Introduction to Neural Network and Deep Learning

Đỗ Trọng Hợp

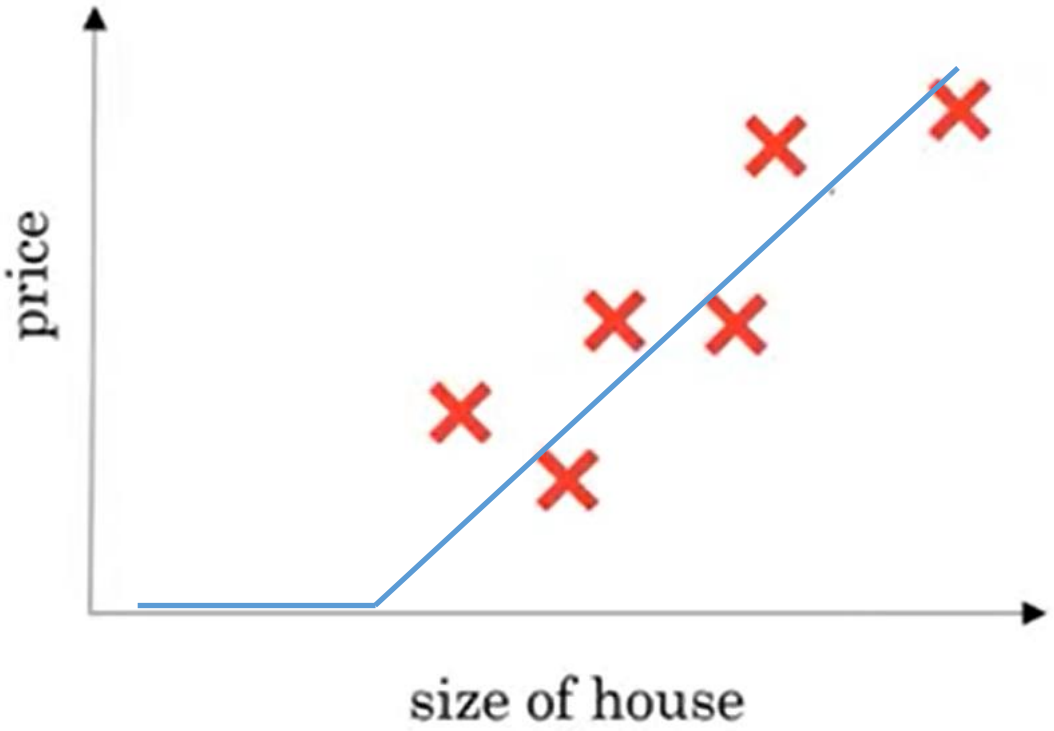
Khoa Khoa Học và Kỹ Thuật Thông Tin

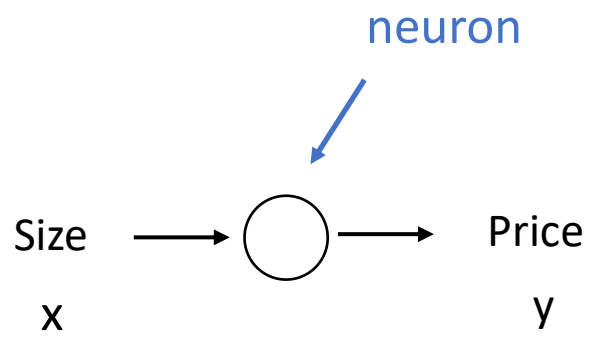
Đại học Công Nghệ Thông Tin TP. Hồ Chí Minh

Introduction to Deep Learning

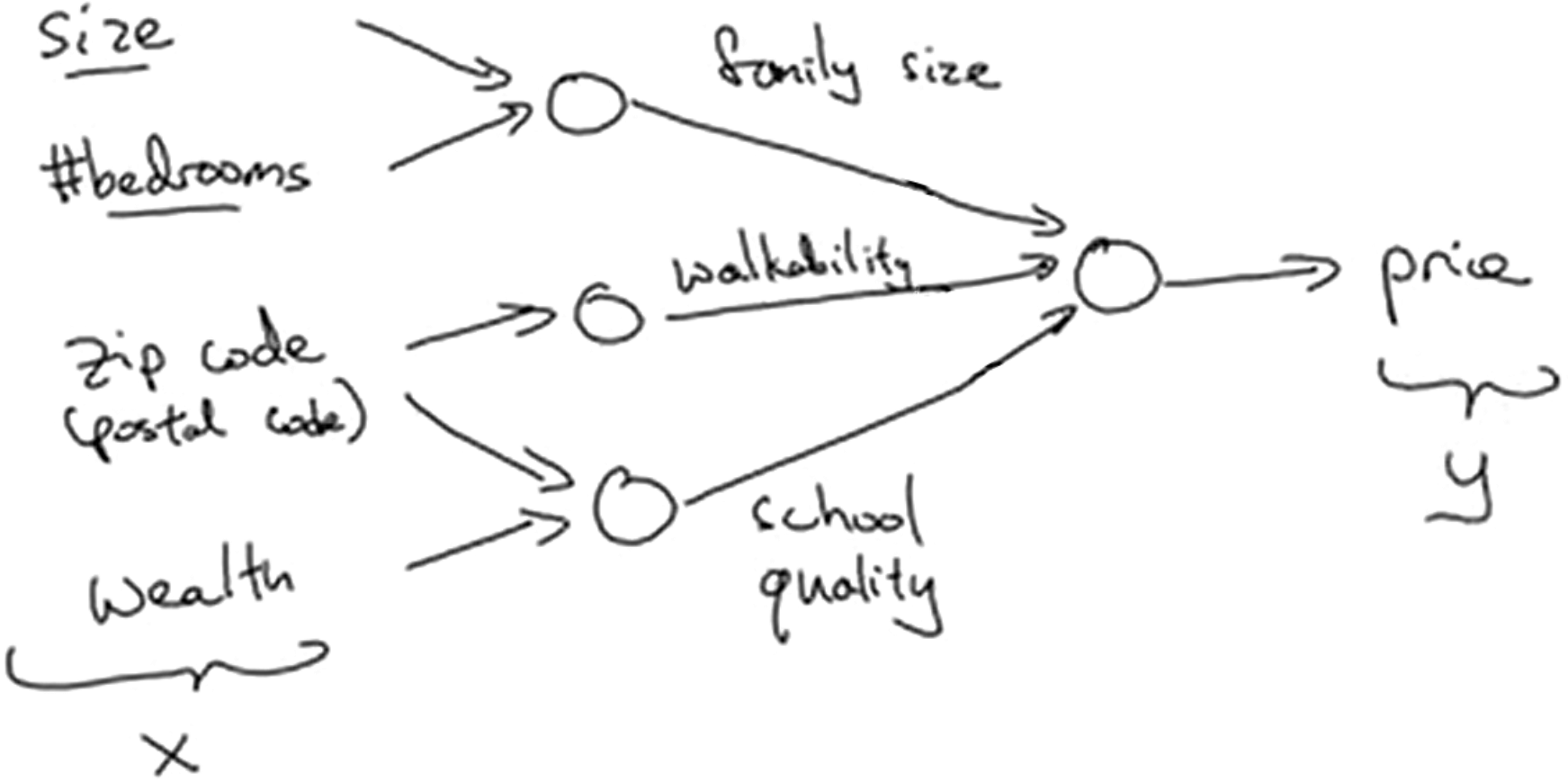
Trong-Hop Do

Housing Price Prediction

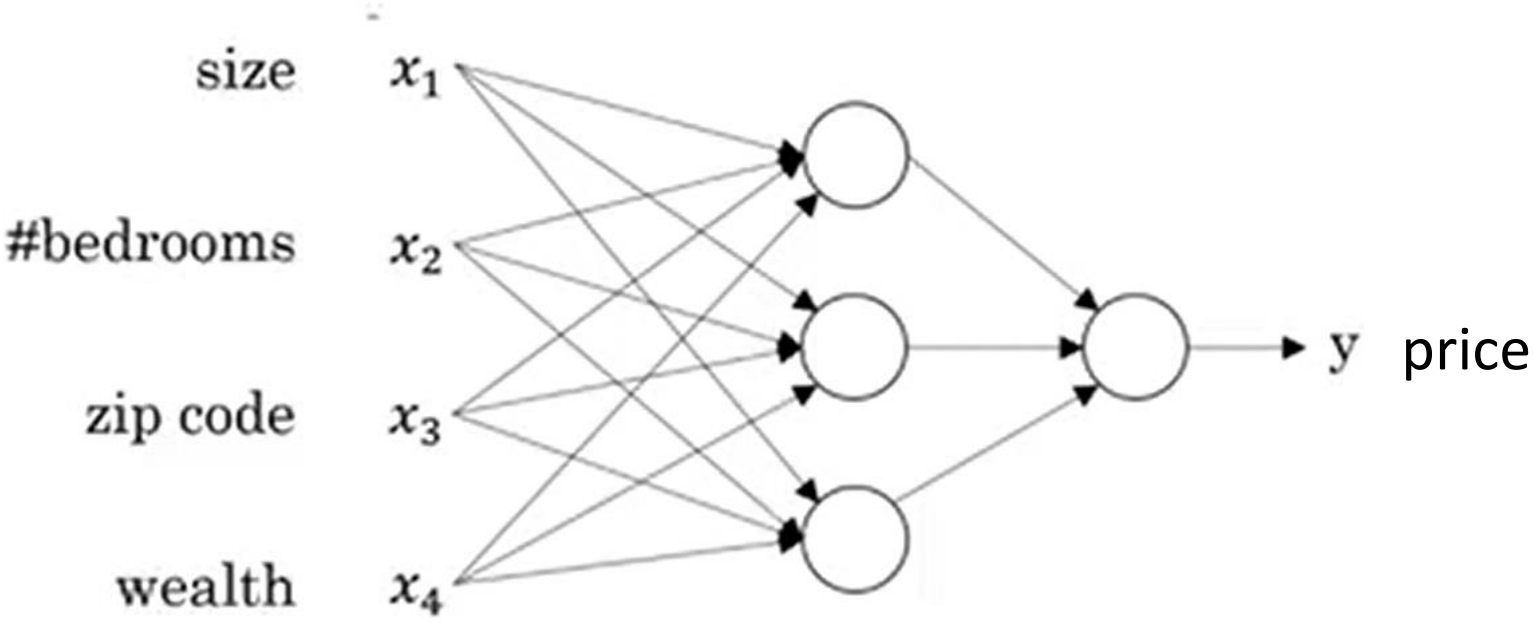




Housing Price Prediction



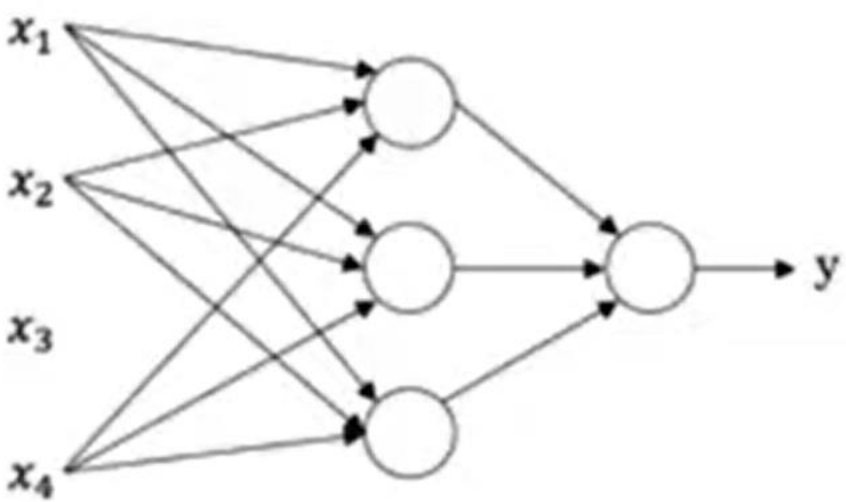
Housing Price Prediction



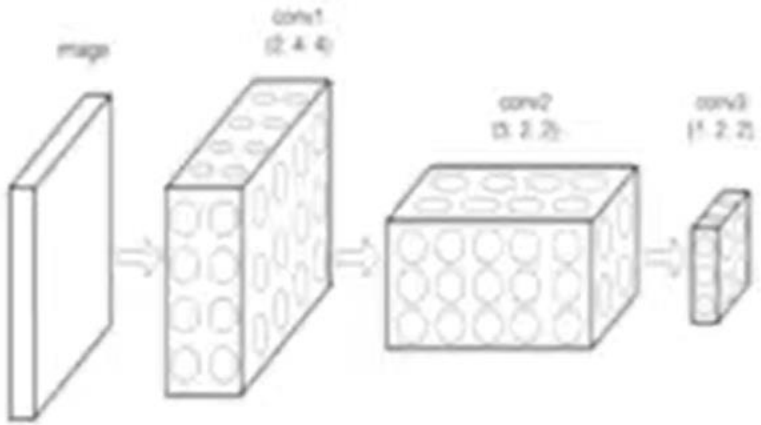
Supervised learning

Input(x)	Output(y)	Application	
Home features	Price	Real Estate	Standard NN
Ad,user info	Click on ad?(0/1)	Online Advertising	
Image	Object(1,⋯,1000)	Photo tagging	CNN
Audio	Text transcript	Speech recognition	RNN
English	Chinese	Machine translation	
Image,Radar info	Position of other cars	Autonomous driving	Hybrid

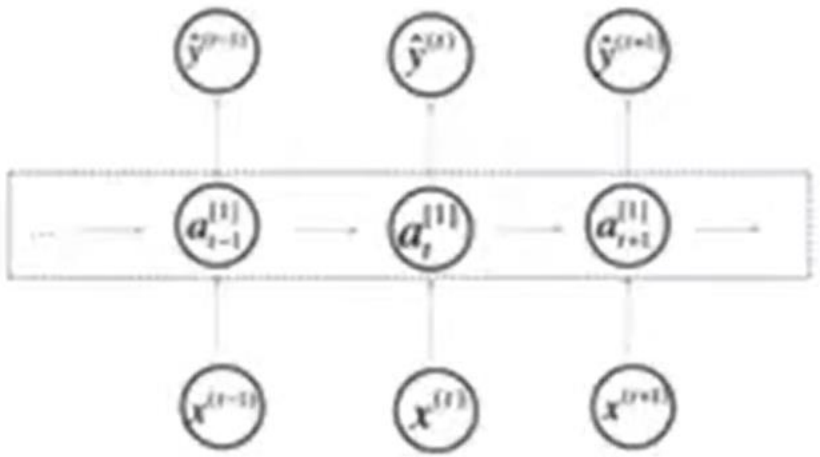
Neural Network examples



Standard NN



Convolutional NN



Recurrent NN

Supervised Learning

Structured Data

Size	#bedrooms	···	Price (1000$s)
2104	3		400
1600	3		330
2400	3		369
⋮	⋮		⋮
3000	4		540

User Age	Ad Id	···	Click
41	93242		1
80	93287		0
18	87312		1
⋮	⋮		⋮
27	71244		1

Unstructured Data

Audio

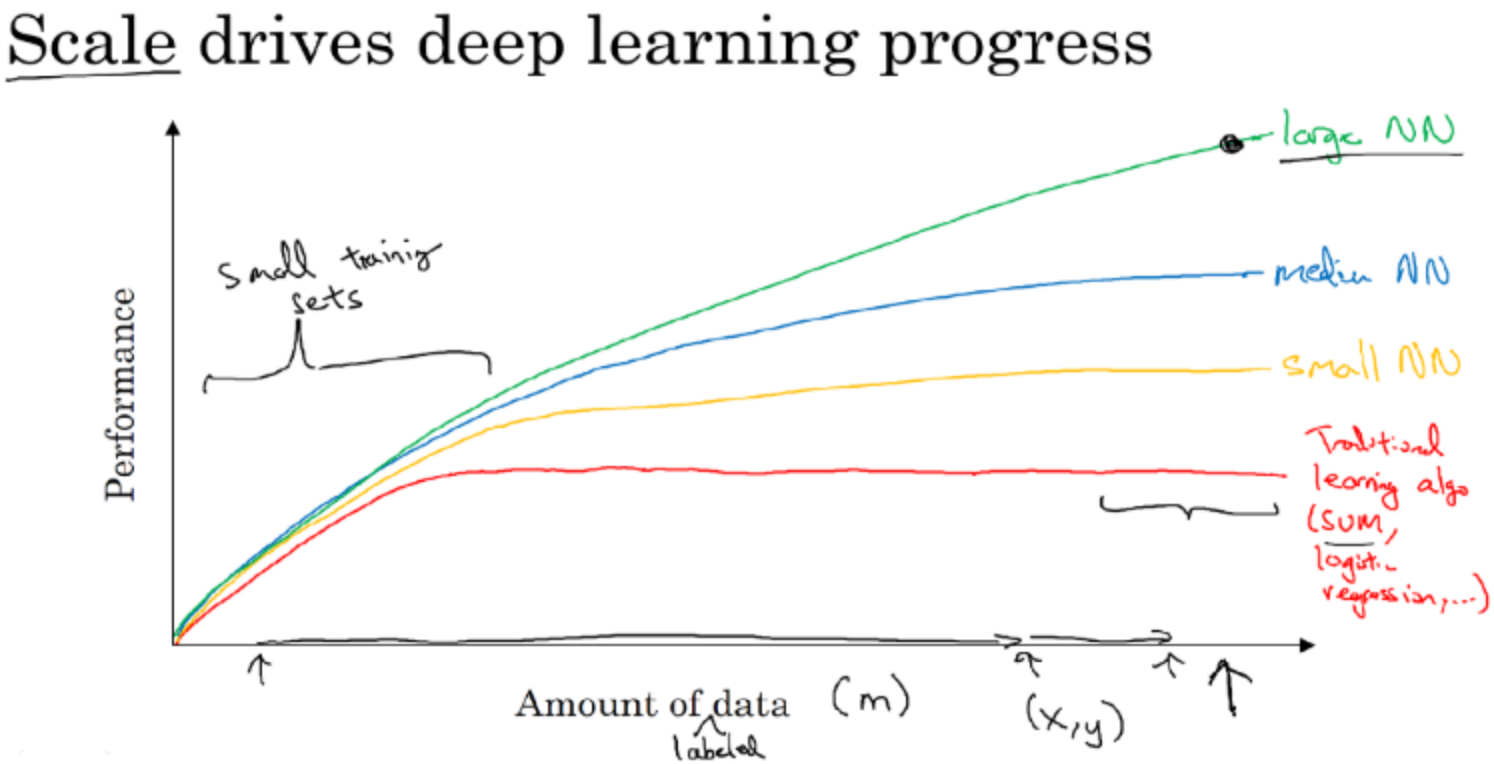
Image

Four scores and seven years ago…

Text

Why is deep learning taking off?

Deep learning is taking off due to a large amount of data available through the digitization of the society, faster computation and innovation in the development of neural network algorithm.

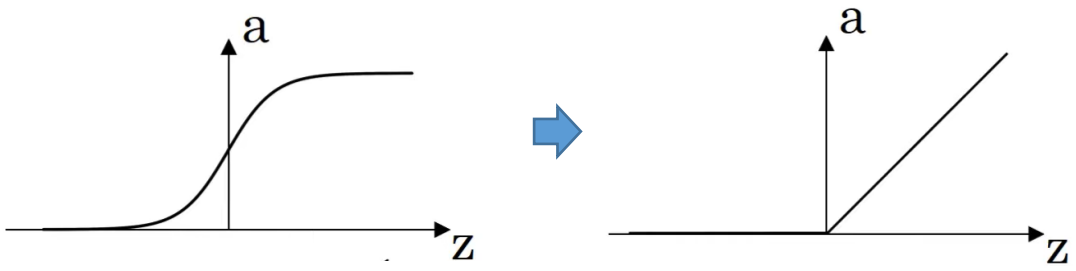


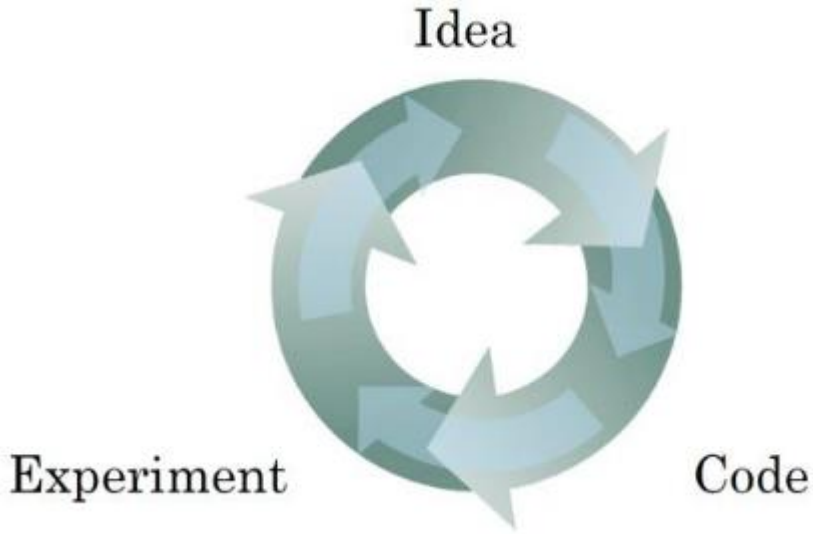
Two things have to be considered to get to the high level of performance:

1. Being able to train a big enough neural network
2. Huge amount of labeled data

Scale drives deep learning progress

- Data
- Computation
- Algorithm





Basics of Neural Network Programming

Trong-Hop Do

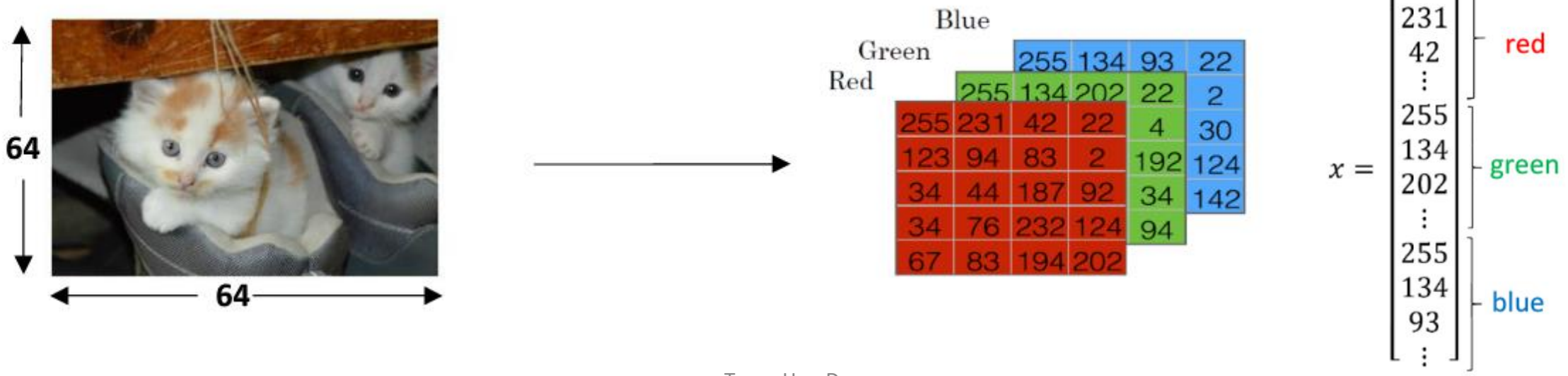
Binary classification

In a binary classification problem, the result is a discrete value output.

For example - account hacked (1) or compromised (0)
- a tumor malign (1) or benign (0)

Example: Cat vs Non-Cat

The goal is to train a classifier that the input is an image represented by a feature vector, x, and predicts whether the corresponding label y is 1 or 0. In this case, whether this is a cat image (1) or a non-cat image (0).



Logistic Regression

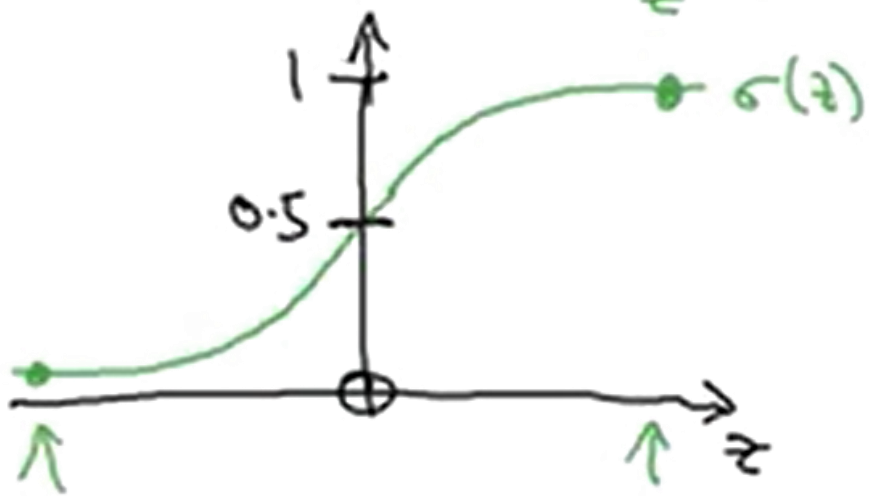
Given x, want $\hat{y} = P(y=1|x)$

$0 \le \hat{y} \le 1$

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

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

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



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

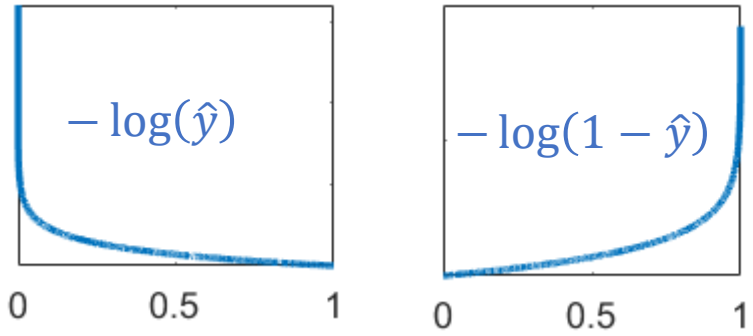
If z large $\sigma(z) \approx \frac{1}{1+0} = 1$

If z large negative number

$$\sigma(z) = \frac{1}{1+e^{-z}} \approx \frac{1}{1+\text{BigNum}} \approx 0$$

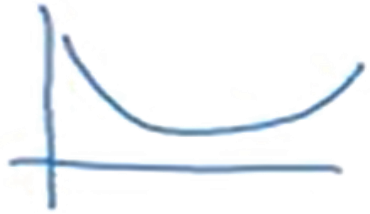
Logistic Regression cost function

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



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

Loss (error) function: $\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$



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

- If $y = 1$: $\mathcal{L}(\hat{y}, y) = -\log \hat{y}$ ← $\hat{y}$ should be close to 1
- If $y = 0$: $\mathcal{L}(\hat{y}, y) = -\log(1-\hat{y})$ ← $\hat{y}$ should be close to 0

Cross Entropy Cost function:

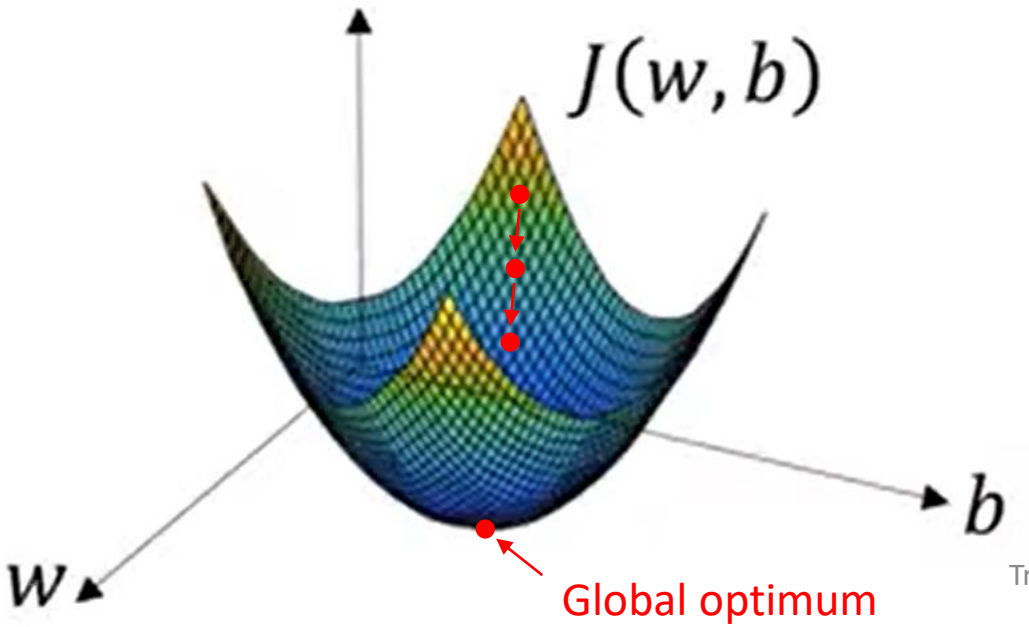
$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i}^{m} \left[y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log(1-\hat{y}^{(i)})\right]$$

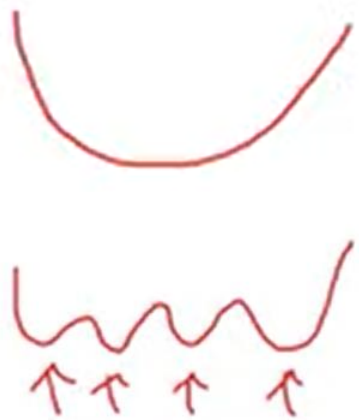
Gradient Descent

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

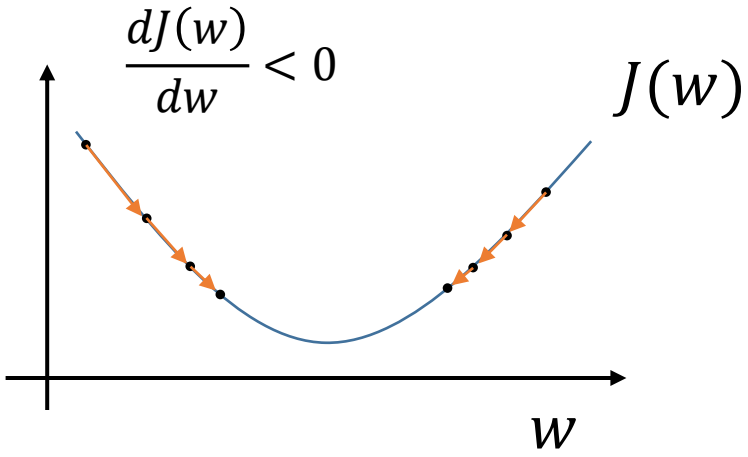
$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

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





Gradient Descent

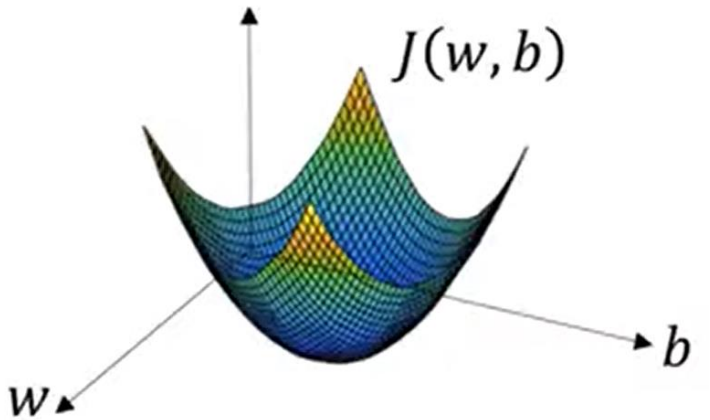


Repeat {

Learning rate

$$w := w - \alpha \underbrace{\frac{dJ(w)}{dw}}_{\text{Update}}$$

}



Repeat {

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

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

}

Computation Graph

$$J(a,b,c) = 3(a + \underbrace{bc}_{u})$$

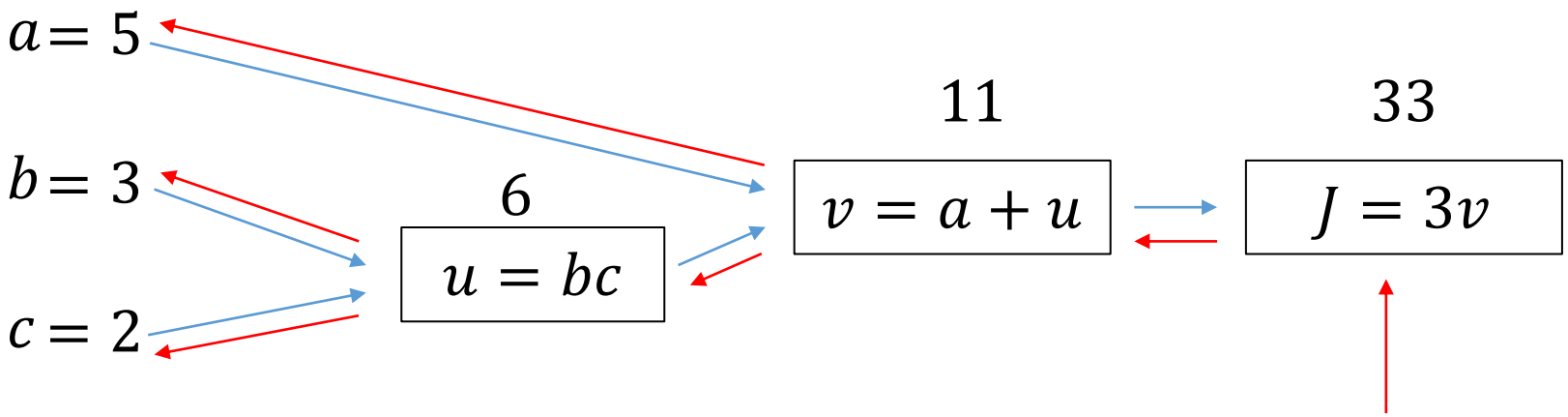
$$\underbrace{a + \underbrace{bc}_{u}}_{v}$$

$$\underbrace{3\underbrace{(a + \underbrace{bc}_{u})}_{v}}_{J}$$

$$u = bc$$

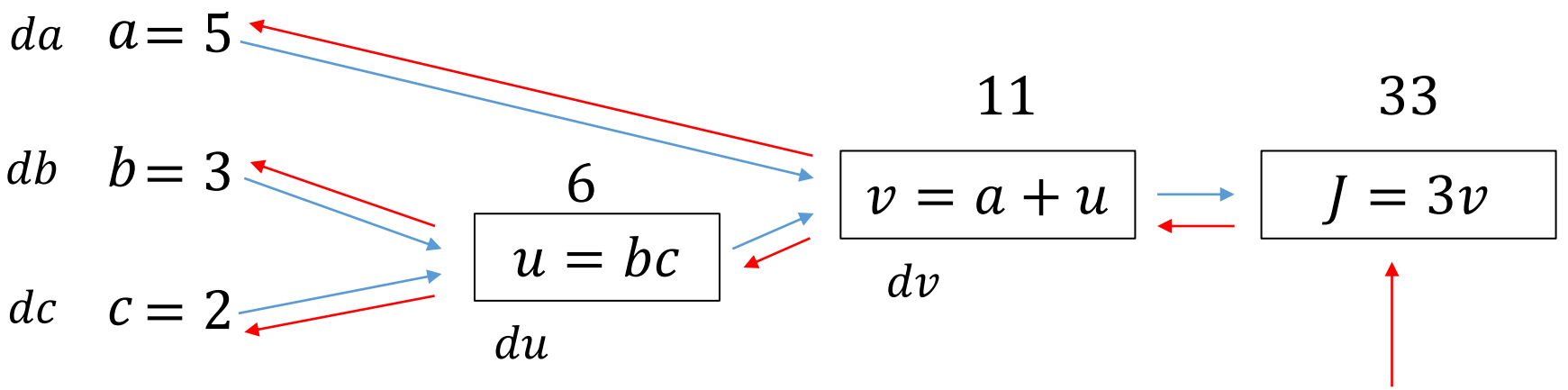
$$v = a + u$$

$$J = 3v$$



Derivatives with Computation Graphs

Chain rule $F(x) = f(g(x)) \Rightarrow F'(x) = f'(g(x))g'(x).$



$$\frac{\partial J}{\partial a} = \frac{\partial J}{\partial v}\frac{\partial v}{\partial a}$$

da 3 1

$$\frac{\partial J}{\partial u} = \frac{\partial J}{\partial v}\frac{\partial v}{\partial u}$$

du 3 1

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial u}\frac{\partial u}{\partial b}$$

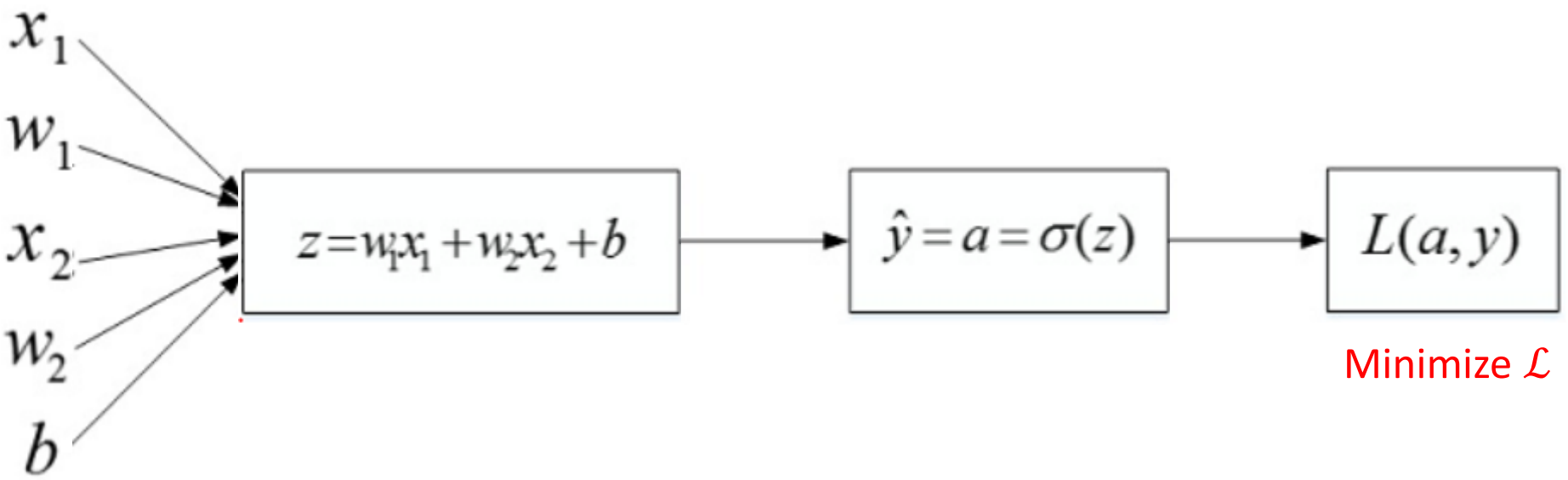
db 3 2

$$\frac{\partial J}{\partial c} = \frac{\partial J}{\partial u}\frac{\partial u}{\partial c}$$

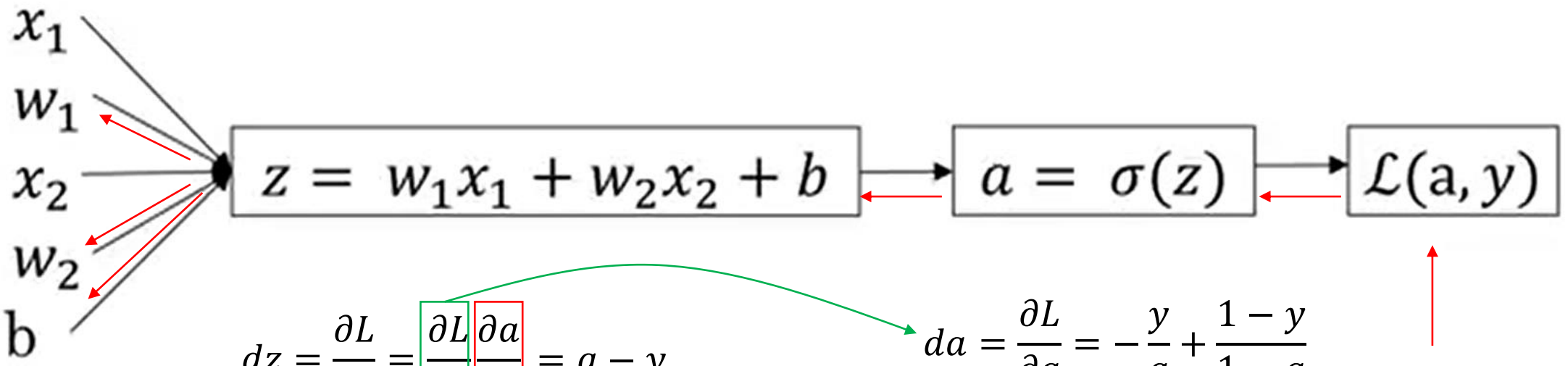
dc 3 3

Logistic Regression Radient descent

$$z = w^T x + b$$
$$\hat{y} = a = \sigma(z)$$
$$\mathcal{L}(a, y) = -(y \log(a) + (1 - y) \log(1 - a))$$



Logistic regression derivatives



$$dz = \frac{\partial L}{\partial z} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z} = a - y$$

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

$a(1-a)$

$$dw_1 = \frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial w_1} = x_1 dz \qquad dw_2 = x_2 dz \qquad db = dz$$

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

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

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

Gradient Descent on m Examples

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

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

$$\frac{\partial J}{\partial w_i} = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial L}{\partial w_i} = \frac{1}{m}\sum_{i=1}^{m} dw_i^{(i)}$$

$$dw_1 = \frac{1}{m}\sum_{i=1}^{m} x_1^{(i)}(a^{(i)} - y^{(i)})$$

$$dw_2 = \frac{1}{m}\sum_{i=1}^{m} x_2^{(i)}(a^{(i)} - y^{(i)})$$

$$db = \frac{1}{m}\sum_{i=1}^{m}(a^{(i)} - y^{(i)})$$

Gradient Descent on m Examples

```
J=0; dw1=0; dw2=0; db=0;
for i = 1 to m
    z(i) = wx(i)+b;
    a(i) = sigmoid(z(i));
    J += -[y(i)log(a(i))+(1-y(i)）log(1-a(i));
    dz(i) = a(i)-y(i);
    dw1 += x1(i)dz(i);
    dw2 += x2(i)dz(i);
    db += dz(i);
J /= m;
dw1 /= m;
dw2 /= m;
db /= m;
```

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

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

$$w_m := w_m - \alpha\, dw_m$$

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

Vectorizing Logistic Regression

dw = np.zeros(n_x,1))

J = 0, ~~dw1 = 0, dw2 = 0,~~ db = 0

for i = 1 to m:

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

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

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

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

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

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

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

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

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

dw /=m

Vetorizing Logistic Regression

$$z^{(1)} = w^T x^{(1)} + b \qquad z^{(2)} = w^T x^{(2)} + b \qquad z^{(3)} = w^T x^{(3)} + b$$

$$a^{(1)} = \sigma(z^{(1)}) \qquad a^{(2)} = \sigma(z^{(2)}) \qquad a^{(3)} = \sigma(z^{(3)})$$

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

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

$[b\ b \cdots b]$: $1 \times m$; $[w^T x^{(1)} + b \quad \cdots]$: $1 \times m$

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

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

Implementing Logistic Regression

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

for i = 1 to m:

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

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

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

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

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

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

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

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

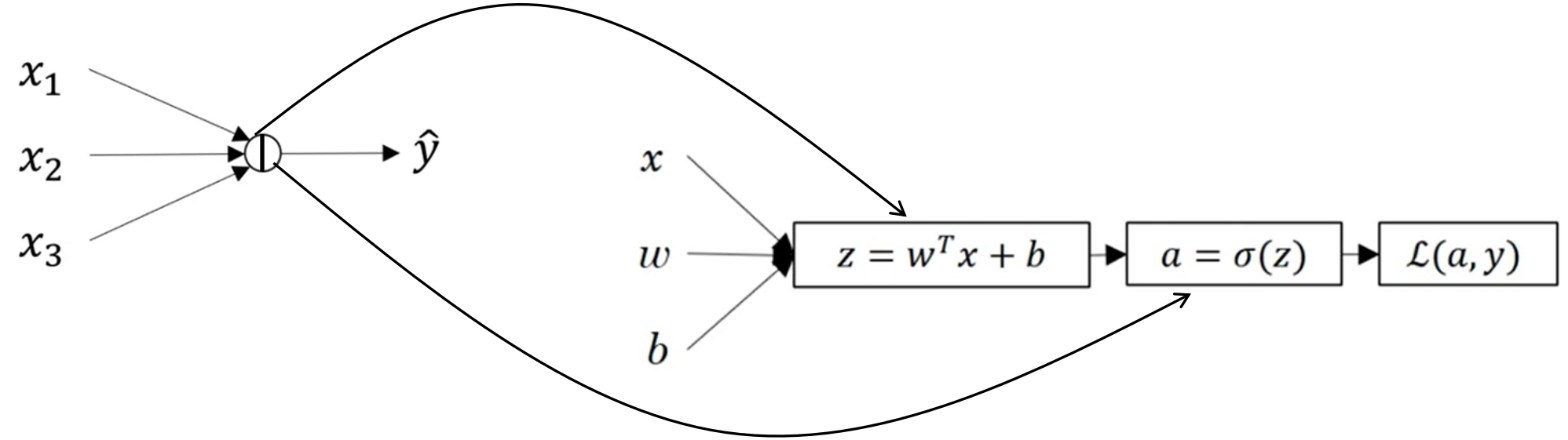
$db = db/m$

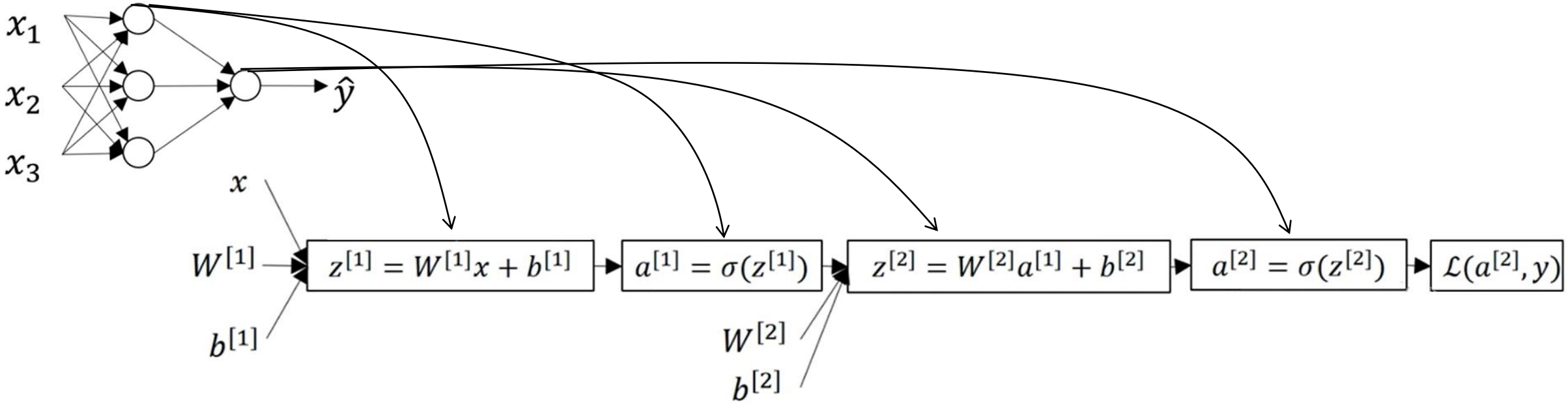
$$db = \frac{1}{m}\sum_{i=1}^{m} dz^{(i)} \qquad dw = \frac{1}{m} X \cdot dZ^T$$

```
Z = np.dot(w.T,X) + b
A = sigmoid(Z)
dZ = A-Y
dw = 1/m*np.dot(X,dZ.T)
db = 1/m*np.sum(dZ)

w = w - alpha*dw
b = b - alpha*db
```

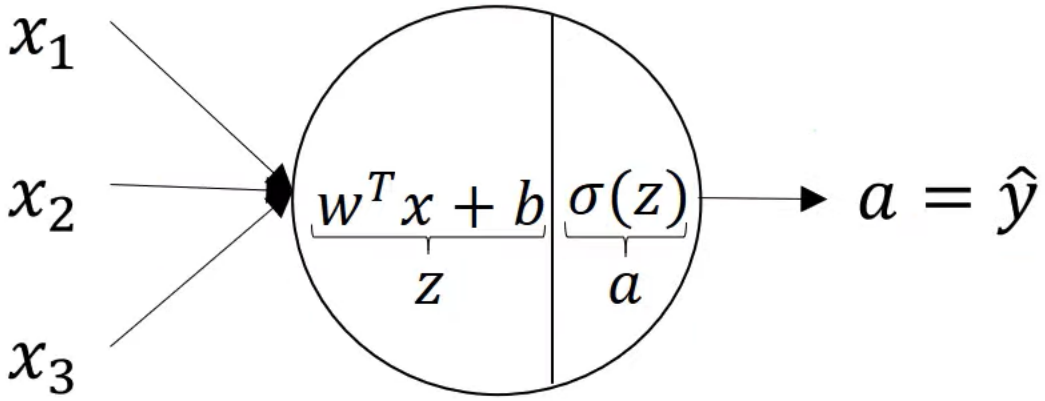
One hidden layer Neural Network

What is neural network?



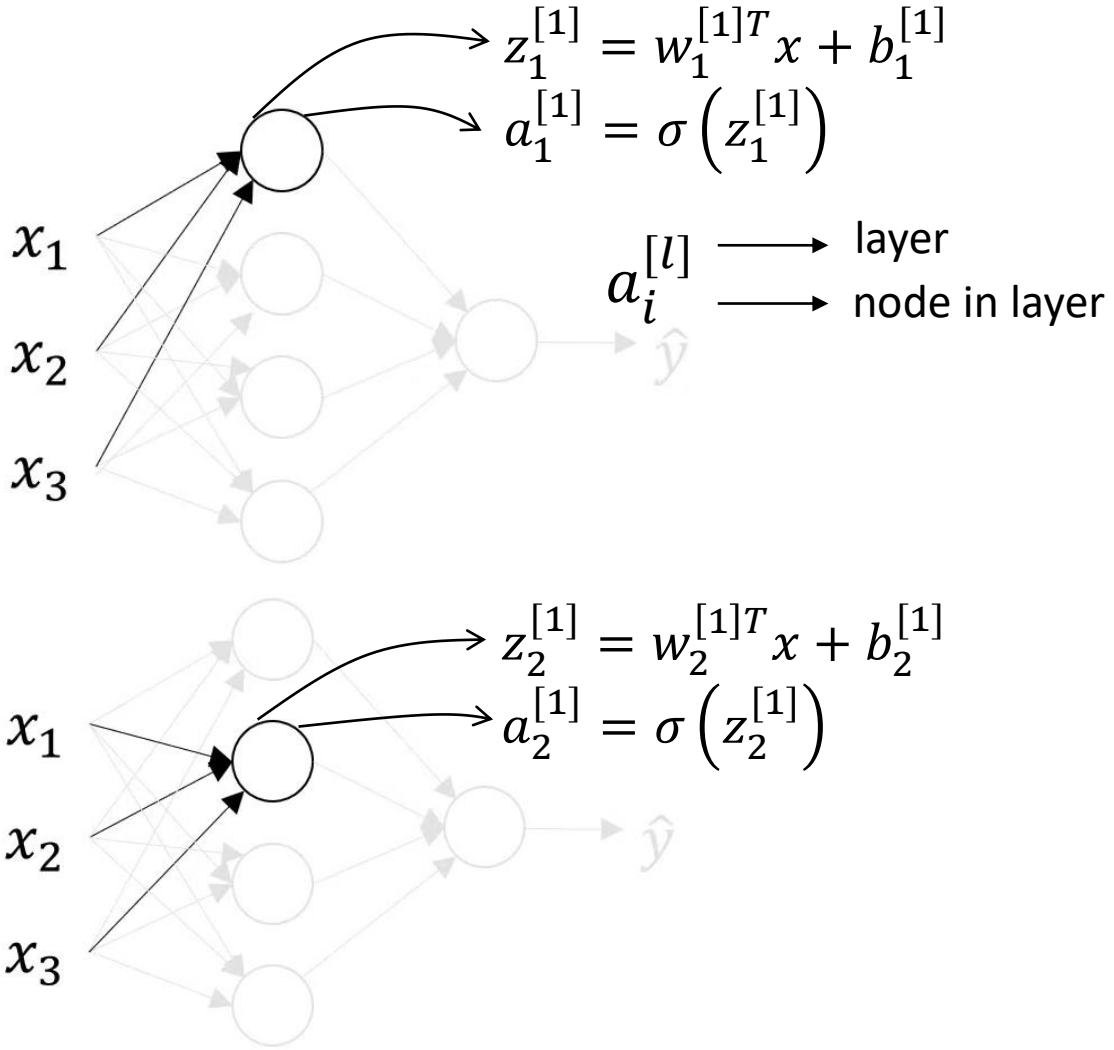


Neural Network Representation

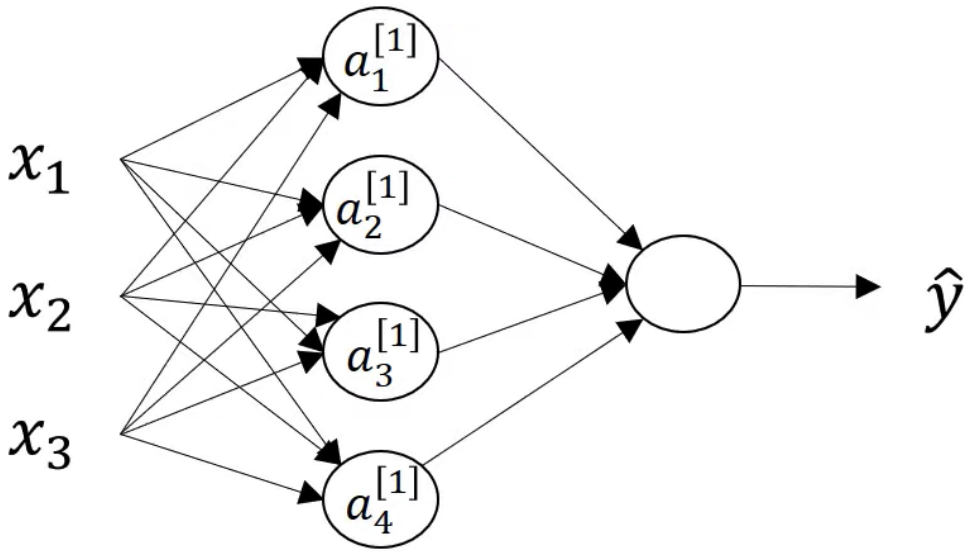


$$z = w^T x + b$$

$$a = \sigma(z)$$



Neural Network Representation



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

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

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

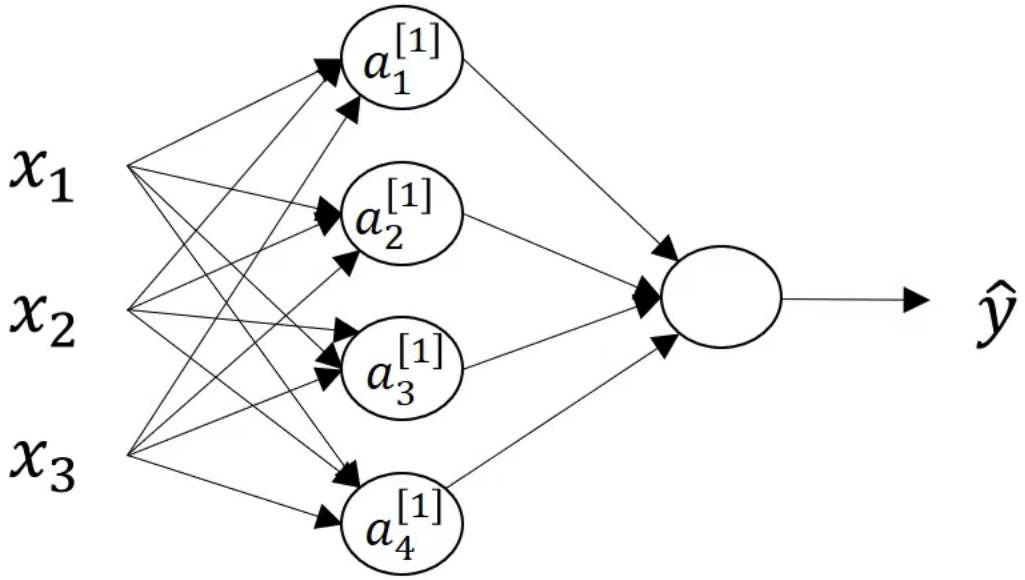
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]},\ a_4^{[1]} = \sigma(z_4^{[1]})$$

(4,3)

$$z^{[1]} = \begin{bmatrix} w_1^{[1]T} \\ w_2^{[1]T} \\ w_3^{[1]T} \\ w_4^{[1]T} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}$$

$W^{[1]}$ $b^{[1]}$

Neural Network Representation learning



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

$$a = \sigma(z)$$

Given input x:

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

(4,1) (4,3) (3,1) (4,1)

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

(4,1) (4,1)

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

(1,1) (1,4) (4,1) (1,1)

$$a^{[2]} = \sigma(z^{[2]})$$

(1,1) (1,1)

Vectorizing across multiple examples

```
for i = 1 to m:
```

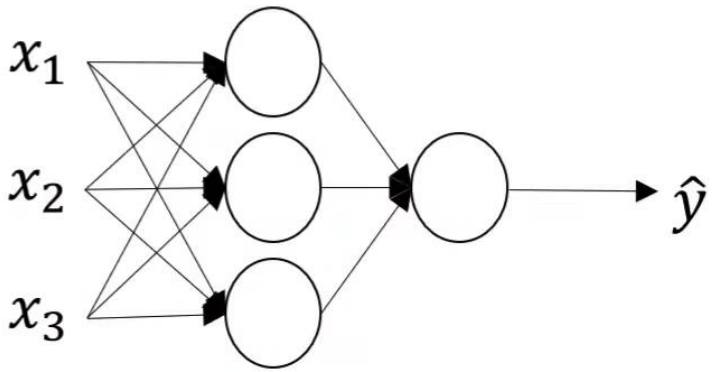
$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$
$$a^{[1](i)} = \sigma(z^{[1](i)})$$
$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$
$$a^{[2](i)} = \sigma(z^{[2](i)})$$

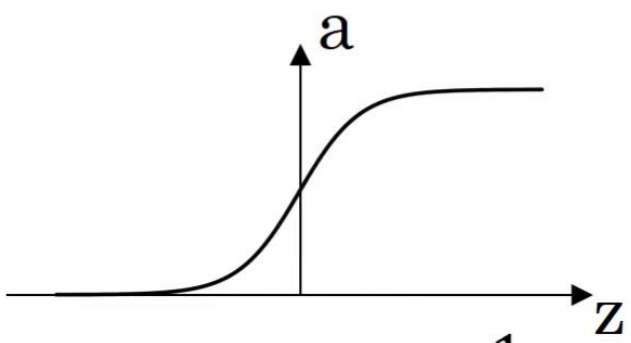
$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

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

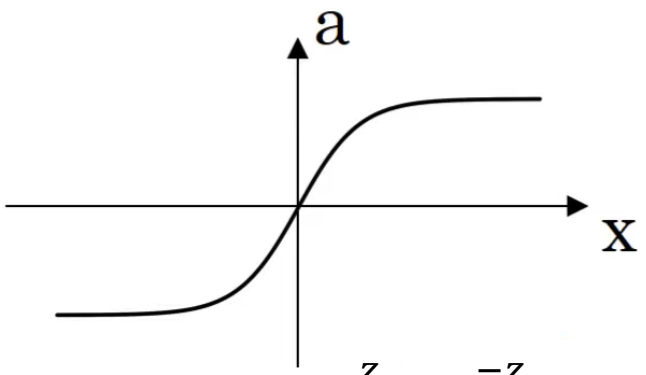
$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

Activation functions





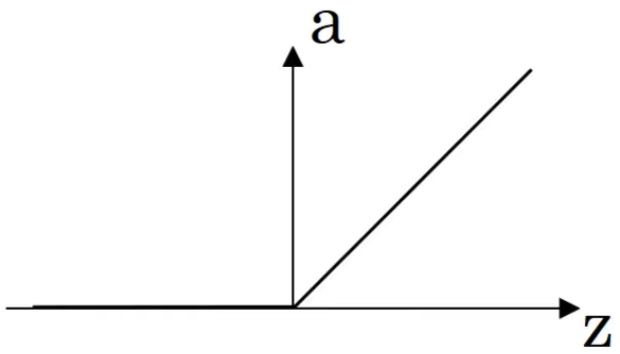
sigmoid: $a = \dfrac{1}{1+e^{-z}}$



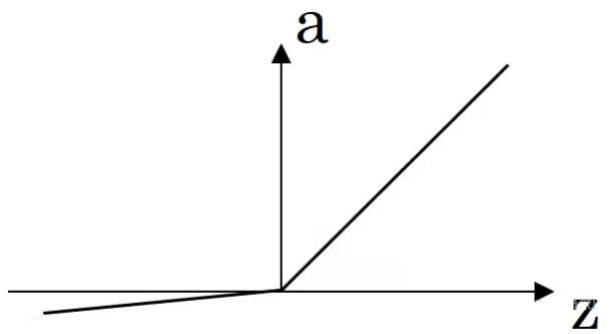
tanh: $a = \dfrac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$

Given x:

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

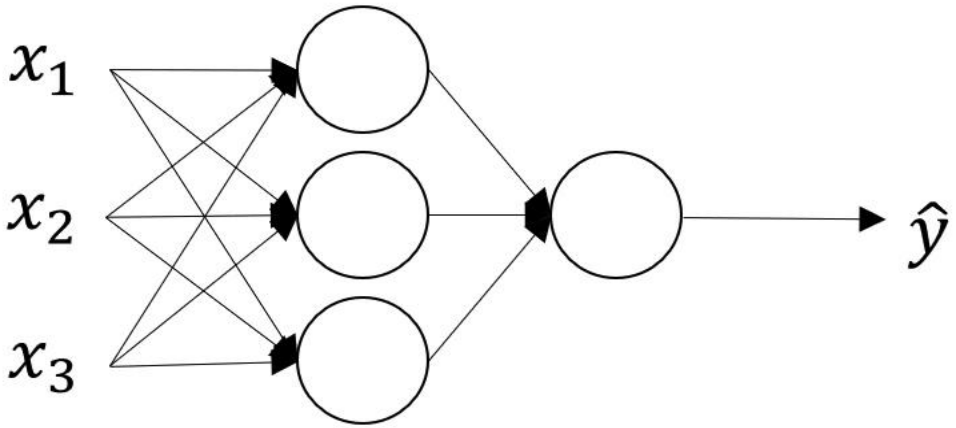


ReLu: $a = \max(0, z)$



Leaky ReLu: $a = \max(0.01z, z)$

Activation function



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

$$= W^{[2]}(W^{[1]}x + b^{[1]}) + b^{[2]}$$

$$= (W^{[2]}W^{[1]})x + (W^{[2]}b^{[1]} + b^{[2]})$$

$$= \boxed{W'x + b'}$$

Given x:

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

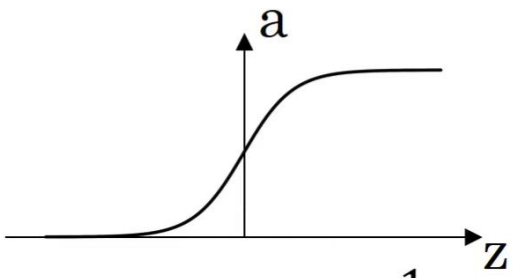
~~$a^{[1]} = g^{[1]}(z^{[1]})$~~ $\quad a^{[1]} = z^{[1]}$

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

~~$a^{[2]} = g^{[2]}(z^{[2]})$~~ $\quad a^{[2]} = z^{[2]}$

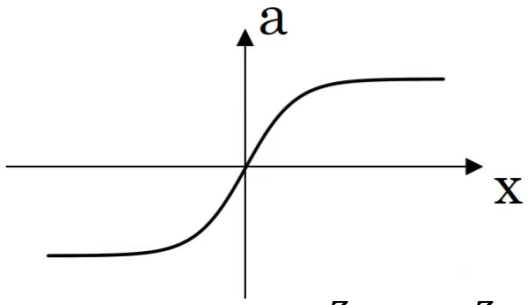
With linear activation function, the outputs on later layers are also linear combination of the input. So the hidden layer (even many of them) has no meaning.

Derivatives of activation function



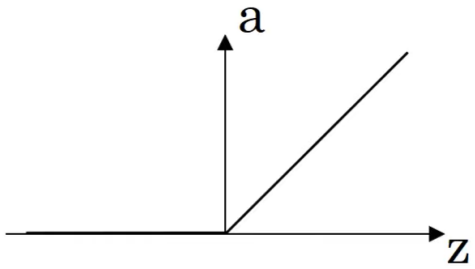
sigmoid: $a = \dfrac{1}{1+e^{-z}}$

$g'(z) = g(z)\big(1-g(z)\big)$
$= a(1-a)$



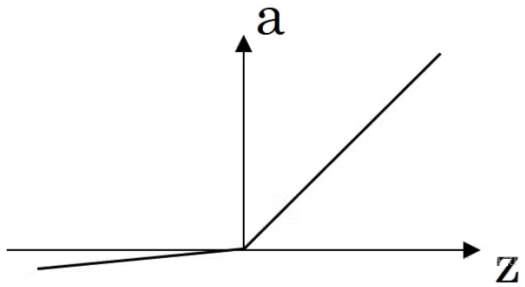
tanh: $a = \dfrac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$

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



ReLu: $a = \max(0, z)$

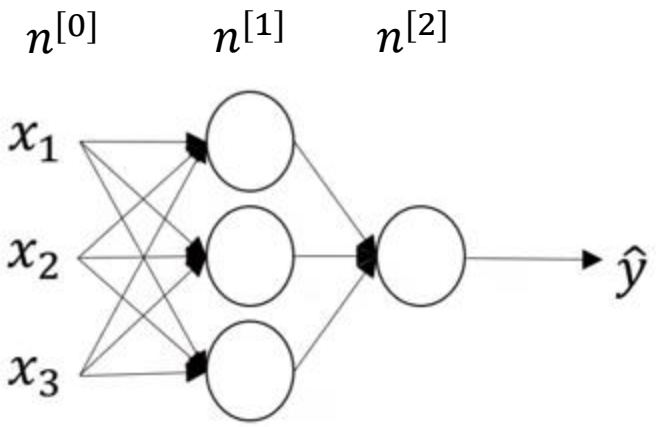
$g'(z) = \begin{cases} 0 \text{ if } z < 0 \\ 1 \text{ if } z \geq 0 \end{cases}$



Leaky ReLu: $a = \max(0.01z, z)$

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

Gradient descent for neural network



(n^1, n^0) $(n^1, 1)$ $(n^{[2]}, n^{[1]})$ $(n^{[1]}, 1)$

Parameter: $W^{[1]}$, $b^{[1]}$, $W^{[2]}$, $b^{[2]}$

Cost function: $J = -\frac{1}{m}\sum_{i=0}^{m}\left(y^{(i)}\log\left(a^{[2](i)}\right) + (1-y^{(i)})\log\left(1-a^{[2](i)}\right)\right)$

$Repeat$ {

Compute $a^{[1]}, z^{[1]}, \boldsymbol{a}^{[2]}, z^{[2]}$

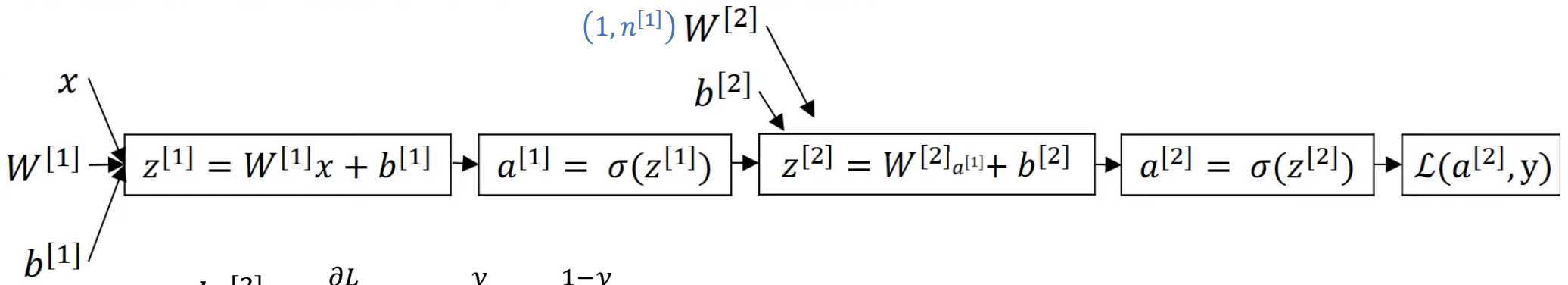
$dW^{[2]} = \frac{\partial J}{\partial W^{[2]}}$, $db^{[2]} = \frac{\partial J}{\partial b^{[2]}}$, $dW^{[1]} = \frac{\partial J}{\partial W^{[1]}}$, $db^{[1]} = \frac{\partial J}{\partial b^{[1]}}$

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

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

}

Gradient descent for neural network



$$da^{[2]} = \frac{\partial L}{\partial a^{[2]}} = -\frac{y}{a^{[2]}} + \frac{1-y}{1-a^{[2]}}$$

$$dz^{[2]} = \frac{\partial L}{\partial z^{[2]}} = \frac{\partial L}{\partial a^{[2]}}\frac{\partial a^{[2]}}{\partial z^{[2]}} = da^{[2]}\big(a(1-a)\big) = a^{[2]} - y$$

$$dW^{[2]} = \frac{\partial L}{\partial W^{[2]}} = \frac{\partial L}{\partial z^{[2]}}\frac{\partial z^{[2]}}{\partial W^{[2]}} = dz^{[2]}a^{[1]^T}$$

$$db^{[2]} = \frac{\partial L}{\partial b^{[2]}} = \frac{\partial L}{\partial z^{[2]}}\frac{\partial z^{[2]}}{\partial b^{[2]}} = dz^{[2]}$$

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

$$da^{[1]} = \frac{\partial L}{\partial a^{[1]}} = \frac{\partial L}{\partial z^{[2]}}\frac{\partial z^{[2]}}{\partial a^{[1]}} = W^{[2]^T}dz^{[2]}$$

$$dz^{[1]} = \frac{\partial L}{\partial z^{[1]}} = \frac{\partial L}{\partial a^{[1]}}\frac{\partial a^{[1]}}{\partial z^{[1]}} = da^{[1]}g^{[1]'}(z^{[1]}) = W^{[2]^T}dz^{[2]} * g^{[1]'}(z^{[1]})$$

Same goes for $dW^{[1]}$ and $db^{[1]}$

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

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

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

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

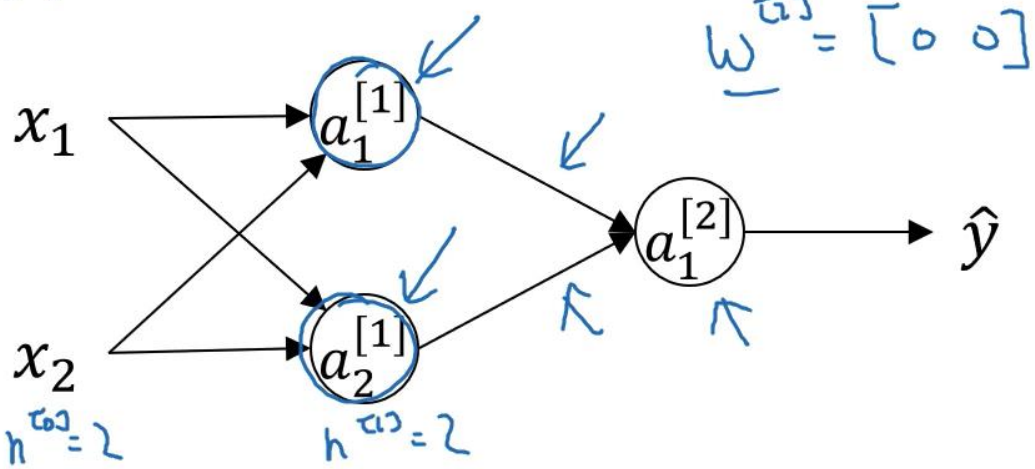
$$dW^{[1]} = dz^{[1]}x^T$$

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

Summary of gradient descent for neural network

$dz^{[2]} = a^{[2]} - y$	$dZ^{[2]} = A^{[2]} - Y$
$dW^{[2]} = dz^{[2]} a^{[1]^T}$	$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]^T}$
$db^{[2]} = dz^{[2]}$	$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$
$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$	$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$
$dW^{[1]} = dz^{[1]} x^T$	$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$
$db^{[1]} = dz^{[1]}$	$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True)$

What happens if you initialize weights to zero?



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

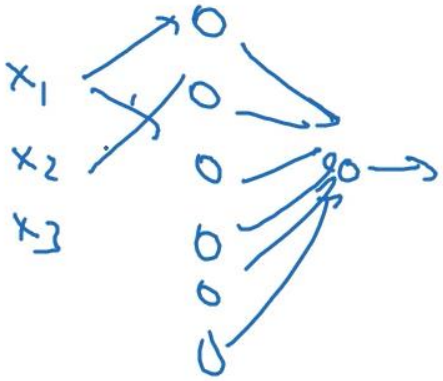
$$w^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$a_1^{[1]} = a_2^{[1]} \qquad dz_1^{[1]} = dz_2^{[1]}$$

$$dw = \begin{bmatrix} u & v \\ u & v \end{bmatrix}$$

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

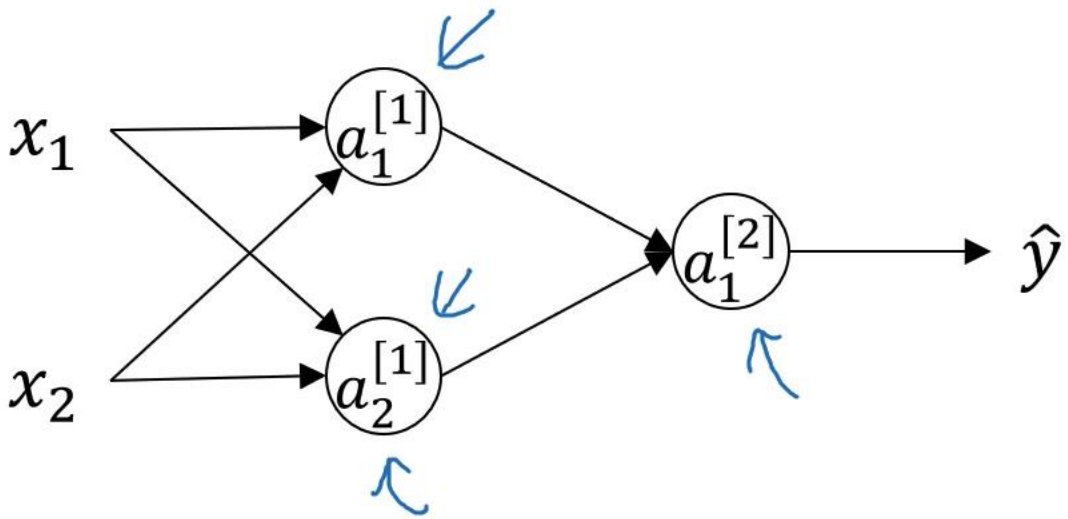
Symmetric



$$w^{[1]} = \begin{bmatrix} \cdots & \cdots \\ \cdots & \cdots \end{bmatrix}$$

If you initialize weights to zeros, all rows in matrix W will be the same.

Random initialization



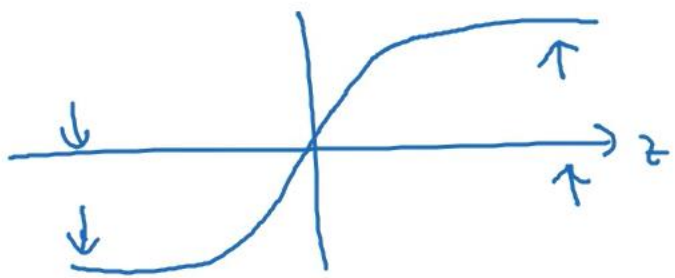
$$z^{[1]} = W^{[1]} x + b^{[1]}$$
$$a^{[1]} = g^{[1]}(z^{[1]})$$

$W^{[1]} = np.random.randn((2,2)) * 0.01$ / 100 ?

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

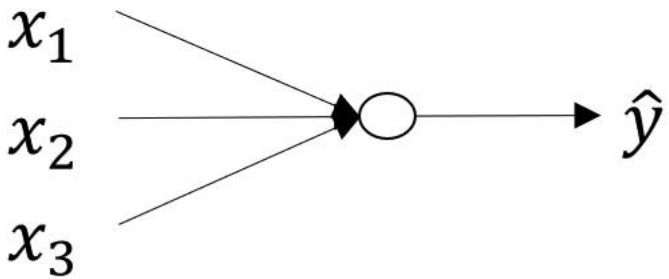
$W^{[2]} = np.random.randn((1,2)) * 0.01$

$b^{[2]} = 0$

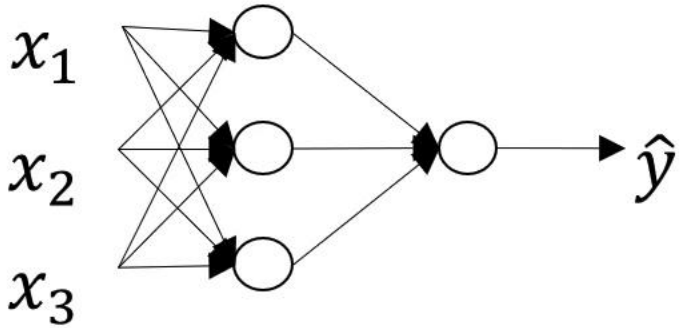


Deep L-Layer Neural Network

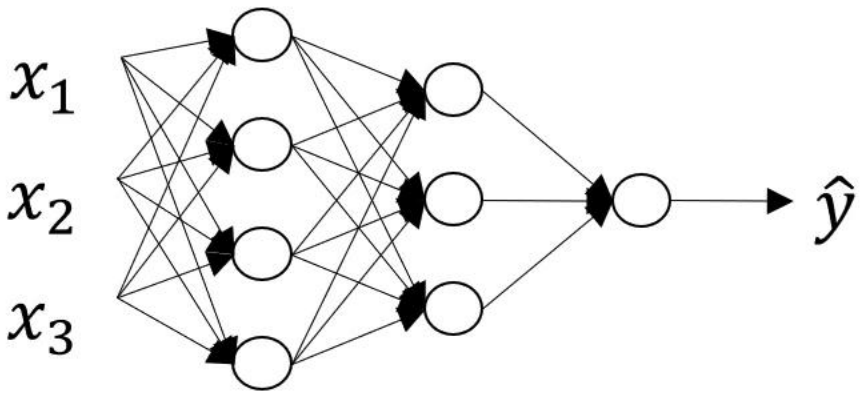
What is a deep neural network?



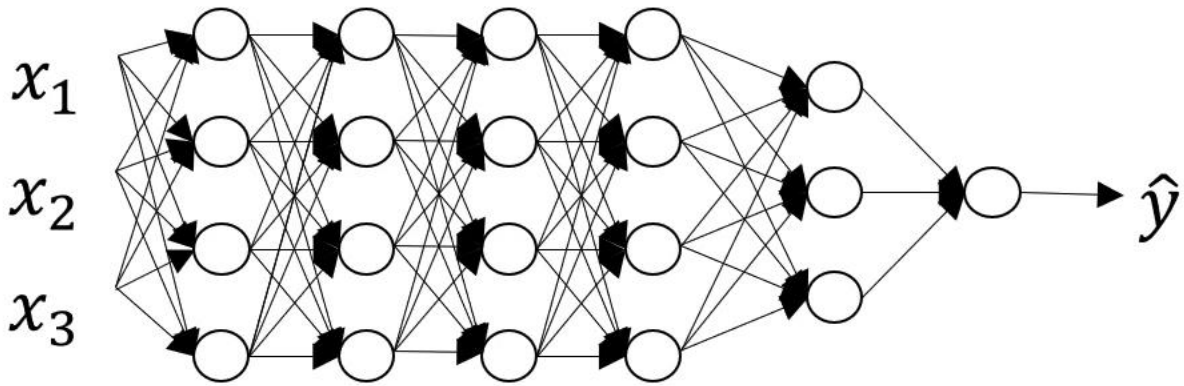
logistic regression



1 hidden layer

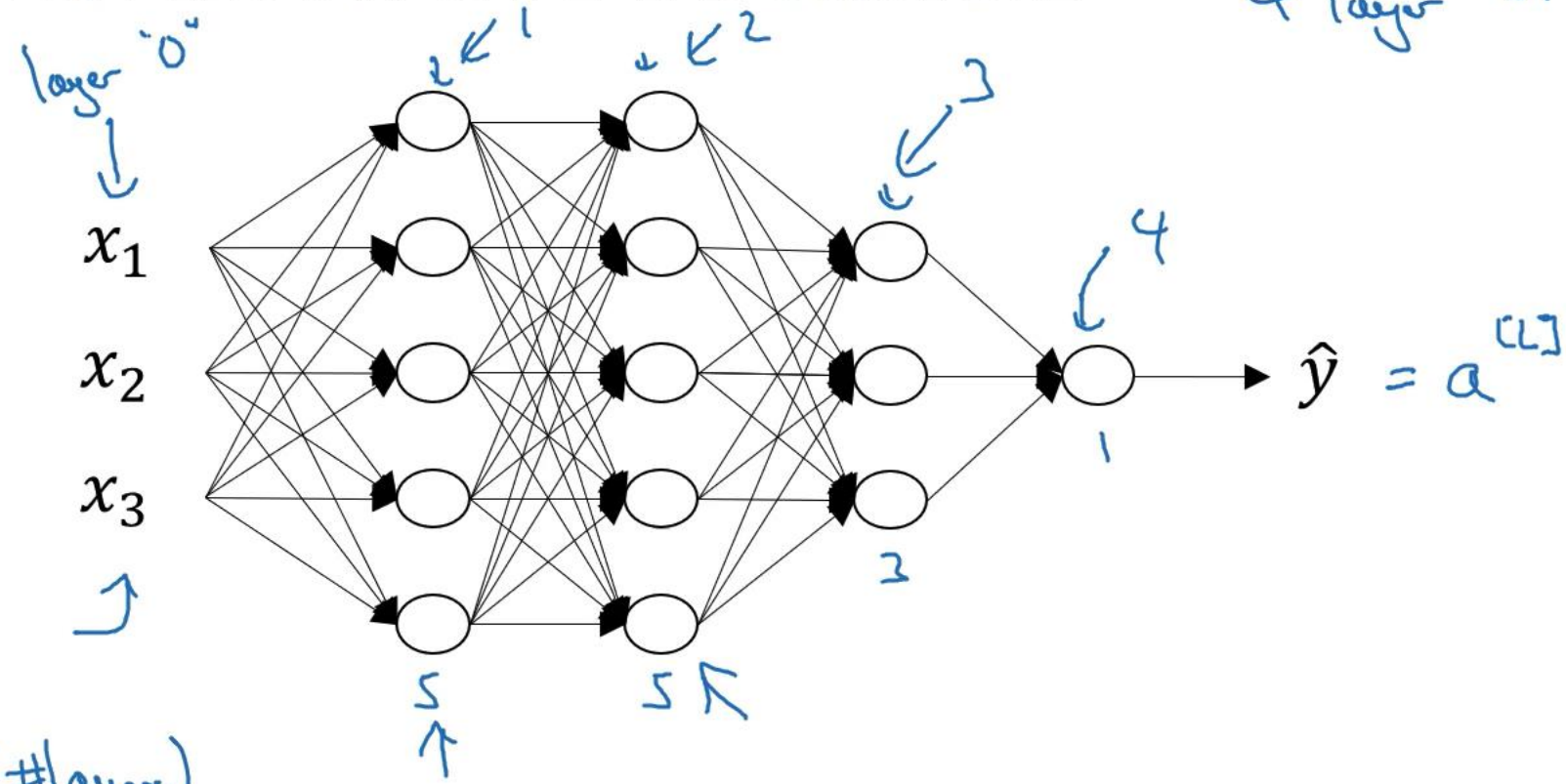


2 hidden layers



5 hidden layers

Deep neural network notation



4 layer NN

$X = a^{[0]}$

$L = 4$ (#layers)

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

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

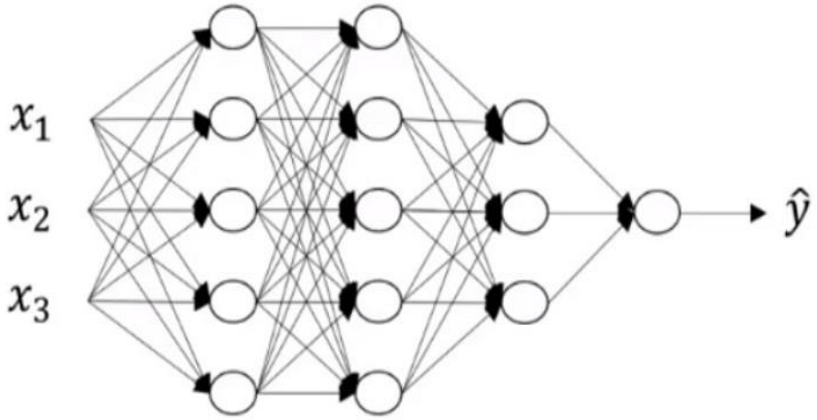
$n^{[0]} = n_x = 3$

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

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

$b^{[l]}$

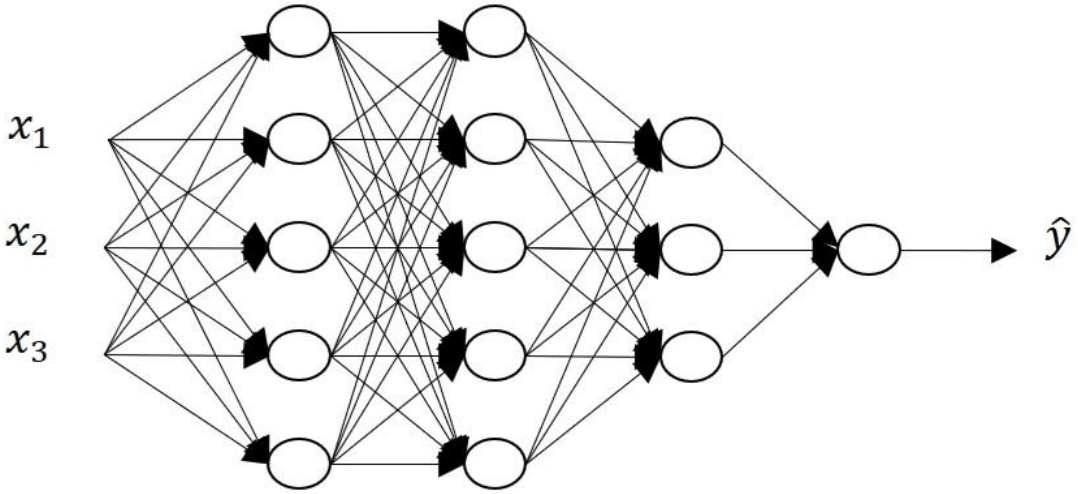
Deep neural network notation



X : $(12288, 209)$ (with $m = 209$ examples)

	Shape of W	Shape of b	Activation	Shape of Activation
Layer 1	$(n^{[1]}, 12288)$	$(n^{[1]}, 1)$	$Z^{[1]} = W^{[1]} X + b^{[1]}$	$(n^{[1]}, 209)$
Layer 2	$(n^{[2]}, n^{[1]})$	$(n^{[2]}, 1)$	$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$	$(n^{[2]}, 209)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Layer L-1	$(n^{[L-1]}, n^{[L-2]})$	$(n^{[L-1]}, 1)$	$Z^{[L-1]} = W^{[L-1]} A^{[L-2]} + b^{[L-1]}$	$(n^{[L-1]}, 209)$
Layer L	$(n^{[L]}, n^{[L-1]})$	$(n^{[L]}, 1)$	$Z^{[L]} = W^{[L]} A^{[L-1]} + b^{[L]}$	$(n^{[L]}, 209)$

Forward Propagation in a Deep Network



Vectorizing across m training example

$$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$$

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

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

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

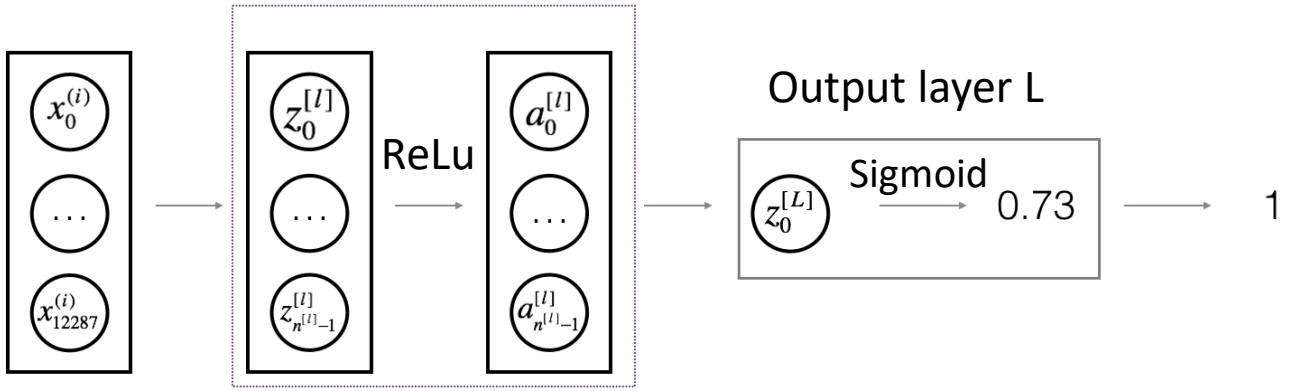
Implement forward propagation

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

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

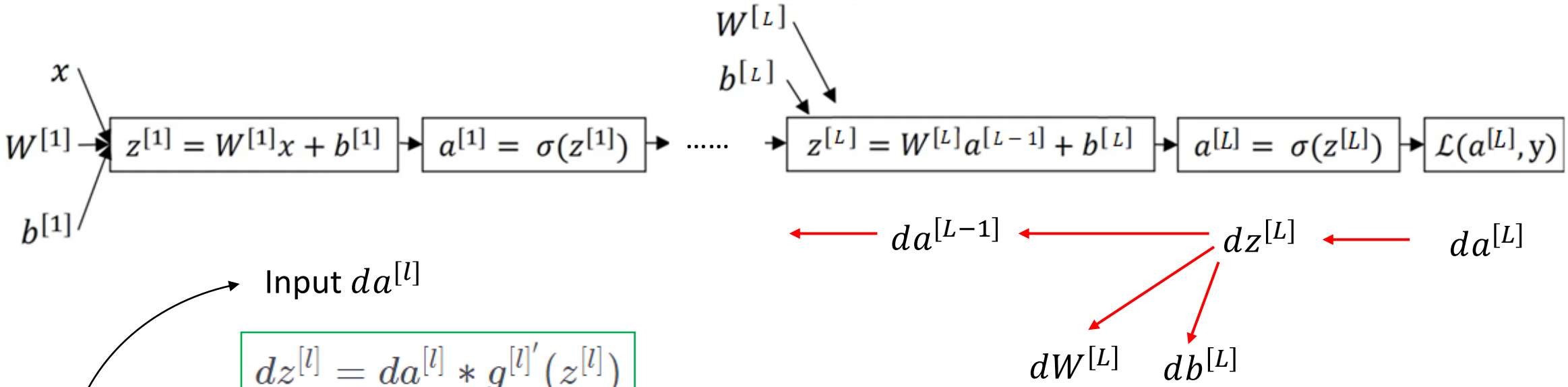
```
def forwardprop(A_prev, W, b, activation):

    Z = np.dot(W, A) + b

    if activation == "sigmoid":

        A = sigmoid(Z)

    elif activation == "relu":

        A = relu(Z)

    elif activation == "tanh":

        A = tanh(Z)

    cache = (Z, W)

    return A, cache
```

Implement forward propagation




```
def L_model_forward(X, parameters):

    A = X

    for l in range(1, L):
        A_prev = A

        A, cache = forwardprop(A_prev, W_l, b_l, activation= 'relu')

        AL, cache = forwardprop(A, W_L, b_L, activation= 'sigmoid')

    return AL, caches
```

Backward Propagation in a Deep Network



Input $da^{[l]}$

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

$$dW^{[l]} = dz^{[l]} \cdot a^{[l-1]^T}$$

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

$$da^{[l-1]} = W^{[l]T} \cdot dz^{[l]}$$

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

With cross-entropy cost function:

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

Vectorization of backward Propagation in a Deep Network

With cross-entropy cost function:

$$dA^{[L]} = \frac{\partial L}{\partial A^{[L]}} = -\frac{Y}{A^{[L]}} + \frac{1-Y}{1-A^{[L]}}$$

$$dZ^{[l]} = dA^{[l]} * g^{[l]'}(Z^{[l]})$$

$$dW^{[l]} = \frac{1}{m} dZ^{[l]} \cdot A^{[l-1]T}$$

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

$$dA^{[l-1]} = W^{[l]T} \cdot dZ^{[l]}$$

Implement backward propagation

$$dZ^{[l]} = dA^{[l]} * g^{[l]'}(Z^{[l]})$$

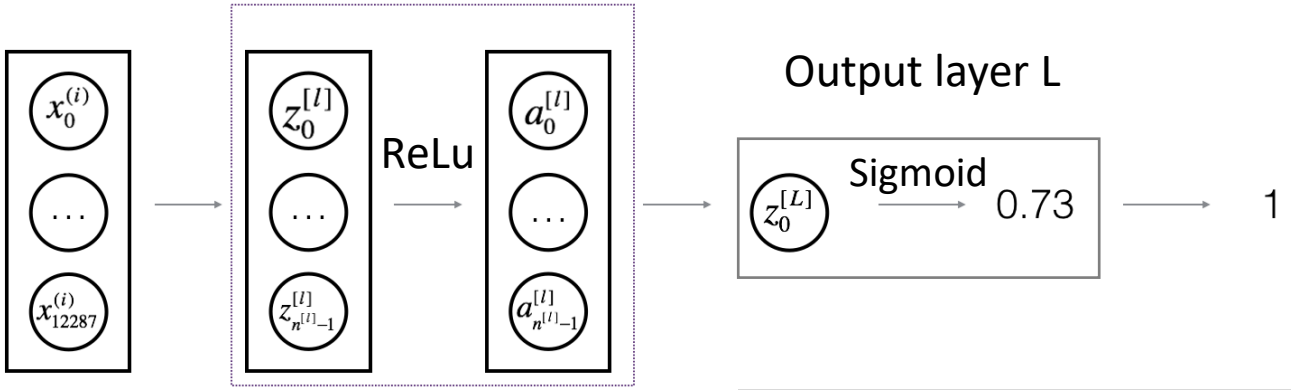
$$dW^{[l]} = \frac{1}{m} dZ^{[l]} \cdot A^{[l-1]T}$$

$$db^{[l]} = \frac{1}{m} np.sum(dZ^{[l]})$$

$$dA^{[l-1]} = W^{[l]T} \cdot dZ^{[l]}$$

```
def backprop(dA, cache, activation):

    Z, W = cache

    if activation == "relu":

        dZ = relu_backward(dA, Z)

    elif activation == "sigmoid":

        dZ = sigmoid_backward(dA, Z)

    elif activation == "tanh":
        dZ = tanh_backward(dA, Z)

    dA_prev, dW, db = linear_backward(dZ, A_prev, W)

    return dA_prev, dW, db
```

Implement backward propagation



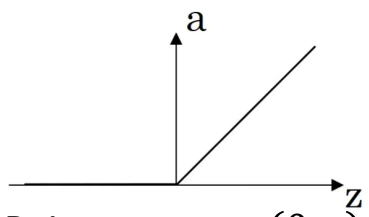
With cross-entropy cost function:

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

```
def L_model_backward(AL, Y, caches):

    # Initializing the backpropagation
    dAL = - (Y/AL)  + (1-Y)/(1-AL)

    dA_pre, dW_L, db_L = backprop(dAL, caches[L], activation = "sigmoid")

    for l in reversed(range(L-1)):

        dA_pre, dW_l, db_l = backprop(dA_pre, caches[l], activation = "relu")

    return [dW, db] of all layers
```

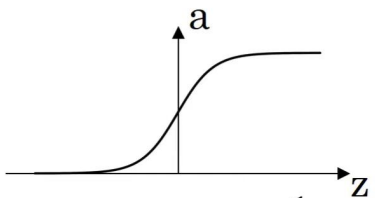
Implement backward propagation

$$dZ^{[l]} = dA^{[l]} * g^{[l]'}(Z^{[l]})$$



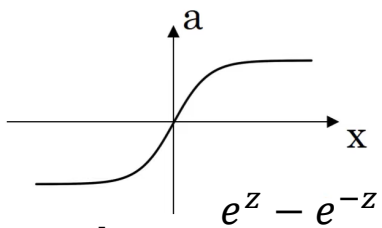
ReLu: $a = \max(0, z)$

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



sigmoid: $a = \dfrac{1}{1+e^{-z}}$

$$g'(z) = g(z)\big(1 - g(z)\big) = a(1-a)$$



tanh: $a = \dfrac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$

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

```
def relu_backward(dA, Z):

    dZ = dA
    dZ[Z <= 0] = 0

    return dZ

def sigmoid_backward(dA, Z):

    A = 1/(1+np.exp(-Z))
    dZ = dA * A * (1-A)

    return dZ

def tanh_backward(dA, Z):

    A = 1/(1+np.exp(-Z))
    dZ = dA * (1-A*A)

    return dZ
```

Implement backward propagation

$$dW^{[l]} = \frac{1}{m} dZ^{[l]} \cdot A^{[l-1]T}$$

$$db^{[l]} = \frac{1}{m} np.sum(dZ^{[l]})$$

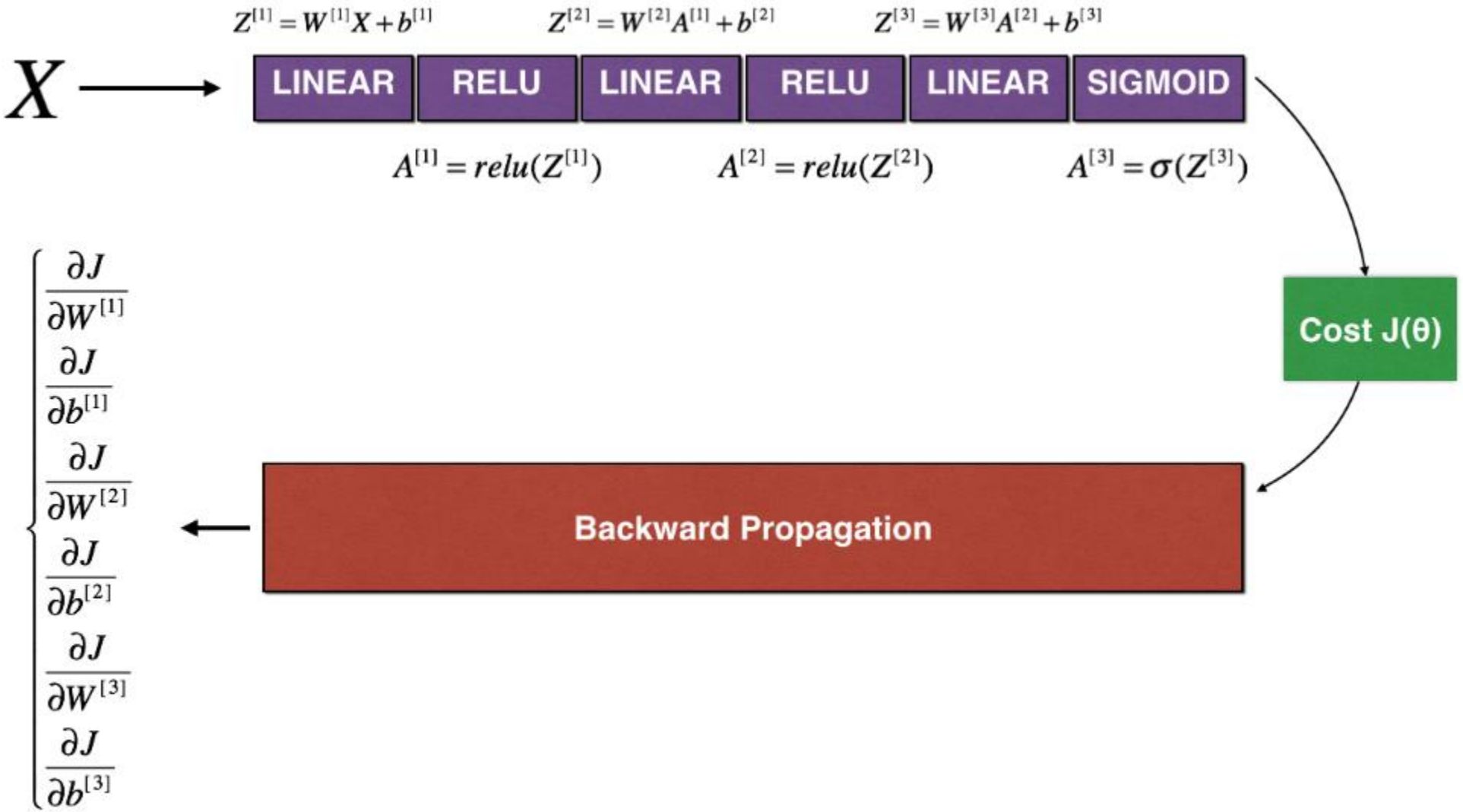
$$dA^{[l-1]} = W^{[l]T} \cdot dZ^{[l]}$$

```
def linear_backward(dZ, A_prev, W):

    m = A_prev.shape[1]


    dW = np.dot(dZ, A_prev.T)/m
    db = dZ.sum(axis = 1, keepdims = True)/m
    dA_prev = np.dot(W.T, dZ)


    return dA_prev, dW, db
```

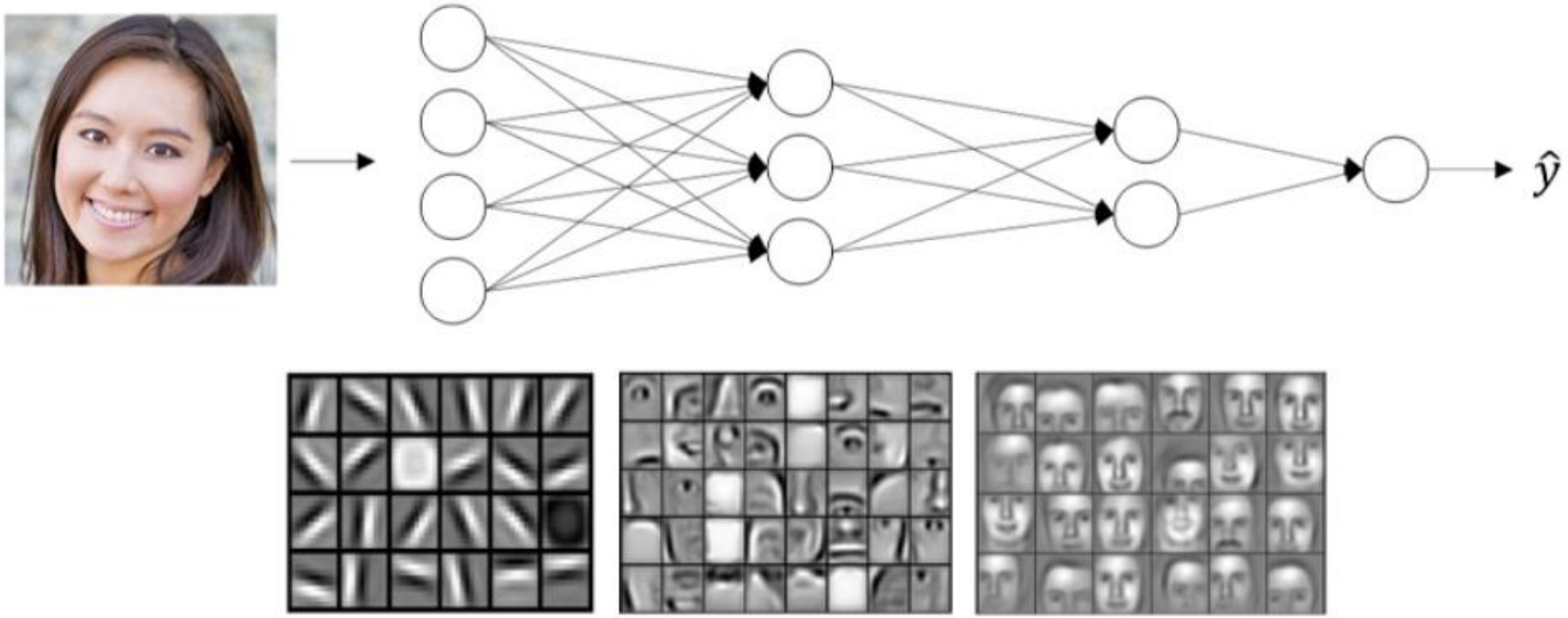
Implementation methodology

1. Initialize parameters / Define hyperparameters
2. Loop for num_iterations:
    a. Forward propagation
    b. Compute cost function
    c. Backward propagation
    d. Update parameters (using parameters, and grads from backprop)
3. Use trained parameters to predict labels

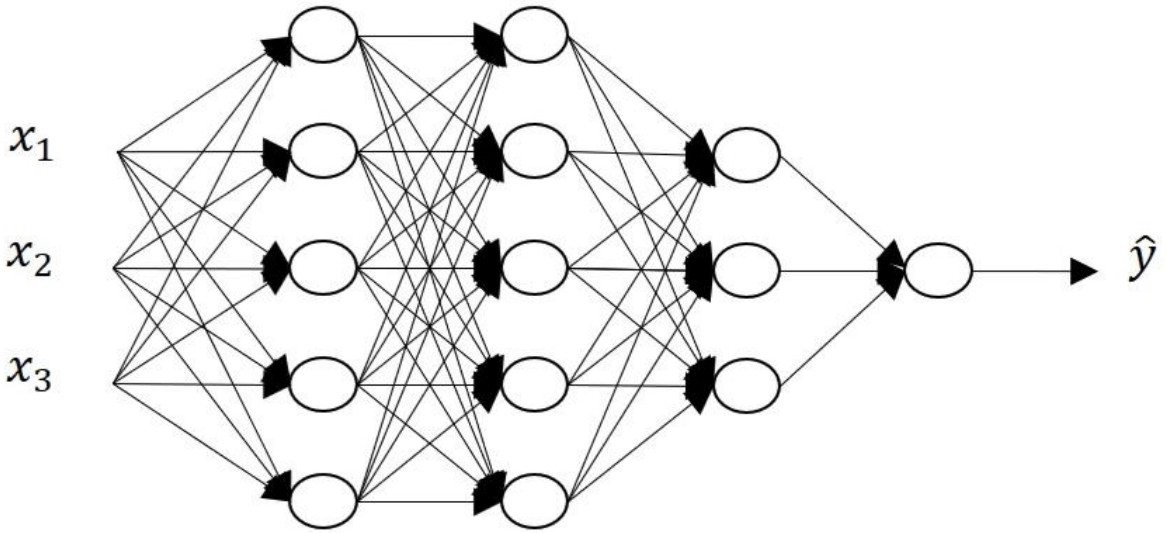
Backward Propagation in a Deep Network

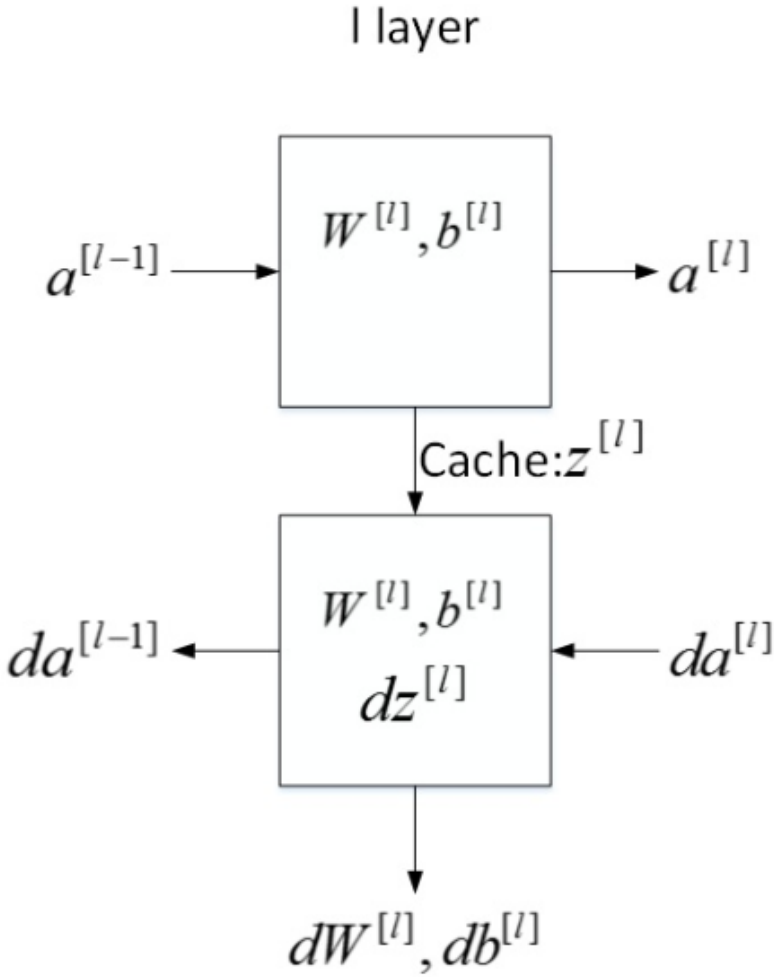


Intuition about deep representation

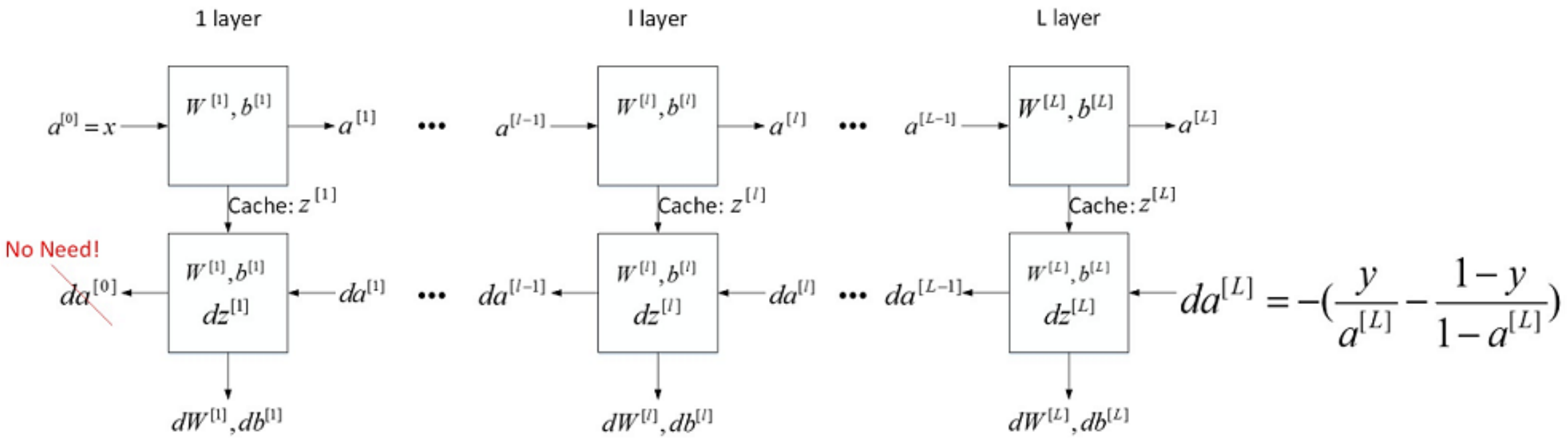


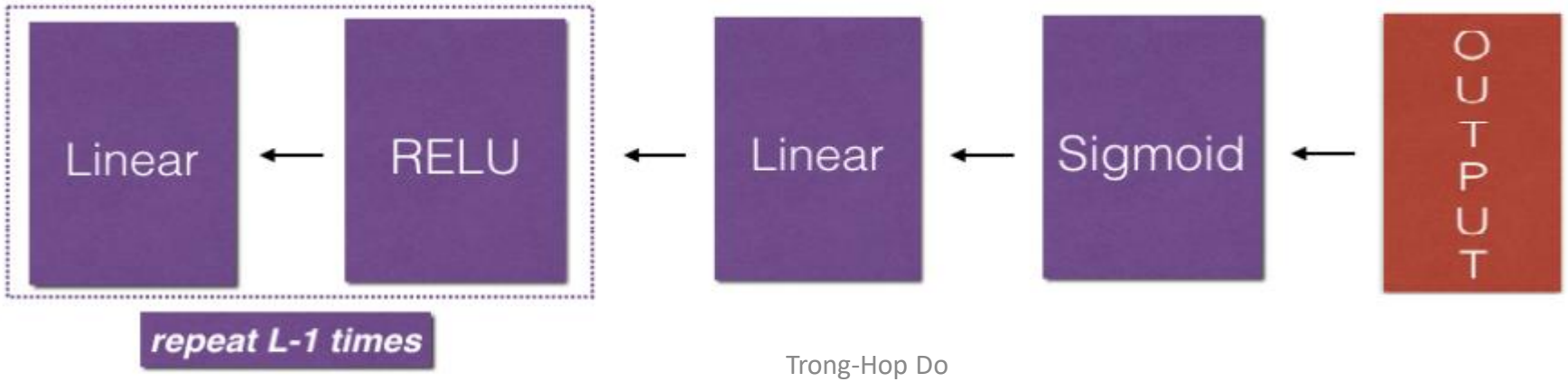
Building blocks of deep neural networks





Building blocks of deep neural networks





Implement cost function

Cross entropy cost function: $J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i}^{m}\left[y^{(i)}\log\hat{y}^{(i)} + \left(1-y^{(i)}\right)\log\left(1-\hat{y}^{(i)}\right)\right]$

```
def compute_cost(AL, Y):

    m = Y.shape[1]

    cost = -np.sum(Y*np.log(AL) + (1 - Y)*np.log(1 - AL))/m

    return cost
```

Future topics

- Introduction to Neural Network and Deep Learning
- Improving Deep Neural Networks
- Convolutional Neural Networks
- Sequence Models
- Deep Learning in data mining and big data analysis
- Other architectures and current research activity in deep learning