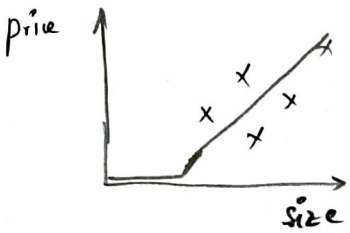
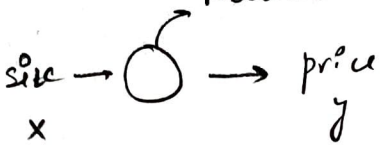
COURSERA Deep Learning Course 1

Neural Networks and Deep Learning

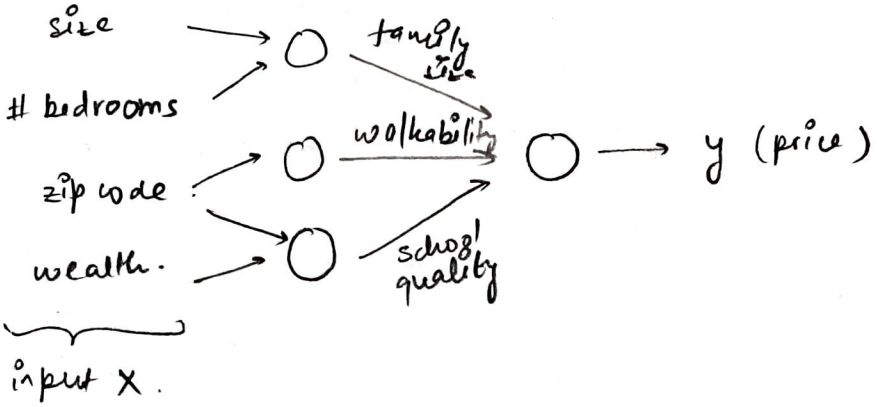
Housing price prediction





↳ Rectified linear unit.

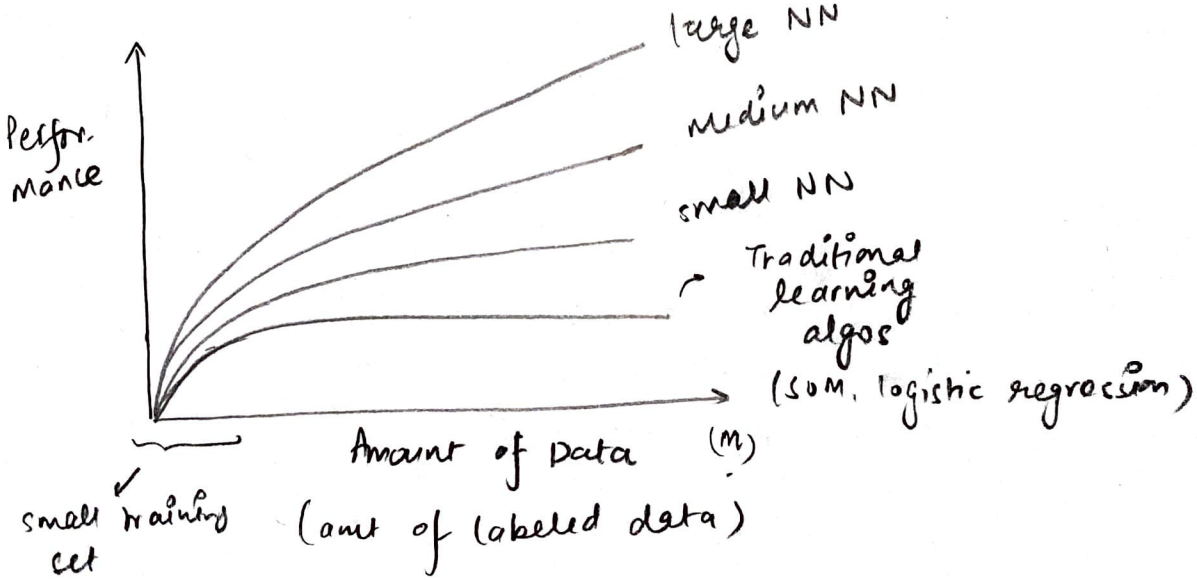
⇒



Structured Data. → proper tabled data, for ex. real estate data.

unstructured data → Audio, images, text

Scale drives deep learning progress



⇒ Binary classification

(input image) ⟶ 1 (cat) vs (0) ↳ non cat

↓

64 x 64 pixels.

y

RGB → 64 x 64 matrices

$$x = \begin{bmatrix} 255 \\ 231 \\ 21 \\ \vdots \\ 255 \\ 134 \\ \vdots \\ 124 \\ 126 \end{bmatrix}$$

↳ 64 x 64 x 3 = 12288

$n = n_x = 12288$

x (input) ⟶ y ↳ 0 or 1

$m = m_{train}$

m_{test} = # test examples.

Notation

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

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

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

(n_x rows, m columns)

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

X shape ⟶ (n_x, m)

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

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

Y shape = $(1, m)$

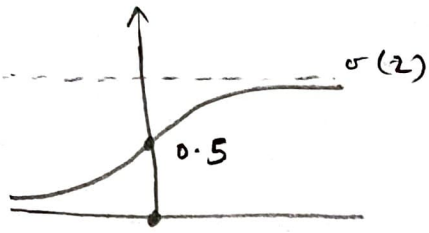
⇒ Logistic Regression → used when $y \in \{0, 1\}$

Given x, want $\hat{y} = P(y = 1 | x)$ → Probability of $y = 1$; given x

$x \in \mathbb{R}^{n_x}$

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

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



$z \uparrow\uparrow$, $\sigma(z) \simeq 1$

$z \downarrow\downarrow$ $\sigma(z) \simeq 0$

$\Rightarrow$ Logistic Regression Cost function

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

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

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

Loss function:

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

If $y=1$: $\mathcal{L}(\hat{y}, y) = -\log \hat{y}$, for minimum $\mathcal{L}(\hat{y}, y)$.
we want $\hat{y}$ to be large. $(\hat{y} \to 1)$

$y=0$: $\mathcal{L}(\hat{y}, y) = -[(1-y)\log(1-\hat{y})] \to \hat{y}$ as small as possible. $(\hat{y} \to 0)$

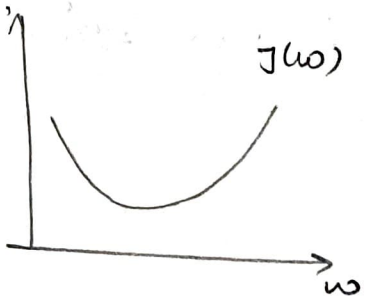
Cost function $J(w,b)$

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

$$-\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)})]$$

$\Rightarrow$ Gradient Descent

→ We want to find w, b so that $J(w,b)$ is minimum



Repeat { ← learning rate.

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

}

"dw" → used in code.

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

$J(w,b)$

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

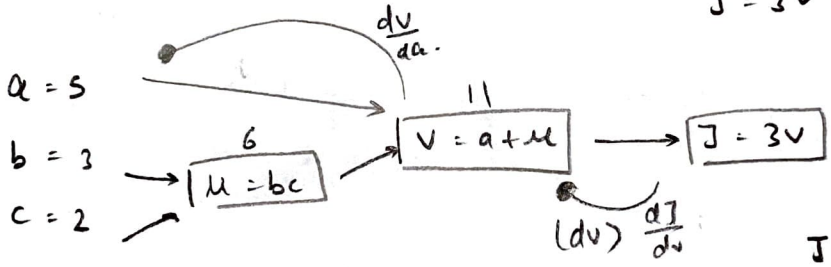
$$b := b - \alpha \frac{\partial (w,b)}{\partial b}$$

} Actual implementation.

⇒ Derivatives

⇒ computation graph.

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



J : final output variable.

$$\frac{dJ}{dv} = 3 \qquad \frac{dJ}{da} = 3 = \frac{dv}{da} \times \frac{dJ}{dv}$$

→ $\frac{d(\text{final output var})}{d\text{var}}$ } → "dvar" → in python code.

→ "dv" = 3 , "da" = 3

$$\frac{dJ}{du} = 3 \quad , \quad du = 3$$

→ $\frac{dJ}{dc} = dc = 3b.$

$$dc|_{b=3} = 9.$$

→ $\frac{dJ}{db} = \frac{dJ}{dv} \times \frac{dv}{du} \times \frac{du}{db}$

$= 3 \times 1 \times c$

$= 3c \quad = db.$

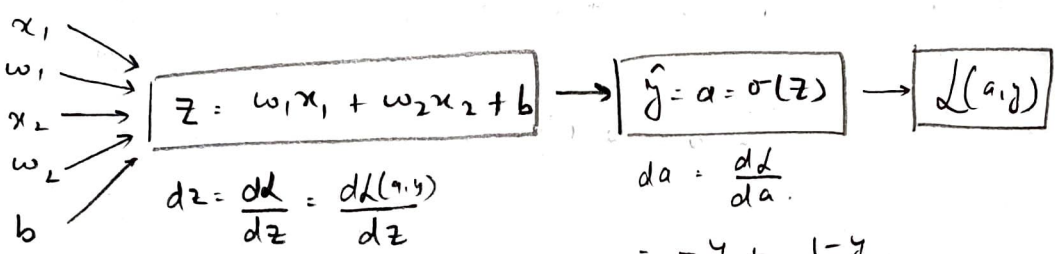
$$db|_{at\ c=2} = 6$$

Gradient Descent for logistic regression (one example)

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

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

let's write as a computation graph., we have two features x_1 and x_2



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

$= (a - y)$

$da = \frac{d\mathcal{L}}{da}$.

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

$= \frac{d\mathcal{L}}{da} \cdot \left(\frac{da}{dz}\right) \rightarrow a(1-a)$.

$\frac{d\mathcal{L}}{dw_1} = "dw_1" = x_1 \cdot dz$ $\quad "dw_2" = x_2 \cdot dz$ $\quad db = dz$.

$$\begin{cases} w_1 := w_1 - \alpha\, dw_1 \\ w_2 := w_2 - \alpha\, dw_2 \\ b := b - \alpha\, db \end{cases}$$

$\Rightarrow$ Gradient Descent on m examples

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

$\rightarrow a^{(i)} = \hat{y}^{(i)} = \sigma(z^{(i)}) = \sigma(w^T x^{(i)} + b)$ $\quad dw_1^{(i)}, dw_2^{(i)}, db^{(i)}$

$$\frac{dJ(w,b)}{dw_1} = \frac{1}{m} \sum \underbrace{\frac{d}{dw_1} \mathcal{L}(a^{(i)}, y^{(i)})}_{dw_1^{(i)}}$$

, Now, calculate as from previous section.

Logistic regression on m examples

$J = 0, \; dw_1 = 0, \; dw_2 = 0 \quad db = 0$ } Initialize.

for i = 1 to m :

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

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

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

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

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

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

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

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

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

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

$J /= m$

$dw_1 = dw_1 / m$

$dw_2 = dw_2 / m.$

⟹ Vectorization

$z = w^T x + b \qquad w = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix} \quad x = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix} \quad w \in \mathbb{R}^{n_x} \quad x \in \mathbb{R}^{n_x}.$

Non vectorised.

z = 0

for i in range n_x.

$z \mathrel{+}= w[i] * x[i]$

$z \mathrel{+}= b.$

Vectorised

$z = np.dot(w, b) + b$

($w^T x$)

→ $v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \quad u = \begin{bmatrix} e^{v_1} \\ e^{v_2} \\ \vdots \\ e^{v_n} \end{bmatrix}$

$u = np.exp(v)$

$u = np.log(v)$

$u = np.abs(v)$

⇒ Vectorising logistic Regression

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

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

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

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

$$\begin{bmatrix} z^{(1)} & z^{(2)} & \cdots & z^{(m)} \end{bmatrix} = w^T X + \underset{(1\times m)}{[b, b \cdots b]}$$

$$w^T = [w_1, w_2 \cdots w_m] \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} + [b \; b \cdots b]$$

$Z = [z^{(1)} \; z^{(2)} \cdots z^{(m)}]$.

$Z = np.dot(w, X) + b$ → $(1,1)$ → but python automatically makes b a row vector.

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

⇒ vectorising Gradient Descent in logistic Regression

$dz^{(1)} = a^{(1)} - y^{(1)}$ $\quad dz^{(2)} = a^{(2)} - y^{(2)}$ $\quad \dots \quad dz^{(m)} = a^{(m)} - y^{(m)}$

$dZ = [dz^{(1)}, dz^{(2)} \cdots dz^{(m)}]_{1\times m}$.

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

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

$dw = 0$

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

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

$\vdots$

dw/m

$db = 0$

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

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

$\vdots$

$db/m.$

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

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

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

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

→ Implementing Logistic Regression

$Z = w^T x + b \quad = np.dot(w.T, x) + b$

$A = \sigma(Z)$

$dz = A - Y$

$dw = \frac{1}{m} x (dz)^T$

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

eg. $w = w - \alpha dw$

$b = b - \alpha db$

Thus, this is a single iteration of Gradient descent.

⇒ Broadcasting in Python

cal : A.sum(axis=0)

axis = 0 → sum downwards ↓

axis = 1 sum horizontally →

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

General principle

(m,n) matrix	$\pm$ / $\times$ /	(1,n) ⇝ (m,n)
		(m,1) ⇝ (m,n)

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

a = np.random.randn(5,1) → a.shape = (5,1) column vector.

a = np.random.randn(1,5) → a.shape = (1,5) row vector.

⇒ Explaination of logistic regression cost function

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

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

If $y=1$: $p(y|x) = \hat{y}$ (probability of $y=1$ is $\hat{y}$)

$y=0$: $p(y|x) = 1-\hat{y}$ (probability of $y=0$ is $1-\hat{y}$)

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

$y=1$: $p(y|x) = \hat{y}^1 (1-\hat{y})^0 = \hat{y}$

$y=0$: $p(y|x) = \hat{y}^0 (1-\hat{y})^1 = 1-\hat{y}$

Lost on m examples

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

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

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

→ np.exp(x) → exponential function to all elements of matrix x.

→ $a = \sigma(z) = \frac{1}{1+e^{-z}}$, $\sigma'(z) = a(1-a)$.

→ $x = \begin{bmatrix} 0 & 3 & 4 \\ 2 & 6 & 4 \end{bmatrix}$ $\|x\|$ along row = np.linalg.norm(x, axis=1, keepdims = True)

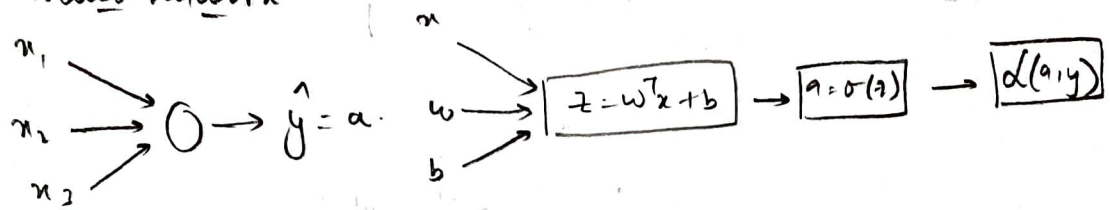
$$\frac{x}{\|x\|} = \begin{bmatrix} 0 & 3/5 & 4/5 \\ \frac{2}{\sqrt{56}} & \frac{6}{\sqrt{56}} & \frac{4}{\sqrt{56}} \end{bmatrix} \quad = \begin{bmatrix} 5 \\ \sqrt{56} \end{bmatrix}$$

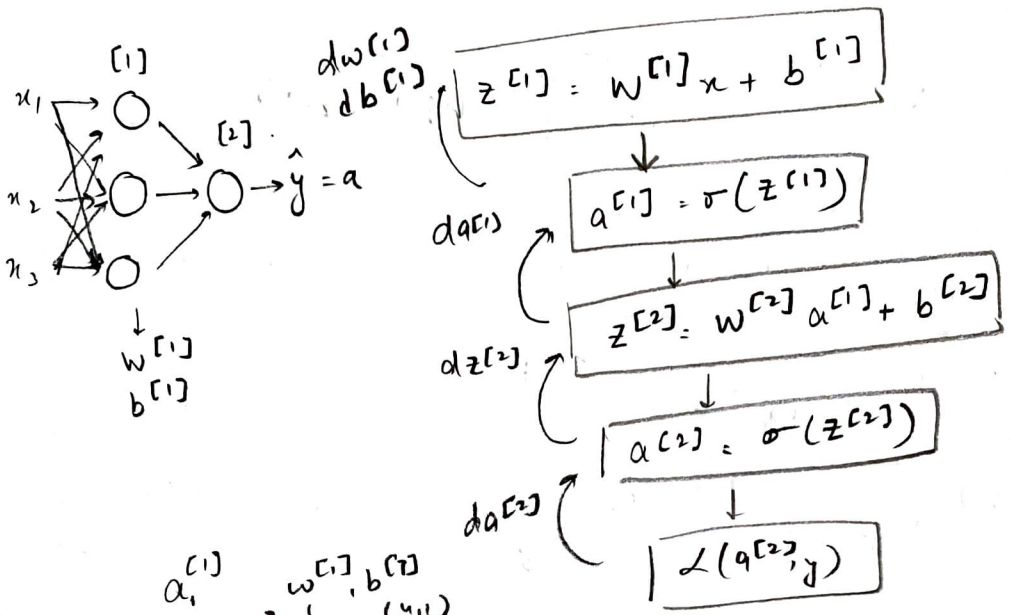
→ keepdims → keepdimensions.

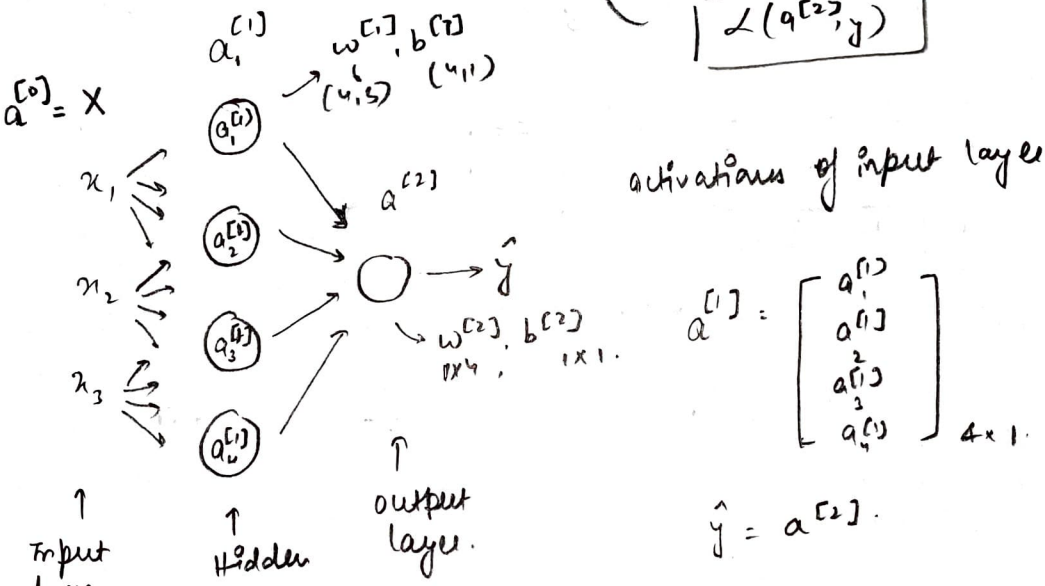
→ x_sum = np.sum(x, axis=1, keepdims = True).

Week-3 Shallow Neural Networks

Neural network





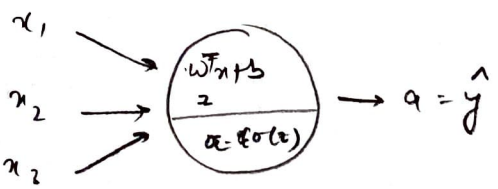


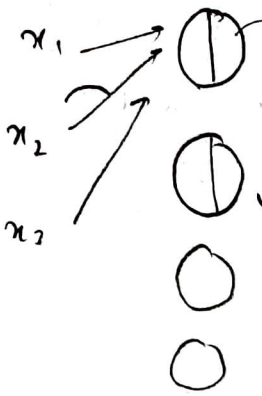
activations of input layer = $a^{[0]}$.

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}_{4\times 1}$$

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

This is a 2 layer. NN.





$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}$$
$$a_1^{[1]} = \sigma(z_1^{[1]})$$

$a_i^{[l]}$ → layer
i → node in layer.

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

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

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

Now, vectorise.

$$w^{[1]T} = \begin{bmatrix} | & | & | & | \\ w_1^{[1]} & w_2^{[1]} & w_3^{[1]} & w_4^{[1]} \\ | & | & | & | \end{bmatrix}_{3\times 4}$$

$$w^{[1]T} = \begin{bmatrix} — & w_1^{[1]T} & — \\ — & w_2^{[1]T} & — \\ — & w_3^{[1]T} & — \\ — & w_4^{[1]T} & — \end{bmatrix}_{4\times 3} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}_{3\times 1} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} =$$

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

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ \vdots \\ a_4^{[1]} \end{bmatrix} = \begin{bmatrix} \sigma(z_1^{[1]}) \\ \vdots \\ \sigma(z_4^{[1]}) \end{bmatrix}$$

Given input $x = a^{[0]}$.

$$z^{[1]}_{(4,1)} = W^{[1]}_{(4\times 3)} a^{[0]}_{(3\times 1)} + b^{[1]}_{(4,1)}$$

W is a vector obtained by stacking up transposes of consecutive w's.

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

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

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

All of this was for one training example, now we, will vectorise for m training examples.

Single training example. $x \longrightarrow a^{[2]} = \hat{y}$

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

For m examples.

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

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

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

$a^{[2](i)}$ → example no° i

→ layer 2.

So, using loop

for i = 1 to m:

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

} using a loop, but we want to vectorize

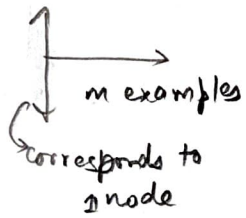
$$X = \begin{bmatrix} | & | & | & | \\ x^{(1)} & x^{(2)} & x^{(3)} & \cdots x^{(m)} \\ | & | & | & | \end{bmatrix} \rightarrow \mathbb{R}^{n_x \times m}$$

(n_x, m)

$$Z^{[1]} = W^{[1]} X + b^{[1]} \quad \text{(all capital cases)}$$

where $Z^{[1]} = \begin{bmatrix} | & | & | & | \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} & \cdots z^{[1](m)} \\ | & | & | & | \end{bmatrix}$

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



$\Rightarrow$ Explaination of vectorised equations

$$z^{[1](1)} = w^{[1]} x^{(1)} + b^{[1]}, \quad z^{[1](2)} = w^{[1]} x^{(2)} + b^{[1]},$$

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

lets say $b^{[1]} = 0$

$$w^{[1]} x^{(1)} = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix}$$ column vector

$$z^{[1](2)} = w^{[1]} x^{(2)} = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix}$$ column vector.

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

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

$$= Z^{[1]}$$

So, for m training examples:-

$$X = A^{[0]}$$

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

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

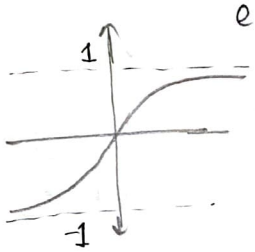
$$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$$

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

⇒ Activation Functions

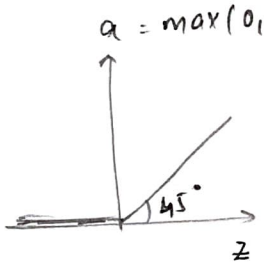
Till now, we have used Sigmoid activation function., so generalising, we would represent $g(z)$.

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



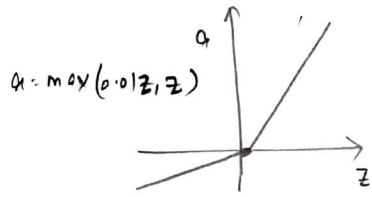
tanh(z) almost always works better as activation function in hidden layers;

slope is nearly zero for $|z| \uparrow\uparrow$, so gradient descent works slowly.



Rectified linear unit (ReLU) → use for hidden layers.

In practice, ReLU is used.



Leaky ReLU (small slope for negative z)

sigmoid(z) → never use except for last layer in binary classification.

ReLU → $a = \max(0, z)$

⇒ Why do we need non linear activation functions?

given x:

$z^{[1]} = w^{[1]}x + b^{[1]}$ $\quad a^{[1]} = g(z^{[1]}) = w^{[1]}(x) + b^{[1]}$

$a^{[1]} = z^{[1]}$ $\quad a^{[2]} = w^{[2]}\left(w^{[1]}x + b^{[1]}\right) + b^{[2]}$

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

$a^{[2]} = z^{[2]}$

↳ so, we will end up with a linear map only.

We can use linear activation function in output layers. (in some cases).

⇒ Derivatives of activation functions

(i) sigmoid

$$g(z) = \frac{1}{1+e^{-z}} \qquad g'(z) = \frac{1}{1+e^{-z}}\left(1 - \frac{1}{1+e^{-z}}\right)$$

$$= g(z)\,(1 - g(z))$$

$$g'(0) = \frac{1}{4} \qquad = a(1-a) \quad \text{as } g(z) = a.$$

(ii) tanh activation function

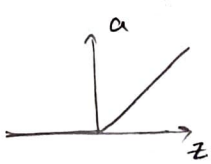
$$g(z) = \tanh(z) \qquad g'(z) = 1 - (\tanh(z))^2$$

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

if $g(z) = a$

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

(iii) ReLU and Leaky ReLU



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

Leaky

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

⇒ Gradient Descent for Neural Network

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

$\hookrightarrow a^{[2]}$

After initializing parameters.

Gradient Descent.

Repeat : {

predict ($\hat{y}^{(i)}$ for $i = 1, \cdots m$)

$dW^{[1]} = \frac{dJ}{dW^{[1]}}$, $db^{[1]} = \frac{dJ}{db^{[1]}}$ ....

$W^{[1]} = W^{[1]} - \alpha\, dW^{[1]}$

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

$W^{[2]} = W^{[2]} - \alpha\, dW^{[2]}$

$b^{[2]} = b^{[2]} - \alpha\, db^{[2]}$

}

Formula for derivatives

$Z^{[1]} = W^{[1]} x + b^{[1]}$ $\quad A^{[1]} = g^{[1]}(Z^{[1]})$, $\quad Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$

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

Back propagation

$dZ^{[2]} = A^{[2]} - Y$ } Assuming binary classification with sigmoid function.

$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$.

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

$dZ^{[1]} = \underbrace{W^{[2]T} dZ^{[2]}}_{(n^{[1]}, m)} \times \underbrace{g^{[1]'}(Z^{[1]})}_{(n^{[1]}, m)}$

$\downarrow$ element wise product

$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$

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

↳ to avoid rank 1 arrays.

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

$\Rightarrow$ x and dx have same matrix dimensions

Summary of Gradient Descent

For single training example.

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

$dW^{[2]} = dz^{[2]} a^{[1]T}$

$db^{[2]} = dz^{[2]}$

$dz^{[1]} = W^{[2]T} dz^{[2]} \times g^{[1]'}(z^{[1]})$

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

$dW^{[1]} = dz^{[1]} x^T$

$db^{[1]} = dz^{[1]}$

For m training examples

$dZ^{[2]} = A^{[2]} - Y$

$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$

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

$dZ^{[1]} = W^{[2]T} dZ^{[2]} \times g^{[1]'}(Z^{[1]})$

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

↓ element-wise multiplication

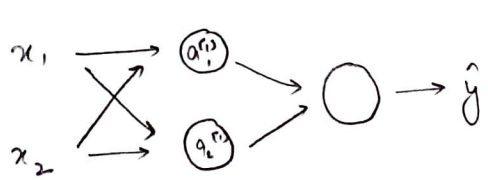
$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$

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

$\Rightarrow$ Random Initialisation

Initialising to 0 won't work

$w^{[1]} = [0 \ 0]$



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

$a_1^{[1]}$ and $a_2^{[1]}$ will be equal. Thus, when we compute back-propagation.

$dz_1^{[1]} = dz_2^{[2]}$

dw becomes $\begin{bmatrix} u & v \\ u & v \end{bmatrix}$ $\quad w^{[1]} = w^{[1]} - \alpha\, dw$

1st row = 2nd row.

So, initialise randomly.

$w^{[1]}$ = np.random.randn((2,2)) * 0.01 → we want weights to be small initially we don't want to end up on flat part of say tanh(z) while backpropagation.

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

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

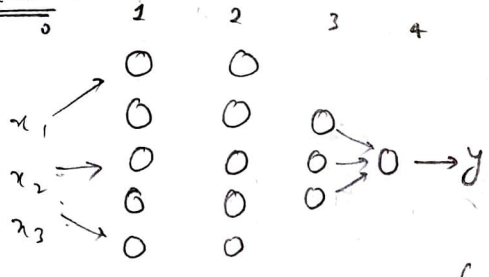
$b^{[2]}$ = 0.

⇒ Week - 4

Deep L-layer neural network

Deep → means more hidden layers

Notation



L = 4 (# layers)

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

no° of nodes in each layer

$n^{[1]} = 5 \quad n^{[2]} = 5$

$n^{[3]} = 3 \quad n^{[4]} = 1$

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

$a^{[l]}$: activations in layer l. $\quad X = a^{[0]} \quad a^{[L]} = \hat{y}$

$a^{[l]} = g^{[l]}(z^{[l]})$, $w^{[l]}$: weights for $z^{[l]}$

$b^{[l]}$.

⇒ forward propagation in deep network

Consider network drawn in previous page

$a^{[0]} = x \rightarrow$ single training example

layer 1

$$z^{[1]} = w^{[1]}x + b^{[1]}$$
$$a^{[1]} = g^{[1]}(z^{[1]})$$

layer 2

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

layer 4

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

z and a small.

So,
$$z^{[l]} = w^{[l]}a^{[l-1]} + b^{[l]}$$
$$a^{[l]} = g^{[l]}(z^{[l]})$$
} general rule

Vectorize → For m training examples.

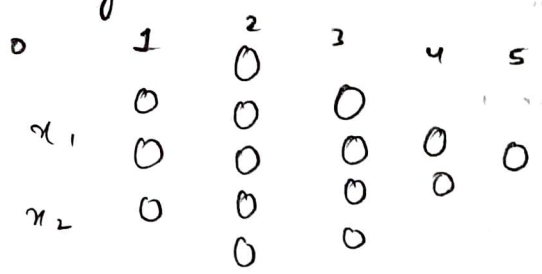
$$z^{[1]} = w^{[1]}X + b^{[1]}$$
$$A^{[1]} = g^{[1]}(z^{[1]})$$
} $X = A^{[0]}$.

$$z^{[2]} = w^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = g^{[2]}(z^{[2]})$$

$$z^{[4]} = w^{[4]}A^{[3]} + b^{[4]}$$
$$A^{[4]} = g^{[4]}(z^{[4]})$$

for loop to calculate

⇒ Getting matrix dimensions right



$L = 5$

$n^{[1]} = 3$

$n^{[2]} = 5$

$n^{[3]} = 4$

$n^{[4]} = 2$

$n^{[5]} = 1$

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

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

$z^{[1]} \longrightarrow (3,1) \rightarrow (n^{[1]}, 1)$, $x \rightarrow (n^{[0]}, 1)$

so $w^{[1]} = (n^{[1]}, n^{[0]})$

generally, $w^{[l]} = (n^{[l]}, n^{[l-1]})$

$w^{[2]} = (n^{[2]}, n^{[1]}) = (5, 3)$

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

$b^{[l]} = (n^{[l]}, 1)$

→ dimensions of $dw = w$.

$dw^{[l]} = (n^{[l]}, n^{[l-1]})$

$db^{[l]} = (n^{[l]}, 1)$

In vectorised implementation

W, b, dW, db dimensions would not change.

$z^{[1]} = w^{[1]} X + b^{[1]}$ → $(n^{[1]}, 1)$ → dimension is still $n^{[1]} \times 1$, it is broadcasted and then added to WX.

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

$Z^{[l]}, A^{[l]} : (n^{[l]}, m)$

$dZ^{[l]}, dA^{[l]} : (n^{[l]}, m)$

Why Deep Representations ?!

Building Blocks of Deep Neural Net

layer l : $W^{[l]}, b^{[l]}$

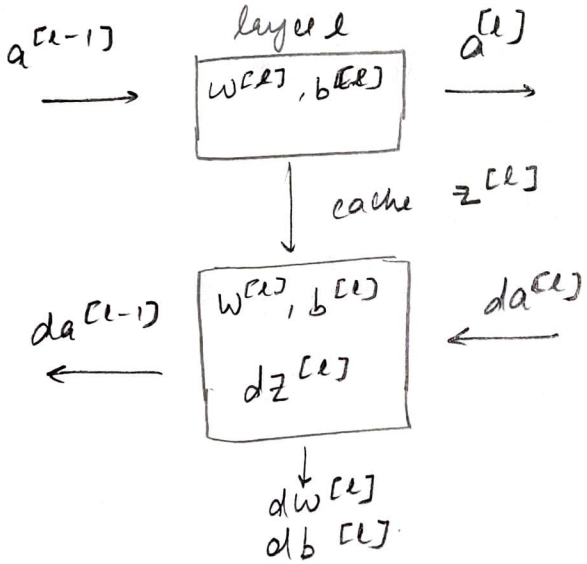
Forward prop : $a^{[l-1]}$, output $\rightarrow a^{[l]}$

$$Z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]} \qquad \text{cache } [Z^{[l]}]$$

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

Back prop. $\rightarrow$ input $da^{[l]}$, output $da^{[l-1]}$

cache $(Z^{[l]})$



⇒ Forward and Backward propagation

FP for layer l

Input $a^{[l-1]}$, output $a^{[l]}$, cache $z^{[l]}$

$$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$$
$$a^{[l]} = g(z^{[l]})$$

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$$
$$A^{[l]} = g(Z^{[l]})$$

Backward propagation for layer l.

Input $da^{[l]}$, output $da^{[l-1]}, dW^{[l]}, db^{[l]}$

$$dz^{[l]} = da^{[l]} * g^{[l]'}(z^{[l]})$$
$$dW^{[l]} = dz^{[l]} \cdot a^{[l-1]T}$$
$$db^{[l]} = dz^{[l]}$$
$$da^{[l-1]} = w^{[l]T} \cdot dz^{[l]}$$

Vectorised

$$dZ^{[l]} = dA^{[l]} * g^{[l]'}(Z^{[l]})$$
$$dW^{[l]} = \frac{1}{m} dZ^{[l]} \cdot A^{[l-1]T}$$
$$db^{[l]} = \frac{1}{m} np.sum(dZ^{[l]}, axis = 1, keepdims = True)$$
$$dA^{[l-1]} = W^{[l]T} dZ^{[l]}.$$

⇒ Parameters v/s Hyperparameters

W, b — Parameters

Hyperparameters: learning rate, no. of iterations, no. of hidden layers L, no. of hidden units $n^{(1)}, n^{(2)} \ldots$, activation functions.