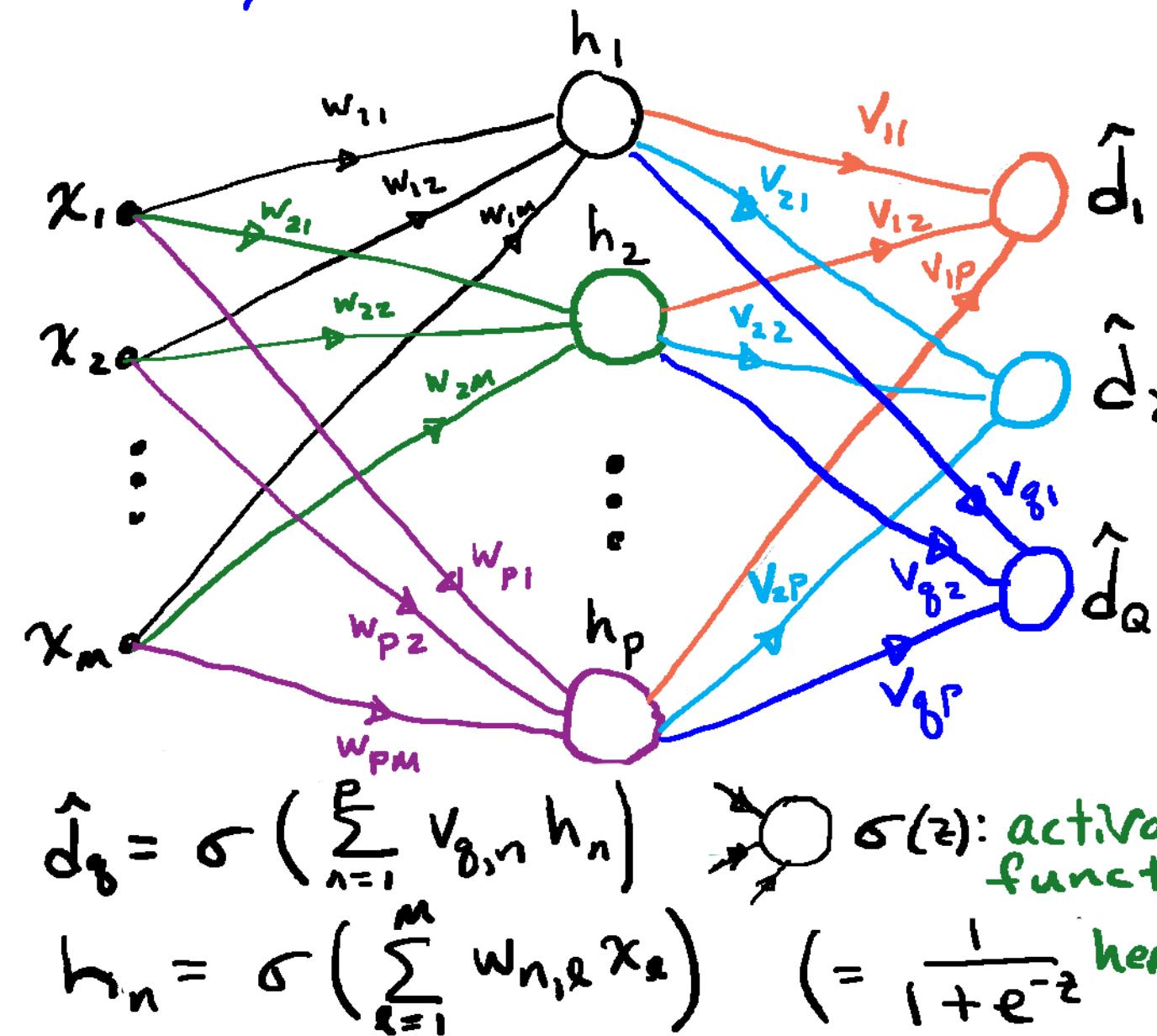


# The Backpropagation Algorithm for Training Neural Networks

# Objectives

- Method for learning weights in a neural network
- Apply stochastic gradient descent (SGD)
- Find gradients of squared error loss with respect to weights

Backpropagation uses SGD to train the weights <sup>2</sup>



$N$  training samples

$(x_1^i, x_2^i \dots x_m^i, d^i, i=1, 2, \dots, N)$

↑ # of samples  
↓ features of one sample

- 1) initialize  $w_{n,k}$ ,  $v_{q,n}$
- 2) for  $t=1, 2, 3, \dots$

Pick  $i_t \in \{1, 2, \dots, N\}$  randomly  
Use that sample to compute  $h_j^{i,t}$ ,  $\hat{d}_k^{i,t}$

$$v_{k,t}^{t+1} = v_{k,t}^t - \alpha_t \frac{\partial f^{i,t}}{\partial v_{k,t}}$$

$$w_{m,j}^{t+1} = w_{m,j}^t - \alpha_t \frac{\partial f^{i,t}}{\partial w_{m,j}}$$

Converged?

Gradients are obtained using the chain rule

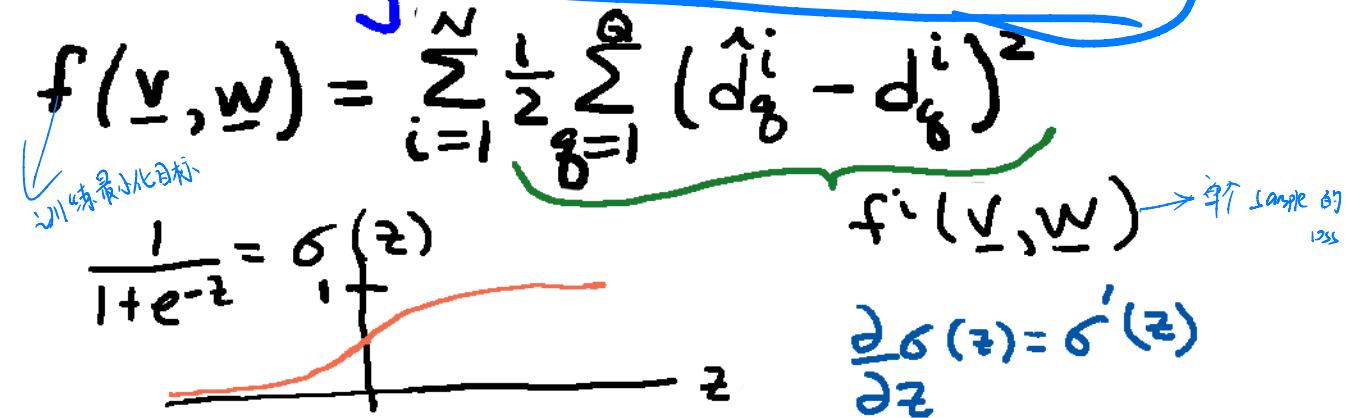
Squared error loss:

$$\frac{\partial f^i}{\partial v_{k,e}} : \left( \frac{\partial f^i}{\partial \hat{d}_k^i} \right) \frac{\partial \hat{d}_k^i}{\partial v_{k,e}}$$

$$\frac{\partial f^i}{\partial v_{k,e}} = (\hat{d}_k^i - d_k^i) \cdot \sigma' \left( \sum_{n=1}^p h_n^i v_{k,n} \right) \cdot \frac{\partial}{\partial v_{k,e}} \left( \sum_{n=1}^p h_n^i v_{k,n} \right)$$

$$\frac{\partial f^i}{\partial v_{k,e}} = (\hat{d}_k^i - d_k^i) \hat{d}_k^i (1 - \hat{d}_k^i) h_e^i$$

$$= \delta_k^i h_e^i$$



$$\sigma'(z) = \frac{1}{(1+e^{-z})^2} = \frac{1}{1+e^{-z}} \left( 1 - \frac{1}{1+e^{-z}} \right) \\ = \sigma(z) (1 - \sigma(z))$$

$$\text{But } \sigma \left( \sum_{n=1}^p h_n^i v_{k,n} \right) (1 - \sigma \left( \sum_{n=1}^p h_n^i v_{k,n} \right)) \\ = \hat{d}_k^i (1 - \hat{d}_k^i)$$

layer input

"error" at output

$$v_{k,e}^{t+1} = v_{k,e}^t - \alpha_t \delta_k^i h_e^i$$

Use a similar approach for  $w_{m,j}$

$$\frac{\partial f^i}{\partial w_{m,j}} = \sum_{g=1}^Q \frac{\partial f^i}{\partial \hat{d}_g^i} \cdot \frac{\partial \hat{d}_g^i}{\partial h_m^i} \frac{\partial h_m^i}{\partial w_{m,j}}$$

$\frac{\partial f^i}{\partial \hat{d}_g^i} = (\hat{d}_g^i - d_g^i)$

loss/output      output/hidden      hidden/prev. layer weights

$$\frac{\partial \hat{d}_g^i}{\partial h_m^i} = \frac{\partial}{\partial h_m^i} \sigma \left( \sum_{n=1}^P h_n^i v_{g,n}^i \right) = \sigma' \left( \sum_{n=1}^P h_n^i v_{g,n}^i \right) \cdot v_{g,m}^i = \hat{d}_g^i (1 - \hat{d}_g^i) v_{g,m}^i$$

$$\frac{\partial h_m^i}{\partial w_{m,j}} = \frac{\partial}{\partial w_{m,j}} \sigma \left( \sum_{k=1}^N w_{m,k} x_k^i \right) = \sigma' \left( \sum_{k=1}^N w_{m,k} x_k^i \right) x_j^i = h_m^i (1 - h_m^i) x_j^i$$

$$\frac{\partial f^i}{\partial w_{m,j}} = \sum_{g=1}^Q (\hat{d}_g^i - d_g^i) \hat{d}_g^i (1 - \hat{d}_g^i) v_{g,m}^i h_m^i (1 - h_m^i) x_j^i = \underline{\gamma_m^i x_j^i}$$

$$w_{m,j}^{t+1} = w_{m,j}^t - \alpha^t \gamma_m^i x_j^i$$

layer input      backpropagated "error"

# Backpropagation involves simple computations

5

Initialize  $W_m^0, j \rightarrow V_{k,2}$

Iterate for  $t=0,1,2,\dots$

SGD

Forward Net

Backward  
Updates

choose  $i_t \in \{1, 2, \dots, N\}$   
at random

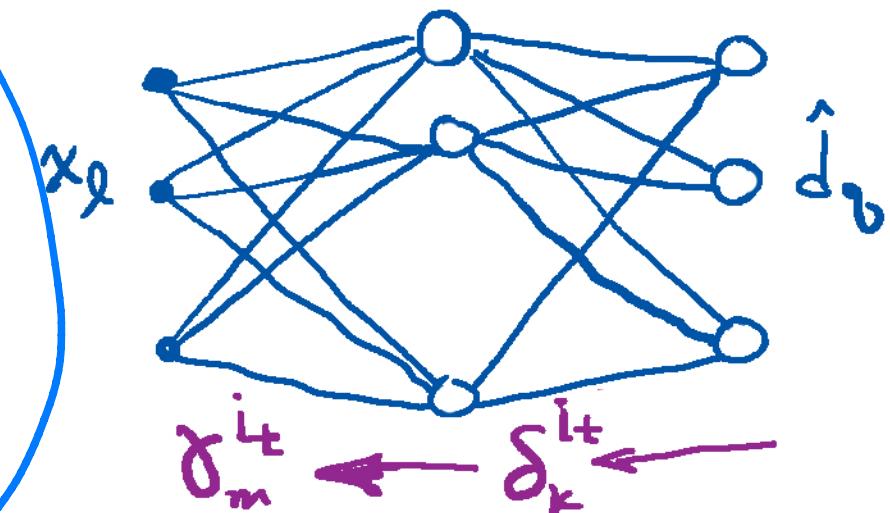
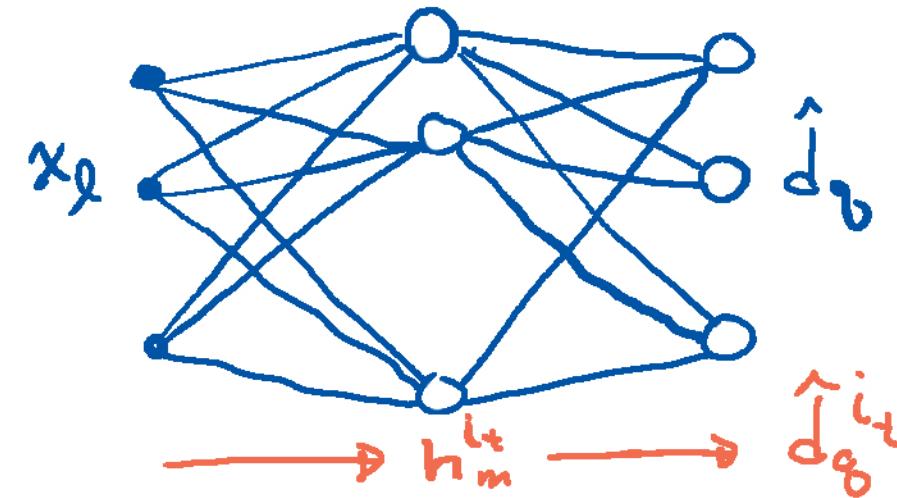
compute  $h_m^{it}$ ,  $\hat{d}_g^{it}$  from  
 $x_k^{it}$ ,  $w_m^t$ ,  $v_{k,2}^t$

$$\delta_k^{it} = (\hat{d}_k^{it} - d_k^{it}) \hat{d}_k^{it} (1 - \hat{d}_k^{it})$$

$$v_{k,2}^{t+1} = v_{k,2}^t - \alpha_t \delta_k^{it} h_g^{it}$$

$$\delta_m^{it} = \sum_{g=1}^G \delta_g^{it} v_{g,m}^{t+1} h_m^{it} (1 - h_m^{it})$$

$$w_m^{t+1} = w_m^t - \alpha_t \delta_m^{it} x_j^{it}$$



Variations follow the same pattern

6

- other activation functions
- deeper networks

Cost is a non convex function of weights

- converge to local minima
- empirical "tricks" e.g. batch SGD, normalization to accelerate convergence

Can add regularization

**Copyright 2019  
Barry Van Veen**