Deep adversarial training

Raphael Chinchilla

March 2020

1 Topology of a Neural Network

The task of training a Neural Network is mathematically expressed as solving

$$\min_w \mathbb{E}_{x,y\sim\mathcal{P}}[c(y, h(w, x))] \tag{1}$$

where $h(w, x)$ is a Neural Network and $c(\cdot)$ is a cost function. A Neural Network is described as a recurrent function: Let n_ℓ be the ℓth layer of neurons, represented as a column vector, with $n^{(0)} := x$, then

$$n_\ell = f_\ell(w_\ell\, n_{\ell-1}) \tag{2}$$

where w_ℓ is a weights matrix which can represent either a fully connected layer or a convolution layer and $f(\cdot)$ is a non-linear function, generally an activation function and possibly a maxpool layer. The last layer is given by

$$n_L = s(w_L\, n_{L-1}) \tag{3}$$

where $s(\cdot)$ is a function such as the softmax.

2 Shallow adversarial training

Classical adversarial training, which we call "Shallow adversarial training", consists of preparing the neural network to attacks by finding the worst perturbation that could be used given some budget. This is normally formulated as

$$\min_w \mathbb{E}_{x,y\sim\mathcal{P}}\left[\max_{d\in\mathcal{D}} c(y, h(w, x + d))\right] \tag{4}$$

where $\mathcal{D}$ is a bounded set such as $\|d\|_p < \epsilon$, $p \in \{1, 2, +\infty\}$. We can generalize (4) by writing

$$\min_w \mathbb{E}_{x,y\sim\mathcal{P}}\left[\max_{d} c(y, h(w, x + d)) + \rho(d)\right]. \tag{5}$$

where $\rho(\cdot)$ is penalization function. If we take $\rho(\cdot) = \log \mathcal{I}_{\mathcal{D}}(\cdot)$, where $\mathcal{I}_{\mathcal{D}}(\cdot)$ is the indicator function of the domain $\mathcal{D}$, then we retrieve the criteria from (4). But one could also chose for instance $\rho(d) = -\lambda\|d\|^2$ with $\lambda > 0$.

3 Deep adversarial training

We call the previous adversarial training strategy of shallow because, in practice, it is only preparing for attacks on the input layer. However, an attack on the input propagates through the network. In deep adversarial training, we also prepare the intermediate layers against attacks. Not only such attack is more powerful and generalizes shallow attacks, it is also not reproducible by an adversary. This means the neural network would be trained against an much stronger attacker than it could encounter.

Formally, a neural network $h(w, x, d)$ subject to deep adversarial attacks is defined by the recursive relation

$$n_\ell = f_\ell(w_\ell\, n_{\ell-1}) + d_\ell \tag{6}$$

The deep adversarial training is defined by

$$\min_w \mathbb{E}_{x,y\sim\mathcal{P}}\left[\max_d c(y, h(w, x, d)) + \rho(d)\right]. \tag{7}$$

We are particularly interested in functions $\rho(\cdot)$ that which is on stages related, *i.e.* that can be written as

$$\min_w \mathbb{E}_{x,y\sim\mathcal{P}}\left[\max_d c(y, h(w, x, d)) + \sum_{\ell=0}^{L-1}\rho_\ell(d_\ell)\right]. \tag{8}$$

One might think that computing the adversarial attack

$$\max_d c(y, h(w, x, d)) + \sum_{\ell=0}^{L-1}\rho_\ell(d_\ell) \tag{9}$$

is significantly harder than the regular shallow attack. We will show that this is actually not the case. The first step is to see that we can rewrite (9) using equality constraints for the recursive relation that defines the neural network such as

$$\begin{aligned}\max_{d_{0:L-1}, n_{0:L-1}} \quad & c(y, s(w_L, n_{L-1})) + \sum_{\ell=0}^{L-1}\rho_\ell(d_\ell)\\ \text{s.t.} \quad & n_\ell = f_\ell(w_\ell, n_{\ell-1}) + d_\ell\\ & n_0 = x + d_0\end{aligned} \tag{10}$$

The relevant aspect is that solving (10) is equivalent to solving

$$\max_{n_{0:L-1}} \; c(y, s(w_L, n_{L-1})) + \rho_0(n_0 - x) + \sum_{\ell=1}^{L-1}\rho_\ell(n_\ell - f_\ell(w_\ell, n_{\ell-1})). \tag{11}$$

The key advantage of rewriting the problem in such a way is that we no longer need to enforce the equality constraints in (10), or, equivalently, the recursive relation in equations (6). Instead, we can maximize directly with respect to the neurons, without needing to be concerned with the network aspect of the neural network.

The equivalence between (11) and (9) relies on computing a very good approximate of the maximum at each iteration. In order to address this, one needs an efficient algorithm to compute the maximum. We are interested in two types of functions $\rho_\ell(\cdot)$. First, consider de case is when $\rho_\ell(\cdot) = -\lambda_\ell\|\cdot\|^2$, where λ_ℓ is a positive constant. Because a neural network is roughly a linear operation,

$$-\lambda_0\|n_0 - x\|^2 - \sum_{\ell=1}^{L-1}\lambda_\ell\|n_\ell - f_\ell(w_\ell, n_{\ell-1})\|^2$$

will be roughly a quadratic cost. By choosing to use the Conjugate Gradient (CG) algorithm, it is possible to obtain an approximate maximum of (11) with very few iterations, as CG is the most efficient first order method to solve a quadratic equation.

The second case is when $\rho_\ell(\cdot) = \log \mathcal{I}_{\mathcal{D}}(\cdot)$. Let us assume $\mathcal{D} = \{d \in \mathbb{R}^{n_\ell} : g(d) \leqslant 0\}$ for some function $g : \mathbb{R}^{n_\ell} \to \mathbb{R}^{n_\ell}$. In this case, we can approximate $\rho_\ell(d)$ by $\hat{\rho}_\ell(d) = -\lambda_\ell(\text{relu}(g(d)))^2$ with λ_ℓ a large positive number. In this case, our reasoning about the cost function being approximately quadratic is still valid.

We summarize the reasoning of this section in Algorithm 1

Algorithm 1 Deep adversarial attack

for each epoch **do**
  for each batch in the training set **do**
    Sample a batch with elements $x(0), x(1), \ldots, x(B)$
    for b from 0 to B **do**
      Initialize $d(b)$ randomly
      Initialize the value of the neurons $n(b)$ by feeding $x(b)$ to the neural network $h(w, x(b), d(b))$.
      Use a Conjugate Gradient algorithm to solve

$$n(b)^* \in \arg\max_{n_{0:L-1}} c(y, s(w_L, n_{L-1})) + \rho_0(n_0 - x(b)) + \sum_{\ell=1}^{L-1} \rho_\ell(n_\ell - f_\ell(w_\ell, n_{\ell-1}))$$

      Compute the optimal disturbances $d(b)^*$ corresponding to $n(b)^*$ using (6)
    end for
    Update the value of the weights using, for instance, Stochastic Gradient Descent

$$w = w - \gamma \sum_{b=1}^{B} \frac{\mathrm{d}c(y, h(w, x(b), d(b)))}{\mathrm{d}w}$$

  end for
end for