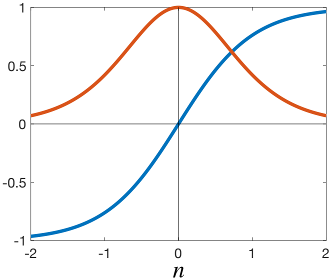
Supplemental Training Procedures

Deep Learning

Objectives

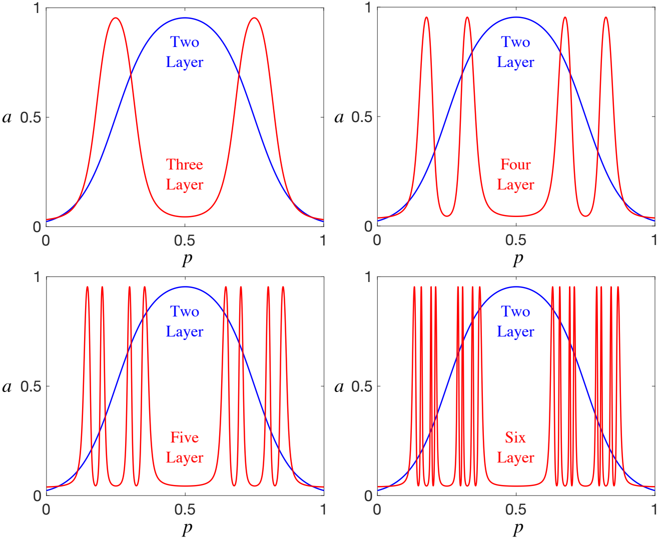
- The basic training methods, like steepest descent and Adam, can be supplemented for improved performance.
- The supplements described in this chapter serve two purposes.
- Speed up the convergence of training.
- Ensure that the final trained network will perform as well on new data as it did on the training data.

- Two things sped up training of deep networks.
  1. The introduction of cuda for programming GPUs.
  2. Overcoming the vanishing gradient problem.
- For deep networks with sigmoid activation, gradient can become very small.



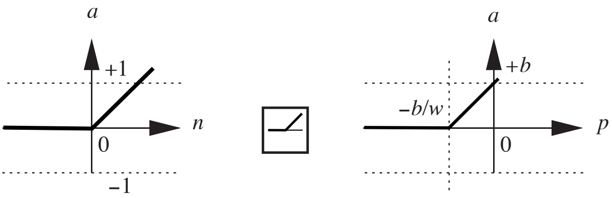
Tansig (tanh) activation function and its derivative

Cascading sigmoid layers



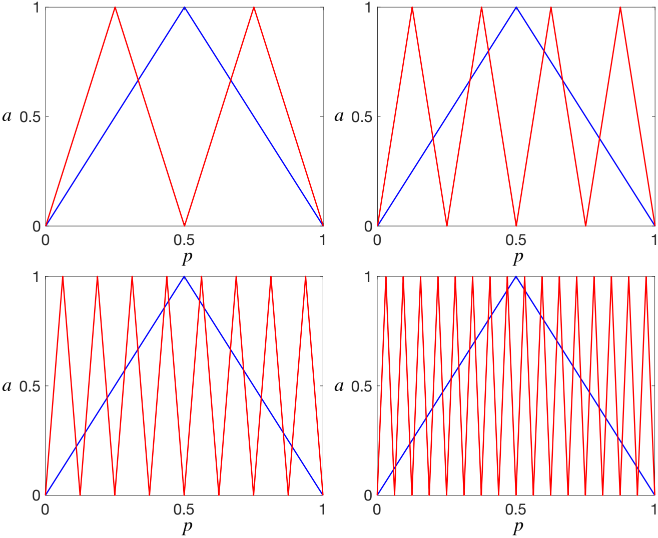
Derivative is close to zero over a wide region.

ReLU (or poslin) activation function



$a = poslin(n)$
Positive Linear Function

$a = poslin(wp + b)$
Single-Input *poslin* Neuron



Derivative is never zero.

Normalization at the input

- Sigmoid activations become saturated for $n > 3$ $(\exp(-3) \simeq 0.05)$.
- $n = wp + b$
- It is best to keep p in a standard range.

Normalize to [-1, 1]

$$\check{\mathbf{p}} = 2\left(\mathbf{p} - \mathbf{p}^{min}\right) \oslash \left(\mathbf{p}^{max} - \mathbf{p}^{min}\right) - 1$$

Normalize to zero mean and unity variance

$$\check{\mathbf{p}} = \left(\mathbf{p} - \mathbf{p}^{mean}\right) \oslash \mathbf{p}^{std}$$

Batch Normalization

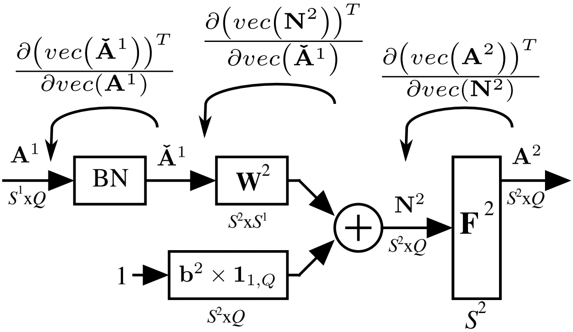
- We want to perform normalization at each layer.
- However, the range of inputs to layers after layer 1 change during training.
- The normalization can be done using batches of data.

$$
\begin{aligned}
\mathbf{a}^{mean} &= \frac{1}{Q}\sum_{q=1}^{Q} \mathbf{a}_q \\
\mathbf{a}^{std} &= \sqrt{\frac{1}{Q}\sum_{q=1}^{Q} \left(\mathbf{a}_q - \mathbf{a}^{mean}\right)^2 + \varepsilon} \\
\bar{\mathbf{a}}_q &= \left(\mathbf{a}_q - \mathbf{a}^{mean}\right) \oslash \mathbf{a}^{std} \\
\check{\mathbf{a}}_q &= \mathbf{w}^{bn} \circ \bar{\mathbf{a}}_q + \mathbf{b}^{bn}
\end{aligned}
$$

Backpropagating the batch

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_Q \end{bmatrix}$$

$$vec\left(\mathbf{A}\right) = \begin{bmatrix} \mathbf{a}_1^T & \mathbf{a}_2^T & \cdots & \mathbf{a}_Q^T \end{bmatrix}^T$$



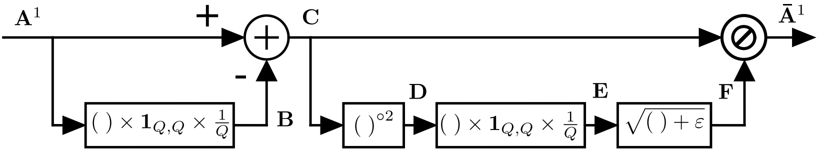
Backpropagating across the activation function and weight

The examples in the batch do not interact.

$$\frac{\partial \left(vec\left(\mathbf{N}^2\right)\right)^T}{\partial vec\left(\check{\mathbf{A}}^1\right)} = \begin{bmatrix} \left(\mathbf{W}^2\right)^T & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \left(\mathbf{W}^2\right)^T & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \left(\mathbf{W}^2\right)^T \end{bmatrix}$$

$$\frac{\partial \left(vec\left(\mathbf{A}^2\right)\right)^T}{\partial vec\left(\mathbf{N}^2\right)} = \begin{bmatrix} \dot{\mathbf{F}}^2\left(\mathbf{n}_1^2\right) & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \dot{\mathbf{F}}^2\left(\mathbf{n}_2^2\right) & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \cdots & \dot{\mathbf{F}}^2\left(\mathbf{n}_Q^2\right) \end{bmatrix}$$

- The examples in each minibatch will interact with each other.
- Break up batch norm into simpler pieces and backpropagate.



- The values for $\mathbf{w}^{bn}$ and $\mathbf{b}^{bn}$ at the end of training can be used for inference.
- During inference, we could have a batch of one, so standard deviation could not be computed.
- Batch norm must be operated with consistent mean and variance.
- One option is to use a smoothed version of the means and variances calculated during training.

$$\tilde{\mathbf{a}}_k^{mean} = \delta \cdot \tilde{\mathbf{a}}_{k-1}^{mean} + (1-\delta) \cdot \mathbf{a}_k^{mean}$$
$$\tilde{\mathbf{a}}_k^{std} = \delta \cdot \tilde{\mathbf{a}}_{k-1}^{std} + (1-\delta) \cdot \mathbf{a}_k^{std}$$

Batch norm summary

- Batch norm was developed to prevent vanishing gradient when using sigmoid activations.
- Normalization keeps inputs from saturating sigmoid functions.
- In practice, it also improves training even for ReLU activations.
- Random changes in scaling from minibatch to minibatch may improve generalization.
- A related technique is layer normalization. More effective than batch norm in recurrent networks.

Weight initialization

- Proper initialization of weights can prevent activation saturation.
- If initial weights are too large, the sigmoid can be saturated.
- Zero initial weights occur at a saddle point of the performance surface.
- The backpropagation steps multiply by weights at each layer. If all weights are small, the gradient will be close to zero.
- We want to find a happy medium for the weight values.

Weight initialization

- Approximate neuron output in linear region of sigmoid.

$$a_i^m \simeq \sum_{j=1}^{S^{m-1}} w_{i,j}^m a_j^{m-1} + b_i^m$$

- Set bias to zero, and assume activations from previous layer have mean zero and variance $\sigma_a^2(m-1)$.
- Assume initial weights are random values with mean zero and variance σ_w^2.
- Assume activations and weights are independent.

$$var\left(a_i^m\right) \equiv \sigma_a^2(m) = S^{m-1}\sigma_w^2\sigma_a^2(m-1)$$

- To maintain activation variance across layers:

$$\sigma_w^2 = \frac{1}{S^{m-1}}$$

Considering the backpropagation process

- The standard backpropagation is:

$$\mathbf{s}^{m-1} = \dot{\mathbf{F}}^{m-1}(\mathbf{n}^{m-1})(\mathbf{W}^{m})^{T}\mathbf{s}^{m}$$

- If we are in the linear activation range:

$$s_i^{m-1} \simeq \sum_{j=1}^{S^m} w_{j,i}^{m} s_j^{m}$$

- To maintain sensitivity variance across layers:

$$var\left(s_i^{m-1}\right) = \sigma_s^2(m-1) = S^m \sigma_w^2 \sigma_s^2(m)$$
$$\sigma_w^2 = \frac{1}{S^m}$$

- Trying to maintain forward and backward variance constant:

$$\sigma_w^2 = \frac{2}{S^{m-1} + S^m},$$ Xavier Initialization

- In deep networks,
  - Small weights can produce small gradients (vanishing gradient).
  - Large weights can produce large gradients (exploding gradient).
- Exploding gradient usually occurs in recurrent networks.
- Solution: scale the gradient to limit the norm.
- This is gradient clipping.

Improving generalization

- A network generalizes well if it performs as well on unseen data as on the training data.
- When a network overfits the training data, it will not generalize well.
- In early stopping, a validation set is removing from training. When validation error goes up, training stops.
- In regularization, a penalty term (usually sum of squared weights) is added to the performance function.
- In dropout a percentage of neurons in a selected layer are randomly deactivated at each iteration.
- Dropout is never used with batch normalization.