-597).
- [SZS13] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. (2013). Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199.
- [W96] Wolpert, David H. (1996). The lack of a *a priori* distinction between learning algorithms. *Neural Computation*

cleverhans-blog

cleverhans-blog

Jekyll blog associated with cleverhans