NLP 202: Training Deep Neural Networks

Jeffrey Flanigan

Winter 2023

University of California Santa Cruz
jmflanig@ucsc.edu

Plan

Tricks for training neural networks

- Issues with training NNs
- Initialization
- Normalization
- Other tricks: residual connections, gradient clipping, curriculum learning

Issues with training neural networks

When training NNs, the objective function (loss function):

- Is **non-convex**
- Has **poor conditioning**
- Contains **flat spots** due to saturated activation functions
- May have **vanishing or exploding gradients**

Saturated Activation Functions

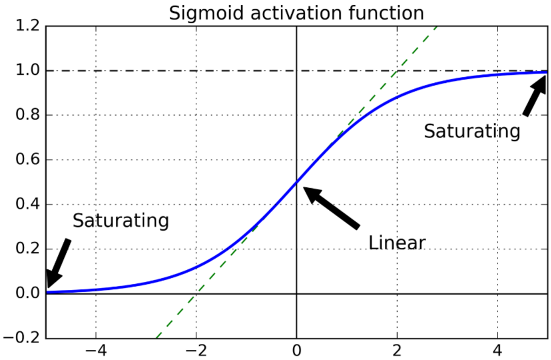


Figure 11-1. Logistic activation function saturation

$$f(x) = \frac{1}{1 + e^{-x}}$$

Leaky ReLU

The leaky rectified linear unit (leaky ReLU) avoids this problem

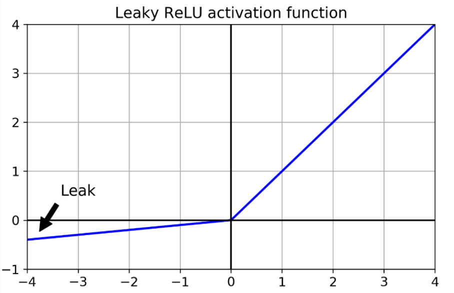


Figure 11-2. Leaky ReLU: like ReLU, but with a small slope for negative values

$$f(x) = \max(\alpha x, x)$$

α is a hyperparameter or can be learned (parametric leaky ReLU, PReLU)

Vanishing/exploding gradients

Consider many composed functions

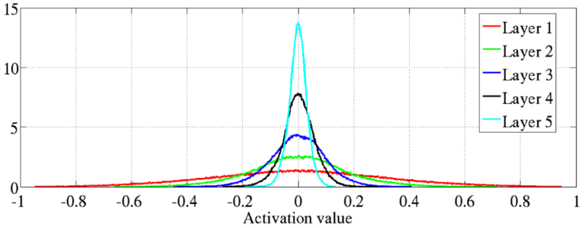
$$f(x) = g_1(g_2(\dots g_n(x)))$$

The partial derivative is:

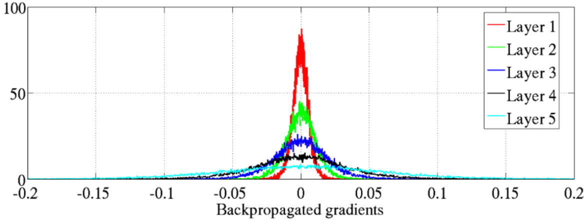
$$\frac{\partial f(x)}{\partial x} = \frac{\partial g_1}{\partial x} \dots \frac{\partial g_n}{\partial x}$$

Many partial derivatives multiplied together
$\Rightarrow$ could become very small (vanish) or very large (explode)

Activation and gradient values



Activation values with standard initialization, source: Xavier paper



Gradient values with standard initialization, source: Xavier paper

Layer 1 is first layer, layer 5 is top layer

Glorot (Xavier) Initialization

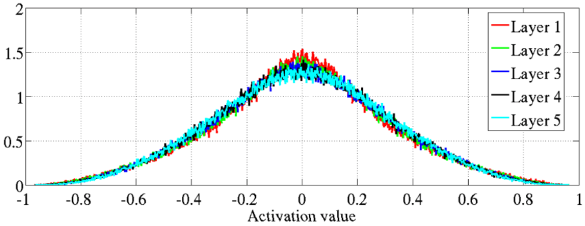
Right after initialization, we want

- For each layer, variance of the outputs to be equal to the variance of the inputs
- Not possible unless $fan_{in} = fan_{out}$
- Compromise: use $fan_{avg} = \frac{fan_{in} + fan_{out}}{2}$
- Glorot and Bengio analysed the Tahn function, and worked out the conditions

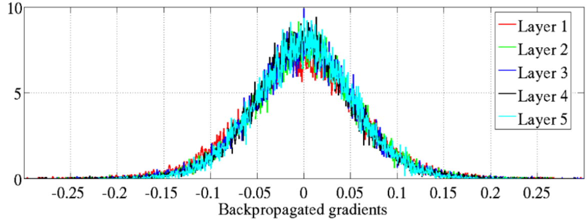
Glorot initialization:

Normal distribution with mean 0 and variance $\sigma^2 = \frac{1}{fan_{\text{avg}}}$

Or a uniform distribution between $-r$ and $+r$, with $r = \sqrt{\frac{3}{fan_{\text{avg}}}}$



Activation values with Xavier initialization, source: Xavier paper



Gradient values with Xavier initialization, source: Xavier paper

Glorot (Xavier) Initialization

TYPE	Shapeset	MNIST	CIFAR-10	ImageNet
Softsign	**16.27**	**1.64**	55.78	**69.14**
Softsign N	**16.06**	**1.72**	**53.8**	**68.13**
Tanh	27.15	**1.76**	55.9	70.58
Tanh N	**15.60**	**1.64**	**52.92**	**68.57**
Sigmoid	82.61	2.21	57.28	70.66

He (Kaiming) Initialization

- Extended Glorot's analysis for ReLU activation function
- Found normal distribution with mean 0 and variance $\sigma^2 = \frac{2}{fan_{in}}$
- For uniform $[-r, r]$ $r = \sqrt{\frac{6}{fan_{in}}}$

Initialization Strategies

Initialization	Activation functions	σ^2 (Normal)
Glorot	None, Tanh, logistic, softmax	$1/fan_{avg}$
He	ReLU and variants	$2/fan_{in}$
LeCun	SELU	$1/fan_{in}$

For uniform distribution $[-r, -r]$ use $r = \sqrt{3\sigma^2}$

Batch Normalization

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

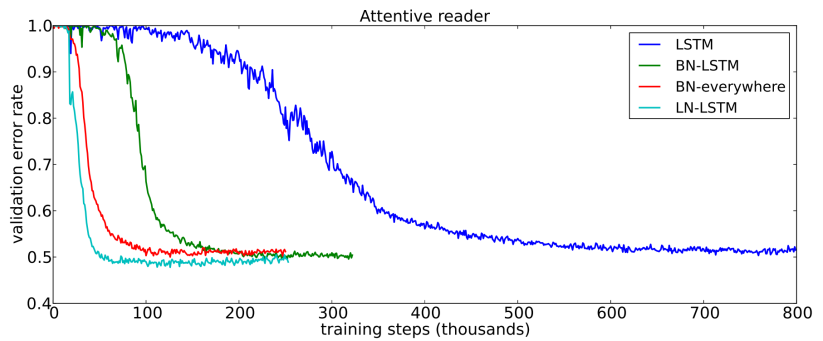
$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Layer Normalization

Rather than normalize activations in a minibatch, normalize activations in the layer.

For NLP, much more effective than batch normalization



Weight Normalization

Reparameterizes weights as

$$\mathbf{w} = \frac{g}{||\mathbf{v}||}\mathbf{v}$$

where g is a number.
Also combined with mean-only batch normalization (batch normalization for g).

Weight Normalization

Model	Test Error
Maxout [6]	11.68%
Network in Network [17]	10.41%
Deeply Supervised [16]	9.6%
ConvPool-CNN-C [26]	9.31%
ALL-CNN-C [26]	9.08%
our CNN, mean-only B.N.	8.52%
our CNN, weight norm.	8.46%
our CNN, normal param.	8.43%
our CNN, batch norm.	8.05%
ours, W.N. + mean-only B.N.	**7.31%**

Figure 2: Classification results on CIFAR-10 without data augmentation.

Architectural changes for avoiding vanishing gradients

- Non-saturating activation functions (Leaky ReLU, etc)
- RNNs: LSTM, GRU cell
- Residual connections (for either feedforward or RNNs)
- Also highway networks ("LSTM" for feedforward networks)

Residual Connections

Can avoid vanishing gradients by using **residual connections**.

Represent each layer in a FFNN or RNN cell as a function like this:

$$\mathbf{y} = \mathcal{F}(\mathbf{W}, \mathbf{x})$$

$\mathbf{x}$ is the input to the layer, $\mathbf{y}$ is the output of the layer

Example:

$$\mathbf{y} = \sigma(\mathbf{W}\mathbf{x})$$

Residual connections: add $\mathbf{x}$ (only works if $\mathbf{y}$ and $\mathbf{x}$ same dimension)

$$\mathbf{y} = \mathcal{F}(\mathbf{W}, \mathbf{x}) + \mathbf{x}$$

If different dimensions, can perform linear projection before adding

$$\mathbf{y} = \mathcal{F}(\mathbf{W}, \mathbf{x}) + \mathbf{W}_r\mathbf{x}$$

Gradient of residual connections

Consider many composed functions, this time with residual connections

$$f(x) = g_1(x) + g_2(g_1(x)) + \ldots + g_n(\ldots g_1(x))$$

The partial derivative is:

$$\frac{\partial f(x)}{\partial x} = \frac{\partial g_1}{\partial x} + \frac{\partial g_2}{\partial x}\frac{\partial g_1}{\partial x} + \ldots + \frac{\partial g_n}{\partial x} \ldots \frac{\partial g_1}{\partial x}$$

The gradient directly includes $\frac{\partial g_1}{\partial x}$ without being multiplied by other partial derivatives $\Rightarrow$ less likely to vanish

Compare to: Vanishing/exploding gradients

Consider many composed functions

$$f(x) = g_1(g_2(\dots g_n(x)))$$

The partial derivative is:

$$\frac{\partial f(x)}{\partial x} = \frac{\partial g_1}{\partial x} \dots \frac{\partial g_n}{\partial x}$$

Many partial derivatives multiplied together
$\Rightarrow$ could become very small (**vanish**) or very large (**explode**)

Gradient Clipping

To avoid exploding gradients clip the gradient

Regular clipping:

$$\hat{g}_i = \min(\alpha, \max(\alpha, g_i)) \in [-\alpha, \alpha]$$

Max-norm clipping:

$$\hat{\mathbf{g}} = \min(\alpha, ||\mathbf{g}||)\frac{\mathbf{g}}{||\mathbf{g}||} \Rightarrow ||\hat{\mathbf{g}}|| \leq \alpha$$

Curriculum Learning

Because we are learning with a non-convex objective, our path through the parameter space will effect the local min we reach.

Curriculum learning: guide the learning by training on easier examples first.

Other training tricks

- Regularization: dropout, weight decay (L2 regularization), early-stopping
- Label smoothing
- Subword units: byte-pair encoding (BPE), word piece, etc