Neural Network

Logistic Regression - Binary Classification

Notation

(x, y) $\quad x \in \mathbb{R}^{n_x}$ $\quad y \in \{0, 1\}$

m - training example : $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$

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

$X \in \mathbb{R}^{n_x \times m}$

$X.shape = (n_x, m)$

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & \cdots & y^{(m)} \end{bmatrix}$$

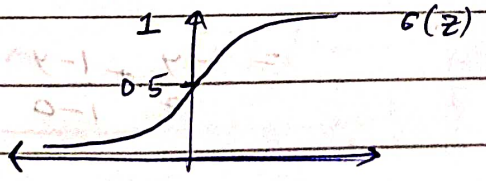
$Y \in \mathbb{R}^{1 \times m}$

$Y.shape = (1, m)$

Given x, $\quad \hat{y} = P(y=1 \mid x)$

$x \in \mathbb{R}^{n_x}$ $\quad$ Parameters : $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$

Output $\hat{y} = \sigma(w^T x + b)$



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

Cost function $\quad$ we want $\hat{y}^{(i)} \approx y^{(i)}$ | Given data. *

Loss function $\quad \mathcal{L}(\hat{y}, y) = -(y \log \hat{y} + (1-y) \log(1-\hat{y}))$

↳ for single training example

Cost function $\quad J(w, b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$

Gradient Descent

Want to find w, b that minimize $J(w,b)$

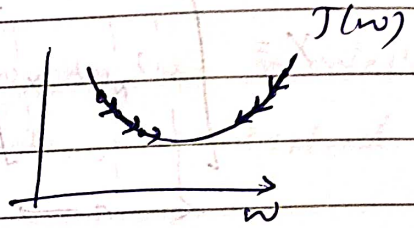
Step toward steepest gradient

Repeat {

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

$\frac{dJ(w)}{dw} \rightarrow dw$

}



$J(w,b)$ $\quad w := w - \alpha dw$

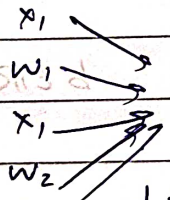
$b := b - \alpha db$

→ for single training example

$z = w^T x + b$

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

$\mathcal{L}(a,y) = -(y \log(a) + (1-y) \log(1-a))$



$z = w_1 x_1 + w_2 x_2 + b \rightarrow a = \sigma(z) \rightarrow \mathcal{L}(a,y)$

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

$= a - y$ $\quad = \frac{-y}{a} + \frac{1-y}{1-a}$

$dw_1 = x_1 \cdot dz$ $\quad dw_2 = x_2 \cdot dz$ $\quad db = dz$

$w_1 := w_1 - \alpha dw_1$

$w_2 := w_2 - \alpha dw_2$

$b := b - \alpha db$

on m example

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

$$a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b)$$

$$\frac{\partial}{\partial w_1} J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \underbrace{\frac{\partial}{\partial w_1} \mathcal{L}(a^{(i)}, y^{(i)})}_{dw_1}$$

$J=0,\ dw_1=0,\ dw_2=0,\ db=0$ $\quad dw = np.zeros((n,x,1$

for $i = 1$ to m

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

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

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

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

for $j = 1$ to n_x (for dw_j):

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

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

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

$$dw \mathrel{+}= x^{(i)} dz^{(i)}$$

$J /= m$

$dw_1 /= m \quad dw_2 /= m \quad db /= m$

$dw /= m$

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

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

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

Vectorization (Avoiding loop)

$z = w^T x + b$ $\quad w = \begin{bmatrix} \\ \\ \end{bmatrix}$ $\quad x = \begin{bmatrix} \\ \\ \end{bmatrix}$

Vectorized	Non vectorized
$z = np.dot(w, x) + b$	for i = 1 to n $z += w[i] * x[i]$ $z += b$

$w^T x$

Faster

eg① $u = Av$

$u_i = \sum_j A_{ij} v_j$

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

```
u = np.dot(A, v)
```

eg② $v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}$ $\quad u = \begin{bmatrix} e^{v_1} \\ \vdots \\ e^{v_n} \end{bmatrix}$

```
u = np.zeros((n,1))
for i in range(n):
    u[i] = math.exp(v[i])
```

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

eg: `np.log(v)`
`np.abs(v)`
`np.maximum(v,0)`

Vectorization (Avoiding loop)

$z = w^T x + b$ $\quad w = \begin{bmatrix} \\ \\ \end{bmatrix}$ $\quad x = \begin{bmatrix} \\ \\ \end{bmatrix}$

Vectorized	Non vectorized
$z = np.dot(w, x) + b$	for i = 1 to n $z += w[i] * x[i]$ $z += b$
($w^T x$)	
↓ Faster	

eg (1) $u = Av$

$u_i = \sum_j A_{ij} v_j$

$u = np.zeros((n,1))$	$u = np.dot(A, v)$
~~$u = np.zeros(n)$~~	
for i for j $u[i] += A[i][j] * v[j]$	

eg (2) $v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}$ $\quad u = \begin{bmatrix} e^{v_1} \\ \vdots \\ e^{v_n} \end{bmatrix}$

```
u = np.zeros((n,1))
for i in range(n):
    u[i] = math.exp(v[i])
```

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

eg: `np.log(v)`
`np.abs(v)`
`np.maximum(v,0)`

Gn logistic regression derivative

J=0 dw = np.zeros((n-x, 1))

Previous pg

Vectorizing LR

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

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

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

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

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

(1,1) → this is called broadcasting

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

$dz^{(1)} = a^{(1)} - y^{(1)}$ $dz^{(2)} = a^{(2)} - y^{(2)}$

$dZ = [dz^{(1)}\ dz^{(2)} \cdots]$

1 x m

we know $A = [a^{(1)} \cdots a^{(m)}]$ $Y = [y^{(1)} \cdots y^{(m)}]$

$dZ = A - Y$

$dw = 0$ $db = 0$

$dw += x^{(1)} dz^{(1)}$ $db += dz^{(1)}$

$dw += x^{(2)} dz^{(2)}$ $db += dz^{(2)}$

$\vdots$ $\vdots$

$dw /= m$ $db /= m$

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

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

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

Implementing in LR

$J=0$, $dw_1 = np.zeros((n_x, 1))$ $db = 0$

for $i = 1$ to m:

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

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

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

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

$dw += x^{(i)} dz^{(i)}$

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

$J = J/m$, $dw = dw/m$ $db = db/m$

for iteration in range

$Z = w^T X + b$

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

$A = \sigma(Z)$

$dZ = A - Y$

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

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

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

$b := b - \alpha db$

Broadcasting

Ex

	Apples	Beef	Egg	Potatoes
Carb	56.0	0.0	4.4	68.0
Protein	1.2	104.0	52.0	8.0
Fat	1.8	135.0	99.0	0.9

$= A_{(3,4)}$

cal = A.sum(axis=0) (axis=1)

↳ sum vertically ↳ sum horizontally

[59, 239 155.4 76.9]

(3,4) (1,4)

percentage = 100 * A / cal.reshape(1,4)

↳ No need (coz already (1,4))

$$\begin{bmatrix} 94.9 & - & - & - \\ 2.03 & - & - & - \\ 3.05 & - & - & - \end{bmatrix}$$

Ex $\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + 100 = \begin{bmatrix} 101 \\ 102 \\ 103 \\ 104 \end{bmatrix}$

$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 100 & 200 & 300 \\ 100 & 200 & 300 \end{bmatrix} = \begin{bmatrix} 101 & 202 & 303 \\ 104 & 205 & 306 \end{bmatrix}$

(m,n) (1,n) → (m,n)

General principle

(m,n) $\pm$ (1,n) → (m,n)

$*$ / (m,1) → (m,n)

```
a = np.random.randn(5)
a.shape = (5,)
"rank 1 array"
```
Don't use

instead `a = np.random.randn(5,1)`

`a.shape = (5,1)`

`assert(a.shape == (5,1))`

`a = a.reshape((5,1))`

Logistic Regression Cost Function

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

If $y=0$ $p(y/x) = 1-\hat{y}$

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

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

If $y=0$ $p(y/x) = 1-\hat{y}$

$$\log p(y/x) = \log \hat{y}^{y}(1-\hat{y})^{(1-y)} = y\log\hat{y} + (1-y)\log(1-\hat{y})$$
$$= -\mathcal{L}(\hat{y}, y)$$

In example

$$\log p(\quad) = \log \prod_{i=1}^{m} p(y^{(i)}/x^{(i)})$$
$$= \sum \underbrace{\log p(y^{(i)}/x^{(i)})}_{-\mathcal{L}(\hat{y}^{(i)}, y^{(i)})}$$

$$= -\sum_{i=1}^{m} \alpha(\hat{y}^{(i)}, y^{(i)})$$

Cost: (minimize) $J(w,b) = \frac{1}{m} \sum_{i=1} \alpha(\hat{y}^{(i)}, y^{(i)})$