CIS 7000-005: Debugging Data & Models Fall 2022

Adversarial — September 15

Prof. Eric Wong

Biases and distribution shift are often discussed in the context of naturally occurring changes. This is usually referred to as the *average* case. But how bad can this get in the *worst* case?

- Where can changes be adversarial?
- How do we make adversarial changes?
- What is the difference between adversarial and average case?

1 Adversarial changes

Adversarial changes are not necessarily that different from biases and distribution shifts. In fact, adversarial settings can often share the same settings. The key change in the adversarial regime is that instead of average case performance (i.e. over a distribution sh ift or a set of biases), we instead look at worst case performance.

1.1 Per-example changes

Adversarial examples are those that maximize the loss of a model subject to some input constraint.

$$x_{adv} = \max_{\delta \in \mathcal{B}(x)} \ell(f(x), y) \tag{1}$$

The ball of local changes determines the type of adversarial example and is often referred to as the *threat model*. For example

1. Noise model: $\mathcal{B}(x) = \{x + \delta : \|\delta\|_p \leq \epsilon$. Usually invisible noise for $p \in \{1, 2, \infty\}$.
2. Patch model: $\mathcal{B}(x) = \{x + p : p$ is a patch$\}$. Unconstrained changes within a region of fixed contiguous size for images
3. Substitutions: $\mathcal{B}(x) = \{y : x$ is a synonym of $y\}$. Word substitutions with synonyms for text.

Finding an adversarial example. Let's first consider the noise model in the vision setting. Earliest way to find an adversarial example is FGSM. It is a one-step, first order approximation to the adversarial example optimization problem. For ℓ_∞ noise, this is

$$\delta_{fgsm} = \epsilon \cdot \text{sign}(g) \tag{2}$$

where $g = \nabla_\delta \ell(f(x + \delta), y)$. This is also known as steepest descent, i.e.

$$\epsilon \cdot \text{sign}(g) = \text{argmax}_{\|v\| \leq \epsilon} v^T g \tag{3}$$

Typically we also clip the input to be the range of real pixels, i.e. $[0, 1]$.

Multi-step attack We can run several of these iterations with smaller step sizes. Since these iterates can leave the original ball $\mathcal{B}(x)$, we also project back onto this space (projected steepest descent), i.e.

$$\delta^{t+1} = \delta^t + \text{Proj}_{\mathcal{B}(x)}(\delta^t + \alpha \cdot \text{sign}(\nabla_\delta \ell(f(x^t + \delta), y))) \tag{4}$$

for some step size α. This is usually much more effective in practice. The projection step for the ℓ_∞ ball with radius ϵ is simply to clamp the inputs at $[-\epsilon, \epsilon]$. This can be implemented in a few lines of code:

```
def pgd_linf(model, X, y, epsilon, alpha, num_iter):
    """ Construct FGSM adversarial examples on the examples X"""
    delta = torch.zeros_like(X, requires_grad=True)
    for t in range(num_iter):
        loss = nn.CrossEntropyLoss()(model(X + delta), y)
        loss.backward()
        delta.data = (delta + alpha*delta.grad.detach().sign()).clamp(-epsilon,epsilon)
        delta.grad.zero_()
    return delta.detach()
```

Perceptual threat model

$$\mathcal{B}(x) = \{x' : d(x, x') \leq \epsilon\} \tag{5}$$

where $d(x, x') = \|\phi(x) - \phi(x')\|_2$, ϕ is the activations of a neural network.

Patch attacks Patch attacks have two main components: the patch itself and the location of the patch. Typically the patch itself can be optimized directly similarly like the PGD attack. However, location is more difficulty:

$$\max_{i,j} \ell(f(x + \text{patch}_{ij}, y) \tag{6}$$

Several strategies:

- Random search
- Numerical gradient optimization

1.2 White vs black box attacks

What can you do with query access only?

- Train surrogate model and transfer
- Random search
- Localized random search (square attack)

1.3 Distributional changes

2 References