Adversarial robustness with partial isometry

1 Motivations

The method presented in this document is an improvement of the *Parseval networks* method introduced in [1]. More precisely, I want to get rid of any regularization method since it seems to be very unstable. Also, I want to avoid at all cost backpropagating the gradients *of the gradients* which is what I was doing when using the Jacobian matrix in the loss function. PyTorch does not like that at all, and I have weird errors during backpropagation leading to *nan* loss. So let's implement a small "improvement" of Parseval networks and hope that it will lead to experimental improvements.

Let $f : \mathcal{X} \to \mathbb{R}^c$ be a neural network, where $\mathcal{X}$ is a compact set[1] of $\mathbb{R}^d$. For $x \in \mathcal{X}$ the output $z = f(x) \in \mathbb{R}^c$ are the logits. We assume that $d > c$ where c is the number of classes. We note $\sigma : \mathbb{R}^c \to \Delta^c$ the softmax function where we removed the last component, such that $\sigma(z) \in \mathbb{R}^{c-1}$ are the parameters of a categorical distribution. Δ^c is the probability simplex with c vertices.

The main idea of [1] is to constrain the Lipschitz constant of the neural network to be lower than 1. Since the Lipschitz constant of a composition of functions is upper-bounded by the product of the Lipschitz constants of each function, it is sufficient to constrain the Lipschitz constant of each layer to be lower than 1. In [1], the authors considered the Lipschitz constant Λ' of the log-likelihood loss function with respect to the Euclidean distance:

$$|l(f(x), y) - l(f(x'), y)| \leq \Lambda' \|x - x'\|_2.$$

Here, we see $\sigma(f(x))$ as a categorical distribution, and we consider the Lipschitz constant Λ between the Euclidean distance on $\mathcal{X}$, and the Fisher-Rao distance d on Δ^c:

$$d(\sigma(f(x)), \sigma(f(x'))) \leq \Lambda \|x - x'\|_2.$$

As in [1], we aim for the stronger condition of constraining every layer to have Lipschitz constant lower than 1. The input space $\mathcal{X}$ as well as every hidden space will be endowed with the Euclidean distance. The output space Δ^c is endowed with the Fisher information metric (FIM) which induces the Fisher-Rao distance.

2 Lipschitz constant of hidden layers

A ResNet is composed of several types of layers. We will give the Lipschitz constant when both the input and the ouput of the layer are endowed with the Euclidean distance:

- **Linear layer** (i.e., fully-connected). If W is the weight matrix then the Lipschitz constant is the spectral norm $\|W\|_2$ of the W, i.e., the largest singular value of W.
- **Convolutional layer**. They are also linear thus they can be written in the form $x_{l+1} = W.U(x_l)$ where W is an *unfolded* weight matrix and U is the *unfolding operator*. This simply corresponds to reorganizing the inputs and the weights. The Lipschitz constant of the unfolding operator is $\sqrt{k}$ where k is the width of the convolution kernel. Thus the Lispchitz constant of a convolutional layer is $\sqrt{k}\|W\|_2$.
- **ReLU activation**. It's Lipschitz constant is lower than 1.
- **Max pooling layer**. It's Lipschitz constant is lower than 1.

[1] Because a C^1 function on a compact set is Lipschitz continuous.

- **Batch normalization**. I guess its the inverse of the standard deviation?? So, no guarantee to have Lipschitz constant smaller than 1? Let's not use batch normalization for now ...
- **Residual connection**. A sum may have Lipschitz constant larger than 1. However, a convex sum has Lisphitz constant lower than 1. In [1], the authors proposed to learn the coefficients of the convex sum. But I guess we can also divide by the number of terms to obtain a convex sum?

We can see that the only layers that don't have Lipschitz constant lower than 1 by construction are the linear and convolutional layers. As in [1], we constrain these layers to have Lipschitz constant (approximately) equal to 1 by constraining the singular values to all be equal to 1. Let $h(z) = Wz$ be a linear layer. Assume that the dimension of $h(z)$ is smaller than the dimension of z. Then the singular values of W are all equal to 1 if and only if the rows of W are orthogonal, or equivalently the columns of W are a Parseval tight frame. This condition can be written $WW^T = I$. If the dimension of $h(z)$ is larger than the dimension of z, then we enforce the condition $W^TW = I$.

The idea is approximately enforce this condition after n SGD updates (in [1], we have $n = 1$). This can be done by minimizing the following loss:

$$R(W) = \beta \|WW^T - I\|_2^2.$$

where β is a hyperparameter that can be interpreted as a regularization coefficient, or a learning rate. The gradient of $R(W)$ is:

$$\nabla_W R(W) = 4\beta(WW^T - I)W.$$

We can minimize $R(W)$ with:

$$\begin{aligned} W_{k+1} &\leftarrow W_k - 4\beta(W_kW_k^T - I)W_k \\ W_{k+1} &\leftarrow (1 + 4\beta)W_k - 4\beta W_kW_k^TW_k. \end{aligned}$$

Povey et al. claimed (without proof and without any reference) that $\beta = 1/8$ would ensure quadratic convergence.

3 The final layer

No we consider the case of the final fully-connected layer followed by softmax:

$$\theta(z) = \sigma(Wz).$$

To ensure that $\theta(z)$ has Lipschitz constant (approximately) equal to 1, it is sufficient to constrain $\theta(z)$ to be a partial isometry. Let G be the FIM on Δ^c using the standard coordinates. We have:

$$G_{ij}(\theta) = \frac{\delta_{ij}}{\theta^i} + \frac{1}{1 - \sum_{k=1}^{c-1} \theta^k}.$$

Then, the pullback of the FIM is $\tilde{G} = W^T J_\sigma^T G J_\sigma W$ where J_σ is the Jacobian matrix of the softmax:

$$J_{\sigma,ij} = \sigma^i(z)(\delta_{ij} - \sigma^j(z)).$$

As already shown in a previous document, we consider the coordinate change, for $i \in \{1, \dots, c-1\}$:

$$\tau^i(\theta) = \frac{2\sqrt{\theta^i}}{1 - \sqrt{1 - \sum_{k=1}^{c-1} \theta^k}}.$$

In these new coordinates, the matrix of the FIM becomes:

$$G_{ij}(\tau(\theta)) = \kappa(\theta) I_{c-1},$$

where:

$$\kappa(\theta) = \frac{4}{\left(1 + \left\|\frac{\tau^k(\theta)}{2}\right\|_2^2\right)^2}.$$

After some computation we get:

$$\kappa(\theta) = 4\left(1 - \sqrt{\theta^c}\right)^2,$$

where $\theta^c = 1 - \sum_{k=1}^{c-1} \theta^k$. Let J_τ be the Jacobian matrix of τ:

$$\begin{aligned} J_{\tau,ij} &= \frac{1}{1-\sqrt{\theta^c}}\left(\frac{\delta_{ij}}{\sqrt{\theta^i}} - \frac{\sqrt{\theta^i}}{\sqrt{\theta^c}\left(1-\sqrt{\theta^c}\right)}\right) \\ &= \frac{\tau^i(\theta)}{2}\left(\frac{\delta_{ij}}{\theta^i} - \frac{\tau^i(\theta)}{2\sqrt{\theta^i\theta^c}}\right). \end{aligned}$$

Then, the pullback of the FIM can be written:

$$\tilde{G} = \kappa W^T J_\sigma^T J_\tau^T J_\tau J_\sigma W.$$

The partial isometry condition is:

$$J_\tau J_\sigma W W^T J_\sigma^T J_\tau^T = \frac{1}{\kappa} I_{c-1}.$$

Thus, using the same update rule as the previous section and writing $J = J_\tau J_\sigma$, we obtain:

$$W_{k+1} \leftarrow W_k - 4\beta J^T \left(J W_k W_k^T J^T - \frac{1}{\kappa} I_{c-1}\right) J W_k.$$

4 Other approach

We may try to implement the regularization term with the Jacobian matrix *but without computing the Jacobian matrix*, in order to avoid the "double backpropagation".

I'm not sure that it makes sense, but we may approximate the Jacobian matrix by multiplying all the weight matrices. This should yield some kind of "upper-bound" of the true Jacobian matrix, since the other layers (ReLU, max pooling) will just "zeroing" some elements.

There may be issue with layers such as bach normalization, average pooling, residual connections. Average pooling and residual connection should be modified to be convex sums. I don't know how to deal with batch normalization. Maybe, remove them altogether. Or we may keep the bias, but fix the weight to 1, or apply a sigmoid to the weights (such that it is smaller than 1).

Such a regularization term (i.e., the product of all weight matrices) will be fast to compute and easy to differentiate wrt the weights.

References

[1] M. Cissé, P. Bojanowski, E. Grave, Y. N. Dauphin, and N. Usunier, "Parseval Networks: Improving Robustness to Adversarial Examples," in *Proceedings of the 34th International Conference on Machine Learning*, pp. 854–863, PMLR, 2017.