[Lecture 18]

0.1 Adversarial Training

We next discuss how to minimize the robust loss, which we recall is

$$L(p,\theta) = \mathbb{E}_{(x,y)\sim p}\left[\max_{\bar{x}:d(x,\bar{x})\leq\epsilon} \ell(\theta;\bar{x},y)\right] \tag{1}$$

Our focus will be on computing the gradient $\nabla_\theta L(p,\theta)$ with respect to θ. If we can compute this gradient, then we can typically minimize $L(p,\theta)$ by gradient descent.

The *envelope theorem* show us how to compute the gradient of a maximum:

Theorem 0.1 (Envelope theorem)**.** *Suppose that $f(\theta,x)$ is continuously differentiable in θ and x, and that $\mathcal{B}$ is a compact set. Then*

$$\nabla_\theta \max_{x\in\mathcal{B}} f(\theta,x) = \nabla_\theta f(\theta,x)|_{x=\arg\max_{x\in\mathcal{B}} f(\theta,x)}, \tag{2}$$

assuming the arg max *is unique. In other words, the gradient of a max is the gradient at the argmax.*

This is actually a special case; the envelope theorem also covers the case where the constraints can depend on θ. Additionally, we can often relax the compactness condition on $\mathcal{B}$, although some topological condition is needed. Finally, in the convex case Danskin's theorem handles situations where the arg max is not unique, and instead shows that the subgradient is the convex hull of $\{\nabla_\theta f(\theta,x) \mid x \in \arg\max\{f(\theta,x) \mid x\in\mathcal{B}\}\}$.

The envelope theorem motivates the following procedure, called **adversarial training**. Given training points $(x_1,y_1),\ldots,(x_n,y_n)$, we do the following:

- Sample a batch of examples $\{(x_i,y_i) \mid i\in S\}$.
- For each example in the batch, let $\bar{x}_i$ approximately maximize $\ell(\theta;\bar{x},y_i)$ such that $d(x_i,\bar{x})\leq\epsilon$.
- Update θ in the direction $\frac{1}{|S|}\sum_{i\in S}\nabla_\theta \ell(\theta;\bar{x}_i,y_i)$.

If there is a regularizer, we would include the regularizer in the update as well.

If we exactly maximize over $\bar{x}$, then this exactly computes a (stochastic) gradient of the robust loss with respect to θ. If we only approximately maximize over $\bar{x}$, then this procedure has no guarantees, but we might hope that sufficiently good approximations lead to a good value of θ (this is only sometimes the case).

As mentioned in the previous section, a typical way of approximating the maximization would be projected gradient ascent in $\bar{x}$: take a gradient step in $\bar{x}$, then project onto the set $\{\bar{x} \mid d(x_i,\bar{x})\leq\epsilon\}$. There isn't much to say about this beyond the empirical story, so we will focus on an alternative procedure called *certified adversarial training*, based on minimizing an upper bound on the robust loss.

0.2 Certified Adversarial Training

Define $\ell_{\mathsf{robust}}(\theta,x) = \max_{\bar{x}:d(x,\bar{x})\leq\epsilon}\ell(\theta;\bar{x},y)$. In certified adversarial training, we seek to construct a function ℓ_{cert} such that $\ell_{\mathsf{cert}}(\theta,x,y)\geq\ell_{\mathsf{robust}}(\theta,x,y)$ for all x,θ. Then we will minimize $\mathbb{E}[\ell_{\mathsf{cert}}(\theta;x,y)]$. Minimizing this upper bound is more desirable than minimizing a lower bound, since ℓ_{cert} is large when it is a bad approximation to ℓ_{robust}, and so the optimizer will tend to avoid those regions and find regions where ℓ_{cert} (and hence ℓ_{robust}) is small.

We will consider two strategies for constructing an appropriate ℓ_{cert}. Both are based on convex relaxations; the first is based on an LP relaxation and the second is based on an SDP relaxation. In both cases, we will need to make use of the specific structure of the function ℓ. Here we assume that ℓ corresponds to a neural

network, and so can be obtained via the following system of equations:

$$\begin{aligned} x^{(0)} &= x \\ z^{(1)} &= A^{(0)}x^{(0)} \\ x^{(1)} &= \max(z^{(1)}, 0) \\ z^{(2)} &= A^{(1)}x^{(1)} \\ &\vdots \\ x^{(L-1)} &= \max(z^{(L-1)}, 0) \\ z^{(L)} &= A^{(L-1)}x^{(L-1)} \\ \ell &= \log(\sum_j \exp(z_j^{(L)})) - z_y^{(L)} \end{aligned} \tag{3}$$

Here the $z^{(i)}$ and $x^{(i)}$ are pre- and post-activation firings in the neural network, and the final loss ℓ is the cross-entropy loss, which is the negative log-probability assigned to y under the distribution $\pi(y) \propto \exp(z_y^{(L)})$. Thus in particular the dimension of $x^{(0)}$ is the same dimension d as that of x, while the dimension of $z^{(L)}$ is the number of classes k.

For simplicity we will consider the two-class case, where up to a monotone transformation we wish to bound $z_2^{(L)} - z_1^{(L)}$. We will also specialize to the distance $d(x, \bar{x}) = \|x - \bar{x}\|_\infty$. Thus, ℓ_{cert} can be obtained from any valid upper bound on the optimization problem

$$\underset{x^{(0:L-1)}, z^{(1:L)}}{\text{maximize}} \quad z_2^{(L)} - z_1^{(L)} \tag{4}$$

$$\text{subject to } z^{(i+1)} = A^{(i)}x^{(i)} \qquad \text{for } i = 0, \ldots, L-1 \tag{5}$$

$$x^{(i)} = \max(z^{(i)}, 0) \qquad \text{for } i = 0, \ldots, L-1 \tag{6}$$

$$|x_j^{(0)} - x_j| \leq \epsilon \qquad \text{for } j = 1, \ldots, d. \tag{7}$$

The first and last constraints are both linear inequality constraints ($|z| \leq \epsilon$ can be written as $z \leq \epsilon$ and $z \geq -\epsilon$), so the only source of non-convexity is the constraint $x^{(i)} = \max(z^{(i)}, 0)$. The LP and SDP relaxations are two different ways of handling this constraint.

Linear programming relaxation. The most obvious way to relax $x = \max(z, 0)$ to an LP is to replace it with $x \geq \max(z, 0)$, which corresponds to the two linear constraints $x \geq z$, $x \geq 0$. However, these constraints allow x to be infinitely large, which in almost all cases will make the LP itself infinite as well (since e.g. we can make any coordinates of $x^{(L-1)}$ infinitely large, which will allow $z_2^{(L)} - z_1^{(L)}$ to become infinitely large if $A_2^{(L-1)} - A_1^{(L-1)}$ has any positive entries).

To fix this, suppose for a moment that we know that $z \in [l, u]$ for some lower and upper bounds $l \leq 0 \leq u$. Then we also know that $x \leq \frac{z-l}{u-l}u$ (this takes on the value 0 at $z = l$ and u at $z = u$). Thus we can relax $x = \max(z, 0)$ to the three constraints:

$$x \geq z, \quad x \geq 0, \quad x \leq \frac{z-l}{u-l}u. \tag{8}$$

Now x is bounded both above and below under the constraints, so we can obtain non-trivial bounds. Also observe that if our bounds $[l, u]$ instead satisfied $l \leq u \leq 0$ or $0 \leq l \leq u$ (i.e. l and u have the same sign) then the constraint is even simpler since we know that either $x = 0$ (if $l, u \leq 0$) or $x = z$ (if $l, u \geq 0$). Thus in all cases we achieve some sort of bound on x, and all that remains is to explain how to compute l and u.

Computing coordinate-wise bounds. Conceptually, we can compute the bounds l and u inductively from layer to layer. To start with, we have $l^{(0)} \leq x^{(0)} \leq u^{(0)}$ where $l_j^{(0)} = -\epsilon$, $u_j^{(0)} = \epsilon$ for all j; this is just the

ℓ_∞ constraint. Then for computing $u^{(I)}$ assuming we already know $l^{(0:I-1)}, u^{(0:I-1)}$, we solve the following LP for each coordinate j:

$$\underset{x^{(0:I-1)}, z^{(1:I)}}{\text{maximize}} \; z_j^{(I)} \tag{9}$$

$$\text{subject to } z^{(i+1)} = A^{(i)} x^{(i)} \qquad \text{for } i = 0, \ldots, I-1 \tag{10}$$

$$x^{(i)} \geq z^{(i)}, \quad x^{(i)} \geq 0, \quad x^{(i)} \leq \frac{z^{(i)} - l^{(i)}}{u^{(i)} - l^{(i)}} u^{(i)} \qquad \text{for } i = 0, \ldots, I-1 \tag{11}$$

$$-\epsilon \leq x^{(0)} \leq \epsilon \tag{12}$$

We can compute $l^{(I)}$ similarly by minimizing instead of maximizing. Of course this is very inefficient since it requires solving a linear program for every single node in the network (and every single training example). We will discuss below how to avoid these costs. Note that these are both of the same form as the original program, but with a different objective instead of the original $z_2^{(L)} - z_1^{(L)}$.

Faster coordinate-wise bounds via duality. To obtain bounds, it is helpful to work in the dual, since any feasible setting of the dual variables yields an upper bound. Define dual variables $\lambda^{(i+1)}$ for the equality constraint, $\lambda_+^{(i)}$, $\mu_+^{(i)}$ for the $x \geq z$ and $x \geq 0$ constraints, and $\lambda_-^{(i)}$ for the linear upper bound constraint on x. Also define variables η_+, η_- for the $x^{(0)} \geq x - \epsilon$ and $x^{(0)} \leq x + \epsilon$ constraints. Then the dual becomes

$$\underset{\lambda, \lambda_+ \geq 0, \mu_+ \geq 0, \lambda_- \geq 0, \eta_+ \geq 0, \eta_- \geq 0}{\text{minimize}} \; \sum_{i=1}^{I-1} \langle \lambda_-^{(i)}, \frac{u^{(i)} l^{(i)}}{u^{(i)} - l^{(i)}} \rangle + \epsilon \mathbb{1}^\top (\eta_+ + \eta_-) \tag{13}$$

$$\text{subject to } \lambda_-^{(i)} - \lambda_+^{(i)} - \mu_+^{(i)} = (A^{(i)})^\top \lambda^{(i+1)}, i > 0 \tag{14}$$

$$(\eta_+ - \eta_-) = (A^{(0)})^\top \lambda^{(1)} \tag{15}$$

$$\lambda^{(i)} = \frac{u^{(i)}}{u^{(i)} - l^{(i)}} \lambda_-^{(i)} - \lambda_+^{(i)}, i < I \tag{16}$$

$$\lambda^{(I)} = e_j, \tag{17}$$

$$\tag{18}$$

where e_j has a single 1 in entry j and zero in all other entries. We can simplify this by exploiting complementary slackness: we must have $\lambda_+ = \mu_+ = 0$ for all coordinates where $(A^{(i)})^\top \lambda^{(i+1)} > 0$, and similarly $\lambda_- = 0$ for all coordinates where $(A^{(i)})^\top \lambda^{(i+1)} < 0$. Letting $(\cdot)_+$, $(\cdot)_-$ denote the positive and negative parts of a vector, we then have

$$\lambda_-^{(i)} = ((A^{(i)})^\top \lambda^{(i+1)})_+, \tag{19}$$

$$\lambda_+^{(i)} + \mu_+^{(i)} = ((A^{(i)})^\top \lambda^{(i+1)})_-. \tag{20}$$

$$\tag{21}$$

Since μ_+ is an arbitrary non-negative vector, the latter constraint can be re-expressed as $\lambda_+^{(i)} = \alpha^{(i)} ((A^{(i)})^\top \lambda^{(i+1)})_-$ for some vector α whose entries lie in $[0, 1]$. Thus simplifying by substituting in α to remove μ_+, and also

merging η_- and η_+ together, we obtain

$$\begin{aligned}
\underset{\lambda,\eta,\lambda_+\geq 0,\lambda_-\geq 0}{\text{minimize}} \ & \sum_{i=1}^{I-1} \langle \lambda_-^{(i)}, \frac{u^{(i)}l^{(i)}}{u^{(i)}-l^{(i)}} \rangle + \epsilon\|\eta\|_1 && (22)\\
\text{subject to } & \lambda_-^{(i)} = \max((A^{(i)})^\top \lambda^{(i+1)}, 0), i > 0 && (23)\\
& \lambda_+^{(i)} = \alpha^{(i)} \max(-(A^{(i)})^\top \lambda^{(i+1)}, 0), i > 0 && (24)\\
& \eta = (A^{(0)})^\top \lambda^{(1)}, && (25)\\
& \lambda^{(i)} = \frac{u^{(i)}}{u^{(i)}-l^{(i)}} \lambda_-^{(i)} - \lambda_+^{(i)}, i < I && (26)\\
& \lambda^{(I)} = e_j, && (27)\\
& && (28)
\end{aligned}$$

For fixed $\alpha^{(i)}$, this further simplifies to a simple backpropagation:

$$\begin{aligned}
\underset{\lambda,0\leq\alpha\leq 1}{\text{minimize}} \ & \sum_{i=0}^{I-1} \langle \max((A^{(i)})^\top \lambda^{(i+1)}, 0), \frac{u^{(i)}l^{(i)}}{u^{(i)}-l^{(i)}} \rangle + \epsilon\|(A^{(0)})^\top \lambda^{(1)}\|_1 && (29)\\
\text{subject to } & \lambda^{(i)} = \frac{u^{(i)}}{u^{(i)}-l^{(i)}} \max((A^{(i)})^\top \lambda^{(i+1)}, 0) - \alpha^{(i)} \max(-(A^{(i)})^\top \lambda^{(i+1)}, 0), i < I && (30)\\
& \lambda^{(I)} = e_j. && (31)
\end{aligned}$$

Finally, we can choose the feasible value $\alpha^{(i)} = \frac{u^{(i)}}{u^{(i)}-l^{(i)}}$, in which case λ is determined and evolves according to the particularly simple backpropagation dynamics $\lambda^{(i)} = \frac{u^{(i)}}{u^{(i)}-l^{(i)}}(A^{(i)})^\top \lambda^{(i+1)}$.

We conveniently no longer need to solve a linear program for each node in the network, but can instead obtain the upper and lower bounds (as well as the final bound) just via backpropagation. This still necessitates many backpropagation operations, and the bound could end up being fairly loose. Indeed, for an arbitrarily chosen network this bound will likely be very loose. However, optimizing this bound will tend to find networks where the bound is fairly tight. See **?** for futher details. Also, **?** train a neural network to *predict* a good choice of the dual variables, rather than guessing them by hand.

Semidefinite programming relaxation. Recall that our goal in the LP relaxation was to handle the constraint $x = \max(z, 0)$. We did this by computing a convex outer bound to this set assuming an upper and lower bound on z. We can instead express $x = \max(z, 0)$ as a system of polynomial constraints and apply an SDP relaxation. Specifically:

$$x = \max(z, 0) \iff x \geq z, \quad x \geq 0, \quad x(x - z) = 0. \tag{32}$$

If we add additional variables X, Y, and Z (meant to represent x^2, xz, and z^2), and let $M = \begin{bmatrix} 1 & x & z \\ x & X & Y \\ z & Y & Z \end{bmatrix}$, we can write these constraints as

$$M \succeq 0, \text{rank}(M) = 1, x \geq z, x \geq 0, X = Y. \tag{33}$$

Dropping the rank constraint and substituting $Y = X$, we obtain the SDP relaxation

$$\begin{bmatrix} 1 & x & z \\ x & X & X \\ z & X & Z \end{bmatrix} \succeq 0, x \geq z, x \geq 0. \tag{34}$$

Applying this to all of the variables in the original optimization problem, we obtain:

$$\underset{x,z,X,Z}{\text{maximize}}\ z_2^{(L)} - z_1^{(L)} \tag{35}$$

$$\text{subject to } z^{(i+1)} = A^{(i)}x^{(i)} \qquad \text{for } i = 0,\dots,L-1 \tag{36}$$

$$Z^{(i+1)} = A^{(i)}X^{(i)}(A^{(i)})^\top \qquad \text{for } i = 0,\dots,L-1 \tag{37}$$

$$x^{(i)} \geq z^{(i)}, x^{(i)} \geq 0 \qquad \text{for } i = 0,\dots,L-1 \tag{38}$$

$$\begin{bmatrix} 1 & (x^{(i)})^\top & (z^{(i)})^\top \\ x^{(i)} & X^{(i)} & X^{(i)} \\ z^{(i)} & X^{(i)} & Z^{(i)} \end{bmatrix} \succeq 0, \tag{39}$$

$$|x_j^{(0)} - x_j| \leq \epsilon \qquad \text{for } j = 1,\dots,d. \tag{40}$$

For efficiency we would want to collapse x and z (and X and Z) into a single variable by exploiting the equality constraints, but we ignore that for simplicity. There is also one problem with the above optimization, which is that the absolute value constraint on $x^{(0}$ does little to control $X^{(0)}$. A better constraint (which is generally necessary to obtain non-vacuous bounds) is $X_{jj}^{(0)} - 2x_j^{(0)}x_j + x_j^2 \leq \epsilon^2$, which is obtained by squaring the absolute value constraint and applying the SDP relaxation (a good exercise is to show that this implies the original absolute value constraint).

Finally, this relaxation generally performs better if it also makes use of upper and lower bounds (obtained, e.g., via the same method as the LP). If we know that $u \leq x \leq l$, we can incorporate this as the quadratic constraint $(x-l)(x-u) \leq 0$, which becomes $X - (l+u)x + ul \leq 0$.

With these combined improvements, the SDP relaxation is sometimes powerful enough to give good certified bounds even for networks that were not trained against the bound; see **?** for details.

Comparison of approaches. The LP relaxation is generally faster while the SDP relaxation is tighter. However, an important point in this setting is that *speed can often be turned into accuracy.* This is because faster solve times allow us to train larger neural networks, which have more flexibility to find parameter configurations for which the verifier works well. The LP currently so far has been more successful, although both methods would likely improve with further development. In addition, other verification strategies such as interval bound propagation have also been used to obtain certified bounds. However, the current limit of any of these approaches seems to be medium-sized datasets such as CIFAR-10; perhaps in the future we will figure out how to scale these approaches to ImageNet.