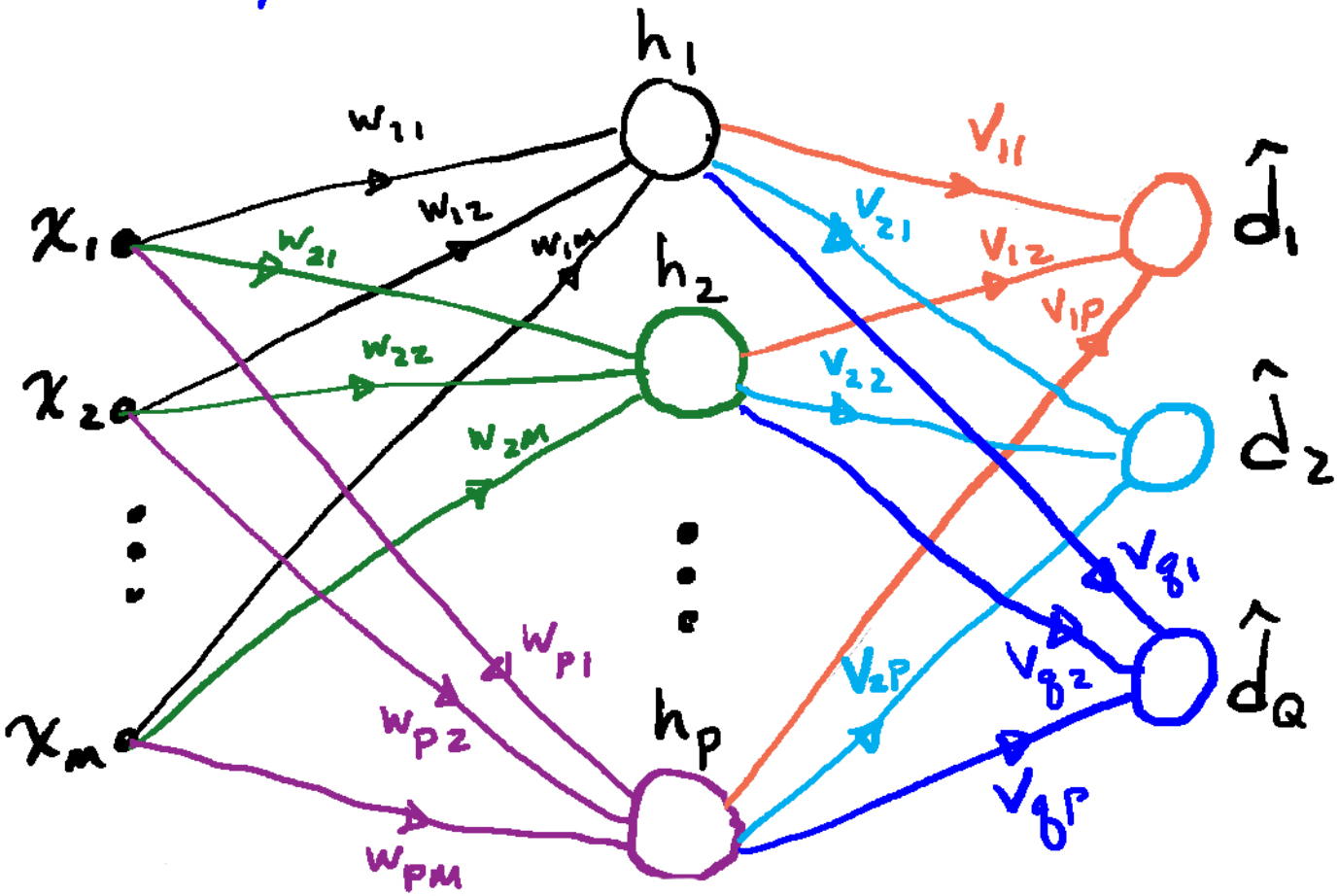
The Backpropagation Algorithm for Training Neural Networks

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 2



N training samples

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

1) initialize $w_{n,l}$, $v_{q,n}$

2) for $t = 1, 2, 3, \cdots$

→ Pick $i_t \in \{1, 2, \cdots, N\}$ randomly

Compute $h_j^{i_t}$, $\hat{d}_k^{i_t}$

$$v_{k,l}^{t+1} = v_{k,l}^t - \alpha_t \frac{d f^{i_t}}{d v_{k,l}}$$

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

— Converged?

$$\hat{d}_q = \sigma\left(\sum_{n=1}^{p} v_{q,n} h_n\right)$$

$\sigma(z)$: activation function

$$h_n = \sigma\left(\sum_{l=1}^{m} w_{n,l} x_l\right) \quad \left(= \frac{1}{1+e^{-z}} \text{ here}\right)$$

Gradients are obtained using the chain rule

Squared error loss: $f(\underline{V},\underline{W}) = \sum_{i=1}^{N} \underbrace{\frac{1}{2}\sum_{q=1}^{Q}(\hat{d}_q^i - d_q^i)^2}_{f^i(\underline{V},\underline{W})}$

$$\frac{\partial f^i}{\partial V_{k,\ell}} = \left(\frac{\partial f^i}{\partial \hat{d}_k^i}\right)\frac{\partial \hat{d}_k^i}{\partial V_{k,\ell}}$$

$\frac{1}{1+e^{-z}} = \sigma(z)$

$\frac{\partial}{\partial z}\sigma(z) = \sigma'(z)$

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

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

$$= \sigma(z)(1-\sigma(z))$$

But $\sigma\left(\sum_{n=1}^{P} h_n^i V_{k,n}\right) = \hat{d}_k^i$

$$\left(1 - \sigma\left(\sum_{n=1}^{P} h_n^i V_{k,n}\right)\right) = \left(1 - \hat{d}_k^i\right)$$

$$\frac{\partial f^i}{\partial V_{k,\ell}} = \underbrace{(\hat{d}_k^i - d_k^i)\,\hat{d}_k^i(1-\hat{d}_k^i)}_{\delta_k^i} h_\ell^i$$

$$= \delta_k^i h_\ell^i$$

$$V_{k,\ell}^{t+1} = V_{k,\ell}^t - \alpha_t \delta_k^i h_\ell^i$$

h_ℓ^i: layer input

δ_k^i: "error" at output

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

$$\frac{\partial f^i}{\partial w_{m,j}} = \sum_{q=1}^{Q} \frac{\partial f^i}{\partial \hat{d}_q^i} \cdot \frac{\partial \hat{d}_q^i}{\partial h_m^i} \frac{\partial h_m^i}{\partial w_{m,j}} \qquad \frac{\partial f^i}{\partial \hat{d}_q^i} = (\hat{d}_q^i - d_q^i)$$

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

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

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

$$\frac{\partial f^i}{\partial w_{m,j}} = \underbrace{\sum_{q=1}^{Q} (\hat{d}_q^i - d_q^i) \hat{d}_q^i (1-\hat{d}_q^i) v_{q,m}^i h_m^i (1-h_m^i)}_{\gamma_m^i} x_j^i = \gamma_m^i x_j^i$$

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

x_j^i: layer input

γ_m^i: backpropagated "error"

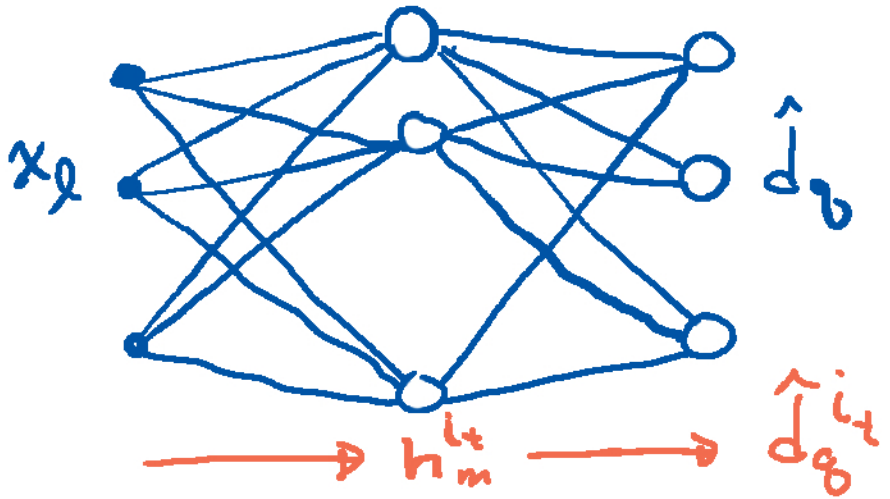
Backpropagation involves simple computations

Initialize $w^0_{m,j}$, $v^0_{k,\ell}$

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

SGD — choose $i_t \in \{1, 2, \ldots, N\}$ at random

Forward Net — compute $h^{i_t}_m$, $\hat{d}^{i_t}_q$ from $x^{i_t}_\ell$, $w^t_{m,j}$, $v^t_{k,\ell}$



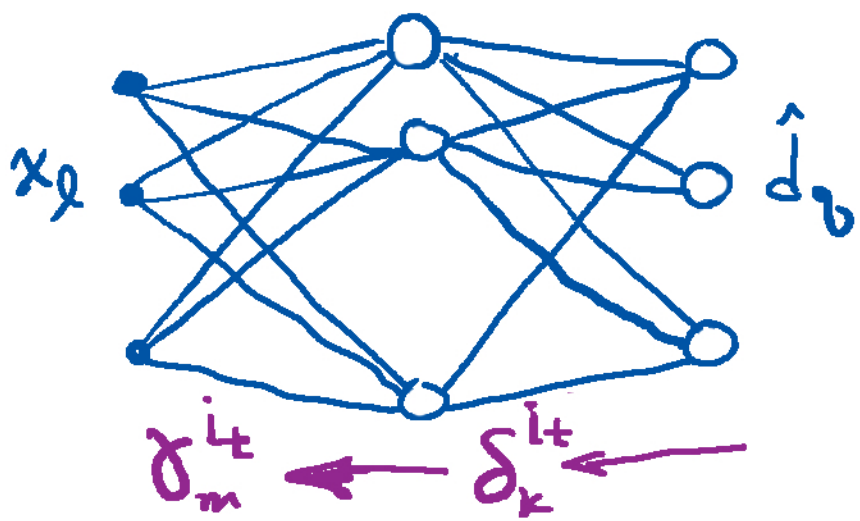
Backward Updates

$$\delta^{i_t}_k = \left(\hat{d}^{i_t}_k - d^{i_t}_k\right) \hat{d}^{i_t}_k \left(1 - \hat{d}^{i_t}_k\right)$$

$$v^{t+1}_{k,\ell} = v^t_{k,\ell} - \alpha_t \delta^{i_t}_k h^{i_t}_\ell$$

$$\gamma^{i_t}_m = \sum_{q=1}^{Q} \delta^{i_t}_q v^t_{q,m} h^{i_t}_m \left(1 - h^{i_t}_m\right)$$

$$w^{t+1}_{m,j} = w^t_{m,j} - \alpha_t \gamma^{i_t}_m x^{i_t}_j$$



Variations follow the same pattern

- 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