Artificial Neural Networks

So far the regression we have done uses fixed basis functions and outputs that are linear in the parameters.

$$\hat{y}(x) = w^T \phi(x)$$

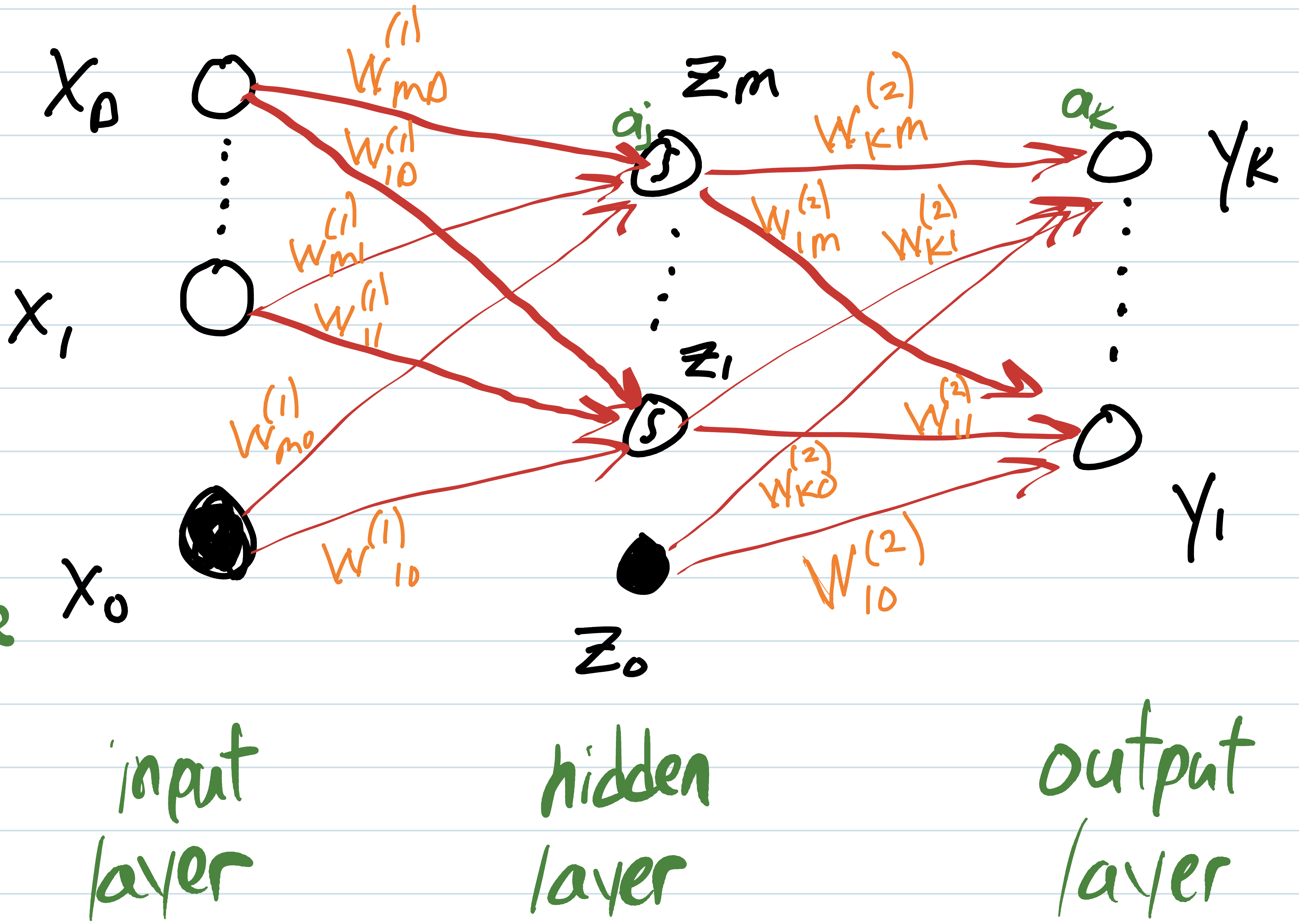
polynomial regression $\phi(x) = [1 \;\; x \;\; x^2 \; \ldots \; x^n]$

locally weighted regression: same as polynomial regression just locally weighted.

Gaussian process regression: basis functions were centered at training data points

Artificial Neural Networks (ANN) use adaptive basis functions instead of lots of fixed basis functions.

ANN basics: ○ node → connection



$D+1$ inputs

$m+1$ hidden nodes

K outputs

2 layer ANN

1 on page 11

The k_{th} output

$$\hat{y}_k(x) = w_{k0}^{(2)} z_0 + \sum_{j=1}^{M} w_{kj}^{(2)} \underbrace{\phi_j\Big(\overbrace{w_{j0}^{(1)} x_0 + \sum_{i=1}^{D} w_{ji}^{(1)} x_i}^{\text{input to hidden layer}}\Big)}_{\text{outputs of hidden layer}}$$

$W_{xy}^{(z)}$

Weight

z = layer number

x = # of node in output layer

y = # of node in the input layer

$z_0 = 1$

$x_0 = 1$

$\vec{y} \in \mathbb{R}^K$ output vector

$\vec{x} \in \mathbb{R}^D$ input vector

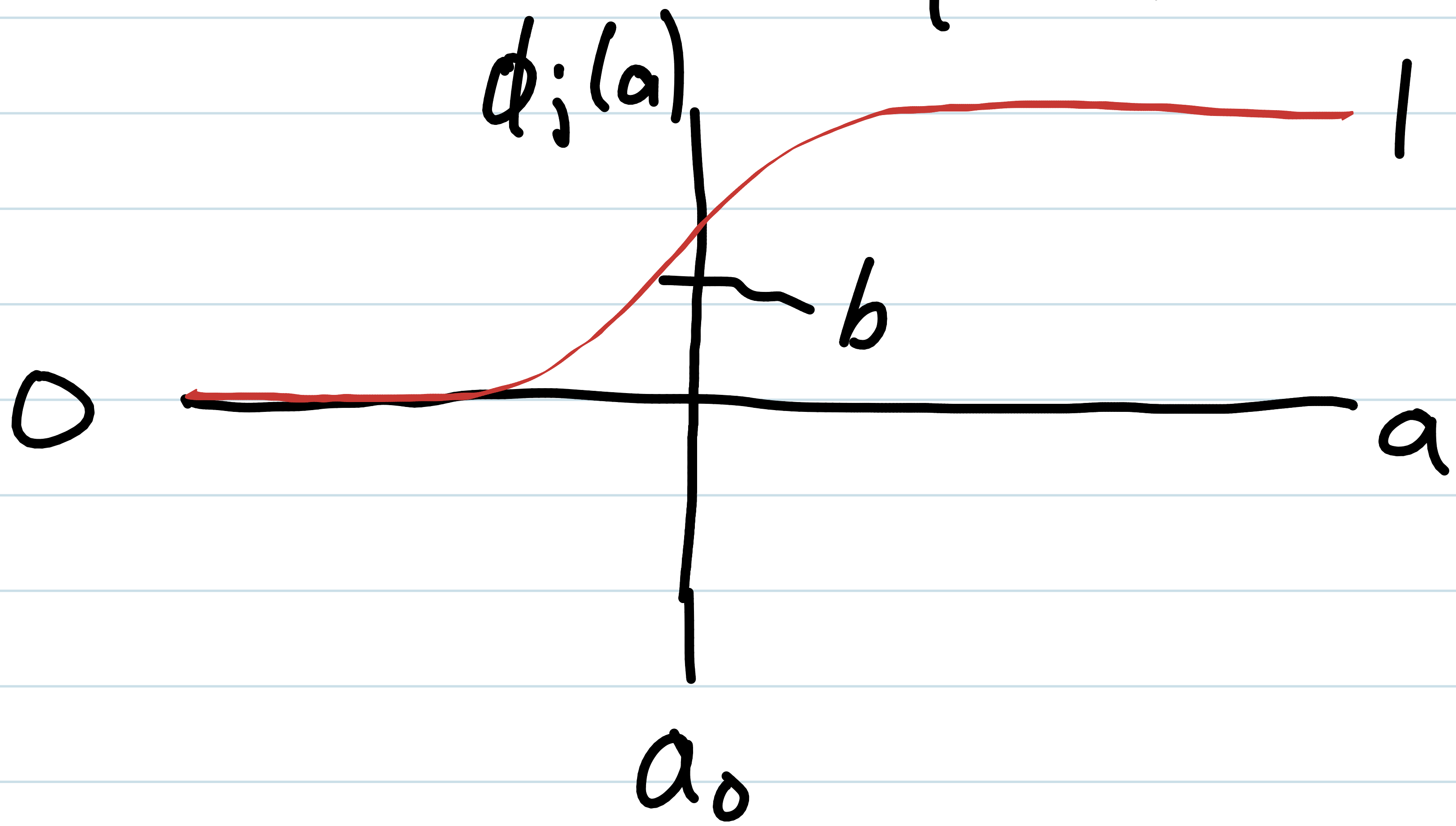
M is the number of nodes in the hidden layer

$\phi_j(\cdot)$ is a nonlinear activation function

Activation function (basis functions for ANN)

$$\phi_j(a) = \frac{1}{1+\exp(-a)}$$

logistic sigmoid function



Could add slope and offset

$$\frac{1}{1+\exp(-b(a-a_0))}$$

Each basis function gets activated depending on the input.

Regression with ANN's using backpropagation

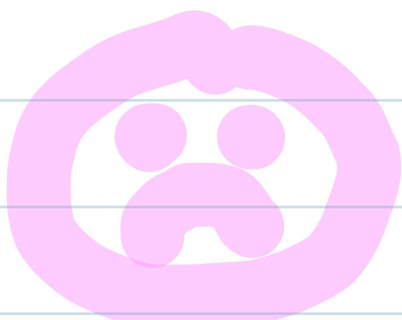
Problem: find $\hat{y}(x_*)$ given x_* (a query) and a training data set $D = \{X\ Y\}$

Solution: find the parameters $\vec{w}$

We want to minimize the sum of squared errors

$$E(\vec{w}) = \frac{1}{2} \sum_{i=1}^{N} \Big[\underbrace{y(x_i, w)}_{\text{actual training output}} - \underset{\substack{\uparrow \\ \text{prediction}}}{\hat{y}_i}\Big]^T \Big[y(x_i, w) - \hat{y}_i \Big]$$

Because of the nonlinear activation functions, there is no analytical solution to $\min_{\vec{w}} E(\vec{w})$ so we have to use an iterative optimization method



Let's use a gradient based optimization method like gradient descent

1) start with an initial guess $\vec{W}_0$

2) take a step in the direction of the negative gradient

$$\vec{W}_{k+1} = \vec{W}_k - \eta \underbrace{\nabla E}_{\text{gradient of error wrt } \vec{w}}$$

learning rate

6 on page 12

$$\nabla E = \left[\frac{\partial E}{\partial w_1} \; \cdots \; \frac{\partial E}{\partial w_Q} \right]^T$$ for Q elements in parameter vector.

We need a way to compute ∇E.

Backpropagation:

The grand error is the sum of individual errors.

$$E(\vec{w}) = \sum_{n=1}^{N} E_n(\vec{w})$$

find $\nabla E_n(\vec{w})$

Consider a linear model with outputs y_k and inputs x_i

$$\hat{y}_k = \sum_i w_{ki} x_i$$

and error function

$$E_n = \frac{1}{2} \sum_k (\underset{\substack{\uparrow \\ \text{estimate}}}{\hat{y}_{nk}} - \underset{\substack{\uparrow \\ \text{measured}}}{y_{nk}})^2 \qquad y_{nk} = y_k(x_n, \vec{w})$$

$$E_n = \frac{1}{2} \sum_k (w_{ki} x_i - y_{nk})^2$$

The gradient w.r.t. the parameter w_{ji}

$$\frac{\partial E_n}{\partial w_{ji}} = (\hat{y}_{nj} - y_{nj}) x_{ni}$$

This is a combination of the error $\hat{y}_{nj} - y_{nj}$ and the input x_{ni}

In a feedforward network the activation a_j of each node is the weighted sum of inputs.

$$a_j = \sum_i w_{ji} z_i$$

activation of jth node

ith weight

ith input to the node

The output of the j^{th} node z_j is transformed by a nonlinear activation function $h(\cdot)$

$$z_j = h(a_j)$$

The derivative of E_n wrt weigh w_{ji}

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}$$

4 on page 11

define $\delta_j \equiv \dfrac{\partial E_n}{\partial a_j}$

$$\frac{\partial a_j}{\partial w_{ji}} = z_i \Rightarrow \frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$$

We need to calculate δ_j for the hidden and output nodes.

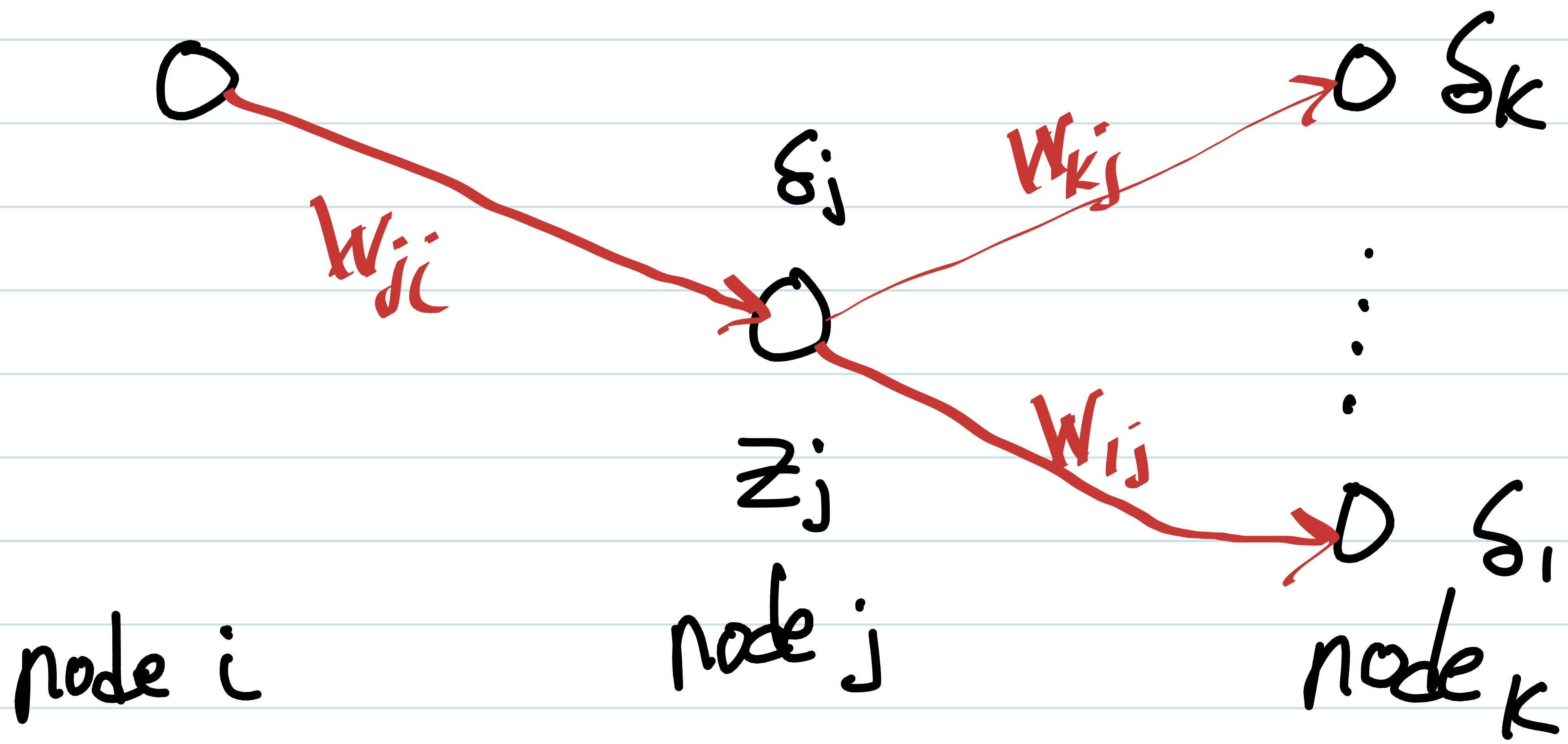
For output nodes

$$\delta_k = \hat{y}_k - y_k$$

2 on page 11

For hidden nodes

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j}$$



this sum runs over all k nodes for which node j sends connections

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k \quad (3)$$

back propagation rule.

previous layer

current layer

Steps for back propagation

1) apply input $\vec{x}_n$ to the network to find activations of all hidden and output nodes.

2) evaluate δ_k for output nodes

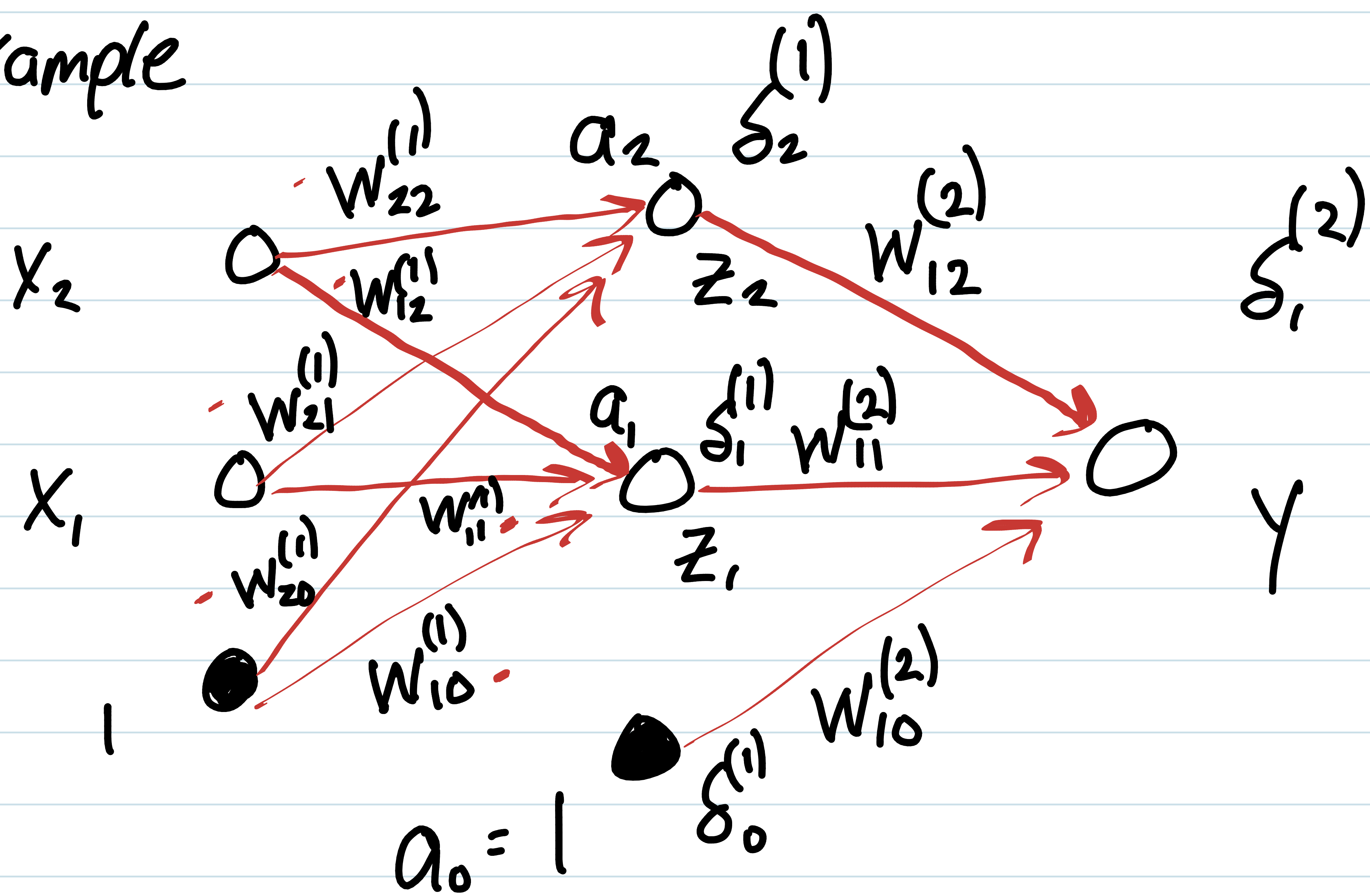
3) backpropage the δ's to obtain δ_j for each hidden node.

4) evaluate the derivatives

5) add derivatives together $\quad \dfrac{\partial E}{\partial w_{ji}} = \sum_n \dfrac{\partial E_n}{\partial w_{ji}}$

b) update estimates of weights

Example



hidden nodes have activation functions $h(a) = \dfrac{1}{1+\exp(-a)}$

$$\frac{dh}{da} = h'(a) = \frac{\exp(-a)}{(1+\exp(-a))^2}$$

Step 1: Do forward forward propagation for each member of the training set.

$$a_0 = 1$$

$$a_1 = w_{10}^1 + w_{11}^1 x_1 + w_{12}^1 x_2$$

$$a_2 = w_{20}^1 + w_{21}^1 x_1 + w_{22}^1 x_2$$

$$z_0 = \frac{1}{1+\exp(-a_0)} \qquad z_1 = \frac{1}{1+\exp(-a_1)} \qquad z_2 = \frac{1}{1+\exp(-a_2)}$$

$$\hat{y} = \underline{w_{10}^2 z_0} + w_{11}^2 z_1 + w_{12}^2 z_2$$

Step 2: Compute δ for output unit for each training data point.

$$\delta^2 = \hat{y} - y$$

estimate actual training output

Step 3: Back propagate to get δ for hidden units

$$\delta_0^1 = \underbrace{\frac{\exp(-a_0)}{(1+\exp(-a_0))^2}}_{\frac{\partial a_k}{\partial a_j}} \underbrace{w_{10}^2 \delta^2}_{\frac{\partial \delta^2}{\partial z_0}}$$

$$\delta_1^1 = \frac{\exp(-a_1)}{(1+\exp(-a_1)^2} \quad w_{11}^2 \delta^2$$

$$\delta_2^1 = \frac{\exp(-a_2)}{(1+\exp(-a_2))^2} \quad w_{12}^2 \delta^2$$

4) evaluate derivatives

$$\frac{\partial E_n}{\partial w^2_{10}} = \delta^2 z_0$$

$$E_n = \frac{1}{2}(\hat{y}-y)^2 \qquad \frac{\partial E_n}{\partial w} = (\hat{y}-y)\frac{\partial(\hat{y}-y)}{\partial w}$$

$$\frac{\partial E_n}{\partial w^2_{11}} = \delta^2 z_1 \qquad \frac{\partial E_n}{\partial w^2_{12}} = \delta^2 z_2$$

$$\frac{\partial E_n}{\partial w^1_{10}} = \delta^1_1 x_0 \qquad \frac{\partial E_n}{\partial w^1_{20}} = \delta^1_2 x_0 \qquad \frac{\partial E_n}{\partial w^1_{12}} = \delta^1_1 x_2$$

$$\frac{\partial E_n}{\partial w^1_{11}} = \delta^1_1 x_1 \qquad \frac{\partial E_n}{\partial w^1_{21}} = \delta^1_2 x_1 \qquad \frac{\partial E_n}{\partial w^1_{22}} = \delta^1_2 x_2$$

Step 5: add derivatives for all the training data

e.g. $$\frac{\partial E}{\partial W} = \sum_{n=1}^{N} \frac{\partial E_n}{\partial W}$$

Step 6: update parameters

$$\vec{W}_{k+1} = \vec{W}_k - \eta \nabla E$$

repeat until $\|W_{k-1} - W_k\| < \varepsilon$

Convergence Criteria

$$\nabla E = \left[\frac{\partial E}{\partial w_{11}^2} \quad \frac{\partial E}{\partial w_{12}^2} \quad \frac{\partial E}{\partial w_{10}^1} \quad \frac{\partial E}{\partial w_{20}^1} \quad \frac{\partial E}{\partial w_{11}^1} \quad \frac{\partial E}{\partial w_{21}^1} \quad \frac{\partial E}{\partial w_{12}^1} \quad \frac{\partial E}{\partial w_{22}^1} \right]^T$$

How to initiate the weights?

1) pick them to be zero

2) pick them randomly

3) pick them based on your weights for a similar problem