Chapter 1 - Introduction to adversarial robustness

✓ The model: $h_\theta : \underline{X} \to \mathbb{R}^K$ mapping from input space to the output space

$\underline{X}$: input space

K: number of classes being predicted

θ: model's parameters

✓ Loss function: $\ell : \mathbb{R}^K \times Z_+ \to \mathbb{R}_+$ mapping from model predictions and true labels to a non-negative number

$\ell(h_\theta(x), y)$: $\begin{cases} h_\theta(x): \text{the model output (logits)} \\ y: \text{the index of the true class} \end{cases}$

✓ A common form of loss used in deep learning: Cross entropy loss

To maximize the log of the true probability of the true class label using softmax operator:

$$\log\left(\frac{\exp(h_\theta(x)_y)}{\sum_{j=1}^{K}\exp(h_\theta(x)_j)}\right) = h_\theta(x)_y - \log\left(\sum_{i=1}^{K}\exp(h_\theta(x)_j)\right) \Rightarrow$$

$$\text{minimize } \ell(h_\theta(x), y) = \log\left(\sum_{i=1}^{K}\exp(h_\theta(x)_j)\right) - h_\theta(x)_y$$

✓ To train a classifier: $\underset{\theta}{\text{minimize}} \frac{1}{m}\sum_{i=1}^{m}\ell(h_\theta(x_i), y_i)$

we solve it by stochastic gradient descent for some minibatch $B \subseteq \{1, \dots, m\}$

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

✓ To form an adversarial example: $\underset{\hat{x}}{\text{maximize}}\ \ell(h_\theta(\hat{x}), y)$

We need to ensure that $\hat{x}$ is close to our original input x.

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

chapter 2 - Linear models

✓ minimize $\frac{1}{|D|}\sum_{x,y\in D}\max_{\|\delta\|\le\epsilon} \ell(\underbrace{W(x+\delta)+b}_{h_\theta(x)},y)$

Inner maximization can be solved exactly for the case of binary classification & minimization problem is still convex ⇒ robust training procedure can also be solved optimally. However: deep network neither the inner maximization problem nor the outer minimization problem can be solved globally.

✓ Binary classification: Binary class entropy:

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

Proof: Probability of predicting class 2: $\frac{\exp(h_\theta(x)_2)}{\exp(h_\theta(x)_1+h_\theta(x)_2)}$

Probability of predicting class 1: $\frac{\exp(h_\theta(x)_1)}{\exp(h_\theta(x)_1+h_\theta(x)_2)}$

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

$$\Rightarrow P(y|x)=\frac{1}{1+\exp(-y\cdot h'_\theta(x))} \xrightarrow{-\log} \log(1+\exp(-y\cdot h'_\theta(x)))$$ Logistic Loss

✓ solving the inner maximization:

Function of $L(z)=\log(1+\exp(-z))$ is monotic decreasing ⇒

$$\max_{\|\delta\|\le\epsilon} L(y\cdot(w^T(x+\delta)+b))=L(\min_{\|\delta\|\le\epsilon} y\cdot(w^T(x+\delta)+b))=L((y\cdot(w^Tx+b)+\min_{\|\delta\|\le\epsilon} y\cdot w^T\delta)$$

$$\Rightarrow \min_{\|\delta\|\le\epsilon} y\cdot w^T\delta$$

if $y=+1$, $\|\delta\|_\infty\le\epsilon \Rightarrow \begin{cases}\delta_i=-\epsilon & \text{for } w_i\ge 0\\ \delta_i=\epsilon & \text{for } w_i<0\end{cases} \Rightarrow \delta^*=-y\epsilon\cdot\text{sign}(w)$

if $y=-1$ Flip

$$y\cdot w^T\delta^*=-y^2\epsilon\sum_i|w_i|=-\epsilon\|w\|_1 \Rightarrow \underset{w,b}{\text{minimize}}\ \frac{1}{D}\sum_{(x,y)\in D} L(y\cdot(w^Tx+b)-\epsilon\|w\|_1)$$

Convex in w,b

Intuitively: if a point is far from the decision boundry, we dont penalize the norm of the parameters, but we do penalize the norm of the parameters for a point where we close to the decision boundry.

✓ A common perturbation set to use is the ℓ_∞ ball:

$$\Delta = \{\delta : \|\delta\|_\infty \leq \epsilon\}, \quad \|z\|_\infty = \max_i |z_i|$$

since each step is followed by a projection back on to the ℓ_∞ ball, this procedure is known as Projected gradient descent (PGD)

✓ targeted attacks

we maximize the loss of the correct class while also minimize the loss of the target class.

$$\underset{\delta \in \Delta}{\text{maximize}} \left(\ell(h_\theta(x+\delta), y) - \ell(h_\theta(x+\delta, y_{target})) \right) \equiv \underset{\delta \in \Delta}{\text{maximize}} \left(h_\theta(x+\delta)_{y_{target}} - h_\theta(x+\delta)_y \right)$$

✓ The risk of a classifier is its expected loss under the true distribution of samples

$$R(h_\theta) = \mathop{E}_{(x,y)\sim D} [\ell(h_\theta(x), y)] \quad D: \text{true distribution over samples}$$

In practice: $\hat{R}(h_\theta, D) = \frac{1}{|D|} \sum_{(x,y)\in D} \ell(h_\theta(x), y) \Rightarrow \text{minize } \hat{R}(h_\theta, D)_{train}$

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

if we are truely operating in an adversarial environment, adversarial risk would provide a more accurate estimate of the expected performance of the classifier.

✓ To train adversarially robust classifiers:

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

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

How to solve it? Danskin's theorem

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

We iteratively compute adversarial examples and then update the classifier based not upon the original data points, but upon these adversarial examples.

Chapter 3: Adversarial examples solving the inner maximization.

✓ Neural network is much more sensitive to adversarial attacks than even the naive linear models and much harder to protect against

$$\text{maximize}_{\|\delta\| \le \epsilon} \ell(h_\theta(x+\delta), y)$$

$h_\theta(x)$ represents a neural network:

$z_1 = x$

$z_{i+1} = f_i(W_i z_i + b_i)$ $i = 1, \dots, d$

$h_\theta(x) = z_{d+1}$

ReLu operator $f_i(z_i) = \max\{0, z_i\}$ for $i = 1, \dots, d-1$

Linear operator $f_i(z)$: for layer d

✓ In the high dimensional setting that we will consider with deep networks, there is a high likelihood that at most any point in input space, there will be some direction along the loss surface that is very steep.

* moving a small distance in input space ⇒ a big increase in loss ⇒ NNs are prone to adversarial examples.

* The cost surface for NN is not convex ⇒ is prone to Local optima

* many directions of high cost increase ⇒ an adversarial example

No optimal adversarial example ⇒

training robust networks will be a big Problem Danskin's theorem X

✓ strategies for the inner maximization:

1: lower bounds 2: exact solutions 3: upper bounds

✓ Three different architectures here: 1: two-layer fully-connected

2: four-layer fully-connected

3: four convolutional layes + one fully-connected

✓ Lower bounding the inner maximization:

1: using backpropagation 2: δ needs to stay within the norm bound ϵ

✓ Fast Gradient sign method (FGSM)

$$g := \nabla_\delta \ell(h_\theta(x+\delta), y) \qquad \delta := \delta + \alpha g$$
$$\|\delta\|_\infty \leq \epsilon$$

If we want to make increase the loss as much as possible, it makes sense to take as large a step as possible. & for $\delta_0 = 0$

if $\alpha >>$ ⇒ the relative sizes of the entries of g won't matter ⇒

Si can be considered to be either $+\epsilon$ or $-\epsilon$ ⇒ $\delta := \epsilon \cdot \text{sign}(g)$

Results: Fully connected networks are particularly succeptible to minor changes and ConvNets slightly less but ofcourse they are still very sensitive.

Points: FGSM is an attack under an ℓ_∞ norm bound

FGSM is exactly the optimal attack against a linear binary classification

NNs are not in fact linear even over a relatively small regoin.

✓ Projected Gradient descent

Repeat:

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

P: Projection on to the ball of interest

we need to determine actual stepsize and the number of iterations.

In this method, step size α is too large. → To solve this issue → Normalized steepest descend method

traditional GD: $z_1 := z_1 - \alpha \nabla_z f(z)$

Normalized steepest descent method: $z_1 := z - \underset{\|v\| \leq \alpha}{\text{argmax}}\ v^T \nabla_z f(z) \Rightarrow$

$$\begin{cases} z := z - \alpha \dfrac{\nabla_z f(z)}{\|\nabla_z f(z)\|_2} & \text{for } \ell_2 \\ z := z - \alpha\, \text{sign}(\nabla_z f(z)) & \text{for } \ell_\infty \end{cases}$$

against linear func

Result: PGD is not much more effective than FGSM

Normalized steepest descent is doing alot better than the FGSM algorithm.

* In normalized steepest descent, α is on the same scale as the total Perturbation bound ϵ.

* we can use randomization to improve the performance of algorithm. Randomization mitigates the problem of local optima we run it multiple times from different random locations. $\Rightarrow$ increases runtime $\rightarrow$ may not be practical.

✓ Targetted ~~classes~~ attacks

We can modify our objective to maximize the target class logit:

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

This func. is more difficult than the previous one, we aren't able to fool the classifier as much, but it is more able to predict the target class.

✓ Non-ℓ_∞ norm

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

Projection on to the ℓ_2 ball of radius ϵ $\leftarrow P_\epsilon(z) = \epsilon \frac{z}{\max\{\epsilon, \|z\|_2\}}$

* ϵ for ℓ_2 norm is larger than ϵ for ℓ_∞ norm

✓ ℓ_∞ attacks lead to small noise everywhere in image

✓ ℓ_2 is more localized in the image

✓ ℓ_1 encourages sparsity in the $\delta \Rightarrow$ afew pixels locations are adjusted.

✓ It is more useful to refer to attacks more formally by 1) the norm ball Pertabration they consider 2) the method they use for optimizing over that norm ball.

✓ Exactly solving the inner maximization

It is for small NNS.

$$z_1 = x$$
$$z_{i+1} = f_i(W_i z_i + b_i) \quad i, \dots, d$$
$$h_\theta(x) = z_{d+1}$$

$f_i(z_i) = \text{ReLU}(z_i)$ for $i = 1, \dots, d-1$ & $f_d(x) = x$

we focus on ℓ_∞

we can rewrite the problem as

$$\text{minimize } (e_y - e_{y_{targ}})^T z_{d+1}$$
$$\text{Subject to } \|z_1 - x\|_\infty \le \epsilon$$
$$z_{i+1} = \max\{0, W_i z_i + b_i\}, \quad i = 1, \dots, d-1$$
$$z_{d+1} = W_d z_d + b_d$$

non-convex & is not handled by most optimization solvers ←

e_i = a vector with a one in the ith position and zeros everywhere else.

A mixed integer programming formulation (MILP) is an extremely well-studied area for small problems.

Linearization: we can express $z_{i+1} = \max\{0, W_i z_i + b_i\}$ using linear constraints if l_i, u_i are lower bound and upper bound for $W_i z_i + b_i$ ⇒

$$z_{i+1} = \max\{0, W_i z_i + b_i\} \equiv \begin{cases} z_{i+1} \ge W_i z_i + b_i \\ z_{i+1} \ge 0 \\ u_i \cdot a_i \ge z_{i+1} \\ W_i z_i + b_i \ge z_{i+1} + (1 - a_i) l_i \\ a_i \in \{0, 1\}^{|v_i|} \end{cases} \Rightarrow \begin{cases} z_{i+1} = W_i z_i + b_i & \text{if } W_i z_i + b_i > 0 \\ z_{i+1} = 0 & \text{if } W_i z_i + b_i = 0 \end{cases}$$

⇓

ReLU operation

we choose a set of much looser bounds for l_i and u_i

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

if we want to minimize $\sum_j w_{ij} z_j + b_i \Rightarrow \begin{cases} z_j = \hat{u}_j & \text{if } W_{ij} < 0 \\ z_j = \hat{l}_j & \text{if } W_{ij} > 0 \end{cases}$

to maximize vice versa

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

if we begin with $\hat{l} = \max\{x - \epsilon, 0\}$, $\hat{u} = \min\{x + \epsilon, 0\}$

These bounds tell us that for a perturbation delta of ϵ, the logit is somewhere in the range $[l, U]$

A final integer programming formulation

$$\underset{z_1,\dots,z_{d+1},\,v_1,\dots,v_{d-1}}{\text{minimize}} \; (e_y - e_{y_{targ}})^T z_{d+1}$$

$$\text{subject to } z_{i+1} \geq W_i z_i + b_i, \quad i = 1,\dots,d-1$$

$$z_{i+1} \geq 0 \quad i = 1,\dots,d-1$$

$$u_i \cdot v_i \geq z_{i+1} \quad i = 1,\dots,d-1$$

$$W_i z_i + b_i \geq z_{i+1} + (1 - v_i) \ell_i, \quad i = 1,\dots,d-1$$

$$v_i \in \{0,1\}^{|z_i|}, \quad i = 1,\dots,d-1$$

$$z_1 \leq x + \delta$$

$$z_1 \geq x - \delta$$

$$z_{d+1} = W_d z_d + b_d$$

We can solve this problem using the cvxpy library, by Gurobi solver

when the resulting objective is negative, we are able to find a perturbation that makes the class logit for the target class is larger than the class logit for the original class.

Certifying robustness

if we want to determine, exactly, whether any adversarial examples exists for a given example, we can run the integer programming solution. If any of these optimization objectives have a negative solution, then there exists an adversarial example. In contrast, if none of the optimization objectives is negative for any target class, then the classifier has been certified to be robust on this example.

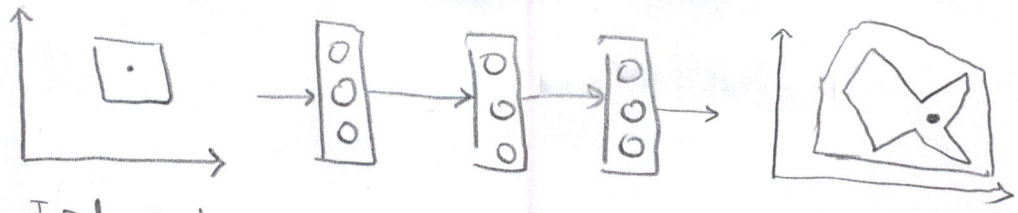
Upper bounding the inner maximization (Convex relaxations)

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.

Method 1: $0 \leq v_i \leq 1$ to relax the constraint that each element of v_i must be either zero or one.

Solving the relaxed version of a targetted attack problem and objective is still positive $\Rightarrow$ this attack will not work $\Longrightarrow$ adv examples will not exist in the original set

relaxed set is larger than the original set



Input x and allowable perturbations

Deep Network

Final layer and Convex outer bound on the adversarial polytope

If no adversarial example exists in the larger set, then none exists in the true adversarial polytope either.

✓ Faster solutions of the convex relaxation

A Procedure known as Conven duality can compute a provable lower bound on the relaxed objective.

Interval-Propagation-based bounds

Using bound propagation techniques with more complex network is more worthwhile than to use a simpler network and the tighter but more costly bounds based upon convex relaxtions

If we have an interval bound on the second-to-last layer $\hat{l} \leq z_d \leq \hat{u}$ we can simply solve the optimization Problem

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

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