

# Supplemental Training Procedures

## Deep Learning

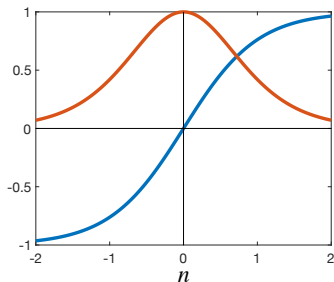


- The basic training methods, like steepest descent and Adam, can be supplemented for improved performance.
- The supplements described in this chapter serve two purposes.
- Speed up the convergence of training.
- Ensure that the final trained network will perform as well on new data as it did on the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



# Cause of slow training: vanishing gradient

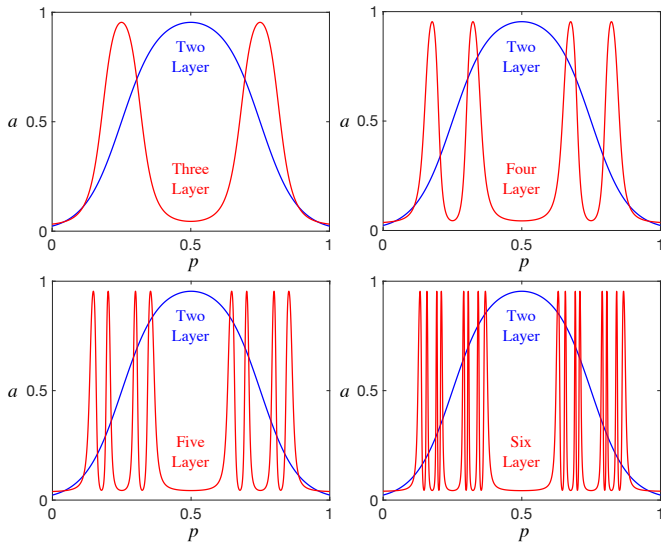
- Two things sped up training of deep networks.
  - ① The introduction of cuda for programming GPUs.
  - ② Overcoming the vanishing gradient problem.
- For deep networks with sigmoid activation, gradient can become very small.



Tansig (tanh) activation function and its derivative



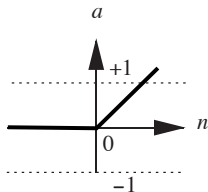
# Cascading sigmoid layers



Derivative is close to zero over a wide region.

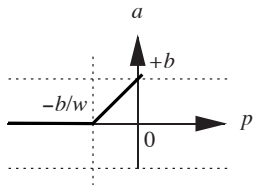


# ReLU (or poslin) activation function



$$a = \text{poslin}(n)$$

Positive Linear Function

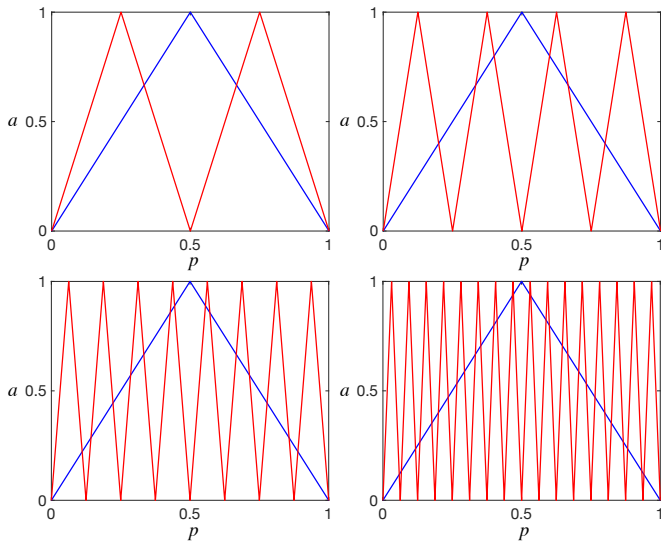


$$a = \text{poslin}(wp + b)$$

Single-Input *poslin* Neuron



# Cascading ReLU layers



Derivative is never zero.



- 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  $[-1, 1]$

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

Normalize to zero mean and unity variance

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



- 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.

$$\mathbf{a}^{mean} = \frac{1}{Q} \sum_{q=1}^Q \mathbf{a}_q$$

$$\mathbf{a}^{std} = \sqrt{\frac{1}{Q} \sum_{q=1}^Q (\mathbf{a}_q - \mathbf{a}^{mean})^2 + \varepsilon}$$

$$\bar{\mathbf{a}}_q = (\mathbf{a}_q - \mathbf{a}^{mean}) \oslash \mathbf{a}^{std}$$

$$\check{\mathbf{a}}_q = \mathbf{w}^{bn} \circ \bar{\mathbf{a}}_q + \mathbf{b}^{bn}$$

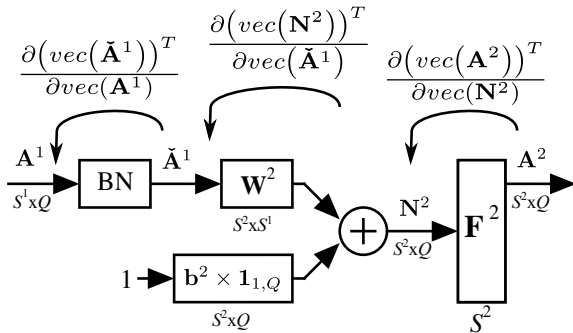




# Backpropagating the batch

$$\mathbf{A} = [\mathbf{a}_1 \quad \mathbf{a}_2 \quad \cdots \quad \mathbf{a}_Q]$$

$$\text{vec}(\mathbf{A}) = [\mathbf{a}_1^T \quad \mathbf{a}_2^T \quad \cdots \quad \mathbf{a}_Q^T]^T$$



# Backpropagating across the activation function and weight

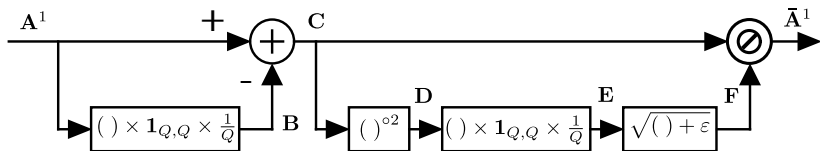
The examples in the batch do not interact.

$$\frac{\partial (\text{vec}(\mathbf{N}^2))^T}{\partial \text{vec}(\check{\mathbf{A}}^1)} = \begin{bmatrix} (\mathbf{W}^2)^T & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & (\mathbf{W}^2)^T & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & (\mathbf{W}^2)^T \end{bmatrix}$$
$$\frac{\partial (\text{vec}(\mathbf{A}^2))^T}{\partial \text{vec}(\mathbf{N}^2)} = \begin{bmatrix} \dot{\mathbf{F}}^2(\mathbf{n}_1^2) & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \dot{\mathbf{F}}^2(\mathbf{n}_2^2) & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & \dot{\mathbf{F}}^2(\mathbf{n}_Q^2) \end{bmatrix}$$



# Backpropagating across the normalization

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



## Batch norm during inference

- 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.

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



- 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.



- 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.



- 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  $\sigma_w^2$ .
- Assume activations and weights are independent.

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

- 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:

$$\begin{aligned} \text{var}(s_i^{m-1}) &= \sigma_s^2(m-1) = S^m \sigma_w^2 \sigma_s^2(m) \\ \sigma_w^2 &= \frac{1}{S^m} \end{aligned}$$

- Trying to maintain forward and backward variance constant:

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





- 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.

