Week 2

→ Binary Classification (output values b/w 0 and 1)

→ Logistic Regression

Notations used

(x, y) → single training example where $x \in \mathbb{R}^{n_x}$ (feature vector), $y \in \{0, 1\}$

m = training examples : $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}) \ldots \ldots (x^{(m)}, y^{(m)})\}$

(M_{train})

M_{test} = test examples

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \ldots & x^{(m)} \\ | & | & & | \end{bmatrix} \quad n_x \downarrow \quad \leftarrow m \rightarrow$$

(this structure makes the implementation easier)

$X \in \mathbb{R}^{n_x \times m}$ → dimensional matrix of $n_x \times m$

X.shape → gives output (n_x, m)

$$Y = [\, y^{(1)}, y^{(2)} y^{(3)} \ldots y^{(m)} \,]$$

$Y \in \mathbb{R}^{1 \times m}$; Y.shape → $(1, m)$

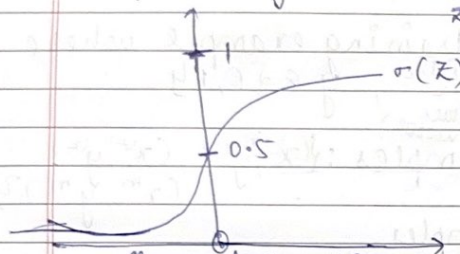
→ Logistic Regression

- learning algorithm that you use when the output labels Y in a supervised learning problem are all either zero or one
- Input given x, we want $\hat{y} = P(y = 1 / x)$

$w, n^x \Rightarrow$ dimensional vector
$b \Rightarrow$ real number

where $0 \le \hat{y} \le 1$

- Parameters : $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$
- Output $= \hat{y} = \sigma(\underbrace{w^T x + b}_{z})$



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

- If z is large, then $\sigma(z) \cong \frac{1}{1+0} = 1$
- If z = large -ve no., then $\sigma(z) \cong 0$

→ Cost function

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

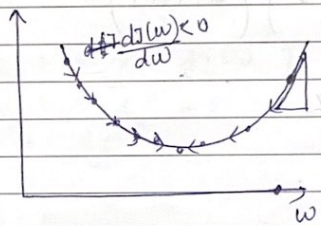
Given $\{(x^{(1)}, y^{(1)}), \dots (x^{(m)}, y^{(m)})\}$, want $\hat{y}^{(i)} \approx y^{(i)}$

- Loss (error) function : $\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$
(optimizatⁿ problem ⇒ you're with multiple local optimum) ~~where~~
- $\mathcal{L}(\hat{y}, y) = -(y \log \hat{y} + (1-y)\log(1-\hat{y}))$
(optimizatⁿ problem ⇒ convex)
if $y=1 \Rightarrow \mathcal{L}(\hat{y}, y) = -\log \hat{y}$ ← want $\log \hat{y}$ large, want $\hat{y}$ large
if $y=0 \Rightarrow \mathcal{L}(\hat{y}, y) = -\log(1-\hat{y})$ ← want $\log(1-\hat{y})$ large, want $\hat{y}$ to be small

- Cost function

$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)})\right]$

→ Gradient Descent



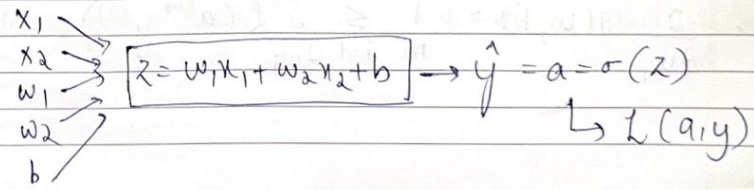
Repeat { $w := w - \alpha \frac{dJ(w)}{dw}$ } (learning rate)

$J(w,b)$ $\quad w := w - \alpha \frac{dJ(w,b)}{dw}$

$b := b - \alpha \frac{dJ(w,b)}{db}$

→ Logistic Regression Gradient descent

$z = w^T x + b$
$\hat{y} = a = \sigma(z)$
$\mathcal{L}(a, y) = -(y \log(a) + (1-y)\log(1-a))$



$$\frac{d\mathcal{L}(a,y)}{da} = -\frac{y}{a} + \frac{1-y}{1-a}$$

$$\frac{dz}{} = \frac{d\mathcal{L}}{dz} = \frac{d\mathcal{L}(a,y)}{dz}$$

$$= a - y$$

$$= \frac{d\mathcal{L}}{da} \cdot \frac{da}{dz}$$

$$= \left(\frac{-y}{a} + \frac{1-y}{1-a} \cdot (-1)\right)\left(a(1-a)\right)$$

$$\frac{d\mathcal{L}}{dw_1} = "dw_1" = x_1 \cdot dz$$

$$dw_2 = x_2 \cdot dz$$

$$db = dz$$

$$w_1 := w_1 - \alpha dw_1$$

$$w_2 := w_2 - \alpha dw_2$$

$$b := b - \alpha$$

→ Gradient descent on m examples

Cost funct^n : $J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(a^{(i)}, y)$

- $a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b)$
- $\frac{\partial}{\partial w_1} J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial w_i} \mathcal{L}(a^{(i)}, y^{(i)})$

- let ~~J=0~~ $J=0$, $dw_1 = 0$, $dw_2 = 0$; $db = 0$
- For $i = 1$ to m

$$z^{(i)} = w^T x^{(i)} + b$$

$$a^{(i)} = \sigma(z^{(i)})$$

$$J += -\left[y^{(i)} \log(a^{(i)}) + (1-y^{(i)}) \log(1-a^{(i)})\right]$$

$$dz^{(i)} = a^{(i)} - y^{(i)}$$

$$dw_1 += x_1^{(i)} dz^{(i)}$$

$$dw_2 += x_2^{(i)} dz^{(i)}$$

$$db += dz^{(i)}$$

$J /= m$

$dw_1 /= m$, $dw_2 /= m$, $db /= m$

$$w_1 := w_1 - \alpha dw_1$$

$$w_2 := w_2 - \alpha dw_2$$

$$b := b - \alpha db$$

→ Vectorization

- art of eliminating for loops in the code
- Vectorized implementation is faster, computing $w^T x$ in logistic regression

  $Z = np.dot(w, x)$, $Z += B$
- import numpy ⇒ create array A ⇒ create array B ⇒ tic = time.time ⇒ c = np.dot(A, B) ⇒ toc = time.time ⇒ print vectorized version

eg. $u = Av$

$u_i = \sum_j A_{ij} v_j$

```
u = np.zeros^T((n,1))
for i
    for j ...
        u[i] += A[i][j] * v[j]
```

Non vectorised version

$u = np.dot(A, v)$ ⇒ vectorized

eg. $v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \rightarrow u = \begin{bmatrix} e^{v_1} \\ e^{v_2} \\ \vdots \\ e^{v_n} \end{bmatrix}$

```
import numpy as np
u = np.exp(v)
```

→ Vectorizing Logistic Regression

$z^{(1)} = w^T x^{(1)} + b \quad z^{(2)} = w^T x^{(2)} + b \quad z^{(3)} = w^T x^{(3)} + b$

$a^{(1)} = \sigma(z^{(1)}) \quad a^{(2)} = \sigma(z^{(2)}) \quad a^{(3)} = \sigma(z^{(3)})$

$$X = \begin{bmatrix} | & | & | & & | \\ x^{(1)} & x^{(2)} & x^{(3)} & \cdots & x^{(m)} \\ | & | & | & & | \end{bmatrix}$$

$$Z = [z^{(1)} \; z^{(2)} \; z^{(3)} \cdots z^{(m)}] = w^T X + [b \; b \cdots b]$$

$$= [w^T x^{(1)} + b \quad w^T x^{(2)} + b \cdots w^T x^{(m)} + b]$$

$Z = np.dot(w^T, X) + b$

$[a^{(1)} \; a^{(2)} \cdots a^{(m)}] = A = \sigma(Z)$

→ Vectorizing Logistic Regression's Gradient Computation

$dz^{(1)} = a^{(1)} - y^{(1)}$

$dz^{(2)} = a^{(2)} - y^{(2)}$

$\vdots$

$dz^{(m)} = a^{(m)} - y^{(m)}$

$dZ = [dz^{(1)} \; dz^{(2)} \cdots dz^{(m)}] \quad (1 \times m)$

$A = [a^{(1)} \cdots a^{(m)}]$

$Y = [y^{(1)} \cdots y^{(m)}]$

$dZ = A - Y = [a^{(1)} - y^{(1)} \; a^{(2)} - y^{(2)} \cdots a^{(m)} - y^{(m)}]$

```
dw = 0
dw += x^(1) dz^(1)
dw += x^2 dz^(2)
...
dw /= m
```

```
db = 0
db += dz^(1)
db += dz^(2)
...
db += dz^(m)
db /= m
```

$db = \frac{1}{m} \sum_{i=1}^{m} dz^{(i)}$

$= \frac{1}{m} np.sum(dZ)$

$$dw = \frac{1}{m} X\, dz^T$$

$$= \frac{1}{m} \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \begin{bmatrix} dz^{(1)} \\ dz^{(2)} \\ \vdots \\ dz^{(m)} \end{bmatrix}$$

$$= \frac{1}{m} \left[x^{(1)} dz^{(1)} + x^{(2)} dz^{(2)} + x^{(3)} dz^{(3)} \cdots\cdots\cdots x^{(m)} dz^{(m)} \right]$$

* Efficient code

$z = w^T X + b$

$= np.dot(w.T, X) + b$

$A = \sigma(z)$

$dz = A - Y$

$dw = \frac{1}{m} X\, dz^T$

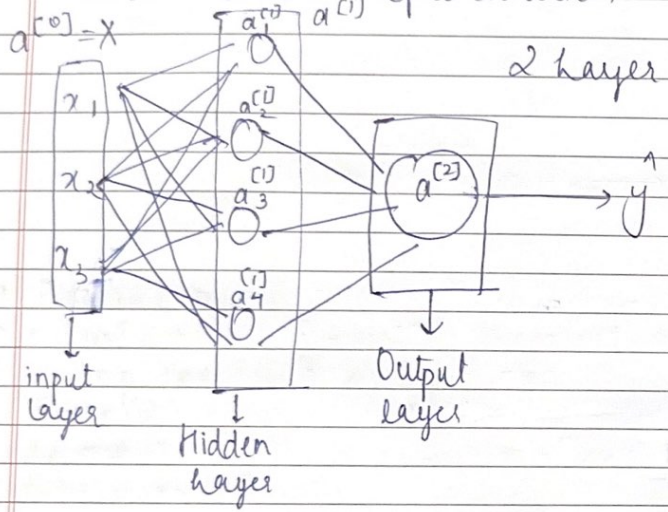
$db = \frac{1}{m} np.sum(dz)$

$w := w - \alpha\, dw$

$b := b - \alpha\, db$

WEEK 3:

→ Neural Network Representation



2 layer NN

→ Computing a Neural Network's Output

~~for 2nd~~

For 1st node

$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}$

$a_1^{[1]} = \sigma(z_1^{[1]})$

$a_1^{[1]} \rightarrow$ layer

$a_1 \rightarrow$ node in layer

Similarly for node 2

$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}$

$a_2^{[1]} = \sigma(z_2^{[1]})$

$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}$, $a_3^{[1]} = \sigma(z_3^{[1]})$

$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}$, $a_4^{[1]} = \sigma(z_4^{[1]})$

→ Vectorizing Across Multiple Examples

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

  $a^{[2](i)} \rightarrow$ example i

  $a^{[2]} \rightarrow$ layer 2

- for $i = 1$ to m:

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

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

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

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

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

$Z^{[1]} = W^{[1]} X + b^{[1]}$

$A^{[1]} = \sigma(Z^{[1]})$

$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$

$A^{[2]} = \sigma(Z^{[2]})$

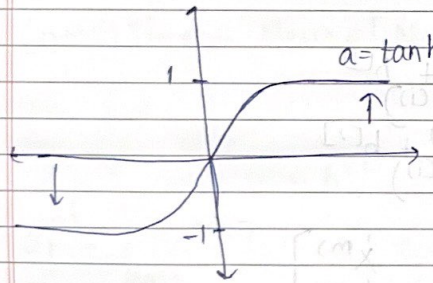
$$Z^{[l]} = \begin{bmatrix} | & | & & | \\ z^{[l](1)} & z^{[l](2)} & \cdots & z^{[l](m)} \\ | & | & & | \end{bmatrix}$$

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

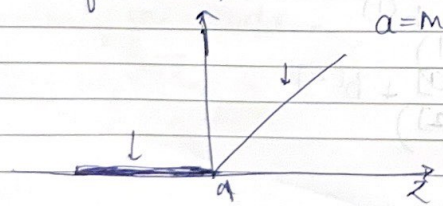
training examples $\longrightarrow$

→ Activation Function (can be different for different layers)

- Sigmoid activation function is the most common, but other choices can work better
- ~~Tanh~~ a tanh(z) activation function is better than sigmoid, as it ranges b/w -1 and 1 and has zero mean



- ReLU function

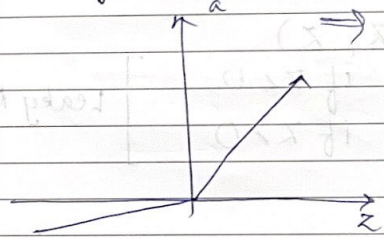


Sigmoid range : $0 \le \hat{y} \le 1$

Tanh(z) range $-1 \le \hat{y} \le 1$

Default values are for Rectified Linear Unit (ReLU)

- Leaky ReLU



⇒ slope of z doesn't become 0, it takes a slight slope

→ Need of Non-Linear Activation Function

Neural networks need a non linear activation function to compute interesting functions

→ Derivatives of Activation functions

- Sigmoid Function

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

$$g'(z) = g(z)(1 - g(z)) = a(1-a)$$

- Tanh function

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

$$g'(z) = 1 - (\tanh(z))^2$$

$$g'(z) = 1 - a^2$$

- ReLU and Leaky ReLU

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

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

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

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

→

WEEK 4

→ Deep L-Layer Neural Network

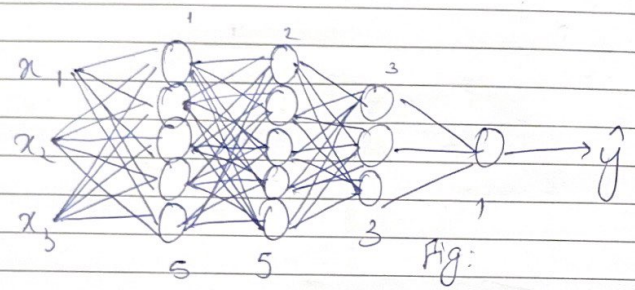


Fig:

$L = 4$ (# layers)

$n^{[l]}$ = # units in layer l

$n^{[1]} = 5,\ n^{[2]} = 5,\ n^{[3]} = 3,\ n^{[4]} = n^{[L]} = 1$

$n^{[0]} = n_x = 3$

$a^{[l]}$ = activations in layer l

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

$w^{[l]}$ = weights for $z^{[l]}$

→ Forward Propagation in a Deep Network

For above figure

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

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

$z^{[2]} = w^{[2]}a^{[1]} + b^{[2]}$

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

$\vdots$

$z^{[4]} = w^{[4]}a^{[3]} + b^{[4]}$

$a^{[4]} = g^{[4]}(z^{[4]})$

General → $z^{[l]} = w^{[l]}a^{[l-1]} + b^{[l]}$

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

Vectorized Version

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

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

$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$

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

$\hat{y} = g(Z^{[4]}) = A^{[4]}$

→ Backpropagation for layer l

- Input $da^{[l]}$
- Output $da^{[l-1]}, dw^{[l]}, db^{[l]}$

$dz^{[l]} = da^{[l]} * g^{[l]\prime}(z^{[l]})$

$dw^{[l]} = dz^{[l]} \cdot a^{[l-1]T}$

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

$da^{[l-1]} = w^{[l]T} \cdot dz^{[l]}$

$dz^{[l]} = w^{[l+1]T} dz^{[l+1]} * g^{[l]\prime}(z^{[l]})$

$dz^{[l]} = dA^{[l]} * g^{[l]\prime}(z^{[l]})$

$dw^{[l]} = \frac{1}{m} dz^{[l]} \cdot A^{[l-1]T}$

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