[Lecture 17]

1 Test-Time Robustness

We now consider a different setting, focused on modeling corruptions at training time, rather than test time. For now assume that we are solving a classification task, so we wish to a predict a label $y \in \mathcal{Y}$ given covariates x (for instance, $\mathcal{Y} = \{-1, +1\}$ corresponds to binary classification, or $\mathcal{Y} = \{0, \dots, k-1\}$ to k-class classification). For a population distribution p, the setting is as follows:

- At train time, we observe samples $(x_1, \dots, x_n) \sim p$.
- A test sample is generated by first drawing $(x, y) \sim p$, and replacing x with $\bar{x}$ such that $d(x, \bar{x}) \leq \epsilon$ for some distance d. We then observe $\bar{x}$ and hope to output y.

This can be placed into our previous framework by letting $\tilde{p} = p$ and saying that $D(p^*, \tilde{p}) \leq \epsilon$ if there is a coupling π from $\tilde{p}$ to p^* such that $d(x, \bar{x}) \leq \epsilon$ and $y = y'$ almost surely for $(x, y, \bar{x}, y') \sim \pi$. In the language of Wasserstein distance, this says that $W_{d,\infty}(\tilde{p}, p^*) \leq \epsilon$ (here we define $W_{c,k}(p, q) = \inf_{\pi \in \Pi(p,q)} \mathbb{E}_{(z,z') \sim \pi}[c(z, z')^k]^{1/k}$).

However, there are some important differences from the setting from before:

- We generally think of $p = \tilde{p}$ as the "nice" distribution and p^* as the "ugly" distribution.
- Since p^* is the "ugly" distribution, we make no distributional assumptions $\mathcal{G}$ about p^*.
- On the other hand, we also generally consider much "smaller" perturbations than in the previous case. For instance, $d(x, \bar{x}) \leq \epsilon$ should imply that x and $\bar{x}$ are close enough that they have the same label y.

A final point is that the worst-case test loss is directly observable, at least given infinitely many samples from p. Indeed, if the (non-robust) loss is

$$L_0(p, \theta) = \mathbb{E}_{(x,y) \sim p}[\ell(\theta; x, y)], \tag{1}$$

then the robust loss is

$$L(p, \theta) = \mathbb{E}_{(x,y) \sim p}\Big[\sup_{\bar{x} : d(x, \bar{x}) \leq \epsilon} \ell(\theta; \bar{x}, y)\Big]. \tag{2}$$

(The worst-case test loss was also observable for train-time corruptions, but it involved a much more complex supremum and depended on $\mathcal{G}$.)

Consequently, given samples $(x_1, y_1), \dots, (x_n, y_n)$, a natural estimator is

$$\hat{\theta} = \arg\min_{\theta} \rho(\theta) + \frac{1}{n} \sum_{i=1}^{n} \sup_{d(x_i, \bar{x}_i) \leq \epsilon} \ell(\theta; \bar{x}_i, y_i), \tag{3}$$

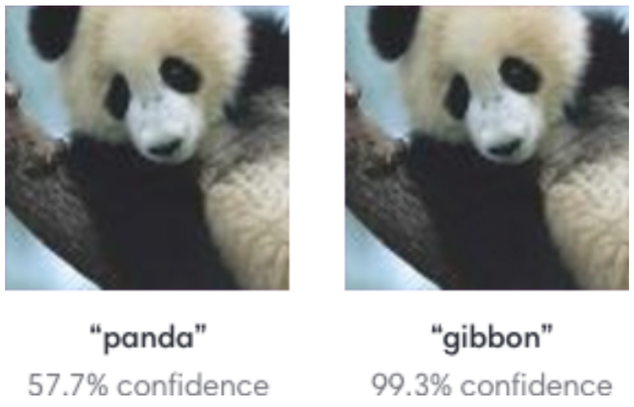
where $\rho(\theta)$ is a regularizer and the other term is an empirical estimate of the worst-case loss. Indeed, this is essentially the estimator we will consider throughout. However, there are several issues to address:

- How well does $\hat{\theta}$ generalize from the training points to the population distribution?
- The supremum over $d(x, \bar{x}) \leq \epsilon$ is generally computationally intractable. What happens if we only approximate this supremum, and what are good approximation schemes?
- How robust are results to the choice of d itself?

Answering these questions will involve a mix of empirical data and theoretical analysis.

1.1 Getting Oriented: Examples of d and basic empirical results

A basic motivating example takes x to be an image containing some object y. Thus for instance $x \in [0,1]^{224\times224\times3}$ is a 150528-dimensional vector (corresponding to a 224×224 RGB image), and y is one of k different object classes (in the most popular computer vision benchmark, ImageNet, we have $k = 1000$). We then take $d(x, \bar{x}) = \|x - \bar{x}\|_\infty$, meaning that we are allowed to perturb each pixel by at most ϵ. A striking phenomenon is that regularly-trained machine learning models are often non-robust at even very small values of ϵ. For instance, the following two images have different classifications under a neural network:

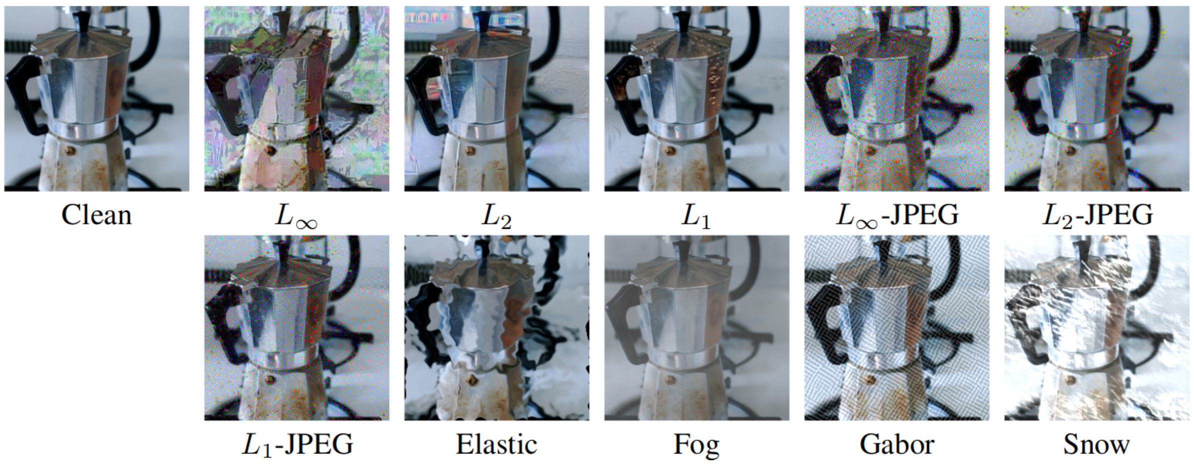


This corresponds to a perturbation in ℓ_∞ of $\epsilon = 0.007$. This is not an isolated phenomenon, and many machine learning models can be successfully adversarially perturbed at small ϵ for most of their inputs.

We could consider many perturbation measures beyond ℓ_∞:

- $d(x, \bar{x}) = \|x\bar{x}\|_\infty$: the maximum amount any pixel changes
- $d(x, \bar{x}) = \|x - \bar{x}\|_1$: the total amount that the pixels change
- $d(x, \bar{x}) = \|\mathsf{JPEG}(x) - \mathsf{JPEG}(\bar{x})\|_2$: the ℓ_2-distance in wavelet space
- The amount of "stretch" needed to move x to $\bar{x}$
- The amount of "fog", "snow", etc. needed to move x to $\bar{x}$.

These and others are illustrated below:



Note that for many of these, d is not a metric (it isn't even symmetric, e.g. for snow).

There are four important phenomena that we will elaborate on below:

- Adversarial perturbations have a much larger effect than random perturbations.

- Adversarial robustness is difficult to measure and most papers get it wrong enough that the results are meaningless.
- Robustness for one choice of d only partially generalizes to other choices of d.
- Minimizing the robust loss seems to increase overfitting for neural networks.

Worst-case vs. random perturbations. As a simplified case suppose that we are doing binary classification and our model is linear: $f_\theta(x) = \mathrm{sign}(\langle \theta, x \rangle)$. We then consider two types of perturbations:

- Random Gaussian perturbations: $\bar{x} = x + z$, where $z \sim \mathcal{N}(0, \sigma^2 I)$ (standard deviation σ per coordinate).
- Worst-case ℓ_2-perturbations: $\bar{x} = x + z$, where $\|z\|_2 \leq \sigma\sqrt{d}$.

Note that the ℓ_2-norm of the perturbation is approximately the same in both cases. For a point x, define the *margin* $\gamma_x = |\langle \theta, x \rangle| / \|\theta\|_2$. How big does the margin need to be for the output f_θ to be robust to perturbations?

For the Gaussian perturbations, we have $\langle \theta, \bar{x} \rangle = \langle \theta, x \rangle + \sigma \|\theta\|_2 \mathcal{N}(0, 1)$, so the output is robust as long as $\gamma_x \gg \sigma$. However, for worst-case ℓ_2-perturbations we have $\langle \theta, \bar{x} \rangle = \langle \theta, x \rangle + \langle \theta, z \rangle$, which in the worst-case is $\langle \theta, x \rangle \pm \sigma\sqrt{d}\|\theta\|_2$. Thus we are only robust to worst-case perturbations when $\gamma_x > \sigma\sqrt{d}$. There is a $\sqrt{d}$ factor difference between random and worst-case perturbations! In fact, a Gaussian perturbation with *per-coordinate* magnitude σ is most similar to a worst-case perturbation with *overall* magnitude σ.

Adversarial robustness is difficult to measure. To measure adversarial robustness, we seek to compute

$$\ell_{\mathsf{robust}}(\theta; x, y) = \sup_{\bar{x} : d(x, \bar{x}) \leq \epsilon} \ell(\theta; \bar{x}, y) \tag{4}$$

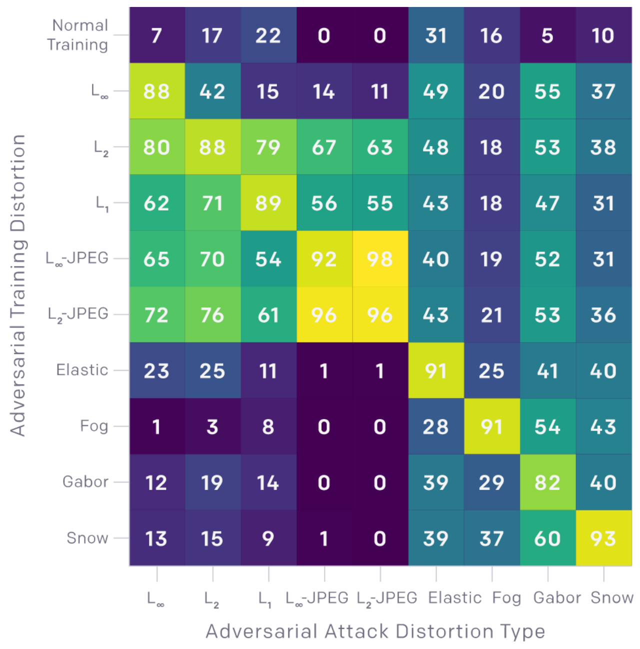
for a given x and y, which is computationally intractable in general. For instance, ℓ is often the loss function for a neural network, which is highly non-convex not just in θ but also in x (which is the relevant variable for our purposes). One idea would be to approximate the supremum via some local search algorithm (e.g. projected gradient descent). This gives us some proxy loss $\ell_{\mathsf{proxy}} \leq \ell_{\mathsf{robust}}$.

The issue arises when we take the natural step of optimizing ℓ_{proxy} to try to find a robust model θ. Since ℓ_{proxy} underestimates ℓ_{robust}, this highly incentivizes finding θ where ℓ_{proxy} is a bad approximation to ℓ_{robust}! For instance, if our proxy loss is obtained via gradient descent, it incentivizes θ for which gradient descent on $\bar{x}$ doesn't converge well. Even if we don't explicitly optimize ℓ_{proxy}, if we design models with ℓ_{proxy} in mind we as the researcher might accidentally find ways to make ℓ_{proxy} but not ℓ_{robust} small. And since ℓ_{proxy} is the only thing we measure, we have no straightforward way of even noticing this!

I estimate that around 95% of papers on test-time robustness succumb to this issue, where the proxy loss measured in the paper is small but the true robust loss is large. Some good papers that discuss this are **?** and **?**, which both also provide best practices for evaluating robustness. (See also **?** for an earlier paper making similar points for the detection of adversarial perturbations.) Some example best practices are:

- If your proxy is based on some optimization algorithm, give the optimizer much more power at evaluation time than at training time (e.g. use 10 gradient descent steps at train time but 1000 at test time).
- Ensure that your proxy works well by making sure that when ϵ is large enough that accuracy should be zero, the proxy also gives an accuracy of zero.
- Make sure that other correlated measures (such as robustness to random perturbations) are also improving. This is more controversial because not everyone agrees on what measures should correlate with adversarial robustness.

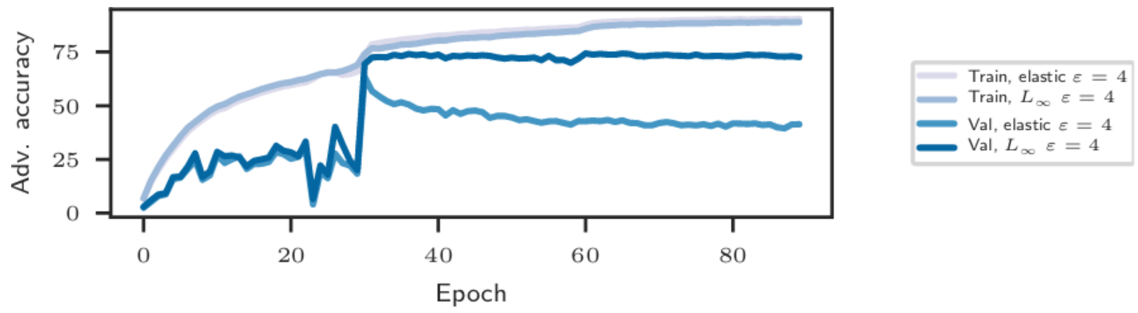
Robustness to choice of d. While this is a nuanced story, the zeroth order summary is that robust for fairly different choices of d is positively correlated at low ϵ, but eventually becomes negatively correlated at high ϵ (although aiming for robustness under the wrong d often still gives better results than doing nothing). Here is a grid illustrating this:



Adversarial Training Distortion \ Adversarial Attack Distortion Type	L_∞	L_2	L_1	L_∞-JPEG	L_2-JPEG	Elastic	Fog	Gabor	Snow
Normal Training	7	17	22	0	0	31	16	5	10
L_∞	88	42	15	14	11	49	20	55	37
L_2	80	88	79	67	63	48	18	53	38
L_1	62	71	89	56	55	43	18	47	31
L_∞-JPEG	65	70	54	92	98	40	19	52	31
L_2-JPEG	72	76	61	96	96	43	21	53	36
Elastic	23	25	11	1	1	91	25	41	40
Fog	1	3	8	0	0	28	91	54	43
Gabor	12	19	14	0	0	39	29	82	40
Snow	13	15	9	1	0	39	37	60	93

Here each row is a model that was trained to be robust to a certain type of perturbation, and each column is a measure of the robustness against a perturbation (higher is better). We will discuss in the next section how to actually perform this training. We see that even for fairly perturbations as different as elastic warping and snow, training against one perturbation helps with the other one. See **?** for a more detailed investigation of this.

Robustness hurts generalization. Training a model to be robust, especially at large ϵ, can harm generalization performance. This becomes even more pronounced if we allow multiple perturbations, for instance if d allows both elastic warping and ℓ_∞ perturbations, we see overfitting even at a moderately-sized $\epsilon = 4$ (this is $4/255$ so approximately 0.016):



We see similar phenomena for a single perturbation, but need ϵ to be larger (e.g. around $32/255$ for ℓ_∞, which is large enough to be hard for humans).