6.883 Science of Deep Learning – Spring 2018 March 19, 2018

Lecture 12: Adversarial Examples and Misclassification Attacks II

Lecturer: Aleksander Mądry *Scribe: Lily Weng, Chi Heem Wong, Oscar Moll*
(Revised by Andrew Ilyas and Dimitris Tsipras—with thanks to Kai Xiao)

In the last lecture, we started our unit on robustness and security in deep learning, and introduced *adversarial examples*, slightly perturbed inputs to machine learning models that induce specific types of misbehaviour. Specifically, we talked about constructing adversarial examples by using Projected Gradient Descent (PGD) to solve the maximization problem $\max_{\delta \in \Delta} \text{loss}(\theta, x + \delta, y)$, where Δ is an allowable class of perturbations.

In this lecture, we will talk about (a) how to still find adversarial examples under more restrictive settings and (b) how to train and *formally verify* the robustness of deep learning models.

1 Black-box adversarial attacks

In the last lecture, when discussing how to construct adversarial examples (i.e. solve the "inner maximization" problem), we assumed a *white-box* setting, where the attacker has full access to the architecture and weights of the model in question. Having access to architecture and weights implies having *first-order* (gradient) access to the function being maximized ($\text{loss}(\theta)$), which is what allowed us to use the projected gradient descent (PGD) approach outlined in the last lecture.

In real-world settings, however, attackers rarely have full access to the model they are attacking. Instead, attackers will have varying levels of *black-box* access to the models they are crafting examples for. In the black-box setting, we no longer have access to $\nabla_\theta \text{loss}(\theta)$, and instead have access to some subset of the classification results like logit values, confidence scores or even only hard labels. Finding adversarial examples under this so-called "black-box setting" is more difficult as we don't have direct access to the model's parameters and gradients (and so we can't use projected gradient descent). Thus, the central question we will try to answer in the first half of this class is:

Can we (efficiently) solve the inner maximization problem in the black-box setting?

Here, we will discuss the two main approaches proposed in the literature: (1) substitute model attacks and (2) gradient estimation attacks.

1.1 Substitute model (transfer) attacks

Substitute model attacks, otherwise known as transfer attacks, exploit a curious property of adversarial examples known as *transferability*. This property refers to the phenomenon by which adversarial examples generated for one deep neural network tend to also induce misclassification in other, independently trained networks on the same dataset [7]. While theoretical explanation for transferability is still lacking, attackers can use this property to their advantage in constructing black-box attacks.

In particular, rather than applying an iterative update such as PGD, the attacker queries the model and, on observing the output, trains an auxiliary *substitute model* whose only objective is to mimic the target model. The attacker can then construct adversarial examples with white-box techniques on the constructed substitute network. Due to transferability, many such examples will also be adversarial for the target model.

Transfer attacks seem to work well empirically across various choices of models and training algorithms [7, 6]. While this method does not find adversarial examples as easily as white-box methods, it presents a very realistic threat, given how little information about the target model is required. The two key drawbacks of substitute model attacks are:

1. **Targeted attacks**: adversarial transferability seems to be limited to misclassification. That is, suppose we have an image-label pair (x, y), and then we construct an adversarial example x' for a model M that (mis)classifies as y'. If we feed in x' to a model M', transferability tells us that $M(x') \neq y$ with decent probability, but very rarely does $M(x') = y'$; if adversarial examples transfer, they usually end up misclassified as some unrelated class. This makes *targeted attacks*, where an attacker wants to induce a specific adversarial mislabeling, somewhat infeasible with substitute models.

2. **Training set**: In order to train the substitute model, the attacker needs many points drawn from the same distribution as the original classifier was trained on. If the adversary has access to the training set, this is not an issue, but otherwise acquiring these points can be somewhat expensive or impossible.

1.2 Gradient estimation attacks

What if instead of building a substitute model, we want to apply an iterative method on the target model directly? One might expect that PGD is not useful in black-box settings. It turns out, however, that this intuition is incorrect.

Specifically, we can still *estimate* the gradient using only such value queries. (In fact, this kind of estimation is the backbone of so-called zeroth-order optimization frameworks.) The fundamental tool of this approach is the *finite difference method*, which allows us to estimate the *directional* derivative $D_v f(x) = \langle \nabla_x f(x), v \rangle$ of some function f at a point x in the direction of a vector v as

$$D_v f(x) = \langle \nabla_x f(x), v \rangle \approx \left(f(x + \delta v) - f(x) \right) / \delta. \tag{1}$$

Here, the step size $\delta > 0$ governs the quality of the gradient estimate. Smaller δ gives more accurate estimates but also decreases reliability, due to precision and noise issues.

Now, we can just use finite differences to construct an estimate of the gradient of $\text{loss}(\theta, x, y)$ with respect to x. In particular, we can find the d components of the gradient by estimating the directional derivative with respect to the standard basis vectors $e_1, \ldots, e_d$ (where $e_i = \{0, 0, \ldots, 1, 0, \ldots, 0\}$):

$$\begin{aligned} \widehat{\nabla}_x \text{loss}(\theta, x, y) &= \sum_{k=1}^{d} e_k \left(\text{loss}(\theta, x + \delta e_k, y) - \text{loss}(\theta, x, y) \right) / \delta \\ &\approx \sum_{k=1}^{d} e_k \langle \nabla_x \text{loss}(\theta, x, y), e_k \rangle \end{aligned} \tag{2}$$

We can then plug this coordinate-wise gradient estimate straight into the PGD proceduce we discussed last time, in place of the gradient. [2] introduce an attack called ZOO-attack (zeroth-order optimization attack), the first attack to use the finite difference method in this basic form to power PGD-based adversarial attacks in the black-box setting.

Chen et al. showed that ZOO creates adversarial examples with close to 100% success rate [2]. Despite its success, ZOO is considered impractical due to its high query complexity. In general, ZOO requires $\Theta(d)$ queries for each gradient computation, where d is the dimension of image vector. For ImageNet, $d \sim 300k$, making the cost of attacks prohibitively high. One can easily thwart such an attack by banning users making a huge number of queries within a short time interval.

This naturally leads to the following question: *Can we reduce the query complexity?* Below are several recent works that contribute to decrease the number of queries.

First: a note on more restricted settings. Note that in order to compute the gradient estimate (2), the attacker needs query access to $\text{loss}(\theta, x, y)$, which in turn implies having access to all of the output probabilities given by the model, i.e. $P(y|x)$ for all possible y. While this can be a rather strong assumption, [3] show that surrogate loss functions can be constructed to make gradient estimation attacks work in settings where even less information than $\text{loss}(\theta, x, y)$ is given (in particular, gradient estimation approaches can still be used even if only the top label is given).

Changing to a random basis [3]. While [2] opt to use the canonical (i.e. pixel) basis in constructing a gradient estimate, there is nothing in the finite difference framework that forces us to do this. Specifically, (2) actually holds true for any basis of vectors $\{e_i\}$.

Ilyas et al. [3] use a set of random Gaussian vectors around an image, x to construct the gradient estimates, but recover only a few components of the gradient with finite differences. Note that although Gaussians do not strictly form a basis, in high dimensions, a few randomly drawn Gaussian vectors tend to be pairwise near-orthogonal. Furthermore, the method benefits from favorable properties of random Gaussian projections, like relative distance preservation[1]. The authors find that using this random basis method[2] led to much more query-efficient black-box adversarial attacks.

Compressive sensing. An alternative but related view of black-box gradient estimation attacks is as solving a system of linear equations. In particular, given any unit vector $\boldsymbol{v}$, we can query $\langle \nabla f, \boldsymbol{v} \rangle = u$. We can write this as a system of equations by making k queries, and stacking $\{\boldsymbol{v_1} \dots \boldsymbol{v_k}\}$ into a matrix A, yielding the system of equations $Aw = u$. Solving this system for w yields precisely the gradient $\nabla f(x)$. While normally we need $k \geq d$ in order to recover meaningful results, a seminal result from Candes, Romberg, and Tao [1] shows that if the gradient is *sparse* under some basis (i.e. if there exists a basis where most of the gradient components are zero), we can *provably* recover the gradient with $k << d$ queries. Concretely, compressive sensing theory shows that if we can assume our data to be s-sparse (i.e. under the chosen basis all but s components are zero), then we require $k \approx s \log \frac{d}{s}$. This would form a very good basis for our images. While it is known that there are sparse bases such as this one for images, it is unclear in our problem if there is a basis under which $\nabla f(x)$ is sparse.

Gradient estimation with priors (and online optimization) [4]. It turns out that finite differences is actually, in a natural sense, the best we can do in terms of asymptotic performance at zeroth-order optimization [4]. So what now? Ilyas et al. (2018b) [4] show that we can actually greatly improve gradient estimation by integrating what the authors call "gradient priors," external information that can help in estimation of the gradient. For example, we know that we are going to use these gradients for (iterative) PGD, and since we don't move much in image space during an adversarial attack, it's likely that successive iterations in optimization will have similar gradients. Ilyas et al. develop a framework inspired by *bandit online optimization* to integrate this prior and others into black-box attacks, resulting in attacks that are several-times more query efficient and successful.

1.3 Verification of deep learning models

At this point in the lecture, we are going to shift our focus from *attacking* DNN models to *defending* against such attacks. Specifically, today we study the problem of *verifying* the robustness of neural network models.

The problem of verification is the following. We are given an input x, a classifier (which we can represent as a function $f(x \in \mathcal{X}) \to \mathcal{Y}$ mapping from images to labels), and a perturbation magnitude ϵ. We would like to use this data to produce a formal certificate either *proving* that $f(x) = f(x')$, $\forall \|x' - x\|_\infty \leq \epsilon$, or finding an x' which "disproves" the statement.

The set of all final-layer activations attained by perturbing x with some Δ with l_∞ norm less than ϵ is called the adversarial polytope. There are several approaches in the literature:

1. **Formal verification methods (e.g. ReLUplex [5]).** First, note that the main bottleneck of directly verifying deep neural networks are the ReLUs, which convert the problem from verifying a linear function to verifying a *piecewise linear function*, leading to a combinatorial blowup in the problem complexity (in particular, each on/off configuration of the ReLUs leads to another subproblem).

   ReLUplex [5] attacks this problem by explicitly splitting up the verification problem into appropriate subproblems via SMT solver. Interested readers are encouraged to consult the appropriate

[1] http://pages.cs.wisc.edu/~jerryzhu/cs731/projection.pdf

[2] Finite differences under the Gaussian basis is actually a fundamental technique found across a variety of fields, and is known by a variety of names (spherical estimator, Johnson-Lindenstrauss estimator, random basis search, etc.)

literature, but the main idea is that the authors extend the real numbers $\mathbb{R}$ to an arithmetic over real numbers and ReLUs ($\mathbb{R}R$). This allows for the application of a Satisfiability Modulo Theories (SMT) solver—an SMT solver is essentially an efficient way of solving decision problems involving logical formulas; it can be thought of as a sort of formalized approach to constraint programming. In ReLUplex, an SMT solver considers all possible settings of ReLUs (active/inactive) and uses linear programming to decide (i) if a possible setting is feasible, and, (ii) whether there exists an example that fools the network. However, it can currently only deal with $\sim$ 300-neuron networks.

2. **Mixed-Integer Optimization [8].** Another technique for verifying neural networks in the presence of ReLUs involves the utilization of *Mixed-Integer Linear Programs* (MILPs/MIPs). MILPs are a generalization of linear programs (LPs), where some of the variables are constrained to be integers. In [8], the authors phrase verification as an MILP problem, by encoding the on/off status of each ReLU with a binary (0 or 1) variable. By taking advantage of the ability to constrain *some* variables (in particular the ReLUs) to be either 0 or 1, it is actually possible to write the entire verification process as an MILP.

   In contrast to linear programming (LP), however, which can be solved efficiently even in the worst case, even binary integer programming is NP-hard—in fact, the decision version was one of Karp's 21 NP-complete problems. Since a binary integer program is actually a special case of the verification MILP above (i.e. where we only have ReLUs), the verification problem is thus also NP-hard under this formulation. Despite this, performant solvers for MILPs have been developed, including Gurobi, CPLEX, etc. Utilizing these solvers allows for MILP verification to scale better than ReLUplex, allowing for the verification of neural networks with $\sim$ 1000 neurons.

   In later work, it was shown that further optimization of the MILP formulation itself, e.g. by removing integer variables corresponding to "stable" ReLUs (either always active or always inactive within the region around the image) allows for verification that is orders of magnitude faster [8]. Exploiting this further, it is shown in [10] that this stability can actually be *induced* via regularization techniques during training. Indeed, training neural networks with ease of verification as an objective now allows for networks of up to $\sim 50,000$ neurons to be verified [10].

3. **Convex (Linear) relaxation [9].** In mathematics, the relaxation of a (mixed) integer linear program is the problem that arises by removing integrality constraints on variables. In [9], the authors use a convex relaxation of the ReLU function (see Figure 1) to obtain an LP relaxation of the MILP formulation in [8]. Specifically, for a ReLU with bounded pre-activation $l \leq \hat{z} \leq u$, the ReLU constraint ($z = \max(0, \hat{z})$) can be written as a mixture of linear and integer constraints (as in an MILP). Relaxing the integer constraints yields the convex region $z \geq 0; z \geq \hat{z}; -u\hat{z} + (u-l)z \leq -u\hat{z}$, depicted in Figure 1. By relaxing every ReLU in this way, one effectively constructs a convex outer bound on the adversarial polytope (Figure 2. Finding an adversarial example in this convex outer bound can be achieved by simply solving the LP given by minimizing the output value of the correct class minus the output value of an incorrect class. If no point in this outer approximation exists that will change the class prediction of an example, then no point within the true adversarial polytope can change its prediction either, i.e., the point is provably robust to all adversarial attacks.

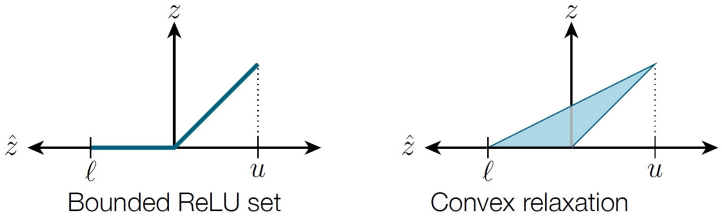

Figure 1: Relaxation of the ReLU activation [9].

While a linear program is certainly more desirable than an MILP, solving an LP in many variables can still be slow. To circumvent this, [9] exploit the *duality* of linear programs. In particular, every

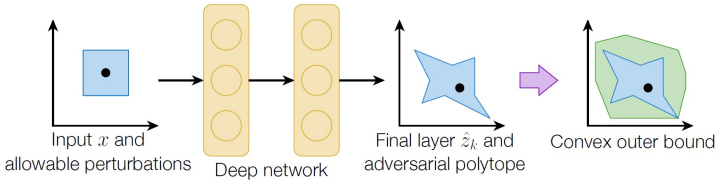

Figure 2: Illustration of adversarial polytope and its convex outer bound [9].

LP minimization has a corresponding (dual) *maximization* LP with the property that: (a) the maximum of the dual LP is the minimum of the original (primal) LP, and (b) any *feasible* solution to the dual LP provides a lower bound on the minimum of the primal LP. These properties, and in particular (b), mean that instead of trying to optimize an LP directly, one can actually bound its optimum value using any feasible solution of the dual LP.

To this end, the authors of [9] consider the *dual* of the convex relaxation LP. Recall that the objective value of any solution to the dual problem provides a lower bound to any objective value of the primal (minimization) problem. Surprisingly, [9] shows that the dual formulation of a feedforward neural network is almost identical to the backpropagation network, except for some free parameters $\alpha_{i,j}$ that can be optimized over. In practice, setting the $\alpha_{i,j}$ to fixed feasible values gives a valid solution that can be computed quickly and already provides a good lower bound on the primal objective value. Thus, this dual formulation can be used to provide a lower bound on the adversarial error. Even though this bound is not tight, it can be computed much more easily than exact verification procedures that solve mixed integer linear programs. However, a major drawback of this method is that it might label examples as potentially adversarial, even when they are not.

References

[1] Emmanuel J Candes, Justin K Romberg, and Terence Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 59(8):1207–1223, 2006.

[2] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 15–26. ACM, 2017.

[3] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. *International Conference on Machine Learning (ICML) 2018*, 2017.

[4] Andrew Ilyas, Logan Engstrom, and Aleksander Madry. Prior convictions: Black-box adversarial attacks with bandits and priors. In *International Conference on Learning Representations (ICLR)*, 2018.

[5] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[6] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. *CoRR*, abs/1610.08401, 2016.

[7] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013.

[8] Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations (ICLR)*, 2019.

[9] Eric Wong and J Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning (ICML)*, 2018.

[10] Kai Y. Xiao, Vincent Tjeng, Nur Muhammad Shafiullah, and Aleksander Madry. Training for faster adversarial robustness verification via inducing ReLU stability. In *International Conference on Learning Representations (ICLR)*, 2019.