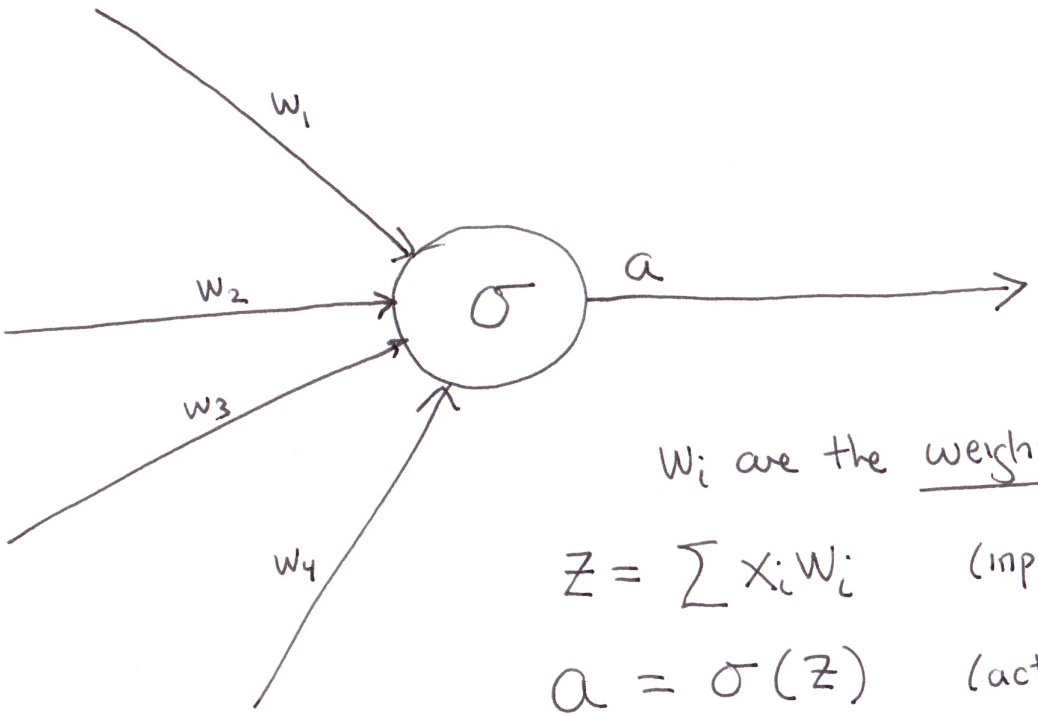
A Neuron



w_i are the weights

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

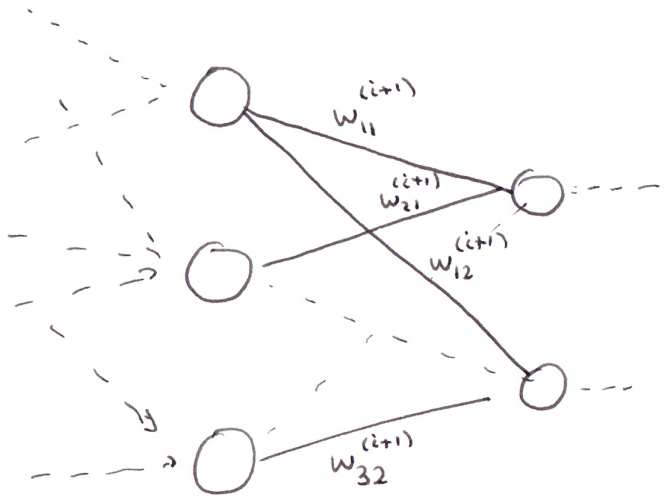
$$a = \sigma(z) \quad \text{(activation)}$$

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

OR $\sigma(t) = t$

Layers

$z^{(i)}$ $z^{(i+1)}$

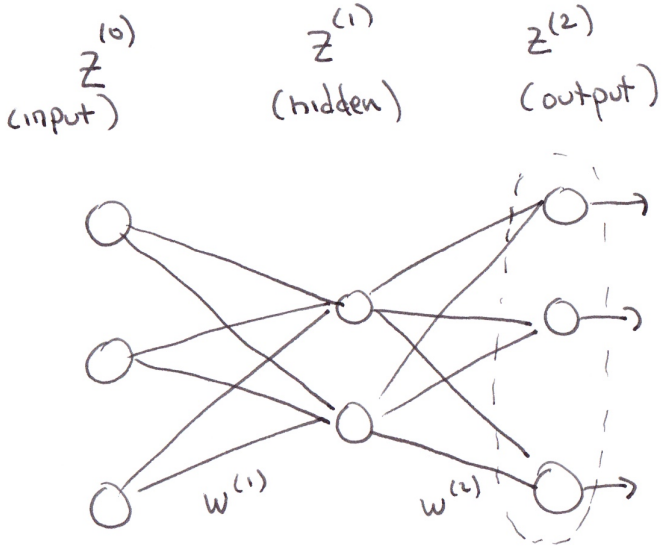


$$z_l^{(i+1)} = \sum_{j=1}^{n^{(i)}} a_j^{(i)} w_{jl}$$

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

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

Called the "feed-forward" function



$z^{(0)}$ 1x3 matrix

$w^{(1)}$ 3x2 matrix

$w^{(2)}$ 2x3 matrix

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

$$\downarrow$$

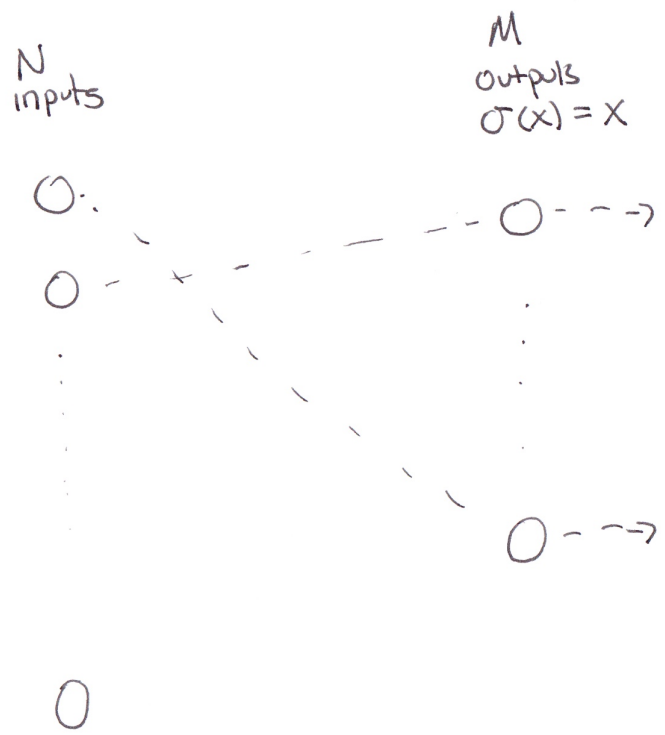
$$F(\sigma(z^{(0)}w^{(1)})w^{(2)})$$

might not be elementwise

Linear Regression

Input vectors X in N dimensions

Response vectors Y in M dimensions



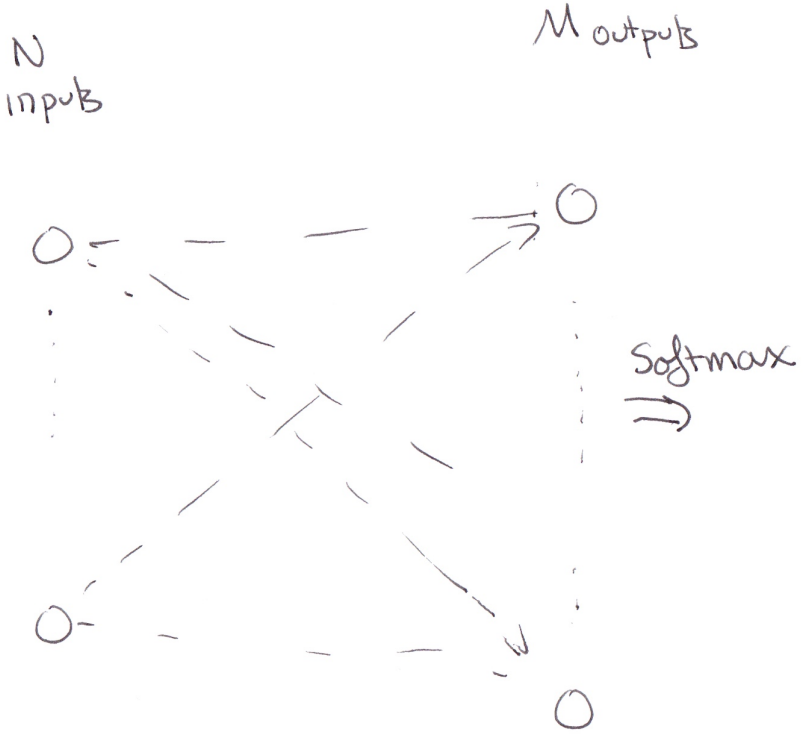
Weights

$W_{ij}^{(1)}$ $\quad 1 \le i \le N$

$\quad\quad\quad\quad\quad\quad 1 \le j \le M$

$$\text{output} = xW^{(1)}$$

Logistic Regression

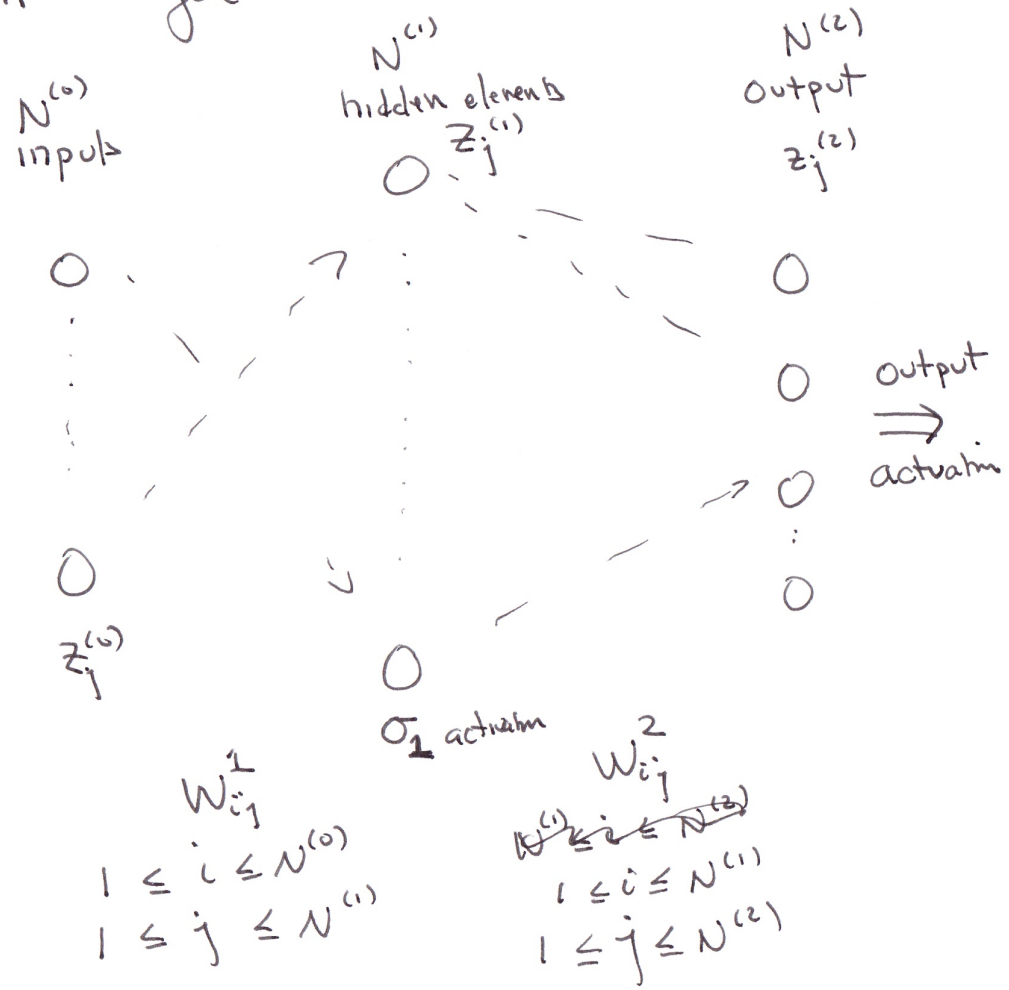


Weights $W_{ij}^{(1)}$

$1 \le i \le N$

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

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

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

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.

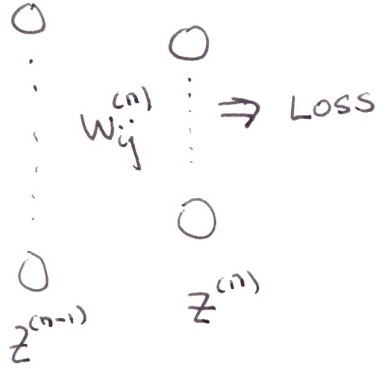
Goal is to Minimize $\mathcal{L}$ by varying W.

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

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

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

Backpropagation



Step 1:

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

Examples: ~~$\mathcal{L} = \|(y^{[i]} - z^{(n)})\|^2$~~

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

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

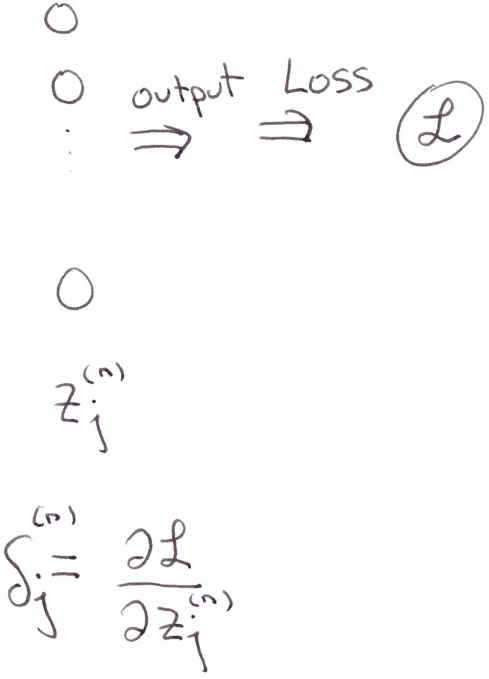
"cross entropy" $\mathcal{L}(y^{[i]}, z_i^{(n)}(x^{[i]})) = \sum_{j=1}^{N_n} y_j \log\left(\frac{e^{z_j}}{H}\right) = \sum_{1}^{N_n} y_j z_j - \log H$

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

Backpropagation

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

Step 1: The output layer



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

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

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

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

$$= -\sum y_i z_i + \log H$$

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

Earlier Layers

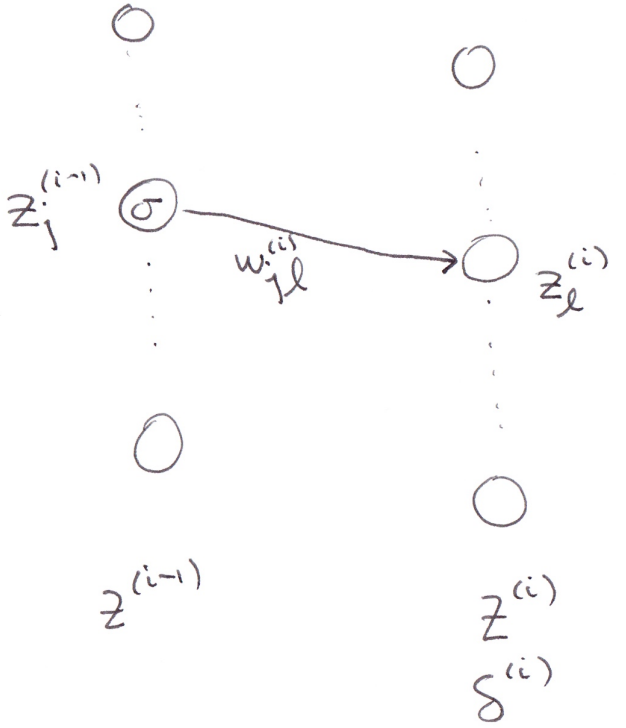
$$\delta_j^{(i-1)} = \sum_l \frac{\partial L}{\partial z_l^{(i)}} \frac{\partial z_l^{(i)}}{\partial z_j^{(i-1)}}$$

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

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

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

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



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

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 $\left(\delta_j^{(\ell)} \sigma(z_i)^{(\ell-1)}\right)$

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 $\frac{\partial \mathcal{L}}{\partial w^{(l)}}$ one each pass

Traning Algorithm

Initalize 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)

Upate 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...