Jupyter / iPython Notebooks

→ Shift + Enter to Execute program

→ Restart ∞ kernel if got any errors

→ submit assignment

Explanation of logistic Regression cost Function

$$\hat{y} = \sigma(w^T x + b) \quad \text{where} \quad \sigma(z) = \frac{1}{1+e^{-z}}$$

Interpret $\hat{y} = P(y=1/x)$

If $y=1$: $P(y/x) = \hat{y}$

$y=0$: $P(y/x) = 1-\hat{y}$

$$P(y/x) = \hat{y}^y (1-\hat{y})^{(1-y)}$$

$$\log P(y/x) = y \log \hat{y} + (1-y) \log (1-\hat{y})$$

$$= -L(\hat{y}, y)$$

cost on m examples

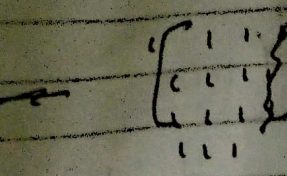
$$\log P(\text{labels in training set}) = \log \prod_{i=1}^{m} P(y^{(i)}/x^{(i)})$$

$$\log P(\dots) = \sum_{i=1}^{m} \log P(y^{(i)}/x^{(i)})$$

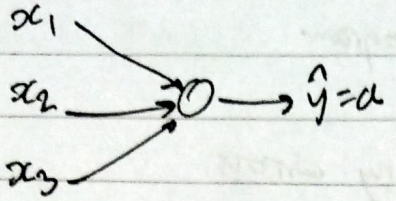
$$-L(\hat{y}^{(i)}, y^{(i)})$$

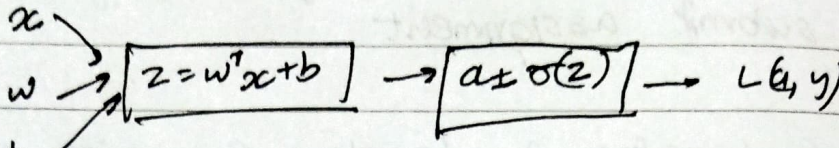
Maximum likelihood estimation

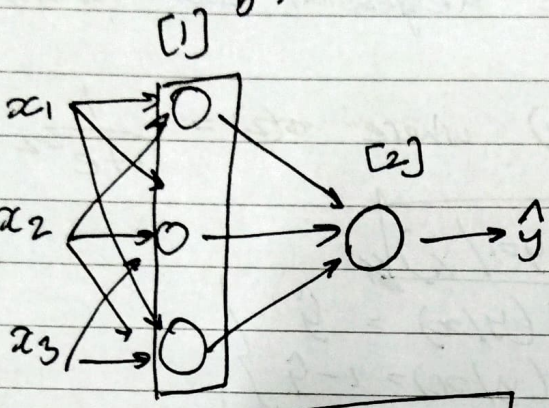
$$\text{cost } J(w,b) = -\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$$

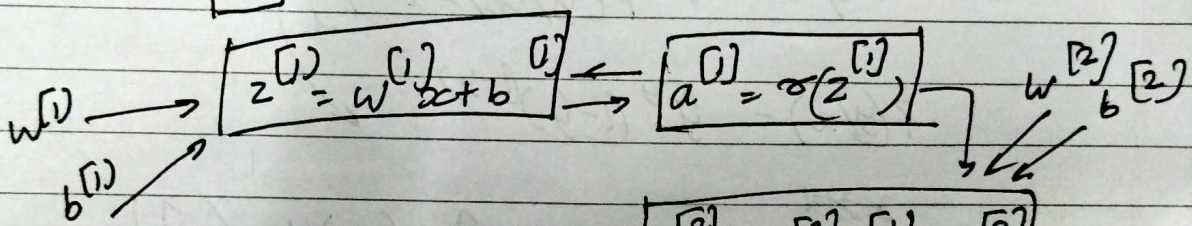


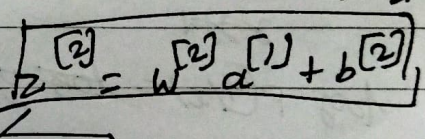
0.5 0.1

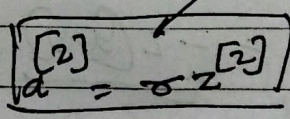


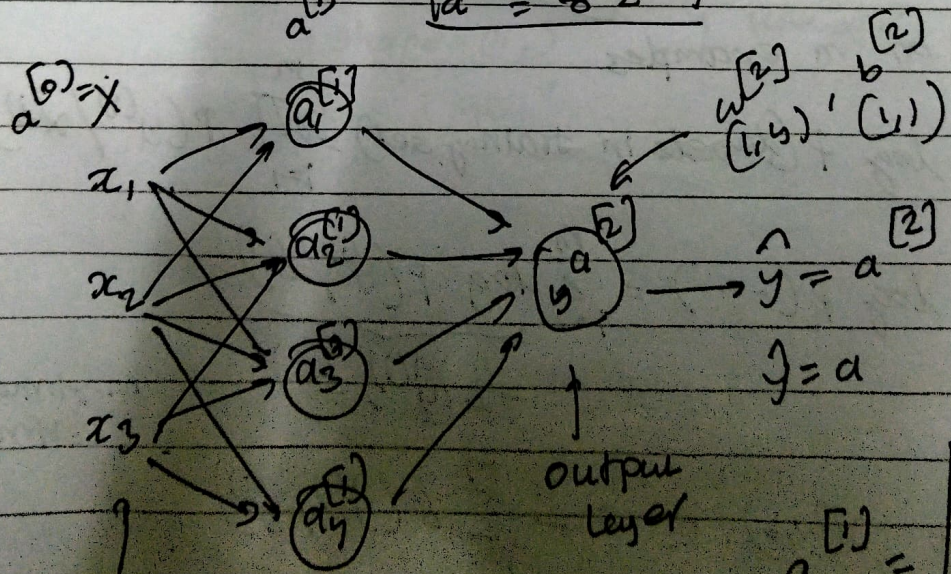






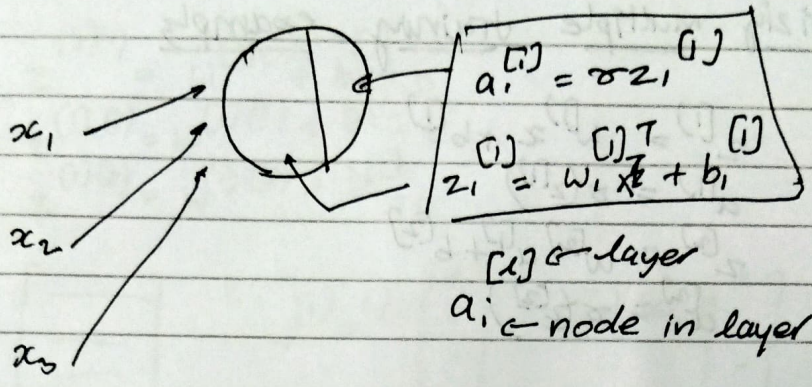


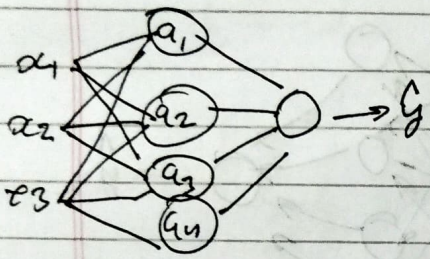




$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}$$

computing a Neural Network's output





Similarly for other nodes

$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}$$
$$a_2^{[1]} = \sigma(z_2^{[1]})$$

$$\begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix} = \begin{bmatrix} (w_1^{[1]})^T \\ w_2^{[1]T} \\ w_3^{[1]T} \\ w_4^{[1]T} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix}$$

4×3 → $W^{[1]}$; $b^{[1]}_{(4,1)}$

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix} = \sigma(z^{[1]})$$

$$z^{[1]} = W^{[1]} x + b^{[1]}$$
$$(4,1)\ (4,3)(3,1)\ (4,1)$$

$$a^{[1]} = \sigma(z^{[1]})$$
$$(4,1)\ (4,1)$$

$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$
$$(1,1)\ (1,4)\ (4,1)\ (1,1)$$

$$a^{[2]} = \sigma(z^{[2]})$$
$$(1,1)\ (1,1)$$

Vectorizing multiple training examples

$$z^{[1]} = W^{[1]} x + b^{[1]}$$
$$a^{[1]} = \sigma(z^{[1]})$$
$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$
$$a^{[2]} = \sigma(z^{[2]})$$

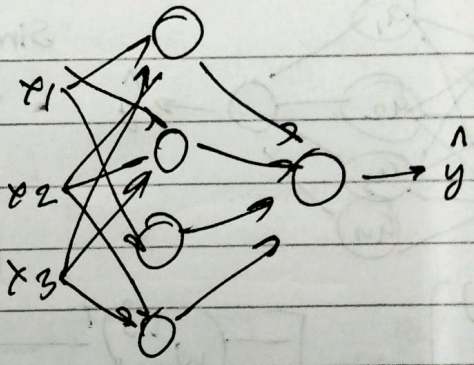
$x \longrightarrow a^{[2]} = \hat{y}^{[2]}$

$x^{(1)} \longrightarrow a^{[2](1)} = \hat{y}^{(1)}$

$x^{(2)} \longrightarrow a^{[2](2)} = \hat{y}^{(2)}$

$\vdots$

$x^{(m)} \longrightarrow a^{[2](m)} = \hat{y}^{(m)}$



$a^{[2](i)}$ → example i, layer 2

$$z^{[1]} = W^{[1]} A^{[0]} + b^{[1]}$$

for $i = 1$ to m:

$$z^{[1](i)} = W^{[1]} x^{(i)} + b^{[1]} \rightarrow Z^{[1]} = W^{[1]} X + b^{[1]}$$
$$a^{[1](i)} = \sigma(z^{[1](i)}) \rightarrow A^{[1]} = \sigma(Z^{[1]})$$
$$z^{[2](i)} = W^{[2]} a^{[1](i)} + b^{[2]} \rightarrow Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$$
$$\rightarrow A^{[2]} = \sigma(Z^{[2]})$$

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \quad (n_x, m)$$

$$Z^{[1]} = \begin{bmatrix} | & | & & | \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ | & | & & | \end{bmatrix}$$

training examples

$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

hidden units → nodes in neural network

Justification for vectorized implementation

$$z^{[1](1)} = w^{[1]}x^{(1)} + b^{[1]} \to 0$$
$$z^{[1](2)} = w^{[1]}x^{(2)} + b^{[1]} \to 0$$
$$z^{[1](3)} = w^{[1]}x^{(3)} + b^{[1]} \to 0$$

$$w^{[1]} = \begin{bmatrix} \text{---} \\ \text{---} \\ \text{---} \end{bmatrix} \quad w^{[1]}x^{(1)} = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix} \quad w^{[1]}x^{(2)} = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix} \quad w^{[1]}x^{(3)} = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix}$$

$$w^{[1]} \begin{bmatrix} | & | & | \\ x^{(1)} & x^{(2)} & x^{(3)} \\ | & | & | \end{bmatrix} = \begin{bmatrix} \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \end{bmatrix} = \begin{bmatrix} z^{[1](1)} & z^{[1](2)} & z^{[1](3)} \\ | & | & | \end{bmatrix} = Z^{[1]}$$

$$+ b^{[1]} \quad + b^{[1]} \quad + b^{[1]}$$

$$w^{[1]}x^{(i)} = z^{[1](i)}$$

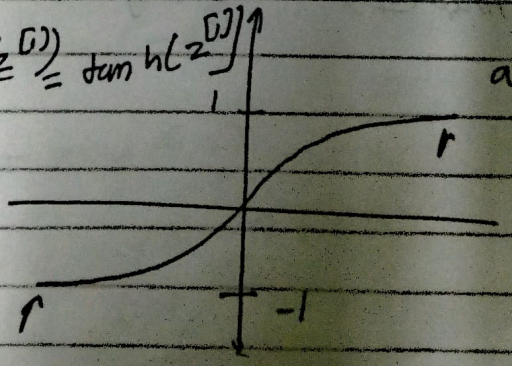
$$z^{[1]} = w^{[1]}X + b^{[1]}$$

ACTIVATION FUNCTIONS

$\sigma \to$ Activation function $\quad \sigma = \frac{1}{1+e^{-z}}$

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

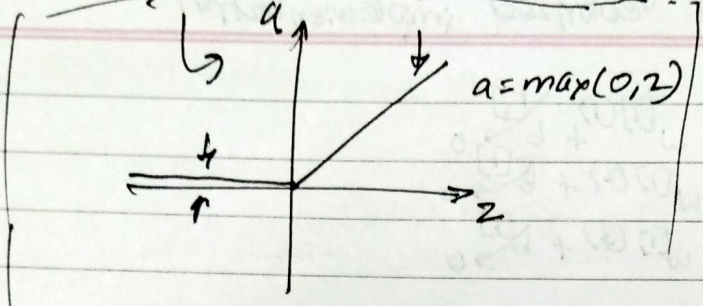
$g(z^{[1]}) = \tanh(z^{[1]})$



$a = \tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$

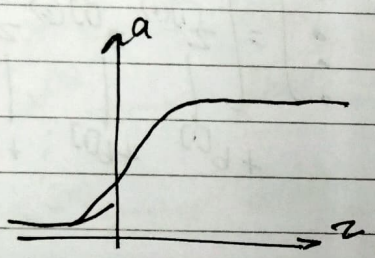
yeh is se sigmoid activation function

ReLU function

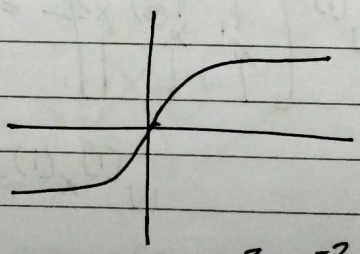


Rectified Linear Unit

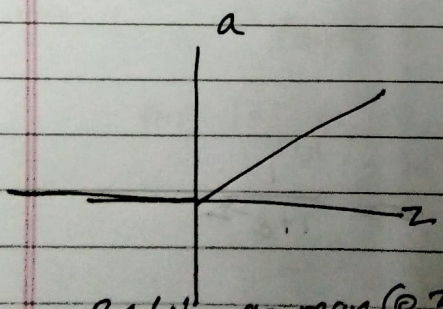
Pros & cons of activation functions



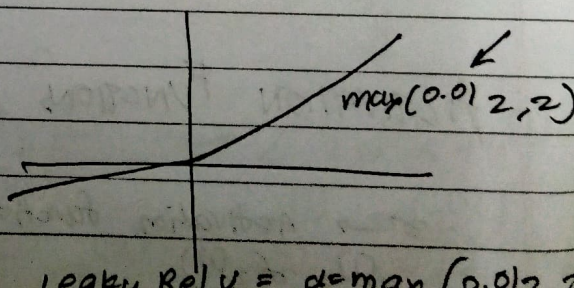
Sigmoid: $a = \frac{1}{1+e^{-z}}$



tanh = $\frac{e^{z}+e^{-z}}{e^{z}+e^{-z}}$



ReLU a = max(0,z)



Leaky ReLU = a = max(0.01z, z)

Why do you need non-linear activation function?

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

$g(z) = z \longrightarrow$ linear activation function

hidden layer doesn't at all come into picture as it just outputs a liner like

Hidden units should not use linear activation function layers

linear activation function are ok for last layer

Slope of functions

$$g(z) = \frac{1}{1+e^{(z)}}$$

$$\frac{d}{dz} g(z) = g(z)(1-g(z))$$

$$g(z) = \tanh(z) = a$$

$$\frac{d}{dz} g(z) = 1 - (\tanh(z))^2$$

$$= 1 - a^2$$

RELU

$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

Leaky RELU

$$g(z) = \max(0.01z, z)$$

$$g'(z) = \begin{cases} 0.01 & z < 0 \\ 1 & z > 0 \end{cases}$$

z & A vectors have matrices have shapes (n, m) — m: training examples

This is no. of neurons in that layer itself

Gradient Descent for neural networks

Parameters $w^{[1]}$ $b^{[1]}$ $w^{[2]}$ $b^{[2]}$

$(n^{[1]}, n^{[0]})$ $(n^{[1]}, 1)$ $(n^{[2]}, n^{[1]})$ $(n^{[2]}, 1)$

$n_x = n^{[0]}, n^{[1]}, n^{[2]} = 1$

cost function

$$J(w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}) = \frac{1}{m} \sum_{i=1}^{n} L(\hat{y}, y)$$

$\hat{y} = a^{[2]}$

gradient descent:

Repeat {

compute predictions $(\hat{y}^{(i)}, i = 1 \dots m)$

$$dw^{[1]} = \frac{dJ}{dw^{[1]}}, \quad db^{[1]} = \frac{dJ}{db^{[1]}}, \dots$$

$w^{[1]} = w^{[1]} - \alpha\, dw^{[1]}$

$b^{[1]} = b^{[1]} - \alpha\, db^{[1]}$

$w^{[2]} = w^{[2]} -$

Forward Propagation

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

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

$z^{[2]} = w^{[2]} x + b^{[2]}$

$A^{[2]} = g^{[2]}(z^{[2]})$

Backpropagation

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

$dw^{[2]} = \frac{1}{m} dz^{[2]} A^{[1]T}$

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

axis = 1 horizontally

keepdims → to save from horizo funny $(3,)$ outputs

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

$(n^{[1]}, m)$ $(n^{[1]}, m)$ ← elementwise product

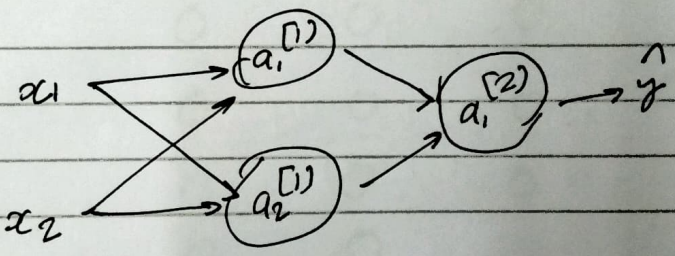
$$dw^{[1]} = \frac{1}{m} dz^{[1]} x^T$$

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

$(n^{[1]}, 1)$ Not $(n^{[1]}, 0)$

So keepdims used

RANDOM INITIALISATION



$n^{[0]} = 2$ $n^{[1]} = 2$

If you initialise weights to zero.

$$w^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \quad b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

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

$$w_2^{[2]} = [0 \; 0]$$

$$\therefore dw = \begin{bmatrix} u & v \\ u & v \end{bmatrix}$$

$$w^{[1]} = w^{[1]} - \alpha dw$$

$$w^{[1]} = \begin{bmatrix} \cdot & - \\ - & - \end{bmatrix}$$

$\therefore w^{[1]} = np.random.randn((2,2)) * 0.01$

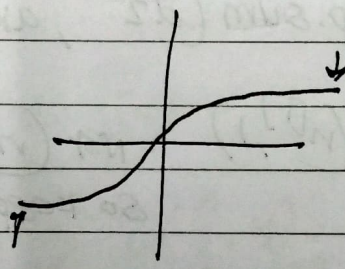
$b^{[1]} = np.zero((2,1))$

why?

$w^{[2]} = np. - - - - -$

$b^{[2]} = 0$

So we want small random values nor huge as they will affect



↓ As you will end up at the large values of z saturating & slowing learning algorithm