Neural Networks and Deep Learning

Optimization

Nicholas Dronen

Department of Computer Science
dronen@colorado.edu

February 5, 2019

University of Colorado **Boulder**

Xavier Initialization

Heuristic: for each layer with n_{in} inputs and n_{out} outputs, set the weight from input i to input j as

- $w_{ji} \sim \text{Gaussian}\left(0, \frac{c^2}{n_{in}}\right)$ with $c = 1$ being sensible.
- second term is variance, so standard deviation is $n_{in}^{-0.5}$

Rationale

- If $x_i \in \{-1, 1\}$, then with this heuristic initialization, net input $z_j = \sum_i w_{ji} x_i \sim \text{Gaussian}\ (0, c^2)$
- Independent of network size
  - » Even when different layers have different fan-ins
  - » Even when you change the number of hidden units in your network.

Applies equally well if $x_i \in \{-\xi, \xi\}$

See: Glorot and Bengio, Understanding the difficulty of training deep feedforward neural networks, 2010

Xavier Initialization

For each layer with n_{in} inputs and n_{out} outputs, the weight from input i to input j should be set as

- $w_{ji} \sim \text{Gaussian}\left(0, \frac{c^2}{n_{out}+n_{in}}\right)$ or

  $w_{ji} \sim \text{Uniform}\left(-c\sqrt{\frac{6}{n_{out}+n_{in}}}, c\sqrt{\frac{6}{n_{out}+n_{in}}}\right)$

Rationale

- Previous heuristic scheme controls variance.
- Xavier initialization scheme aims to control both activation variance and gradient variance.

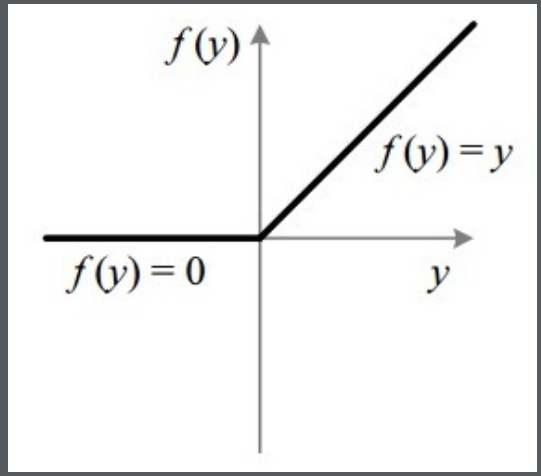
Initialization scheme will depend on activation functions you're using

- Most schemes are focused on logistic/sigmoid, tanh, softmax functions
- Exception: Delving Deeper into Rectifiers, He et al - Kaiming (or He) initialization

Rectified Linear Unit (ReLU)

- Activation function :
  $y = max(0, z)$
- Derivative :

$$\frac{\partial y}{\partial z} = \begin{cases} 0 & z \leq 0 \\ 1 & otherwise \end{cases}$$



Advantages

- Fast to compute activations and derivatives.
- No squashing of back propagated error signal as long as unit is activated.

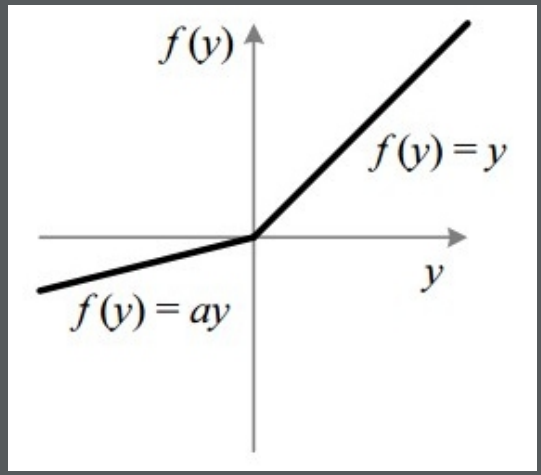
Disadvantages

- Can potentially lead to exploding gradients and activations.
- Units that are never activated above threshold won't learn.

Leaky ReLU

- Activation function :

$$y = \begin{cases} z & z > 0 \\ \alpha z & otherwise \end{cases}$$



- Derivative :

$$\frac{\partial y}{\partial z} = \begin{cases} 1 & z > 0 \\ \alpha & otherwise \end{cases}$$

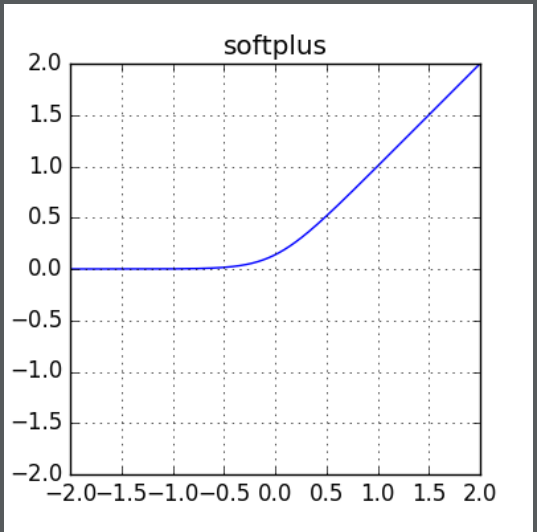
Reduces to standard ReLU if $\alpha = 0$
Trade off

- $\alpha = 0$ leads to ineffecient use of resources.
- $\alpha = 1$ lose non-linearity essential for interesting computation.
- See also: https://arxiv.org/abs/1502.01852, He et al - Parametric ReLU (PReLU)

SoftPlus

- Activation function :
  $y = ln(1 + e^z)$
- Derivative :

$$\frac{\partial y}{\partial z} = \frac{1}{1 + e^{-z}} = logistic(z)$$



Advantages

- Defined everywhere
- Zero only for z $\longrightarrow -\infty$

Exponential linear unit

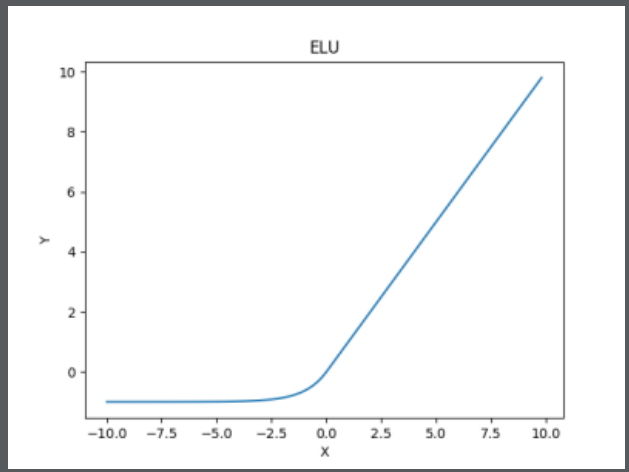
- Activation function :

$$y = \begin{cases} z & z > 0 \\ \alpha(e^z - 1) & otherwise \end{cases}$$

- Derivative :

$$\frac{\partial y}{\partial z} = \begin{cases} 1 & z > 0 \\ y + \alpha & otherwise \end{cases}$$

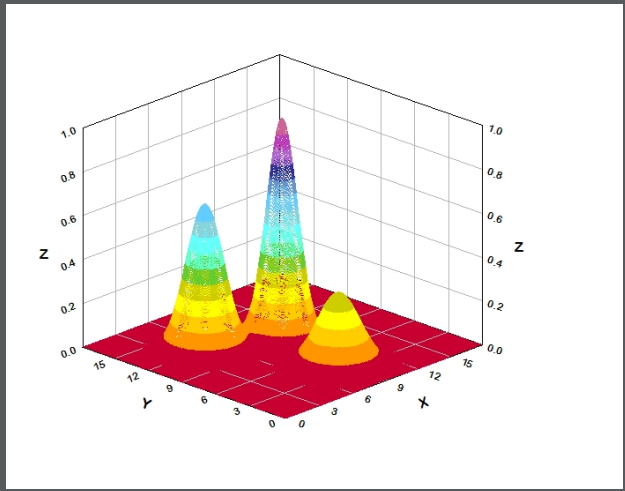
Reduces to standard ReLU if $\alpha = 0$



Radial Basis Function

- Activation function :

$$y = exp(-||x - w||^2)$$



Sparse Activation

- Many units just don't learn.
- Same issues as ReLU

Clever schemes to initialize weight

- E.g. set w near clusters of x's

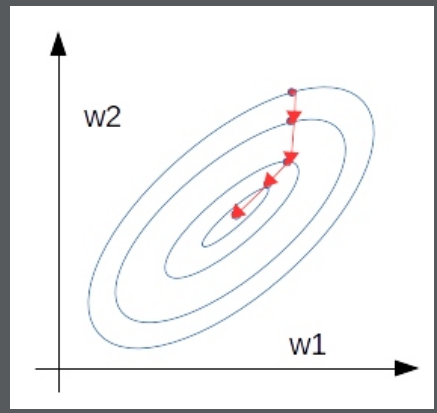
Input Normalization

What determines shape of the bowl?

$$
\begin{aligned}
E &= \sum_{\alpha} (y^{\alpha} - [w_1 x_1^{\alpha} + w_2 x_2^{\alpha}])^2 \\
&= \sum_{\alpha} (y - w_1 x_1 - w_2 x_2)(y - w_1 x_1 - w_2 x_2) \\
&= \sum_{\alpha} (y^2 - 2 y x_1 w_1 - 2 y x_2 w_2 + x_1^2 w_1^2 + x_2^2 w_2^2 + 2 x_1 x_2 w_1 w_2) \\
&= \sum_{\alpha} y^2 - 2 \sum_{\alpha} (y x_1) w_1 - 2 \sum_{\alpha} (y x_2) w_2 + \sum_{\alpha} (x_1^2) w_1^2 \\
&+ \sum_{\alpha} (x_2^2) w_2^2 + 2 \sum_{\alpha} (x_1 x_2) w_1 w_2
\end{aligned}
$$

Input Normalization

- Curvature (elongation) along w_1-axis is determined by $\sum_\alpha X_1^2$ (or, average value of X_1^2 across training inputs)
- Curvature (elongation) along w_2-axis is determined by $\sum_\alpha X_2^2$ (or, average value of X_2^2 across training inputs)



True whether w_1 and w_2 are in same or different layers.

- We want activations of the units in the input layer to have the same center and scale.
- We want activations of units in *different* layers to have the same center and scale.

Input Normalization

Sensible to have inputs normalized to mean 0 and standard deviation 1.

- $E[\hat{x_i}] = \frac{1}{p}\sum_{\alpha=1}^{p} \hat{x_{i,\alpha}} = 0$
  $\hat{x_{i,\alpha}}$: transformed input dimension i for training example α
- $E[\hat{x_i}^2] = \frac{1}{p}\sum_{\alpha=1}^{p} \hat{x_{i,\alpha}}^2 = 1$

To achieve this, compute mean μ_i and std dev σ_i for each input dimension i over training set.

- $\hat{x_{i,\alpha}} = \frac{x_{i,a} - \mu_i}{\sigma_i}$

For test set, need to apply same transformation.

- Use μ_i and σ_i from training set.
- They need to be stored along with network weights.

Batch Normalization (BN)

- When the input distribution to a learning system changes, it is said to experience covariate shift.
- BN is based on the premise that covariate shifts complicate the training of machine learning systems, and removing it from internal activations of the network may aid in training.
- Advantages
  - » Beneficial effect on gradient flow by reducing the dependence of gradients on the scale of the parameters or of their initial values. This allows the use of much higher learning rates without the risk of divergence.
  - » Claim: BN reduces the need for regularization

Batch Normalization

- Input

$$\text{Values of } x \text{ over a mini-batch} : x_i ... x_m$$

$$\text{Parameters to be learned} : \gamma, \beta$$

- Output

$$\text{Mini-batch mean} : \mu_B \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i$$

$$\text{Mini-batch variance} : \sigma_B^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2$$

$$\text{Normalize} : \hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}}$$

$$\text{Scale and shift} : y_i \leftarrow \gamma \hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$$

Squared Error Loss

Sensible regardless of output range and output activation function?

$$E = \frac{1}{2}\sum_j (y_j - \hat{y}_j)^2$$

$$\frac{\partial E}{\partial \hat{y}_j} = -(y_j - \hat{y}_j)$$

with logistic/sigmoid output unit,

$$\hat{y}_j = \frac{1}{1 + e^{-z_j}} \qquad \frac{\partial \hat{y}_j}{\partial z_j} = \hat{y}_j(1 - \hat{y}_j)$$

with tanh output unit,

$$\hat{y}_j = \frac{2}{1 + e^{-z_j}} - 1 \qquad \frac{\partial \hat{y}_j}{\partial z_j} = (1 + \hat{y}_j)(1 - \hat{y}_j)$$

Logistic/Sigmoid versus Tanh

- Output = .5
  - » when no input evidence, bias=0
- Will trigger activation in next layer
- Need large biases to neutralize
  - » biases on different scale than other weights
- Does not satisfy weight initialization assumption of mean activation = 0

- Output = 0
  - » when no input evidence, bias=0
- Won't trigger activation in next layer
- Don't need large biases
- Satisfies weight initialization assumption

Cross Entropy Loss

- Used when the target output represents a probability distribution
  - » E.g. a single output unit that indicates the classification decision (yes, no) for an input.
    Output $\hat{y} \in [0, 1]$ denotes Bernoulli likelihood of class membership.
    Target y indicates true class probability (typically 0 or 1)
    single value represents probability distributions over 2 alternatives
- Cross entropy, H, measures distance in bits from predicted distributions to target distribution.

$$E = H = -yln(\hat{y}) - (1 - y)ln(1 - \hat{y})$$

$$\frac{\partial E}{\partial \hat{y}} = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})}$$

Squared Error vs. Cross Entropy

$$\frac{\partial E_{sqerr}}{\partial y} = \hat{y} - y$$

$$\frac{\partial y}{\partial z} = \hat{y}(1 - \hat{y})$$

$$\frac{\partial E_{sqerr}}{\partial z} = (\hat{y} - y)\hat{y}(1 - \hat{y})$$

$$\frac{\partial E_{xentropy}}{\partial y} = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})}$$

$$\frac{\partial y}{\partial z} = \hat{y}(1 - \hat{y})$$

$$\frac{\partial E_{xentropy}}{\partial z} = \hat{y} - y$$

Essentially, cross entropy does not suppress learning when output is confident (near 0,1)

- net devotes its efforts to fitting target values exactly.
- consider situation where $\hat{y} = 0.99$ and y =1

Softmax

Each input can belong to one category

- $\hat{y}_j$ denotes the probability that the input's category is j

To interpret $\hat{y}$ as a probability distribution over the alternatives.

- $\sum_j \hat{y}_j = 1$ and $0 \leq \hat{y}_j \leq 1$

Activation function

- $\hat{y}_j = \frac{e^{z_j}}{\sum_k e^{z_k}}$
  - » exponentiation ensures non-negative values
  - » denominator ensures sum to 1

Derivatives of Categorical Outputs

For softmax output function

$$\hat{y}_j = \frac{e^{z_j}}{\sum_k e^{z_k}} \qquad z_j = \sum_i w_{ji} x_i$$

Weight update is same as for two category case when expressed in terms of $\hat{y}$

$$\Delta w_{ji} = \varepsilon \delta_j x_i$$

$$\delta_j = \frac{\partial E}{\partial \hat{y}_j} \hat{y}_j (1 - \hat{y}_j) \quad \text{for output unit}$$

$$= \left(\sum_k w_{kj} \delta_k \right) \hat{y}_j (1 - \hat{y}_j) \quad \text{for hidden unit}$$

Adagrad

- Initialization

$$\text{Gradient accumulation variable} : r = 0$$
$$\text{Learning rate} : \varepsilon$$
$$\text{Initial parameter} : \theta$$
$$\text{Constant for numerical stability} : \delta = 10^{-7}$$

- Update

$$\text{Compute gradient} : g$$
$$\text{Accumulate squared gradient} : r \leftarrow r + g \odot g$$
$$\text{Compute parameter update} : \Delta\theta \leftarrow \frac{-\varepsilon}{\delta + \sqrt{r}} \odot g$$
$$\text{Apply update} : \theta = \theta + \Delta\theta$$

RMSProp

- Initialization

$$\text{Accumulation variable} : r = 0$$
$$\text{Learning rate} : \varepsilon$$
$$\text{Decay rate} : \rho$$
$$\text{Initial parameter} : \theta$$
$$\text{Constant for numerical stability} : \delta = 10^{-7}$$

- Update

$$\text{Compute gradient} : g$$
$$\text{Accumulate squared gradient} : r \leftarrow \rho r + (1-\rho) g \odot g$$
$$\text{Compute parameter update} : \Delta\theta \leftarrow \frac{-\varepsilon}{\sqrt{\delta + r}} \odot g$$
$$\text{Apply update} : \theta = \theta + \Delta\theta$$

Adam

- Initialization

$$\begin{aligned}
\text{Step size} &: \varepsilon \\
\text{Exponential decay rates} &: \rho_1, \rho_2 \text{ in } [0,1) \\
\text{Constant for numerical stability} &: \delta = 10^{-7} \\
\text{Initial parameter} &: \theta \\
\text{Time step} &: t = 0 \\
\text{First and second moment variables} &: s = 0, r = 0
\end{aligned}$$

Adam

- Update

$$\text{Compute gradient} : g$$
$$\text{Increment time step} : t = t + 1$$
$$\text{Update biased first moment estimate} : s \leftarrow \rho_1 s + (1 - \rho_1) g$$
$$\text{Update biased second moment estimate} : r \leftarrow \rho_2 r + (1 - \rho_2) g \odot g$$
$$\text{Correct bias in first moment} : \hat{s} = \frac{s}{1 - \rho_1^t}$$
$$\text{Correct bias in second moment} : \hat{r} = \frac{r}{1 - \rho_2^t}$$
$$\text{Compute parameter update} : \Delta\theta \leftarrow -\varepsilon \frac{\hat{s}}{\delta + \sqrt{\hat{r}}}$$
$$\text{Apply update} : \theta = \theta + \Delta\theta$$

Setting the Learning Rate

Initial guess for learning rate

- If error doesnt drop consistently, lower initial learning rate and try again
- If error falls reliably but slowly, increase learning rate.
- Effective number of free parameters (model complexity) increases with training

Toward end of training

- Error will often jitter, at which point you can lower the learning rate down to 0 gradually to clean up weights

Remember, plateaus in error often look like minima

- Have some idea a priori how well you expect your network to be doing, and print statistics during training that tell you how well its doing
- Plot epochwise error as a function of epoch, even if you're doing minibatches

Setting the Learning Rate

Momentum

- $\Delta w_{t+1} = \theta \Delta w_t - (1 - \theta)\varepsilon \frac{\partial E}{\partial w_t}$

Adaptive and neuron-specific learning rates

- Observe error on epoch $t - 1$ and epoch t
- If decreasing, then increase global learning rate, ε_{global}, by an additive constant
- If increasing, decrease global learning rate by a multiplicative constant
- If fan-in of neuron j is f_j, then $\varepsilon_j = \frac{\varepsilon_{global}}{\sqrt{f_j}}$

When to Stop Training

Early stopping with a validation set.

- Hidden units all try to grab the biggest sources of error.
- As training proceeds, they start to differentiate from one another
- Effective number of free parameters (model complexity) increases with training

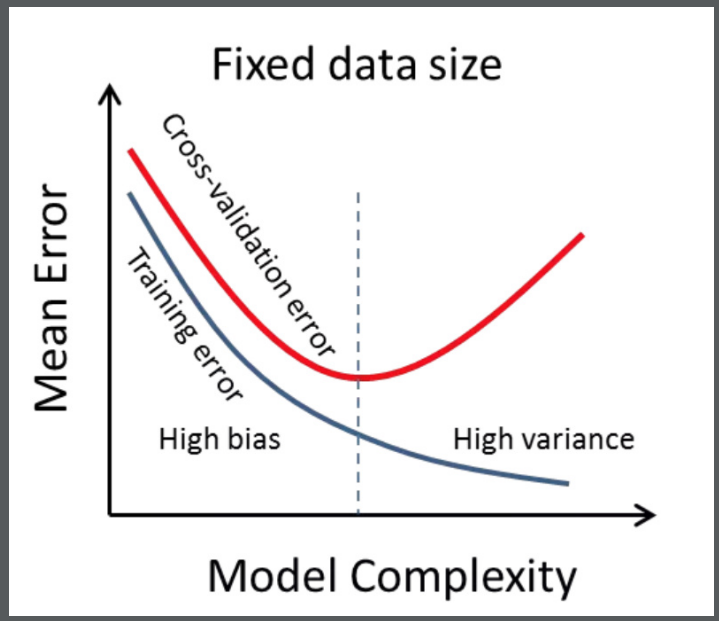


Figure source: https://www.fast.ai/

Challenges in Optimization

- Ill-conditioning
- Local Minima
- Plateaus, Saddle Points and Other Flat Regions
- Cliffs and exploding gradients
- Long-term dependencies
- Inexact gradients
- Poor correspondence between local and global structure
- Theoretical limits of optimization