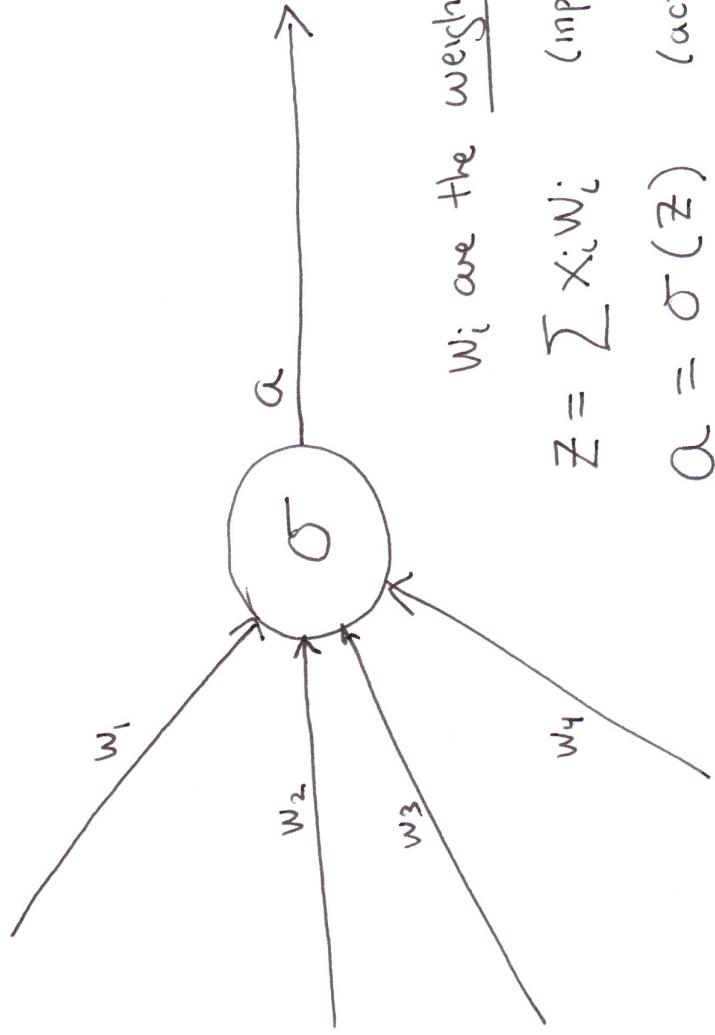


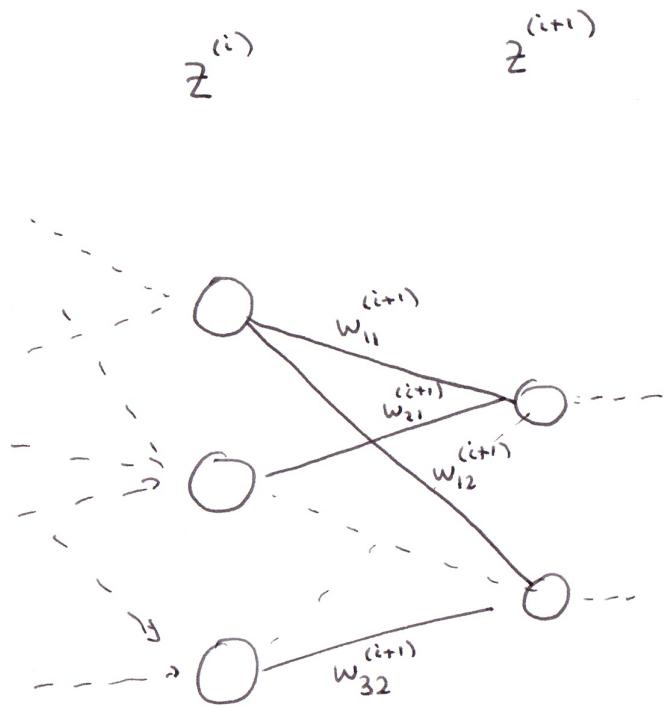
## A Neuron



$$\sigma(t) = \frac{1}{1+e^{-t}} \quad \text{or} \quad \sigma(t) = \begin{cases} t & \text{if } t \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\text{or} \quad \sigma(t) = t$$

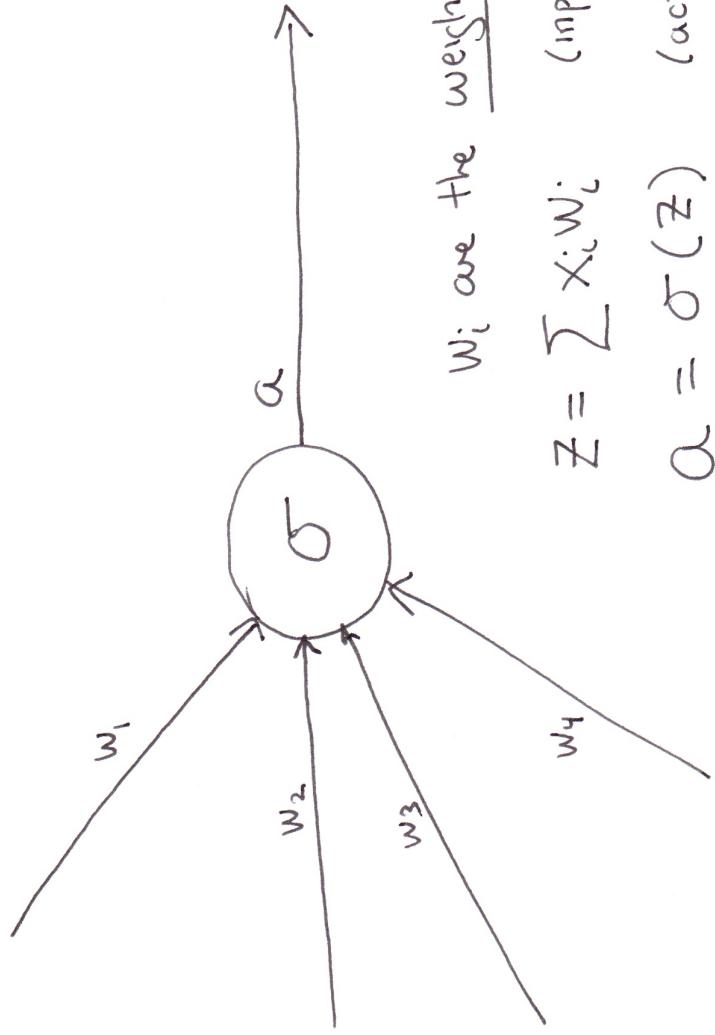
Layers



$$z_l^{(i+1)} = \sum_{j=1}^{n^{(i)}} a_j^{(i)} w_{jl}^{(i+1)}$$

$$a_l^{(i+1)} = \sigma(z_l^{(i+1)})$$

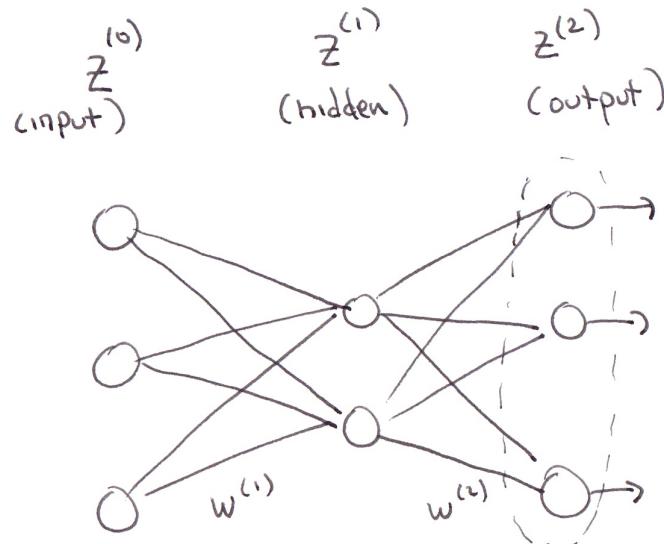
## A Neuron



$$\sigma(t) = \frac{1}{1+e^{-t}} \quad \text{or} \quad \sigma(t) = \begin{cases} t & \text{if } t \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\text{or} \quad \sigma(t) = t$$

with all weights specified, a network defines a function  
 called the "feed-forward" function



$z^{(0)}$        $1 \times 3$  matrix

$w^{(1)}$        $3 \times 2$  matrix

$w^{(2)}$        $2 \times 1$  matrix

$$z^{(0)} \rightarrow z^{(0)}w^{(1)} \rightarrow \sigma(z^{(0)}w^{(1)}) \rightarrow \sigma(z^{(0)}w^{(1)})w^{(2)}$$

(elementwise)

$$\boxed{F(\sigma(z^{(0)}w^{(1)})w^{(2)})}$$

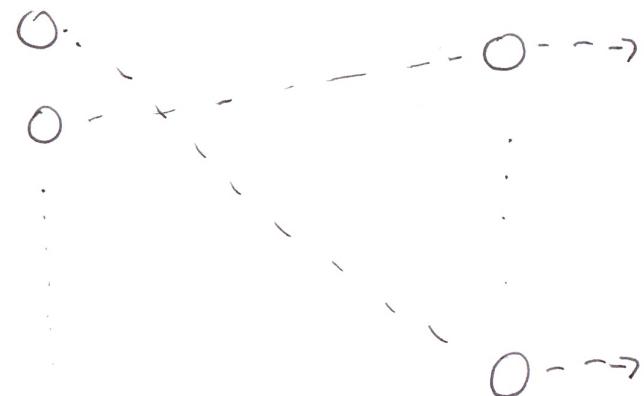
might not be elementwise

Linear Regression

Input vectors  $X$  in  $N$  dimensions

Response vectors  $Y$  in  $M$  dimensions

$N$  inputs  
 $M$  outputs  
 $\sigma(x) = X$



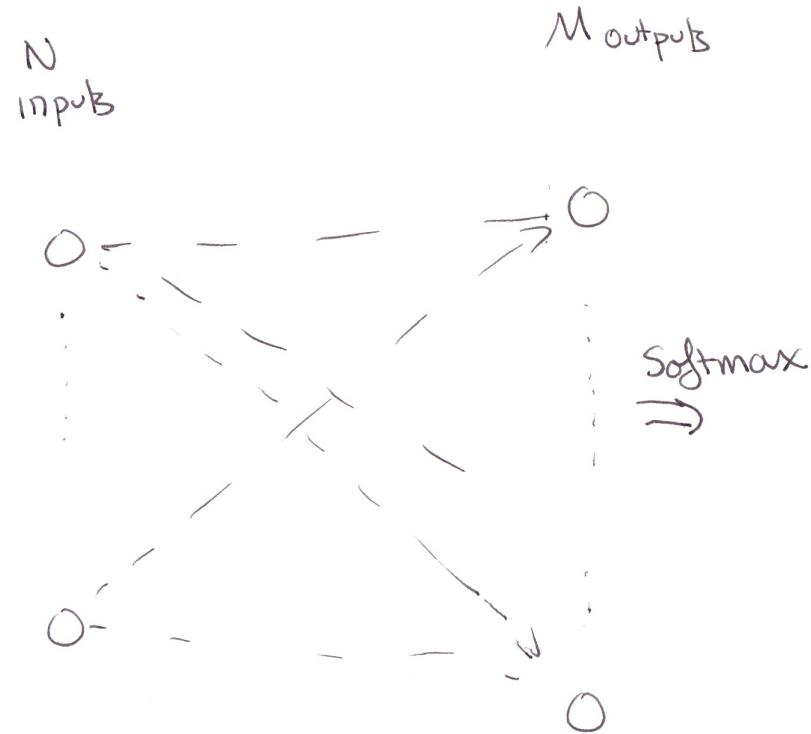
Output

Weights

$$w_{ij}^{(l)} \quad \begin{matrix} 1 \leq i \leq N \\ 1 \leq j \leq M \end{matrix}$$

$$\text{output} = X w^{(l)}$$

# Logistic Regression



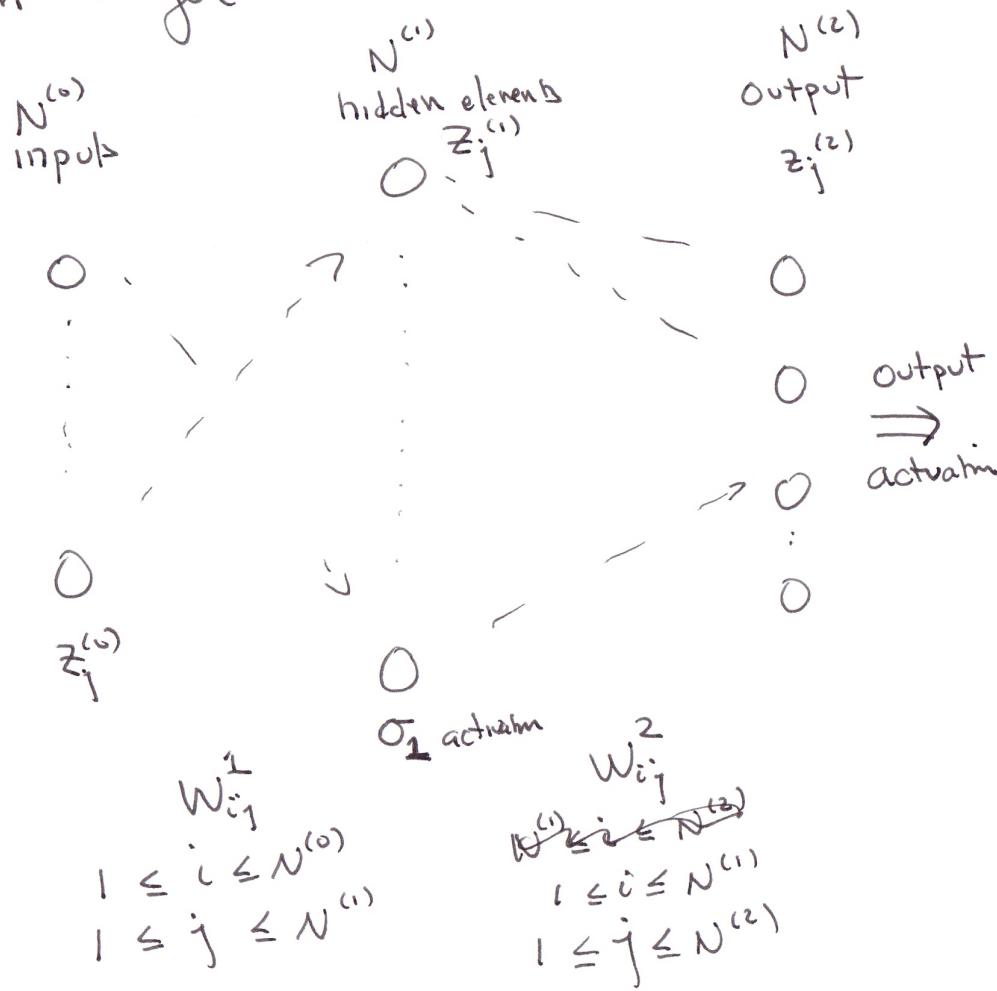
Weights  $W_{ij}^{(l)}$

$$1 \leq i \leq N$$

$$1 \leq j \leq M$$

$$\text{output} = \text{Softmax}(xW^{(l)})$$

# A "hidden" Layer



Training: Given a network  $F_w$  depending on the weights

Ingredients: data  $(x^{[i]}, y^{[i]}) \quad i=1, \dots, M$

A loss function

$$L_w = \frac{1}{M} \sum_{i=1}^M L(y^{[i]}, F_w(x^{[i]}))$$

that compares the output  
 $F_w(x^{[i]})$  to  $y^{[i]}$

NOTE:  $L_w$  is a function of the WEIGHTS

The "data"  $(x^{[i]}, y^{[i]})$  is fixed.

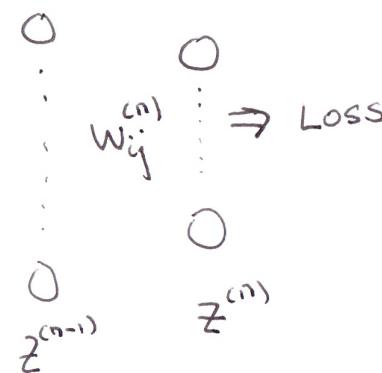
Goal is to minimize  $L$  by varying  $w$ .

Strategy: We have  $w_{ij}^{(l)}$   $l=1, \dots, L$  for  $L$  layers

Compute  $\frac{\partial L}{\partial w_{ij}^{(l)}}$  and use gradient descent

$$w_{ij}^{(l)} = w_{ij}^{(l)} - \lambda \frac{\partial L}{\partial w_{ij}^{(l)}}$$

# Backpropagation



Step 1:

$$\text{Compute } \frac{\partial L}{\partial z_j^{(n)}}$$

Examples:  $L = \frac{1}{2} \|y - z\|^2$

$$\text{MSE } L(y^{(i)}, z_j^{(n)}(x^{(i)})) = \frac{1}{2} \|y^{(i)} - z_j^{(n)}\|^2 = \frac{1}{2} \sum_{j=1}^{N_n} (y_j^{(i)} - z_j^{(n)})^2$$

$$\frac{\partial L}{\partial z_j^{(n)}} = y_j^{(i)} - z_j^{(n)}$$

$$\text{"cross entropy"} \quad L(y^{(i)}, z_j^{(n)}(x^{(i)})) = \sum_{j=1}^{N_n} y_j \log \left( \frac{e^{z_j}}{H} \right) = \sum_{j=1}^{N_n} y_j z_j - \log H$$

$$\frac{\partial L}{\partial z_j} = y_j - p_j$$

$$\text{where } p_j = e^{z_j}/H$$

## Backpropagation

An algorithm to efficiently compute  $\frac{\partial L}{\partial w_{ij}^{(e)}}$  using the graph structure.

Step 1: The output layer

$$\begin{array}{c} \circ \\ \circ \xrightarrow{\text{output}} \xrightarrow{\text{Loss}} L \\ \vdots \end{array}$$

○

$z_j^{(n)}$

$$\delta_j^{(n)} = \frac{\partial L}{\partial z_j^{(n)}}$$

MSE loss

$$\begin{aligned} L(y_j^{(n)}, z^{(n)}(x^{(n)})) \\ = \frac{1}{2} \|y_j^{(n)} - z^{(n)}\|^2 \end{aligned}$$

$$\frac{\partial L}{\partial z_j^{(n)}} = y_j - z_j^{(n)}$$

Crossentropy

$$L(y_j^{(n)}, z^{(n)}) = - \sum_{j=1}^n y_j \log \frac{e^{z_j}}{H}$$

$$\text{where } H = \sum_{j=1}^n e^{z_j} = - \sum y_j z_j + \log H$$

$$\begin{aligned} \frac{\partial L}{\partial z_j^{(n)}} &= + \sum_{j=1}^n y_j z_j \\ &- y_j + \frac{e^{z_j}}{H} = -y_j + p_j \end{aligned}$$

# Earlier Layers

$$\delta_j^{(i-1)} = \sum_l \frac{\partial L}{\partial z_l^{(i)}} \frac{\partial z_l^{(i)}}{\partial z_j^{(i-1)}}$$

$$= \sum_l \delta_l^{(i)} \sigma'(z_j^{(i-1)}) w_{jl}^{(i)}$$

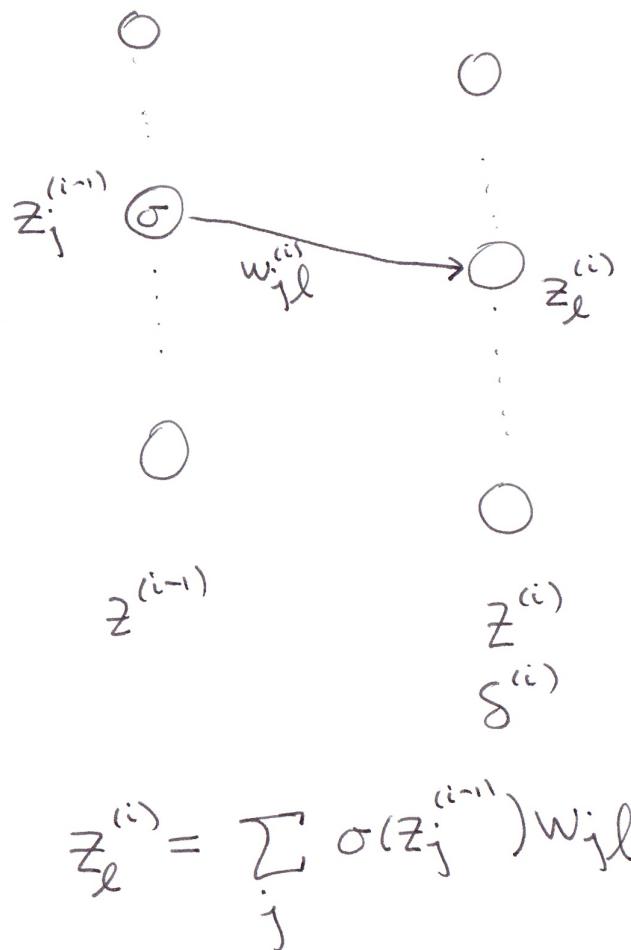
$$= \sigma'(z_j^{(i-1)}) \underbrace{\sum_l w_{jl} \delta_l^{(i)}}_{l^{\text{th}} \text{ entry of } {}^T W^{(i)} \delta^{(i)}}$$

so

~~$\delta^{(i-1)} = \sigma'(z^{(i-1)}) {}^T W \delta^{(i)}$~~ 

$$= \sigma'(z^{(i-1)}) {}^T W \delta^{(i)}$$

where the multiplication by  $\sigma'(z^{(i-1)})$  means entry by entry.



## Backpropagation cont'd

$$\frac{\partial L}{\partial w_{ij}^{(e)}} = \sum \frac{\partial L}{\partial z_j^{(e)}} \frac{\partial z_j^{(e)}}{\partial w_{ij}^{(e)}} = \sum s_j^{(e)} \sigma(z_i^{(e-1)})$$

since  $z_j^{(e)} = \sum_i \sigma(z_i^{(e-1)}) w_{ij}^{(e)}$

In matrix terms

$$\frac{\partial L}{\partial w^{(e)}}$$
 is the matrix with entries  $(s_j^{(e)} \sigma(z_i^{(e-1)}))$

This is the "outer product" of the vectors  $s^{(e)}$  and  $\sigma(z^{(e-1)})$

where  $\sigma(z^{(e-1)})$  means apply  $\sigma$  element wise.

## Backpropagation cont'd

To exploit this, during the forward pass,  
save the  $z^{(i)}$  and also compute  $\sigma'(z^{(i)})$   
Then make a backward pass to compute  
the  $S^{(i)}$  using the weights from the forward  
pass.

Since the total  $L$  is the sum of  $L(y^{[i]}, F_w(x^{[i]}))$   
you can accumulate the  $S^{(i)}$  on each pass,  
the  $\frac{\partial L}{\partial w^{(i)}}$  on each pass

## Training Algorithm

Initialize network with random weights. Set all the  $S_{\star}^{(i)} \nabla w_{\star}$  to zero

For  $x, y$  in the data:

- make a forward pass through the network, computing and saving the inputs  $z^i$  at each stage

- make a backwards pass through the network

Compute  $\delta^{(i)}$  and  $\nabla w^{(i)}$  at each

stage using the back propagation equations.

$$\text{Accumulate } S_{\star}^{(i)} = S_{\star}^{(i)} + \delta^{(i)} \quad \text{and} \quad \nabla w_{\star}^{(i)} = \nabla w_{\star}^{(i)} + \nabla w^{(i)}$$

- periodically (maybe every time, maybe after  $B$  data points, maybe only at the end of the data)

$$\text{Update the weights } W^{(i)} = W^{(i)} - \lambda \nabla w^{(i)}$$

Reset the accumulators to zero

This is ONE TRAINING EPOCH

Repeat until the loss stops decreasing...