Mollifying Networks

Caglar Gulcehre[1] Marcin Moczulski[2] Francesco Visin[3] Yoshua Bengio [1]

[1]University of Montreal,

[2]University of Oxford,

[3]Politecnico di Milano

ICLR,2017
Presenter: Arshdeep Sekhon

Motivation

A DNNs: highly non-convex nature of loss function

B A number of recently proposed methods to make optimization easier:

1. curriculum learning
2. training RNNs with diffusion
3. noise injection

Mollification for Neural Networks

1. novel method for training neural networks
2. A sequence of optimization problems of increasing complexity, where the first ones are easy to solve but only the last one corresponds to the actual problem of interest.

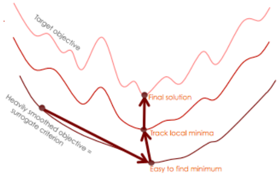

Figure: *

3. The training procedure iterates over a sequence of objective functions starting from the simpler ones i.e. with a smoother loss surface and moving towards more complex ones until the last, original, objective function is reached

Mollifiers

1. To smooth the loss function $\mathcal{L}$, parametrized by $\theta \in \mathbb{R}^n$ by convolving it with another function $K(\cdot)$ with stride $\tau \in \mathbb{R}^n$

$$\mathcal{L}_K(\theta) = \int_{-\infty}^{\infty} (\mathcal{L}(\theta - \tau)K(\tau))(d\tau) \tag{1}$$

2. Many choices for K but must be a mollifier

Mollifier

1. A mollifier is an infinitely differentiable function that behaves like an approximate identity in the group of convolutions of integrable functions.
2. If K() is an infinitely differentiable function, that converges to the Dirac delta function when appropriately rescaled and for any integrable function $\mathcal{L}$, then it is a mollifier

Mollifier

$$\mathcal{L}_K(\theta) = (\mathcal{L} * K)(\theta) \tag{2}$$

$$\mathcal{L}_K(\theta) = \lim_{\epsilon \to 0} \int_{-\infty}^{\infty} \epsilon^{-n} K(\frac{\tau}{\epsilon}) \mathcal{L}(\theta - \tau) d\tau \tag{3}$$

Mollifiers: Gradients

1. gradients of the mollified loss:

$$\nabla_\theta \mathcal{L}_K(\theta) = \nabla_\theta(\mathcal{L} * K)(\theta) = \mathcal{L} * \nabla(K)(\theta) \tag{4}$$

2. How does this $\nabla_\theta \mathcal{L}_K(\theta)$ relate to $\nabla_\theta \mathcal{L}(\theta)$?
3. Use weak gradients

Weak Gradients

1. For an integrable function $\mathcal{L}$ in space $\mathcal{L} \in L([a, b]), g \in L([a, b]^n$ is an n-dimensional weak gradient of $\mathcal{L}$ if it satisfies:

$$\int g(\tau)K(\tau)d\tau = -\int \mathcal{L}(\tau)\nabla K(\tau)d\tau \quad (5)$$

where $K(\tau)$ is an infinitely differentiable function vanishing at infinity, $C \in [a, b]^n$ and $\tau \in \mathbb{R}^n$

Mollified Gradients

$$\int g(\tau)K(\tau)d\tau = -\int \mathcal{L}(\tau)\nabla K(\tau)d\tau \tag{6}$$

$$\nabla_\theta \mathcal{L}_K(\theta) = \nabla_\theta(\mathcal{L} * K)(\theta) = \mathcal{L} * \nabla(K)(\theta) \tag{7}$$

$$\nabla_\theta \mathcal{L}_K(\theta) = \int \mathcal{L}(\theta - \tau)\nabla K(\tau)d\tau \tag{8}$$

$$\nabla_\theta \mathcal{L}_K(\theta) = -\int g(\theta - \tau)K(\tau)d\tau \tag{9}$$

For a differentiable almost everywhere function $\mathcal{L}$, the weak gradient $g(\theta)$ is equal to $\nabla_\theta \mathcal{L}$ almost everywhere

$$\nabla_\theta \mathcal{L}_K(\theta) = -\int \nabla_\theta \mathcal{L}(\theta - \tau)K(\tau)d\tau \tag{10}$$

weight noise methods: Gaussian Mollifiers

1. Use a gaussian mollifier $K(\cdot)$:
   1. infinitely differentiable
   2. a sequence of properly rescaled Gaussian distributions converges to the Dirac delta function
   3. vanishes in infinity

$$\nabla_\theta \mathcal{L}_{K=\mathcal{N}}(\theta) = -\int \nabla_\theta \mathcal{L}(\theta - \tau) p(\tau) d\tau \tag{11}$$

$$\nabla_\theta \mathcal{L}_{K=\mathcal{N}}(\theta) = \mathbb{E}[\nabla_\theta \mathcal{L}(\theta - \tau)] \tag{12}$$

$\tau \mathcal{N}(0, \boldsymbol{I})$

$$\nabla_\theta \mathcal{L}_{K=\mathcal{N}}(\theta) = -\int \nabla_\theta \mathcal{L}(\theta - \tau) p(\tau) d\tau \tag{13}$$

2. sequence of mollifiers indexed by ϵ

$$\nabla_\theta \mathcal{L}_{K=\mathcal{N}}(\theta) = -\int \nabla_\theta \mathcal{L}(\theta - \tau) \epsilon^{-1} p(\frac{\tau}{\epsilon}) d\tau \tag{14}$$

$\tau \mathcal{N}(0, \epsilon^2 \boldsymbol{I})$

weight noise methods: Gaussian Mollifiers

$$\nabla_\theta \mathcal{L}_{K=\mathcal{N}}(\theta) = -\int \nabla_\theta \mathcal{L}(\theta - \tau)\epsilon^{-1} p(\frac{\tau}{\epsilon}) d\tau \tag{15}$$

$\tau \mathcal{N}(0, \epsilon^2 \boldsymbol{I})$

$$\nabla_\theta \mathcal{L}_{K=\mathcal{N},\epsilon}(\theta) = \mathbb{E}_\tau[\nabla_\theta \mathcal{L}(\theta - \tau)] \tag{16}$$

$\tau \mathcal{N}(0, \epsilon^2 \boldsymbol{I})$

This satisfies the property:

$$\lim_{\epsilon \to 0} \nabla_\theta \mathcal{L}_{K=\mathcal{N},\epsilon}(\theta) = \nabla_\theta \mathcal{L}(\theta) \tag{17}$$

weight noise methods: Gaussian Mollifiers

$$\mathcal{L}_K(\theta) = \int (\mathcal{L}(\theta - \xi)K(\xi))(d\xi) \tag{18}$$

By monte carlo estimate:

$$\approx \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}(\theta - \xi^i) \tag{19}$$

$$\frac{\partial \mathcal{L}_K(\theta)}{\partial \theta} \approx \frac{1}{N}\sum_{i=1}^{N} \frac{\partial \mathcal{L}_K(\theta - \xi^i)}{\partial \theta} \tag{20}$$

Therefore introducing additive noise to the input of $\mathcal{L}(\theta)$ is equivalent to mollification.

Using this mollifier for neural networks

$$\boldsymbol{h}^l = f(\boldsymbol{W}^l \boldsymbol{h}^{l-1}) \tag{21}$$

$$\boldsymbol{h}^l = f((\boldsymbol{W}^l - \xi^l)\boldsymbol{h}^{l-1}) \tag{22}$$

$\xi^l \mathcal{N}(\mu, \sigma^2)$

Generalized Mollifiers

Generalized Mollifier

A generalized mollifier is an operator, where $T_\sigma(f)$ defines a mapping between two functions, such that $T_\sigma : f \to f^*$:

$$\lim_{\sigma \to 0} T_\sigma f = f \tag{23}$$

$$f^0 = \lim_{\sigma \to \infty} T_\sigma f$$

$$\text{is an identity function} \tag{24}$$

$$\frac{\partial T_\sigma f(x)}{\partial x} exists \; \forall x, \sigma > 0$$

Noisy Mollifier

Noisy Mollifier

A stochastic function $\phi(x, \xi_\sigma)$ with input x and noise ξ is a noisy mollifier if its expected value corresponds to the application of a generalized mollifier T_σ

$$(T_\sigma f)(x) = \mathbb{E}[\phi(x, \xi_\sigma)] \tag{25}$$

1. When $\sigma = 0$ no noise is injected and therefore the original function will be optimized.
2. If $\sigma \to \infty$ instead, the function will become an identity function

Method: Mollify the cost of an NN

1. During training minimize a sequence of increasingly complex noisy objectives $\{\mathcal{L}^1(\theta, \xi_{\sigma_1}), \mathcal{L}^2(\theta, \xi_{\sigma_2}), \cdots, \mathcal{L}^k(\theta, \xi_{\sigma_k})\}$ by annealing the scale (variance) of the noise σ_i
2. algorithm satisfies the fundamental properties of the generalized and noisy mollifiers

Algorithm

1. start by optimizing a convex objective function that is obtained by configuring all the layers between the input and the last cost layer to compute an identity function, {by skipping both the affine transformations and the blocks followed by nonlinearities.}

Algorithm

1. start by optimizing a convex objective function that is obtained by configuring all the layers between the input and the last cost layer to compute an identity function, {by skipping both the affine transformations and the blocks followed by nonlinearities.}
2. During training, the magnitude of noise which is proportional to p is annealed, allowing to gradually evolve from identity transformations to linear transformations between the layers.

Algorithm

1. start by optimizing a convex objective function that is obtained by configuring all the layers between the input and the last cost layer to compute an identity function, {by skipping both the affine transformations and the blocks followed by nonlinearities.}
2. During training, the magnitude of noise which is proportional to p is annealed, allowing to gradually evolve from identity transformations to linear transformations between the layers.
3. Simultaneously, as we decrease the p, the noisy mollification procedure allows the element-wise activation functions to gradually change from linear to be nonlinear
4. Thus changing both the shape of the cost and the model architecture

Feedforward Networks

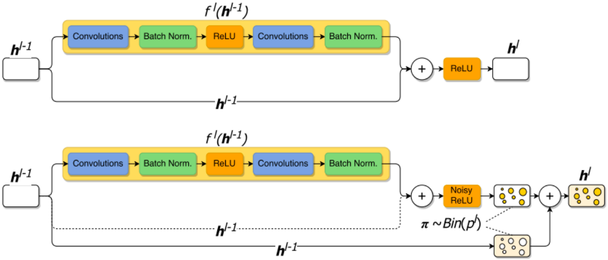

Figure: *

$$\tilde{\mathbf{h}}^l = \psi(\mathbf{h}^{l-1}, \boldsymbol{\xi}; \mathbf{W}^l)$$
$$\phi(\mathbf{h}^{l-1}, \boldsymbol{\xi}, \boldsymbol{\pi}^l; \mathbf{W}^l) = \boldsymbol{\pi}^l \odot \mathbf{h}^{l-1} + (1 - \boldsymbol{\pi}^l) \odot \tilde{\mathbf{h}}^l$$
$$\mathbf{h}^l = \phi(\mathbf{h}^{l-1}, \boldsymbol{\xi}, \boldsymbol{\pi}^l; \mathbf{W}^l).$$

Linearizing the Network

1. adding noise to the activation function: may suffer from excessive random exploration when the noise is very large
2. Solution: bounding the element-wise activation function $f(\cdot)$ with its linear approximation when the variance of the noise is very large, after centering it at the origin
3. f^* is bounded and centered at the origin

Linearizing the Network

Figure: *

$$\psi(x_i, \xi_i; \mathbf{w}_i) = \mathrm{sgn}(\mathbf{u}^*(x_i))\mathrm{min}(|\mathbf{u}^*(x_i)|, |\mathbf{f}^*(x_i) + \mathrm{sgn}(\mathbf{u}^*(x_i))|s_i||) + \mathbf{u}(0)$$

Figure: *

u(x) is the first order Taylor approximation of the original activation function around zero and u*(x) stands for the centered u(x) which is obtained by shifting u(x) towards the origin.

Algorithm

Algorithm 1 Activation of a unit i at layer l.

1: $x_i \leftarrow \mathbf{w}_i^\top \mathbf{h}^{l-1} + b_i$ ▷ an affine transformation of $\mathbf{h}^{l-1}$
2: $\Delta_i \leftarrow \mathrm{u}(x_i) - \mathrm{f}(x_i)$ ▷ Δ_i is a measure of a saturation of a unit
3: $\sigma(x_i) \leftarrow (\mathrm{sigmoid}(a_i \Delta_i) - 0.5)^2$ ▷ std of the injected noise depends on Δ_i
4: $\xi_i \sim \mathcal{N}(0,\ 1)$ ▷ sampling the noise from a basic Normal distribution
5: $s_i \leftarrow p^l\, c\, \sigma(x_i)|\xi_i|$ ▷ Half-Normal noise controlled by $\sigma(x_i)$, const. c and prob-ty p^l
6: $\psi(x_i, \xi_i) \leftarrow \mathrm{sgn}(\mathrm{u}^*(x_i))\min(|\mathrm{u}^*(x_i)|,\ |\mathrm{f}^*(x_i) + \mathrm{sgn}(\mathrm{u}^*(x_i))|s_i||) + \mathrm{u}(0)$ ▷ noisy activation
7: $\pi_i^l \sim \mathrm{Bernoulli}(p^l)$ ▷ p^l controls the variance of the noise AND the prob of skipping a unit
8: $\tilde{h}_i^l = \psi(x_i, \xi_i)$ ▷ $\tilde{h}_i^l$ is a noisy activation candidate
9: $\phi(\mathbf{h}^{l-1}, \xi_i, \pi_i^l; \mathbf{w}_i) = \pi_i^l h_i^{l-1} + (1 - \pi_i^l)\tilde{h}_i^l$ ▷ make a HARD decision between h_i^{l-1} and $\tilde{h}_i^l$

Figure: *

Annealing schedules for p

a different schedule for each layer of the network, such that the noise in the lower layers will anneal faster.

1. Exponential Decay

$$p_t^l = 1 - e^{-\frac{kv_t l}{tL}} \tag{26}$$

2. Square root decay

$$min(p_{min}, 1 - \sqrt{\frac{t}{N_{epochs}}}) \tag{27}$$

3. Linear decay

$$min(p_{min}, 1 - \frac{t}{N_{epochs}}) \tag{28}$$

Experiments: Deep Parity,CIFAR

Figure 8: The learning curves of a 6-layers MLP with sigmoid activation function on 40 bit parity task.

	Test Accuracy
Stochastic Depth	93.25
Mollified Convnet	92.45
ResNet	91.78

Table 1: CIFAR10 deep convolutional neural network.

Figure: *

Different Annealing Schedules

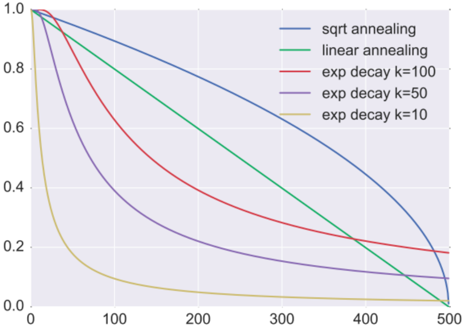

Figure: *