Summary:

$dz^{[2]} = a^{[2]} - y$

$dW^{[2]} = dz^{[2]} a^{[1]T}$

$db^{[2]} = dz^{[2]}$

$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]})$ ($n^{[1]}, 1$)

$dW^{[1]} = dz^{[1]} x^T$

$db^{[1]} = dz^{[1]}$

Vectorised:

$J(\dots) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}, y)$

$dZ^{[2]} = A^{[2]} - Y$

$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$

$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis=1, keepdims$

$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]'}(Z^{[1]})$ ($n^{[1]}, m$) ($n^{[1]}, m$) ($n^{[1]}, m$)

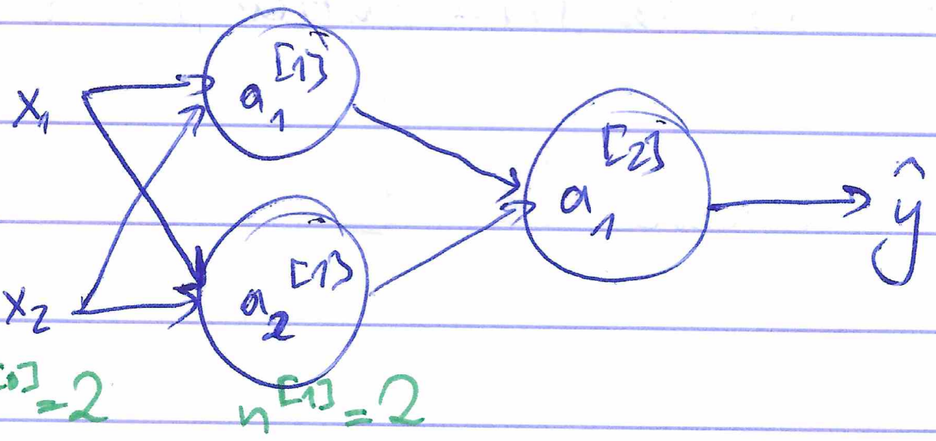
$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$

$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis=1, keepdims$

Full derivation is hard.

Random Initialisation (of weights)

For log. reg., it's OK to init. weights to 0 - not for NN:

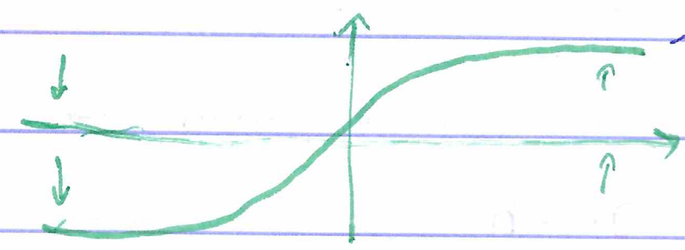


- Identical (symmetrical) hidden units - still computing the same function after iter.

$W^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$, $b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, $W^{[2]} = [0 \ 0]$

$a_1^{[1]} = a_2^{[1]}$, $dz_1^{[1]} = dz_2^{[1]}$

$dW = \begin{bmatrix} u & v \\ u & v \end{bmatrix}$, $W^{[1]} := W^{[1]} - \alpha dW$, $W^{[1]} =$



So:

$W^{[1]} = np.random.randn((2,2)) * 0.01$ → We like very small value

$z^{[1]} = W^{[1]} x + b^{[1]}$

$a^{[1]} = g^{[1]}(z^{[1]})$

if W too large, very large z at start of training

$b^{[1]} = np.zero((2,1))$

No symmetry breaking problem for b