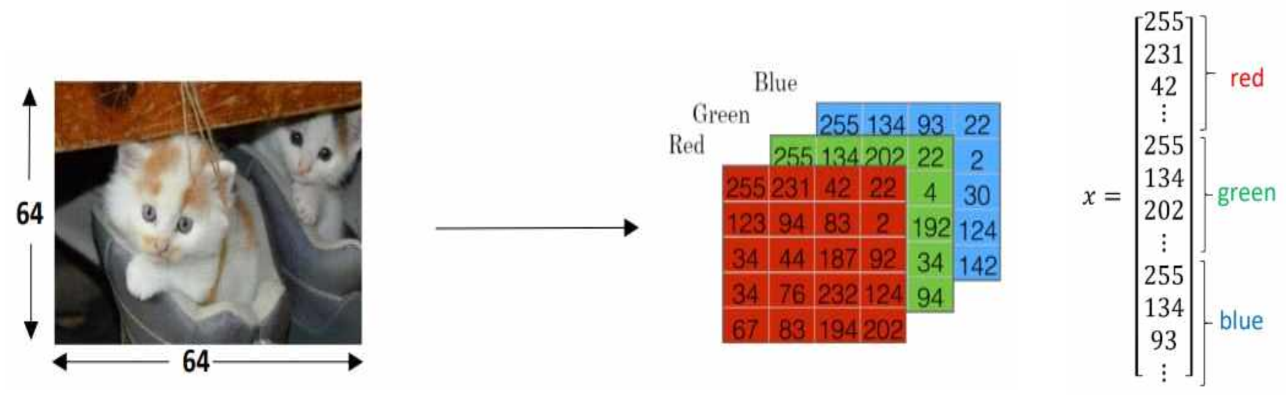
Neural Networks Basics

Binary Classification



64x64x3 = 12288 차원 → $n = n_x = 12288$ 차원

<u>notation</u>

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

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

$m = m_{train} \quad m_{test} = \#test\,example$

$$X = \underbrace{\begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}}_{m} \updownarrow n_x \quad X \in R^{n_x \times m} \quad X.shape = (n_x, m)$$

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & \cdots & y^{(m)} \end{bmatrix} \quad Y \in R^{1 \times m} \quad Y.shape = (1, m)$$

Logistic Regression

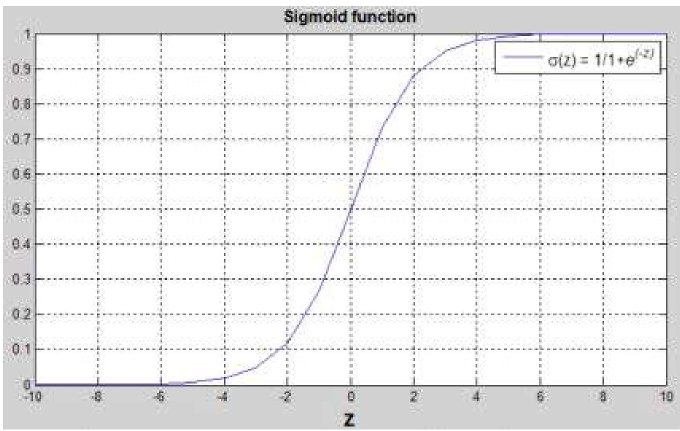
Given $x, where\ x \in R^{n_x}$, want $\hat{y} = P(y=1|x), where\ 0 \le \hat{y} \le 1$

Parameters $w \in R^{n_x}$, $b \in R$

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

$$z = w^T x + b$$

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



Cost Function

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

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

Loss(error) function:

$$L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$ ~ local minima의 가능성 있다

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

if $y^{(i)} = 1$: $L(\hat{y}, y) = -\log \hat{y} \Rightarrow L(\hat{y}, y)\downarrow \ \rightarrow \log \hat{y} \uparrow \ \rightarrow \hat{y} \uparrow \ \rightarrow \hat{y} = 1$

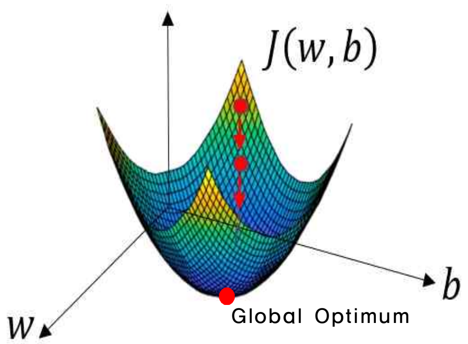
if $y^{(i)} = 0$: $L(\hat{y}, y) = -\log(1-\hat{y}) \Rightarrow L(\hat{y}, y)\downarrow \ \rightarrow \log(1-\hat{y}) \uparrow \ \rightarrow \hat{y} \downarrow \ \rightarrow \hat{y} = 0$

Cost function: mean of Losses

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

Gradient Descent

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



Repeat {

$$w := w - \alpha \frac{\partial J(w, b)}{\partial w}, \ \alpha = learning rate$$

$$b := b - \alpha \frac{\partial J(w, b)}{\partial b}, \ \alpha = learning rate$$

}

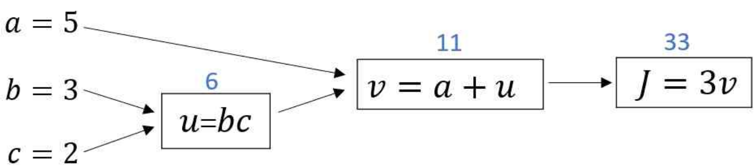
$$\frac{\partial J(w,b)}{\partial w} = dw, \ w := w - \alpha\, dw$$

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

Computation Graph

$u = bc, \ v = a + u, \ J = 3v$

$J(a, b, c) = 3(a + bc) = 3(a + u) = 3v$



Chain Rule에 의해

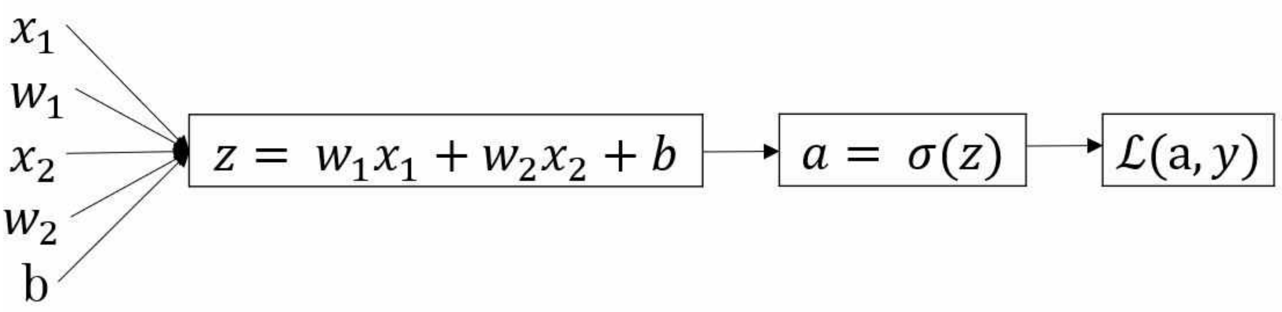
$$\frac{dJ}{db} = \frac{dJ}{du}\frac{du}{db} \ \rightarrow \ \frac{\partial Final\ Output\ Var}{\partial\, Var} = d\,Var$$

Logistic Regression Gradient Decent on Computation Graph

$z = w^T x + b$

$\hat{y} = a = \sigma(z)$

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



$$\begin{aligned} dz &= \frac{\partial L}{\partial z} = \frac{\partial L(a,y)}{\partial z} \\ &= a - y \\ &= \frac{\partial L}{\partial a}\frac{\partial a}{\partial z} = \left(-\frac{y}{a} + \frac{1-y}{1-a}\right)(a(1-a)) \end{aligned}$$

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

$$dw_1 = \frac{\partial L}{\partial w_1} = x_1 dz, \quad dw_2 = \frac{\partial L}{\partial w_2} = x_2 dz, \quad db = dz$$

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

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

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

Gradient Decent on m examples

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

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

$J = 0, \; dw_1 = 0, \; dw_2 = 0, \; db = 0$

for $i = 1$ to m

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

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

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

$$dz^{(i)} = a^{(i)} - y^{(i)}$$
$$dw_1 = dw_1 + x_1^{(i)} dz^{(i)}$$
$$dw_2 = dw_2 + x_2^{(i)} dz^{(i)}$$
$$db = db + dz^{(i)}$$
$$J = J/m$$
$$dw_1 = dw_1/m,\ dw_2 = dw_2/m,\ db = db/m$$

⇒ Vectorization needed for removing 2 loops

1. $dw_1, dw_2, \cdots$
2. for $i = 1$ to m

Vectoring Logistic Regression Derivatives

first step: removing $dw_1, dw_2, \cdots$

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

for $i = 1$ to m

$$z^{(i)} = w^T x^{(i)} + b$$
$$a^{(i)} = \sigma(z^{(i)})$$
$$J = J - (y^{(i)} \log a^{(i)} + (1 - y^{(i)}) \log(1 - a^{(i)}))$$
$$dz^{(i)} = a^{(i)} - y^{(i)}$$
$$dw = dw + x^{(i)} dz^{(i)}$$
$$db = db + dz^{(i)}$$
$$J = J/m$$
$$dw_1 = dw_1/m,\ dw_2 = dw_2/m,\ db = db/m$$

Vectoring Logistic Regression

$$\begin{array}{llll} z^{(1)} = w^T x^{(1)} + b & z^{(2)} = w^T x^{(2)} + b & \cdots & z^{(m)} = w^T x^{(m)} + b \\ a^{(1)} = \sigma(z^{(1)}) & a^{(2)} = \sigma(z^{(2)}) & & a^{(m)} = \sigma(z^{(m)}) \end{array}, \quad X = \underbrace{\begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}}_{m} \updownarrow n_x$$

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

→ $Z = np.dot(w.T, X) + b$ ~ 'Broadcasting': b.shape=(1, 1) → (1, m)

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

$$dz^{(1)} = a^{(1)} - y^{(1)},\ dz^{(2)} = a^{(2)} - y^{(2)},\ \cdots,\ dz^{(m)} = a^{(m)} - y^{(m)}$$

$$\Rightarrow dZ = \left[dz^{(1)}\ dz^{(2)} \cdots dz^{(m)}\right]$$

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

$$\Rightarrow db = \frac{1}{m}\sum_{i=1}^{m} dz^{(i)} = \frac{1}{m} np.sum(dZ)$$

$$\Rightarrow dw = \frac{1}{m}\sum_{i=1}^{m} x^{(i)} dz^{(i)} = \frac{1}{m} X\, dZ^T$$

second step: removing for $i = 1$ to m

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

for $iter$ in range(1000):

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

$$A = \sigma(Z)$$

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

$$dZ = A - Y$$

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

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

$J = J/m$

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

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

Procedure of Building Logistic Regression Neural Network

1. Overview Dataset: #data, dimenstion
2. Activation function
3. Initialize Parameters with zeros or etc
4. Forward Propagation
5. Optimization(Back Propagation)
6. Predict
7. Merge all functions into model