

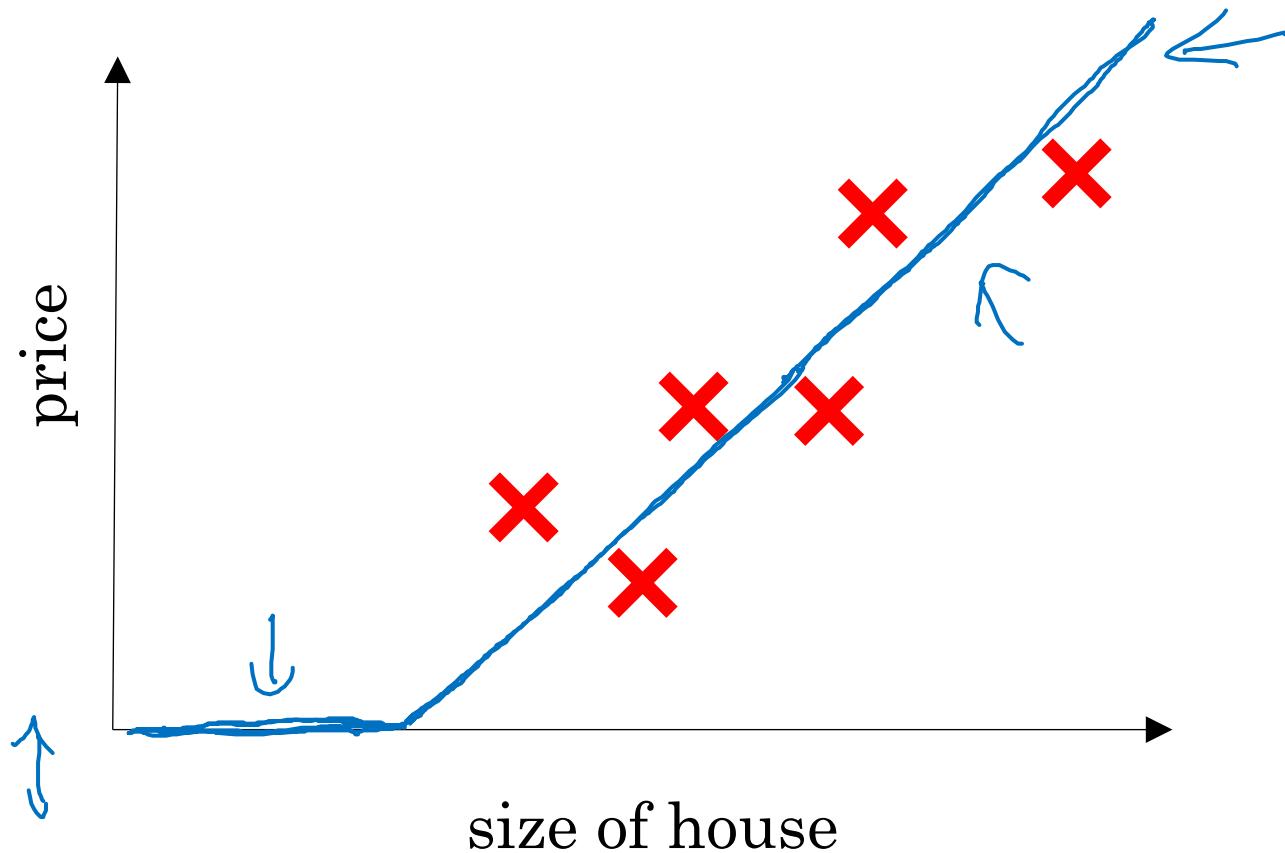


deeplearning.ai

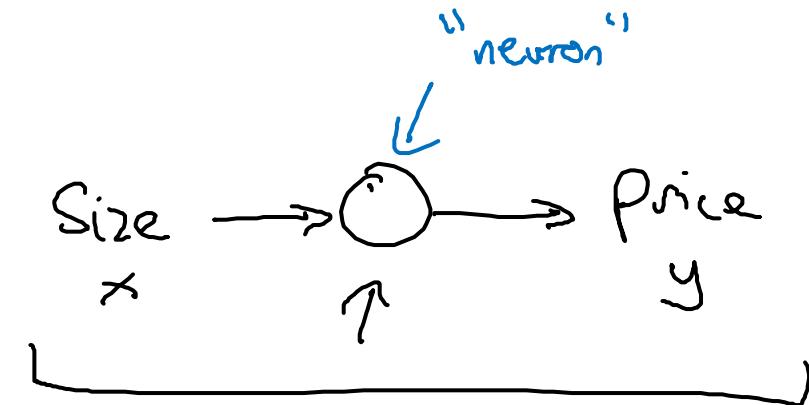
Introduction to Deep Learning

What is a Neural Network?

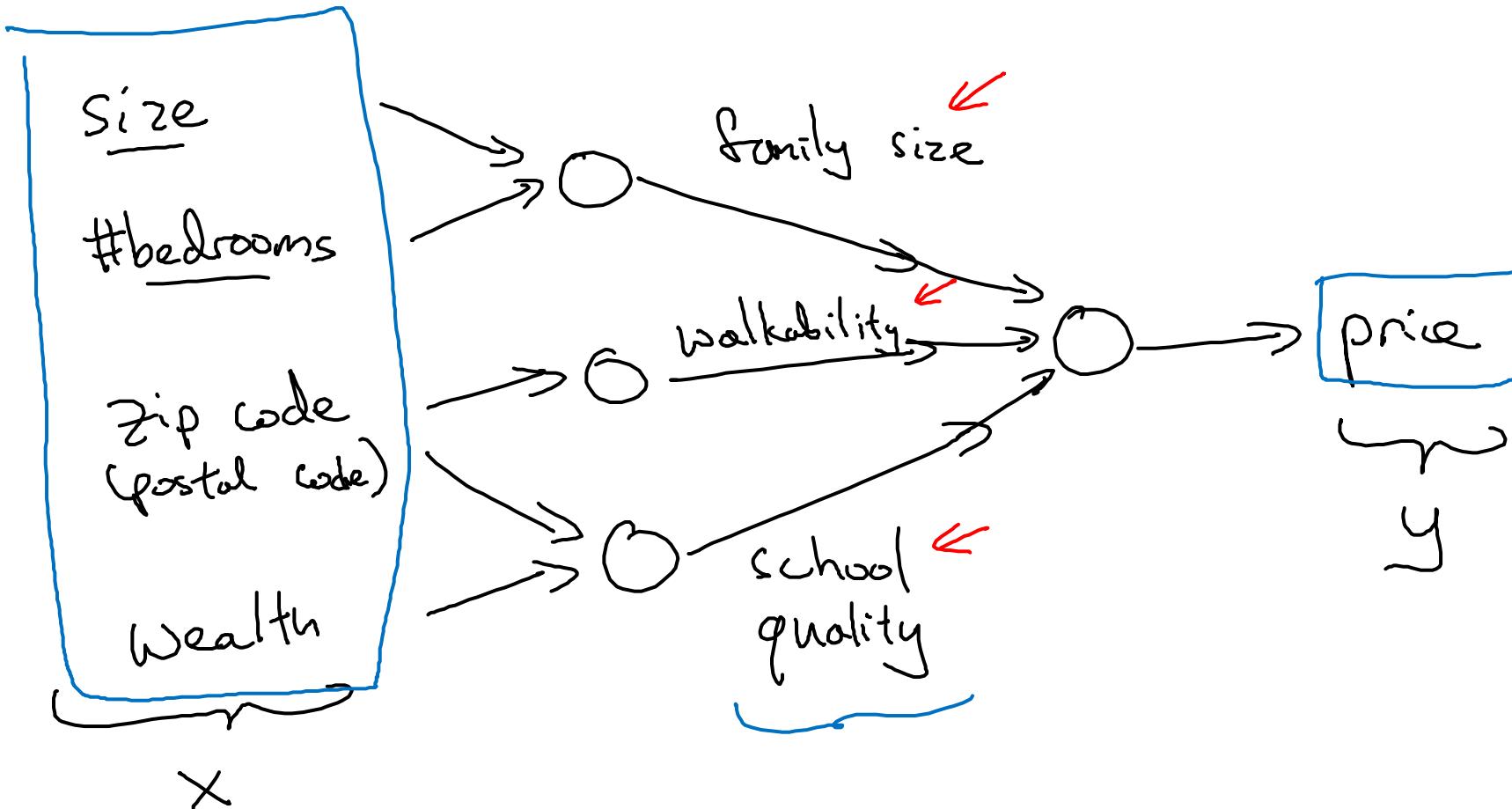
Housing Price Prediction



ReLU
Rectified
Linear
Unit

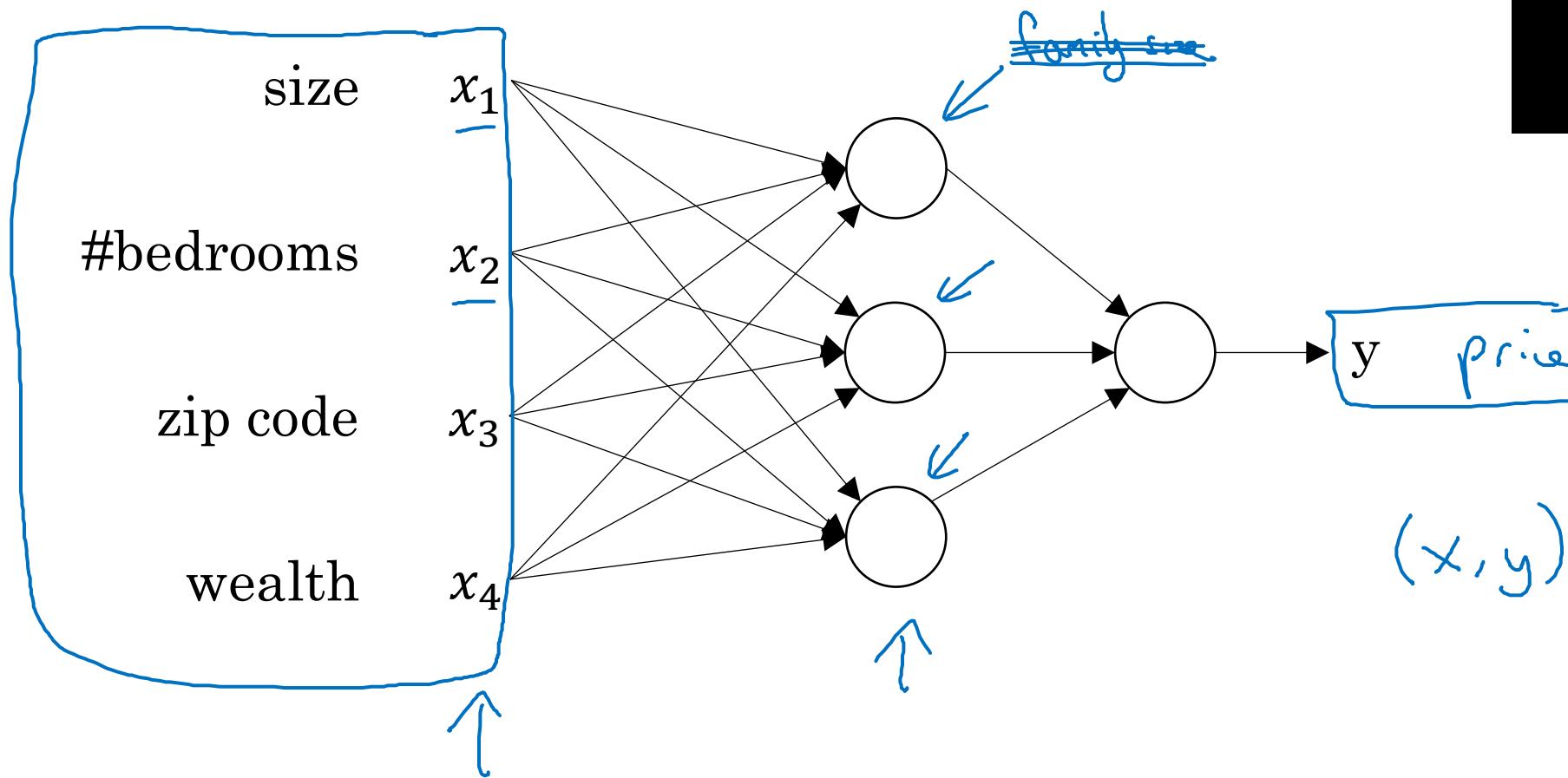


Housing Price Prediction



Housing Price Prediction

Drawing of
previous Image





Introduction to Deep Learning

Supervised Learning with Neural Networks

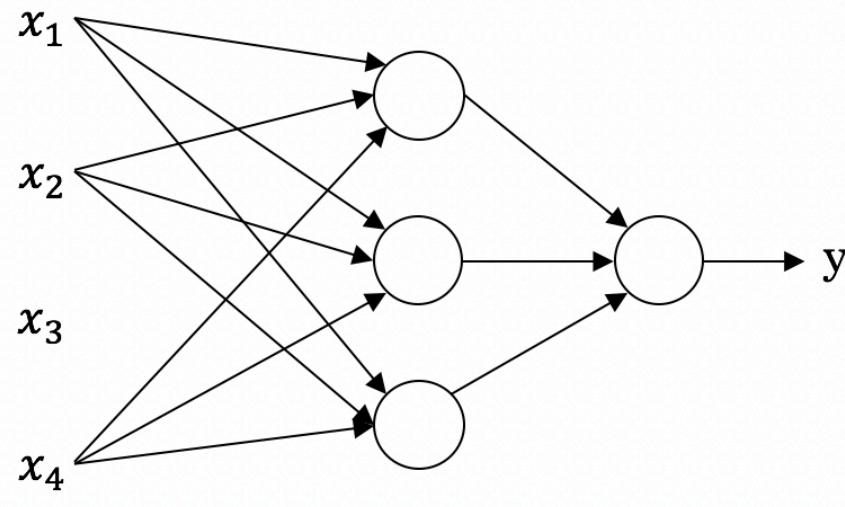
Supervised Learning

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

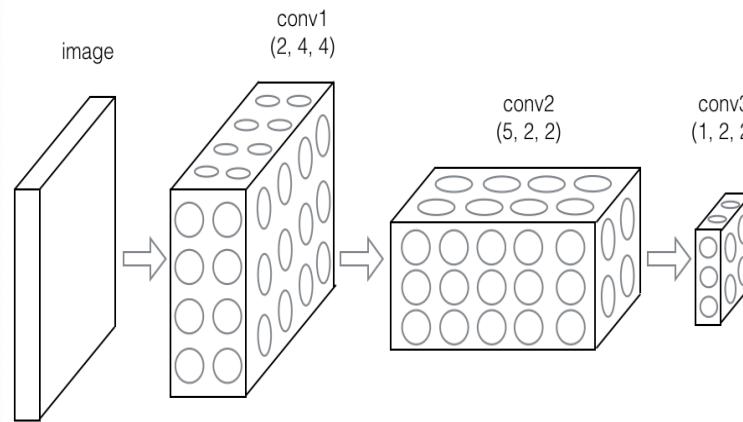
Annotations:

- A blue arrow points from "Input(x)" to the first row.
- A blue arrow points from "Output (y)" to the first row.
- A blue bracket groups "Real Estate" and "Online Advertising" under the heading "Standard NN".
- A blue bracket groups "Photo tagging" and "Speech recognition" under the heading "CNN".
- A blue bracket groups "Machine translation" and "Autonomous driving" under the heading "RNN".
- A blue bracket groups "Autonomous driving" and "Machine translation" under the heading "Custom Hybrid".
- A blue arrow points upwards from the bottom row towards the "Autonomous driving" application.

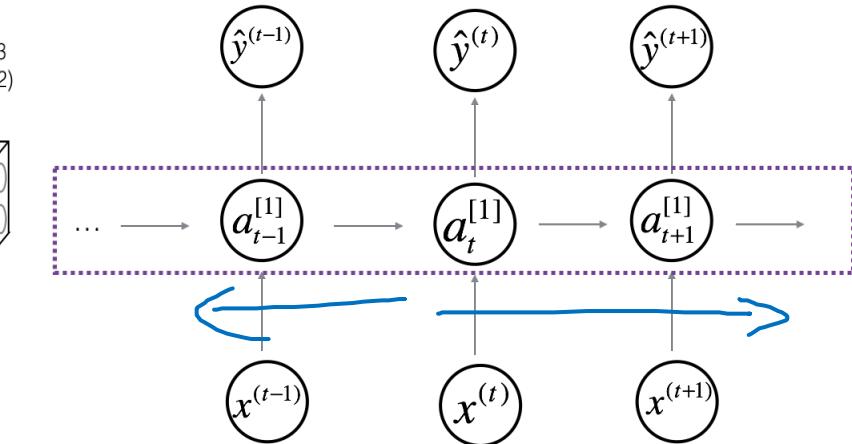
Neural Network examples



Standard NN



Convolutional NN



Recurrent NN

Supervised Learning

Structured Data

Size	#bedrooms	...	Price (1000\$)
2104	3		400
1600	3		330
2400	3		369
:	:		:
3000	4		540

Unstructured Data



Audio

Image

User Age	Ad Id	...	Click
41	93242		1
80	93287		0
18	87312		1
:	:		:
27	71244		1

Four scores and seven years ago...

Text

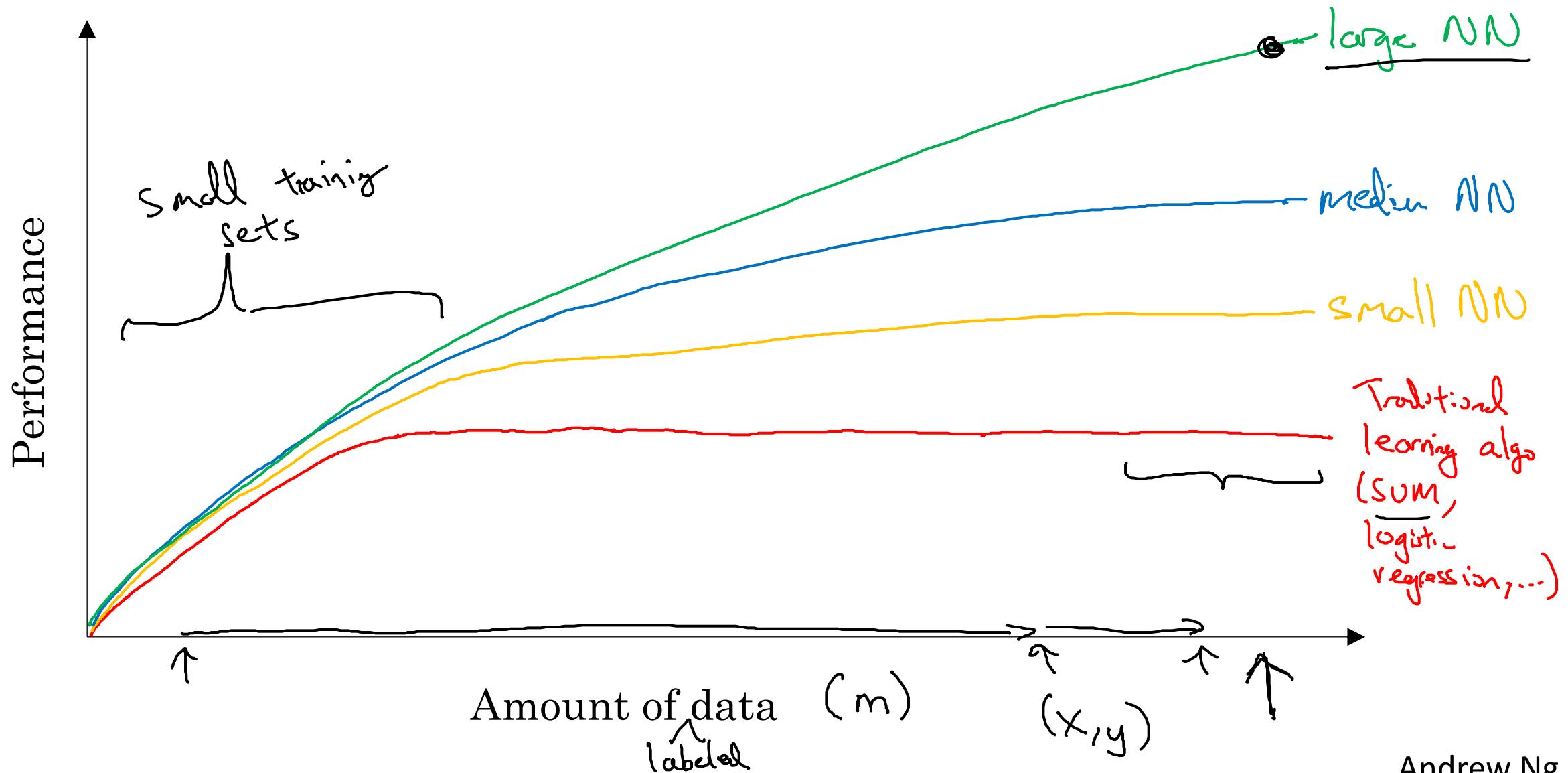


deeplearning.ai

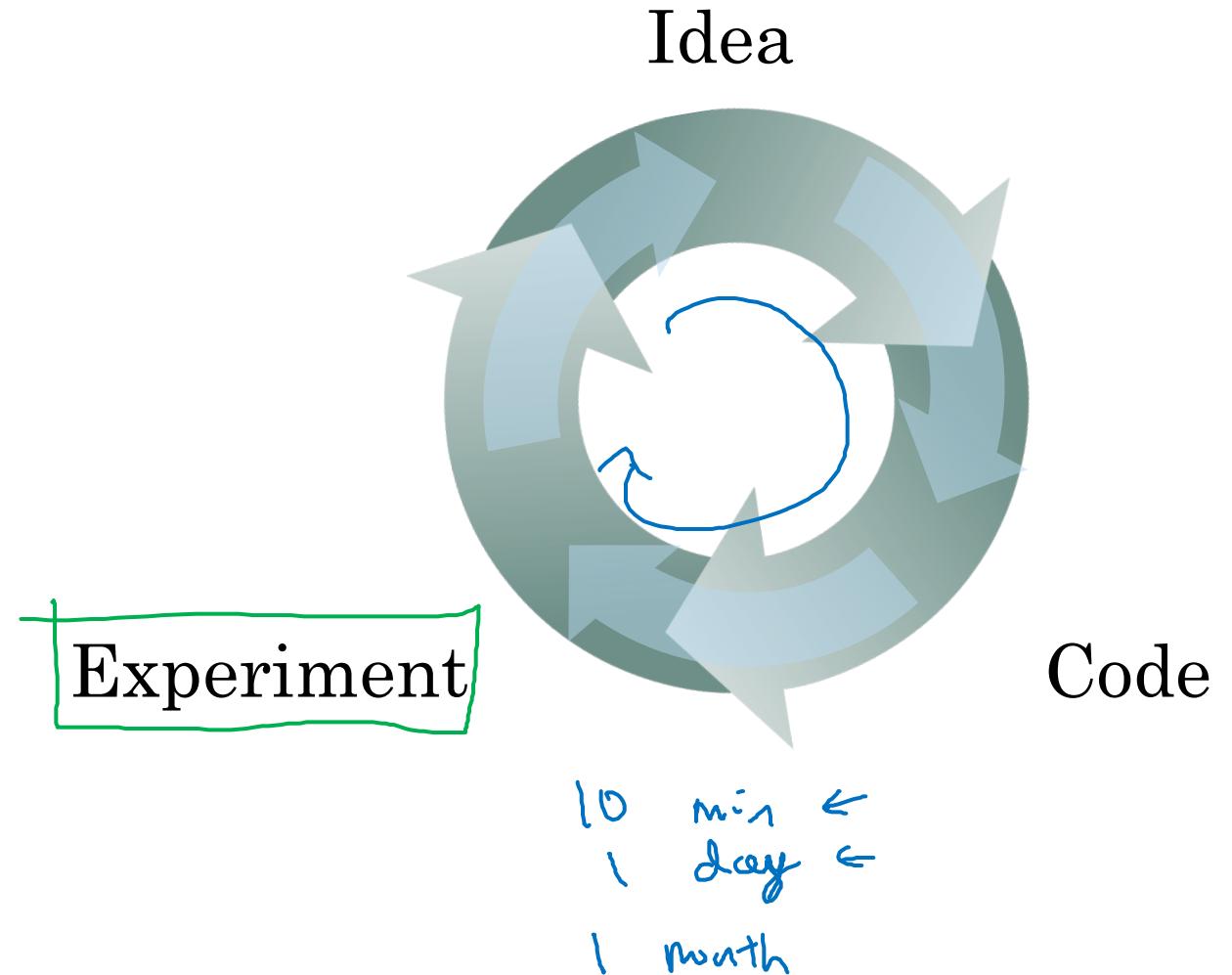
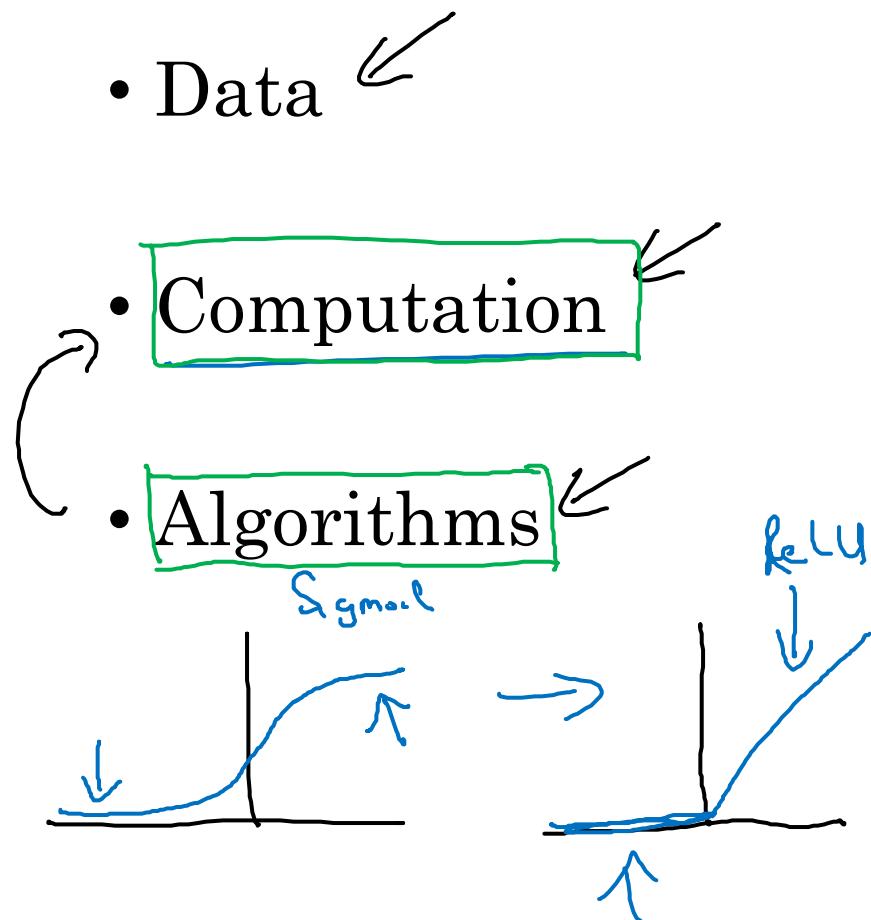
Introduction to Neural Networks

Why is Deep Learning taking off?

Scale drives deep learning progress



Scale drives deep learning progress





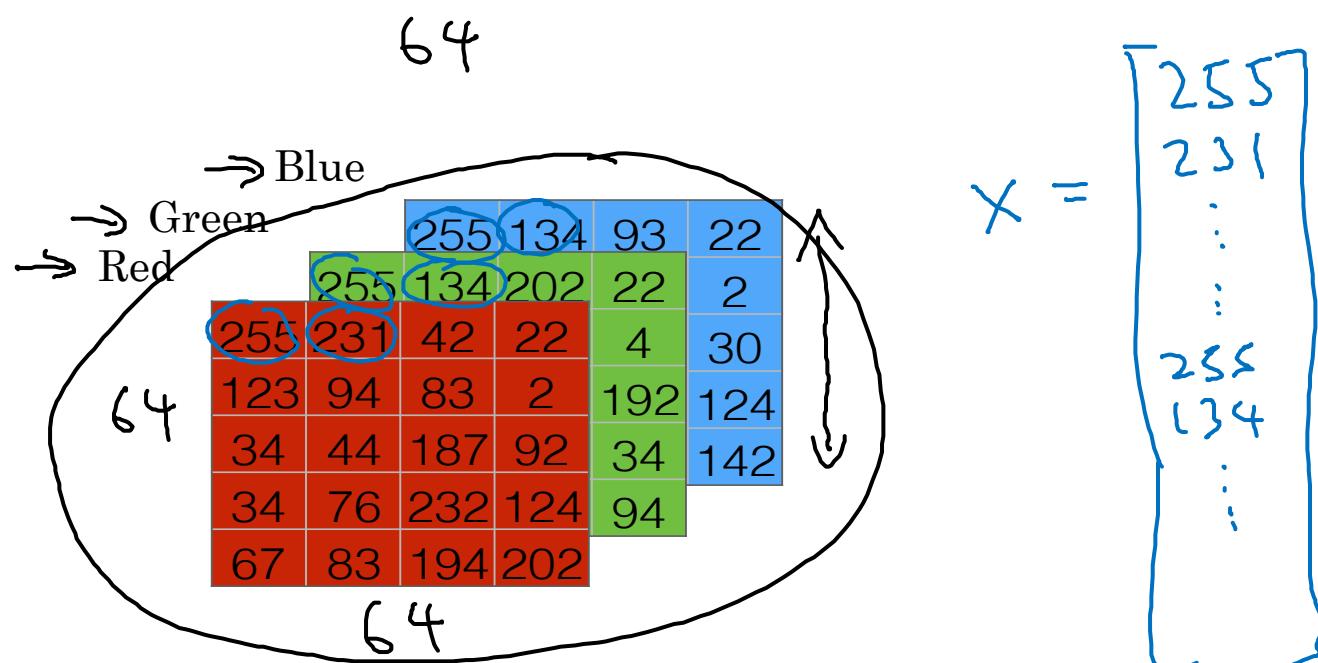
deeplearning.ai

Basics of Neural Network Programming

Binary Classification

Binary Classification

64 →  → 1 (cat) vs 0 (non cat)



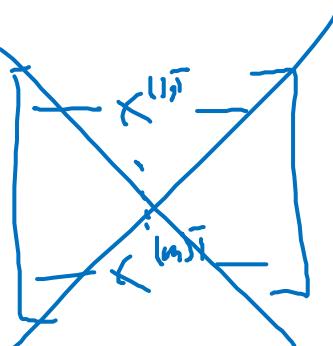
$$64 \times 64 \times 3 = 12288$$

$$n = n_x = 12288$$

$$X \rightarrow y$$

Notation

(x, y) $x \in \mathbb{R}^{n_x}$, $y \in \{0, 1\}$
 m training examples : $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$
 $M = M_{\text{train}}$ $M_{\text{test}} = \# \text{test examples.}$

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

$$Y = [y^{(1)} \ y^{(2)} \ \dots \ y^{(m)}]^{1 \times m}$$
$$Y \in \mathbb{R}^{1 \times m}$$
$$Y.\text{shape} = (1, m)$$

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



deeplearning.ai

Basics of Neural Network Programming

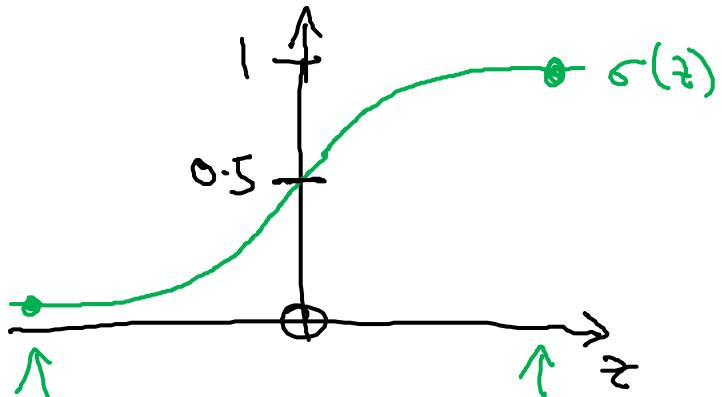
Logistic Regression

Logistic Regression

Given x , want $\hat{y} = P(y=1 | x)$
 $x \in \mathbb{R}^{n_x}$

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

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



$$x_0 = 1, \quad x \in \mathbb{R}^{n_x+1}$$

$$\hat{y} = \sigma(w^T x)$$

$$w = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_{n_x} \end{bmatrix} \quad \left\{ \begin{array}{l} b \leftarrow w_0 \\ w \leftarrow \text{rest} \end{array} \right.$$

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

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

If z large negative number

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



deeplearning.ai

Basics of Neural Network Programming

Logistic Regression cost function

Logistic Regression cost function

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

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

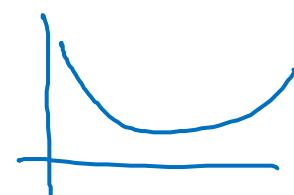
Loss (error) function:

$$L(\hat{y}, y) = \frac{1}{2} (\hat{y} - y)^2$$

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

$x^{(i)}$
 $y^{(i)}$
 $z^{(i)}$

i-th example.



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

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

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



deeplearning.ai

Basics of Neural Network Programming

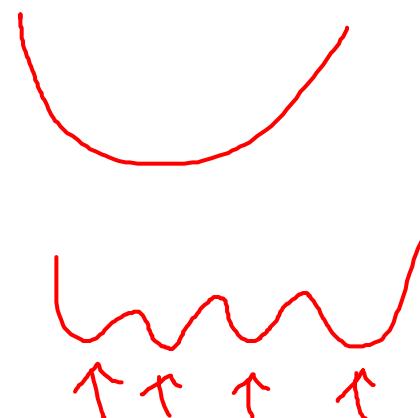
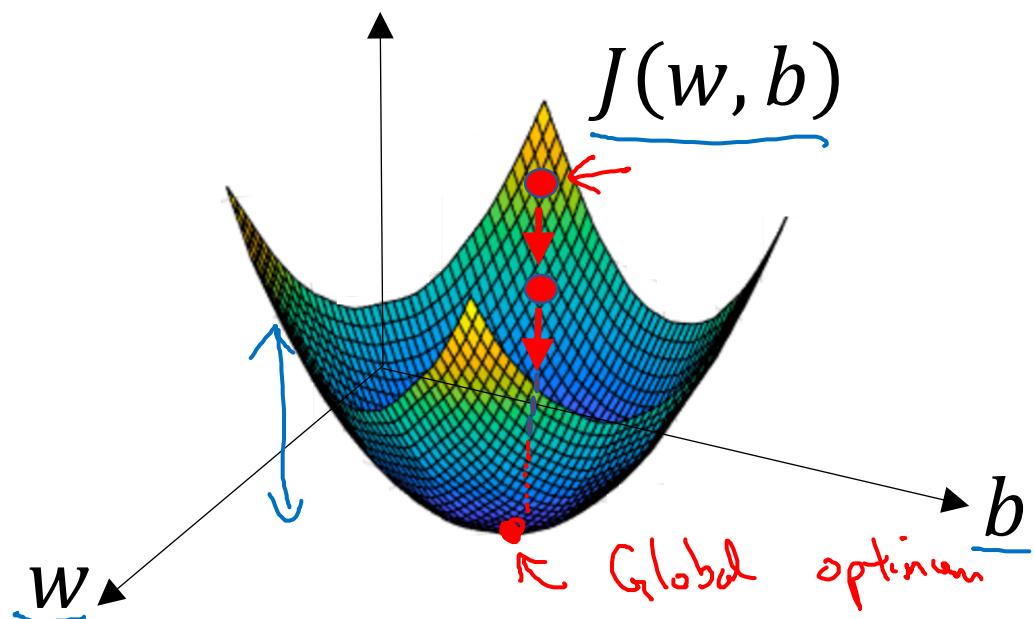
Gradient Descent

Gradient Descent

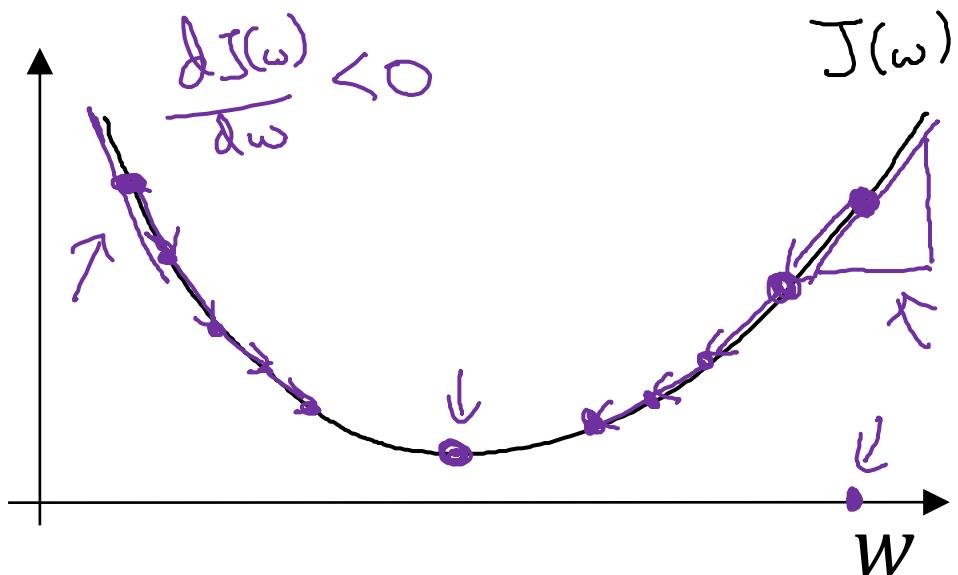
Recap: $\hat{y} = \sigma(w^T x + b)$, $\sigma(z) = \frac{1}{1+e^{-z}}$ 

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

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



Gradient Descent



Repeat {

$$\omega := \omega - \alpha \frac{dJ(\omega)}{d\omega}$$

}

$\omega := \omega - \alpha \frac{dJ(\omega)}{d\omega}$

learning rate

$\frac{dJ(\omega)}{d\omega} = ?$

$$J(\omega, b)$$

$$\omega := \omega - \alpha \frac{dJ(\omega, b)}{d\omega}$$

$$b := b - \alpha \frac{dJ(\omega, b)}{db}$$

$$\frac{dJ(\omega, b)}{d\omega}$$

$$\frac{\partial J(\omega, b)}{\partial \omega}$$

$$\frac{\partial J(\omega, b)}{\partial b}$$

"partial derivative"

J

$$\frac{d\omega}{db}$$

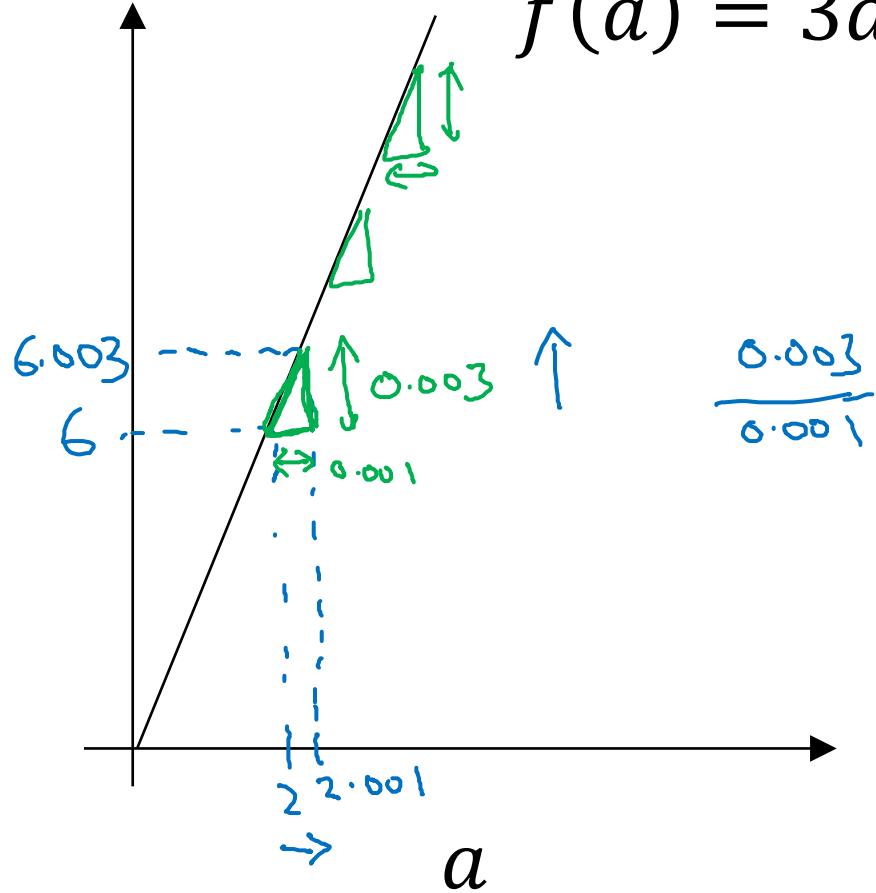


deeplearning.ai

Basics of Neural Network Programming

Derivatives

Intuition about derivatives



$$\frac{0.003}{0.001}$$

height
width

$$\rightarrow a = 2$$

$$a = 2.001$$

$$f(a) = 6$$

$$f(a) = 6.003$$

slope (derivative) of $f(a)$

at $a=2$ is 3

$$\rightarrow a = 5$$

$$a = 5.001$$

$$f(a) = 15$$

$$f(a) = 15.003$$

slope at $a=5$ is also 3

$$\frac{d f(a)}{da} = 3 = \frac{d}{da} f(a)$$

0.001
0.00000001
0.0000000001

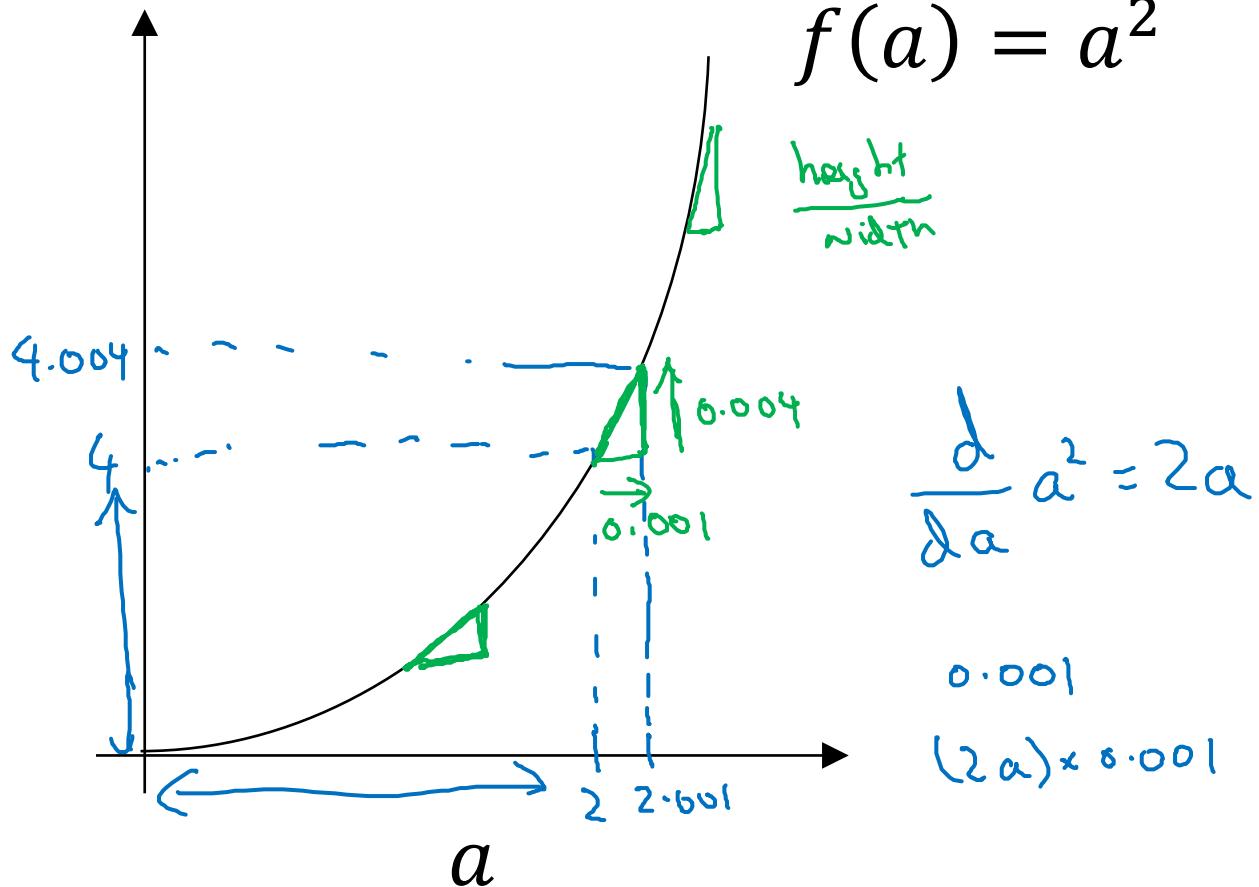


deeplearning.ai

Basics of Neural Network Programming

More derivatives
examples

Intuition about derivatives



$a = 2$ $f(a) = 4$

$a = 2.001$ $f(a) \approx 4.004$

$\frac{(4.004 - 4)}{0.001}$

slope (derivative) of $f(a)$ at $a=2$ is 4.

$\frac{d}{da} f(a) = 4$ when $a=2$.

$a = 5$ $f(a) = 25$

$a = 5.001$ $f(a) \approx 25.010$

$\frac{d}{da} f(a) = 10$ when $a=5$

$\frac{d}{da} f(a) = \frac{d}{da} a^2 = 2a$

More derivative examples

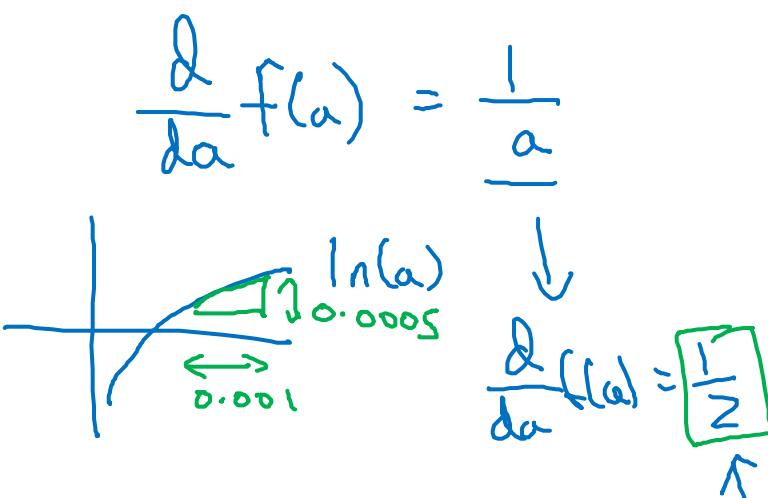
$$f(a) = a^2$$

$$\frac{\partial}{\partial a} f(a) = \underbrace{2a}_{4}$$

$$f(a) = a^3$$

$$\frac{\partial}{\partial a} f(a) = \underbrace{3a^2}_{3 \times 2^2} = 12$$

$$f(a) = \begin{matrix} \log_e(a) \\ \ln(a) \end{matrix}$$



$$a = 2$$

$$a = 2.001$$

$$f(a) = 4$$

$$f(a) \approx 4.004$$

$$a = 2$$

$$a = \underline{2.001}$$

$$f(a) = 8$$

$$f(a) \approx \underline{8.012}$$

$$a = 2$$

$$a = \underline{2.001}$$

$$f(a) \approx 0.69315$$

$$f(a) \approx \underline{0.69365}$$

$$\frac{0.0005}{0.0005}$$



deeplearning.ai

Basics of Neural Network Programming

Computation Graph

Computation Graph

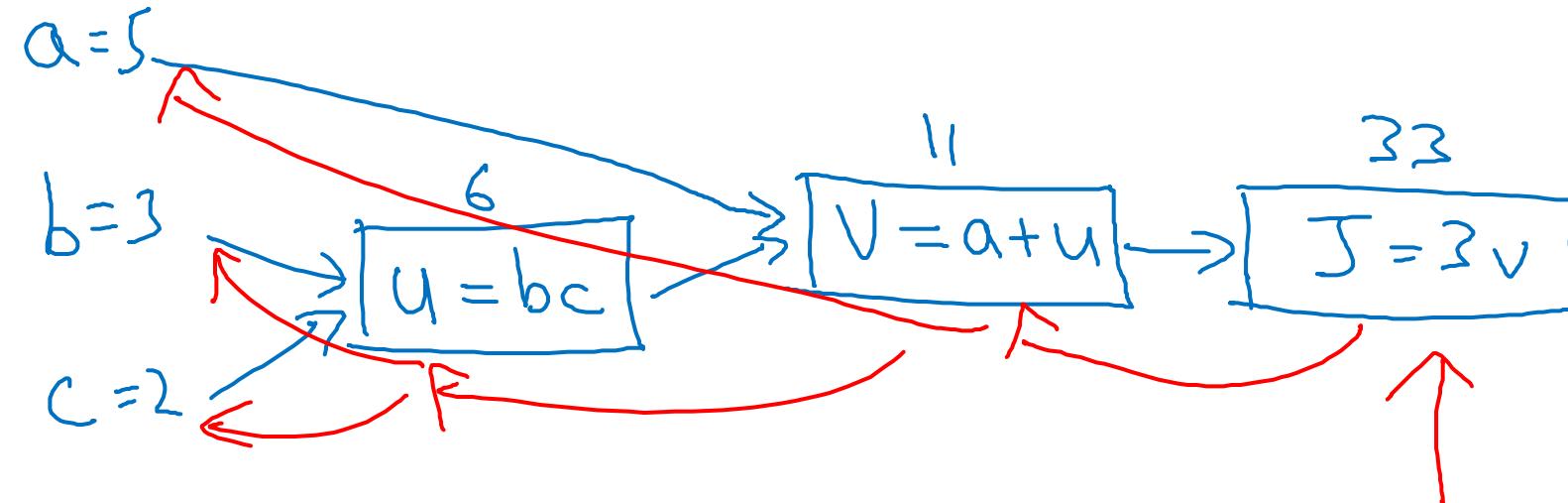
$$J(a, b, c) = 3(u + bc) = 3(5 + 3 \times 2) = 33$$

$\underbrace{u}_{\downarrow}$
 $\underbrace{v}_{\downarrow}$
 $\underbrace{J}_{\downarrow}$

$$u = bc$$

$$v = a + u$$

$$J = 3v$$



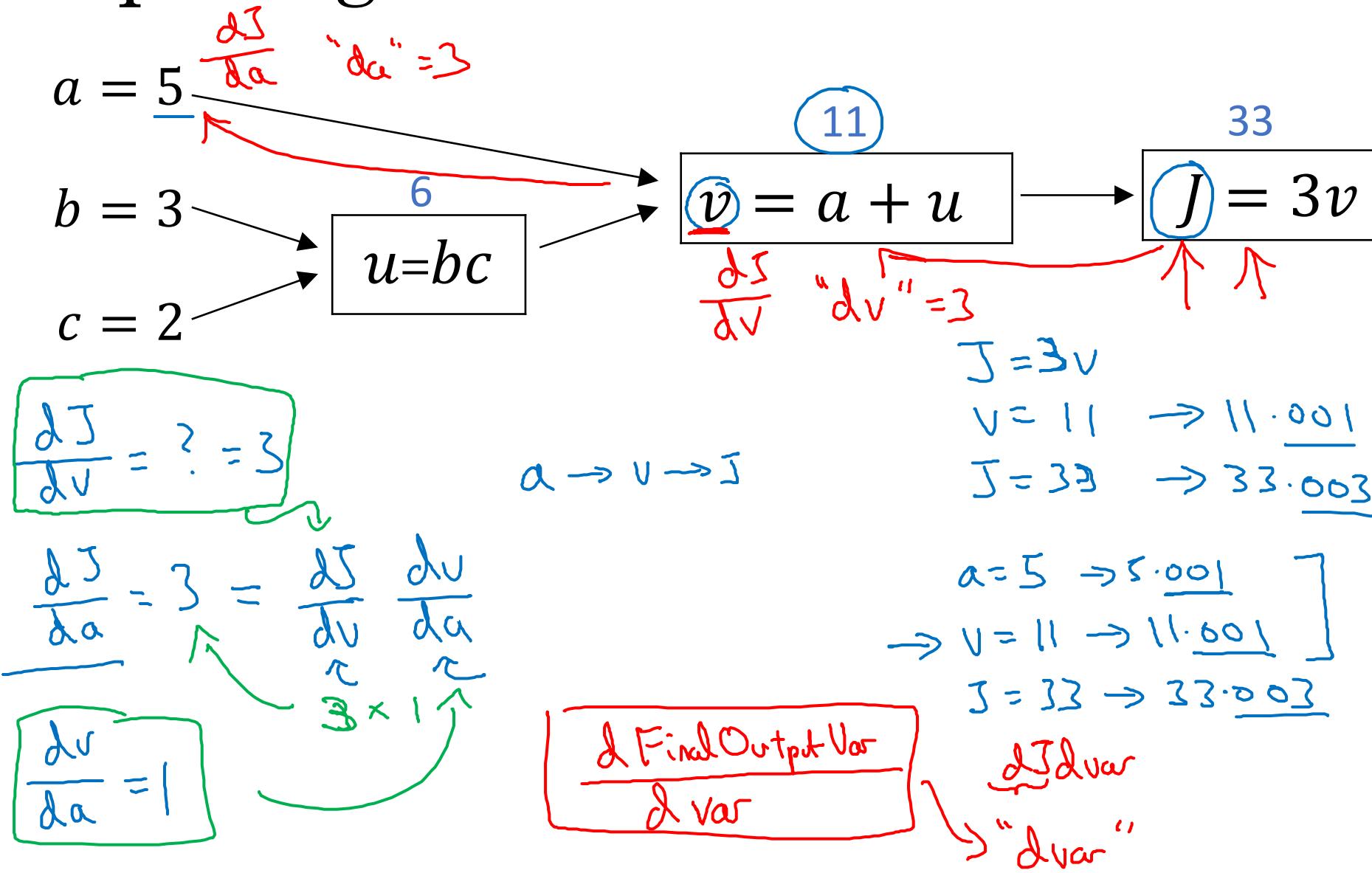


deeplearning.ai

Basics of Neural Network Programming

Derivatives with a Computation Graph

Computing derivatives



$$\begin{aligned}
 f(a) &= 3a \\
 \frac{df(b)}{da} &= \frac{df}{da} = 3 \\
 J &= 3v \\
 \frac{dJ}{dv} &= 3
 \end{aligned}$$

Computing derivatives

Diagram illustrating the computation of derivatives for the function $J = 3v$, where $v = a + u$, $u = bc$, $a = 5$, $b = 3$, and $c = 2$.

Derivatives calculated:

- $\frac{\partial J}{\partial a} \rightarrow \frac{\partial a}{\partial a} = 3$
- $\frac{\partial J}{\partial b} \rightarrow \frac{\partial b}{\partial b} = 6$
- $\frac{\partial J}{\partial c} \rightarrow \frac{\partial c}{\partial c} = 9$
- $\frac{\partial J}{\partial u} = 3 = \frac{\partial J}{\partial v} \cdot \frac{\partial v}{\partial u}$ (with $\frac{\partial J}{\partial v} = 3$ and $\frac{\partial v}{\partial u} = 1$)
- $\frac{\partial J}{\partial b} = 6 = \frac{\partial J}{\partial u} \cdot \frac{\partial u}{\partial b}$ (with $\frac{\partial J}{\partial u} = 3$ and $\frac{\partial u}{\partial b} = 2$)
- $\frac{\partial J}{\partial a} = 9 = \frac{\partial J}{\partial u} \cdot \frac{\partial u}{\partial a}$ (with $\frac{\partial J}{\partial u} = 3$ and $\frac{\partial u}{\partial a} = 3$)

Intermediate values and final results:

- $u = 6$
- $v = 11$
- $J = 33$
- $u = 6 \rightarrow 6.001$
- $v = 11 \rightarrow 11.001$
- $J = 33 \rightarrow 33.003$
- $b = 3 \rightarrow 3.001$
- $u = b \cdot c = 6 \rightarrow 6.002$
- $c = 2.006$
- $v = 11.002$
- $J = 33.006$



deeplearning.ai

Basics of Neural Network Programming

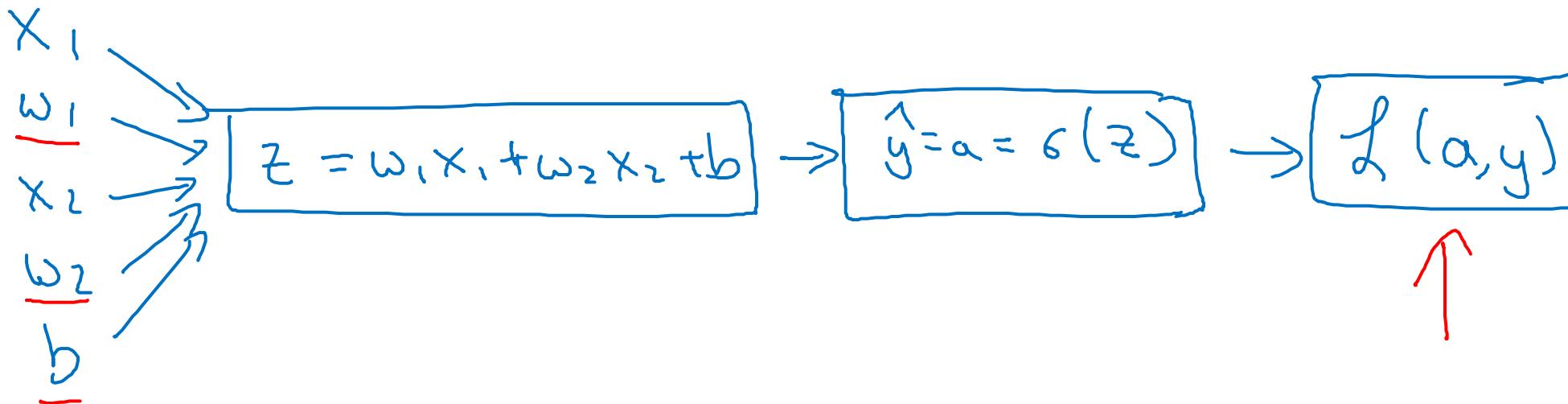
Logistic Regression Gradient descent

Logistic regression recap

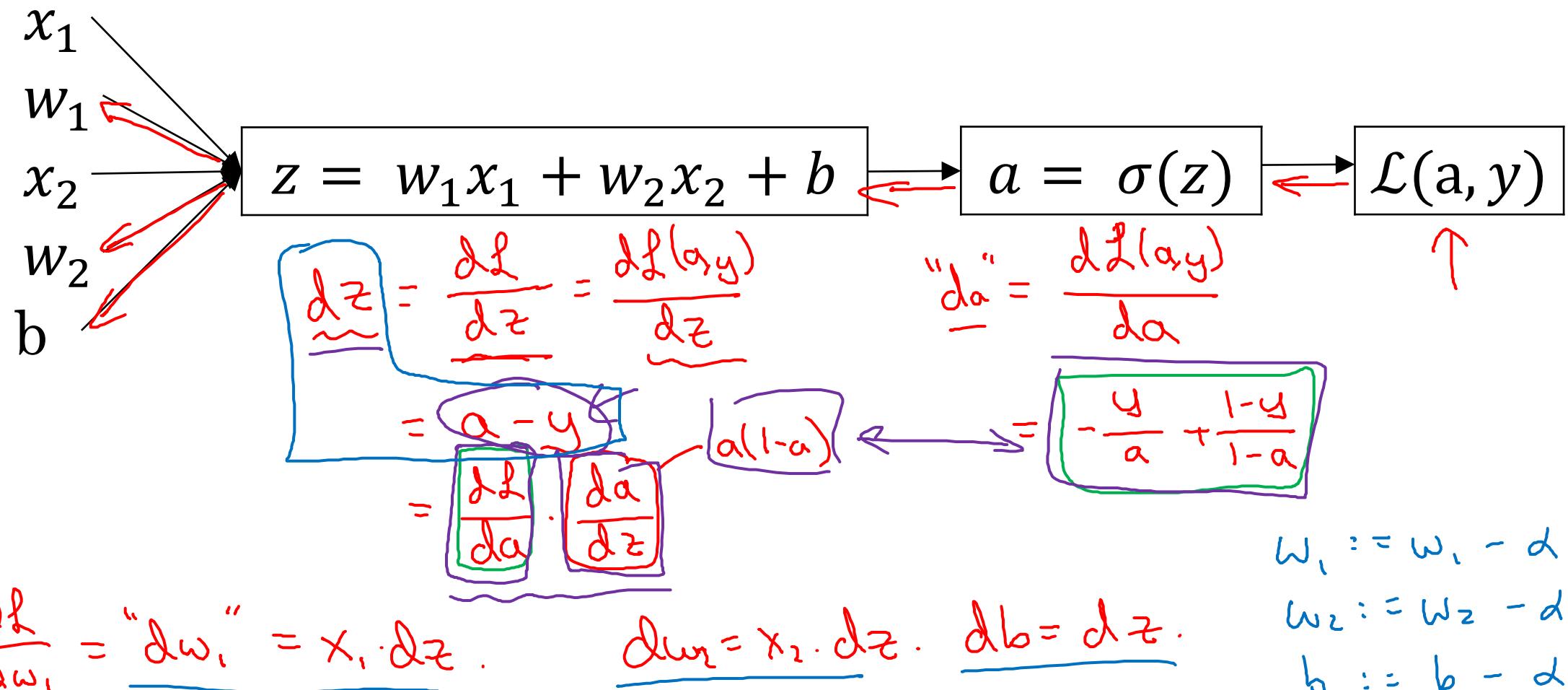
$$\rightarrow z = w^T x + b$$

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

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



Logistic regression derivatives





deeplearning.ai

Basics of Neural Network Programming

Gradient descent
on m examples

Logistic regression on m examples

$$\underline{J(w,b)} = \frac{1}{m} \sum_{i=1}^m l(a^{(i)}, y^{(i)}) \quad (x^{(i)}, y^{(i)})$$
$$\Rightarrow a^{(i)} = \hat{y}^{(i)} = g(z^{(i)}) = g(w^\top x^{(i)} + b) \quad \underline{dw_1^{(i)}}, \underline{dw_2^{(i)}}, \underline{db^{(i)}}$$

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

Logistic regression on m examples

$$J=0; \underline{\Delta w_1}=0; \underline{\Delta w_2}=0; \underline{\Delta b}=0$$

→ For $i = 1$ to m

$$z^{(i)} = \omega^\top x^{(i)} + b$$

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

$$J_t = -[y^{(i)} \log \alpha^{(i)} + (1-y^{(i)}) \log (1-\alpha^{(i)})]$$

$$\underline{\Delta z^{(i)}} = \alpha^{(i)} - y^{(i)}$$

$$\begin{aligned} \Delta w_1 &+= x_1^{(i)} \Delta z^{(i)} \\ \Delta w_2 &+= x_2^{(i)} \Delta z^{(i)} \end{aligned}$$

$$\begin{aligned} \Delta w_3 & \\ \vdots & \\ \Delta w_n & \Delta b += \Delta z^{(i)} \end{aligned}$$

$$J / m \leftarrow$$

$$\Delta w_1 / m; \Delta w_2 / m; \Delta b / m. \leftarrow$$

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

$$w_1 := w_1 - \alpha \underline{\Delta w_1}$$

$$w_2 := w_2 - \alpha \underline{\Delta w_2}$$

$$b := b - \alpha \underline{\Delta b}.$$

Vectorization



deeplearning.ai

Basics of Neural Network Programming

Vectorization

What is vectorization?

$$z = \underbrace{\omega^T x}_{\text{Non-vectorized}} + b$$

Non-vectorized:

$$z = 0$$

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

$$z += b$$

$$\omega = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix} \quad x = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix}$$

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

Vectorized

$$z = \underbrace{\text{np.dot}(\omega, x)}_{w^T x} + b$$

\rightarrow GPU } SIMD - single instruction
 \rightarrow CPU } multiple data.



deeplearning.ai

Basics of Neural Network Programming

More vectorization examples

Neural network programming guideline

Whenever possible, avoid explicit for-loops.

Neural network programming guideline

Whenever possible, avoid explicit for-loops.

$$u = Av$$

$$u_i = \sum_j A_{ij} v_j$$

$u = np.zeros((n,))$

for i ...

 for j ...

$u[i] += A[:, i] * v[j]$

$$u = np.dot(A, v)$$

Vectors and matrix valued functions

Say you need to apply the exponential operation on every element of a matrix/vector.

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

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

```
import numpy as np  
u = np.exp(v) ←  
↑  
np.log(v)  
np.abs(v)  
np.maximum(v, 0)  
v**2  
v/v
```

Logistic regression derivatives

$$J = 0, \boxed{dw_1 = 0, dw_2 = 0}, db = 0$$

$$d\omega = np.zeros((n_x, 1))$$

for i = 1 to n:

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

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

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

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

for j=1..n^x
dw_j += $\sum_{i=1}^m x_j^{(i)} dz^{(i)}$
db += $\sum_{i=1}^m dz^{(i)}$

$$d\omega += x^{(i)} dz^{(i)}$$

$$J = J/m, \boxed{dw_1 = dw_1/m, dw_2 = dw_2/m, db = db/m}$$

$$d\omega /= m.$$



deeplearning.ai

Basics of Neural Network Programming

Vectorizing Logistic Regression

Vectorizing Logistic Regression

$$\begin{array}{l}
 \boxed{\begin{aligned} z^{(1)} &= w^T x^{(1)} + b \\ a^{(1)} &= \sigma(z^{(1)}) \end{aligned}} \quad \boxed{\begin{aligned} z^{(2)} &= w^T x^{(2)} + b \\ a^{(2)} &= \sigma(z^{(2)}) \end{aligned}} \quad \boxed{\begin{aligned} z^{(3)} &= w^T x^{(3)} + b \\ a^{(3)} &= \sigma(z^{(3)}) \end{aligned}}
 \end{array}$$

$X = \begin{bmatrix} x^{(1)} & x^{(2)} & \dots & x^{(m)} \end{bmatrix}$
 $\in \mathbb{R}^{n_x \times m}$
 $\omega^T \begin{bmatrix} x^{(1)} & x^{(2)} & \dots & x^{(m)} \end{bmatrix}$

$$\begin{aligned}
 \bar{z} &= \begin{bmatrix} z^{(1)} & z^{(2)} & \dots & z^{(m)} \end{bmatrix} = w^T X + \underbrace{\begin{bmatrix} b & b & \dots & b \end{bmatrix}}_{1 \times m} = \begin{bmatrix} w^T x^{(1)} + b \\ w^T x^{(2)} + b \\ \vdots \\ w^T x^{(m)} + b \end{bmatrix} \\
 &\rightarrow \bar{z} = \text{np.dot}(\omega^T, X) + \underbrace{b}_{\in \mathbb{R}^{(1,1)}} \quad \text{"Broadcasting"}
 \end{aligned}$$

$$A = \begin{bmatrix} a^{(1)} & a^{(2)} & \dots & a^{(m)} \end{bmatrix} = \sigma(\bar{z})$$



deeplearning.ai

Basics of Neural Network Programming

Vectorizing Logistic Regression's Gradient Computation

Vectorizing Logistic Regression

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

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

$$dz = \begin{bmatrix} dz^{(1)} & dz^{(2)} & \dots & dz^{(m)} \end{bmatrix}^T$$

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

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

$$\begin{aligned} \rightarrow dw &= 0 \\ dw + &= \frac{x^{(1)} dz^{(1)}}{} \\ dw + &= \frac{x^{(2)} dz^{(2)}}{} \\ &\vdots \\ dw &= m \end{aligned}$$

$$\begin{aligned} db &= 0 \\ db + &= dz^{(1)} \\ db + &= dz^{(2)} \\ &\vdots \\ db + &= dz^{(m)} \\ db &= m \end{aligned}$$

$$\begin{aligned} db &= \frac{1}{m} \sum_{i=1}^m dz^{(i)} \\ &= \frac{1}{m} \text{np.sum}(dz) \end{aligned}$$

$$\begin{aligned} dw &= \frac{1}{m} X dz^T \\ &= \frac{1}{m} \begin{bmatrix} x^{(1)} & \dots & x^{(m)} \end{bmatrix} \begin{bmatrix} dz^{(1)} \\ \vdots \\ dz^{(m)} \end{bmatrix} \\ &= \frac{1}{m} \left[\underline{x^{(1)} dz^{(1)}} + \dots + \underline{x^{(m)} dz^{(m)}} \right] \\ &\qquad\qquad\qquad n \times 1 \end{aligned}$$

Implementing Logistic Regression

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

for i = 1 to m:

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

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

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

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

$$\left. \begin{array}{l} dw_1 += x_1^{(i)} dz^{(i)} \\ dw_2 += x_2^{(i)} dz^{(i)} \end{array} \right\} dw += X^{(i)} * dz^{(i)}$$

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

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

$$db = db/m$$

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



deeplearning.ai

Basics of Neural Network Programming

Broadcasting in Python

Broadcasting example

Calories from Carbs, Proteins, Fats in 100g of different foods:

	Apples	Beef	Eggs	Potatoes
Carb	56.0	0.0	4.4	68.0
Protein	1.2	104.0	52.0	8.0
Fat	1.8	135.0	99.0	0.9

$\downarrow \quad \downarrow \quad \downarrow \quad \downarrow$

$A = \begin{bmatrix} 56.0 & 0.0 & 4.4 & 68.0 \\ 1.2 & 104.0 & 52.0 & 8.0 \\ 1.8 & 135.0 & 99.0 & 0.9 \end{bmatrix}_{(3,4)}$

\downarrow^0

$\xrightarrow{1}$

59cal

$\frac{56}{59} \approx 94.9\%$

Calculate % of calories from Carb, Protein, Fat. Can you do this without explicit for-loop?

`cal = A.sum(axis = 0)`

`percentage = 100*A/(cal.reshape(1,4))`

$\uparrow (3,4) / (1,4)$

Broadcasting example

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + \begin{bmatrix} 100 \\ 100 \\ 100 \\ 100 \end{bmatrix} \xrightarrow{100}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 100 & 200 & 300 \\ 100 & 200 & 300 \end{bmatrix} \xleftarrow{(m,n) \quad (2,3)} \xrightarrow{(1,n) \rightsquigarrow (m,n) \quad (2,3)}$$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} + \begin{bmatrix} 100 & 100 & 100 \\ 200 & 200 & 200 \end{bmatrix} = \xleftarrow{(m,n)}$$

General Principle

$$\begin{array}{ccc} (m, n) & \xrightarrow{\quad \pm \quad} & (1, n) \rightsquigarrow (m, n) \\ \underline{\text{matrix}} & \cancel{*} & (m, 1) \rightsquigarrow (m, n) \end{array}$$

$$\begin{array}{ccccc} (m, 1) & + & \mathbb{R} & & \\ \left[\begin{smallmatrix} 1 \\ 2 \\ 3 \end{smallmatrix} \right] & + & 100 & = & \left[\begin{smallmatrix} 101 \\ 102 \\ 103 \end{smallmatrix} \right] \\ [1 \ 2 \ 3] & + & 100 & = & [101 \ 102 \ 103] \end{array}$$

Matlab/Octave: bsxfun



deeplearning.ai

Basics of Neural Network Programming

A note on python/
numpy vectors

Python Demo

Python / numpy vectors

```
import numpy as np  
  
a = np.random.randn(5)  
  
a = np.random.randn( (5,1) )  
  
a = np.random.randn( (1,5) )  
  
assert(a.shape = (5,1))
```

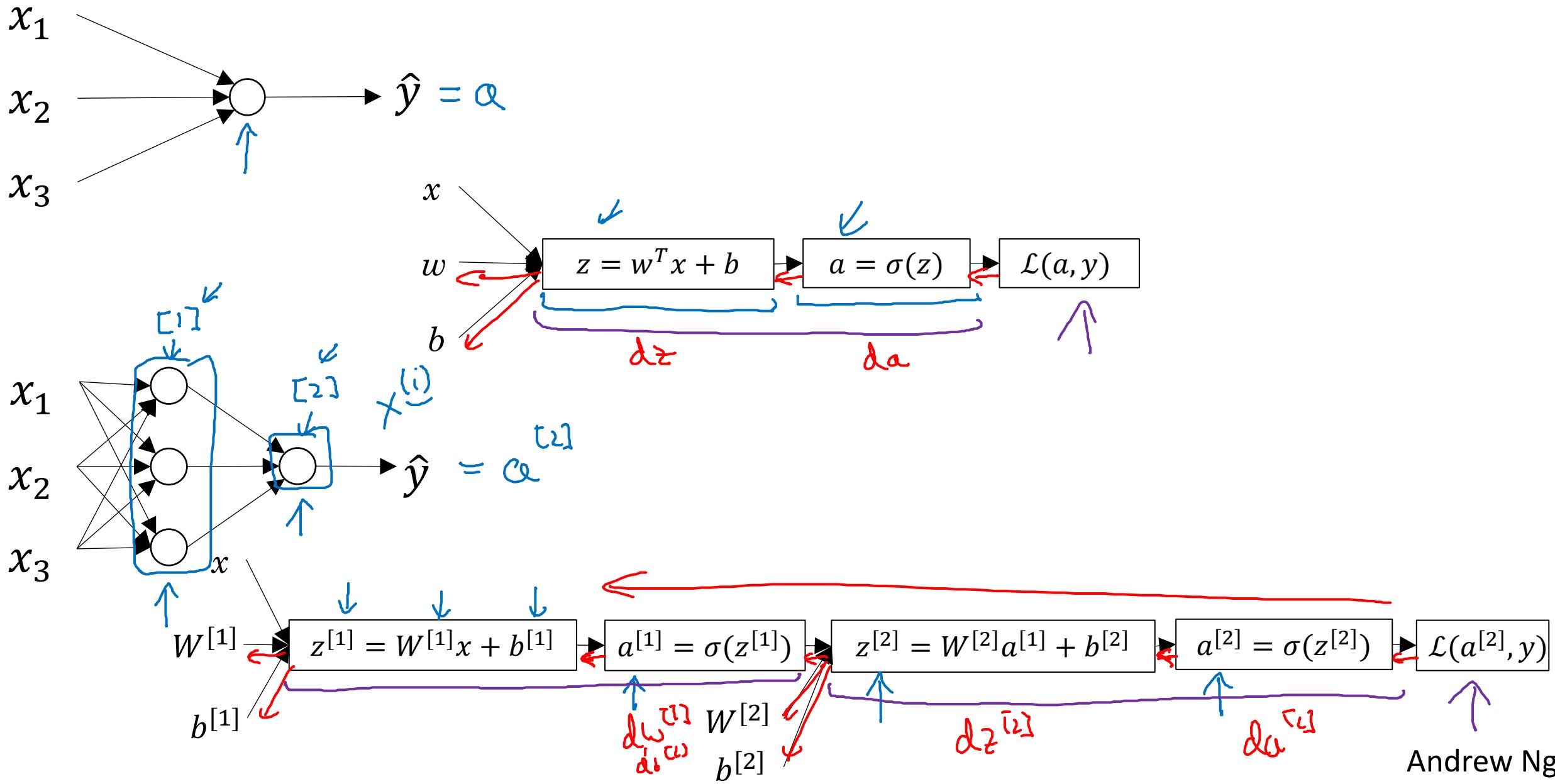


deeplearning.ai

One hidden layer
Neural Network

Neural Networks
Overview

What is a Neural Network?



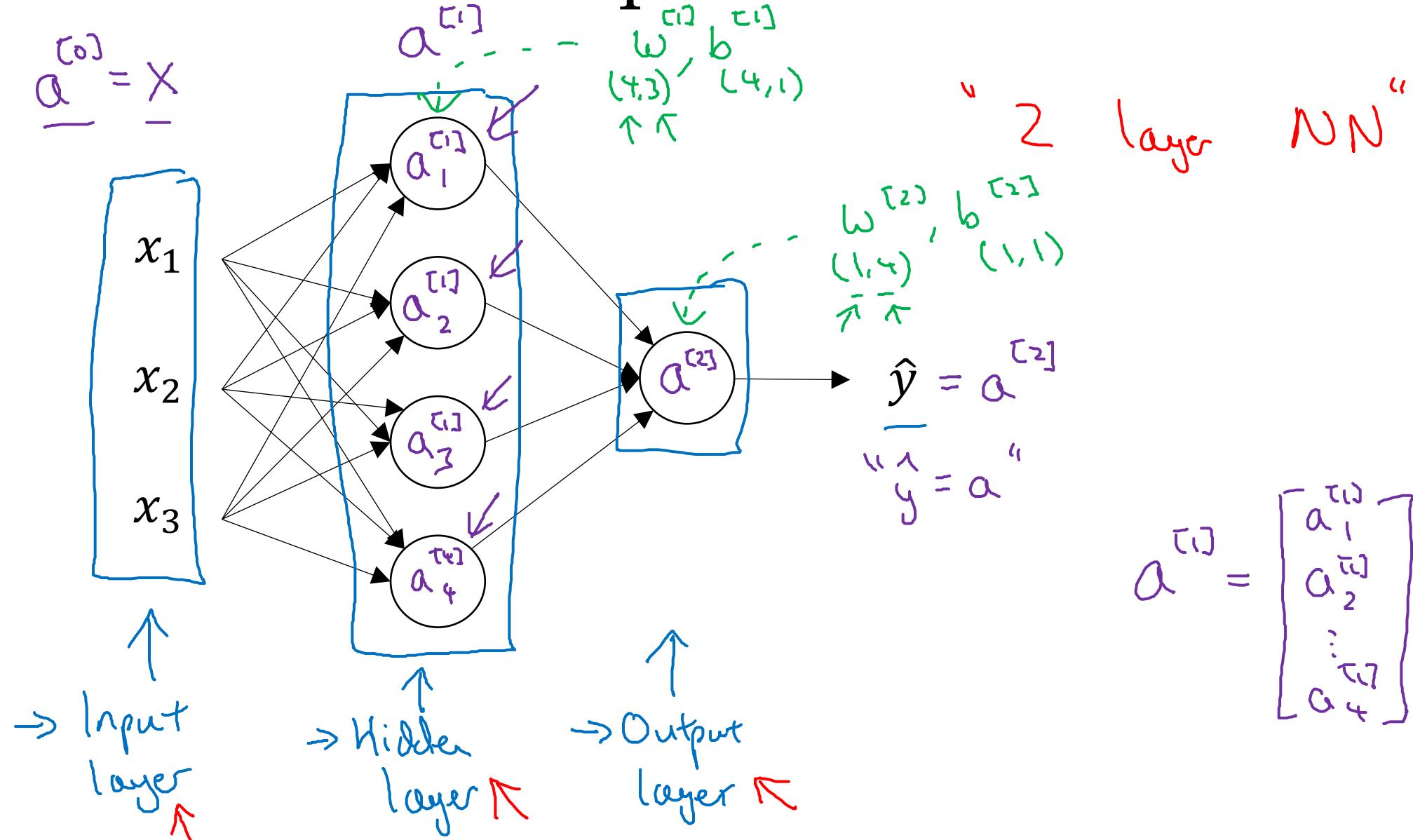


deeplearning.ai

One hidden layer
Neural Network

Neural Network
Representation

Neural Network Representation



$$a^{(1)} = \begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ \vdots \\ a_4^{(1)} \end{bmatrix}$$

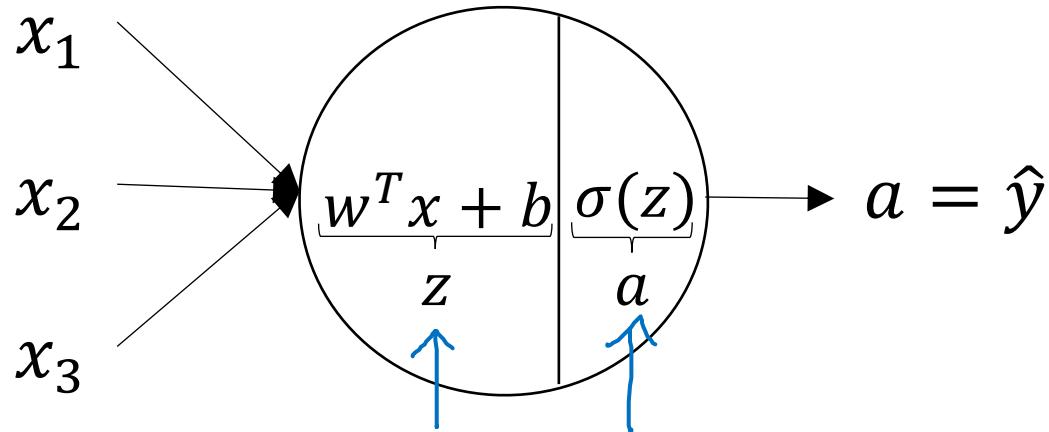


deeplearning.ai

One hidden layer Neural Network

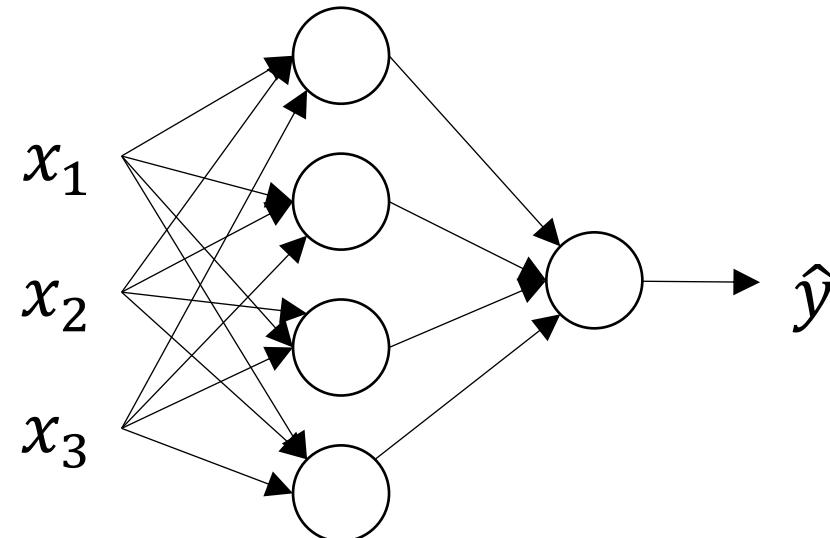
Computing a Neural Network's Output

Neural Network Representation

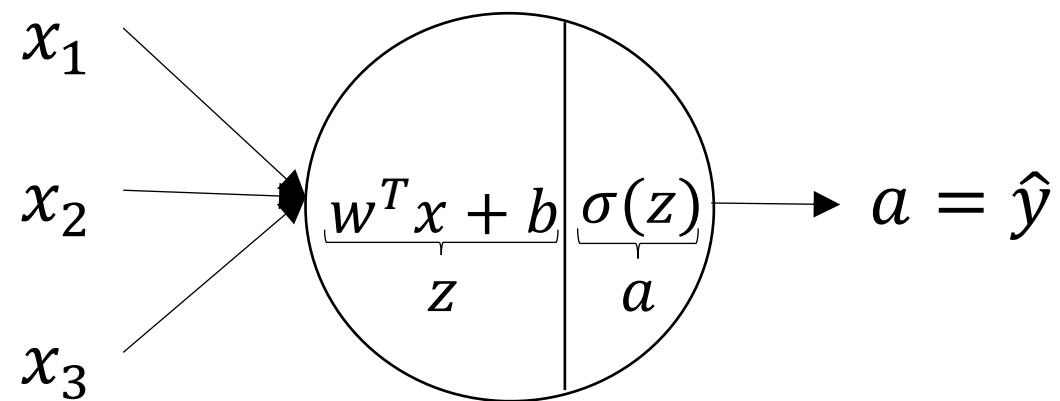


$$z = w^T x + b$$

$$a = \sigma(z)$$

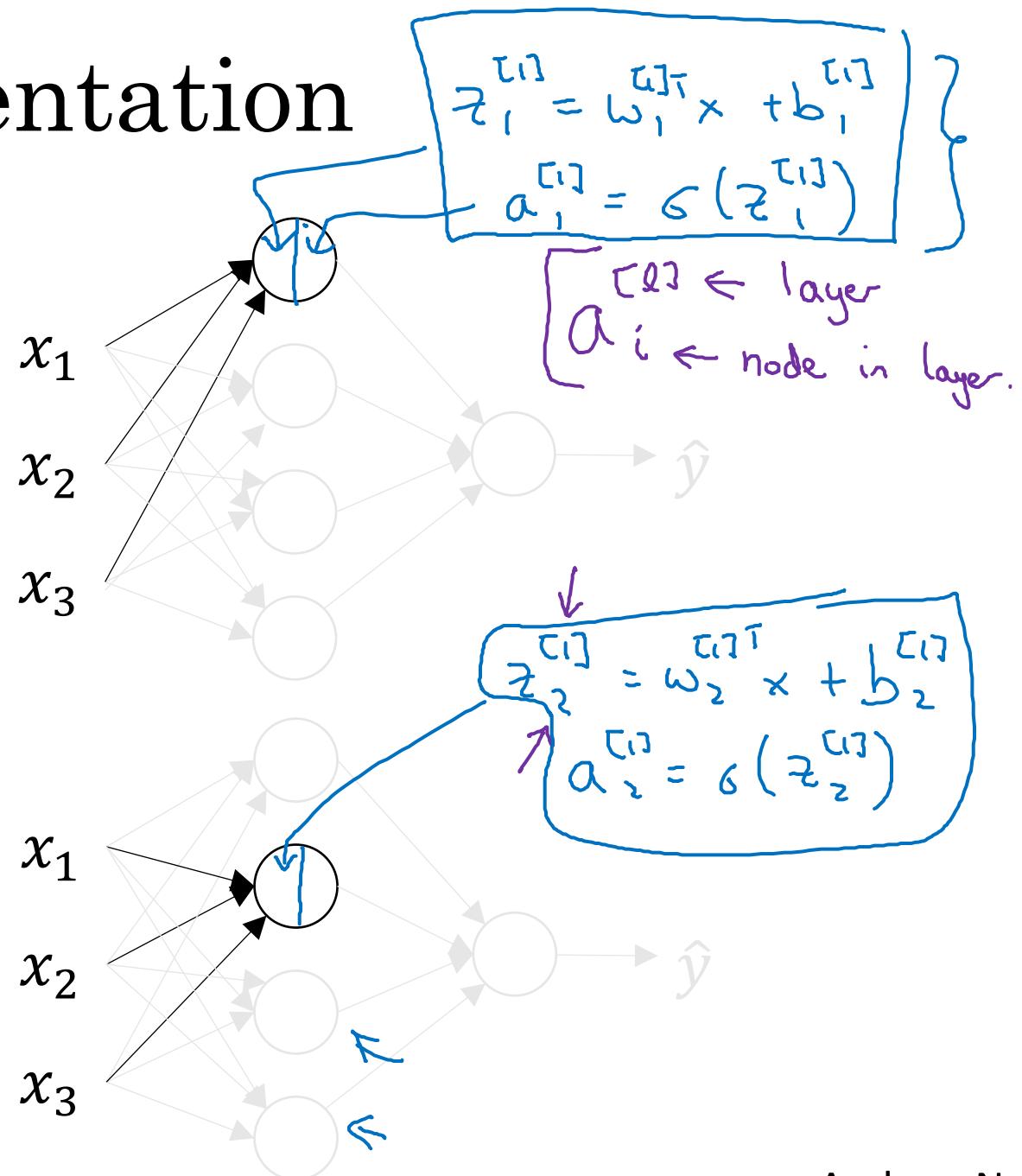


Neural Network Representation

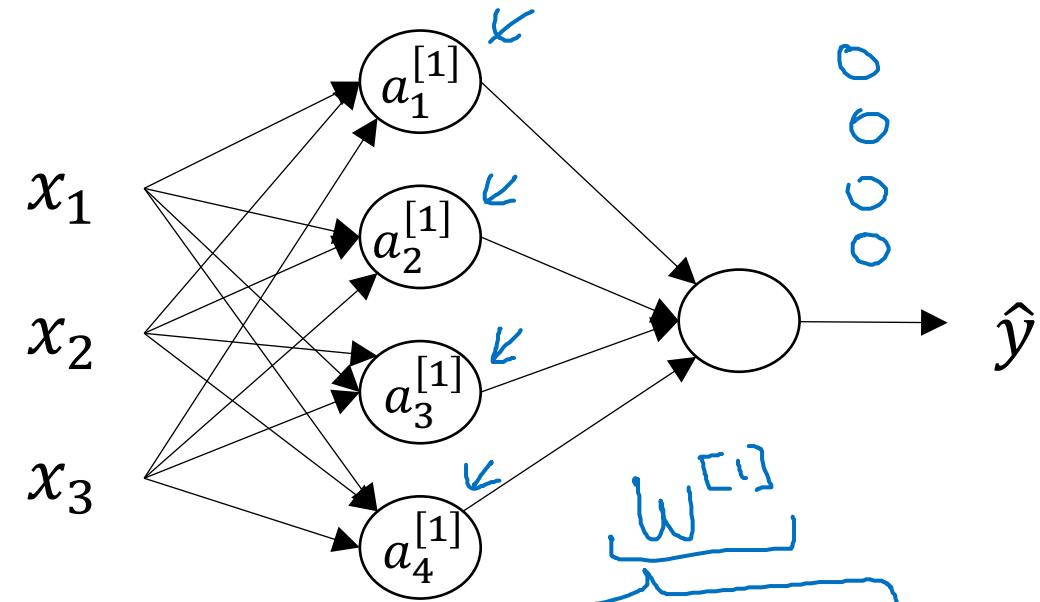


$$z = w^T x + b$$

$$a = \sigma(z)$$



Neural Network Representation



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

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

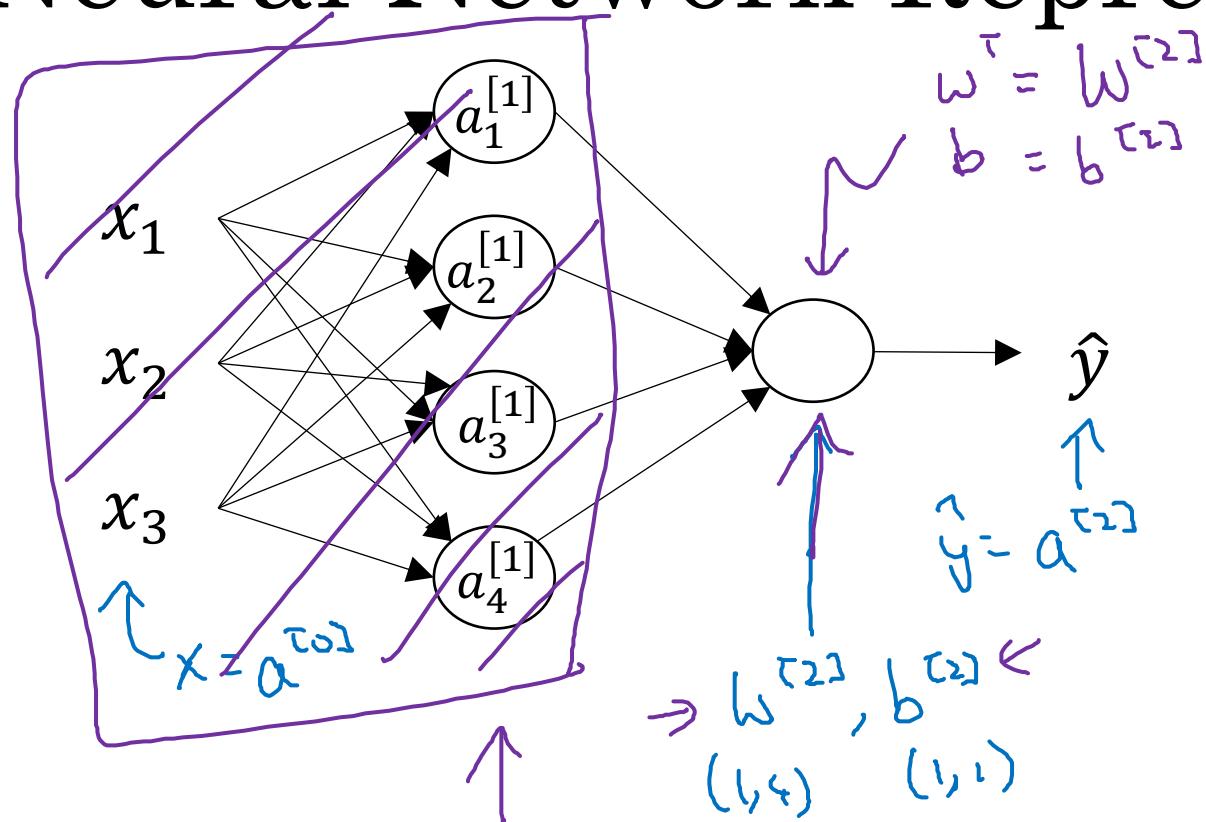
Diagram illustrating the mathematical representation of the neural network layers:

- Layer 1: $z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}$, $a_1^{[1]} = \sigma(z_1^{[1]})$
- Layer 2: $z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}$, $a_2^{[1]} = \sigma(z_2^{[1]})$
- Layer 3: $z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}$, $a_3^{[1]} = \sigma(z_3^{[1]})$
- Layer 4: $z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}$, $a_4^{[1]} = \sigma(z_4^{[1]})$

Annotations:

- $(w_1^{[1]T} x)$ is highlighted in blue.
- $a^{[1]}$ is highlighted in red.
- $b^{[1]}$ is highlighted in green.
- $z^{[1]}$ is highlighted in purple.
- $\sigma(z^{[1]})$ is highlighted in black.

Neural Network Representation learning



Given input x :

$$\begin{aligned} z^{[1]} &= W^{[1]} a^{[0]} + b^{[1]} \\ a^{[1]} &= \sigma(z^{[1]}) \\ z^{[2]} &= W^{[2]} a^{[1]} + b^{[2]} \\ a^{[2]} &= \sigma(z^{[2]}) \end{aligned}$$

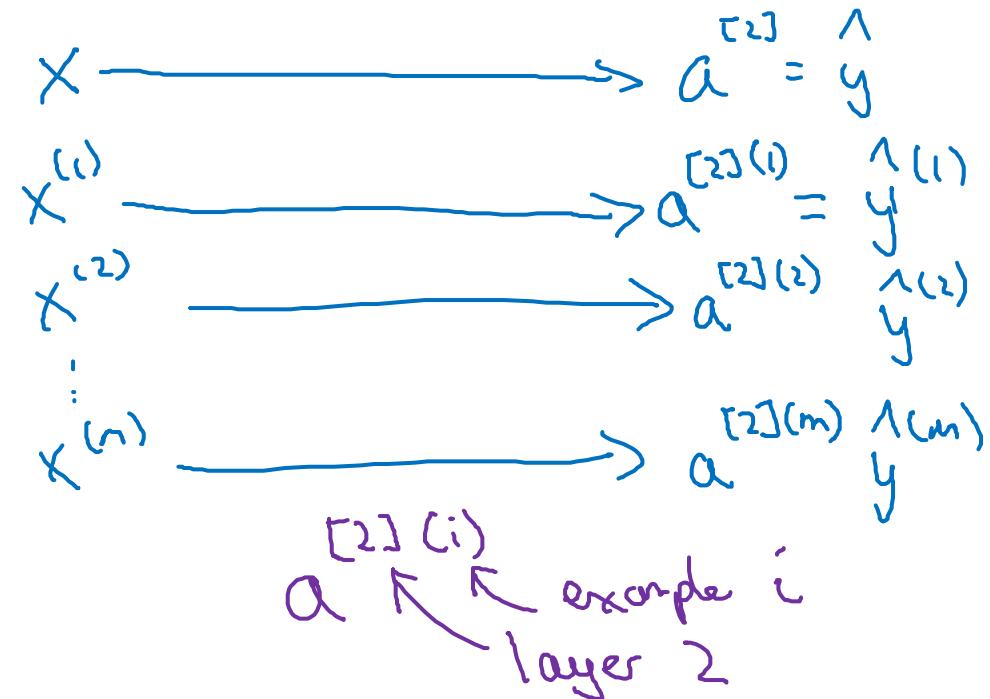
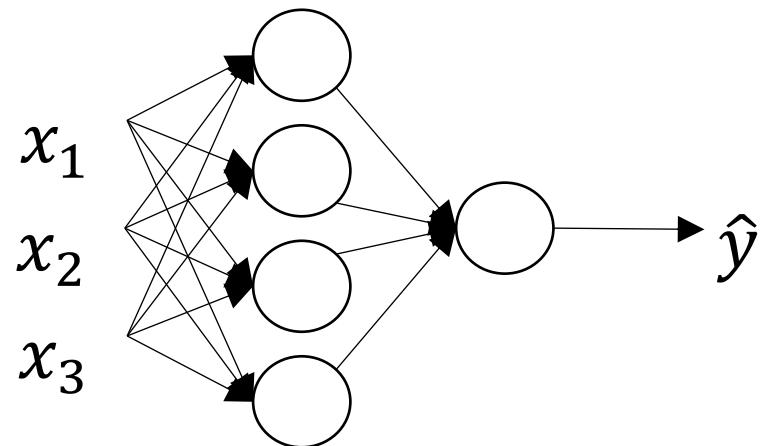


deeplearning.ai

One hidden layer Neural Network

Vectorizing across
multiple examples

Vectorizing across multiple examples



$\left. \begin{array}{l} z^{[1]} = W^{[1]}x + b^{[1]} \\ a^{[1]} = \sigma(z^{[1]}) \\ z^{[2]} = W^{[2]}a^{[1]} + b^{[2]} \\ a^{[2]} = \sigma(z^{[2]}) \end{array} \right\} \quad \nwarrow$

for $i = 1$ to m ,

$$\begin{aligned} z^{[1](i)} &= w^{[1]}x^{(i)} + b^{[1]} \\ a^{[1](i)} &= \sigma(z^{[1](i)}) \\ z^{[2](i)} &= w^{[2]}a^{[1](i)} + b^{[2]} \\ a^{[2](i)} &= \sigma(z^{[2](i)}) \end{aligned}$$

Vectorizing across multiple examples

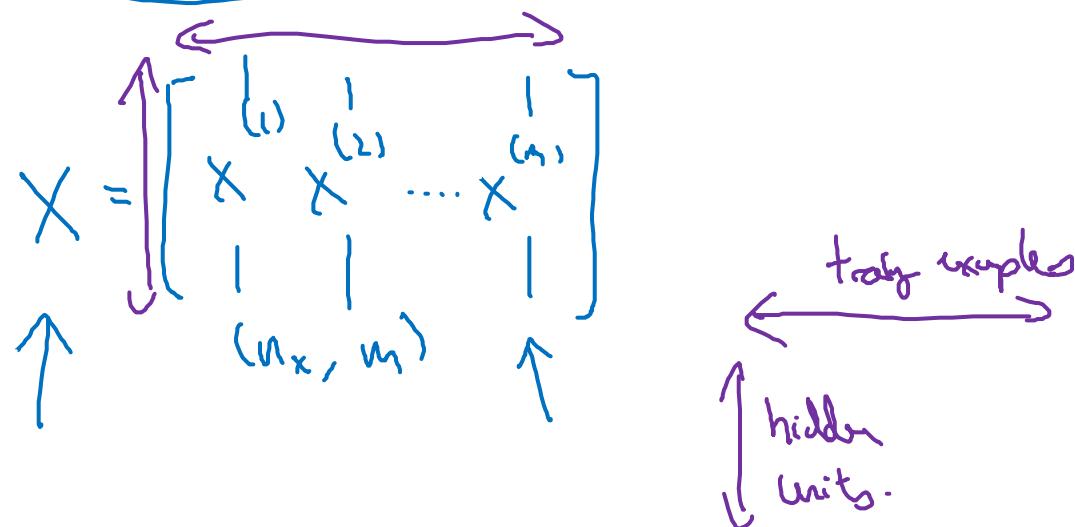
for $i = 1$ to m :

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

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

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

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

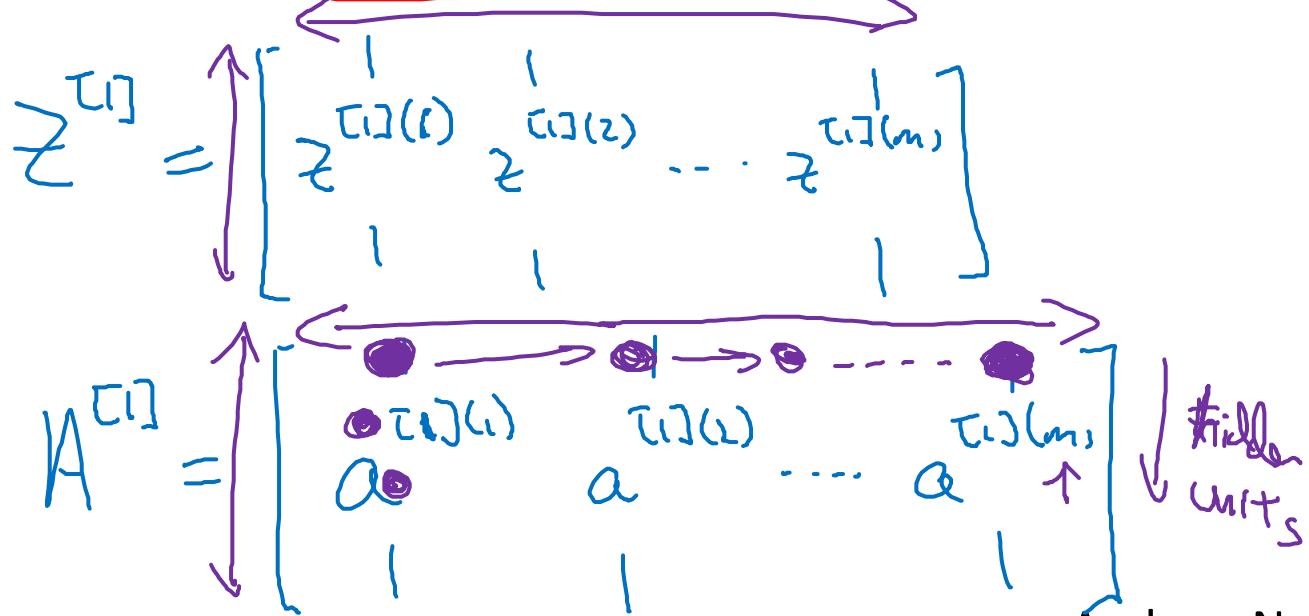


$$z^{[1]} = W^{[1]}X + b^{[1]}$$

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

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

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





deeplearning.ai

One hidden layer Neural Network

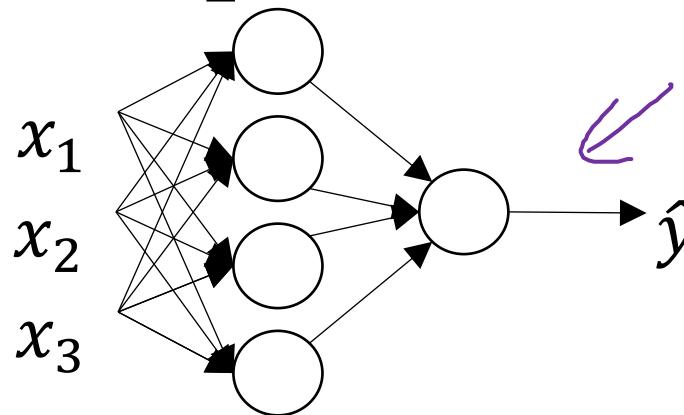
Explanation for vectorized implementation

Justification for vectorized implementation

$$z^{(1)(1)} = w^{(1)} x^{(1)} + b^{(1)}, \quad z^{(1)(2)} = w^{(1)} x^{(2)} + b^{(1)}, \quad z^{(1)(3)} = w^{(1)} x^{(3)} + b^{(1)}$$
$$w^{(1)} = \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix}, \quad w^{(1)} x^{(1)} = \begin{bmatrix} \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad w^{(1)} x^{(2)} = \begin{bmatrix} \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad w^{(1)} x^{(3)} = \begin{bmatrix} \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$
$$z^{(1)} = w^{(1)} X + b^{(1)} = \begin{bmatrix} 1 & 1 & 1 \\ x^{(1)} & x^{(2)} & x^{(3)} \dots \end{bmatrix} = \begin{bmatrix} \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \end{bmatrix} = \begin{bmatrix} z^{(1)(1)} & z^{(1)(2)} & z^{(1)(3)} \dots \\ | & | & | \\ + b^{(1)} & + b^{(1)} & + b^{(1)} \end{bmatrix} = z^{(1)}$$
$$w^{(1)} x^{(1)} = z^{(1)(1)}$$

Andrew Ng

Recap of vectorizing across multiple examples



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

A purple arrow points from the text "vectorizing across multiple examples" to this equation.

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

A purple arrow points from the text "vectorizing across multiple examples" to this equation.

```

for i = 1 to m
    z[1](i) = W[1]x(i) + b[1]
    → a[1](i) = σ(z[1](i))
    → z[2](i) = W[2]a[1](i) + b[2]
    → a[2](i) = σ(z[2](i))
    x = a[1]           x(i) = a[1](i)
    A[0]

```

```

Z[1] = W[1]X + b[1] ← wT,1A[0]+bT,1
A[1] = σ(Z[1])
Z[2] = W[2]A[1] + b[2]
A[2] = σ(Z[2])

```

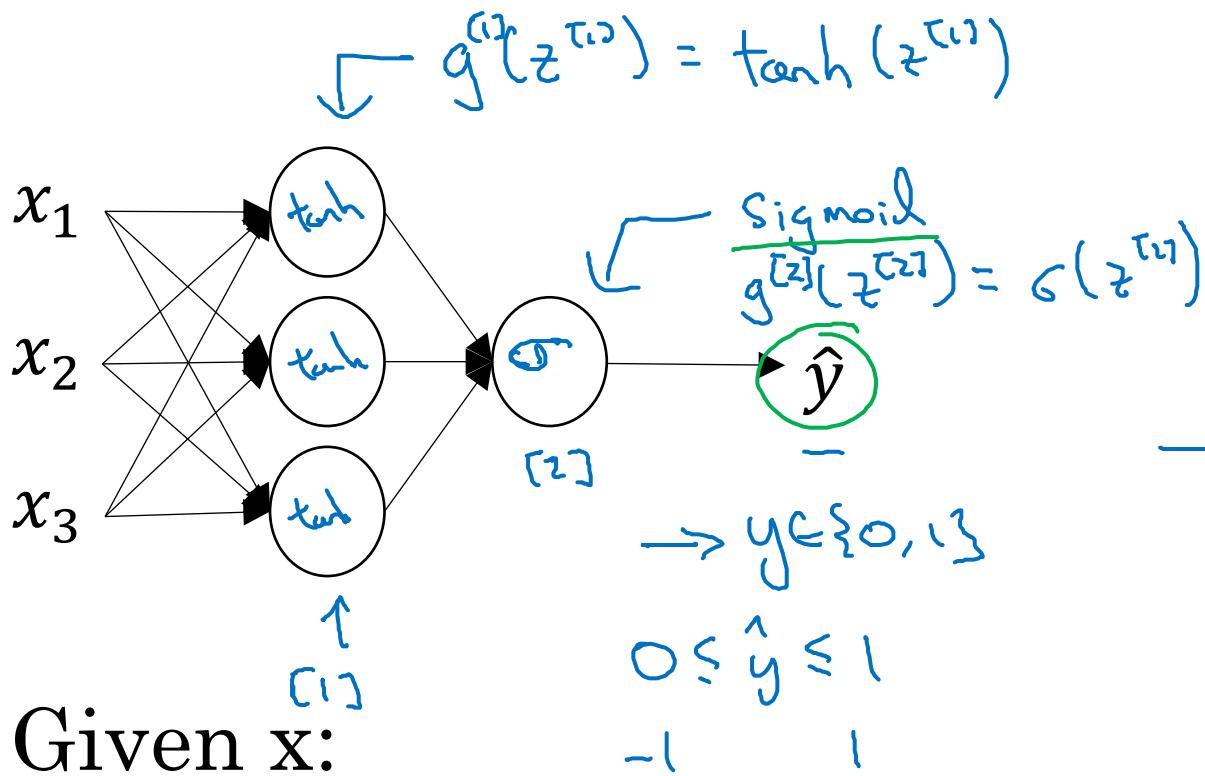


deeplearning.ai

One hidden layer Neural Network

Activation functions

Activation functions



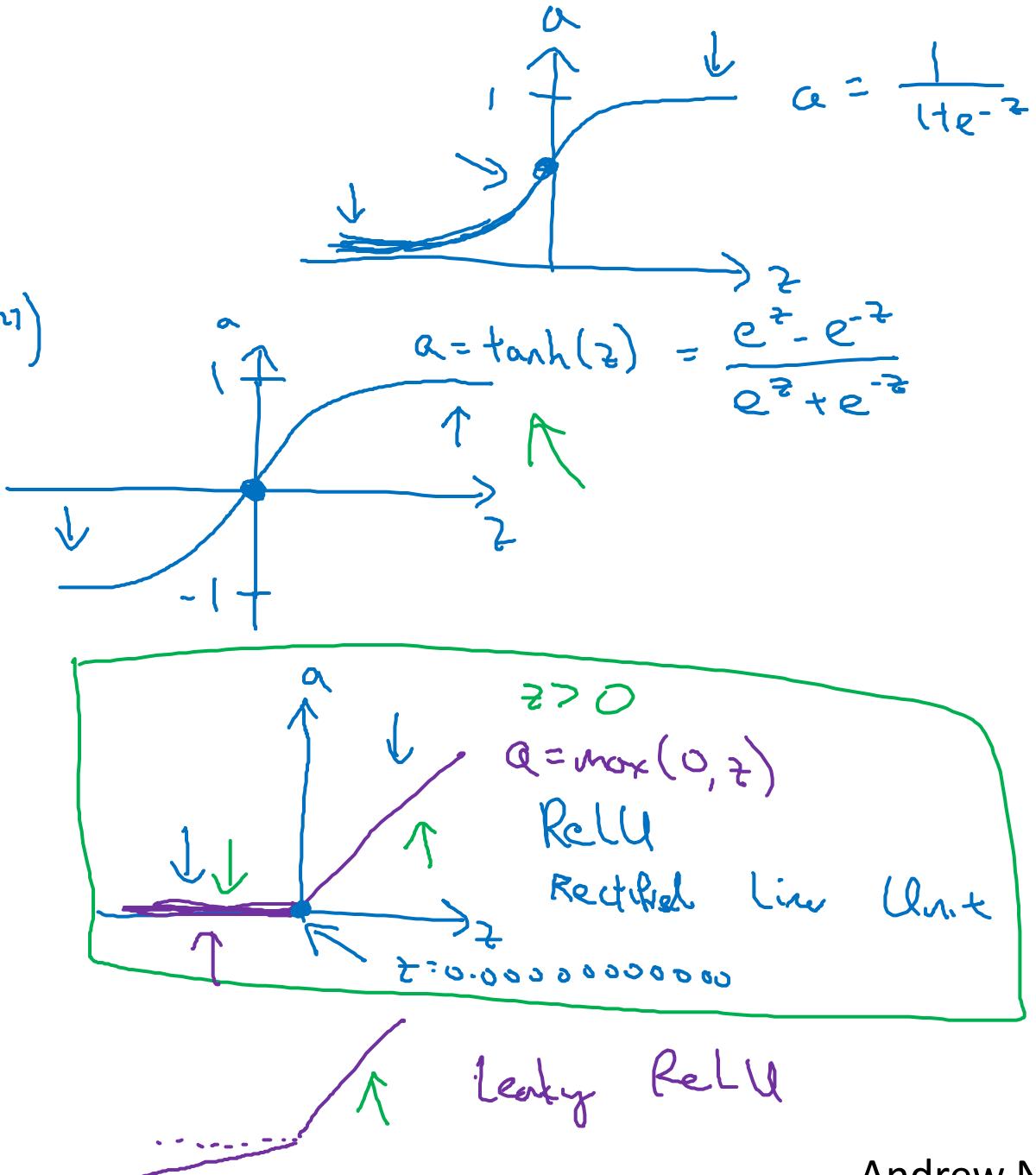
Given x :

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

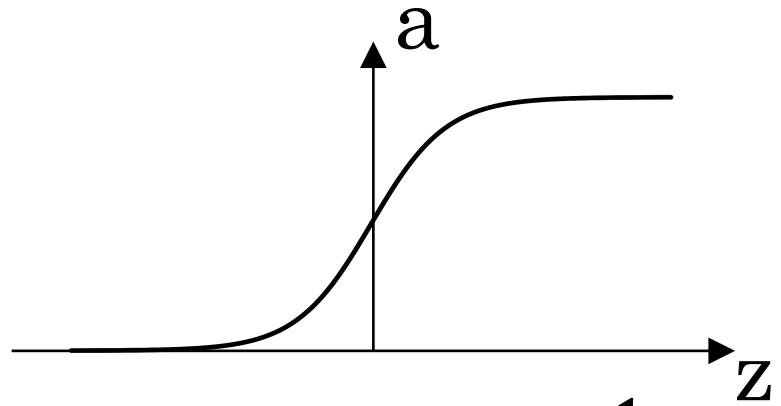
$$\rightarrow a^{[1]} = \cancel{\sigma(z^{[1]})} \quad g^{[1]}(z^{[1]})$$

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

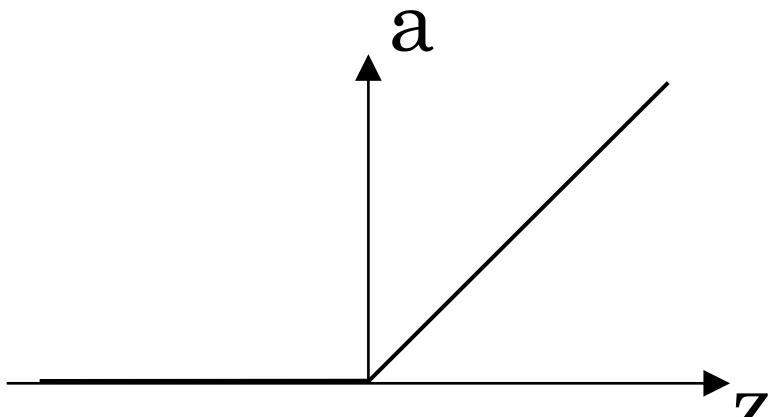
$$\rightarrow a^{[2]} = \cancel{\sigma(z^{[2]})} \quad g^{[2]}(z^{[2]})$$



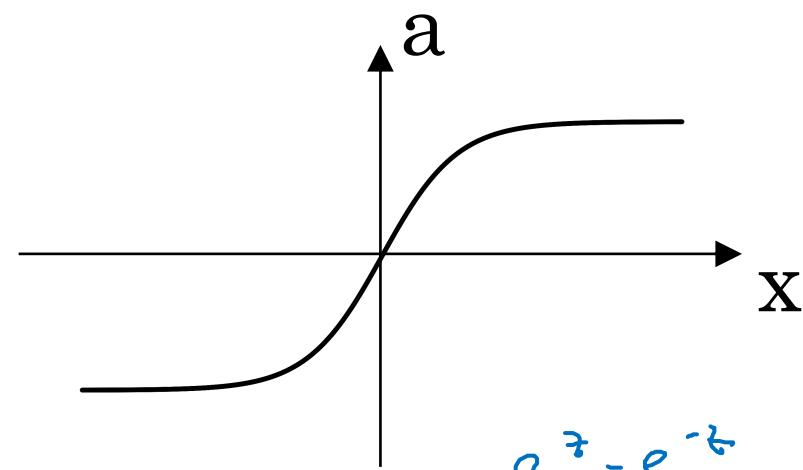
Pros and cons of activation functions



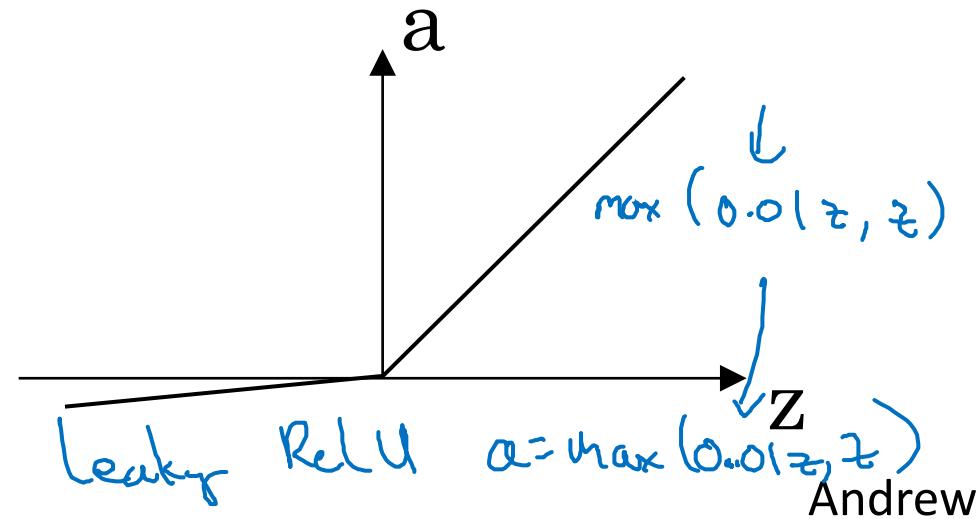
$$\text{sigmoid: } a = \frac{1}{1 + e^{-z}}$$



$$\text{ReLU} \quad a = \max(0, z)$$



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



$$\text{Leaky ReLU} \quad a = \max(0.01z, z) \quad \text{Andrew Ng}$$

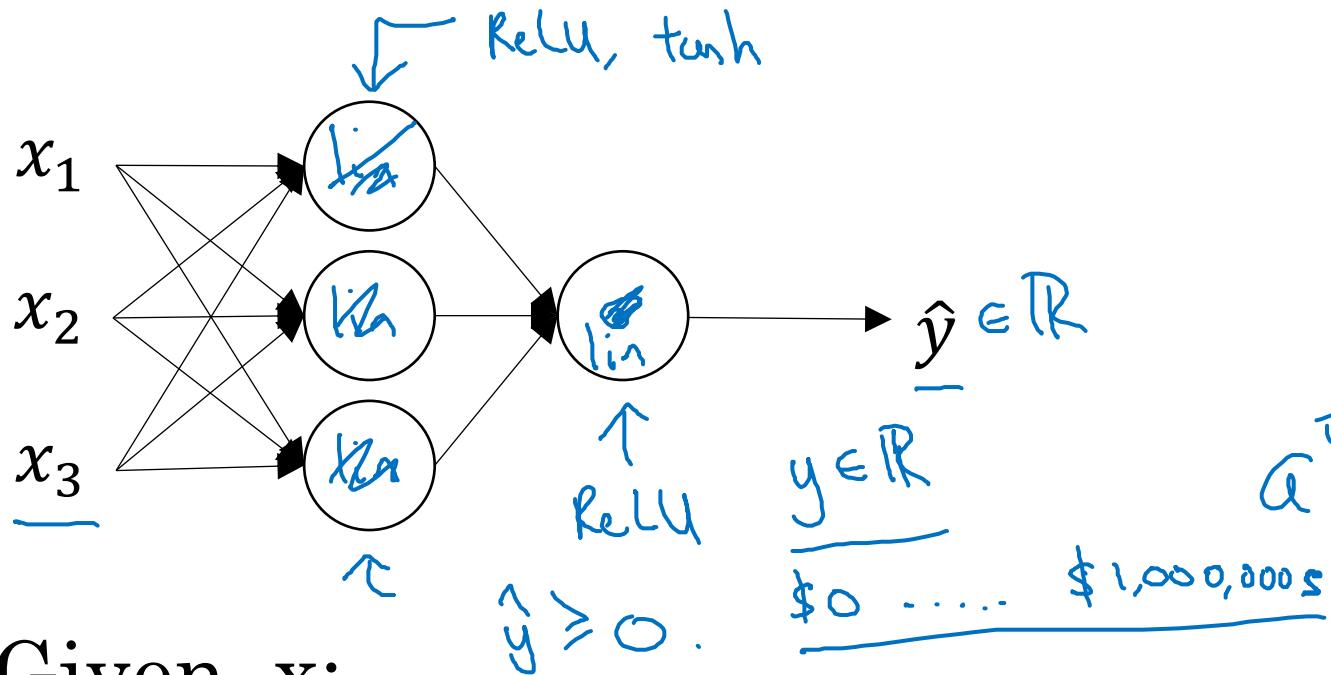


deeplearning.ai

One hidden layer Neural Network

Why do you
need non-linear
activation functions?

Activation function



Given x :

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

$$\rightarrow a^{[1]} = \underline{g^{[1]}(z^{[1]})} \geq^{[1]}$$

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

$$\rightarrow a^{[2]} = \underline{g^{[2]}(z^{[2]})} \geq^{[2]}$$

$g(z) = z$
"linear activation
function"

$$a^{[1]} = z^{[1]} = \underbrace{W^{[1]}x + b^{[1]}}_{a^{[1]}}$$

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

$$a^{[2]} = W^{[2]} \left(\underbrace{W^{[1]}x + b^{[1]}}_{a^{[1]}} \right) + b^{[2]}$$

$$= \underbrace{(W^{[2]}W^{[1]})}_{w'} x + \underbrace{(W^{[2]}b^{[1]} + b^{[2]})}_{b'}$$

$$= \underline{w'x + b'}$$

$$g(z) = z$$

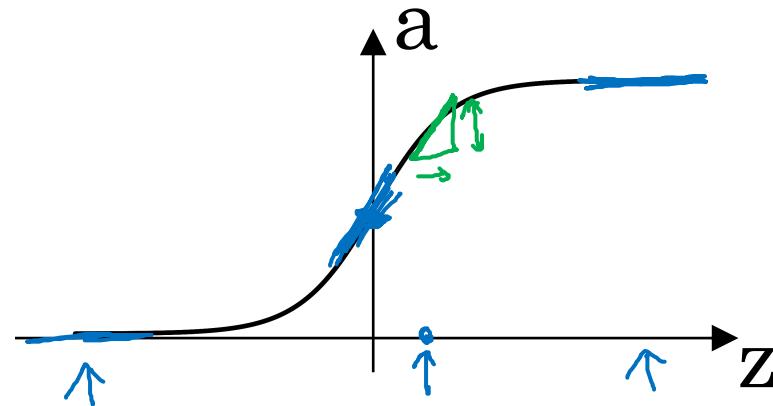


deeplearning.ai

One hidden layer
Neural Network

Derivatives of
activation functions

Sigmoid activation function



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

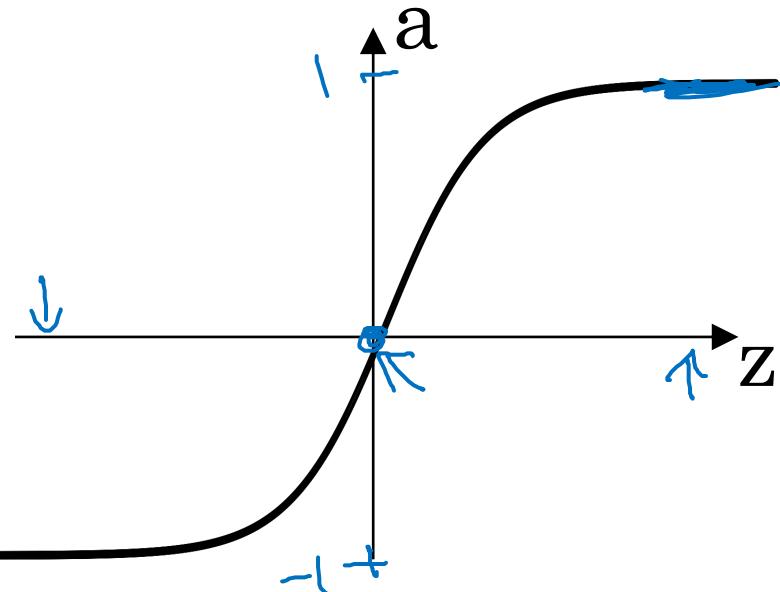
$$a = g(z) = \frac{1}{1 + e^{-z}}$$

$$\begin{aligned} g'(z) &= \boxed{\frac{d}{dz} g(z)} \\ &= \frac{1}{1 + e^{-z}} \left(1 - \frac{1}{1 + e^{-z}} \right) \\ &= g(z) \left(1 - g(z) \right) \leftarrow \boxed{g'(z) = a(1-a)} \\ &= \boxed{a(1-a)} \end{aligned}$$

$$\begin{aligned} z = 10, \quad g(z) &\approx 1 \\ \frac{d}{dz} g(z) &\approx 1(1-1) \approx 0 \\ z = -10, \quad g(z) &\approx 0 \\ \frac{d}{dz} g(z) &\approx 0 \cdot (1-0) \approx 0 \\ z = 0, \quad g(z) &= \frac{1}{2} \\ \frac{d}{dz} g(z) &= \frac{1}{2}(1-\frac{1}{2}) = \frac{1}{4} \end{aligned}$$

Andrew Ng

Tanh activation function



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

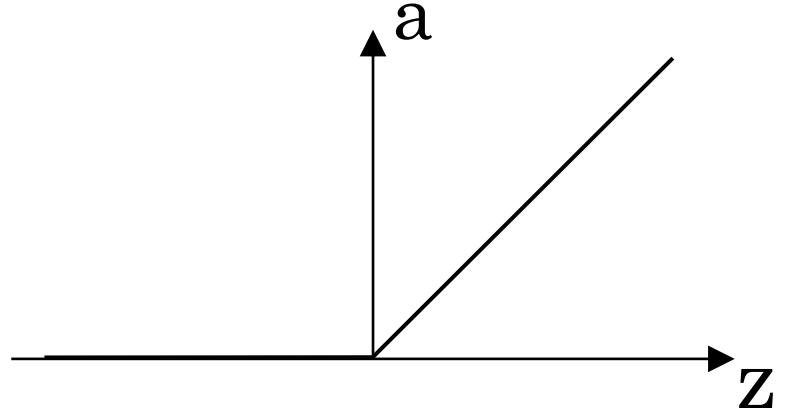
$$= \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$\begin{aligned} g'(z) &= \frac{d}{dz} g(z) = \text{slope of } g(z) \text{ at } z \\ &= 1 - \underline{\underline{(\tanh(z))^2}} \end{aligned}$$

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

$$\left| \begin{array}{ll} z = 10 & \tanh(z) \approx 1 \\ & g'(z) \approx 0 \\ z = -10 & \tanh(z) \approx -1 \\ & g'(z) \approx 0 \\ z = 0 & \tanh(z) = 0 \\ & g'(z) = 1 \end{array} \right.$$

ReLU and Leaky ReLU

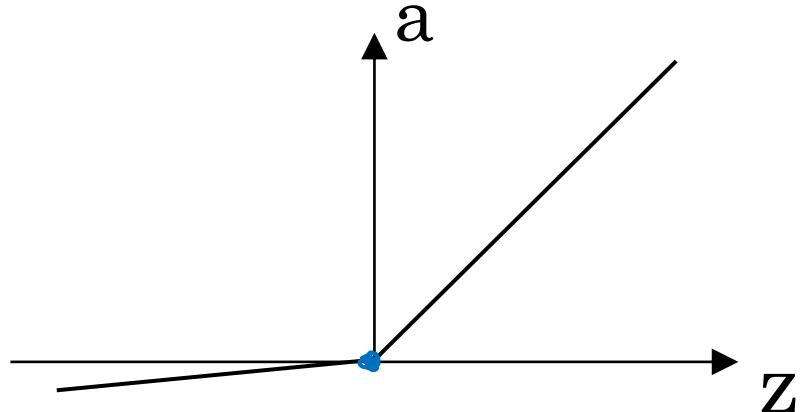


ReLU

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

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

~~undefined if $z=0$~~
 $z = 0.0000\cdots 0$



Leaky ReLU

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

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



deeplearning.ai

One hidden layer
Neural Network

Gradient descent for
neural networks

Gradient descent for neural networks

Parameters: $(\omega^{[1]}, b^{[1]}, \dots, \omega^{[L]}, b^{[L]})$ $n_x = n^{[0]}, n^{[1]}, \dots, \underline{n^{[L]}} = 1$

Cost function: $J(\underbrace{\omega^{[1]}, b^{[1]}, \dots, \omega^{[L]}, b^{[L]}}, \underbrace{\hat{y}, y}) = \frac{1}{m} \sum_{i=1}^m l(\hat{y}_i, y_i)$

Gradient Descent:

→ Repeat {

→ Compute $\hat{y}^{(i)}$, $i=1 \dots m$

$$\frac{\partial J}{\partial \omega^{[l]}} = \frac{\partial J}{\partial \omega^{[l]}} , \quad \frac{\partial J}{\partial b^{[l]}} = \frac{\partial J}{\partial b^{[l]}} , \dots$$

$$\omega^{[l]} := \omega^{[l]} - \alpha \frac{\partial J}{\partial \omega^{[l]}}$$

$$b^{[l]} := b^{[l]} - \alpha \frac{\partial J}{\partial b^{[l]}}$$

↳

Formulas for computing derivatives

Forward propagation:

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

$$A^{[1]} = g^{[1]}(z^{[1]}) \leftarrow$$

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

$$A^{[2]} = g^{[2]}(z^{[2]}) = \underline{\underline{\sigma(z^{[2]})}}$$

Back propagation:

$$dz^{[2]} = A^{[2]} - Y \leftarrow$$

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

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

$$dz^{[1]} = \underbrace{w^{[2]T} dz^{[2]}}_{(n^{[2]}, m)} \times \underbrace{g^{[2]'}(z^{[2]})}_{\text{element-wise product}} \underbrace{(n^{[1]}, m)}$$

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

$$\cancel{db^{[1]} = \frac{1}{m} \underline{\underline{\text{np.sum}(dz^{[1]}, \text{axis}=1, \text{keepdims=True})}}} \quad \cancel{(n^{[1]}, 1)} \quad (n^{[1]}, 1) \quad \cancel{\text{reshape} \uparrow}$$

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

$$(n^{[1]}) \leftarrow$$

$$\cancel{\downarrow} (n^{[2]}, 1) \leftarrow$$



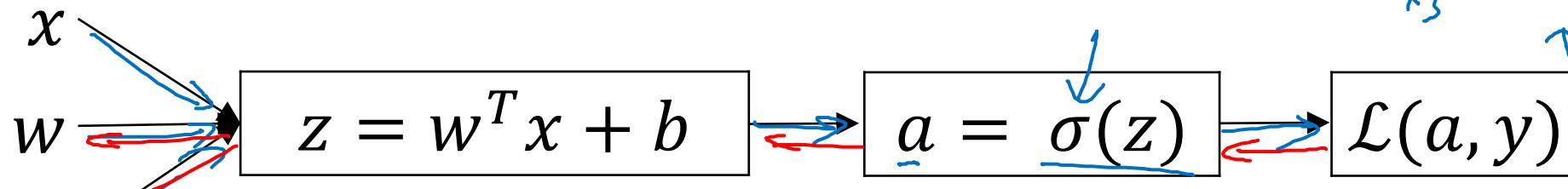
deeplearning.ai

One hidden layer
Neural Network

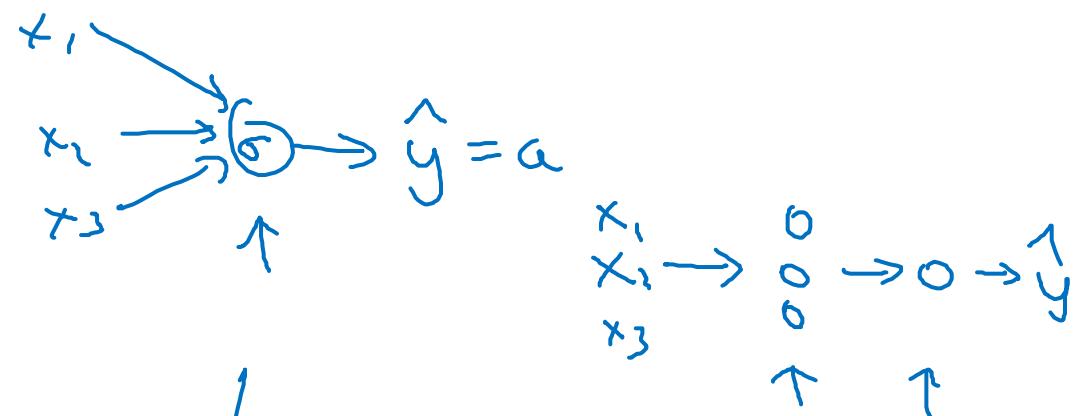
Backpropagation
intuition (Optional)

Computing gradients

Logistic regression



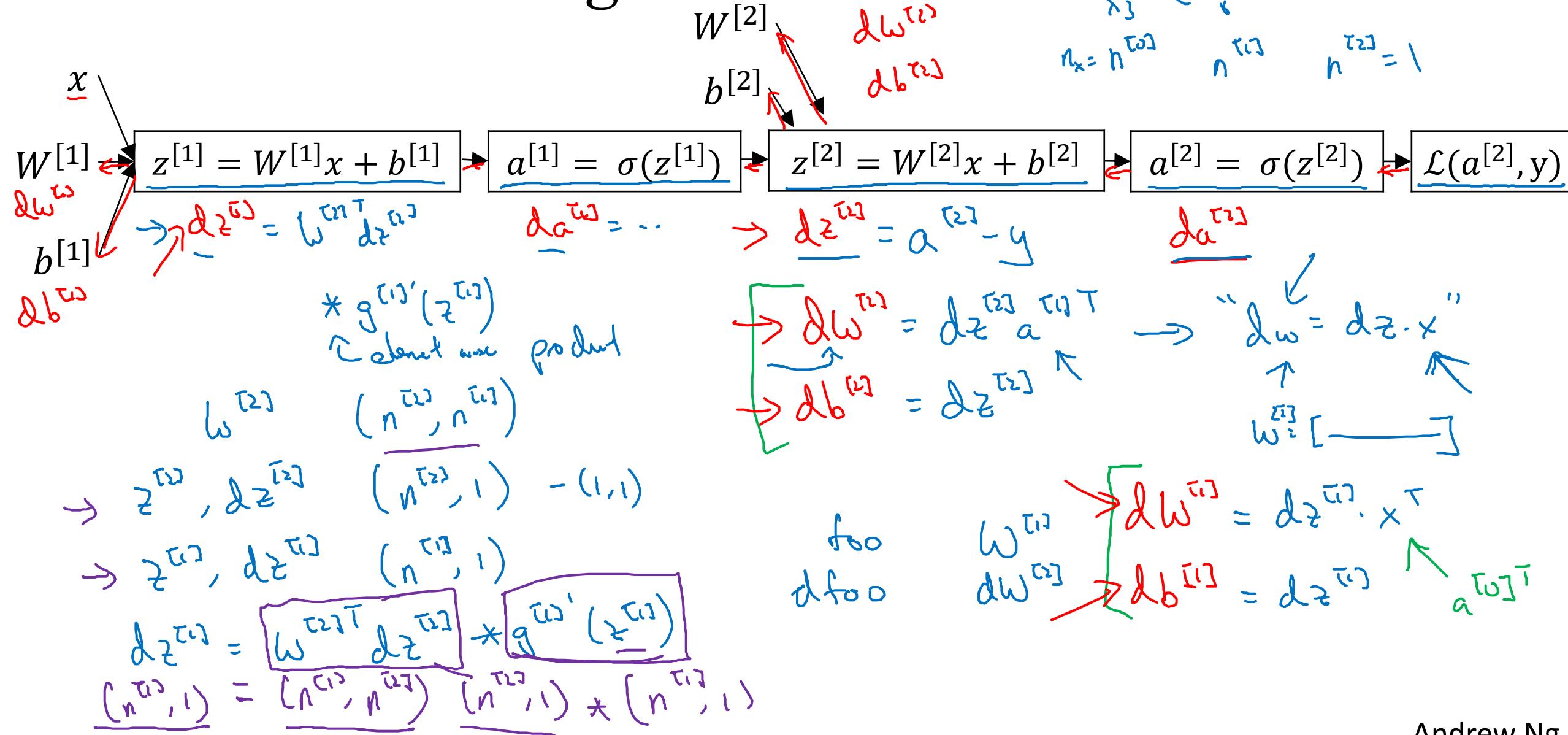
$$\begin{aligned} \frac{\partial z}{\partial w} &= x \\ \frac{\partial z}{\partial b} &= 1 \\ \frac{\partial z}{\partial a} &= g'(z) \\ g(z) &= \sigma(z) \\ \frac{\partial \mathcal{L}}{\partial z} &= \frac{\partial \mathcal{L}}{\partial a} \cdot \frac{\partial a}{\partial z} \\ "dz" &= "da" \end{aligned}$$



$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial a} &= \frac{d}{da} \mathcal{L}(a, y) = -y \log a - (1-y) \log(1-a) \\ &= -\frac{y}{a} + \frac{1-y}{1-a} \end{aligned}$$

$$\frac{d}{dz} g(z) = g'(z)$$

Neural network gradients



Summary of gradient descent

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

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

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

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

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

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

Vectorized implementation:

$$\begin{aligned} z^{[1]} &= \underbrace{w^{[1]} x + b^{[1]}}_{\text{Implementation}} \\ a^{[1]} &= g^{[1]}(z^{[1]}) \end{aligned}$$

$$Z^{[1]} = \begin{bmatrix} | & | & | & | \\ z^{1} & z^{[1](2)} & \dots & z^{[1](n)} \\ | & | & | & | \end{bmatrix}$$

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

Summary of gradient descent

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

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

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

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

(n^{T₁}, 1)

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

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

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

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

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

$$dZ^{[1]} = \underbrace{W^{[2]T} dZ^{[2]}}_{(n^{T₂}, m)} * \underbrace{g^{[1]'}(Z^{[1]})}_{(n^{T₁}, m)}$$

elementwise product

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

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

$$J(\cdot) = \frac{1}{m} \sum_{i=1}^m \mathcal{L}(\hat{y}_i, y_i)$$

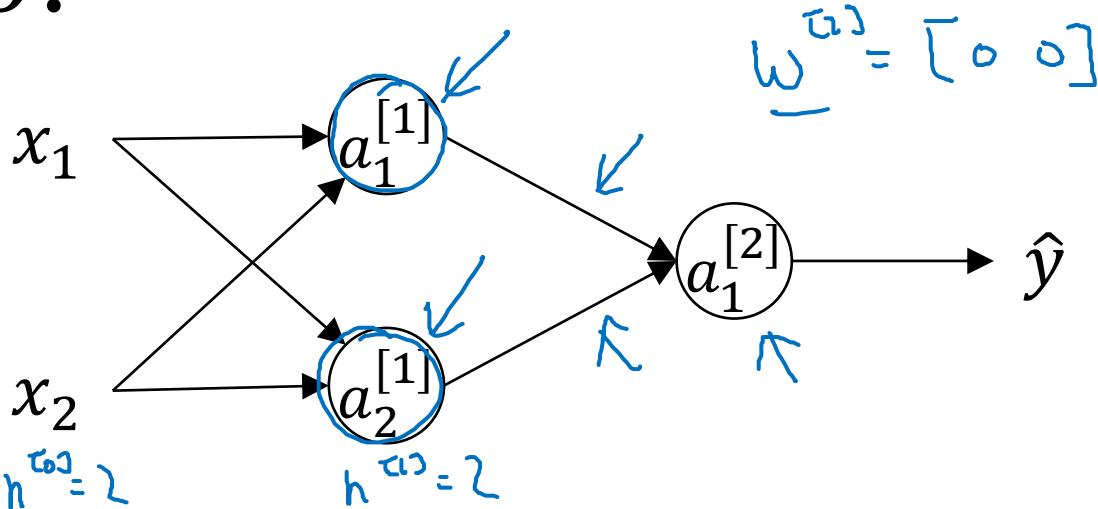


deeplearning.ai

One hidden layer
Neural Network

Random Initialization

What happens if you initialize weights to zero?



$$w^L = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

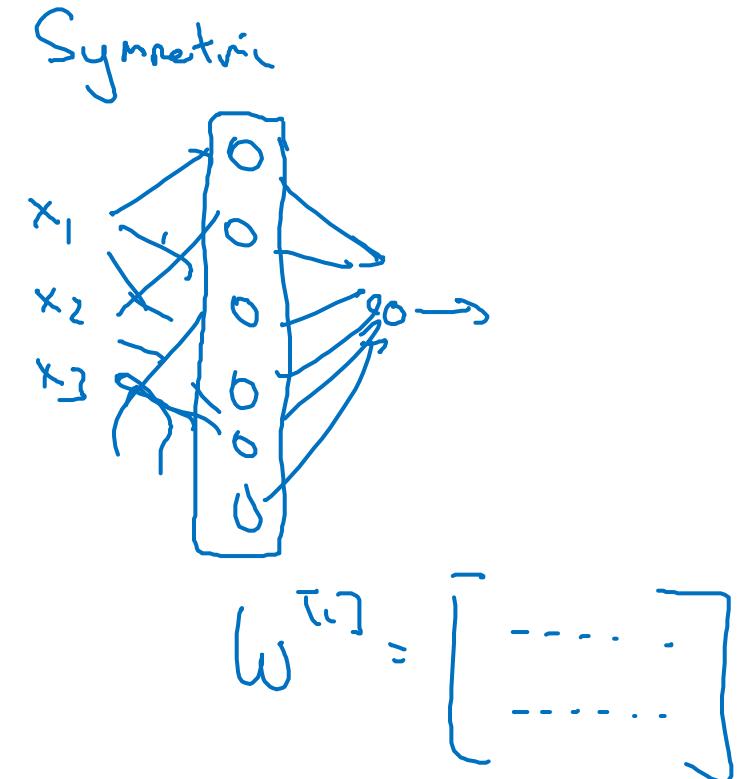
$$b^L = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$a_1^{[1]} = a_2^{[1]}$$

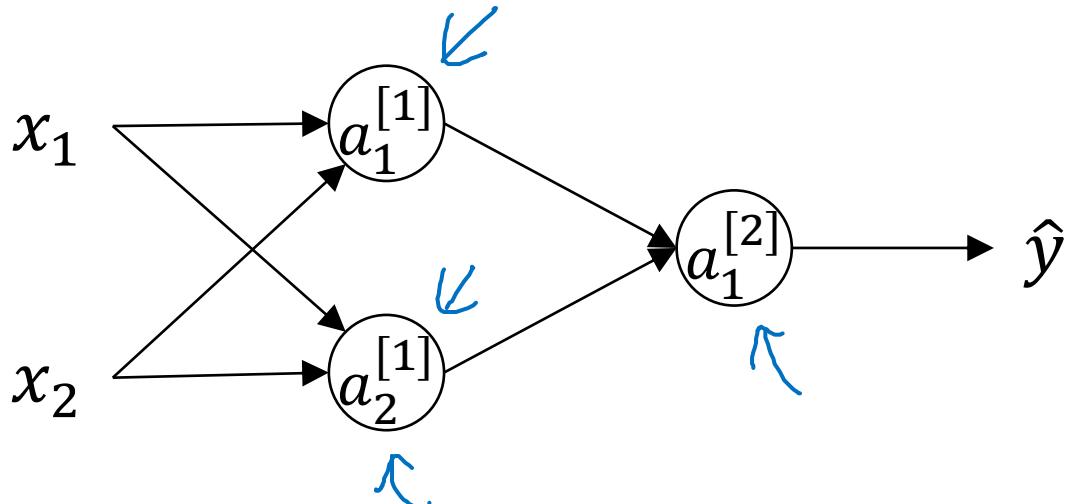
$$\delta z_1^{[1]} = \delta z_2^{[1]}$$

$$\delta w = \begin{bmatrix} u & v \\ u & v \end{bmatrix}$$

$$w^{[1]} = w^{[1]} - \lambda \delta w$$



Random initialization



$$\rightarrow w^{[1]} = \text{np.random.randn}(2, 2)$$

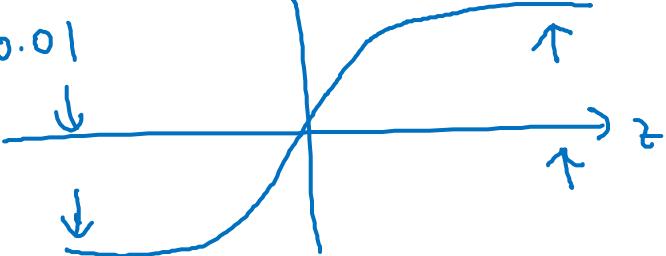
$$b^{[1]} = \text{np.zeros}(2, 1)$$

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

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

$$\times \frac{0.01}{100?}$$

$$\begin{aligned} z^{[1]} &= w^{[1]} \times + b^{[1]} \\ a^{[1]} &= g^{[1]}(z^{[1]}) \end{aligned}$$



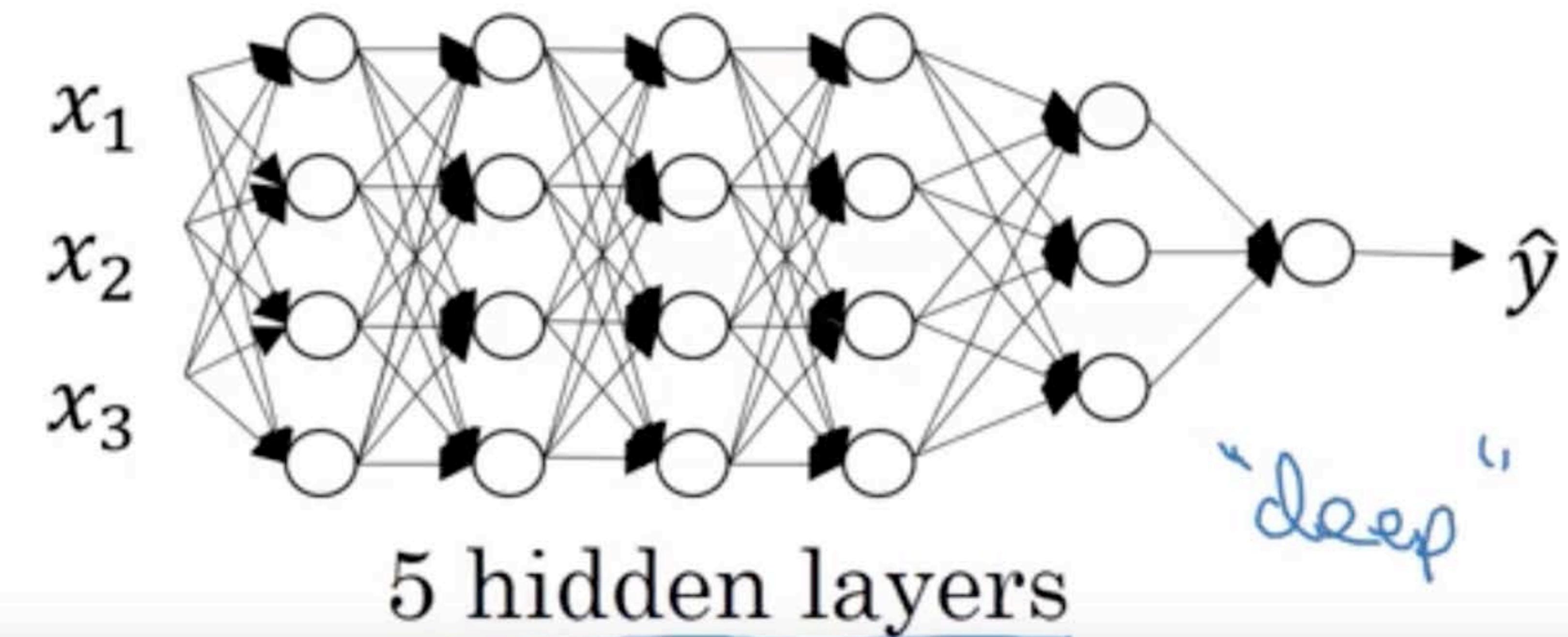
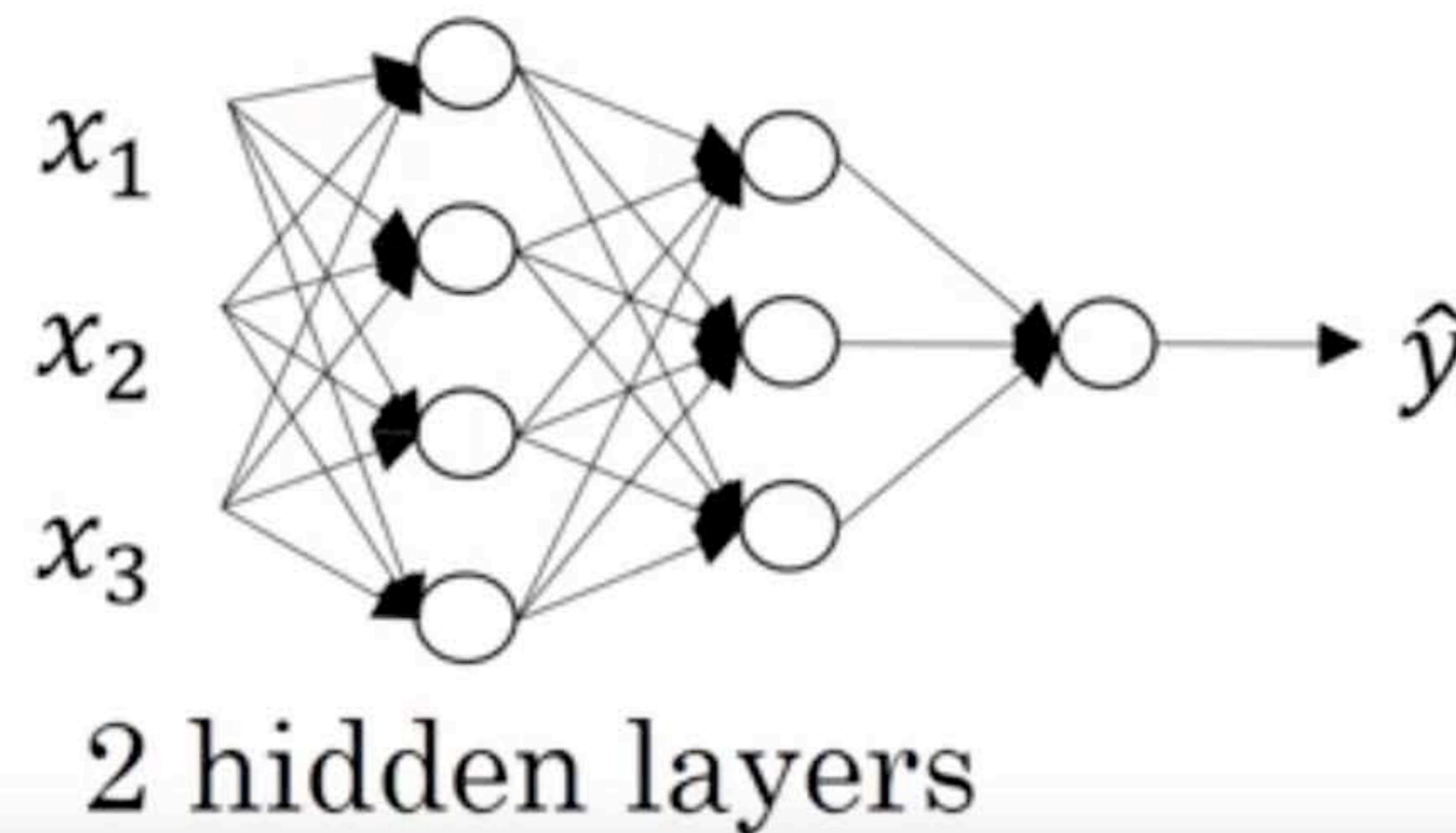
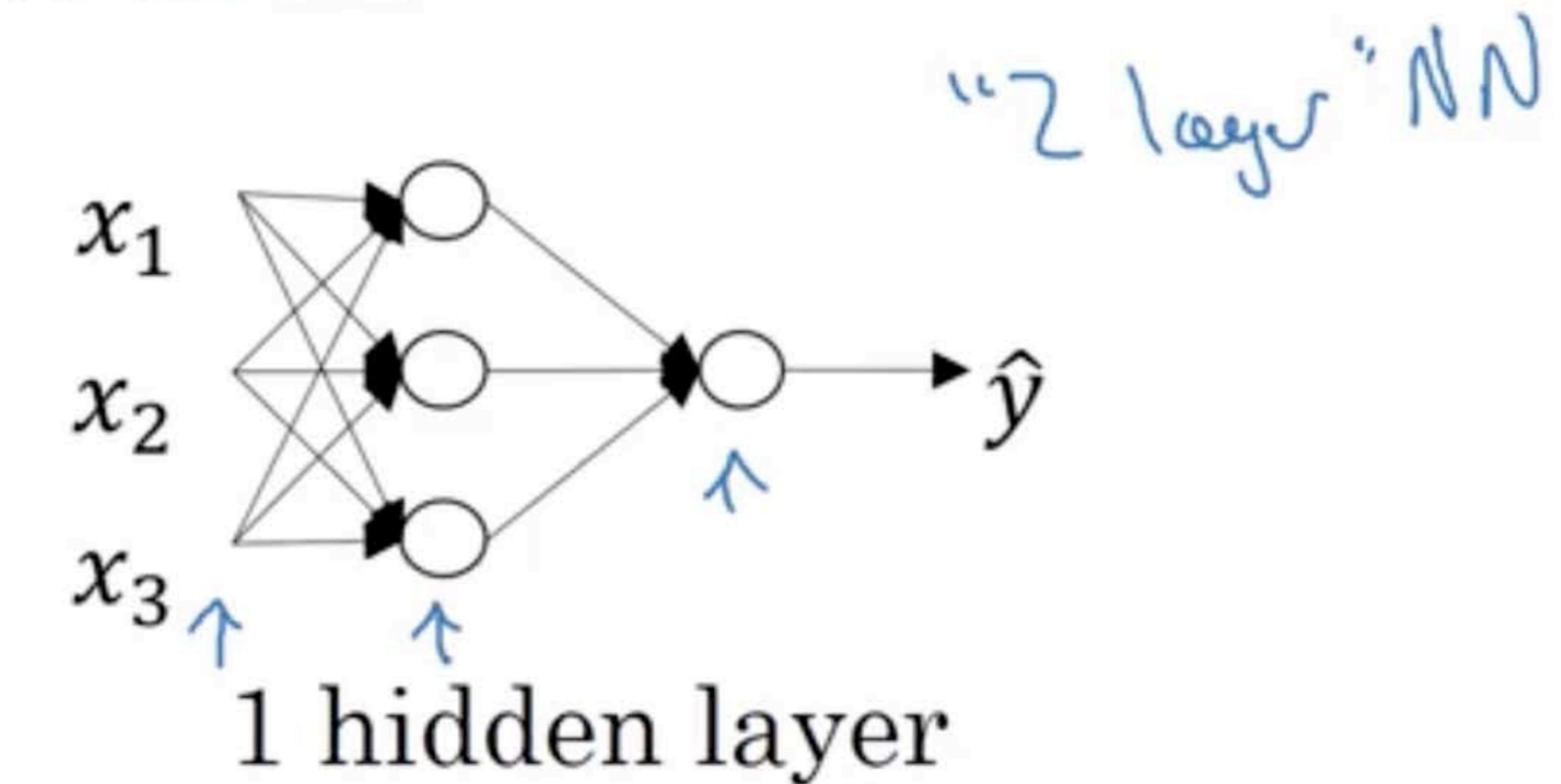
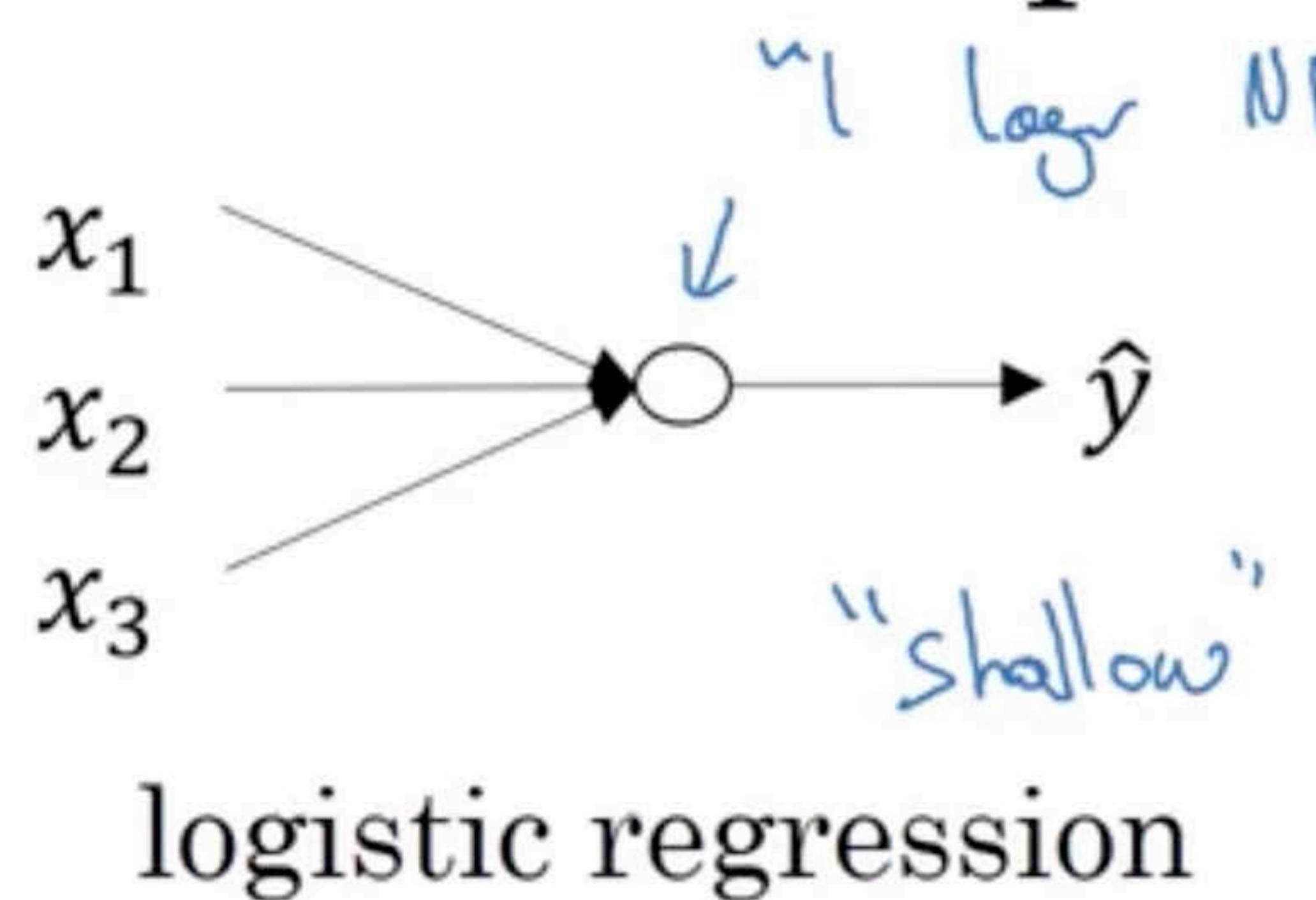


deeplearning.ai

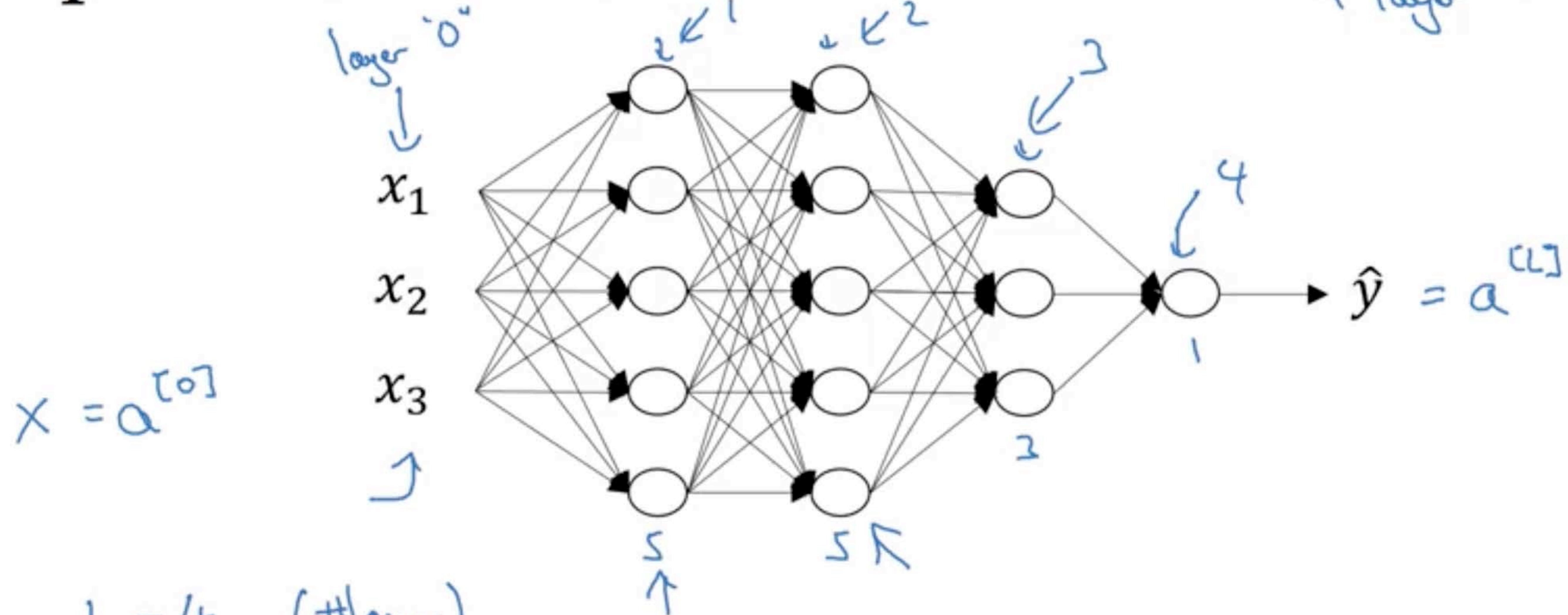
Deep Neural Networks

Deep L-layer
Neural network

What is a deep neural network?



Deep neural network notation



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

$$l = 4 \quad (\# \text{layers})$$

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

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

$a^{[l]} = g^{[l]}(z^{[l]})$, $w_{j,n}^{[l]}$ = weights for $\underline{z}^{[l]}$

$$n^{[1]} = 5, n^{[2]} = 5, n^{[3]} = 3, n^{[4]} = n^{[L]} = 1$$
$$n^{[0]} = n_x = 3$$

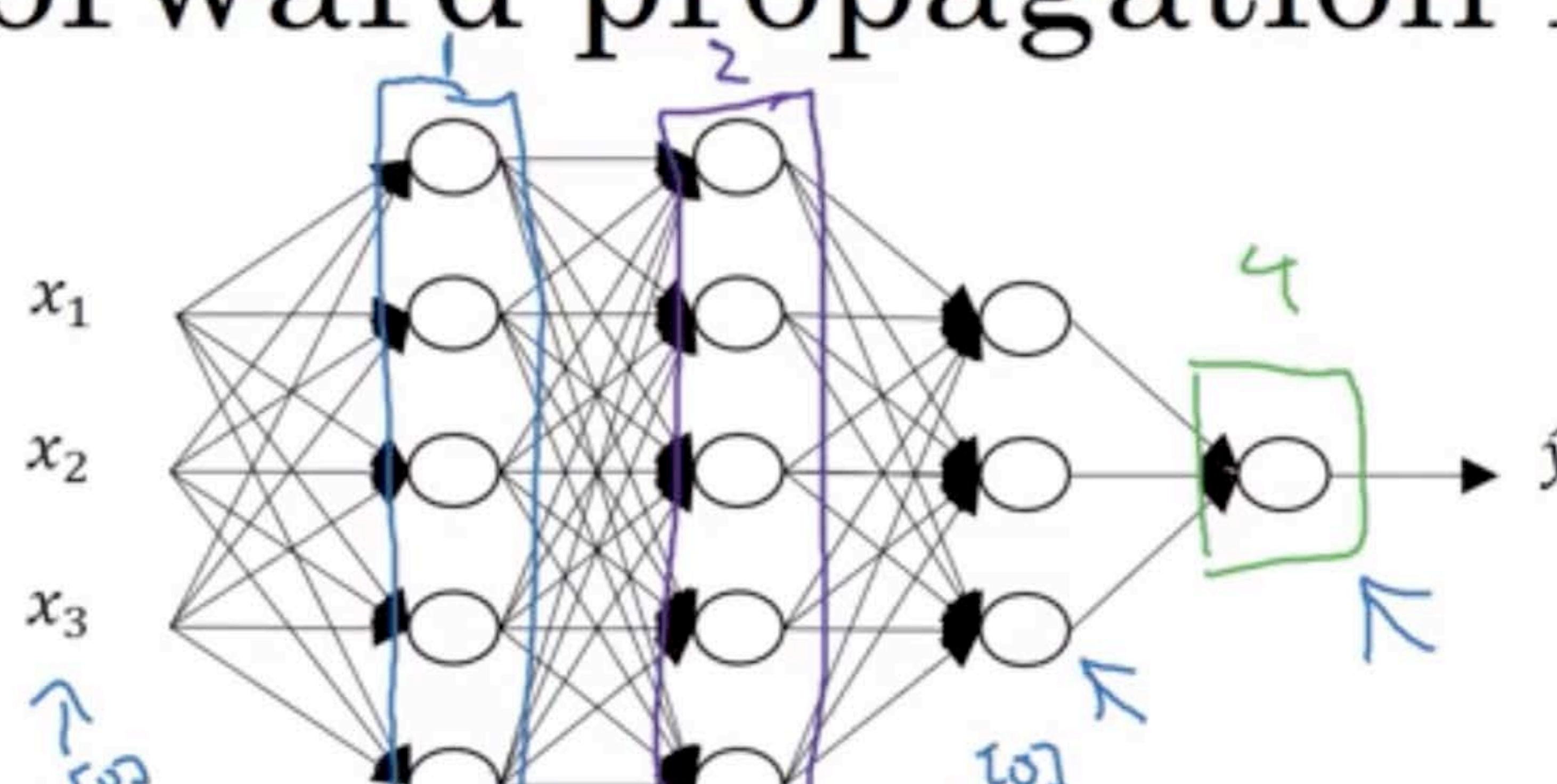


deeplearning.ai

Deep Neural Networks

Forward Propagation in a Deep Network

Forward propagation in a deep network



$$x : z^{[1]} = \underline{w^{[1]}} \underline{a^{[0]}} + \underline{b^{[1]}}$$

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

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

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

$A^{[0]} = X$

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

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

Vertebralized:

$$z^{[l]} = \underline{w^{[l]}} \underline{\cancel{A^{[0]}}} \underline{b^{[l]}}$$

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

$$\rightarrow z^{[2]} = \underline{w^{[2]}} \underline{A^{[1]}} + \underline{b^{[2]}}$$

$$\rightarrow A^{[2]} = g^{[2]}(z^{[2]})$$

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



deeplearning.ai

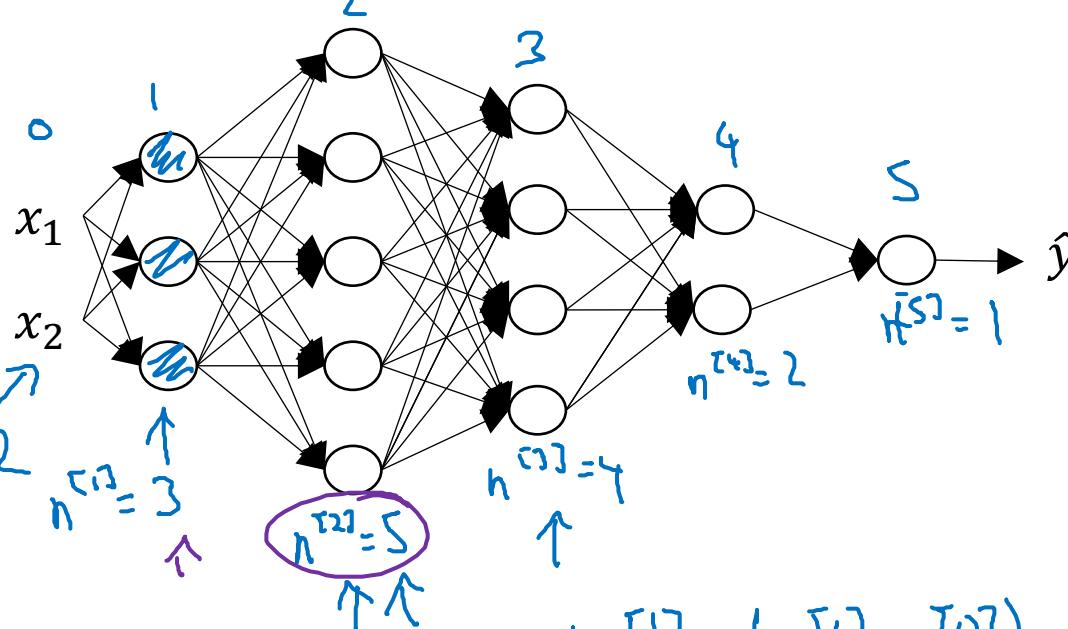
Deep Neural Networks

Getting your matrix dimensions right

Parameters $W^{[l]}$ and $b^{[l]}$

$$\begin{array}{c} \downarrow \\ z^{[0]} = g^{[0]}(a^{[0]}) \\ \uparrow \\ \theta^{[0]} \end{array}$$

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



$$\begin{array}{c} \downarrow \\ z^{[1]} = \boxed{\underline{W^{[1]}} \cdot x} + \boxed{\underline{b^{[1]}}} \\ (3,1) \leftarrow (3,2) \quad (2,1) \\ (\underline{n^{[1]}}, 1) \quad (\underline{n^{[1]}, n^{[2]}}) \quad (\underline{n^{[2]}}, 1) \end{array}$$

$$\begin{bmatrix} \vdots \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots & \vdots \\ \vdots & \vdots \end{bmatrix} \quad \begin{bmatrix} \vdots \\ \vdots \end{bmatrix}$$

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

$$W^{[2]}: (5, 3) \quad (n^{[2]}, n^{[1]})$$

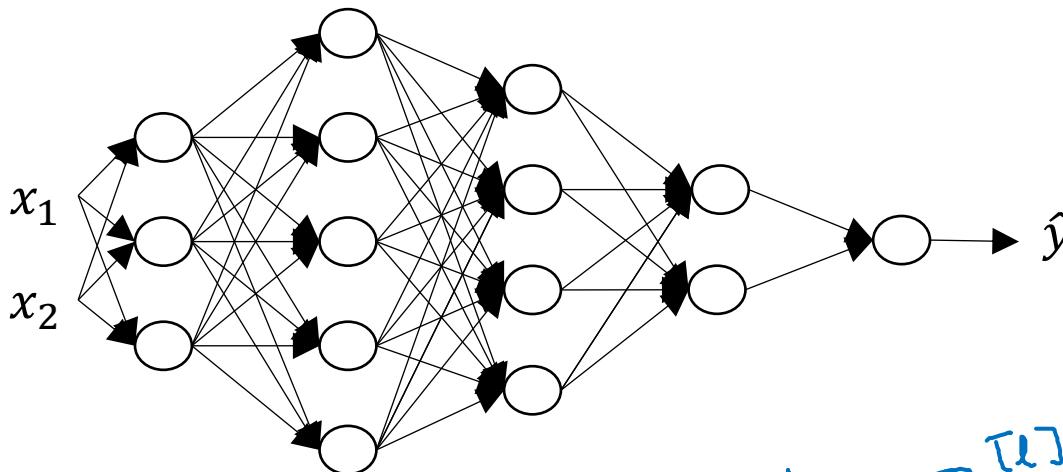
$$\begin{array}{c} z^{[2]} = \boxed{W^{[2]} \cdot a^{[1]}} + \boxed{b^{[2]}} \\ \uparrow \quad \uparrow \quad \uparrow \\ (5,1) \quad (5,3) \quad (2,1) \quad (5,1) \\ (\underline{n^{[2]}}, 1) \quad (\underline{n^{[2]}}, 1) \end{array}$$

$$W^{[3]}: (4, 5) \\ W^{[4]}: (2, 4) \quad , \quad W^{[5]}: (1, 2)$$

$$L=5$$

$$\begin{cases} W^{[L]}: (n^{[L]}, n^{[L-1]}) \\ b^{[L]}: (n^{[L]}, 1) \\ \delta W^{[L]}: (n^{[L]}, n^{[L-1]}) \\ \delta b^{[L]}: (n^{[L]}, 1) \end{cases}$$

Vectorized implementation



$$z^{[l]} = w^{[l]} \cdot x + b^{[l]}$$

$$(n^{[l]}, 1) \quad (n^{[l]}, n^{[l+1]}) \quad (n^{[l]}, 1)$$
$$\underbrace{\begin{bmatrix} z^{[l](1)} & z^{[l](2)} & \dots & z^{[l](m)} \end{bmatrix}}_{(n^{[l]}, m)}$$

$$\overbrace{\begin{bmatrix} z^{[l]} \\ \vdots \\ z^{[l]} \end{bmatrix}}^{\uparrow} = \underbrace{w^{[l]} \cdot X + b^{[l]}}_{\uparrow} \quad \underbrace{(n^{[l]}, 1)}_{(n^{[l]}, m)}$$

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

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

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

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

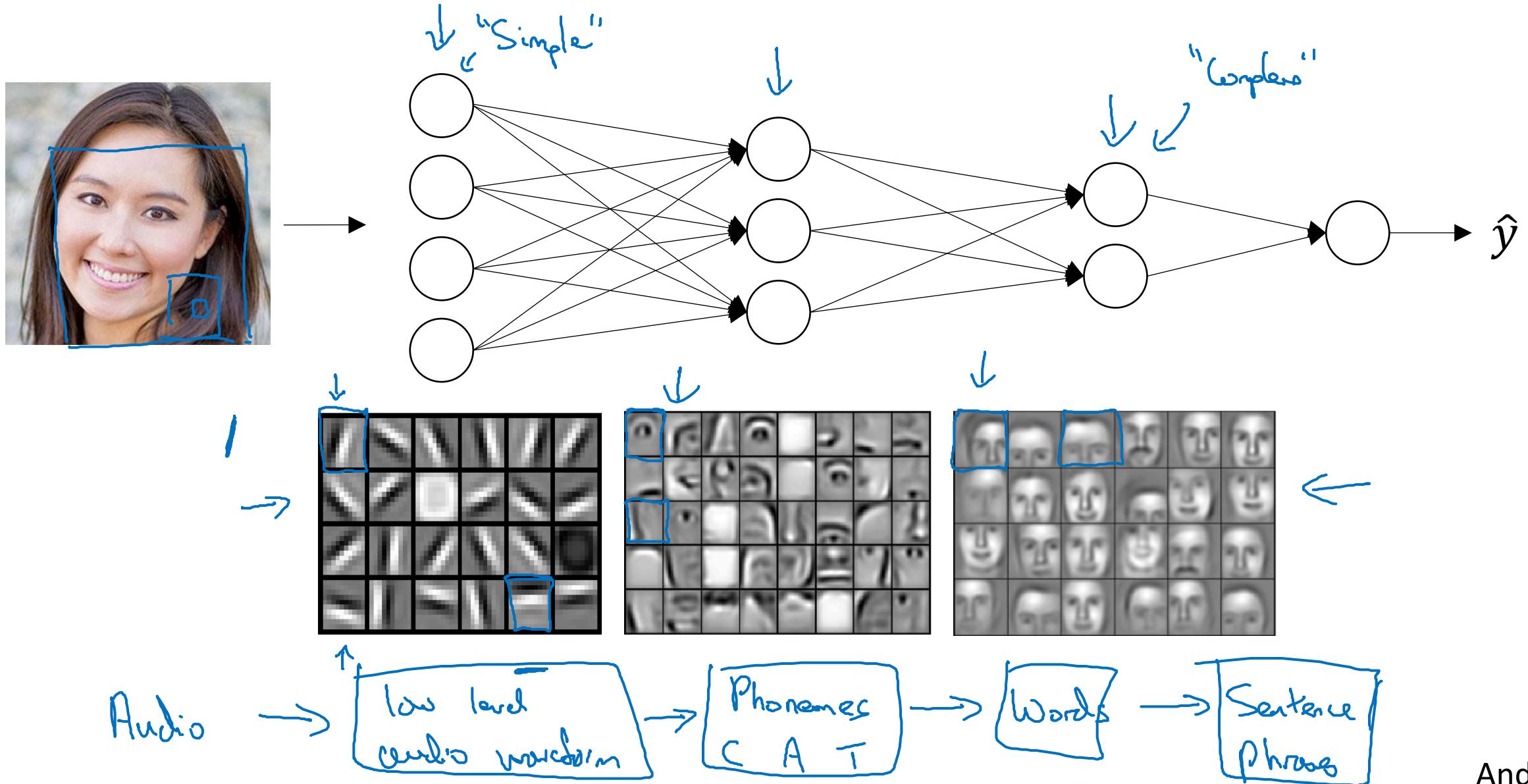


deeplearning.ai

Deep Neural Networks

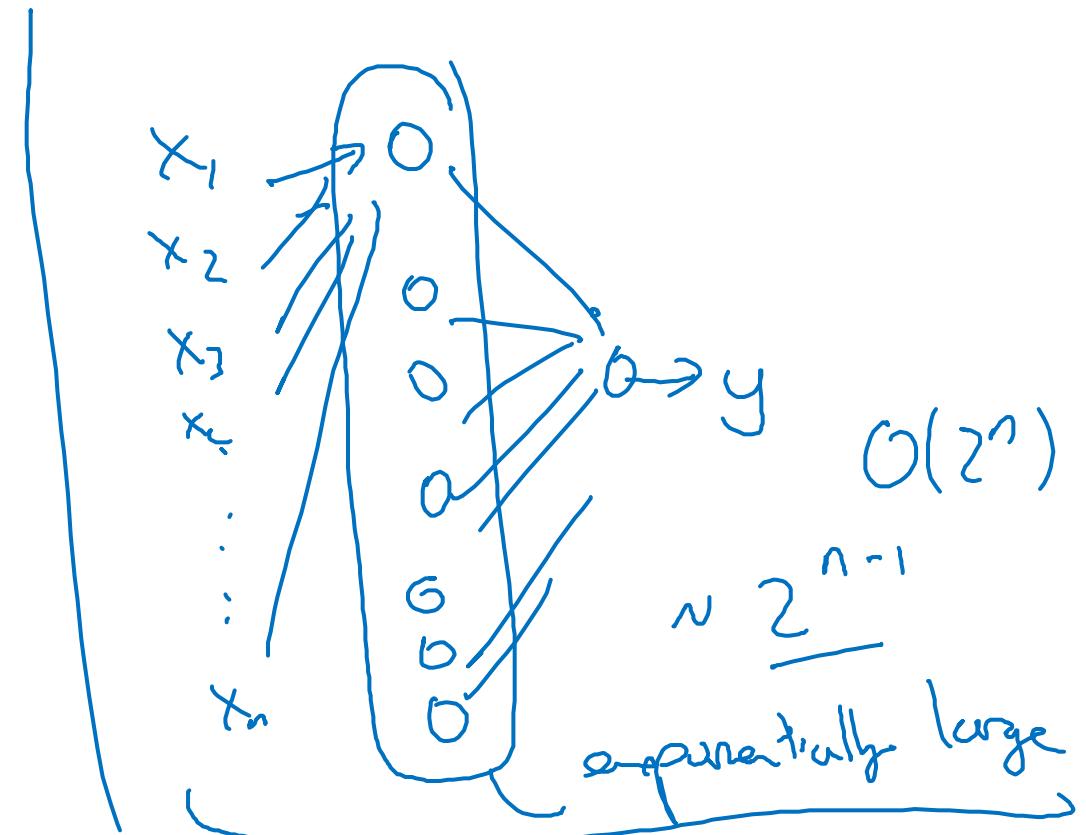
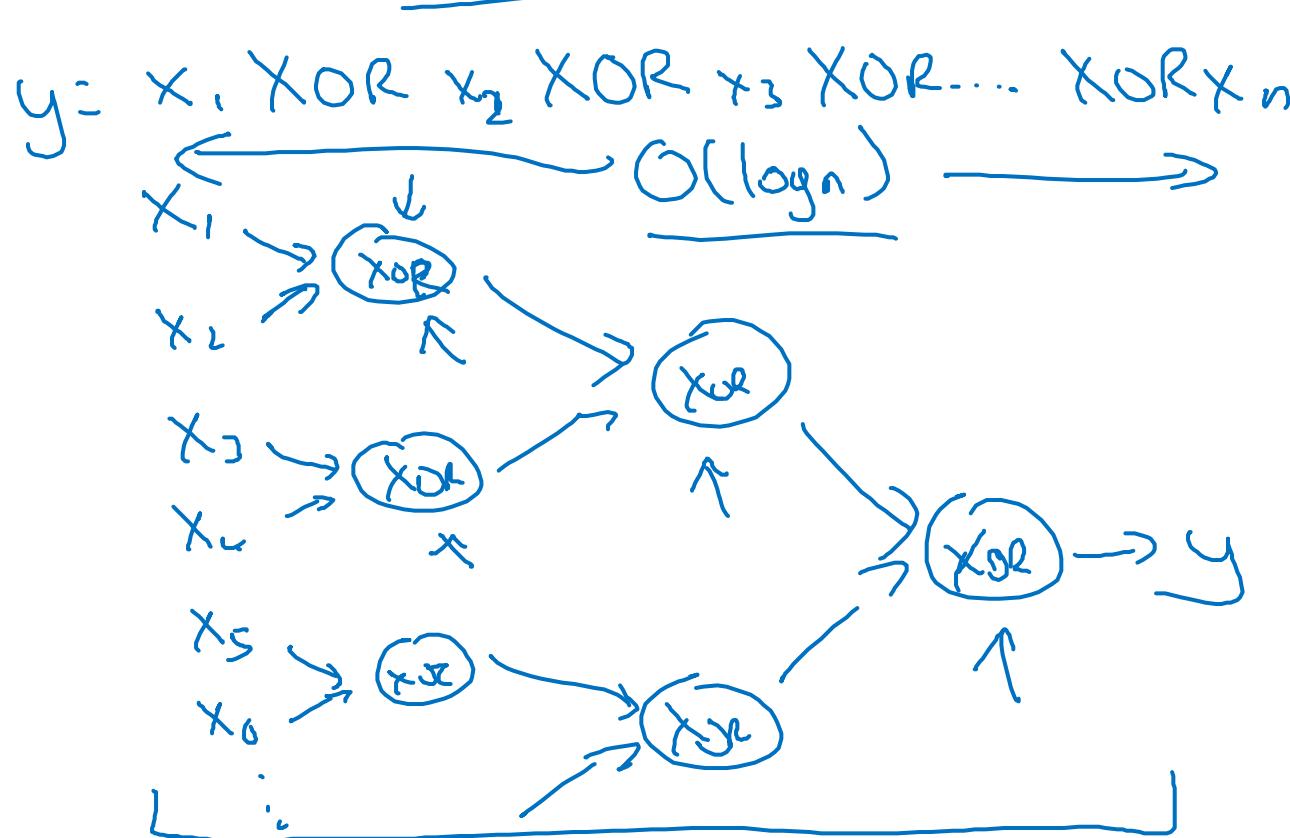
Why deep representations?

Intuition about deep representation



Circuit theory and deep learning

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.



Andrew Ng

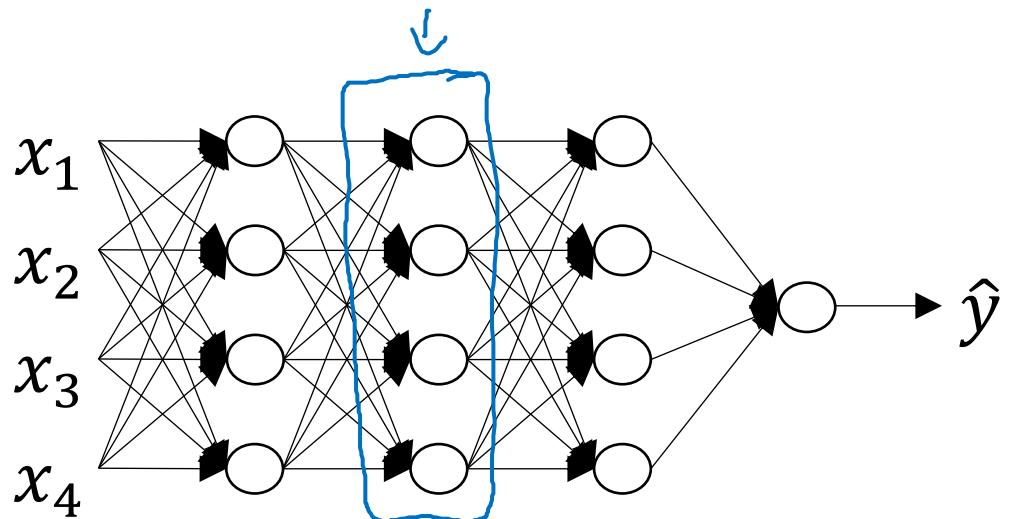


deeplearning.ai

Deep Neural Networks

Building blocks of
deep neural networks

Forward and backward functions

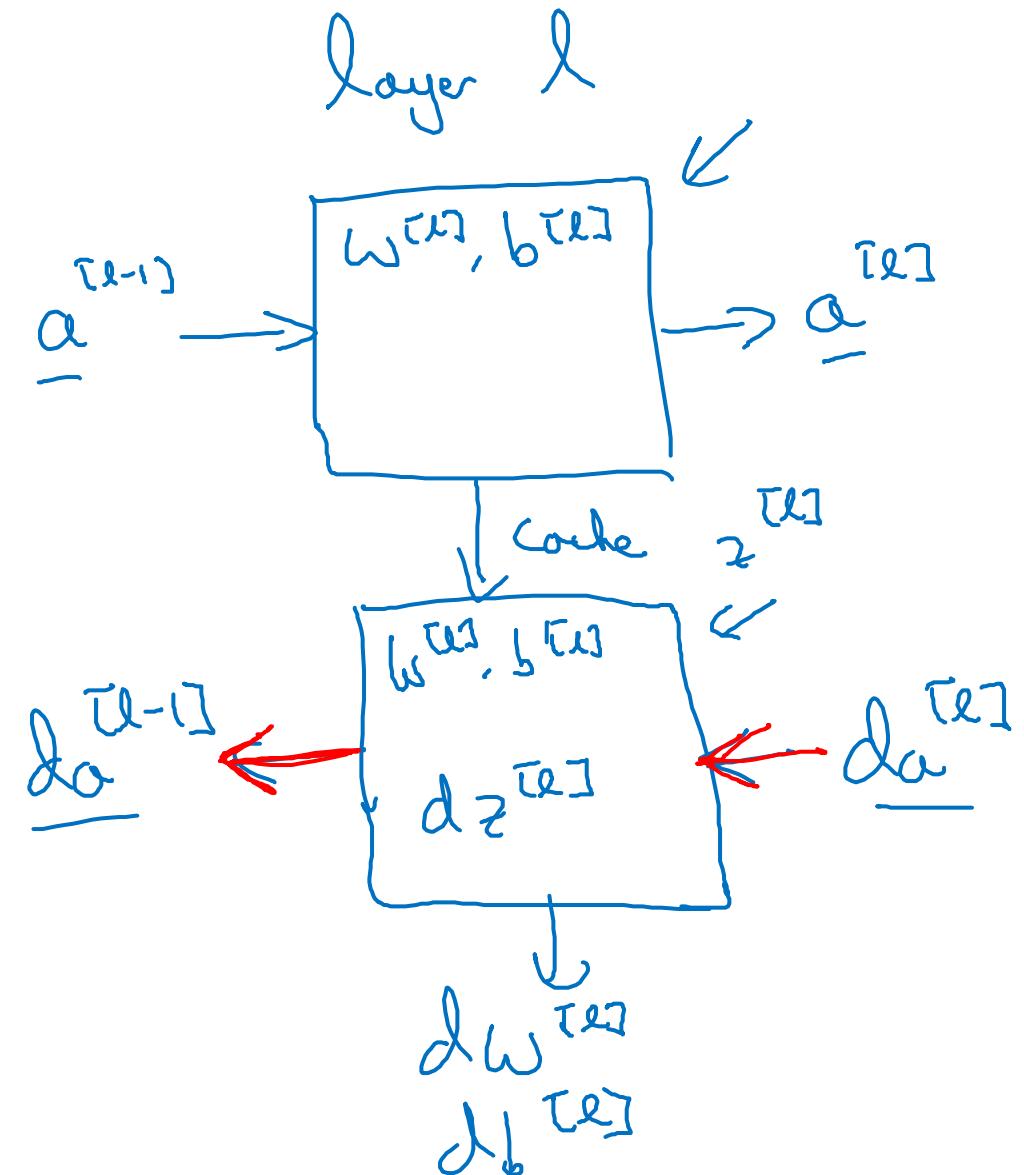


layer l : $w^{[l]}, b^{[l]}$

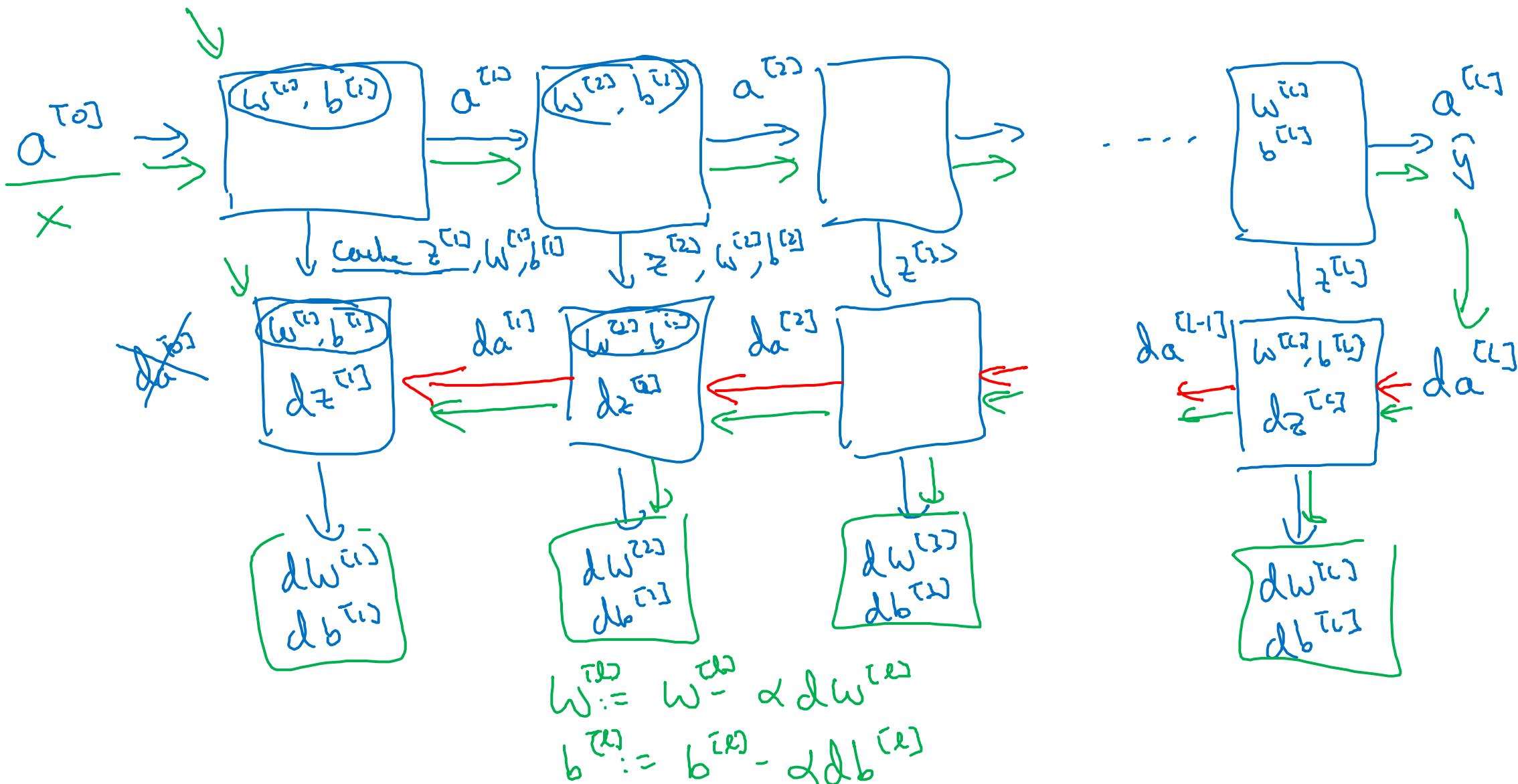
→ Forward: Input $a^{[l-1]}$, output $a^{[l]}$

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

→ Backward: Input $da^{[l]}$, output $\begin{matrix} da^{[l-1]} \\ dw^{[l]} \\ db^{[l]} \end{matrix}$



Forward and backward functions





deeplearning.ai

Deep Neural Networks

Forward and backward propagation

Forward propagation for layer l

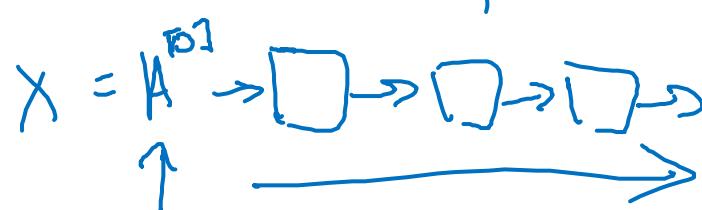
→ Input $a^{[l-1]} \leftarrow$

→ Output $a^{[l]}$, cache ($\underline{z^{[l]}}$)

$$z^{[l]} = w^{[l]} \cdot a^{[l-1]} + b^{[l]}$$

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

$$\begin{matrix} a^{[0]} \\ A^{[0]} \end{matrix}$$



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

Vertwijf:

$$z^{[l]} = w^{[l]} \cdot A^{[l-1]} + b^{[l]}$$

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

Backward propagation for layer l

→ Input $da^{[l]}$

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

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

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

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

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

$$dz^{[l]} = W^{[l+1]} \cdot dz^{[l+1]} * g'(z^{[l]})$$

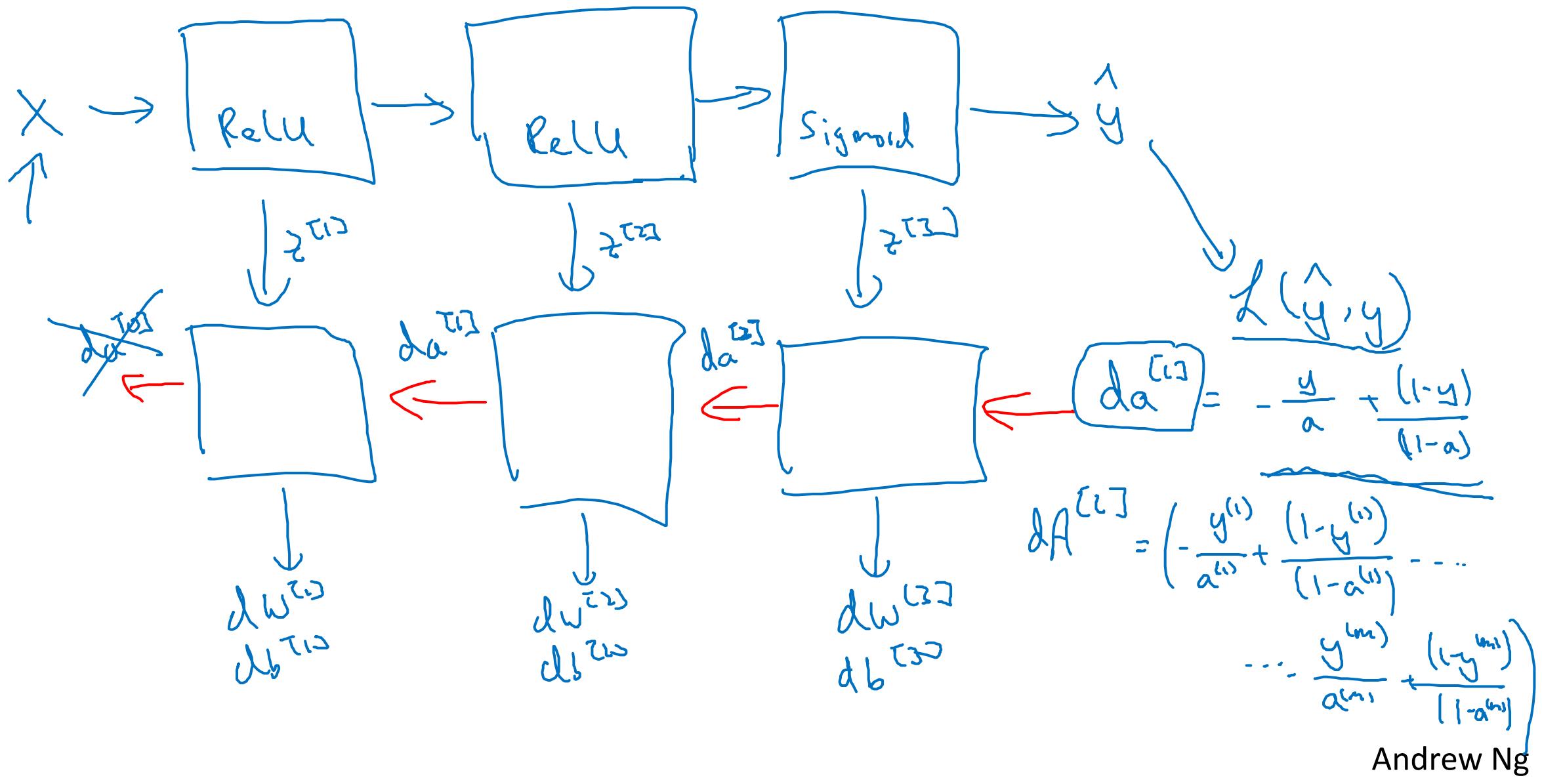
$$dz^{[l]} = da^{[l]} * g'(z^{[l]})$$

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

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

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

Summary





deeplearning.ai

Deep Neural Networks

Forward and backward propagation

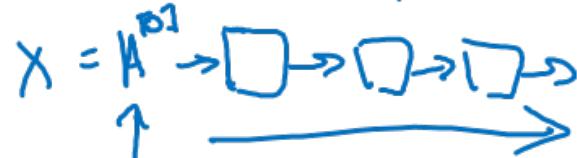
Forward propagation for layer l



$$z^{(l)} = w^{(l)} \cdot a^{(l-1)} + b^{(l)}$$

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

$$\begin{matrix} a^{(0)} \\ A^{(0)} \end{matrix}$$



Verteilung:

$$z^{(l)} = w^{(l)} \cdot A^{(l-1)} + b^{(l)}$$

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

Backward propagation for layer l

Diagram showing forward pass flow:

\rightarrow \rightarrow \rightarrow

Forward pass equations:

$$\underline{dz^{[l]}} = \text{do}^{[l]} * g^{[l]}(z^{[l]})$$

$$\underline{dw^{[l]}} = \underline{dz^{[l]}} \cdot \underline{a^{[l-1]}}$$

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

$$\boxed{\underline{da^{[l-1]}}} = \underline{w^{[l]T}} \cdot \underline{dz^{[l]}}$$

$$\underline{dz^{[l-1]}} = \underline{w^{[l-1]}} \cdot \underline{dz^{[l]}} * g^{[l-1]}(z^{[l]})$$

Backward pass equations:

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

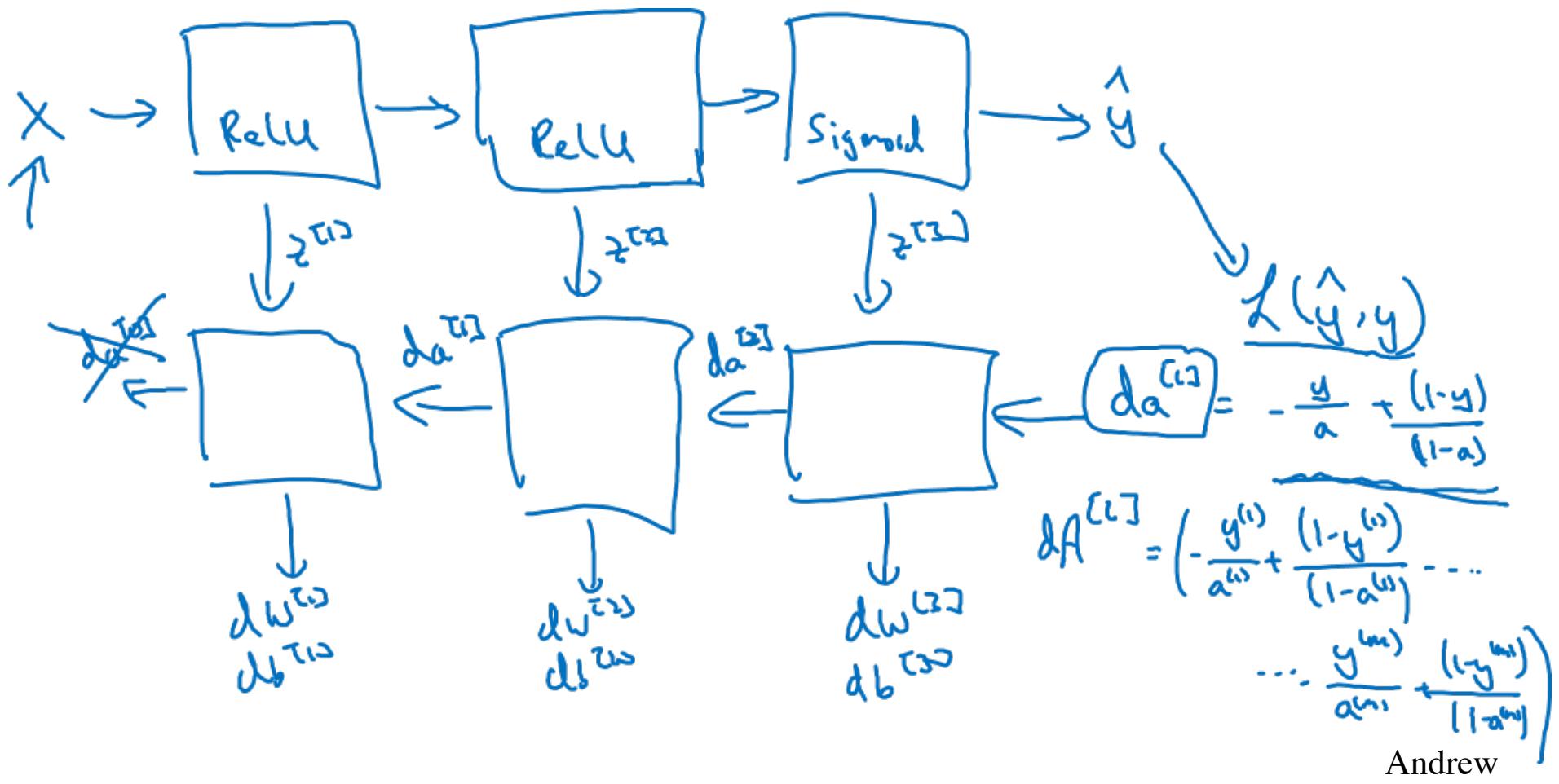
$$dw^{[l]} = \frac{1}{m} dz^{[l]} \cdot A^{[l-1]T}$$

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

$$\boxed{dA^{[l-1]}} = \underline{w^{[l]T}} \cdot \underline{dz^{[l]}}$$

Andrew

Summary



Andrew



deeplearning.ai

Deep Neural Networks

Parameters vs Hyperparameters

What are hyperparameters?

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

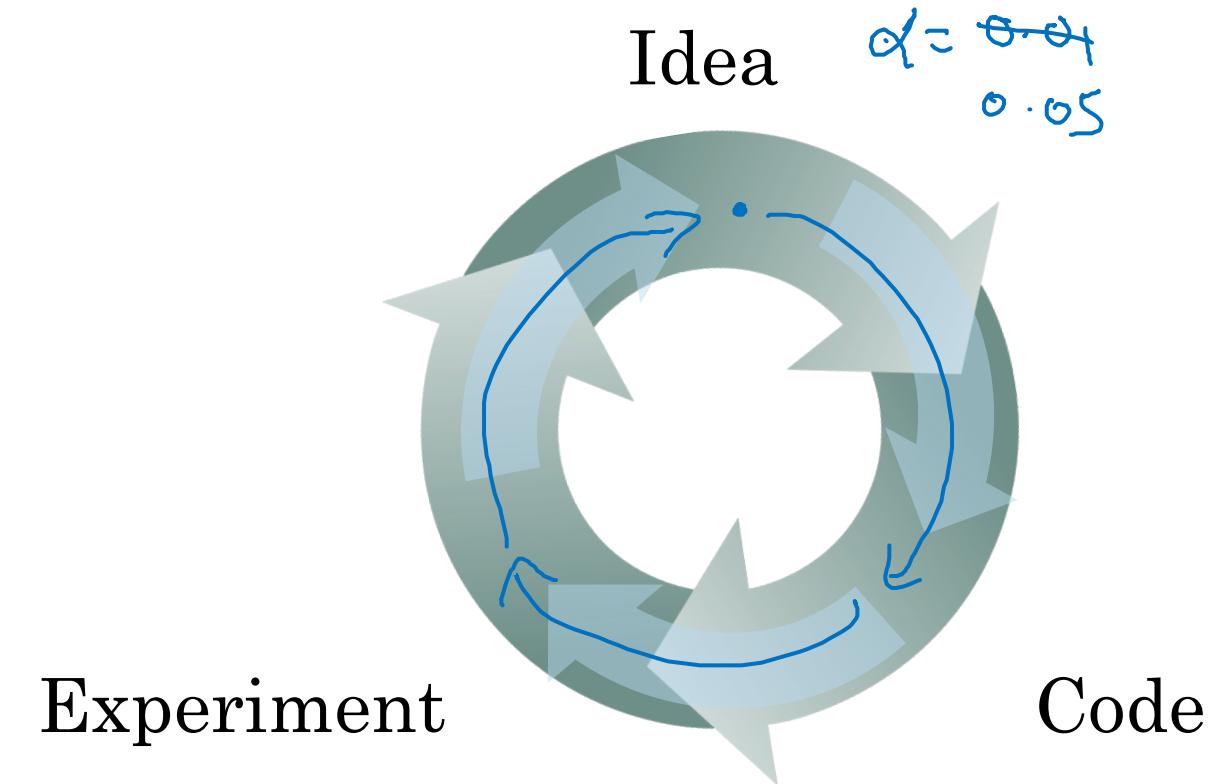
Hyperparameters:

- learning rate $\frac{\alpha}{n}$
- #iterations
- #hidden layers L
- #hidden units $n^{[1]}, n^{[2]}, \dots$
- choice of activation function

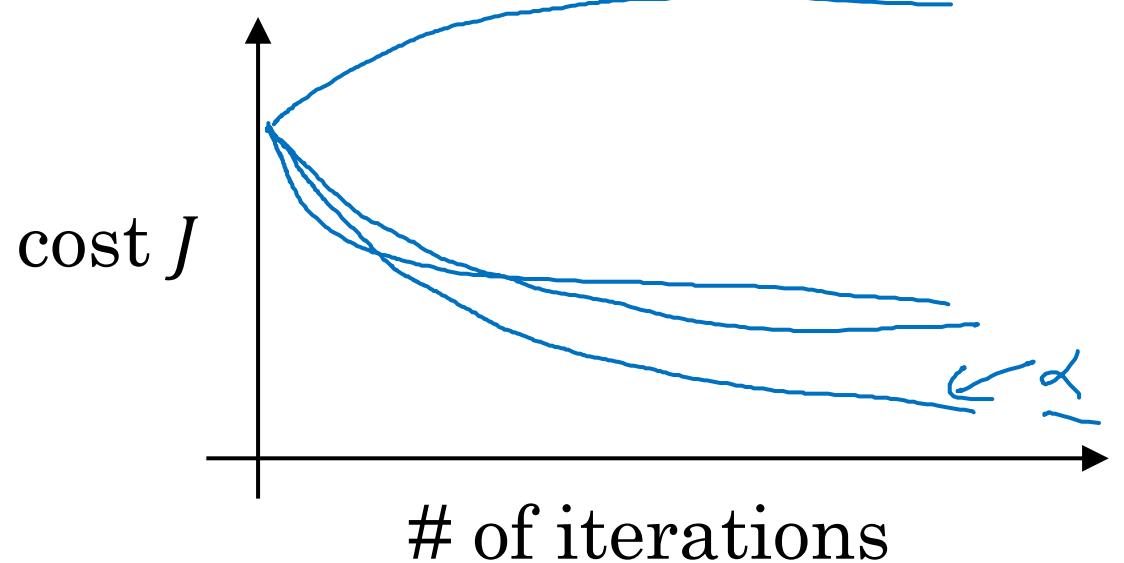


Later: Momentum, minibatch size, regularizations, ...

Applied deep learning is a very empirical process



Vision, Speech, NLP, Ad, Search, Reinforcement.





deeplearning.ai

Deep Neural Networks

What does this
have to do with
the brain?

Forward and backward propagation

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

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

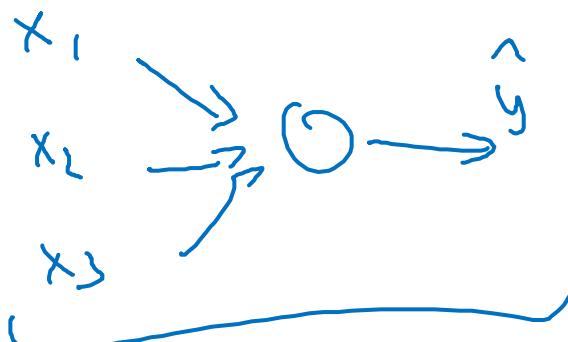
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

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

:

$$A^{[L]} = g^{[L]}(Z^{[L]}) = \hat{Y}$$

"It's like the brain"



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

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

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

$$dZ^{[L-1]} = dW^{[L]T} dZ^{[L]} g'^{[L]}(Z^{[L-1]})$$

:

$$dZ^{[1]} = dW^{[L]T} dZ^{[2]} g'^{[1]}(Z^{[1]})$$

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

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

