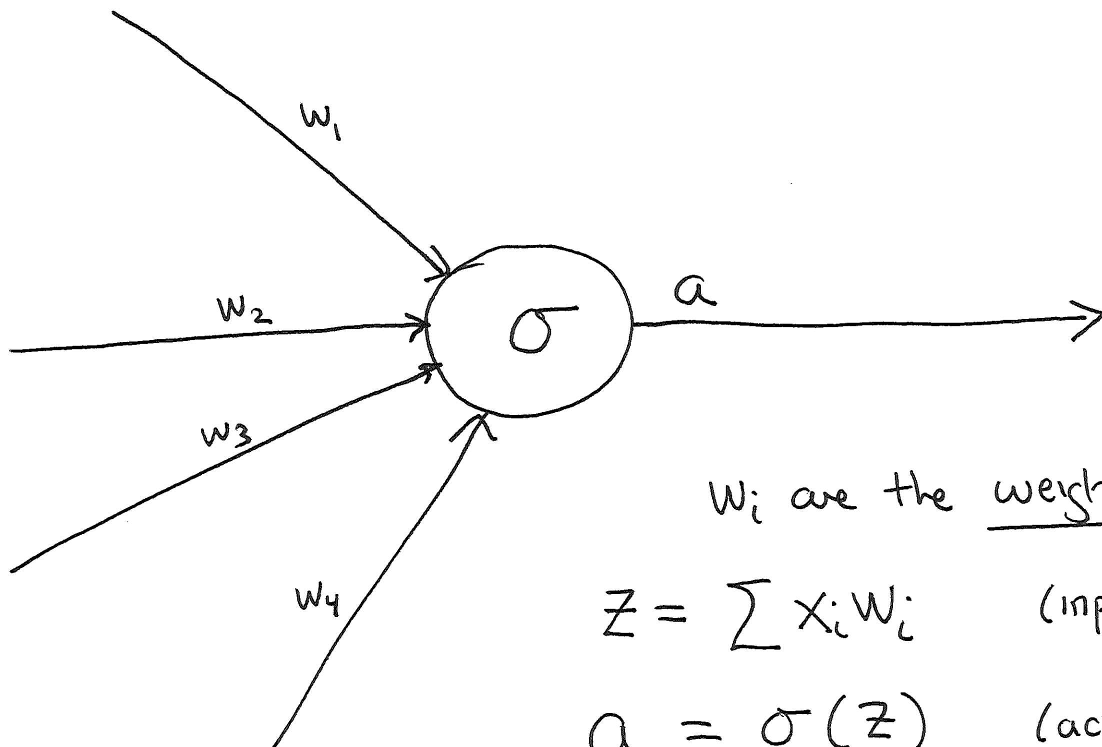
A Neuron



W_i are the <u>weights</u>

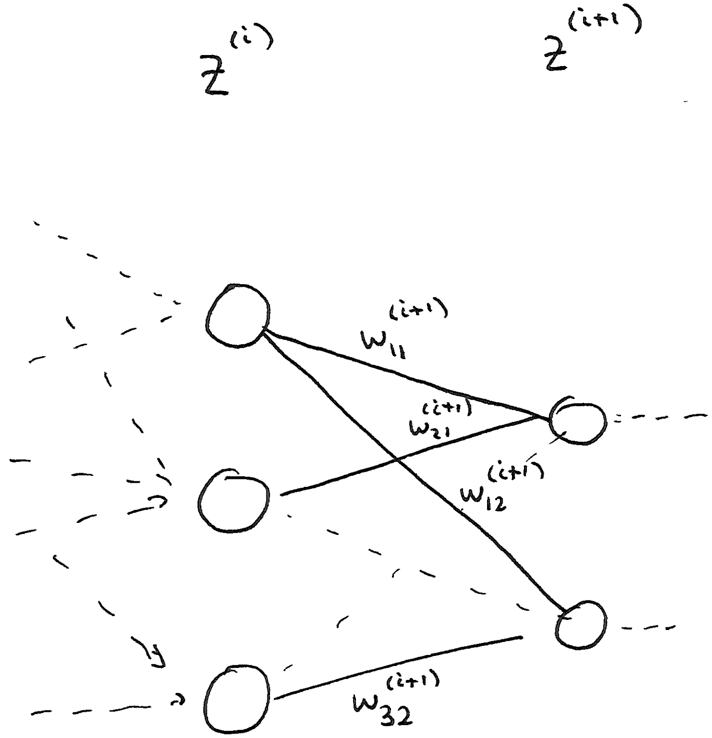
$$z = \sum x_i w_i \quad \text{(input)}$$

$$a = \sigma(z) \quad \text{(activation)}$$

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

$$\underline{\text{OR}} \quad \sigma(t) = t$$

Layers

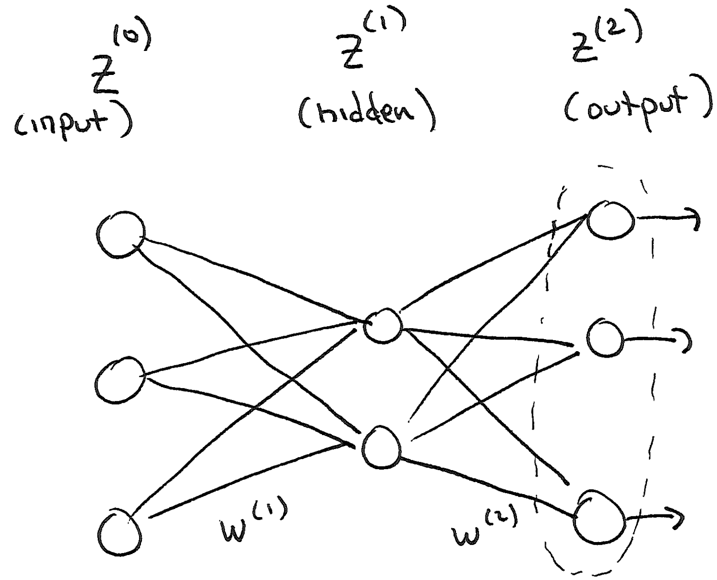


$$z_l^{(i+1)} = \sum_{j=1}^{n^{(i)}} a_j^{(i)} w_{jl}$$

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

with all weights specified, a network defines a <u>function</u>

Called the "feed-forward" function



$z^{(0)}$	1×3	matrix
$w^{(1)}$	3×2	matrix
$w^{(2)}$	2×3	matrix

$$z^{(0)} \longrightarrow z^{(0)}w^{(1)} \longrightarrow \underset{\text{(elementwise)}}{\sigma(z^{(0)}w^{(1)})} \longrightarrow \sigma(z^{(0)}w^{(1)})w^{(2)}$$

$$\downarrow$$

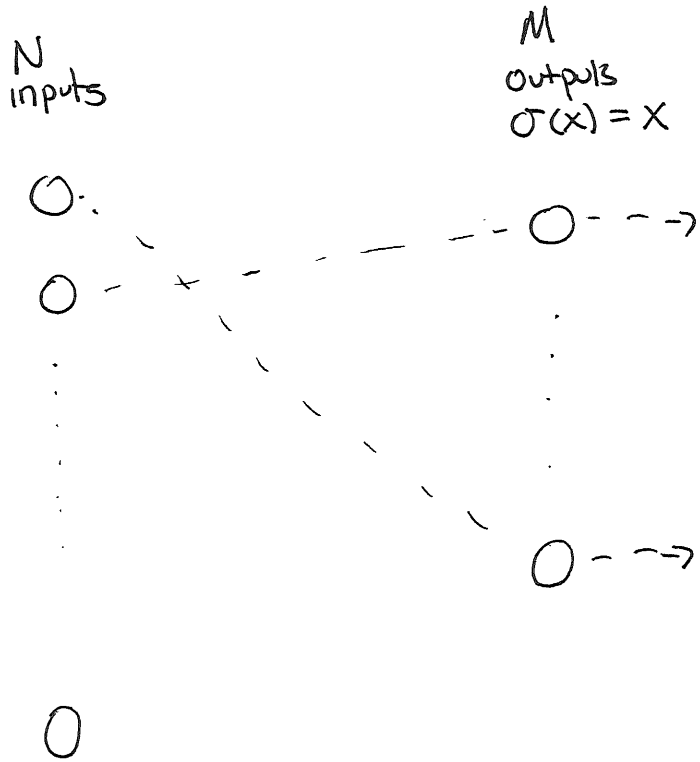
$$F(\sigma(z^{(0)}w^{(1)})w^{(2)})$$

might not be elementwise

Linear Regression

Input vectors X in N dimensions

Response vectors Y in M dimensions



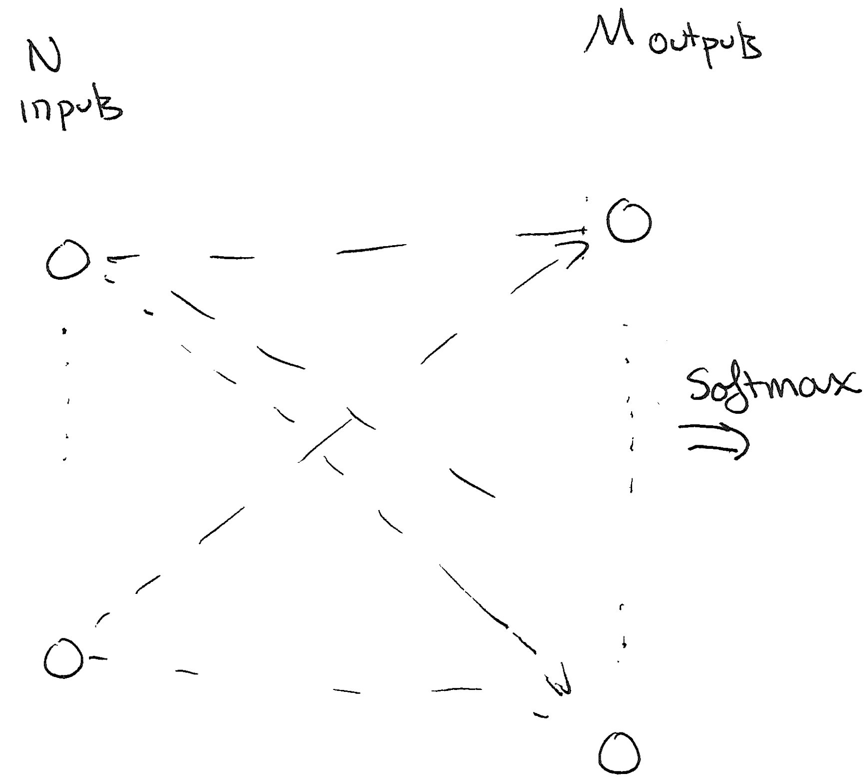
Weights

$W^{(1)}_{ij}$ $1 \leq i \leq N$

$1 \leq j \leq M$

output $= xW^{(1)}$

Logistic Regression

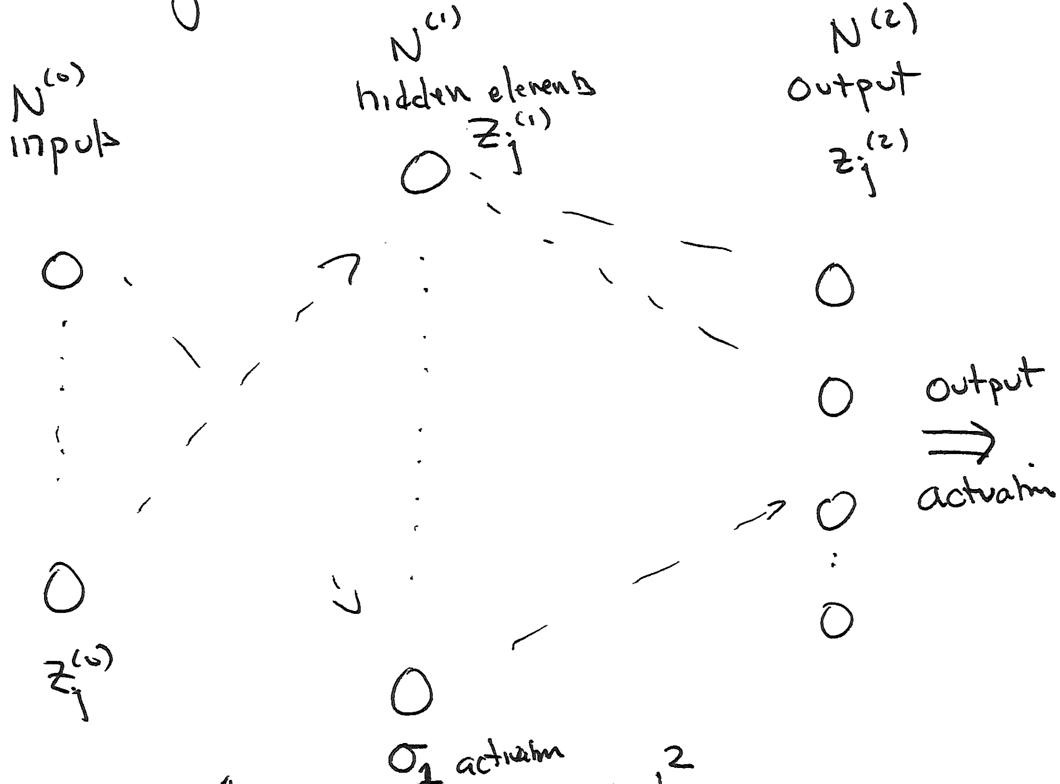


Weights $W_{ij}^{(1)}$

$1 \le i \le N$

$1 \le j \le M$ $\quad$ output $= \text{Softmax}(XW^{(1)})$

A "hidden" Layer



W^1_{ij}

$1 \le i \le N^{(0)}$

$1 \le j \le N^{(1)}$

W^2_{ij}

$\cancel{N^{(1)} \le i \le N^{(2)}}$

$1 \le i \le N^{(1)}$

$1 \le j \le N^{(2)}$

__Training__ : Given a network ~~with~~ F_W depending on the weights

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

A loss function

$$\mathcal{L}_W = \frac{1}{M} \sum_{i=1}^{M} \mathcal{L}(y^{[i]}, F_W(x^{[i]}))$$

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

__NOTE__: $\mathcal{L}_W$ is a function of the __WEIGHTS__

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

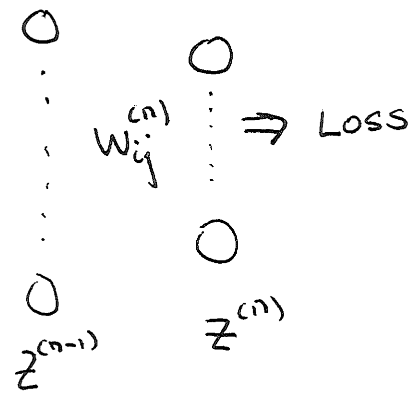
Goal is to Minimize $\mathcal{L}$ by varying W.

Strategy: We have $W_{ij}^{(\ell)}$ $\quad \ell = 1, \dots, L$ for L layers

Compute $\dfrac{\partial \mathcal{L}}{\partial W_{ij}^{(\ell)}}$ and use gradient descent

$$W_{ij}^{(\ell)} = W_{ij}^{(\ell)} - \lambda \frac{\partial \mathcal{L}}{\partial W_{ij}^{(\ell)}}$$

Backpropagation



Step 1:

Compute $\dfrac{\partial \mathcal{L}}{\partial z_j^{(n)}}$

Examples:

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

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

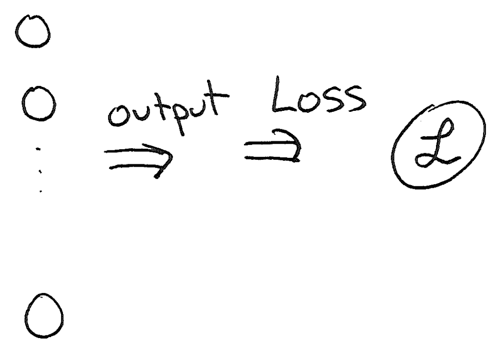
"cross entropy" $\quad \mathcal{L}(y^{[i]}, z_j^{(n)}(x^{[i]})) = \sum_{j=1}^{N_n} y_j \log\left(\frac{e^{z_j}}{H}\right) = \sum_{1}^{N_n} y_j z_j - \log H$

$$\frac{\partial \mathcal{L}}{\partial z_j} = y_j - p_j \qquad \text{where} \quad p_j = e^{z_j}/H.$$

Backpropagation

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

Step 1: The output layer



$z_j^{(n)}$

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

MSE loss

$$\mathcal{L}(y^{[i]}, z^{(n)}(x^{[i]})) = \frac{1}{2} \| y^{[i]} - z^{(n)} \|^2$$

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

Crossentropy

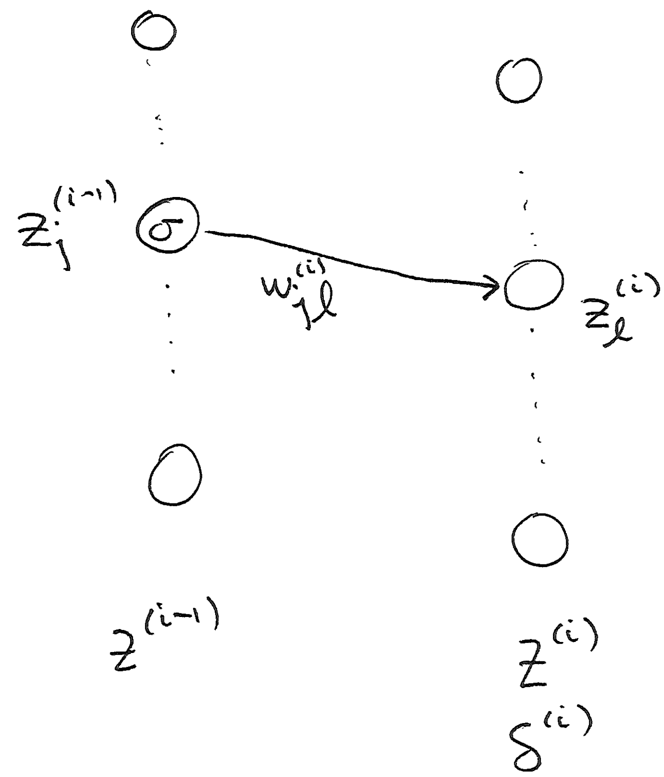
$$\mathcal{L}(y^{[i]}, z^{(n)}) = -\sum_{j=1}^{L_n} y_j \log \frac{e^{z_j}}{H}$$

where $H = \sum_{j=1}^{L_n} e^{z_j}$

$$= -\sum y_j z_j + \log H$$

$$\frac{\partial \mathcal{L}}{\partial z_j^{(n)}} = -y_j + \frac{e^{z_j}}{H} = -y_j + P_j$$

Earlier Layers



$$z_l^{(i)} = \sum_j \sigma(z_j^{(i-1)}) w_{jl}$$

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

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

$$= \sigma'(z_j^{(i-1)}) \underbrace{\sum_l w_{jl} \delta_l^{(i)}}_{l^{th} \text{ entry of } {}^tW^{(i)} \delta^{(i)}}$$

so $\delta^{(i-1)}$ $\cancel{{}^tW^{(i)} \delta^{(i)}}$

$$= \sigma'(z^{(i-1)})\, {}^tW \delta^{(i)}$$

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

Backpropagation cont'd

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^{(\ell)}} = \sum \frac{\partial \mathcal{L}}{\partial z_j^{(\ell)}} \frac{\partial z_j^{(\ell)}}{\partial w_{ij}^{(\ell)}} = \sum \delta_j^{(\ell)} \sigma(z_i^{(\ell-1)})$$

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

In matrix terms

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

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

where $\sigma(z^{(\ell-1)})$ means apply σ 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 $\delta^{(i)}$ using the weights from the forward pass.

Since the total $\mathcal{L}$ is the sum of $\mathcal{L}(y^{[i]}, F_w(X^{[i]}))$ you can accumulate the $\delta^{(i)}$ on each pass, the $\frac{\partial \mathcal{L}}{\partial W^{(\ell)}}$ on each pass

Training Algorithm

Initialize network with random weights. Set all the $\delta_*^{(i)}$ ∇W_* to zero

For x,y in the data:

- make a forward pass through the network, computing and saving the inputs z^i ~~and~~ 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.
  Accumulate $\delta_*^{(i)} = \delta_*^{(i)} + \delta^{(i)}$ and $\nabla W_*^{(i)} = \nabla W_*^{(i)} + \nabla W^{(i)}$
- periodically (maybe every time, maybe after B data points, maybe only at the end of the data)
  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...