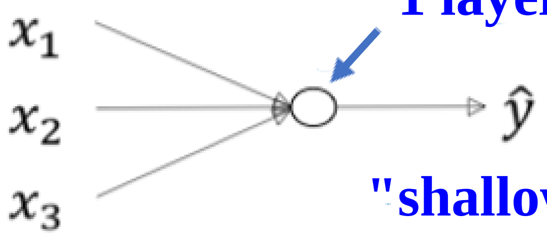
Deep Neural Network

Deep L-layer Neural Network

What is a deep neural network

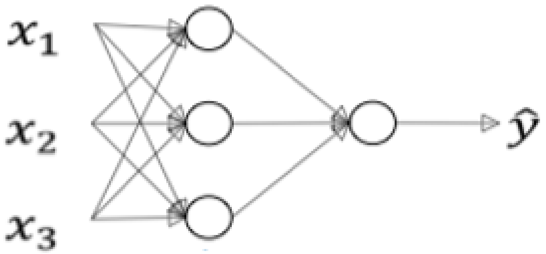
"1 layer NN"



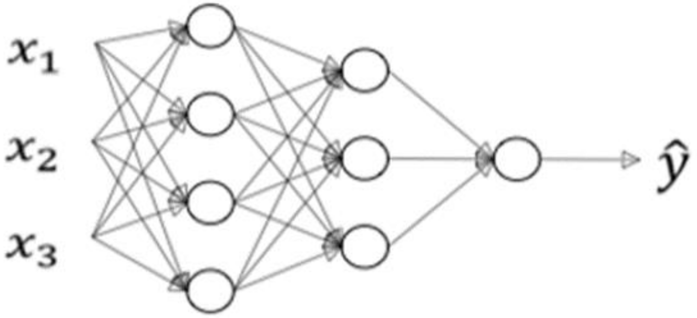
"shallow"

logistic regression

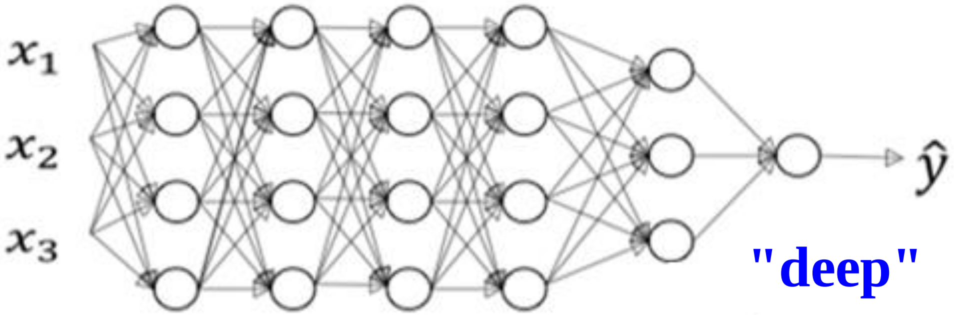
"2 layer NN"



1 hidden layer



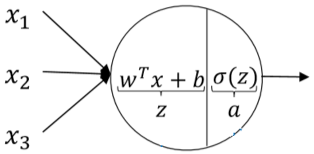
2 hidden layers



"deep"

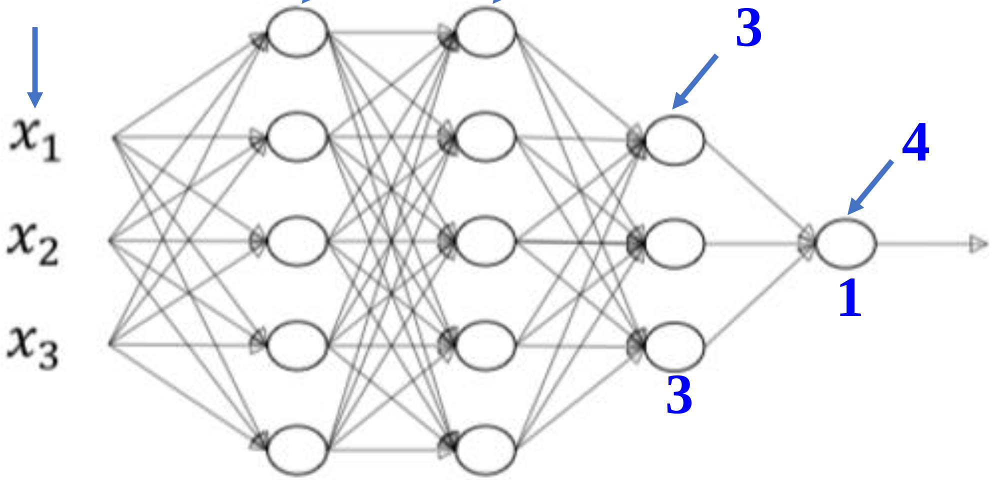
5 hidden layers

Deep neural network notation



layer "0"

4 layer NN



1 2 3 4

$x = a^{[0]}$

$\hat{y} = a^{[L]}$

5 5 3 1

L = 4 (#layer)

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

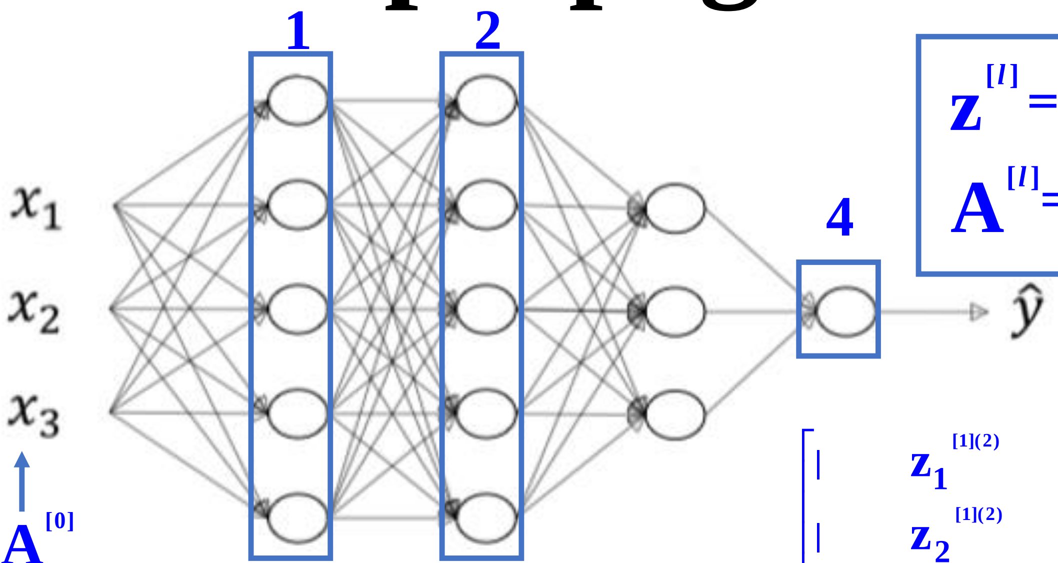
$a^{[l]}$ = activation in layer l

$a^{[l]} = g^{[l]}(z^{[l]})$, $\quad W^{[l]}$, $b^{[l]}$ = weights for $z^{[l]}$

$n^{[1]} = 5,\ n^{[2]} = 5,\ n^{[3]} = 3,\ n^{[4]} = n^{[L]} = 1$

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

Forward propagation in a deep network



$$z^{[l]} = w^{[l]T} A^{[l-1]} + b^{[l]}$$
$$A^{[l]} = g^{[l]}(z^{[l]})$$
$$A^{[0]} = x$$

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

x : $A^{[0]}$

Vectorized:

除了層外

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

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

$\cdots\cdots$

$$\hat{Y} = g(z^{[4]}) = A^{[4]}$$

for $l = 1 \ldots 4$

$$\begin{bmatrix} | & z_1^{[1](2)} & & | \\ | & z_2^{[1](2)} & & | \\ z^{[1](1)} & z_3^{[1](2)} & \cdots & z^{[1](m)} \\ | & z_4^{[1](2)} & & | \\ | & z_5^{[1](2)} & & | \end{bmatrix} = \begin{bmatrix} w_{1,1}^{[1]} & w_{1,2}^{[1]} & w_{1,3}^{[1]} & w_{1,4}^{[1]} & w_{1,5}^{[1]} \\ w_{2,1}^{[1]} & w_{2,2}^{[1]} & w_{2,3}^{[1]} & w_{2,4}^{[1]} & w_{2,5}^{[1]} \\ w_{3,1}^{[1]} & w_{3,2}^{[1]} & w_{3,3}^{[1]} & w_{3,4}^{[1]} & w_{n^{[0]},n^{[1]}}^{[1]} \end{bmatrix}^T \begin{bmatrix} A_1^{[0](1)} & A_1^{[0](2)} & \cdots & A_1^{[0](m)} \\ A_2^{[0](1)} & A_2^{[0](2)} & \cdots & A_2^{[0](m)} \\ A_3^{[0](1)} & A_3^{[0](2)} & \cdots & A_3^{[0](m)} \end{bmatrix}$$

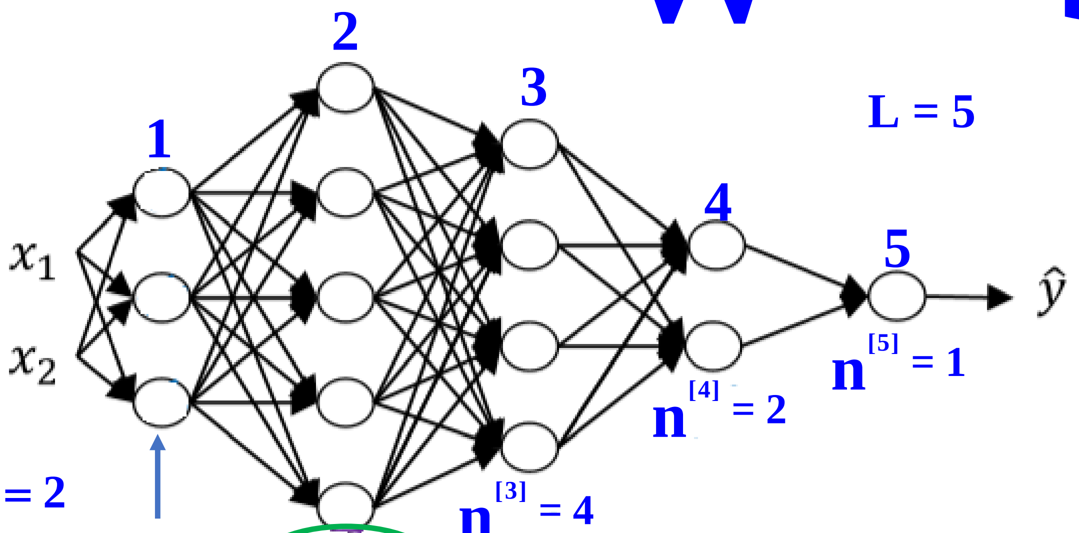
$(n^{[1]}, m)$

$$= \begin{bmatrix} w_{1,1}^{[1]} & w_{2,1}^{[1]} & w_{3,1}^{[1]} \\ w_{1,2}^{[1]} & w_{2,2}^{[1]} & w_{3,2}^{[1]} \\ w_{1,3}^{[1]} & w_{2,3}^{[1]} & w_{3,3}^{[1]} \\ w_{1,4}^{[1]} & w_{2,4}^{[1]} & w_{3,4}^{[1]} \\ w_{1,5}^{[1]} & w_{2,5}^{[1]} & w_{3,5}^{[1]} \end{bmatrix} \begin{bmatrix} A_1^{[0](1)} & A_1^{[0](2)} & \cdots & A_1^{[0](m)} \\ A_2^{[0](1)} & A_2^{[0](2)} & \cdots & A_2^{[0](m)} \\ A_3^{[0](1)} & A_3^{[0](2)} & \cdots & A_3^{[0](m)} \end{bmatrix} + b^{[1]}$$

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

broadcasting

Parameters: W^l and b^l



L = 5

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

$n^{[1]} = 3$

$n^{[2]} = 5$

$n^{[3]} = 4$

$n^{[4]} = 2$

$n^{[5]} = 1$

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

有d無d,
同矩陣大小(以此記)

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

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

$(3,1) = (3,2)\,(2,1)$

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

Assume

1 sample

$$\begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix} = \begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \\ \cdot & \cdot \end{bmatrix} \begin{bmatrix} \cdot \\ \cdot \end{bmatrix}$$

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

$z^{[2]} = w^{[2]T} a^{[1]} + b^{[2]}$, $\quad w^{[2]T} : (5,3),\ (n^{[2]}, n^{[1]})$

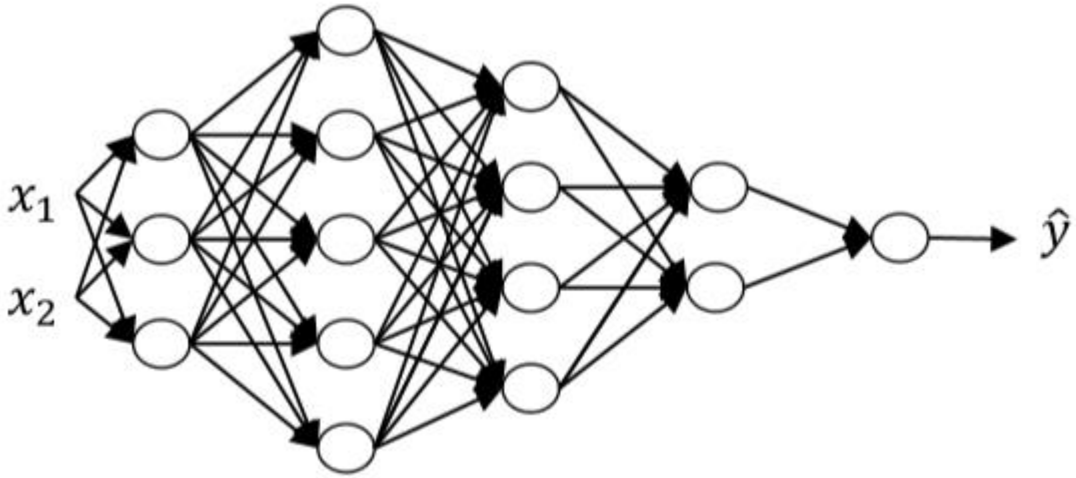
(5,1) (5,3) (3,1) (5,1)

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

$w^{[3]T} : (4,5)$

$w^{[4]T} : (2,4)$, $\quad w^{[5]T} : (1,2)$

Vectorized implementation



$$\underset{(n^{[1]},1)}{z^{[1]}} = \underset{(n^{[1]},n^{[0]})}{w^{[1]}}\underset{(n^{[0]},1)}{x} + \underset{(n^{[1]},1)}{b^{[1]}}$$

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

$$\underset{(n^{[1]},m)}{z^{[1]}} = \underset{(n^{[1]},n^{[0]})}{w^{[1]}}\underset{(n^{[0]},m)}{x} + b^{[1]} \quad \underset{(n^{[1]},m)}{\underbrace{(n^{[1]},1)}}$$

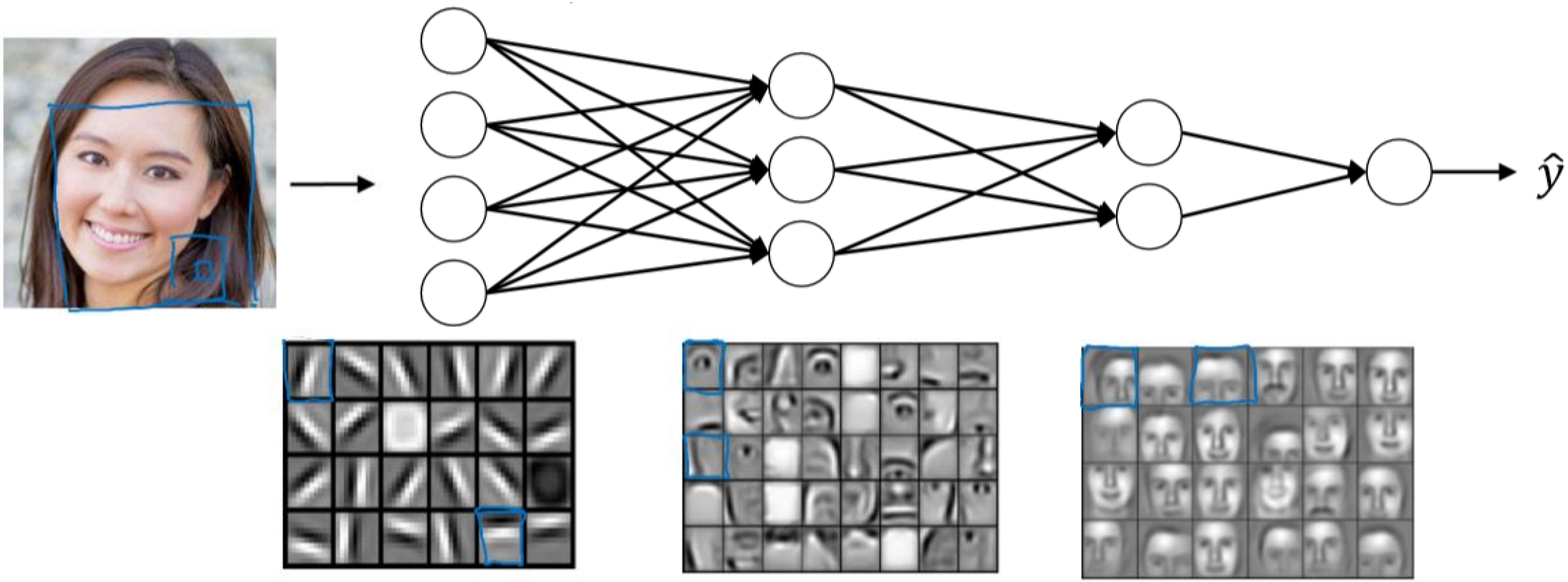
$$z^{[l]}, a^{[l]} : (n^{[l]},1)$$

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

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

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

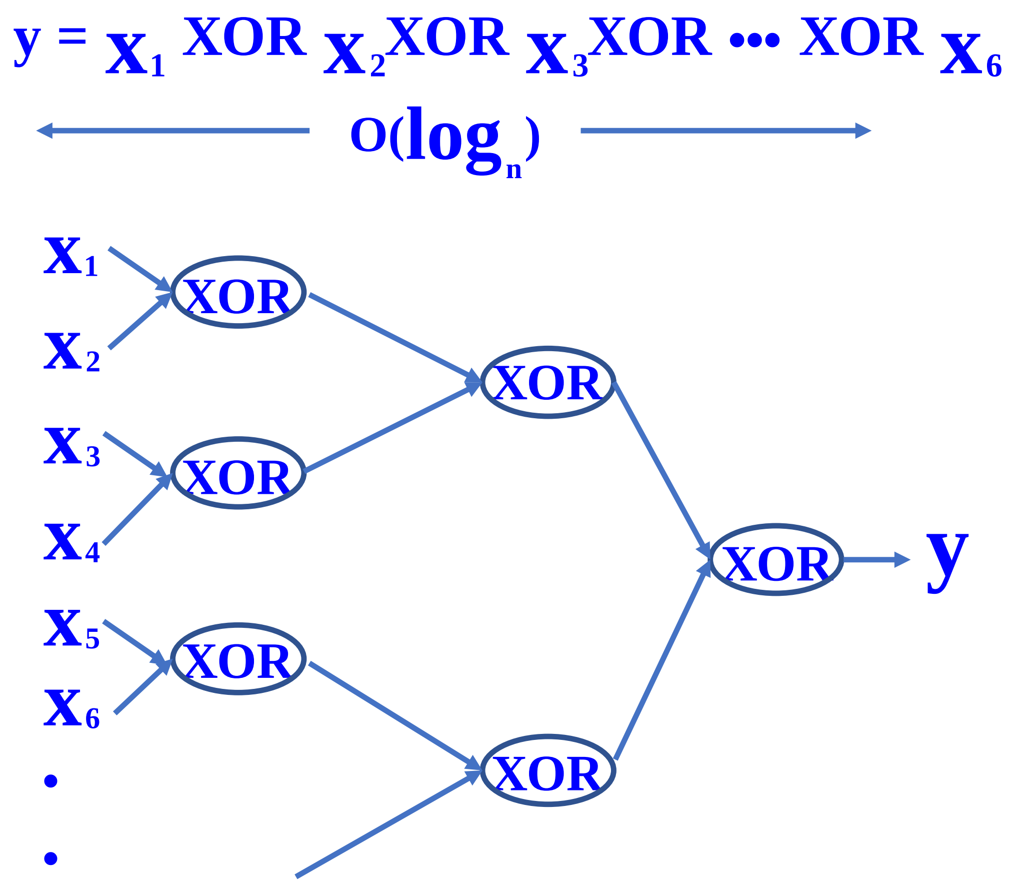
Intuition about deep representation

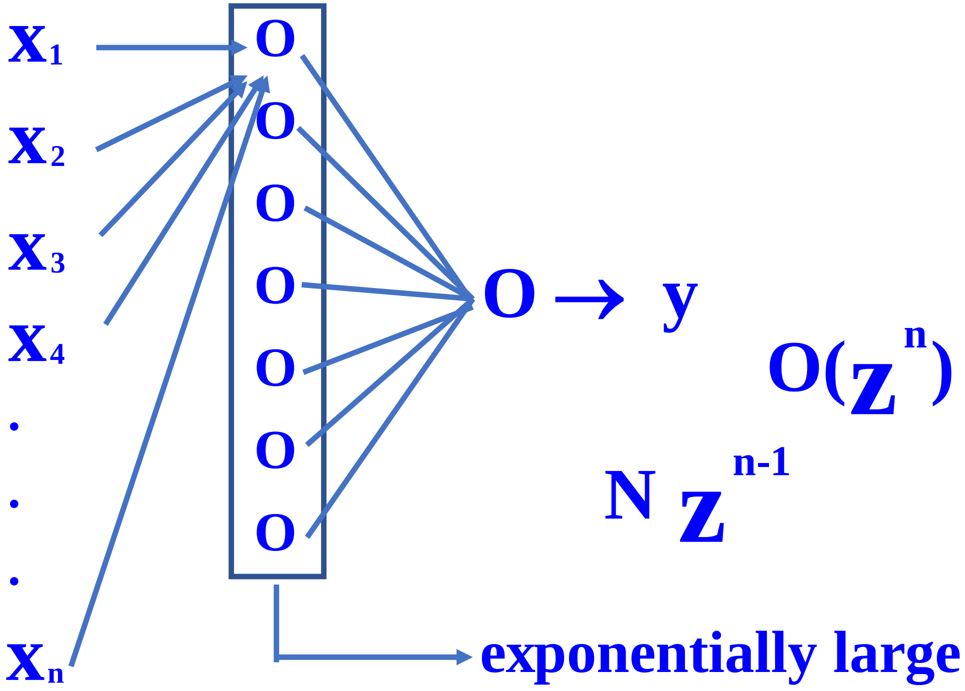


Audio → **low level audio waveform** → **Phonemes C A T** → **Words** → **Sentence phrase**

Circuit theory and deep learing

Informally: There are functions you can compute with a "*small*" L-layer deep neural network that shallower networks require exponentially more hidden units to compute.

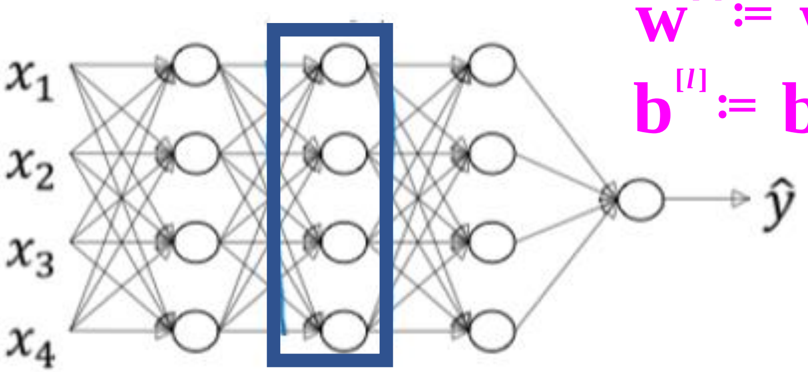




Deep Neural Network

Building blocks of deep neural networks

Forward and backward functions



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

$b^{[l]} := b^{[l]} - \alpha db^{[l]}$

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

Forward: Input $a^{[l-1]}$, Output $a^{[l]}$

$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$ cache $z^{[l]}$...

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

Backward: Input $da^{[l]}$, Output $da^{[l-1]}$

cache($z^{[l]}$)... $dw^{[l]}$

$db^{[l]}$

$dz^{[2]} = dA^{[2]} * g^{[2]'}(z^{(2)})$

$dw^{[2]} = \frac{1}{m} dz^{[2]} x^{T} = \frac{1}{m} dz^{[2]} A^{[1]^T}$

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

$dA^{[1]} = w^{[2]} dz^{[2]}$

layer l

$a^{[l-1]} \rightarrow$ $\boxed{w^{[l]}, b^{[l]}}$ $\rightarrow a^{[l]}$

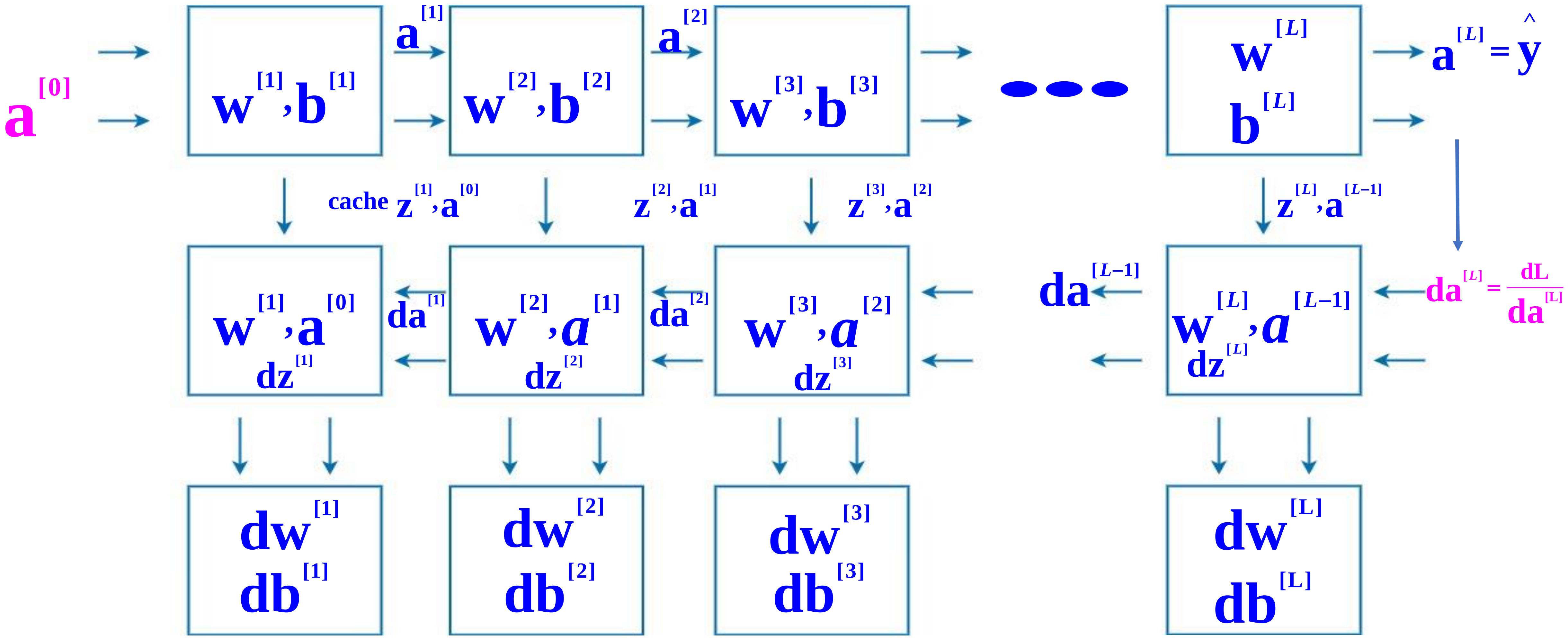
cache $z^{[l]}$...

$da^{[l-1]} \leftarrow$ $\boxed{w^{[l]}, a^{[l-1]} \quad dz^{[l]}}$ $\leftarrow da^{[l]}$

$dw^{[l]}$

$db^{[l]}$

Forward and backward functions



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

Deep Neural Network

Forward and backward propagation

Backward propagation for layer *l*

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

Output $da^{[l-1]}, dw^{[l]}, db^{[l]}$

m samples

1 sample

$$dz^{[l]} = da^{[l]} * g^{[l]'}(z^{[l]})$$

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

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

$$da^{[l-1]} = w^{[l]} dz^{[l]}$$

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

$$dz^{[l]} = dA^{[l]} * g^{[l]'}(z^{[l]})$$

$$dw^{[l]} = \frac{1}{m} A^{[l-1]} dz^{[l]T}$$

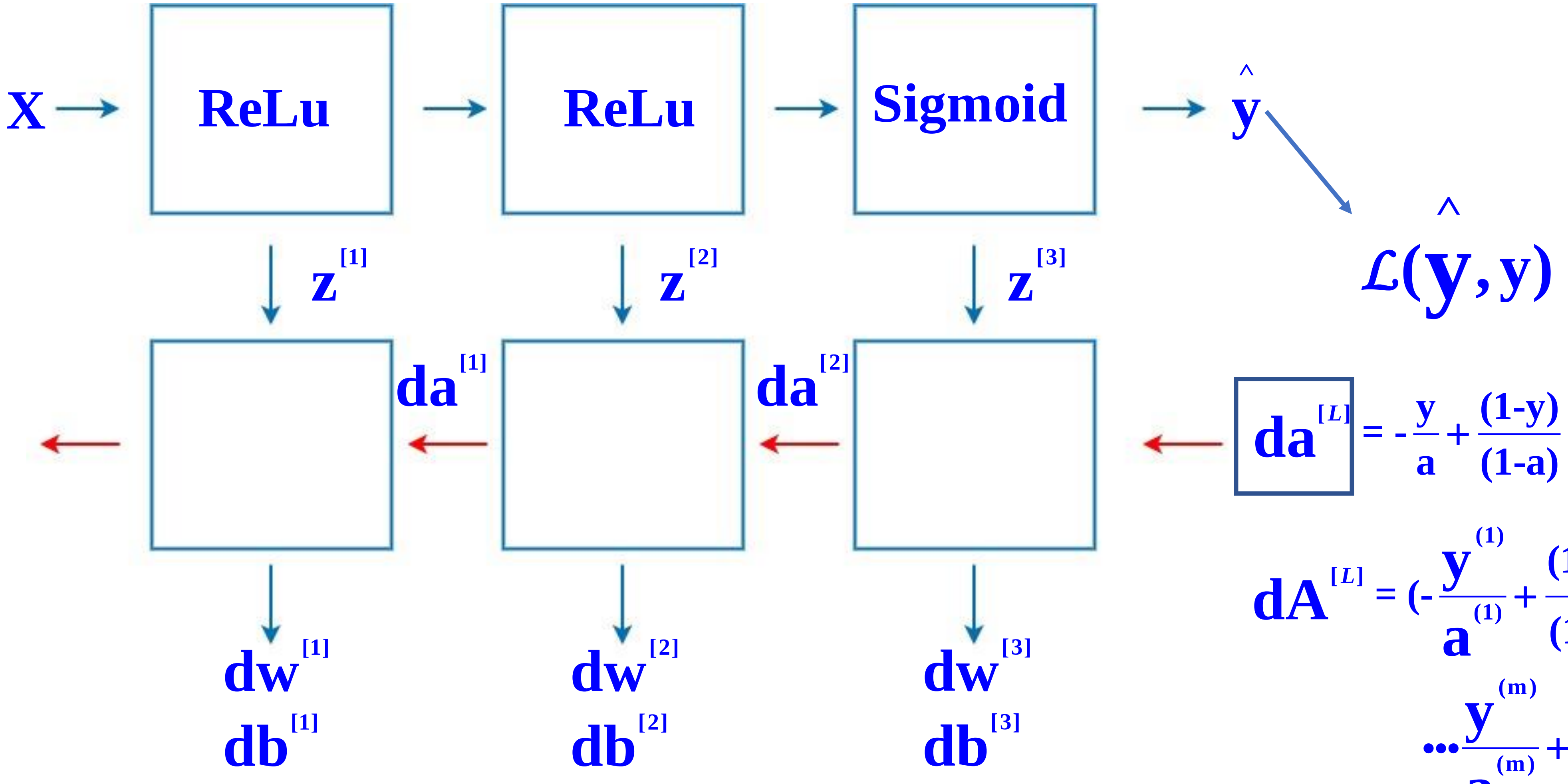
$(n^{[l-1]}, n^{[l]})$ $(n^{[l-1]}, m)$ $(m, n^{[l]})$

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

$$dA^{[l-1]} = w^{[l]} dz^{[l]}$$

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

Summary



$$\boxed{da^{[L]}} = -\frac{y}{a} + \frac{(1-y)}{(1-a)}$$

$$dA^{[L]} = \left(-\frac{y^{(1)}}{a^{(1)}} + \frac{(1-y^{(1)})}{(1-a^{(1)})} \cdots \cdots \frac{y^{(m)}}{a^{(m)}} + \frac{(1-y^{(m)})}{(1-a^{(m)})}\right)$$

Deep Neural Network

Parameters vs Hyperparameters

What are hyperparamerters?

Parameters : $\mathbf{W}^{[1]}, \mathbf{b}^{[1]}, \mathbf{W}^{[2]}, \mathbf{b}^{[2]}, \mathbf{W}^{[3]}, \mathbf{b}^{[3]}$

Hyperparameters : learning rate α

#iterations

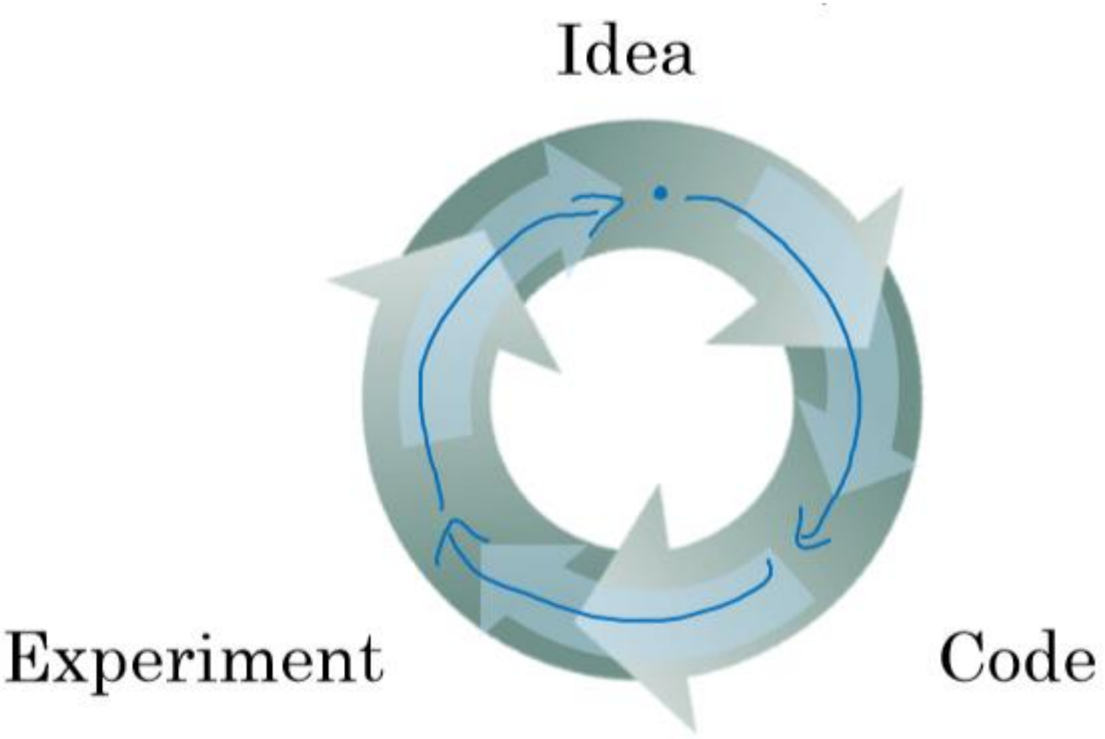
#hidden layers L

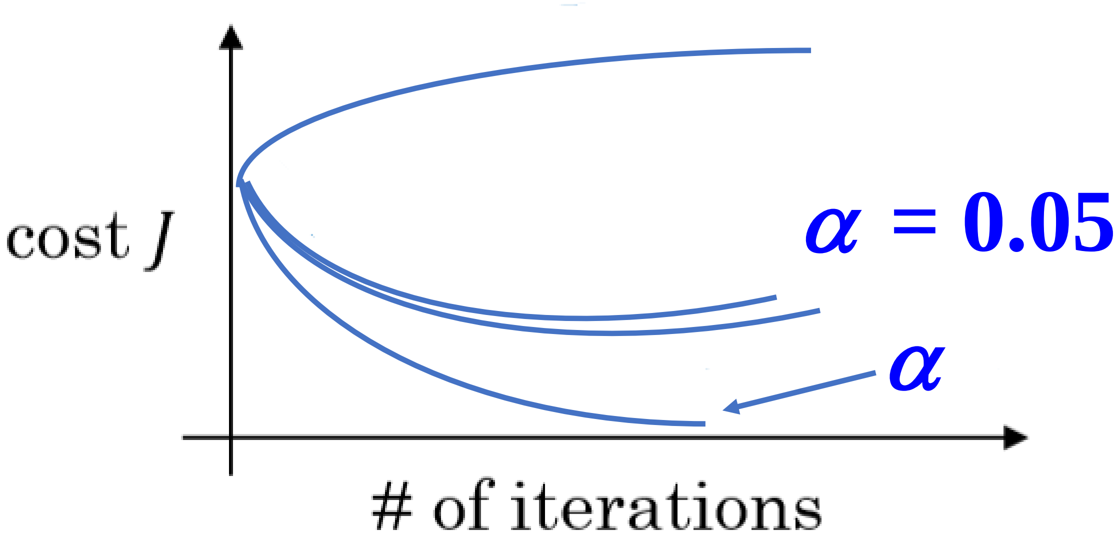
#hidden units $\mathbf{n}^{[1]}, \mathbf{n}^{[2]}, \cdots$

choice of activation function

Later: momentum,minibatch size,regularzation,•••

Applied deep learning is a very empirical process





Vision, Speech,NLP,Ad,Search,Recommendations

Deep Neural Network

What does this have to do with the brain?

Forward and backward propagation

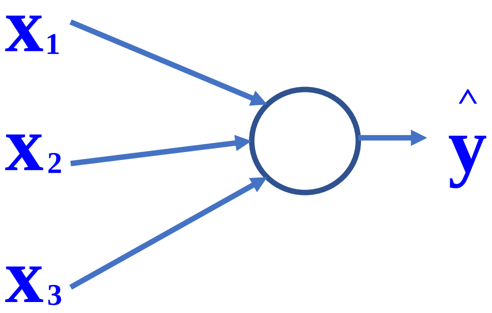
$$
\begin{aligned}
Z^{[1]} &= W^{[1]}X + b^{[1]} \\
A^{[1]} &= g^{[1]}(Z^{[1]}) \\
Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\
A^{[2]} &= g^{[2]}(Z^{[2]}) \\
&\vdots \\
A^{[L]} &= g^{[L]}(Z^{[L]}) = \hat{Y}
\end{aligned}
$$

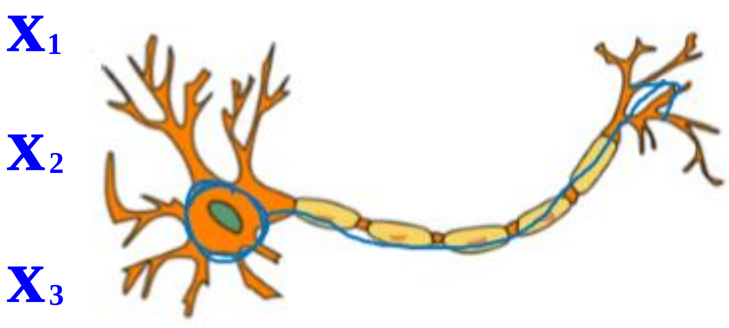
$$
\begin{aligned}
dZ^{[L]} &= A^{[L]} - Y \\
dW^{[L]} &= \frac{1}{m} dZ^{[L]} A^{[L]^T} \\
db^{[L]} &= \frac{1}{m} np.\mathrm{sum}(\mathrm{d}Z^{[L]}, axis = 1, keepdims = True) \\
dZ^{[L-1]} &= dW^{[L]^T} dZ^{[L]} g'^{[L]}(Z^{[L-1]}) \\
&\vdots \\
dZ^{[1]} &= dW^{[L]^T} dZ^{[2]} g'^{[1]}(Z^{[1]}) \\
dW^{[1]} &= \frac{1}{m} dZ^{[1]} A^{[1]^T} \\
db^{[1]} &= \frac{1}{m} np.\mathrm{sum}(\mathrm{d}Z^{[1]}, axis = 1, keepdims = True)
\end{aligned}
$$

$$\mathbf{dw}^{[l]} = \frac{1}{m} \mathbf{A}^{[l-1]} \mathbf{dz}^{[l]T}$$

看W怎樣定義

"It's like the brain"





x → y

3 – layer neural network programming

1. **Change alpha**
2. **Change initial condition to others, ex. All zero, can you make it faster**
3. **Reduce layer numbers to two, one layer and redraw the distribution plot**
4. **Change A0 by:**

```
import sklearn; import sklearn.datasets
# load image dataset: blue/red dots in circles
train_X, train_Y = sklearn.datasets.make_circles(n_samples=35, noise=.05)
A0=train_X.T; Y=train_Y.reshape(1,35)
print("np.shape(Y):",np.shape(Y),"np.shape(A0):",np.shape(A0))
for i in range(35):
    if(Y[0][i]<0.5):
        plt.scatter(A0[0][i], A0[1][i],c="blue",marker="o")
    else:
        plt.scatter(A0[0][i], A0[1][i],c="purple",marker="x")
plt.title('x1, x2, y')
plt.show()
```

3 – layer neural network programming

a standard Normal distribution (mean=0, stdev=1).

```
import matplotlib
import time
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
n0=2;n1=3;n2=4;n3=1;m=35
alpha=0.01
```

```
# Prepare input A0 and labelling output Y
x1=[0.4,1.2,1.1,1.9,1.8,1.5,2.6,2.2,3.3,3.0,4.1,4.2,5.6,0.7,0.7
x2=[4.1,1.5,4.8,0.7,2.2,3.8,1.8,4.8,1.0,4.3,1.8,3.1,6.0,5.9,7.0
y0=[0,0,0,0,0,0,0,0,0,0,0,0,0];y1=[1,1,1,1,1,1,1,1,1,1,1,1,1,1,
A0=np.array([x1,x2])
Y= np.concatenate((y0,y1),axis=None).reshape(1,35)
print("np.shape(Y):",np.shape(Y),"np.shape(A0):",np.shape(A0),
      "A0[0][0]:",A0[0][0],"A0[0][1]:",A0[0][1],"Y",Y[0][0:3])
plt.scatter(A0[0][0:13], A0[1][0:13],marker="o")
plt.scatter(A0[0][13:35], A0[1][13:35],marker="x")
plt.title('x1, x2, y')
plt.show()
```

```
# Initialization
np.random.seed(1)
WT1=np.random.randn(n1,n0)*0.01;b1=np.random.randn(n1,1)
WT2=np.random.randn(n2,n1)*0.01;b2=np.random.randn(n2,1)
WT3=np.random.randn(n3,n2)*0.01;b3=np.random.randn(n3,1)
```

3 – layer neural network programming

```
# Initialization
np.random.seed(1)
WT1=np.random.randn(n1,n0)*0.01;b1=np.random.randn(n1,1)
WT2=np.random.randn(n2,n1)*0.01;b2=np.random.randn(n2,1)
WT3=np.random.randn(n3,n2)*0.01;b3=np.random.randn(n3,1)
```

```
def forwardfunc():
    global A0,A1,A2,A3,WT1,WT2,WT3,b1,b2,b3
    Z1=np.dot(WT1,A0)+b1;
    A1=(np.exp(Z1)-np.exp(-Z1))/(np.exp(Z1)+np.exp(-Z1))
    Z2=np.dot(WT2,A1)+b2;
    A2=(np.exp(Z2)-np.exp(-Z2))/(np.exp(Z2)+np.exp(-Z2))
    Z3=np.dot(WT3,A2)+b3;
    A3=1/(1+np.exp(-Z3))
```

3 – layer neural network programming

```
def backwardprop():
    global Y,A3,A2,A1,A0,dA2,dA1,dZ3,dZ2,dZ1,
    dWT3,dWT2,dWT1,db3,db2,db1
#     dZ3=A3-Y  #dZ3 comes from dA3 and dA3/dZ3
    dA3=-Y/A3+(1-Y)/(1-A3)
    dZ3=dA3*(A3*(1-A3))
    dWT3=1/m*np.dot(dZ3,A2.T)
    db3=1/m*np.sum(dZ3,axis=1,keepdims=True)
    dA2=np.dot(WT3.T,dZ3)
    dZ2=dA2*(1-A2**2)
    dWT2=1/m*np.dot(dZ2,A1.T)
    db2=1/m*np.sum(dZ2,axis=1,keepdims=True)
    dA1=np.dot(WT2.T,dZ2)
    dZ1=dA1*(1-A1**2)
    dWT1=1/m*np.dot(dZ1,A0.T)
    db1=1/m*np.sum(dZ1,axis=1,keepdims=True)
```

3 – layer neural network programming

```
itera=0
cost = 100
while (cost > 0.05):  #0.05, 0.2
    forwardfunc()
    cost=-1/m*np.sum((Y*np.log(A3)+(1-Y)*np.log(1-A3)),axis=1,keepdims=True)
    if(itera%10000==0):
        print("itera ", itera, "cost ", cost)
    backwardprop()
    WT3=WT3-alpha*dWT3; b3=b3-alpha*db3; WT2=WT2-alpha*dWT2; b2=b2-alpha*db2
    WT1=WT1-alpha*dWT1; b1=b1-alpha*db1
    itera=itera+1
print("np.shape(Y)",np.shape(Y), "np.shape(A3)",np.shape(A3))
```

```
itera  0 cost  [[0.65971979]]
itera  10000 cost  [[0.65971121]]
itera  20000 cost  [[0.65970933]]
itera  30000 cost  [[0.65967928]]
```

3 – layer neural network programming

```
# Inference
A0 = np.zeros((n0,1600))#[xx1, xx2][xy1,xy2]
for i in range(40):
    for j in range(40):
        A0[0][i*39+j]=i*0.2
        A0[1][i*39+j]=j*0.2
forwardfunc()
print("A3 ",A3)
plt.scatter(A0[0][0:2], A0[1][0:2],c="red",marker=".")
plt.scatter(A0[0][2], A0[1][2],c="green",marker=".")
for i in range(40):
    for j in range(40):
        if(A3[0][i*39+j]>0.5):
            plt.scatter(A0[0][i*39+j],
             A0[1][i*39+j],c="red",marker=".")
        else:
            plt.scatter(A0[0][i*39+j],
             A0[1][i*39+j],c="green",marker=".")
```

3 – layer neural network programming

```python
    A0=np.array([x1,x2])
    forwardfunc()
    for i in range(m):
        if(A3[0][i]<0.5):
            plt.scatter(A0[0][i], A0[1][i],c="blue",marker="o")
        else:
            plt.scatter(A0[0][i], A0[1][i],c="purple",marker="x")
    plt.title('xx, xy, y')
    plt.show()
```