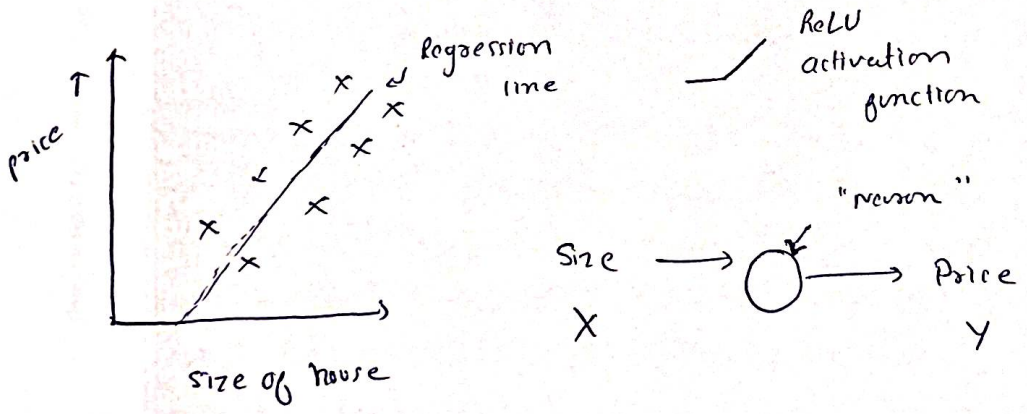
Deep Learning

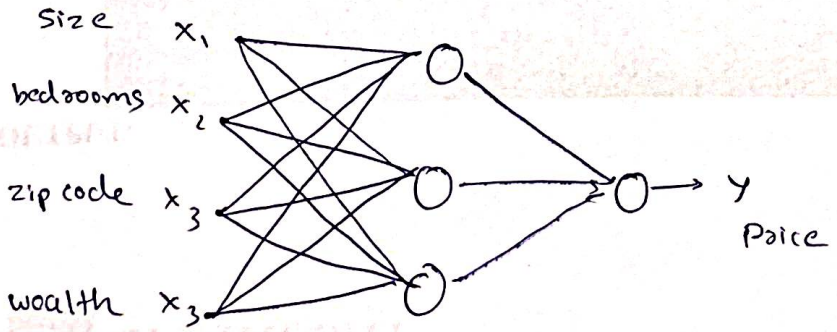
Course 1

* Learning outcomes

1. Neural networks and deep learning
2. Improve Deep learning Networks. Hyperparameter tuning, Regularization and optimization
3. Structuring your ML project [end-to-end DL]
4. Convolutional Neural Networks
5. Natural Longuage Processing : Building Sequence models [RNN, LSTM]

1. Neural Network & DL

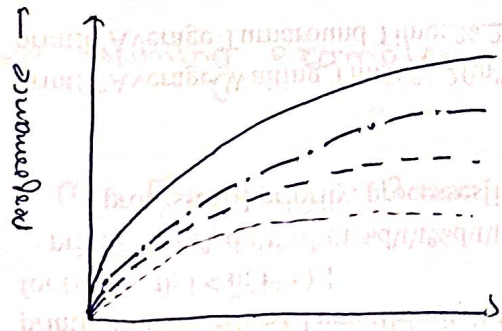




⇒ Supervised Learning [labeled data]**

(Features) Input (x)	(label) output (y)	Application	
Home feat.	Price	Real Estate	} standard NN
Ad, user info	Click on ad?	Online Ad.	
Image	object (1... 1000)	Photo tagging	} CNN
Audio	Text transcript	Speech recog.	} RNN
English	chinese	machine Translation	
Image, Radar info	Position of cars	Autonomous driving	} custom / complex Hybrid NN

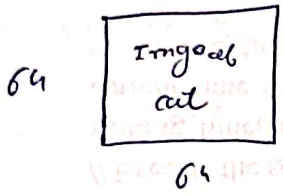
⇒ Scale drives deep learning progress

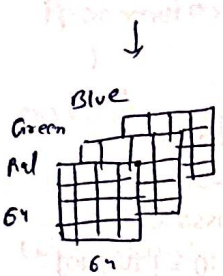


- Data ↑
- Computation ↑
- Algorithms ↑

(labeled data)
(x, y) for training

* Binary classification





Feature vector

$$X = \begin{bmatrix} 255 \\ 125 \\ \vdots \end{bmatrix}$$

no. of feature $= 64 \times 64 \times 3$

$n_x = 12288$

or

$n = 12288$

⇒ Notation

- $(x, y) \quad x \in \mathbb{R}^{n_x}, \quad y \in \{0, 1\}$
- m training examples : $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}) \dots (x^{(m)}, y^{(m)})\}$
  - $m = m_{train}$; m_{test}
  - $$X = \begin{bmatrix} | & | & \dots & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & \dots & | \end{bmatrix} \quad \leftarrow m \rightarrow$$

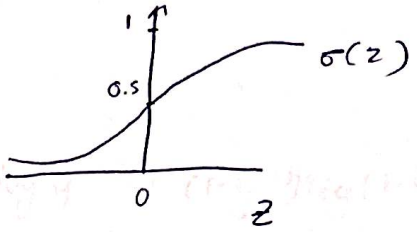
$$Y = [y^{(1)} \; y^{(2)} \dots y^{(m)}]$$

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

$X \in \mathbb{R}^{n_x \times m}$ $\quad X.shape = (n_x, m)$ $\quad Y.shape = (1, m)$

* Logistic Regression

- Given x, $\hat{y} = P(y=1 \mid x)$ $\quad 0 \le \hat{y} \le 1$
- $x \in \mathbb{R}^{n_x}$ → feature vector
- Parameters : $w \in \mathbb{R}^{n_x}$, $b \in \mathbb{R}$
- output $\hat{y} = \sigma(\underbrace{w^T x + b}_{z})$



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

- We have to learn w & b parameter so, that we can get $\hat{y}$ as the probability of $(y=1 \mid x)$.

If z large $\sigma(z) \simeq \frac{1}{1+0} \simeq 1$

If z large -ve $\sigma(z) \simeq \frac{1}{1+\text{Big}} \simeq 0$

⇒ Cost function

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

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

- loss function : must be as small as possible.

$$L(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$, But, we don't use this Because gradient descent doesn't work properly

$$\Rightarrow \boxed{L(\hat{y}, y) = -(y \log \hat{y} + (1-y) \log(1-\hat{y}))}$$ → single training example

If $y=1$: $L(\hat{y}, y) = -\log \hat{y}$ ← want $\log \hat{y}$ large, want $\hat{y}$ large $\simeq 1$

If $y=0$: $L(\hat{y}, y) = -\log(1-\hat{y})$ ← want $\log(1-\hat{y})$ large, want $\hat{y}$ small $\simeq 0$

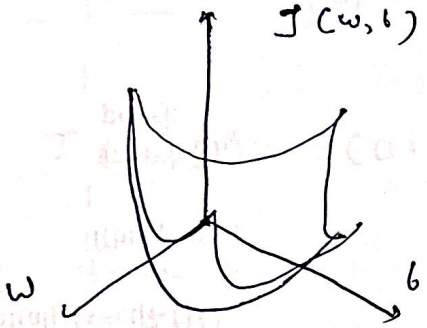
- cost function : $\boxed{J(w,b) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})}$ → see how w & b are working on entire dataset.

* Gradient Descent

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

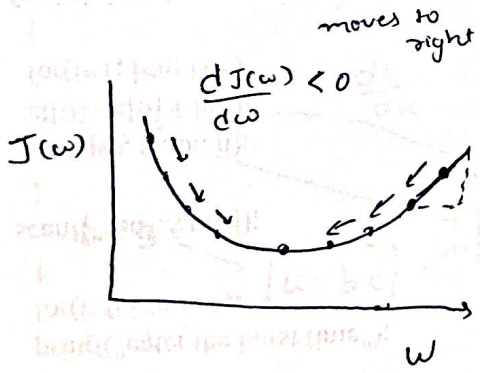
- $J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)})$

- Wont to find optimal w & b that minimize $J(w,b)$



- $J(w,b)$ dimension is always perpependicular to all the parameters dimension.

-



$$w := w - \alpha \left[\frac{dJ(w)}{dw}\right]$$

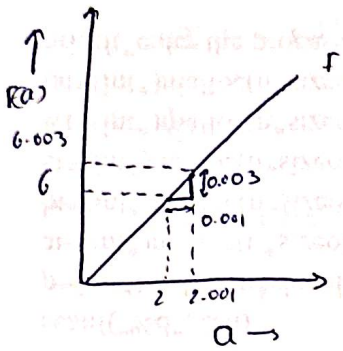
- $\therefore$ $w := w - \alpha \frac{dJ(w)}{dw}$ or $\frac{\partial J(w,b)}{\partial w}$

$d \rightarrow$ for single param.

∂ - for multiple param.

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

⇒ Intuition about derivatives



slope of f(a) = $\frac{\text{height}}{\text{weight}} = \frac{0.003}{0.001} = 3$

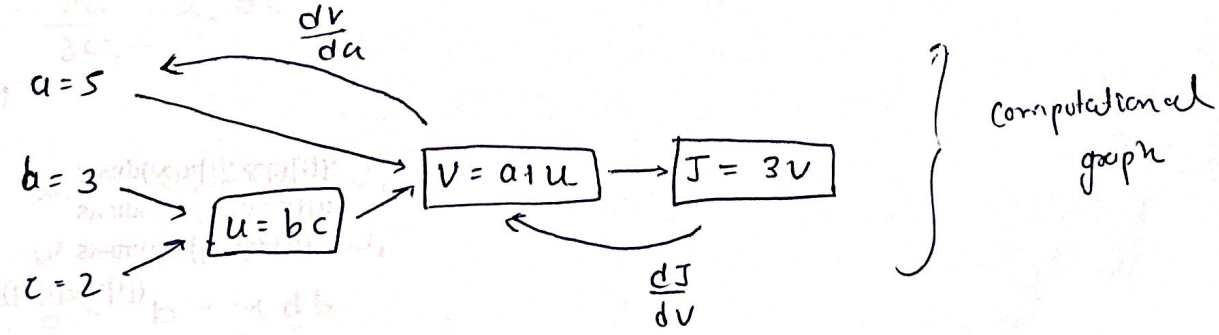
↓

Derivative of f(a) = $\frac{d f(a)}{da}$ → $\frac{\text{change in f(a)}}{\text{change in 'a'}}$

⇒ computational graph

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

- $u = bc$
  $v = a + u$
  $J = 3v$



$\therefore \frac{dJ}{da} = \frac{dJ}{dv} \cdot \frac{dv}{da}$

$\frac{d \text{ final output var}}{d \text{ var}}$

- $\frac{dJ}{du} = \frac{dJ}{dv} \cdot \frac{dv}{du}$

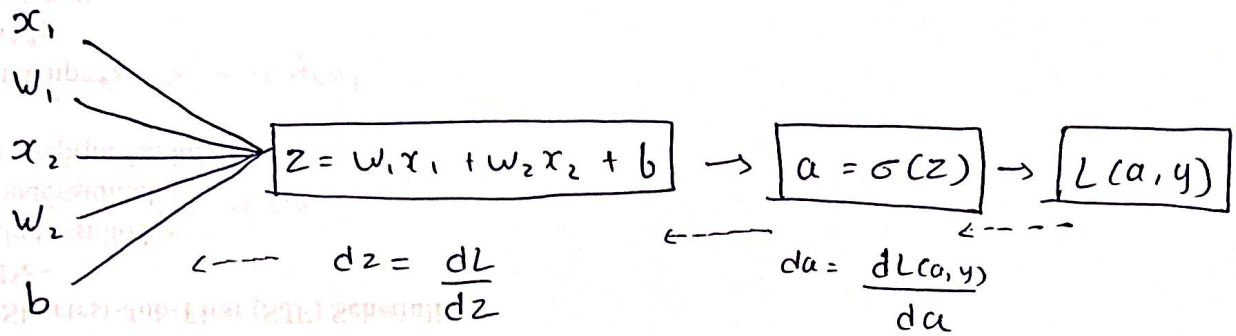
- $\frac{dJ}{db} = \frac{dJ}{du} \cdot \frac{du}{db}$

⇒ logistic regression Gradient Descent

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

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

$$l(a, y) = -(y \log a + (1-y) \log(1-a))$$



$$dz = \frac{dL}{dz} = \frac{dL}{da} \cdot \frac{da}{dz}$$

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

$$= a - y$$

$$dw_1 = \frac{\partial L}{\partial w_1} = x_1 dz$$

$$dw_2 = \frac{\partial L}{\partial w_2} = x_2 dz$$

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

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

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

- Logistic regression for m examples

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

for $i = 1$ to m

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

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

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

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

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

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

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

$J /= m$

$dw_1 /= m$; $dw_2 /= m$; $db /= m$.

$$dw_1 = \frac{\partial J}{\partial w_1}$$

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

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

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

⇒ Vectorization

- used to remove explicit for loops so computational time can be reduced.

$$w = \begin{bmatrix} \vdots \end{bmatrix} \quad x = \begin{bmatrix} \vdots \end{bmatrix} \qquad w \in \mathbb{R}^{n_x} \quad x \in \mathbb{R}^{n_x}$$

- $z = w^T x + b$

Non-vectorized

```
z = 0
for i in range (n-x):
    z += w[i] * x[i]
z += b
```

Vectorized

```
z = np.dot (w, x)
z += b
```

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

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

```
u = np.exp(v)
```

⇒ Vectorization of logistic regression

$J = 0$, ~~$dw_1 = 0$, $dw_2 = 0$~~, $db = 0$

$dw = np.zeros((n_x, 1))$ ← vector

For $i = 1$ to m:

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

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

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

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

for $j = 1$ to n_x:

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

$dw_2 \mathrel{+}= x_2^{(i)} dz^{(i)}$ } $n_x = 2$

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

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

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

dw/m

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

~~$w^T = [\ldots]$~~

$w^T = [\ldots\ldots]$

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

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

Syntax = `z = np.dot(w.T, x) + b`

↓ Real no. converted to 1×1 vector

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

⇒ Vectorizing logistic Regression

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

$$\Rightarrow dz = [dz^{(1)}\ dz^{(2)} \dots dz^{(m)}]_{1 \times m}$$

$$\Rightarrow A = [a^{(1)} \dots a^{m}] \qquad Y = [y^{(1)} \dots y^{(m)}]$$

$$\Rightarrow dz = A - Y = [a^{(1)} - y^{(1)}\ \ a^{(2)} - y^{(2)} \dots]$$

$$dw = 0$$
$$dw\ += x^{(1)} dz^{(1)}$$
$$dw\ += x^{(2)} dz^{(2)}$$
$$\vdots$$
$$dw\ /= m$$

$$db = 0$$
$$db\ += dz^{(1)}$$
$$db\ += dz^{(2)}$$
$$\vdots$$
$$db\ += dz^{(m)}$$
$$db\ /= m$$

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

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

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

$$= \frac{1}{m} \begin{bmatrix} x^{1} \dots x^{m} \end{bmatrix} \begin{bmatrix} dz^{1} \\ \vdots \\ dz^{m} \end{bmatrix}$$

$$= \frac{1}{m} [x^{1} dz^{1} + \dots x^{m} dz^{m}]_{n}$$

Vectorized Logistic reg. ***

$$Z = w^{T} X + b$$
$$= np.dot(w.T, X) + b$$
$$A = \sigma(Z)$$
$$dz = A - Y$$
$$dw = \frac{1}{m} X\, dz^{T}$$
$$db = \frac{1}{m}\ np.sum(dz)$$

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

⇒ Broadcasting

broadcasting

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

(m, 1) ⇝ (m, n)

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

⇒ Python - Numpy Vectors

- a = np.random.randn(5)
  a.shape = (5,)
  "rank 1 array"

  } don't use

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

- assert(a.shape == (5, 1))
  a = a.reshape((5, 1))

* logistic regression cost function

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

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

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

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

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

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

$= \hat{y}$

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

$= (1-\hat{y})$

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

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

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

=> cost of m examples

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

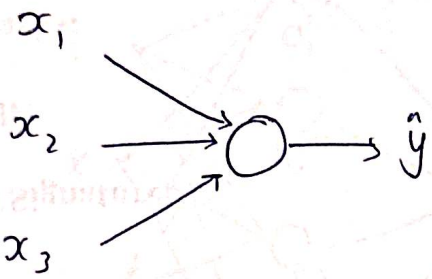
$\log p(\dots) = \sum_{i=1}^{m} \log p(\hat{y}^{(i)}|x^{(i)})$

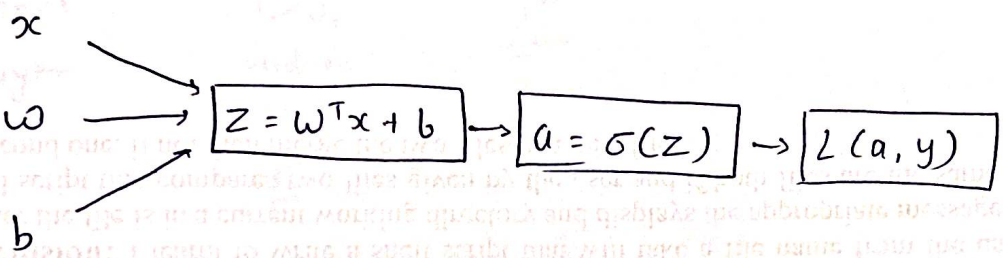
$= -\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$

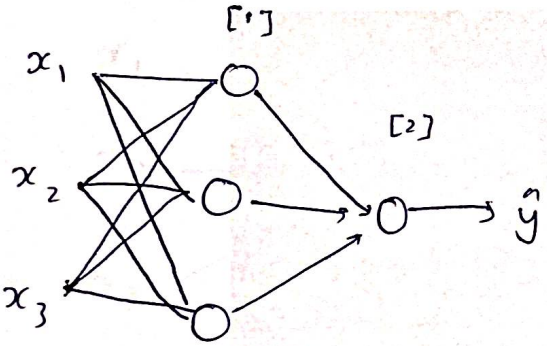
cost : $J(w,b) = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$

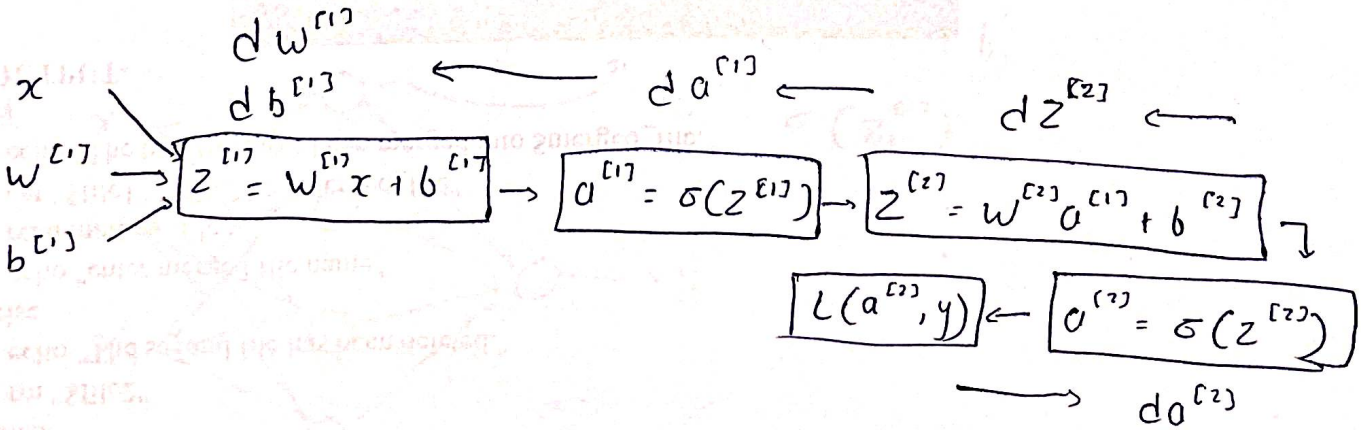
(minimize)

* What is Neural Network

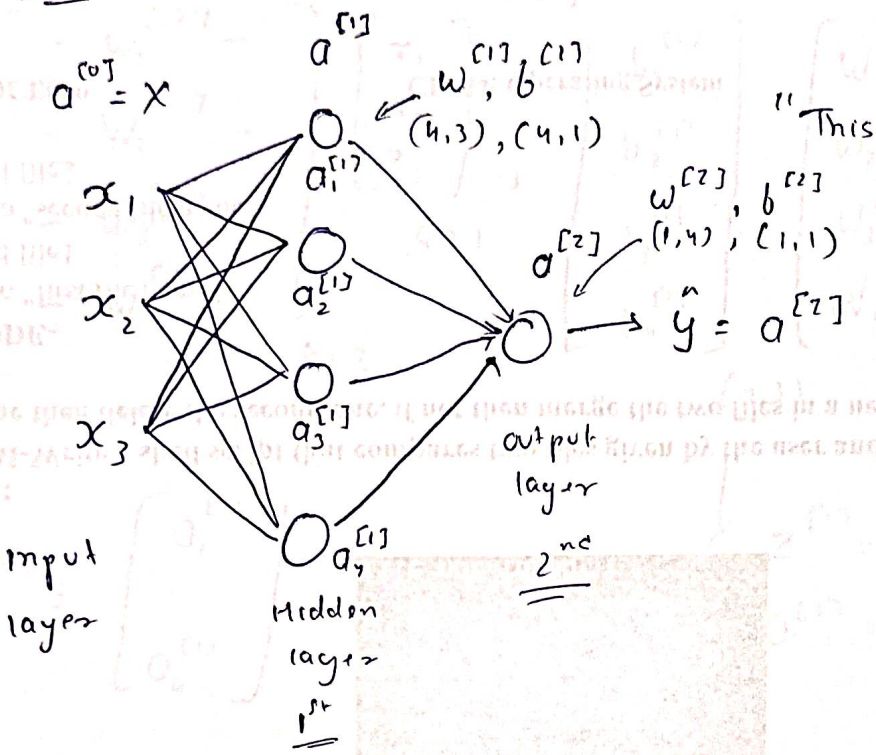




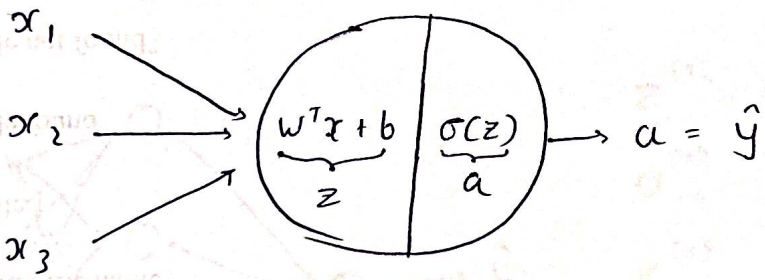


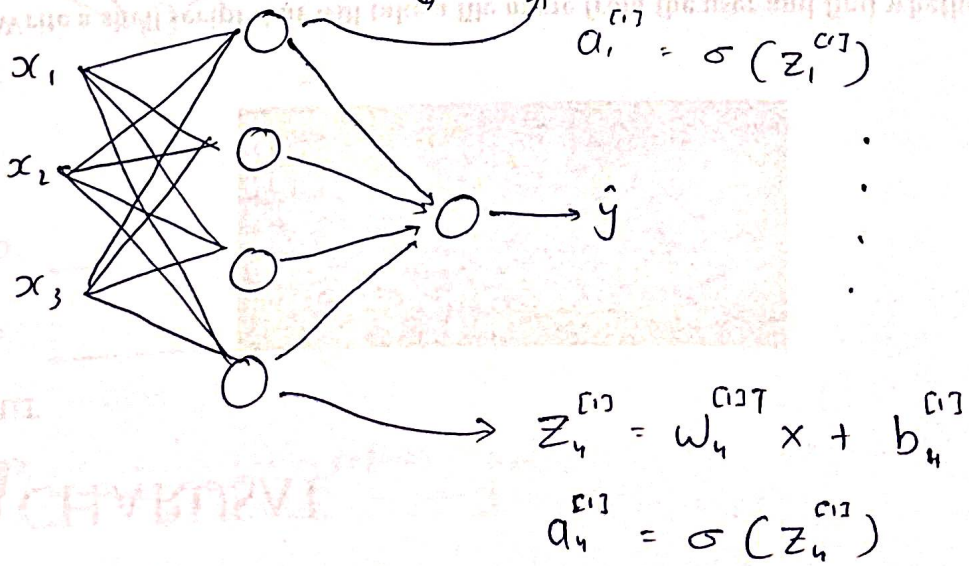


=> Neural Network representation



→



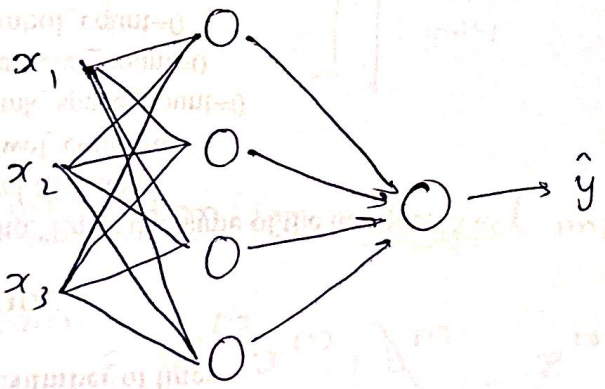


$$z^{[1]} = \underbrace{\begin{bmatrix} - w_1^{[1]T} - \\ - w_2^{[1]T} - \\ - w_3^{[1]T} - \\ - w_4^{[1]T} - \end{bmatrix}}_{4\times 3} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{3\times 1} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}}_{(4,1)} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix}_{4\times 1} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}_{4\times 1}$$

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

$$\left.\begin{aligned} z^{[1]} &= w^{[1]} x + b^{[1]} \\ a^{[1]} &= \sigma(z^{[1]}) \end{aligned}\right\}$$

⇒ Vectorizing across multiple examples



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

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

$a^{[2](i)}$ → example
→ layers

- For i=1 to m:

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

$$\Rightarrow \quad Z^{[1]} = W^{[1]} X + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

-

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

training examples

Hidden units

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

⇒ Justification for vectorized implementation

$$z^{[1](1)} = W^{[1]} x^{(1)} + \cancel{b^{[1]}}_{0}, \quad z^{[1](2)} = W^{[1]} x^{(2)} + \cancel{b^{[1]}}_{0}, \quad z^{[1](3)} = W^{[1]} x^{(3)} + \cancel{b^{[1]}}_{0}$$

$$W^{[1]} = \begin{bmatrix} & & \\ & & \\ & & \\ & & \end{bmatrix} \quad W^{[1]} x^{(1)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} \quad W^{[1]} x^{(2)} = \begin{bmatrix} x \\ x \\ x \\ x \end{bmatrix} \quad W^{[1]} x^{(3)} = \begin{bmatrix} - \\ - \\ - \\ - \end{bmatrix}$$

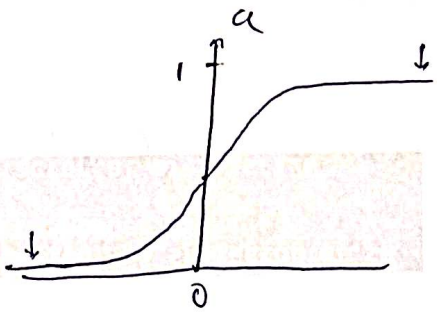
$$W^{[1]} \begin{bmatrix} | & | & | \\ x^{(1)} & x^{(2)} & x^{(3)} \cdots \\ | & | & | \end{bmatrix} = \begin{bmatrix} \cdot & x & - \\ \cdot & x & - \\ \cdot & x & - \\ \cdot & x & \cdot \end{bmatrix} = \begin{bmatrix} | & | & | \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} \\ | & | & | \end{bmatrix} = Z^{[1]}$$

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

* Activation function

i) sigmoid function (output layer)

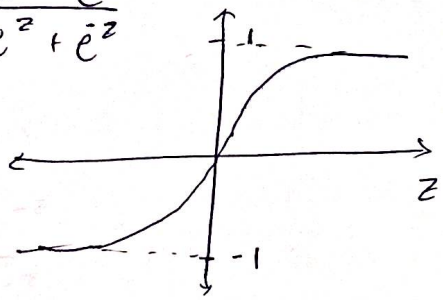
$$a = \frac{1}{1+e^{-z}}$$



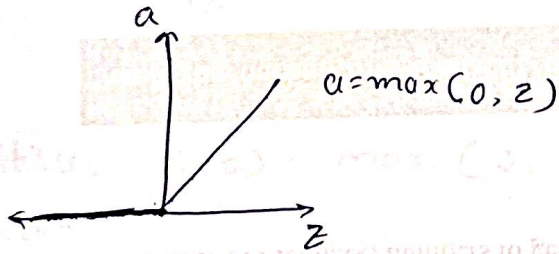
When z is very large or very small, gradient becomes very small or 0 which slowes down GD called vanishing GD.

ii) tanh function (Better than sigmoid) { hidden layer }

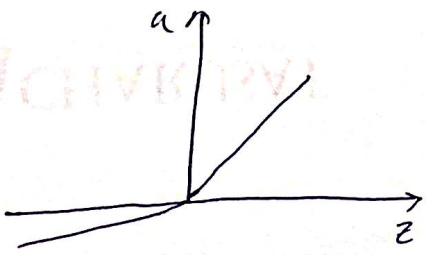
$$a = \tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$



iii) ReLU function (choose as a Default function)



iv) Leaky ReLU function



changeable value

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

⇒ Derivatives of Activation function

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

$$\frac{d\,g(z)}{dz} = \frac{d\left(\frac{1}{1+e^{-z}}\right)}{dz}$$

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

$$= a(1-a)$$

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

$$= \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$g'(z) = \frac{d\,g(z)}{dz} = 1-(\tan(z))^2$$

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

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

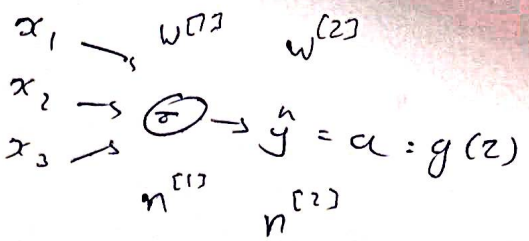
undefined if $z=0$

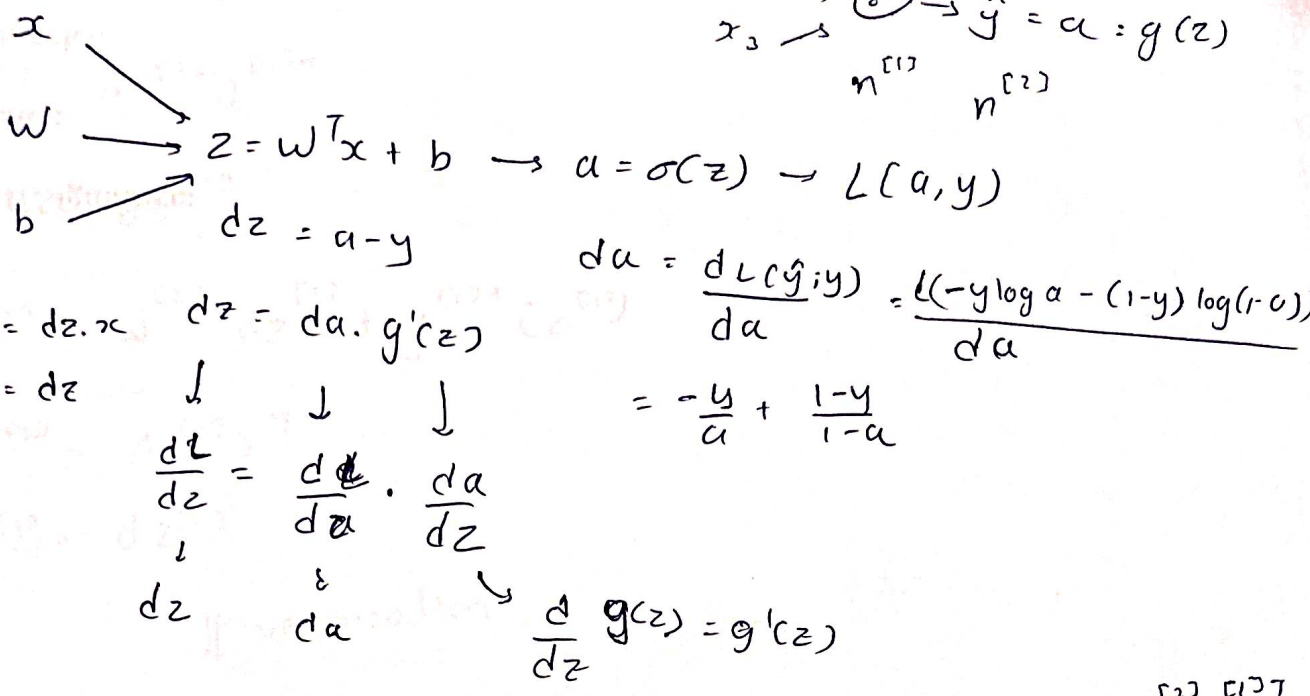
(leaky relu)

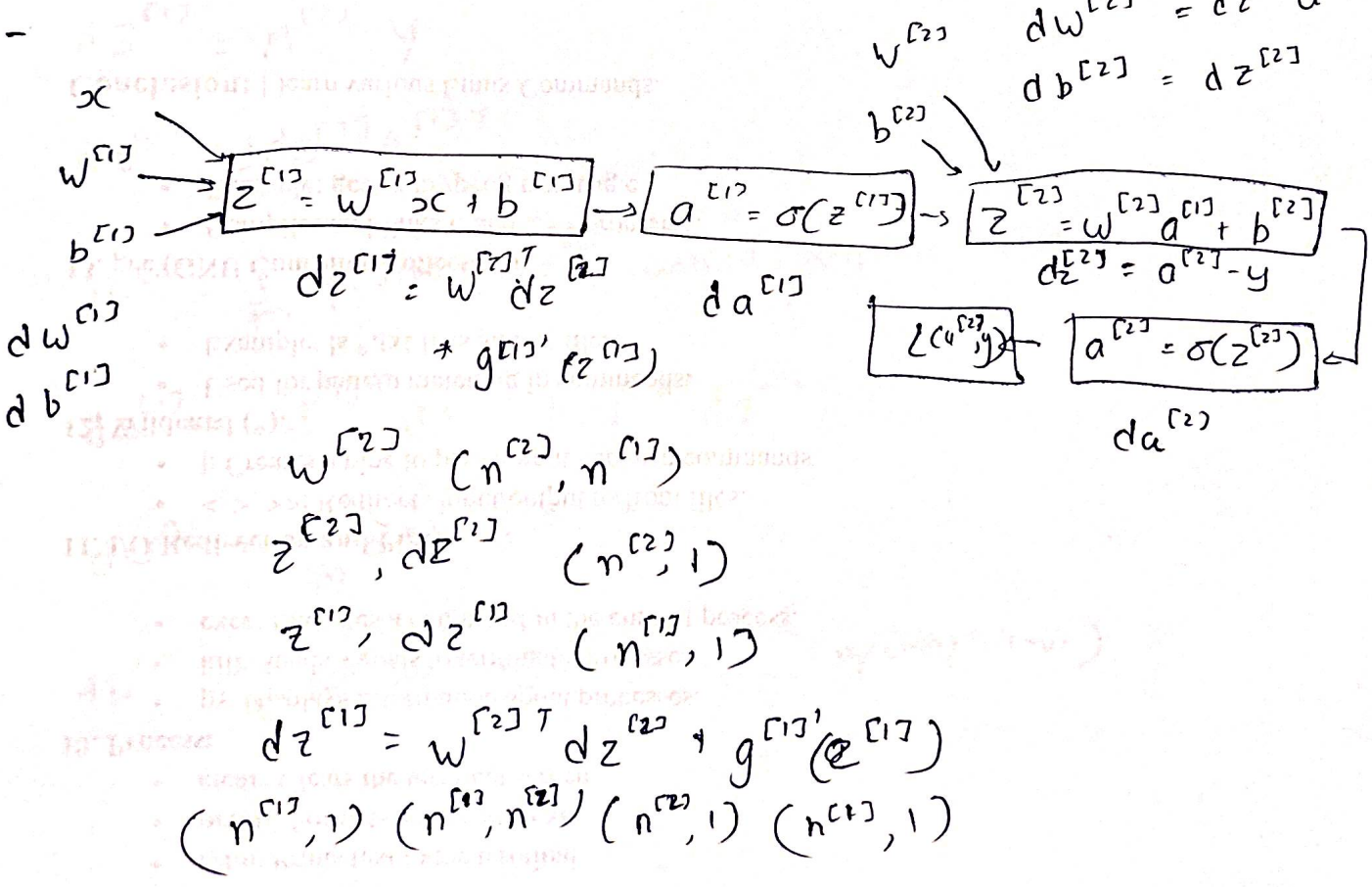
$g(z) = \max(0.01z, z)$

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

* Backpropogation intuition







$w^{[2]} \quad (n^{[2]}, n^{[1]})$

$z^{[2]}, dz^{[2]} \quad (n^{[2]}, 1)$

$z^{[1]}, dz^{[1]} \quad (n^{[1]}, 1)$

$dz^{[1]} = w^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]})$

$(n^{[1]}, 1) \quad (n^{[1]}, n^{[2]}) \quad (n^{[2]}, 1) \quad (n^{[1]}, 1)$

$\Rightarrow$

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

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

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

$$dz^{[1]} = w^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]})$$

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

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

$\Downarrow$ vectorization

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

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

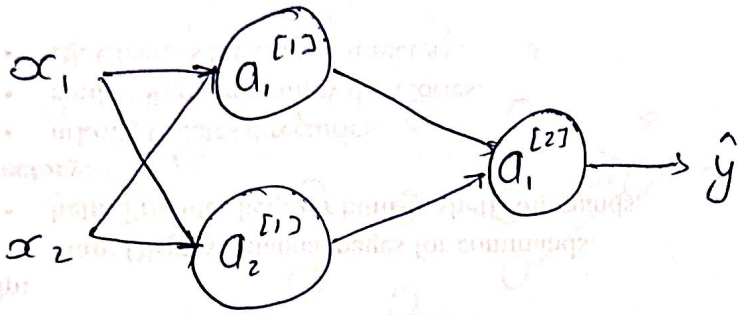
$$db^{[2]} = \frac{1}{m} np.sum(dz^{[2]}, axis=1, keepdims=True)$$

$$dz^{[1]} = w^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]})$$

$$dw^{[1]} = \frac{1}{m} dz^{[1]} x^T$$

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

⇒ Random initialization



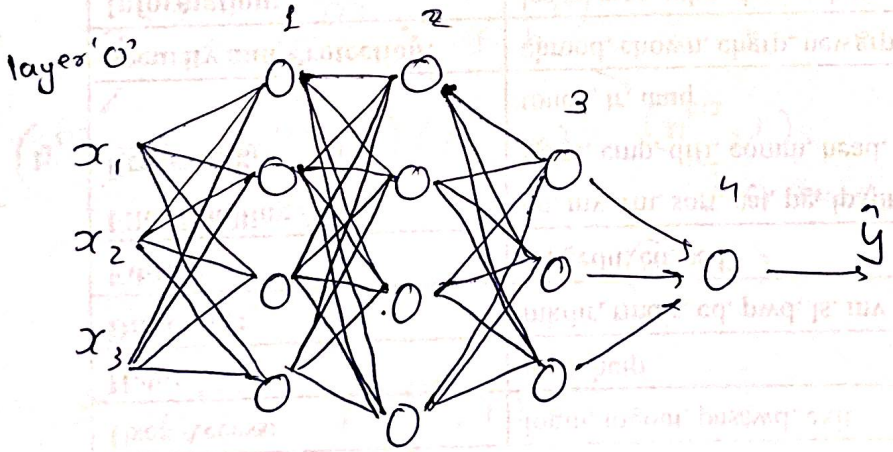
$W^{[1]} = np.random.randn((2,2)) * 0.01$ → if this is large no. we get z as large, that will saturate the activation function.

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

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

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

* Deep neural network notation



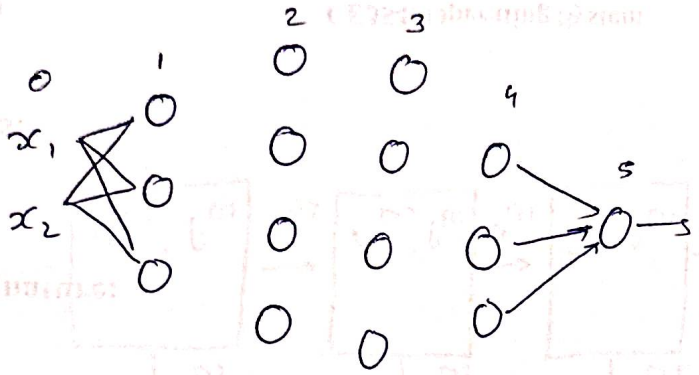
$L = 4$ # layers

$n^{[l]}$ = # units in layer l & $(n^{[0]} = 3, n^{[1]} = 4, n^{[2]} = 4, n^{[3]} = 3, n^{[4]} = n^{[L]} = 1)$

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

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

⇒ Matrix Dimentions



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

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

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

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

$w^{[2]} : (4, 3)\ (n^{[2]}, n^{[1]})$

$$w^{[l]} : (n^{[l]}, n^{[l-1]})$$
$$b^{[l]} : (n^{[l]}, 1)$$

- Vectorized implementation

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

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

} py broadcasting

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

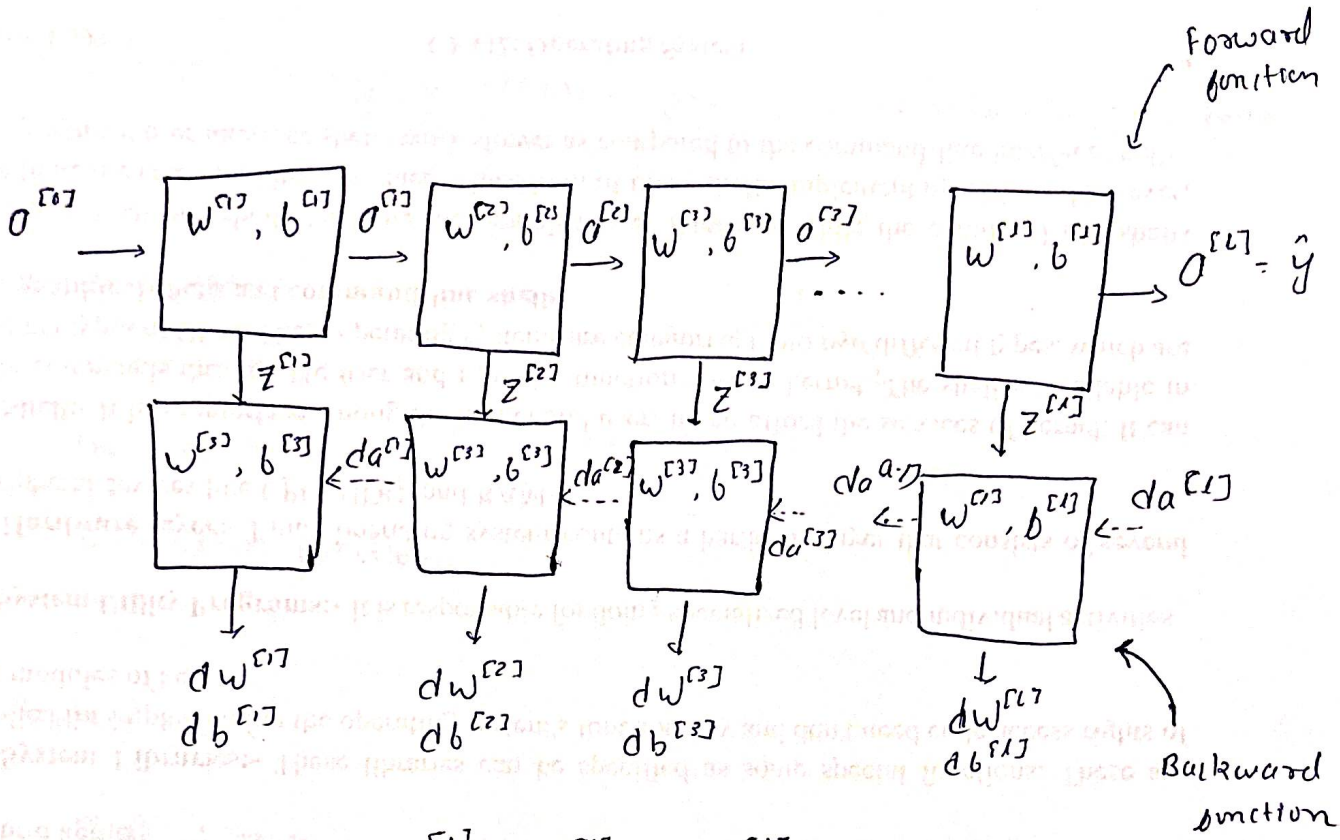
- $z^{[l]}, a^{[l]} : (n^{[l]}, 1)$

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

$l = 0 \quad A^{[0]} = X = (n^{[0]}, m)$

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

⇒ Forward and Backward functions



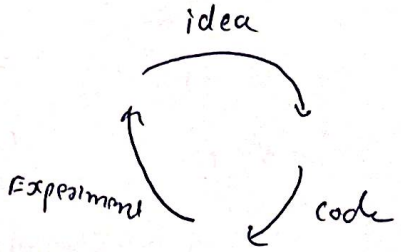
$$w^{[1]} = w^{[1]} - \alpha\, dw^{[1]}$$
$$b^{[1]} = b^{[1]} - \alpha\, db^{[1]}$$

- Forward :
  - input $a^{[l-1]}$, output $a^{[l]}$ & cache $(z^{[l]})$
  - $z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$
  - $a^{[l]} = g^{[l]}(z^{[l]})$

- backward :
  - input $da^{[l]}$, output $da^{[l-1]}$
    cache $(z^{[l]})$
    $dw^{[l]}$
    $db^{[l]}$

- Applied ML is highly iterative process, we have to fine tune hyperparameters.

  - # layers
  - # hidden units
  - learning rate
  - activation functions.
  - ...



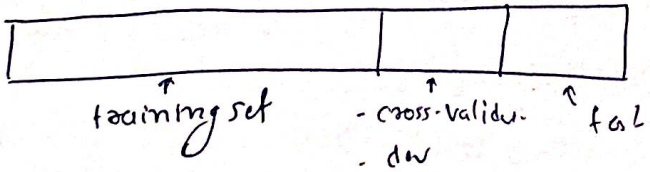
- Applications:

  NLP, Vision, speech, structural data

  Ads, search, security, logistic ...

=> Train / dev / test sets

- Data



Train test

70/30 %.

or

Train dev test

60/20/20 %.

- Big Data : 1,000,000

98/1/1 % for large dataset.

- make sure dev and test comes from same distribution.

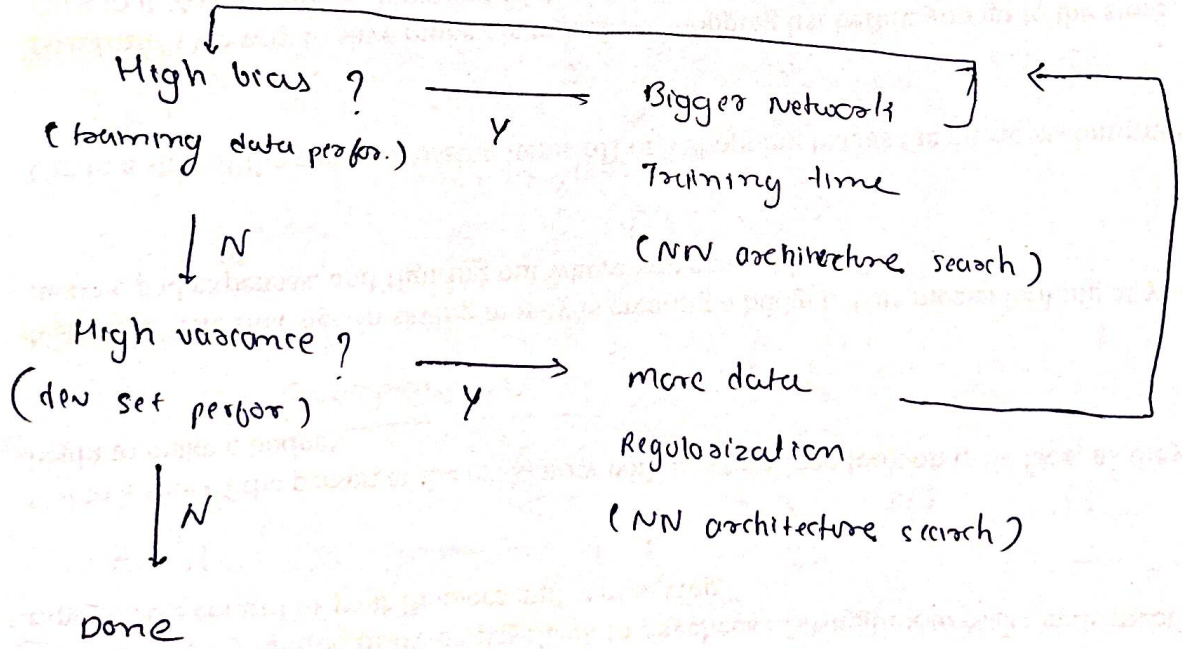
=> Bias and variance

- Train set error :	1 %	15 %.	15 %.	0.5 %
Dev set error :	11 %.	16 %.	30 %.	1 %.
	High variance (overfitting)	High bias (underfitting)	High bias High variance	low bias low variance

- High variance → complex model (To fix use regularization)
  low variance → simple model

  High bias → more errors
  low bias → low errors

-



* Regularization

- In logistic regression

$$\min_{w,b} J(w,b) \quad , \quad w \in \mathbb{R}^{n_x} \ , \ b \in \mathbb{R}$$

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\|w\|_2^2$$

(regularization parameter)

- L_2 regularization $\|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w$

- L_1 regularization $\frac{\lambda}{2m}\sum_{j=1}^{n_x} |w_j| = \frac{\lambda}{2m}\|w\|_1$

- In neural Network

$$J(w^{[1]}, b^{[1]} \dots w^{[L]}, b^{[L]}) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m}\sum_{l=1}^{L} \|w^{[l]}\|^2$$

$$\|w^{[l]}\|^2 = \sum_{i=1}^{n^{[l-1]}} \sum_{j=1}^{n^{[l]}} (w_{ij}^{[l]})^2$$

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

'Frobenius norm' $\|\cdot\|_2^2$ or $\|\cdot\|_F^2$

- $dw^{[l]} = (\text{from backprop.}) + \frac{\lambda}{m} w^{[l]}$ $\quad \frac{\partial J}{\partial w^{[l]}}$

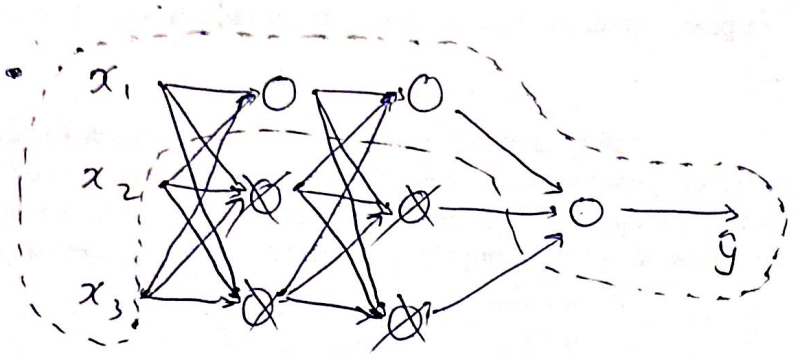
$$w^{[l]} := w^{[l]} - \alpha\, dw^{[l]}$$

- L_2 Regularization or weight decay

$$w^{[l]} := w^{[l]} - \alpha\left((\text{from backprop.}) + \frac{\lambda}{m} w^{[l]}\right)$$

$$= w^{[l]} - \frac{\alpha\lambda}{m} w^{[l]} - \alpha\,(\text{from backprop})$$

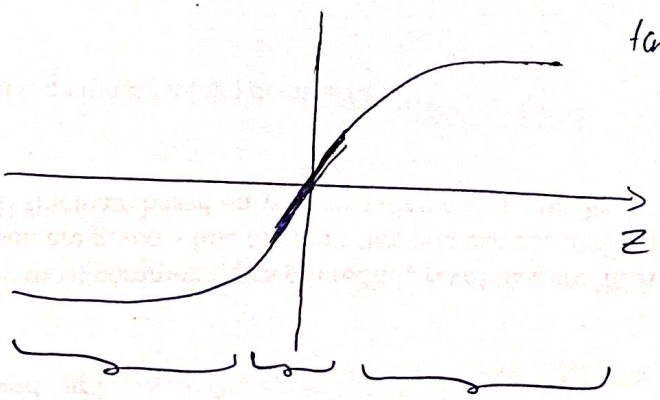
=> How does regularization prevent overfitting ?



$$J(w^{[l]}, b^{[l]}) = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) + \underbrace{\frac{\lambda}{2m}}_{\text{very high}} \sum_{l=1}^{L} \|w^{[l]}\|_F^2$$

$$w^{[l]} \approx 0$$

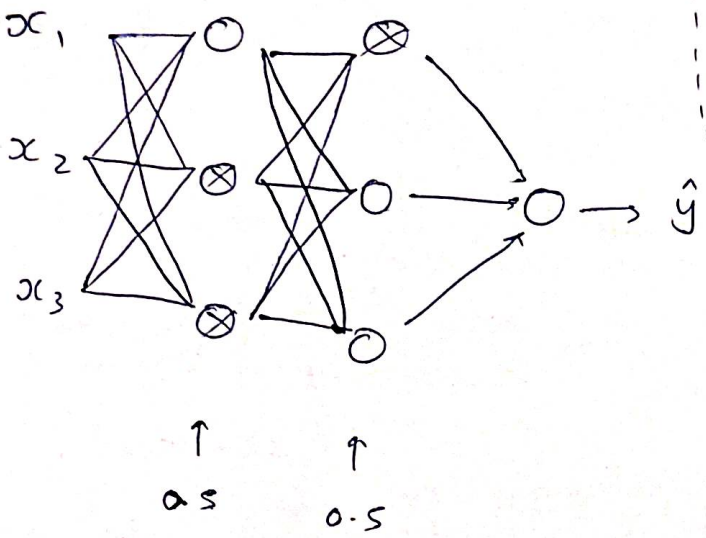
- It will cancell out some hidden units by setting it's w to zero. Thus NN becomes simple and overfitting is solved



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

$\lambda \uparrow \rightarrow w^{[l]} \downarrow \quad z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$ (mostly simple nn or linear)

⇒ Dropout Regularization



* Other ways of regularization
  - Data Augmentation (inc data)
  - early stoping

This means 50% of hidden units will be droped to making nn a more simple network. i.e 50% probability that a unit will be considered or not.

- For $L=3$ layers NN keep-prob = 0.8 (80% hidden unit will be keept and 20% will be drop)

```
d3 = np.random.rand(a3.shape[0], a3.shape[1]) < keep-prob
a3 = np.multiply(a3, d3)   # a3 *= d3
a3 /= keep-prob.
```

- 50 units → 10 unit shut off (20%)

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

↳ reduced by 20%. /= 0.8.

- At $a^{[0]}$ -> No drop out (input layer)

⇒ Normalizing inputs

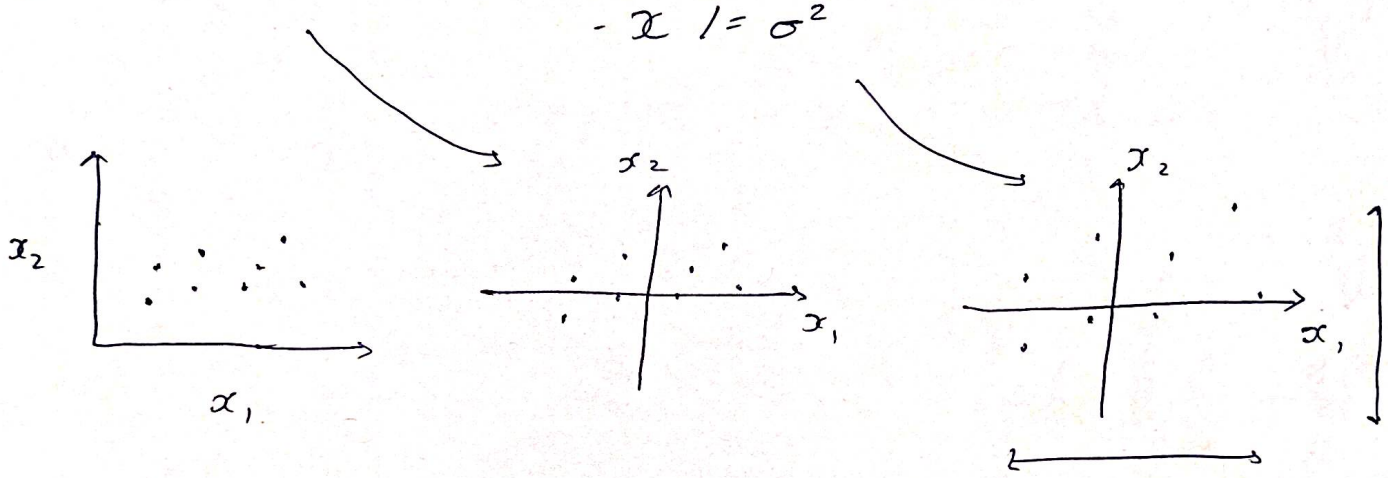
$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

1. subtract mean:

- $\mu = \frac{1}{m} \sum_{i=1}^{m} x^{(i)}$
- $x := x - \mu$

2. Normalize variance

- $\sigma^2 = \frac{1}{m} \sum_{j=1}^{m} x^{(i)} ** 2$ → element wise
- $x /= \sigma^2$



- use the same μ & σ to normalize test size.

⇒ Vanishing Gradient & Exploding gradients

- In deep NN

  if $w^{[l]} > I$ (Identity matrix) then it result exploding G

  if $w^{[l]} < I$ then it result vanishing GD.

- This can be solved by initializing the weights

$$w^{[l]} = np.random.randn(shape) * np.sqrt\left(\frac{x}{n^{[l-1]}}\right)$$

$$g^{(l)}(z) = ReLU(z)$$