

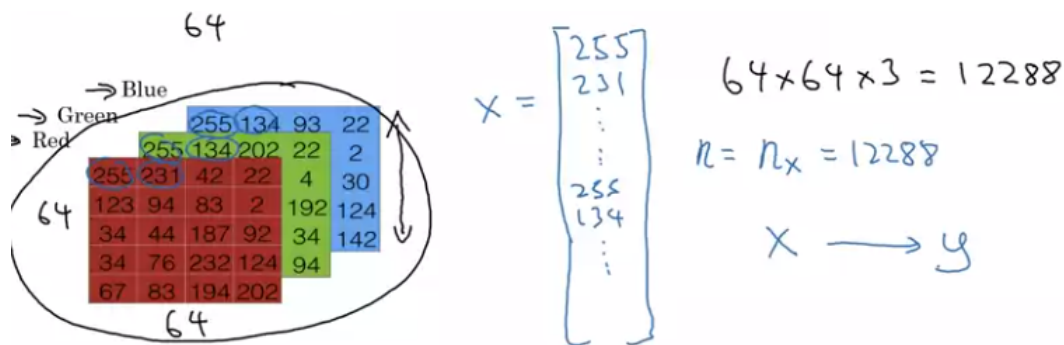
Notations

- $m = m_{train}$ —train examples
- m_{test} — test examples
- $n^{[l]}$ —number of hidden units on layer l
- $g^{[l]}$ — activation function on layer l
- $w^{[l]}, b^{[l]}$ - parameters on layer l
- $dW^{[l]}, dB^{[l]} \dots dA^{[l]}$ - derivatives on layer l
- $par^{[l](i)}$ — the value of example i parameter on layer l
- α — learning rate

Binary classification, Logistic Regression

Input : picture(x) \rightarrow Output : 1(cat) or 0(non cat)

Interpret the picture like a RGB matrix.



RGB matrix

Let, $n_x = 12288$.

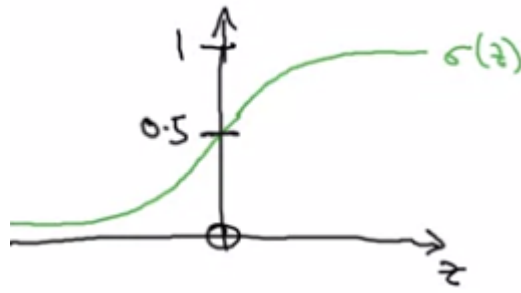
m training examples : $\{(x^{(1)}, y^{(1)}) \dots (x^{(m)}, y^{(m)})\}, \forall i : x^{(i)} \in \mathbb{R}^{n_x}, y^{(i)} \in \{0, 1\}$.

Let, $X = (x^{(1)}, \dots, x^{(m)}) \in Mat_{n_x \times m}, Y = (y^{(1)}, \dots, y^{(m)}) \in Mat_{1 \times m}$.

Given, x , want $\hat{y} = p(y = 1|x), \hat{y} \in [0, 1]$.

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

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



Sigma function

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

$$z \rightarrow \infty \Rightarrow, \sigma(z) \rightarrow 1$$

$$z \rightarrow 0 \Rightarrow, \sigma(z) \rightarrow 0$$

Logistic regression, cost and error function

Let, for i-th example $\hat{y}^{(i)} = \sigma(\omega^T x^{(i)} + b)$, $z^{(i)} = \omega^T x^{(i)} + b$

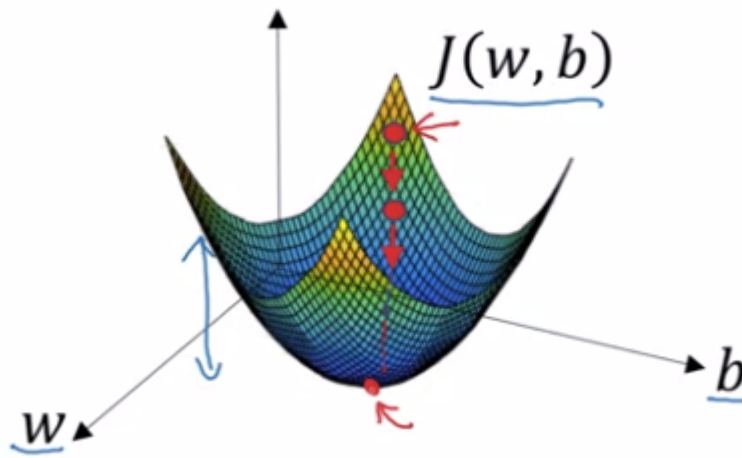
$$\begin{cases} x^{(i)} \\ y^{(i)} \\ z^{(i)} \end{cases} \quad \text{— for i — th example}$$

Loss(error) function : $L(\hat{y}, y) = -(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$ - use for a single test.

Cost function : $J(w, b) = \frac{1}{m} \sum_{i=1}^m L(\hat{y}^{(i)}, y^{(i)})$ - use for train/test dataset (shows how good parameters w and b).

Gradient Descent

Want to find, w, b , than minimize $J(w, b)$. We will use gradient descent.



Gradient descent

Algorithm

Repeat :

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

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

α - learning rate.

Logistic regression on m examples

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

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

Algorithm

$$\begin{aligned}
 &J=0; \underline{dw_1}=0; \underline{dw_2}=0; \underline{db}=0 \\
 &\rightarrow \text{For } i=1 \text{ to } m \\
 &\quad z^{(i)} = \omega^T x^{(i)} + b \\
 &\quad a^{(i)} = \sigma(z^{(i)}) \\
 &\quad J += -[y^{(i)} \log a^{(i)} + (1-y^{(i)}) \log(1-a^{(i)})] \\
 &\quad \underline{dz^{(i)}} = a^{(i)} - y^{(i)} \\
 &\quad \left. \begin{aligned} &dw_1 += x_1^{(i)} dz^{(i)} \\ &dw_2 += x_2^{(i)} dz^{(i)} \\ &db += dz^{(i)} \end{aligned} \right\} n=2 \\
 &\quad J /= m \leftarrow \\
 &\quad dw_1 /= m; dw_2 /= m; db /= m. \leftarrow \\
 &\quad \uparrow \qquad \qquad \uparrow \qquad \qquad \uparrow
 \end{aligned}$$

$$\begin{aligned}
 dw_1 &= \frac{\partial J}{\partial w_1} \\
 w_1 &:= w_1 - \alpha \underline{dw_1} \\
 w_2 &:= w_2 - \alpha \underline{dw_2} \\
 b &:= b - \alpha \underline{db} \\
 &\text{Vectorization}
 \end{aligned}$$

An

Pseudocode

$$\begin{aligned}
 &J = 0, \quad \underline{dw_1} = 0, \quad \underline{dw_2} = 0, \quad db = 0 \qquad dw = \text{np.zeros}((n-x, 1)) \\
 &\rightarrow \text{for } i = 1 \text{ to } m: \\
 &\quad z^{(i)} = w^T x^{(i)} + b \\
 &\quad a^{(i)} = \sigma(z^{(i)}) \\
 &\quad J += -[y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1 - \hat{y}^{(i)})] \\
 &\quad dz^{(i)} = a^{(i)}(1 - a^{(i)}) \\
 &\quad \left. \begin{aligned} &dw_1 += x_1^{(i)} dz^{(i)} \\ &dw_2 += x_2^{(i)} dz^{(i)} \\ &db += dz^{(i)} \end{aligned} \right\} n_x=2 \qquad dw += x^{(i)} dz^{(i)} \\
 &\quad J = J/m, \quad \underline{dw_1} = dw_1/m, \quad \underline{dw_2} = dw_2/m, \quad db = db/m \\
 &\qquad \qquad \qquad dw /= m.
 \end{aligned}$$

A...

Pseudocode with vectorization

Calculate without for-loop

$$X = (x^{(1)}, \dots, x^{(m)}) \in \text{Mat}_{n_x \times m}$$

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

$$Z = (z^{(1)}, \dots, z^{(m)}) = \omega^T X + \left(\underbrace{b \dots b}_m \right) = (\omega^T x^{(1)} + b \dots \omega^T x^{(m)} + b)$$

$$A = \sigma(Z)$$

$$\begin{aligned}
 dz^{(i)} &= a^{(i)} - y^{(i)} & dz^{(2)} &= a^{(2)} - y^{(2)} & \dots \\
 \underline{dz} &= \begin{bmatrix} dz^{(1)} & dz^{(2)} & \dots & dz^{(m)} \end{bmatrix}_{1 \times m} \leftarrow \\
 A &= [a^{(1)} \dots a^{(m)}] & Y &= [y^{(1)} \dots y^{(m)}] \\
 \rightarrow dz &= A - Y = \begin{bmatrix} a^{(1)} - y^{(1)} & a^{(2)} - y^{(2)} & \dots \end{bmatrix} \\
 \left\{ \begin{array}{l} dw = 0 \\ dw += \frac{x^{(i)} dz^{(i)}}{m} \\ dw += \frac{x^{(2)} dz^{(2)}}{m} \\ \vdots \\ dw /= m \end{array} \right. & \left\{ \begin{array}{l} db = 0 \\ db += dz^{(1)} \\ db += dz^{(2)} \\ \vdots \\ db += dz^{(m)} \\ db /= m \end{array} \right. \\
 \end{aligned}$$

$$\begin{aligned}
 db &= \frac{1}{m} \sum_{i=1}^m dz^{(i)} \\
 &= \frac{1}{m} \text{np.sum}(dz) \\
 dw &= \frac{1}{m} X dz^T \\
 &= \frac{1}{m} \begin{bmatrix} x^{(1)} & \dots & x^{(m)} \\ 1 & & 1 \end{bmatrix} \begin{bmatrix} dz^{(1)} \\ \vdots \\ dz^{(m)} \end{bmatrix} \\
 &= \frac{1}{m} \left[\underbrace{x^{(1)} dz^{(1)}}_{n \times 1} + \dots + \underbrace{x^{(m)} dz^{(m)}}_{n \times 1} \right]
 \end{aligned}$$

An

Calculate dz and db and dw without for-loop

```

for iter in range(1000):
    z = w.T * X + b
    = np.dot(w.T, X) + b
    A = sigma(z)
    dz = A - Y
    dw = 1/m * X * dz.T
    db = 1/m * np.sum(dz)

    w := w - alpha * dw
    b := b - alpha * db
    ]

```

Upgraded logistic regression algorithm

Notes

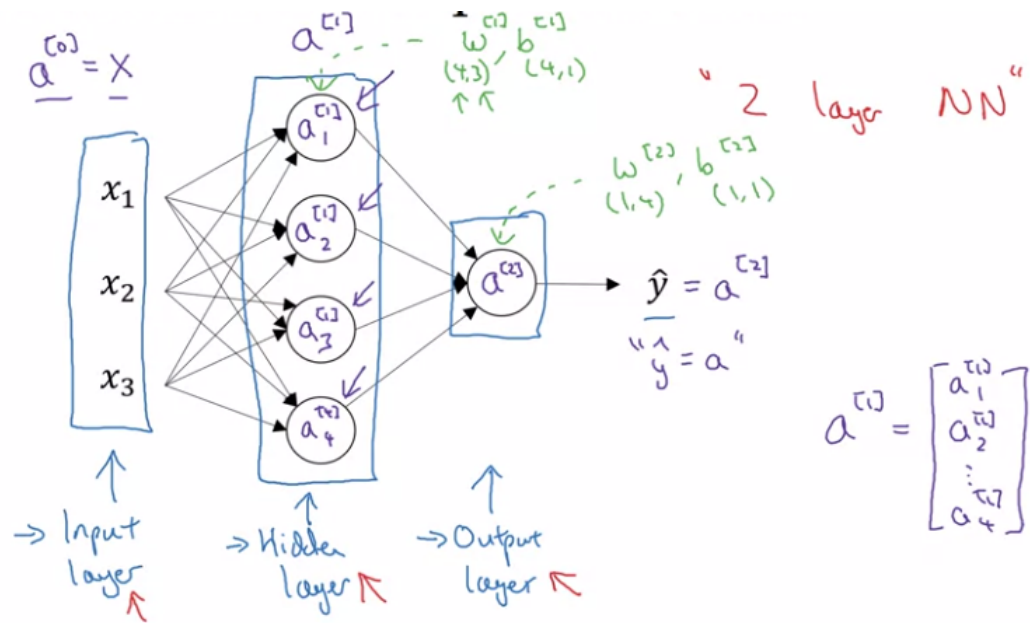
Common steps for pre-processing a new dataset are:

- Figure out the dimensions and shapes of the problem (Mtrain, Mtest, numpx, ...)
- Reshape the datasets such that each example is now a vector of size (numpx * numpx * 3, 1)
- "Standardize" the data

2 layer NN

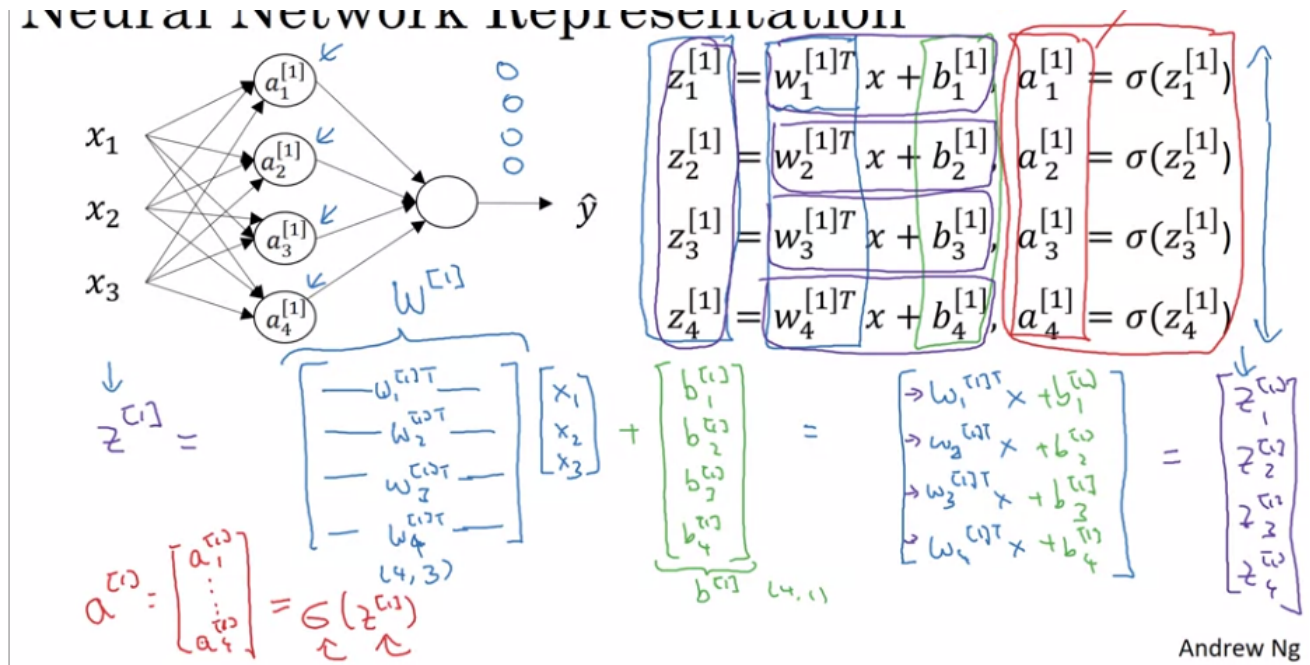
$var_j^{[i]}$ means i-th layer and j-th node in the i-th layer. (picture 2)

$$\begin{aligned}
 \underbrace{x}_{(3 \times 1)} &= \underbrace{a^{[0]}}_{(3 \times 1)} \\
 \underbrace{z^{[1]}}_{(4 \times 1)} &= \underbrace{W^{[1]}x}_{(4 \times 3) \times (3 \times 1)} + \underbrace{b^{[1]}}_{(4 \times 1)} \\
 \underbrace{a^{[1]}}_{(4 \times 1)} &= \underbrace{\sigma(z^{[1]})}_{(4 \times 1)} \\
 \underbrace{z^{[2]}}_{(1 \times 1)} &= \underbrace{W^{[2]}a^{[1]}}_{(1 \times 4) \times (4 \times 1)} + \underbrace{b^{[2]}}_{(1 \times 1)} \\
 \underbrace{a^{[2]}}_{(1 \times 1)} &= \underbrace{\sigma(z^{[2]})}_{(1 \times 1)}
 \end{aligned}$$



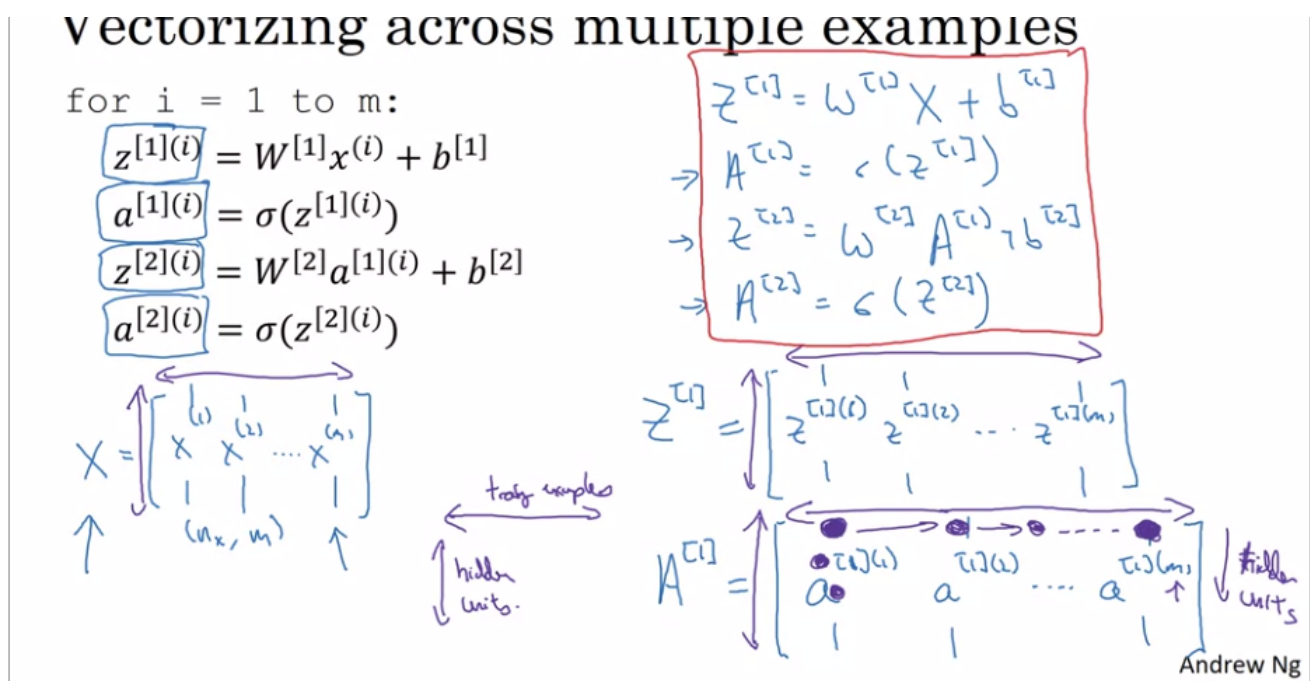
2 layer NN (1)

Neural Network Representation

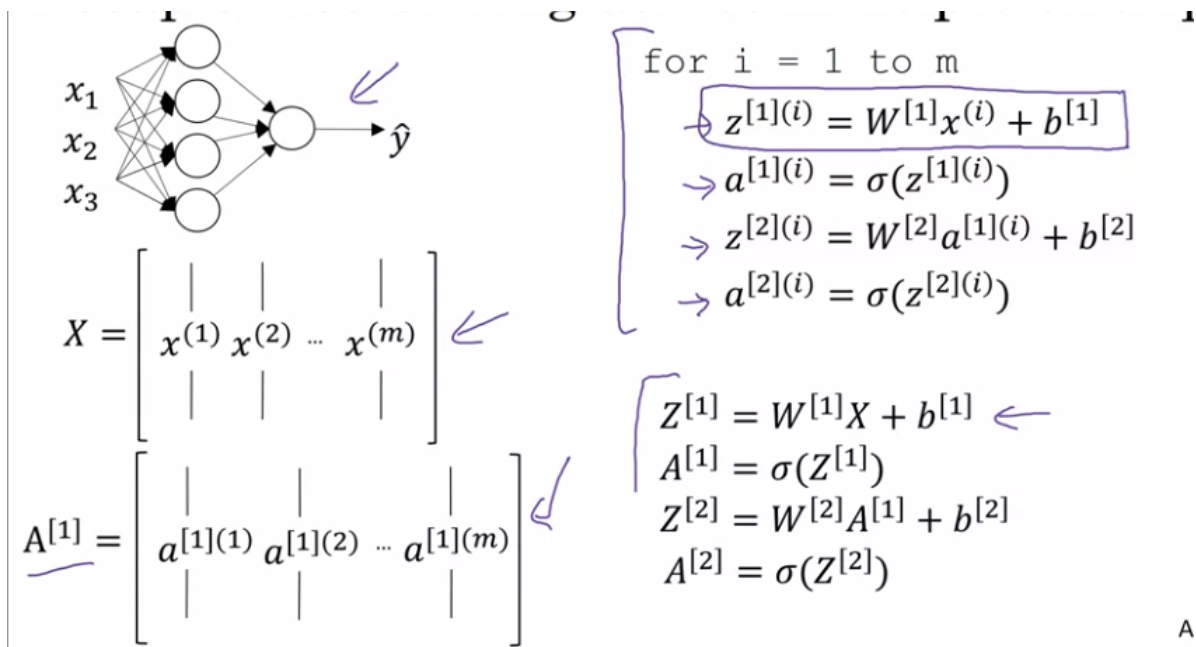


2 layer NN (2)

Vectorizing across multiple examples



Vectorizing across multiple examples



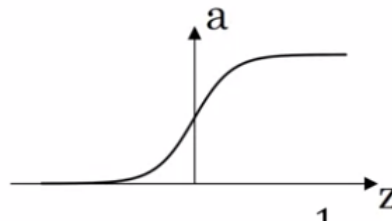
Vectorizing across multiple examples

Activation functions

Sigmoid :

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

$$\sigma(x)' = \sigma(x) \cdot (1 - \sigma(x))$$

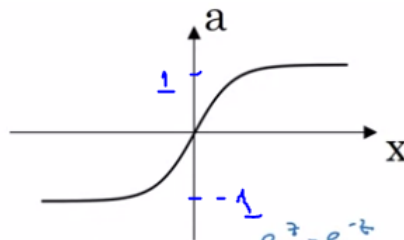


Sigmoid

Hyperbolic tangent :

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

$$\tanh(x)' = 1 - (\tanh(x))^2$$

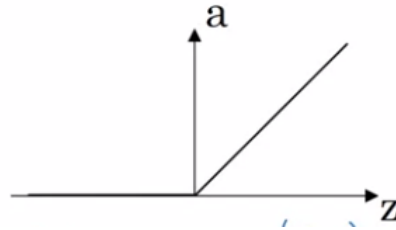


Hyperbolic tangent

Relu :

$$relu(x) = \max(0, x)$$

$$relu(x)' = \begin{cases} 0, & x < 0 \\ 1, & x > 0 \\ \text{undefined}, & x = 0 \end{cases}$$

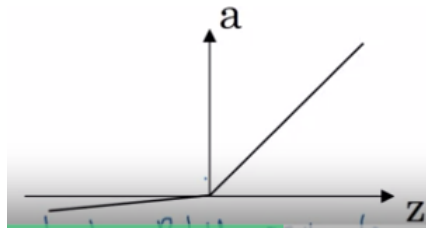


Relu

Leaky Relu :

$$lrelu(x) = \max(0.01 \cdot x, x)$$

$$lrelu(x)' = \begin{cases} 0.01, & x < 0 \\ 1, & x \geq 0 \end{cases}$$



Leaky Relu

Forward and back - propagation for 2-layer NN

Parameters : $\underbrace{w^{[1]}}_{(n^{[1]} \times n^{[0]})}$, $\underbrace{b^{[1]}}_{(n^{[1]} \times 1)}$, $\underbrace{w^{[2]}}_{(n^{[2]} \times n^{[1]})}$, $\underbrace{b^{[2]}}_{(n^{[2]} \times 1)}$

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

Forward propagation :

$$\begin{aligned} Z^{[1]} &= W^{[1]}X + b^{[1]} \\ A^{[1]} &= g^{[1]}(Z^{[1]}) \\ Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\ A^{[2]} &= g^{[2]}(Z^{[2]}) \end{aligned}$$

Back propagation :

$$\begin{aligned} Y &= (y^{(1)} \dots y^{(m)}) \text{ (stacked in column)} \\ dZ^{[2]} &= A^{[2]} - Y \\ dW^{[2]} &= \frac{1}{m} dZ^{[2]} A^{[1]T} \\ dB^{[2]} &= \frac{1}{m} \cdot \text{np.sum}(dZ^{[2]}, \text{axis}=1, \text{keepdims}=\text{True}) \end{aligned}$$

$$\begin{aligned}
dZ^{[1]} &= W^{[2]T} dZ^{[2]} * g^{[1]'}(Z^{[1]}) \\
dW^{[1]} &= \frac{1}{m} dZ^{[1]} X^T \\
dB^{[1]} &= \frac{1}{m} \cdot \text{np.sum}(dZ^{[1]}, \text{axis}=1, \text{keepdims}=\text{True})
\end{aligned}$$

Random Initialization

```

W1 = np.random.randn((n[1], n[0])) * 0.01
b1 = np.zeros((n[1], 1))
W2 = np.random.randn((n[2], n[1])) * 0.01
b2 = np.zeros((n[2], 1))

```

Deep L - layer neural network

Parameters : $w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]} \dots w^{[L]}, b^{[L]}$

Dimensions : $\underbrace{w^{[l]}}_{(n^{[l]} \times n^{[l-1]})}, \underbrace{dW^{[l]}}_{(n^{[l]} \times n^{[l-1]})}, \underbrace{b^{[l]}}_{(n^{[l]} \times 1)}, \underbrace{db^{[l]}}_{(n^{[l]} \times 1)}, \underbrace{Z^{[l]}}_{(n^{[l]} \times m)}, \underbrace{dZ^{[l]}}_{(n^{[l]} \times m)}, \underbrace{A^{[l]}}_{(n^{[l]} \times m)}, \underbrace{dA^{[l]}}_{(n^{[l]} \times m)}, \underbrace{X}_{(n^{[0]} \times m)}$

Forward propagation :

$$\begin{aligned}
\underbrace{Z^{[1]}}_{(n^{[1]} \times m)} &= \underbrace{W^{[1]}}_{(n^{[1]} \times n^{[0]})} \underbrace{X}_{(n^{[0]} \times m)} + \underbrace{b^{[1]}}_{(n^{[1]} \times m)} = (z^{1} \dots z^{[1](m)}) \text{ (stacked in column)} \\
A^{[1]} &= g^{[1]}(Z^{[1]}) = (a^{1} \dots a^{[1](m)}) \text{ (stacked in column)} \\
Z^{[2]} &= W^{[2]} A^{[1]} + b^{[2]} \\
A^{[2]} &= g^{[2]}(Z^{[2]}) \\
&\vdots \\
Z^{[L]} &= W^{[L]} A^{[L-1]} + b^{[L]} \\
A^{[L]} &= g^{[L]}(Z^{[L]})
\end{aligned}$$

Back propagation :

$$\begin{aligned}
\text{input : } dA^{[l]} &\rightarrow \text{output : } dA^{[l-1]}, dW^{[l]}, db^{[l]} \\
dZ^{[l]} &= dA^{[l]} * g^{[l]'}(Z^{[l]}) \\
dW^{[l]} &= \frac{1}{m} dZ^{[l]} A^{[l-1]T} \\
db^{[l]} &= \frac{1}{m} \text{np.sum}(dZ^{[l]}, \text{axis}=1, \text{keepdims}=\text{True}) \\
dA^{[l-1]} &= W^{[l]T} dZ^{[l]}
\end{aligned}$$

Hyperparameters

- Learning rate α
- iterations
- hidden layers L
- hidden units $n^{[1]}, n^{[2]} \dots n^{[L]}$
- choice of activation functions