Adversarial Robustness

Rojina Kashefi

Neural Network

1. Model: (K is the output dimensionality)

$$h_\theta : \mathcal{X} \to \mathbb{R}^k$$

2. Loss Function is a mapping from the model predictions and true labels to a non-negative number.

$$\mathbb{R}^k \times \mathbb{Z}_+ \to \mathbb{R}_+$$

3. Cross entropy Loss :

$$\ell(h_\theta(x), y) = \log\left(\sum_{j=1}^{k} \exp(h_\theta(x)_j)\right) - h_\theta(x)_y$$

where $h_\theta(x)_j$ denotes the jth elements of the vector $h_\theta(x)$

4. Prob = e ^ -loss
5. In basic neural network

$$\underset{\theta}{\text{minimize}} \frac{1}{m} \sum_{i=1}^{m} \ell(h_\theta(x_i), y_i)$$

$$\theta := \theta - \frac{\alpha}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla_\theta \ell(h_\theta(x_i), y_i)$$

Adversary Network

1. $\underset{\hat{x}}{\text{maximize}}\, \ell(h_\theta(\hat{x}), y)$

2. If we change the image entirely, say to a dog, then it's not particularly impressive that we can "fool" the classifier into thinking it's not a pig.

3. We would like Δ to capture anything that humans visually feel to be the "same" as the original input x.

$$\underset{\delta \in \Delta}{\text{maximize}}\, \ell(h_\theta(x + \delta), y)$$

4. Give a mathematically rigorous definition of all the perturbations that *should* be allowed.

$$\Delta = \{\delta : \|\delta\|_\infty \leq \epsilon\}$$

allow the perturbation to have magnitude between $[-\epsilon, \epsilon]$

Adversary in Images

1. Need to ensure that $x+\delta$ is also bounded between $[0,1]$ so that it is still a valid image.

2. Projected Gradient Descent (PGD) : simply clipping the values that exceed ϵ magnitude to $\pm\epsilon$.

Targeted Attacks

Instead of trying to just maximize the loss of the correct class, we maximize the loss of the correct class while also minimizing the loss of the target class.

```
loss = (-nn.CrossEntropyLoss()(pred, torch.LongTensor([341])) +
        nn.CrossEntropyLoss()(pred, torch.LongTensor([404])))
```

Traditional Risk

1. The risk of a classifier is it's expected loss under the true distribution of samples, i.e.

$$R(h_\theta) = \mathbf{E}_{(x,y)\sim\mathcal{D}}[\ell(h_\theta(x)), y)]$$

2. we do not know the underlying distribution of the actual data, so we approximate this quantity by considering a finite set of samples drawn draw D.

$$\hat{R}(h_\theta, D) = \frac{1}{|D|} \sum_{(x,y)\in D} \ell(h_\theta(x)), y).$$

3. Empirical risk

$$D = \{(x_i, y_i) \sim \mathcal{D}\}, i = 1, \ldots, m$$

4. The traditional process of training a machine learning algorithm is that of finding parameters that minimize the empirical risk on some training set denoted D_{train}.

$$\underset{\theta}{\text{minimize}}\, \hat{R}(h_\theta, D_{\text{train}}).$$

Adversarial Risk

This is like the traditional risk, except that instead of suffering the loss on each sample point $\ell(h_\theta(x),y)$ we suffer the *worst case* loss in some region around the sample point.

$$R_{\text{adv}}(h_\theta) = \mathbf{E}_{(x,y)\sim\mathcal{D}}\left[\max_{\delta\in\Delta(x)} \ell(h_\theta(x+\delta)), y)\right]$$

$$\hat{R}_{\text{adv}}(h_\theta, D) = \frac{1}{|D|}\sum_{(x,y)\in D} \max_{\delta\in\Delta(x)} \ell(h_\theta(x+\delta)), y).$$

Why might we prefer to use the Adversarial Risk instead of the Traditional Risk?

1. If we are truly operating in an adversarial environment, where an adversary is capable of manipulating the input with full knowledge of the classifier, then this would provide a **more accurate estimate** of the expected performance of a classifier.

2. It is very difficult to actually draw samples d, from the true underlying distribution.

Training Adversarially Robust Classifiers

1. Training a classifier that is robust to adversarial attacks (or equivalently, one that minimizes the empirical adversarial risk).
2. ***min-max* or *robust optimization* formulation**

$$\underset{\theta}{\text{minimize}}\, \hat{R}_{\text{adv}}(h_\theta, D_{\text{train}}) \equiv \underset{\theta}{\text{minimize}}\, \frac{1}{|D_{\text{train}}|} \sum_{(x,y)\in D_{\text{train}}} \max_{\delta\in\Delta(x)} \ell(h_\theta(x+\delta)), y).$$

$$\theta := \theta - \frac{\alpha}{|B|} \sum_{(x,y)\in B} \nabla_\theta \max_{\delta\in\Delta(x)} \ell(h_\theta(x+\delta)), y).$$

$$\delta^\star = \underset{\delta\in\Delta(x)}{\text{argmax}}\, \ell(h_\theta(x+\delta)), y)$$

$$\nabla_\theta \max_{\delta\in\Delta(x)} \ell(h_\theta(x+\delta)), y) = \nabla_\theta \ell(h_\theta(x+\delta^\star)), y)$$

$$\theta := \theta - \frac{\alpha}{|B|} \sum_{(x,y)\in B} \nabla_\theta \ell(h_\theta(x+\delta^\star(x))), y).$$

Compute adversarial examples, and then update the classifier based upon these adversarial examples.

Challenges

1. We cannot solve the inner maximization problem optimally. If done via gradient descent, is a non-convex optimization problem.

2. Although in theory one can take just the worst-case perturbation as the point at which to compute the gradient, better to incorporate multiple perturbations with different random initializations and potentially also a gradient based upon the initial point with no perturbation.

Every adversarial attack and defense are a method for approximately solving the inner maximization and/or outer minimization problem respectively.

Linear Models

1. we can solve the inner **maximization exactly** for the case of **binary optimization**, and provide a relatively tight **upper bound** for the case of **multi-class classification.**

2. The resulting **minimization** problem is still convex in θ, the resulting robust training procedure can *also* be solved optimally, and thus we can achieve the globally optimal robust classifier.

Binary Classification

$$\frac{\exp(h_\theta(x)_1)}{\exp(h_\theta(x)_1) + \exp(h_\theta(x)_2)} = \frac{1}{1 + \exp(h_\theta(x)_2 - h_\theta(x)_1)}$$

$$\frac{\exp(h_\theta(x)_2)}{\exp(h_\theta(x)_1) + \exp(h_\theta(x)_2)} = \frac{1}{1 + \exp(h_\theta(x)_1 - h_\theta(x)_2)}.$$

Single scalar-valued hypothesis

$$h'_\theta(x) \equiv h_\theta(x)_1 - h_\theta(x)_2$$

$$p(y|x) = \frac{1}{1 + \exp(-y \cdot h'_\theta(x))}$$

$$p(y = +1|x) = \frac{1}{1 + \exp(-h_\theta(x))}.$$

$$-\log \frac{1}{1 + \exp(-y \cdot h'_\theta(x))} = \log(1 + \exp(-y \cdot h'_\theta(x)))$$

$$\ell(h_\theta(x), y) = \log(1 + \exp(-y \cdot h_\theta(x))) \equiv L(y \cdot h_\theta(x))$$

Solving Inner Maximization Problem

$$\underset{\|\delta\| \leq \epsilon}{\text{maximize}}\, \ell(w^T(x+\delta), y) \equiv \underset{\|\delta\| \leq \epsilon}{\text{maximize}}\, L(y \cdot (w^T(x+\delta)+b)).$$

If the function is monotonic decreasing, maximize this function applied to a scalar, that is equivalent to just minimizing the scalar quantity.

$$\begin{aligned}\max_{\|\delta\| \leq \epsilon} L\left(y \cdot (w^T(x+\delta)+b)\right) &= L\left(\min_{\|\delta\| \leq \epsilon} y \cdot (w^T(x+\delta)+b)\right) \\ &= L\left(y \cdot (w^T x + b) + \min_{\|\delta\| \leq \epsilon} y \cdot w^T \delta\right)\end{aligned}$$

We need to solve this:

$$\min_{\|\delta\| \leq \epsilon} y \cdot w^T \delta.$$

In general

$$\min_{\|\delta\| \leq \epsilon} y \cdot w^T \delta = -\epsilon \|w\|_*$$

where $\|\cdot\|_*$ denotes the the dual norm of our original norm bound on θ ($\|\cdot\|_p$ and $\|\cdot\|_q$ are dual norms for 1/p+1/q=1)

For minimization of y=1 : set $\delta_i = -\epsilon$ for $w_i \geq 0$ and $\delta_i = \epsilon$ for $w_i < 0$

$$\delta^\star = -y\epsilon \cdot \operatorname{sign}(w)$$

$$\min_{\|\delta\| \leq \epsilon} y \cdot w^T \delta. \quad = \quad y \cdot w^T \delta^\star = y \cdot \sum_{i=1} -y\epsilon \cdot \operatorname{sign}(w_i) w_i = -y^2 \epsilon \sum_i |w_i| = -\epsilon \|w\|_1.$$

$$\max_{\|\delta\|_\infty \leq \epsilon} L\left(y \cdot (w^T(x+\delta)+b)\right) = L\left(y \cdot (w^T x + b) - \epsilon \|w\|_1\right).$$

Now instead of having min-max problem we have minimization problem.

$$\underset{w,b}{\text{minimize}} \frac{1}{D} \sum_{(x,y)\in D} L\left(y \cdot (w^T x + b) - \epsilon \|w\|_1\right).$$

So in general we need to solve this problem

$$\text{minimize}_{w,b} \frac{1}{D} \sum_{(x,y)\in D} L(y \cdot (w^T x + b)) + \epsilon \|w\|_*$$

Multi-class Label Classification

In mulit-class label it turns out that it is no longer possible to optimally solve the inner maximization problem.

$$h_\theta(x) = Wx + b$$

$$\max_{\|\delta\| \leq \epsilon} \ell(W(x+\delta) + b, y).$$

$$\max_{\|\delta\| \leq \epsilon} \left(\log \left(\sum_{j=1}^{k} \exp(w_j^T (x+\delta) + b_i) \right) - (w_y^T (x+\delta) + b_y) \right).$$

We cannot push the max over δ inside the nonlinear function (the log-sum-exp function is convex, so maximizing over it is difficult in general).

MNIST problem

MNIST is actually a fairly poor choice of problem for many reasons:

a. Being very small for modern ML

b. It also has the property that it can easily be “binarized”, i.e., because the pixel values are essentially just black and white, we can remove more ℓ_∞ noise by just rounding to 0 or 1, and the classifying the resulting image.

Neural Network

$$
\begin{aligned}
z_1 &= x \\
z_{i+1} &= f_i(W_i z_i + b_i), \quad i, \ldots, d \\
h_\theta(x) &= z_{d+1}
\end{aligned}
$$

1. Z_i denote the activations at layer i
2. F_i denotes the activation function for layer i, which we will often take to be e.g. the ReLU operator $f_i(z)=\max\{0,z\}$.
3. Parameters of the network are given by $\theta=\{W_1,b_1,\ldots,W_d,b_d\}$ (in the above, W_i is most obviously interpreted a matrix, but it could really be any linear operator including convolutions).

Neural Network Loss Surface

1. Loss surface of standard neural networks is much more "**irregular**" than for linear models. Because neural networks have much more **modeling power** than linear models, they have the ability to have much bumpier function surface.

2. Loss surfaces lead to two main challenges.First, in the **high dimensional setting** (deep networks), there is a high likelihood that at almost **any point in input space** there will be *some* **direction** along the **loss surface** that is **very steep**, i.e., which causes the loss to either increase or decrease substantially.

3. The second challenge is that unlike the linear case, it is ***not* easy to solve the inner maximization** problem over our perturbation. This is because, **the cost surface** for neural networks is **not convex**, and is especially **prone to local optima.**

4. This second point may be somewhat less of an issue if our **goal is just to *construct* an adversarial example** against a standard deep network.After all, there are many directions of high cost increase, and just following the gradient typically leads to ***an* adversarial example,** even if it is **not the *optimal* adversarial example**. But when we later consider *training* robust networks, this will be a big problem.

Solving Inner Maximization

1. Lower bounds
2. Exact solutions
3. Upper bounds

Lower Bounds

1. We can find a *lower bound* on the optimization objective.

2. Because (by definition) ***any* feasible δ** will give us **a lower bound.**

3. This is equivalent to just “trying to empirically solve the optimization problem” or “find an adversarial example.”

4. This is by far the most common strategy for solving the inner maximization.

5. However, in order to both find strong adversarial examples *and*, use this approach to train robust classifiers, it is important that we solve this problem *well*.

The Fast Gradient Sign Method (FGSM)

Using backpropagation, we can compute the **gradient of the loss function with respect to the perturbation δ** itself, to maximize our objective. (adjust δ in the direction of it's gradient)

$$g := \nabla_\delta \ell(h_\theta(x+\delta), y)$$

If we're evaluating this gradient at δ=0 (as we would at the first timestep), then this is also just equal to $\nabla x \ell(h\theta(x),y)$.

$$\delta := \delta + \alpha g$$

Ensure that δ **stays within the norm bound ϵ**, so after each step, we can **project back into this space**.

$$\delta := \text{clip}(\alpha g, [-\epsilon, \epsilon]).$$

How big of a step size should we take?

If we want to make increase the loss as much as possible, it makes sense to take as large a step as possible take **α to be very large.**

For α large enough, the relative sizes of the entries of g won't matter: we will simply take δi to be either +ϵ or −ϵ depending upon the sign of gi.

$$\delta := \epsilon \cdot \text{sign}(g).$$

FGSM : one of the first methods for constructing adversarial examples

The Fast Gradient Sign Method (FGSM)

1. An attack under an ℓ_∞ norm bound.
2. A single projected gradient descent step under the ℓ_∞ constraint.
3. Evaluate FGSM in the context of other ℓ_∞ attacks. (easy to generalize this procedure to other norms)
4. *Exactly* the optimal attack against a *linear* binary classification model under the ℓ_∞norm.
5. Assumes that the linear approximation of the hypothesis given by its gradient at the point x is a reasonably good approximation to the function over the entire region.
6. Neural networks are *not* in fact linear even over a relatively small region, if we want a stronger attack we likely want to consider better methods at maximizing the loss function than a single projected gradient step.

Projected Gradient Descent

1. Just like FGSM but with a smaller step size.
2. This was also called the “basic iterative procedure”
3. The basic PGD algorithm simply reiterates the basic approach

Repeat:

$$\delta := \mathcal{P}(\delta + \alpha \nabla_\delta \ell(h_\theta(x+\delta), y))$$

4. where P denotes the projection onto the ball of interest (for example, clipping in the case of the ℓ_∞ norm).
5. More choices we need to make when specifying the attack, such as the actual step size itself, and the number of iterations.

Projected Gradient Descent Problem

1. First we take Extremely large step size α=1e4. This is all due to the fact that gradient with respect to δ and δ=0 is usually very small,so we need to scale it by a relatively large α to make any progress at all.

2. Once we "break out" of the initial region around δ=0, the gradients typically increase in magnitude substantially, and at this point our large α is too large, and the method takes too big a step toward the boundary.

3. If we want to minimize some function $f:R_n \rightarrow R$ over the input z, the traditional gradient descent algorithm repeats the update.

$$z := z - \alpha \nabla_z f(z).$$

4. The trouble with this update, is that it is highly sensitive to the absolute scale of the gradient and adjusts the parameters in a scale that always corresponds to the relative scaling of the gradient terms.

Steepest Descent

1. This is a completely standard optimization procedure.It varies slightly from the traditional gradient descent algorithm.

2. The normalized steepest descent method applies find some negative update direction v, where *we choose* v *to maximize the inner product between* v *and the gradient* subject to a norm constraint on v.

$$z := z - \underset{\|v\| \leq \alpha}{\operatorname{argmax}} v^T \nabla_z f(z).$$

3. If we use L2 norm :

$$\underset{\|v\|_2 \leq \alpha}{\operatorname{argmax}} v^T \nabla_z f(z) = \alpha \frac{\nabla_z f(z)}{\|\nabla_z f(z)\|_2}$$

4. If we use ℓ_∞ norm:

$$\underset{\|v\|_\infty \leq \alpha}{\operatorname{argmax}} v^T \nabla_z f(z) = \alpha \cdot \operatorname{sign}(\nabla_z f(z)).$$

5. Choose a steepest descent norm to match the norm that we are ultimately minimizing with respect to.

6. Choose step sizes than before. Since the step size α is on the same scale as the total perturbation bound, it makes sense to choose α to be some reasonably small fraction of ϵ, and then choose the number of iterations to be a small multiple of ϵ/α.

Randomization

1. The performance of PGD is still limited by the possibility of local optima within the objective.

2. Not possible to avoid the reality of local optima entirely, we can mitigate the problem slightly by random restarts.

3. we don’t just run PGD once, but we run it multiple times from different random locations within the ℓ_∞ ball of choice.

4. Challenges : Increases the runtime by a factor equal to the number of restarts.

Targeted Attacks

1. Maximizing the probability of target class and Minimizing the probability true label class

$$\underset{\|\delta\| \leq \epsilon}{\text{maximize}}(\ell(h_\theta(x+\delta), y) - \ell(h_\theta(x+\delta), y_{\text{targ}})) \equiv \underset{\|\delta\| \leq \epsilon}{\text{maximize}} \left(h_\theta(x+\delta)_{y_{\text{targ}}} - h_\theta(x+\delta)_y\right)$$

2. Maximizing the probability of target class and Minimizing probability of all other class

$$\underset{\|\delta\| \leq \epsilon}{\text{maximize}} \left(h_\theta(x+\delta)_{y_{\text{targ}}} - \sum_{y' \neq y_{\text{targ}}} h_\theta(x+\delta)_{y'}\right)$$

3. More difficult objective than the previous one, so we aren't able to fool the classifier as much. But when we *do* fool the classifier, it more consistently (even if still not perfectly) able to predict the target class.

Non-ℓ_∞ norms

1. Been focusing on attacks where δ has bounded ℓ_∞ norm (and additionally, where we don't worry about clipping to the actual allowable [0,1] range of the image.
2. Consider ℓ_2 and include the constraint that $x+\delta$ needs to actually lie in the [0,1] range.

$$\delta := \mathcal{P}_\epsilon \left(\delta - \alpha \frac{\nabla_\delta \ell(h_\theta(x+\delta), y)}{\|\nabla_\delta \ell(h_\theta(x+\delta), y)\|_2} \right)$$

3. P_ϵ now denotes the projection onto the ℓ_2 ball of radius ϵ. This projection in turn is just given by normalizing δ to have ℓ_2 norm ϵ if it is greater than ϵ, i.e

$$\mathcal{P}_\epsilon(z) = \epsilon \frac{z}{\max\{\epsilon, \|z\|_2\}}.$$

4. Note that the ϵ you need to consider for $\ell_2$norm perturbations is larger than what you need for ℓ_∞ perturbations, because the volume of the ℓ_2 ball is proportional to $\sqrt{n}$ times the volume of the ℓ_∞ ball, where nis the input dimension.

Non-ℓ_∞ norms

1. ℓ_∞ attacks lead to small noise everywhere in the image

2. ℓ_2 attacks lead to perturbations that are more localized in the image.

3. All the same considerations also apply to ℓ_1 attacks.

Exact Solutions

1. This is going to be a challenging problem but for **many networks activations** functions we can formulation the exact maximization problem as a ***combinatorial* optimization** problem, and solve it exactly using techniques such as **mixed integer programming**.

2. These methods will of course have **substantial challenges in scaling to large models**, but for **small problems they highlight an important point**, that it *is* possible to construct exact solutions to the inner maximization problem in some cases.

Exact Solutions

1. Actually don't exactly solve the inner maximization problem in the multiclass case, but we *can* determine exactly whether or not an adversarial example exists within a certain radius.

2. We can write the targeted attack as an optimization problem again, but this time, we are going to make the actual structure of the network explicit in the optimization problem.

3. Specifically, we are going to formulate the optimization problem as one over the input x *and* all the intermediate actions zi, but with constraints that constrain these activations to follow the behavior of the neural network. We'll also focus back on the case of $\ell\infty$ attacks.

$$\begin{aligned} \underset{z_{1,\ldots,d+1}}{\text{minimize}} \quad & (e_y - e_{y_{\text{targ}}})^T z_{d+1} \\ \text{subject to} \quad & \|z_1 - x\|_\infty \leq \epsilon \\ & z_{i+1} = \max\{0, W_i z_i + b_i\}, \quad i = 1, \ldots, d-1 \\ & z_{d+1} = W_d z_d + b_d \end{aligned}$$

where ei denotes the unit basis, i.e., a vector with a one in the ith position and zeros everywhere else; and where we removed the explicit δ term in favor of a constraint that simply requires z1(the input to the first layer), to be within ϵ of x.

A mixed integer programming formulation

1. The specific problem here is the equality constraint involving the max operator, which is not a convex constraint, nor is it one that is natively handled by most optimization solvers.
2. In order to solve the problem, then we need to convert it into an alternative form, specifically a (binary) mixed integer linear program (MILP). A binary MILP is an optimization problem that consists of a *linear* objective.
3. But MILPs in general are NP-hard, and so we don’t expect to be able to ever scale this approach to the size of modern neural networks.
4. However, for small problems, MILPs are an extremely well-studied area.

Linearization

How do we express the constraint $z_{i+1}=\max\{0, W_i z_i + b_i\}$ using linear constraints and binary integer constraint?

Suppose we have some known upper and lower bound for the values that $W_i z_i + b_i$ can take on, which we'll denote l_i and u_i respectively (these are fixed).

We will also introduce a set of binary variables v_i that is the same size as z_{i+1}.

$z_{i+1} = \max\{0, W_i z_i + b_i\}$:

$$
\begin{aligned}
z_{i+1} &\geq W_i z_i + b_i \\
z_{i+1} &\geq 0 \\
u_i \cdot v_i &\geq z_{i+1} \\
W_i z_i + b_i &\geq z_{i+1} + (1 - v_i) l_i \\
v_i &\in \{0, 1\}^{|v_i|}
\end{aligned}
$$

Proof next slide

First suppose that $W_i z_i + b_i > 0$ (for simplicity, you can think of these as just scalar values for this discussion, but of course it applies elementwise to the vector). If we choose $v_i = 0$, then the third constraint would imply that $z_{i+1} \leq 0$, while the first constraint would imply that $z_{i+1} \geq W_i z_i + b_i > 0$, which results in an infeasible solution; thus, for $W_i z_i + b_i > 0$, we *need* to choose $v_i = 1$. And if we do choose $v_i = 1$, then the constraints reduce to

$$\begin{aligned} z_{i+1} &\geq W_i z_i + b_i \\ z_{i+1} &\geq 0 \\ u_i &\geq z_{i+1} \\ W_i z_i + b_i &\geq z_{i+1}, \end{aligned}$$

the first and fourth inequalities together imply that $z_{i+1} = W_i z_i + b_i$, and the second and third inequalities are always satisifed respectively because 1) the first inequality is stricter than the second and 2) u_i is an upper bound on $W_i z_i + b_i$ so will also be greater than z_{i+1}.

Alternatively, suppose that $W_i z_i + b_i < 0$. Then choosing $v_{i+1} = 1$ causes the fourth constraint implies that $zi + 1 \leq W_i z_i + b_i < 0$, which conflicts with the constraint $z_{i+1} \geq 0$. Therefore, we must choose $v_{i+1} = 0$, where the constaints reduce to

$$\begin{aligned} z_{i+1} &\geq W_i z_i + b_i \\ z_{i+1} &\geq 0 \\ 0 &\geq z_{i+1} \\ W_i z_i + b_i &\geq z_{i+1} + l_i. \end{aligned}$$

Then the second and third consraints imply that $z_{i+1} = 0$. And the first and fourth constraints are satisfied respectively because 1) the second inequality is stricter than the first, and 2) l_i is a lower bound on $W_i z_i + b_i$.

In other words, we have shown that under these constraints if $W_i z_i + b_i > 0$ then $z_{i+1} = W_i z_i + b_i$ and if $W_i z_i + b_i <= 0$ then $z_{i+1} = 0$. These are precisely the conditions of the ReLU operation (again, the above logic is applied elementwise to every entry of z_{i+1}, so the operation is effectively an elementwise ReLU).

Finding upper and lower bounds

First Technique

1. Mathematically, this doesn't actually matter.

2. Practical solution time of integer programming solvers will depend heavily upon having good upper and lower bounds.

3. Compute *exact* value of these bounds the precisely the same way as we compute targeted attacks: just like a targetted attack would minimize $(e_y - e_{y_{targ}})$ times the last layer of weights in a network, we could computer an upper or lower bound by minimizing the single activation value at an intermediate layer, i.e. $(l_k)_j$ would be the solution to the optimization problem.

$$
\begin{aligned}
\underset{z_{1,\ldots,k+1}}{\text{minimize}} \quad & e_j^T z_{k+1} \\
\text{subject to} \quad & \|z_1 - x\|_\infty \leq \epsilon \\
& z_{i+1} = \max\{0, W_i z_i + b_i\}, \quad i = 1, \ldots, k-1 \\
& z_{k+1} = W_k z_k + b_k.
\end{aligned}
$$

extremely impractical.

Finding upper and lower bounds

Second Technique

1. Choose a set of much looser bounds that are very fast to compute.
2. One natural choice here is to use simple interval bounds. If we have some bounds on z, l^≤z≤u^, then the question is: How large or small could we make the term Wz+b?
3. consider a single entry (Wz+b)i

$$(Wz+b)_i = \sum_j w_{ij} z_j + b_i$$

If we want to minimize this term:

1. $W_{ij}<0$ is negative, we should choose $z_j=\hat{u}_j$
2. $W_{ij}>0$ we should choose $z_j=\hat{l}_j$.
3. Set of bounds on Wz+b, given by:

$$\max\{W,0\}\hat{l} + \min\{W,0\}\hat{u} + b \leq (Wz+b) \leq \max\{W,0\}\hat{u} + \min\{W,0\}\hat{l}$$

A final integer programming formulation

$$
\begin{aligned}
\underset{z_{1,\ldots,d+1}, v_{1,\ldots,d-1}}{\text{minimize}} \quad & (e_y - e_{y_{\text{targ}}})^T z_{d+1} \\
\text{subject to} \quad & z_{i+1} \geq W_i z_i + b_i, \;\; i = 1 \ldots, d-1 \\
& z_{i+1} \geq 0, \;\; i = 1 \ldots, d-1 \\
& u_i \cdot v_i \geq z_{i+1}, \;\; i = 1 \ldots, d-1 \\
& W_i z_i + b_i \geq z_{i+1} + (1 - v_i) l_i, \;\; i = 1 \ldots, d-1 \\
& v_i \in \{0, 1\}^{|v_i|}, \;\; i = 1 \ldots, d-1 \\
& z_1 \leq x + \delta \\
& z_1 \geq x - \delta \\
& z_{d+1} = W_d z_d + b_d.
\end{aligned}
$$

Certifying robustness

If we want to determine, *exactly*, whether any adversarial example exists for a given example and :

1. run the integer programming solution using a targeted attack for every possible alternative class label.
2. If any of these optimization objectives have a negative solution, then there exists and adversarial example, and the optimization formulation provides it for us.
3. In contrast, if none of the optimization objectives is negative for *any* target class, then the classifier has been *formally certified* to be robust on this example.

Upper bound

1. The basic strategy here will be consider a ***relaxation*** of the network structure, such that this relaxed version contains the original network, but is built in a manner that is much easier to optimize exactly over.

2. As we will see, this will involve **building *convex relaxations* of the network structure** itself. These methods are a bit different in that they **do *not* typically construct an actual adversarial example for the real network** (because they operate over a relaxed model which is not equivalent to the original one), but they can produce **certifications that a network is *provably* robust against an adversarial attack**.

3. Further, when combined with **robust optimization for training**, these methods make up the state of the art in training provably robust models.

Upper bound

1. While the exact solution to the inner maximization problem based upon integer programming is valuable (for small networks), the approach does not scale to substantially larger networks.
2. These are combinatorial problems that *do* scale exponentially at some point, so for even modestly-sized networks, it is easy to find problems where the optimization will *never* finish (no matter what computational resources) are available.
3. Because of this, if we want to provide formal guarantees of robustness, it is important to be able to obtain fast *upper bounds* on the inner maximization problem.
4. If, for example, we can attain an upper bound which still shows that no targeted attack can change the class label, this also provides a verification that no attack is possible.

Convex Relaxation

1. The element of the integer program above, which is what makes the program combinatorially hard to solve, is precisely the binary integer constraint that we introduced to exactly capture the ReLU operator

$$v_i \in \{0, 1\}^{|v_i|}.$$

2. This is not a convex set, and hence hard to optimize over.

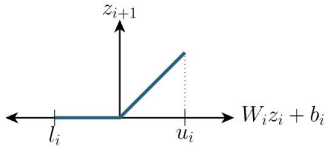
3. This is the *only* difficult constraint in the problem; if it were removed, we would have a linear program, for which there exist very fast solution methods.

4. *relaxation* of the above problem, an obvious idea is simply to relax this contraint that each element of v_i must be either zero or one to the constraint that each element just be between zero or one, but allowed to take on fractional values.

$$0 \leq v_i \leq 1.$$

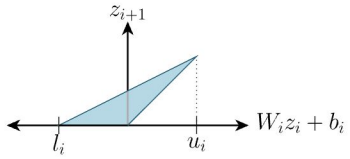
Convex Relaxation

Solving the relaxed problem does not actually produce a true adversarial example anymore.



Bounded ReLU set.

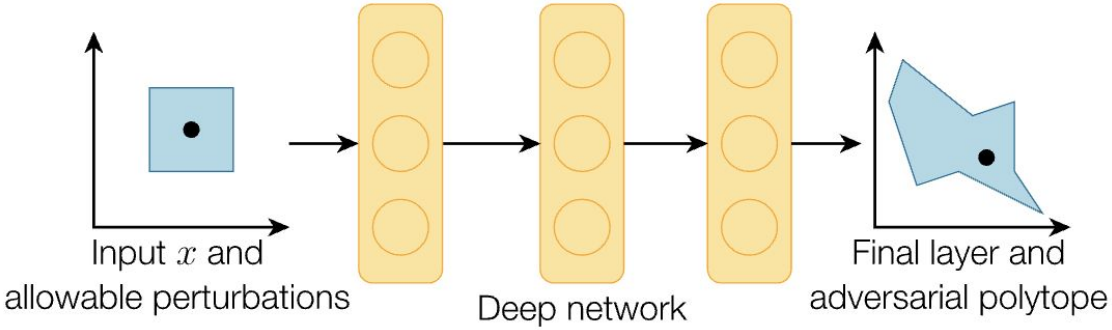
Allowing the v_i terms to be fractional-valued essentially means that we can allows the ReLU to be "partially off and partially on".



Convex hull of the bounded ReLU set.

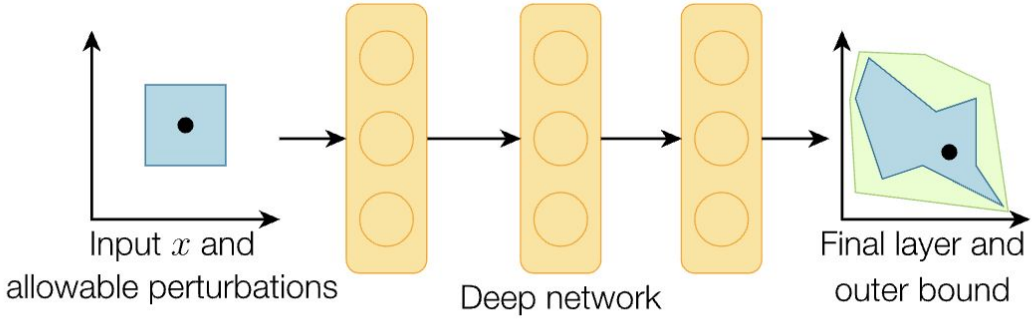
For instance certain pre-ReLU values could be negative, while the corresponding z_{i+1} is positive.

What is achieved by relaxation is not the construction of adversarial example itself, but rather the objective value of the optimization problem, which can formally certify that *no* adversarial example exists.



non-convex set.

Norm-bounded input, and resulting adversarial polytope.



convex

it is a strictly larger set, so if no adversarial example exists in the larger set, then none exists in the true adversarial polytope either.

Convex outer bound on the adversarial polytope

Faster Solutions of Convex Relaxation

Although the convex relaxation we presented here was billed as being “fast” compared to the “slow” integer program, let’s qualify these terms. Running it on perhaps on slightly larger models, this is still not particularly feasible.

Due to this fact, most of the research work focused on convex relaxation approaches to verification are actually based upon methods for *quickly solving* (possibly approximately, but ideally in a manner that still gives guarantees), the convex relaxations.

The details are beyond the scope of this tutorial, but it turns out that using a procedure known as convex duality, plus some manipulation of the optimization problem, we can quickly compute a provable lower bound on the relaxed objective (which in turn gives an even “looser” relaxation of the original problem) **using a single backward pass through the network**.

This is fast enough that we can even use the procedure to compute tighter interval bounds than the simple bound propagation we described earlier.

However, finding the right trade-offs between complexity of the verification procedure and tightness of the bounds is still an open research question, and no truly scalable solution (that is, a procedure that is both computationally efficient *and* which provides tight bounds for truly large-scale networks) has been found yet.

Bound propagation

These bounds are weaker than those based on the convex relaxation, and they do not provide even a “fake” adversarial example like the convex relaxation approaches, but they have the advantage that they are **extremely efficient**.

Training robust networks, it is more worthwhile to use these bound propagation techniques (potentially with a more complex network) than it is to use a simpler network and the tighter but more costly bounds based upon convex relaxations.

Bounding propagation

1. propagating bounds through to the *very last* layer is wasteful,since we trying to minimize some specific linear function of the last layer.
2. instead is propagate these bounds to the second-to-last layer and then analytically solve the minimization problem at the last layer.

$$\begin{aligned} \underset{z_d, z_{d+1}}{\text{minimize}} \quad & c^T z_{d+1} \\ \text{subject to} \quad & z_{d+1} = W_d z_d + b_d \\ & \hat{l} \leq z_d \leq \hat{u}. \end{aligned}$$

But this problem has an easy analytical solution, based upon the same strategy we used for computing the bounds before. Specifically, if we just eliminate the z_{d+1} variable using the first constraint, this problem is equivalent to

$$\begin{aligned} \underset{z_d}{\text{minimize}} \quad & c^T (W_d z_d + b_d) \equiv (W_d^T c)^T z_d + c^T b_d \\ \text{subject to} \quad & \hat{l} \leq z_d \leq \hat{u}. \end{aligned}$$

This is the problem of minimizing a linear function subject to bound constraints, which was exactly the task we had before. The analytic solution is just to choose $(z_d)_j = \hat{l}_j$ if $(W_d^T c)_j > 0$, and $(z_d)_j = \hat{u}_j$ otherwise. This results in the optimal objective value

$$\max\{c^T W_d c, 0\}\hat{l} + \min\{c^T W_d, 0\}\hat{u} + c^T b_d$$

These approaches are all different methods for optimizing (approximately or otherwise) the inner optimization problem of the adversarial robustness objective (or, in some cases, optimizing a linear function of the last level logits, which is effectively the same thing).

In some cases this gives us an actual attack perturbation (indeed, this is often viewed as the whole point of the inner optimization),

But in other cases it merely provided a bound on the optimization objective, useful for provably verifying (and later, for training) robust classifiers.

Convey the nature of the inner optimization problem

1. Just heuristics for approximately optimizing it in practice.
2. Task of finding adversarial examples.

Inner maximization

Is about to find an adversary attack

three main techniques for doing this:

1. local gradient-based search (providing a lower bound on the objective),
2. exact combinatorial optimization (exactly solving the objective),
3. and convex relaxations (providing a provable upper bound on the objective).

Outer Maximization

No matter what attack an adversary uses, we want to have a model that performs well (especially if we don't know the precise strategy that the attacker is going to use)

$$\underset{\theta}{\text{minimize}} \frac{1}{|S|} \sum_{x,y \in S} \max_{\|\delta\| \leq \epsilon} \ell(h_\theta(x+\delta), y).$$

For each of the three methods for solving this inner problem (1) lower bounding via local search, 2) exact solutions via combinator optimziation, and 3) upper bounds via convex relaxations), there would be an equivalent manner for training an adversarially robust system.

However, the second option here is not tennable in practice; solving integer programs is already extremely time consuing,

Outer Maximization

1. Using lower bounds, and examples constructed via local search methods, to train an (empirically) adversarially robust classifier.

2. Using convex upper bounds, to train a provably robust classifier.

There are trade-offs between both approaches here: while the first method may seem less desireable, it will turn out that the first approach empircally creates strong models.

Adversarial training with Adversarial examples

simply create and then incorporate adversarial examples into the training process.

$$\underset{\theta}{\text{minimize}} \frac{1}{|S|} \sum_{x,y \in S} \max_{\|\delta\| \leq \epsilon} \ell(h_\theta(x+\delta), y).$$

Solve it using gradient descent and optimize theta by stochastic gradient descent

$$\theta := \theta - \alpha \frac{1}{|B|} \sum_{x,y \in B} \nabla_\theta \max_{\|\delta\| \leq \epsilon} \ell(h_\theta(x+\delta), y).$$

How do we go about computing this inner gradient?
Danskin's Theorem:

compute the (sub)gradient of a function containing a max term, we need to simply
1) find the maximum, and
2) compute the normal gradient evaluated at this point.

$$\nabla_\theta \max_{\|\delta\| \leq \epsilon} \ell(h_\theta(x+\delta), y) = \nabla_\theta \ell(h_\theta(x+\delta^\star(x)), y)$$

$$\delta^\star(x) = \underset{\|\delta\| \leq \epsilon}{\text{argmax}} \ell(h_\theta(x+\delta), y).$$

Clean training next to adversarial training

Repeat:

1. Select minibatch B, initialize gradient vector $g := 0$
2. For each (x, y) in B:
   a. Find an attack perturbation $\delta^\star$ by (approximately) optimizing
   $$\delta^\star = \underset{\|\delta\| \leq \epsilon}{\operatorname{argmax}} \ell(h_\theta(x + \delta), y)$$
   b. Add gradient at $\delta^\star$
   $$g := g + \nabla_\theta \ell(h_\theta(x + \delta^\star), y)$$
3. Update parameters θ
   $$\theta := \theta - \frac{\alpha}{|B|} g$$

1. *Also* include a bit of the standard loss (i.e., also take gradient steps in the original data points), as this tends to also slightly improve the performance of the "standard" error of the task.

2. Randomize over the starting positions for PGD

Other Attacks

1. Whenever we train a network against a *specific* kind of attack, it's incredibly easy to perform well against that particular attack in the future.
2. Deep network are incredibly good at predicting precisely the class of data they were trained against.
3. What about if we run some other attack, like FGSM? What if we run PGD for longer? Or with randomization?
4. FGSM *is* really just one step of PGD with a step size of $\alpha=\epsilon$

The model was trained under one single attack model; of course it will not work well to prevent some completely different attack model. If one *does* desire a kind of "generalization" across multiple attack models, then we need to formally define the set of attack models we care about, and train the model over multiple different draws from these attack models.

What is happening with these robust models?

1. why do these models work well against robust attacks?
2. why have some other proposed methods for training robust models (in)famously come up short in this regard?

The robust model has a loss that is quite flat both in the gradient direction (that is the steeper direction), and in the random direction, whereas the traditionally trained model varies quite rapidly both in the gradient direction and (after moving some in the gradient direction) in the random direction.

Relaxation-based robust training

1. Use the convex relaxation methods not just to verify networks, but also to train them.
2. Focusing on the interval-based bounds.
3. Using our interval bound to try to *verify* robustness for the empirically robust classifier we just trained.
4. A classifier is verified to be robust against an adversarial attack if the optimization objective is positive for *all* targeted classes.
5. relaxation-based verification approaches, is that they are vacuous when evaluated upon a network trained without knowledge of these bounds.
6. Additionally, these errors tend to accumulate with the depth of the network, precisely because the interval bounds as we have presented them also tend to get looser with each layer of the network

Training using provable criteria

1. if the verifiable bounds we get are this loose, even for empirically robust networks, of what value could they be?
2. if we *train* a network specifically to minimize a loss based upon this upper bound, we get a network where the bounds are meaningful.
3. use the interval bounds to upper bound the cross entropy loss of a classifier and then minimize this upper bound.

Conclusion

Robust models are seemingly pretty close to their traditional counterparts

on large-scale problems we are nowhere *close* to building robust models that can match standard models in terms of their performance.

Even on a dataset like CIFAR10, for example, the best known robust models that can handle a perturbation of 8/255=0.031 color values achieve an (empirical) robust *error* of of 55%, and the best provably robust models have an error greater than 70%.

On the flipside, the choices we have with regards to training procedures, network architecture, regularization, etc, have barely been touched in the robust optimization context.

Resources

1. https://adversarial-ml-tutorial.org