

Copyright Notice

These slides are distributed under the Creative Commons License.

[DeepLearning.AI](#) makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite [DeepLearning.AI](#) as the source of the slides.

For the rest of the details of the license, see <https://creativecommons.org/licenses/by-sa/2.0/legalcode>



deeplearning.ai

Introduction to Deep Learning

Welcome



- AI is the new Electricity
- Electricity had once transformed countless industries: transportation, manufacturing, healthcare, communications, and more
- AI will now bring about an equally big transformation.

What you'll learn



Courses in this sequence (Specialization):

1. Neural Networks and Deep Learning
2. Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization
3. Structuring your Machine Learning project
4. Convolutional Neural Networks CNN
5. Natural Language Processing: Building sequence models

train / dev / test
end-to-end

RNN, LSTM

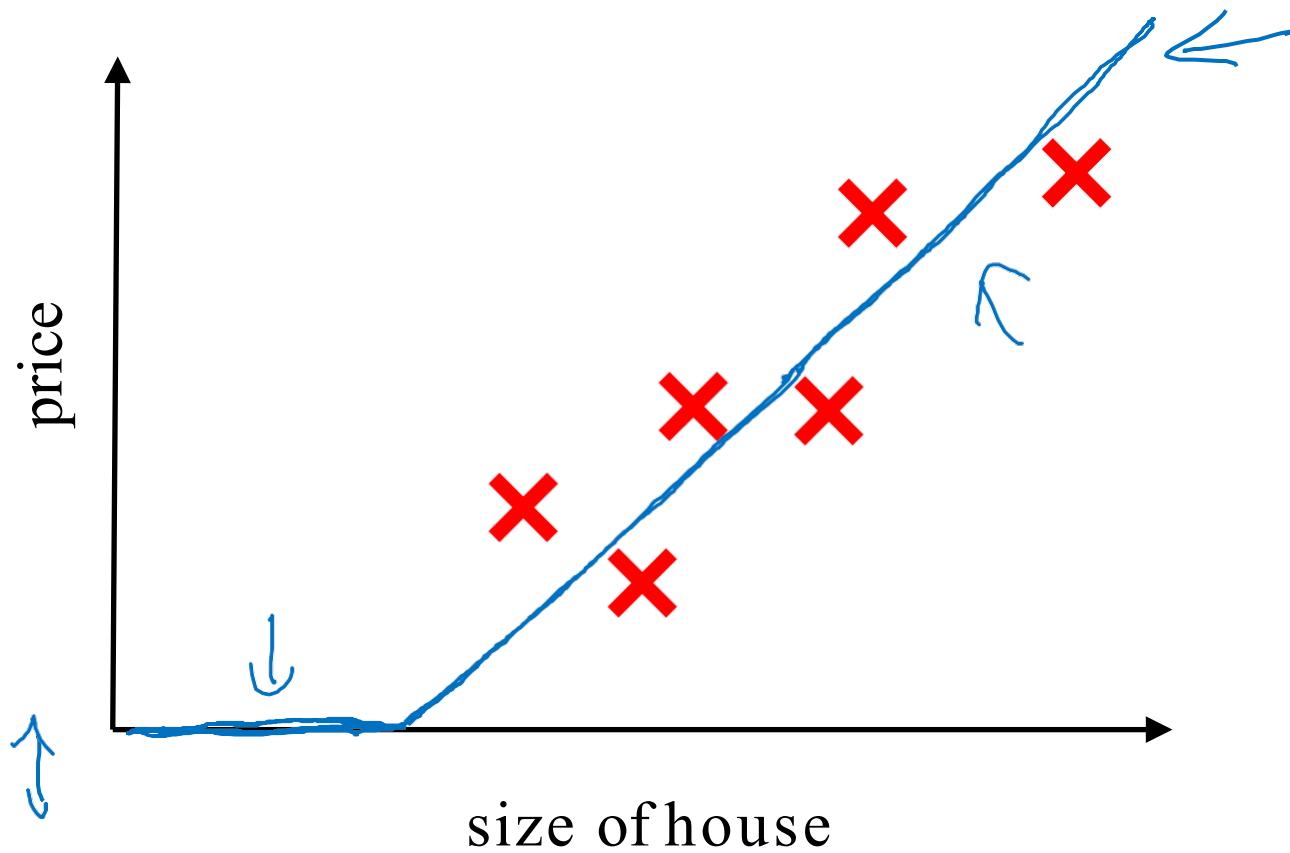


deeplearning.ai

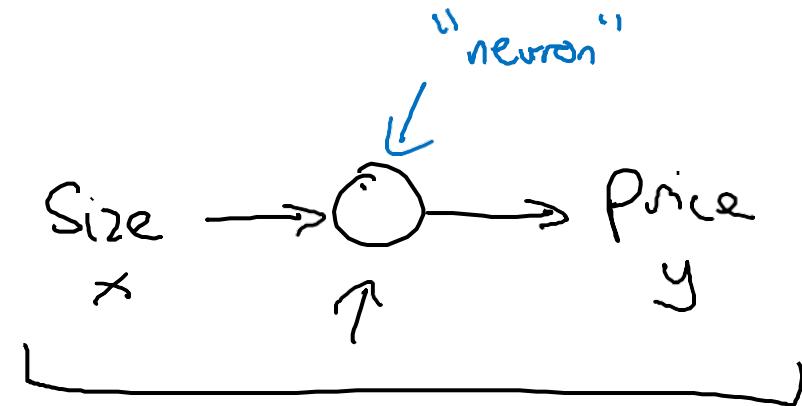
Introduction to Deep Learning

What is a Neural Network?

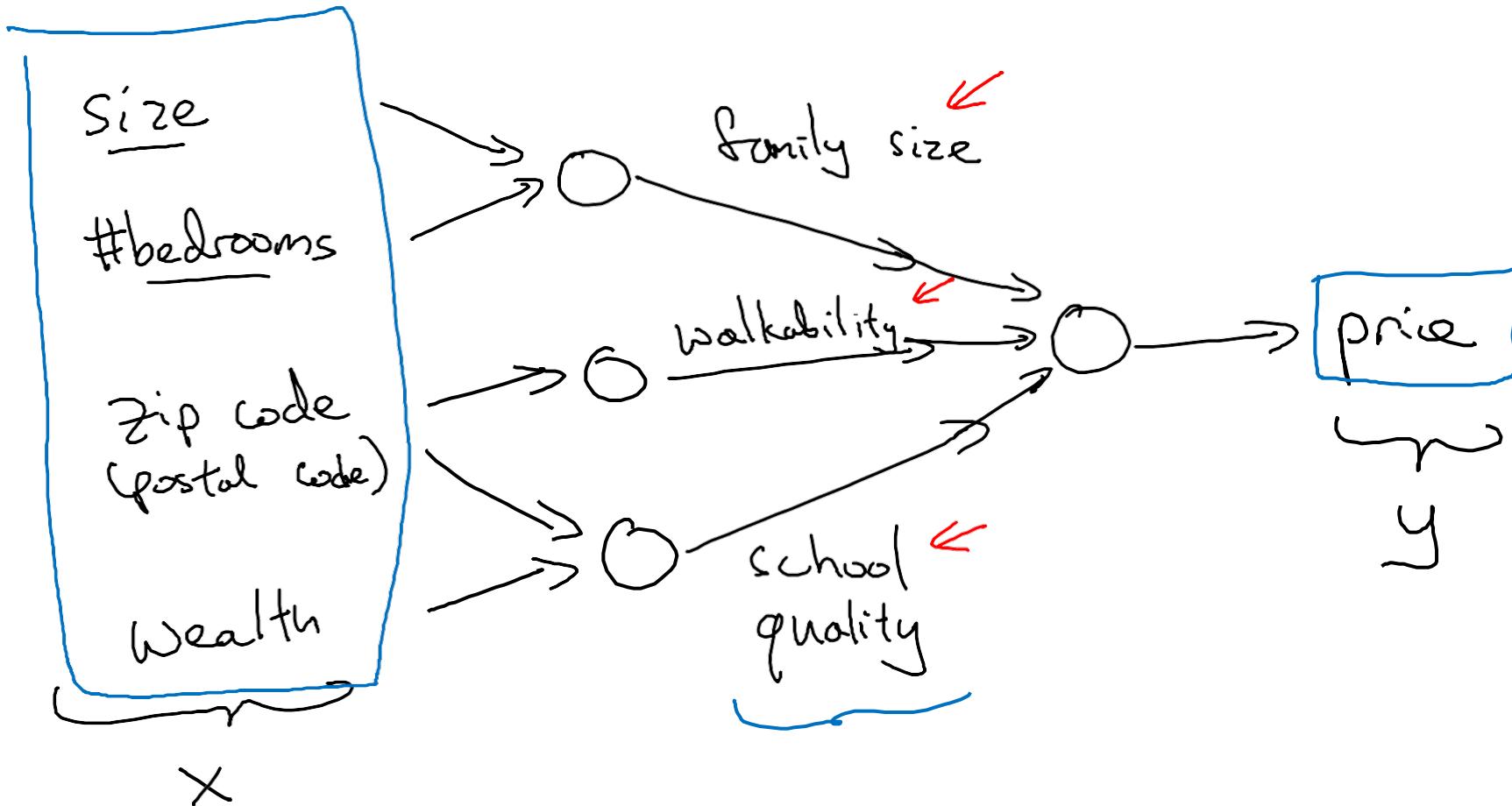
Housing Price Prediction



ReLU
Rectified
Linear
Unit

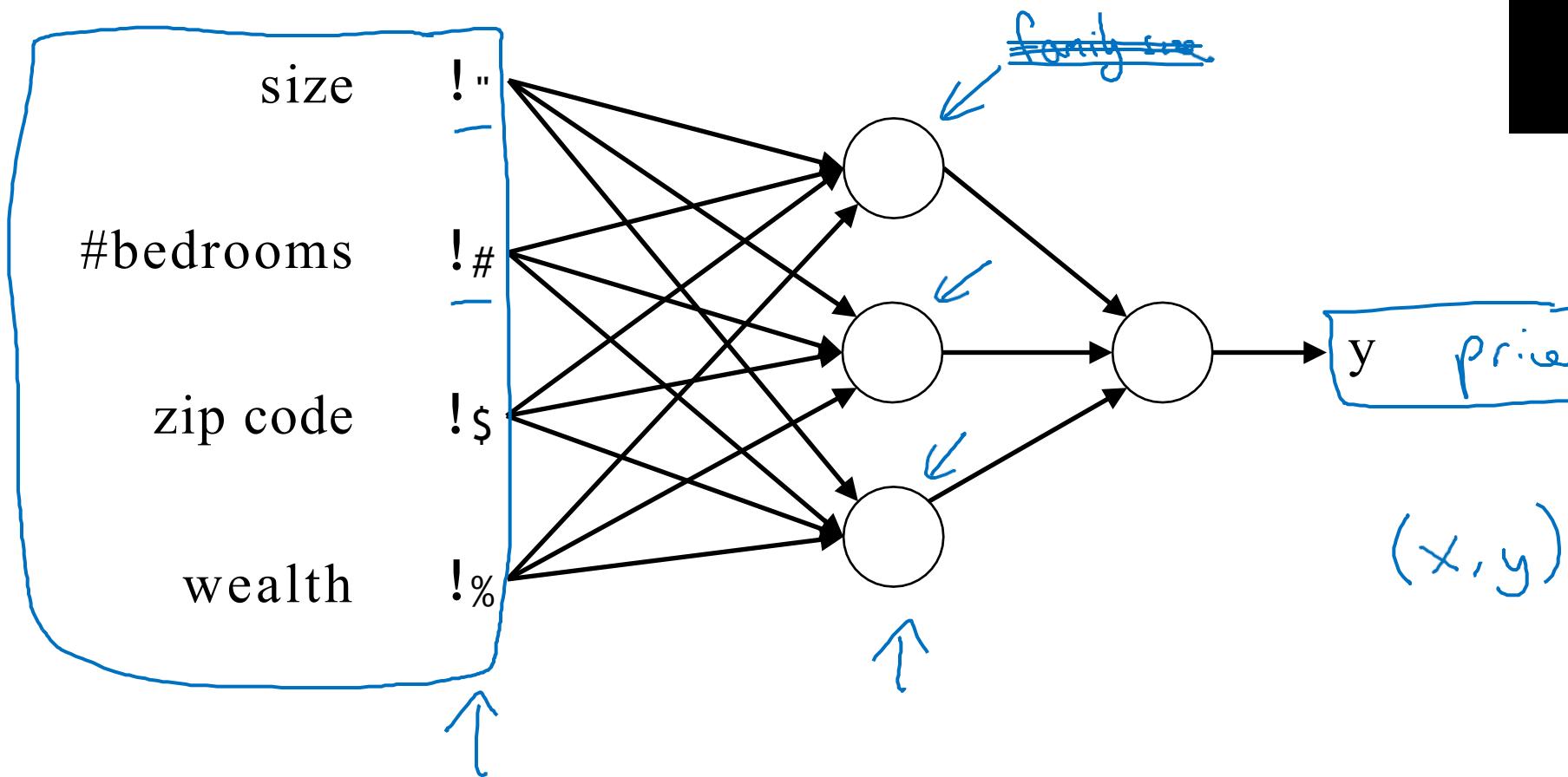


Housing Price Prediction



Housing Price Prediction

Drawing of
previous Image





deeplearning.ai

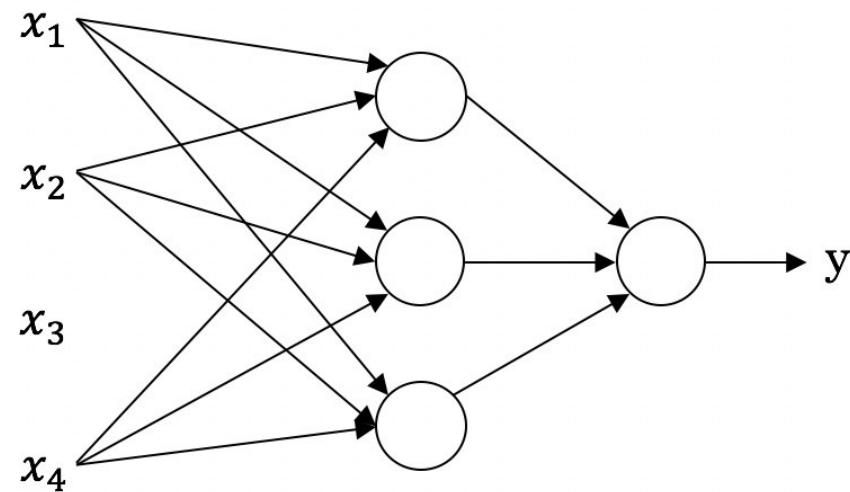
Introduction to Deep Learning

Supervised Learning with Neural Networks

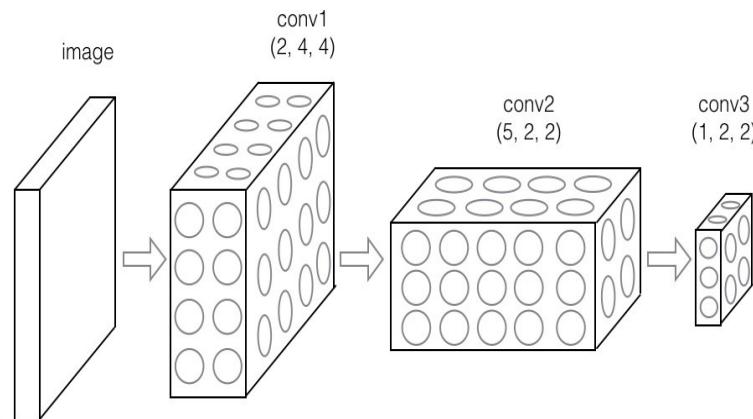
Supervised Learning

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
English	Chinese	Machine translation
Image, Radar info	Position of other cars	Autonomous driving

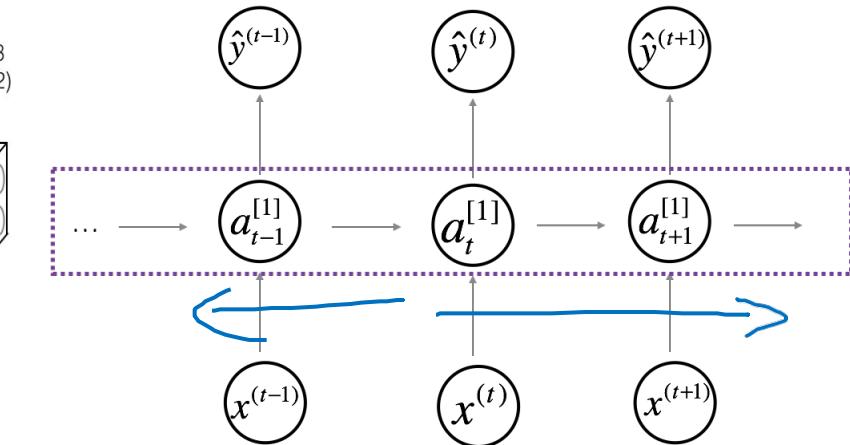
Neural Network examples



Standard NN



Convolutional NN



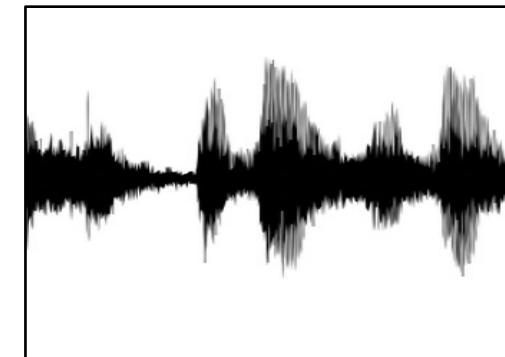
Recurrent NN

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

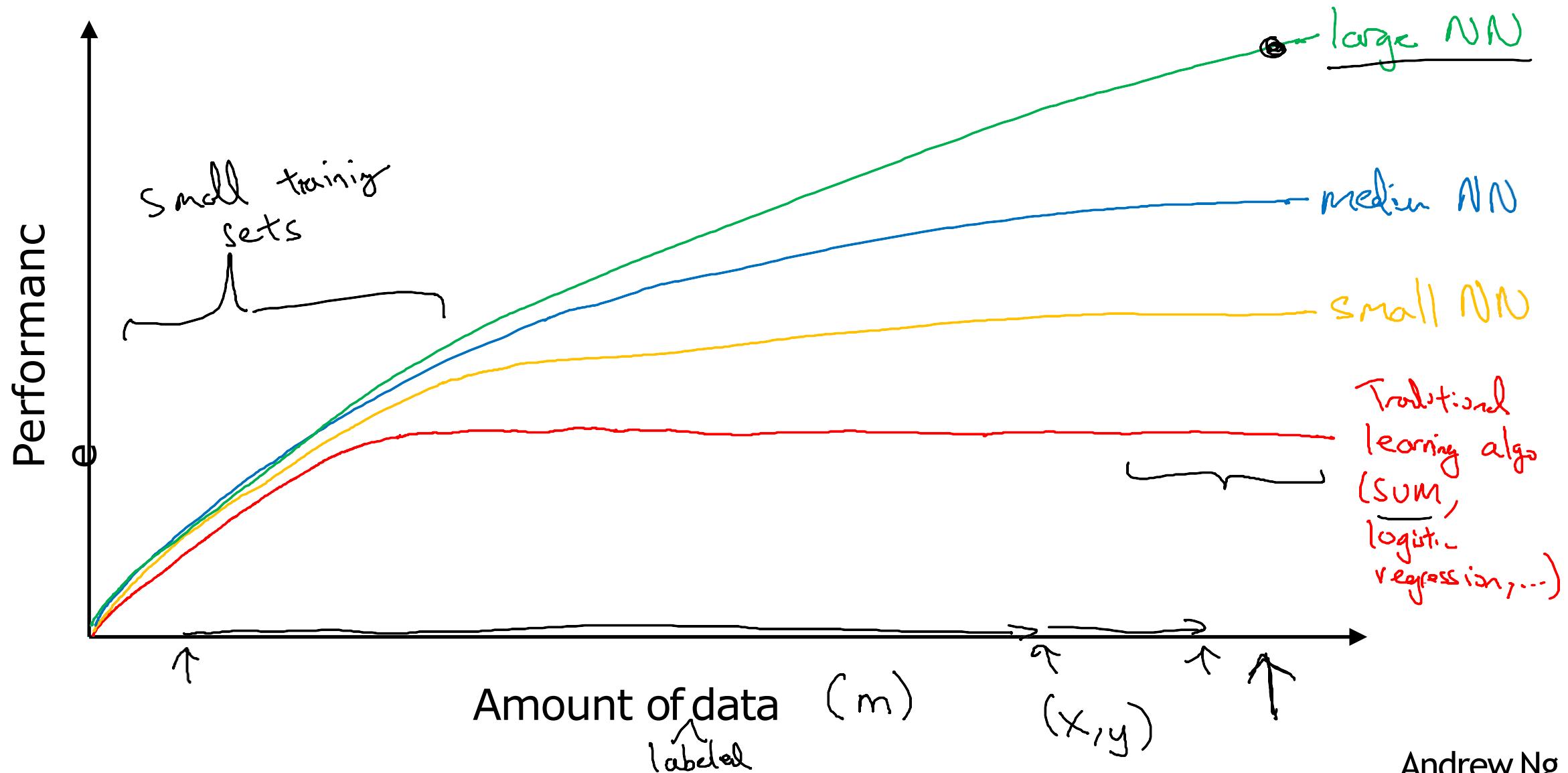


deeplearning.ai

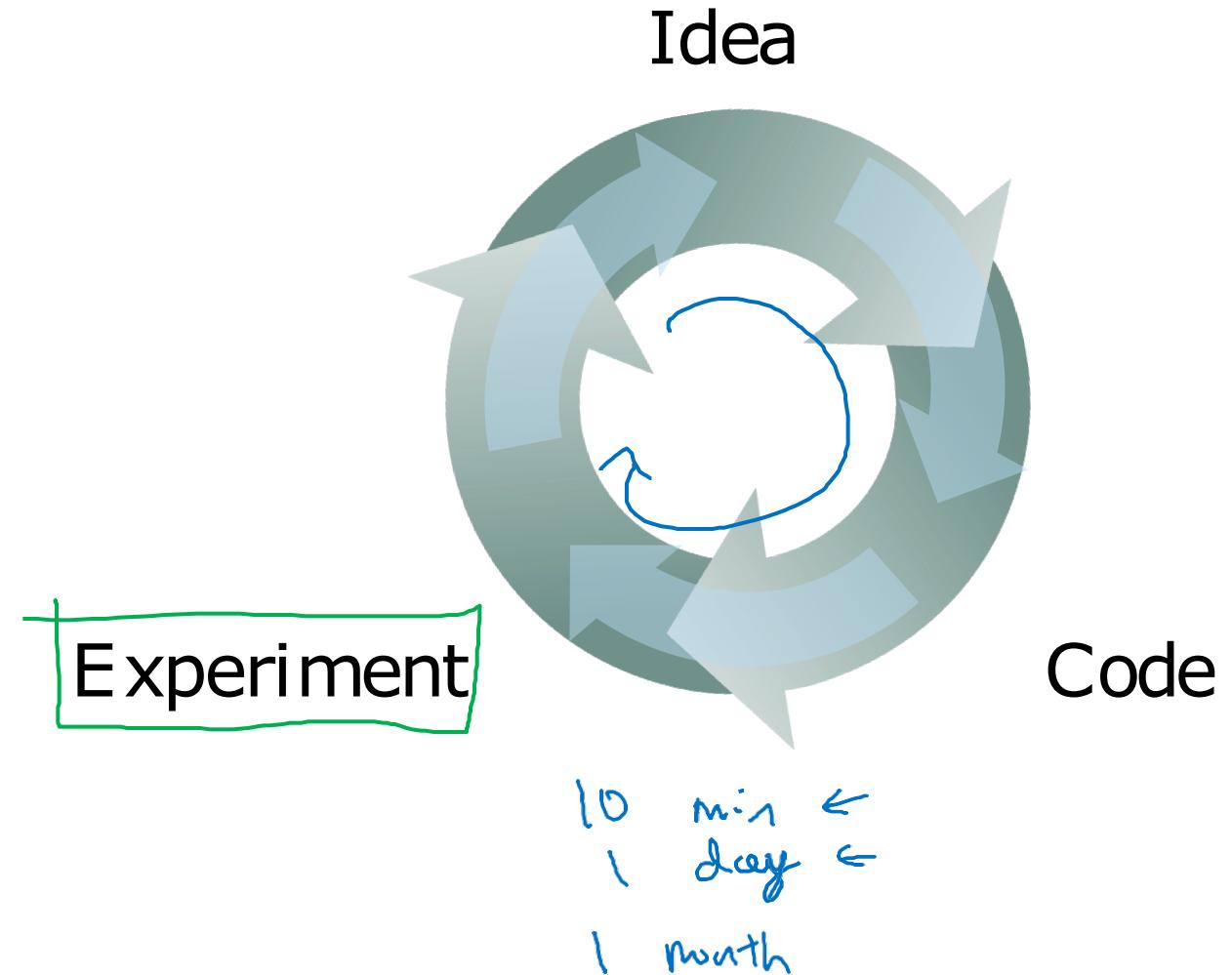
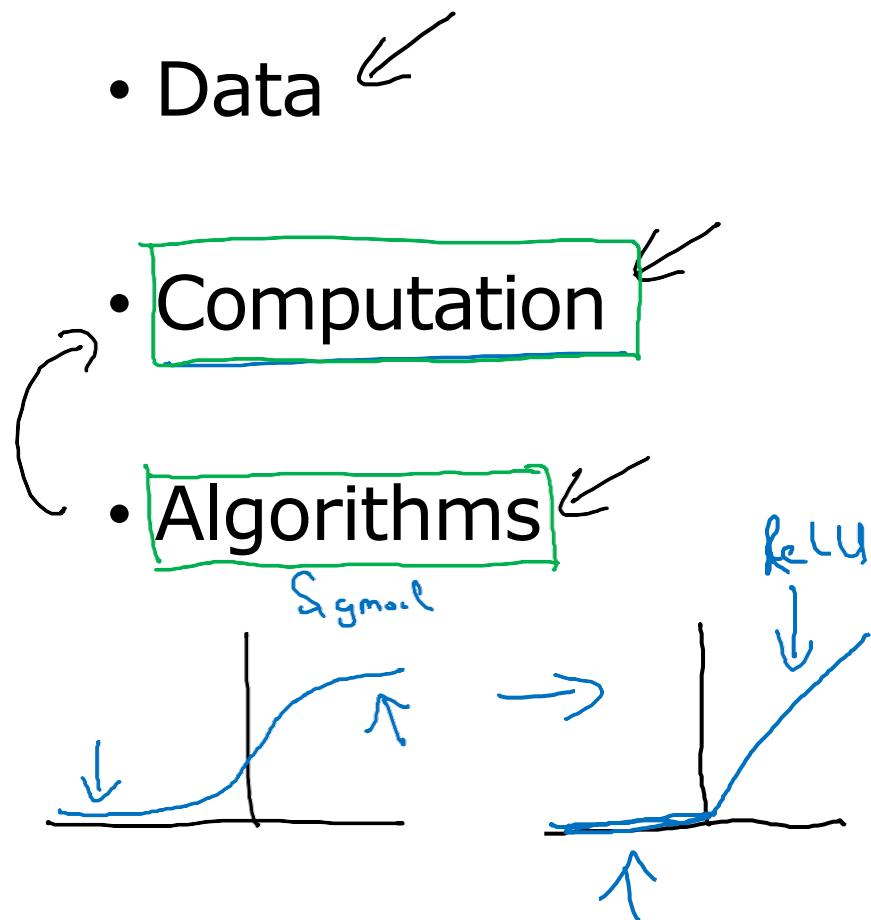
Introduction to Neural Networks

Why is Deep Learning taking off?

Scale drives deep learning progress



Scale drives deep learning progress





deeplearning.ai

Introduction to Neural Networks

About this Course

Courses in this Specialization

1. Neural Networks and DeepLearning ←
2. Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization
3. Structuring your Machine Learning project
4. Convolutional Neural Networks
5. Natural Language Processing: Building sequence models

Outline of this Course

Week 1: Introduction

Week 2: Basics of Neural Network programming

Week 3: One hidden layer Neural Networks

Week 4: Deep Neural Networks

Copyright Notice

These slides are distributed under the Creative Commons License.

[DeepLearning.AI](#) makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite [DeepLearning.AI](#) as the source of the slides.

For the rest of the details of the license, see <https://creativecommons.org/licenses/by-sa/2.0/legalcode>



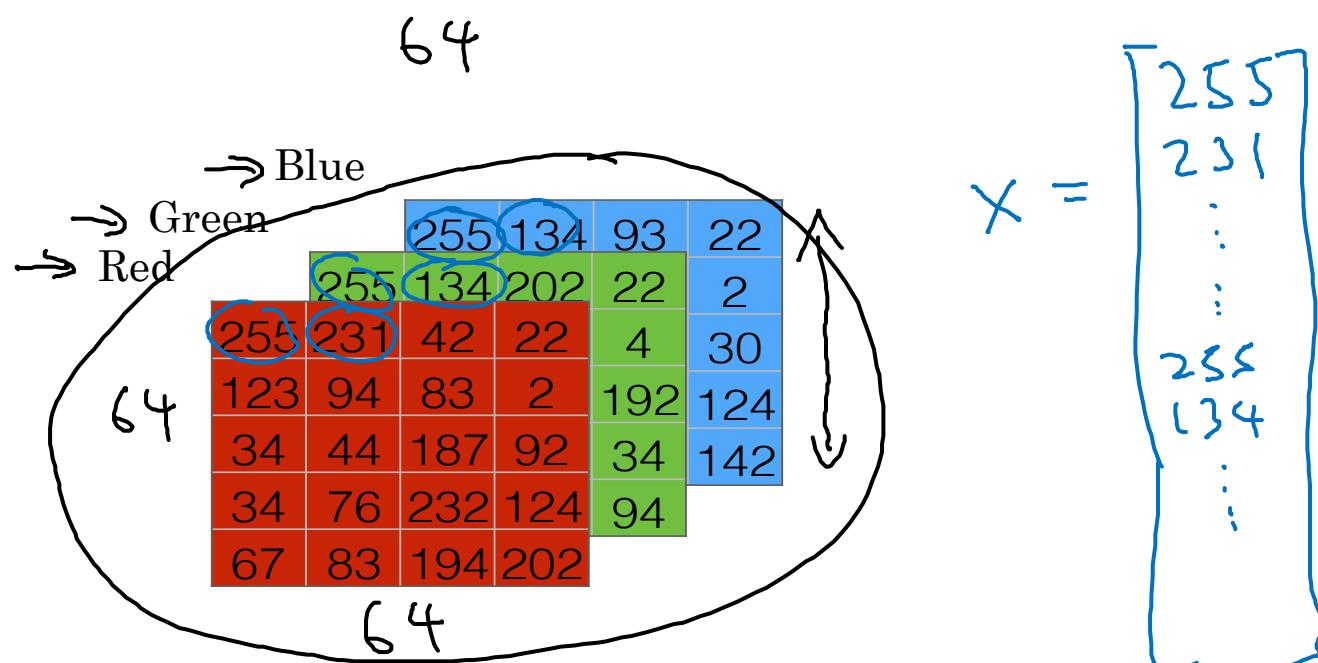
deeplearning.ai

Basics of Neural Network Programming

Binary Classification

Binary Classification

64 →  → 1 (cat) vs 0 (non cat)



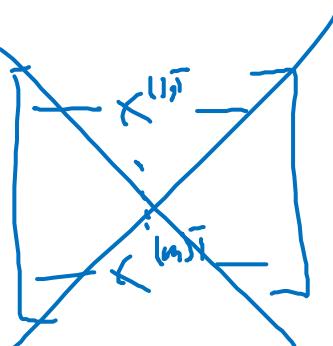
$$64 \times 64 \times 3 = 12288$$

$$n = n_x = 12288$$

$$X \rightarrow y$$

Notation

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

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

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

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



deeplearning.ai

Basics of Neural Network Programming

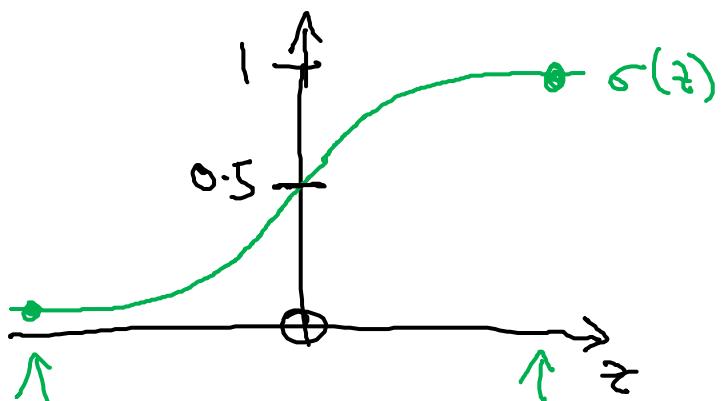
Logistic Regression

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



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

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

$$w = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_{n_x} \end{bmatrix} \quad \left. \begin{array}{l} \{w_0\} \rightarrow b \\ \{w_1, w_2, \dots, w_{n_x}\} \rightarrow w \end{array} \right.$$

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

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 $\hat{y}^{(i)} \approx y^{(i)}$.

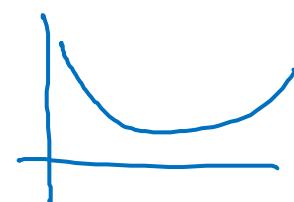
Loss (error) function:

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

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

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

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

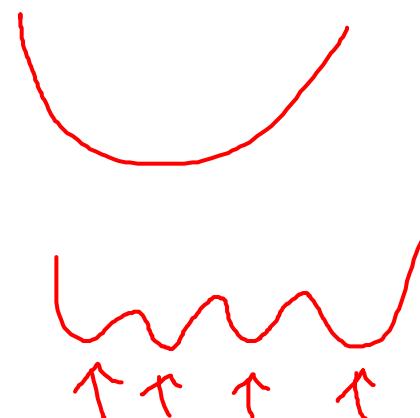
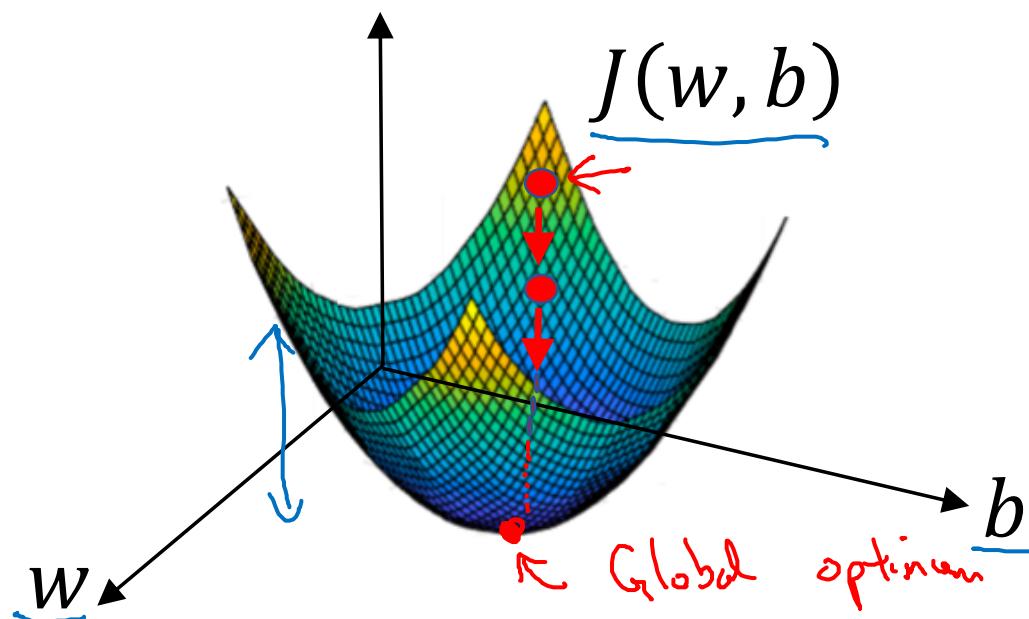
Gradient Descent

Gradient Descent

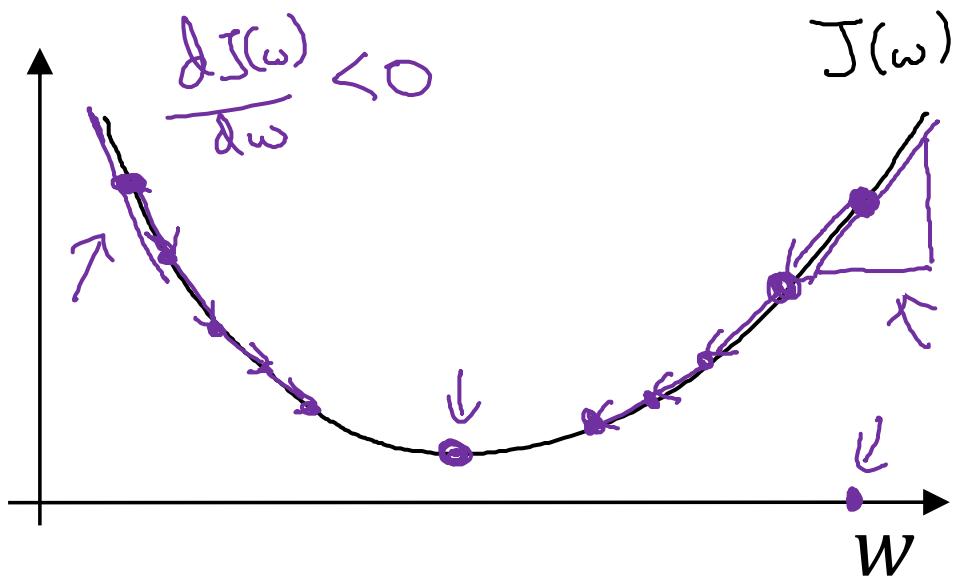
Recap: $\hat{y} = \sigma(w^T x + b)$, $\sigma(z) = \frac{1}{1+e^{-z}}$ 

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

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

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

$$J(\omega, b)$$

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

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

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

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

"partial derivative" J

$d\omega$

db

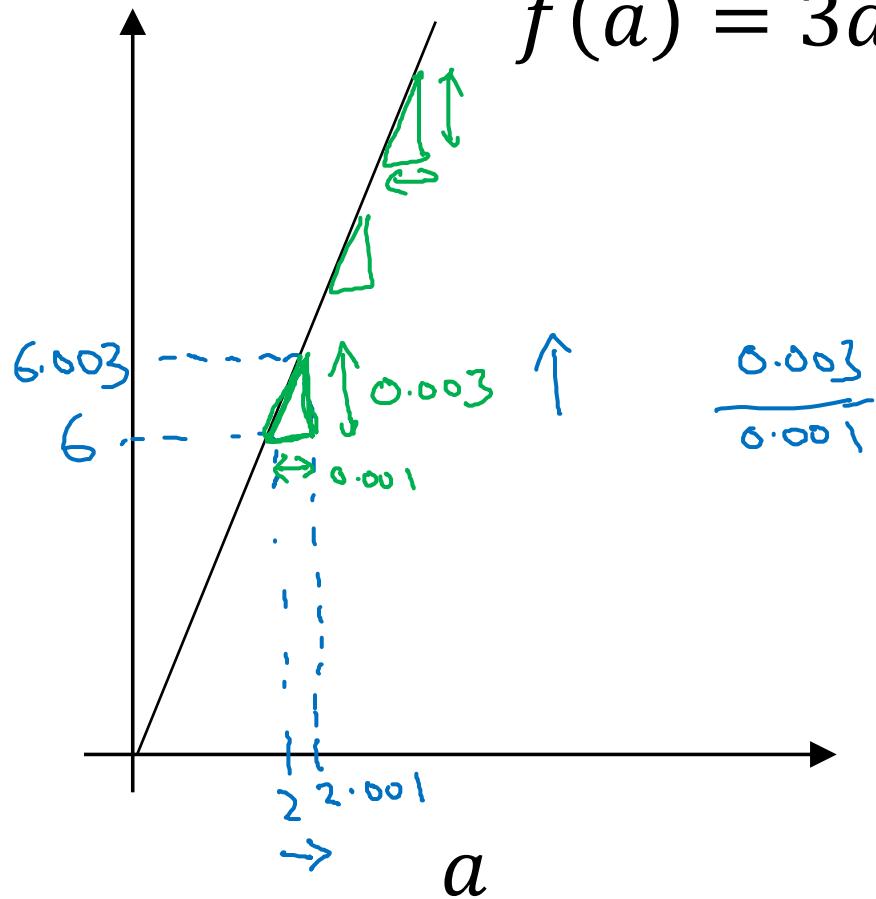


deeplearning.ai

Basics of Neural Network Programming

Derivatives

Intuition about derivatives



$$f(a) = 3a$$

$$\rightarrow a = 2$$

$$f(a) = 6$$

$$a = 2.001$$

$$f(a) = 6.003$$

height
width

slope (derivative) of $f(a)$

at $a=2$ is 3

$$\rightarrow a = 5$$

$$f(a) = 15$$

$$a = 5.001$$

$$f(a) = 15.003$$

slope at $a=5$ is also 3

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

0.001 ←
0.00000001
0.0000000001

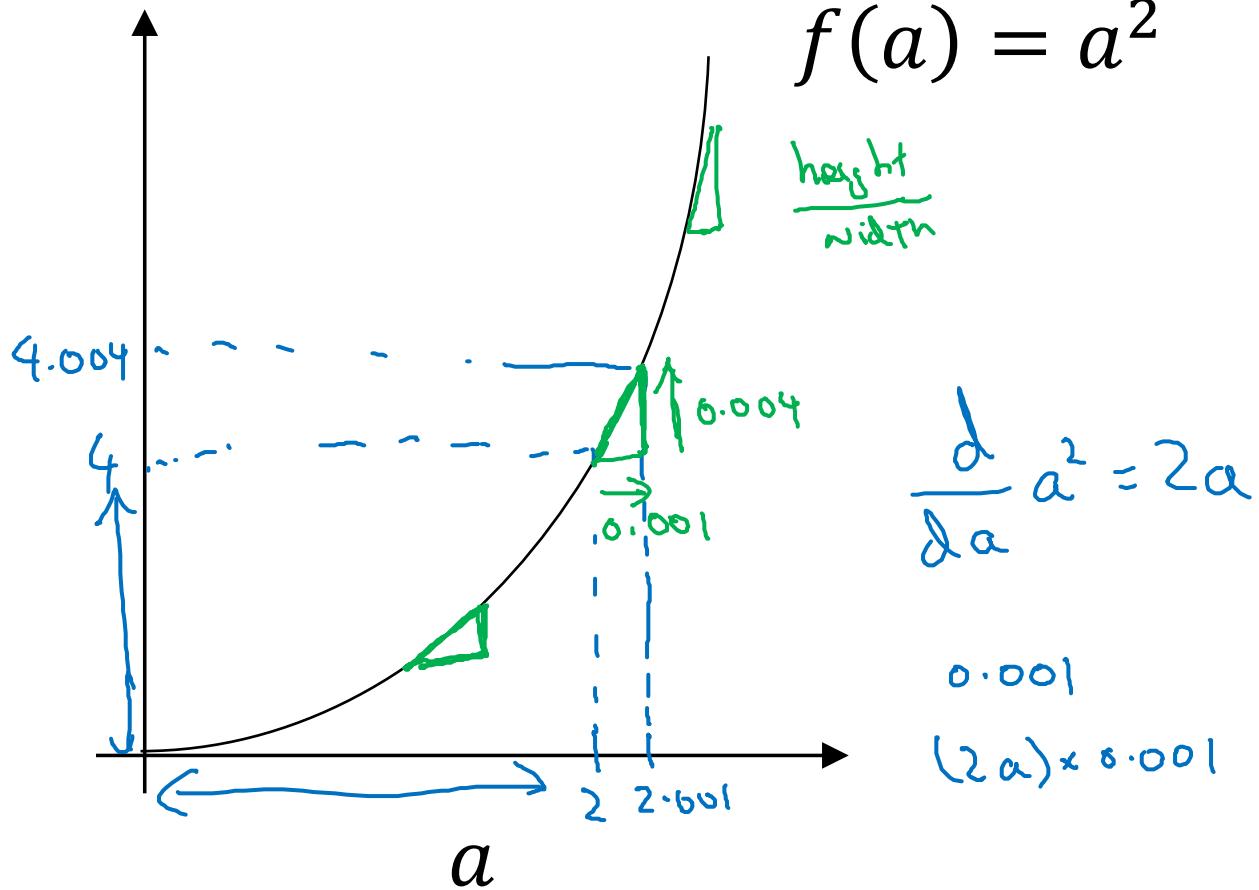


deeplearning.ai

Basics of Neural Network Programming

More derivatives
examples

Intuition about derivatives



$a = 2$ $f(a) = 4$
 $a = 2.001$ $f(a) \approx 4.004$

$0.001 \leftarrow$
 $0.00000\dots 01 \leftarrow$

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

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

$a = 5$ $f(a) = 25$
 $a = 5.001$ $f(a) \approx 25.010$

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

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

More derivative examples

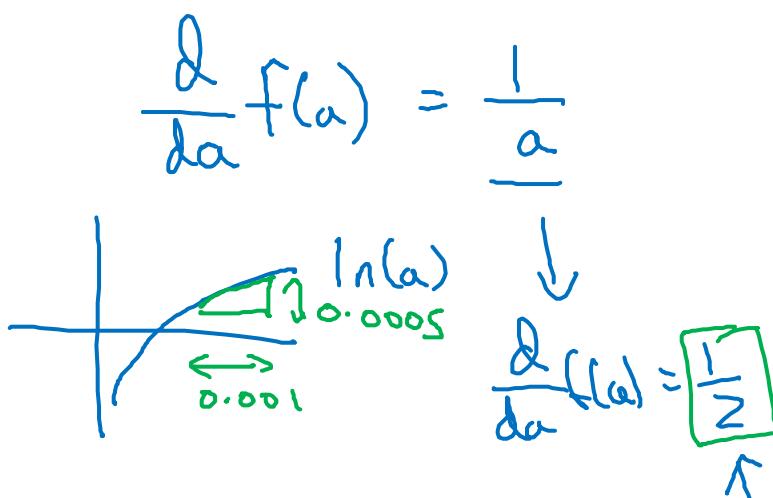
$$f(a) = a^2$$

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

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

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

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



$$a = 2$$

$$a = 2.001$$

$$f(a) = 4$$

$$f(a) \approx 4.004$$

$$a = 2$$

$$a = \underline{2.001}$$

$$f(a) = 8$$

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

$$a = 2$$

$$a = \underline{2.001}$$

$$f(a) \approx 0.69315$$

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

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



deeplearning.ai

Basics of Neural Network Programming

Computation Graph

Computation Graph

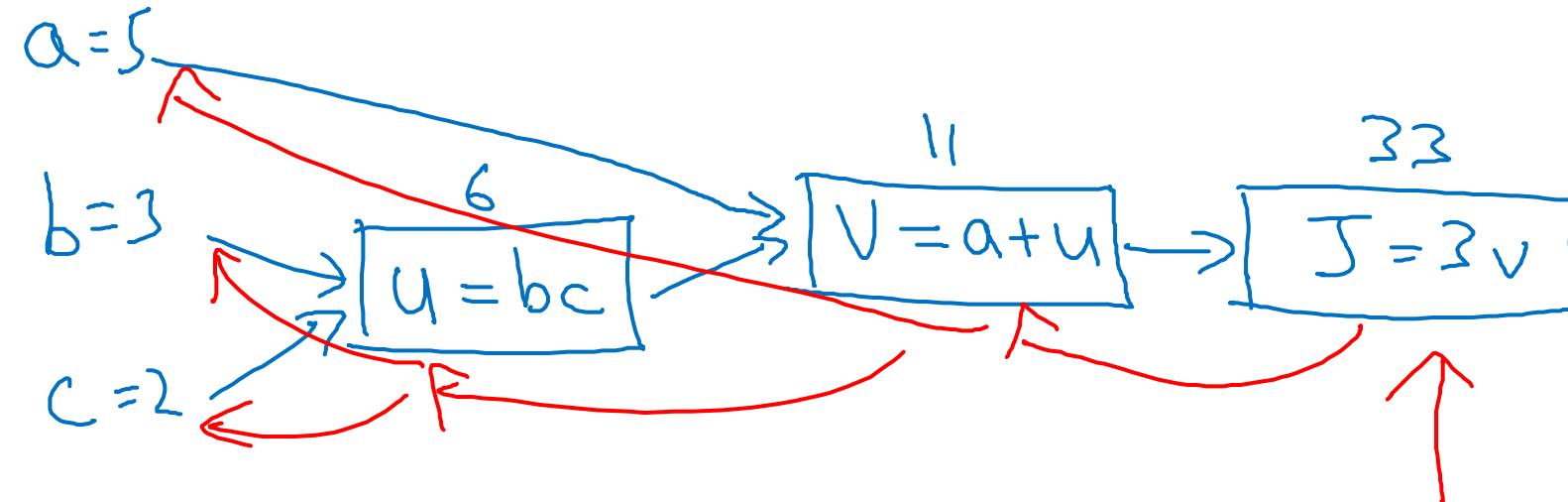
$$J(a, b, c) = 3(u + bc) = 3(5 + 3 \times 2) = 33$$

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

$$u = bc$$

$$v = a + u$$

$$J = 3v$$



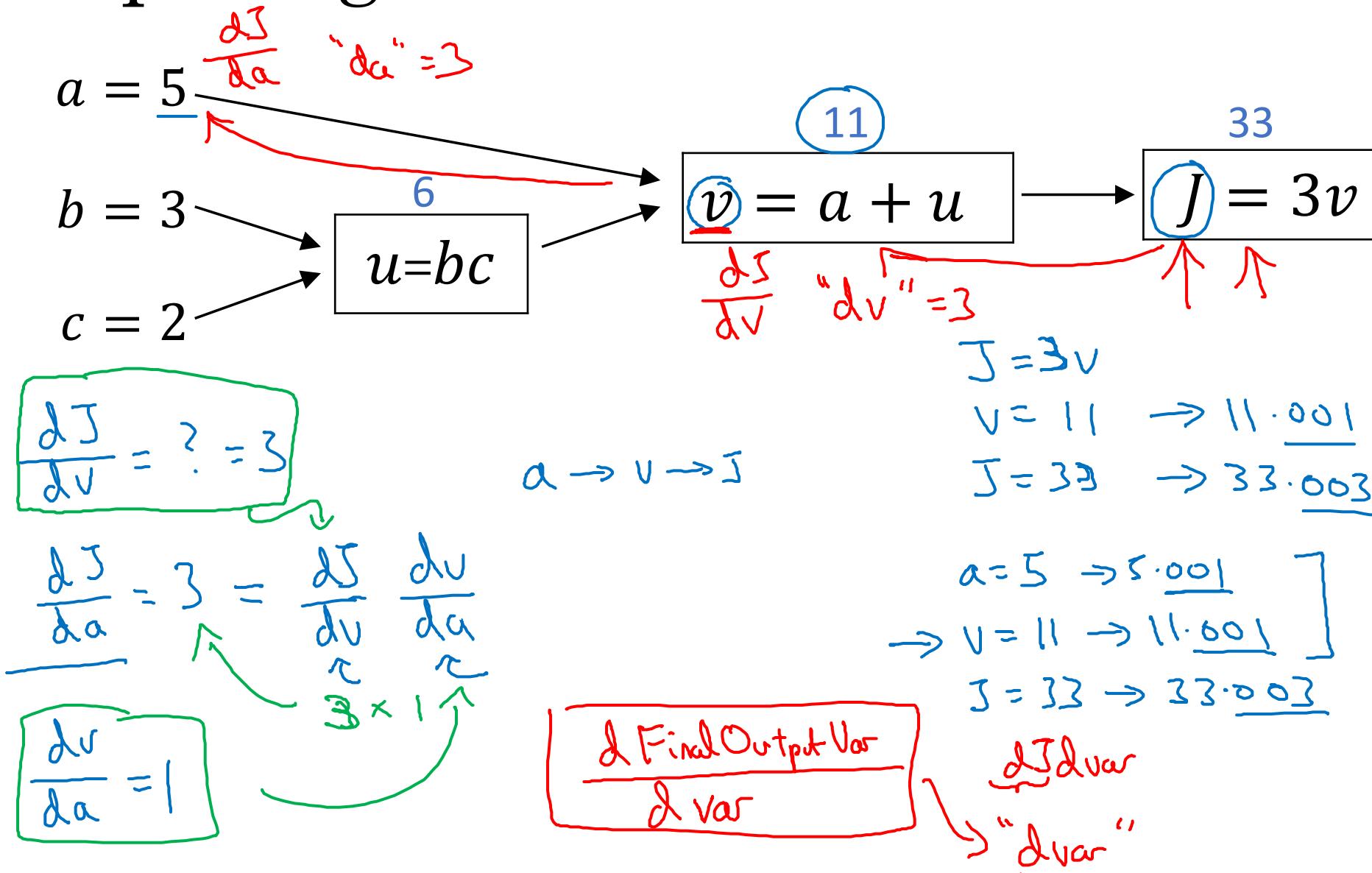


deeplearning.ai

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

$\frac{\partial J}{\partial a} \rightarrow \underline{a = 5}$
 $\frac{\partial J}{\partial b} \rightarrow \underline{b = 3}$
 $\frac{\partial J}{\partial c} \rightarrow \underline{c = 2}$
 $\frac{\partial J}{\partial u} = 3$
 $\frac{\partial J}{\partial v} = 3$
 $\frac{\partial J}{\partial J} = 33$

$a = 5 \rightarrow \frac{\partial a}{\partial a} = 3$
 $b = 3 \rightarrow \frac{\partial b}{\partial b} = 6$
 $c = 2 \rightarrow \frac{\partial c}{\partial c} = 9$
 $u = bc \rightarrow \frac{\partial u}{\partial u} = 3$
 $v = a + u \rightarrow \frac{\partial v}{\partial v} = 3$
 $J = 3v \rightarrow \frac{\partial J}{\partial J} = 33$

$u = 6 \rightarrow 6.001$
 $v = 11 \rightarrow 11.001$
 $J = 33 \rightarrow 33.003$

$\frac{\partial J}{\partial b} = \boxed{\frac{\partial J}{\partial u}} \cdot \underbrace{\frac{\partial u}{\partial b}}_{=2} = 6$
 $\frac{\partial J}{\partial a} = \boxed{\frac{\partial J}{\partial u}} \cdot \underbrace{\frac{\partial u}{\partial a}}_{=3} = 9$

$b = 3 \rightarrow 3.001$
 $u = b \cdot c = 6 \rightarrow 6.002$
 $J = 33.006$
 $v = 11.002$
 $J = 3v$



deeplearning.ai

Basics of Neural Network Programming

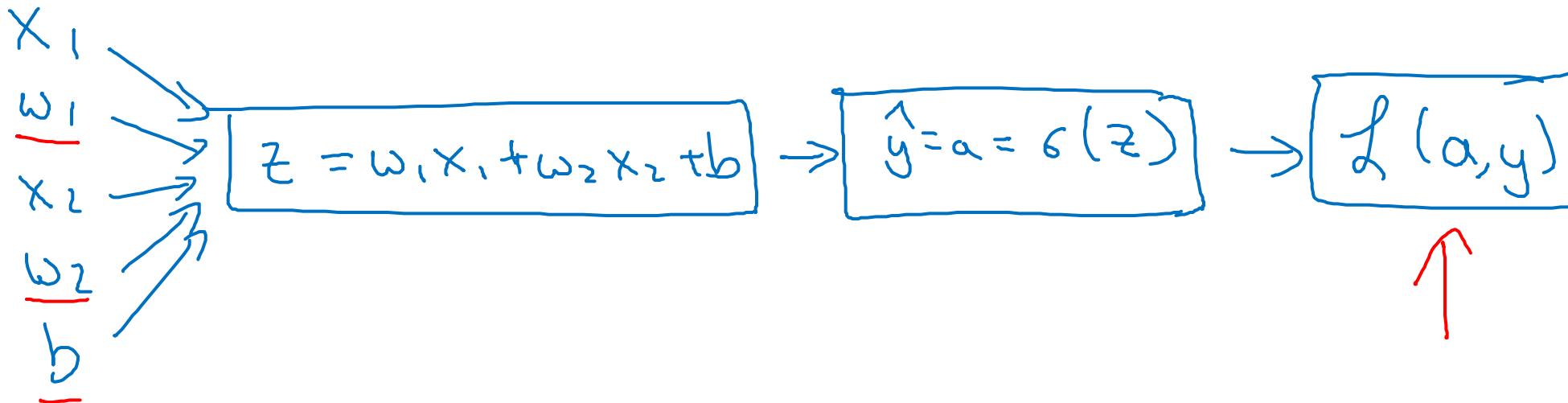
Logistic Regression Gradient descent

Logistic regression recap

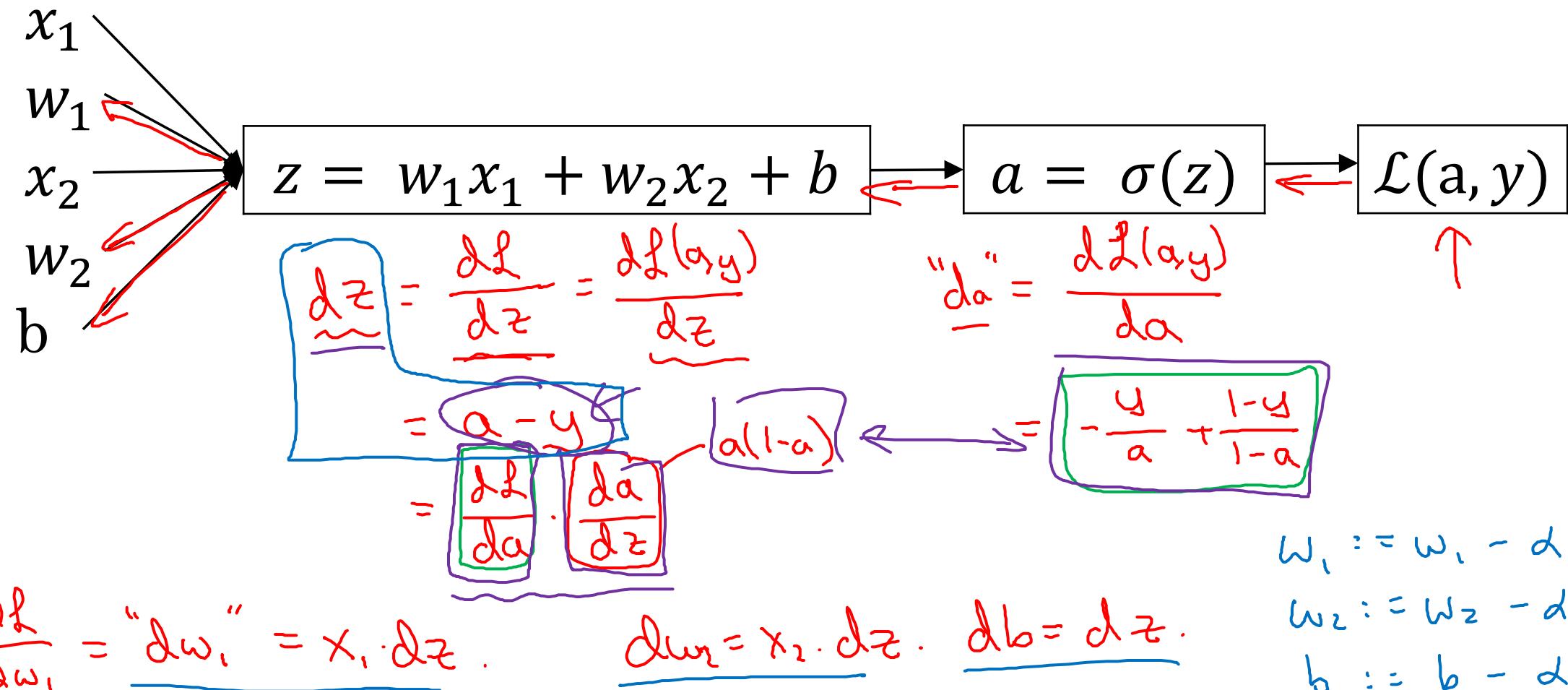
$$\rightarrow z = w^T x + b$$

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

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



Logistic regression derivatives





deeplearning.ai

Basics of Neural Network Programming

Gradient descent
on m examples

Logistic regression on m examples

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

$$\underline{\frac{\partial}{\partial \omega_1} J(\omega, b)} = \frac{1}{m} \sum_{i=1}^m \underbrace{\frac{\partial}{\partial \omega_1} l(a^{(i)}, y^{(i)})}_{\underline{dw_1^{(i)}} - (x^{(i)}, y^{(i)})}$$

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):
 $z += \omega[i] * x[i]$

$$z += b$$

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

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

Vectorized

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

\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.

$$u = Av$$

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

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

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 $j=1 \dots n_x$
 $d\omega_j := \dots$

$$\boxed{\begin{aligned} dw_1 &+= x_1^{(i)} dz^{(i)} \\ dw_2 &+= x_2^{(i)} dz^{(i)} \\ db &+= dz^{(i)} \end{aligned}} \quad | \quad n_x = 2$$

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

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

$$d\omega /= m.$$



deeplearning.ai

Basics of Neural Network Programming

Vectorizing Logistic Regression

Vectorizing Logistic Regression

$$\begin{aligned} \rightarrow z^{(1)} &= w^T x^{(1)} + b \\ \rightarrow a^{(1)} &= \sigma(z^{(1)}) \end{aligned}$$

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

$$\begin{aligned} z^{(3)} &= w^T x^{(3)} + b \\ a^{(3)} &= \sigma(z^{(3)}) \end{aligned}$$

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

$\xrightarrow{\text{---}}$

$$\frac{(n_x, m)}{\mathbb{R}^{n_x \times m}}$$

$$\overline{\omega^T} \begin{bmatrix} 1 & & & 1 \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \end{bmatrix}$$

$$\overline{z} = \begin{bmatrix} z^{(1)} & z^{(2)} & \dots & z^{(m)} \end{bmatrix} = \underline{\omega^T X} + \underbrace{\begin{bmatrix} b & b & \dots & b \end{bmatrix}_{1 \times m}}_{\rightarrow} = \begin{bmatrix} \underline{\omega^T x^{(1)}} + b \\ \underline{\omega^T x^{(2)}} + b \\ \vdots \\ \underline{\omega^T x^{(m)}} + b \end{bmatrix}$$

$$\rightarrow \overline{z} = np.\text{dot}(\underline{\omega^T}, \underline{X}) + \underline{\underline{b}}_{(1,1)}$$

\mathbb{R}

"Broadcasting"

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



deeplearning.ai

Basics of Neural Network Programming

Vectorizing Logistic Regression's Gradient Computation

Vectorizing Logistic Regression

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

$$dZ = [dz^{(1)} \ dz^{(2)} \ \dots \ dz^{(m)}] \quad | \times m$$

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

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

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

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

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

$$\begin{aligned} dw &= \frac{1}{m} X dZ^T \\ &= \frac{1}{m} \left[\begin{array}{c|c} x^{(1)} & \cdots & x^{(m)} \\ \hline 1 & & 1 \end{array} \right] \left[\begin{array}{c} dz^{(1)} \\ \vdots \\ dz^{(m)} \end{array} \right] \\ &= \frac{1}{m} \left[\underline{x^{(1)} dz^{(1)}} + \cdots + \underline{x^{(m)} dz^{(m)}} \right] \\ &\qquad\qquad\qquad n \times 1 \end{aligned}$$

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

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

$$= A_{(3,4)}$$



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

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

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

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

General Principle

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

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

Matlab/Octave: bsxfun



deeplearning.ai

Basics of Neural Network Programming

Explanation of logistic
regression cost function
(Optional)

Logistic regression cost function

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

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

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

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

Logistic regression cost function

$$\begin{aligned} \rightarrow & \boxed{\text{If } y = 1: \quad p(y|x) = \hat{y}} \\ \rightarrow & \boxed{\text{If } y = 0: \quad p(y|x) = 1 - \hat{y}} \end{aligned}$$

$p(y|x)$

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

←

$$\text{If } y=1: \quad p(y|x) = \hat{y} \underset{=} {=} 1$$
$$\text{If } y=0: \quad p(y|x) = \hat{y}^0 (1-\hat{y})^{(1-y)} = 1 \times (1-\hat{y}) = 1 - \hat{y}$$
$$\uparrow \log p(y|x) = \log \hat{y}^y (1-\hat{y})^{(1-y)} = y \log \hat{y} + (1-y) \log (1-\hat{y})$$
$$= -\frac{1}{m} \sum_{i=1}^m (\hat{y}_i - y_i)$$

Cost on m examples

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

$$\log p(\dots) = \sum_{i=1}^m \underbrace{\log p(y^{(i)} | x^{(i)})}_{-L(\hat{y}^{(i)}, y^{(i)})}$$

Maximum likelihood
estimation \nearrow

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

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

Copyright Notice

These slides are distributed under the Creative Commons License.

[DeepLearning.AI](#) makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite [DeepLearning.AI](#) as the source of the slides.

For the rest of the details of the license, see <https://creativecommons.org/licenses/by-sa/2.0/legalcode>

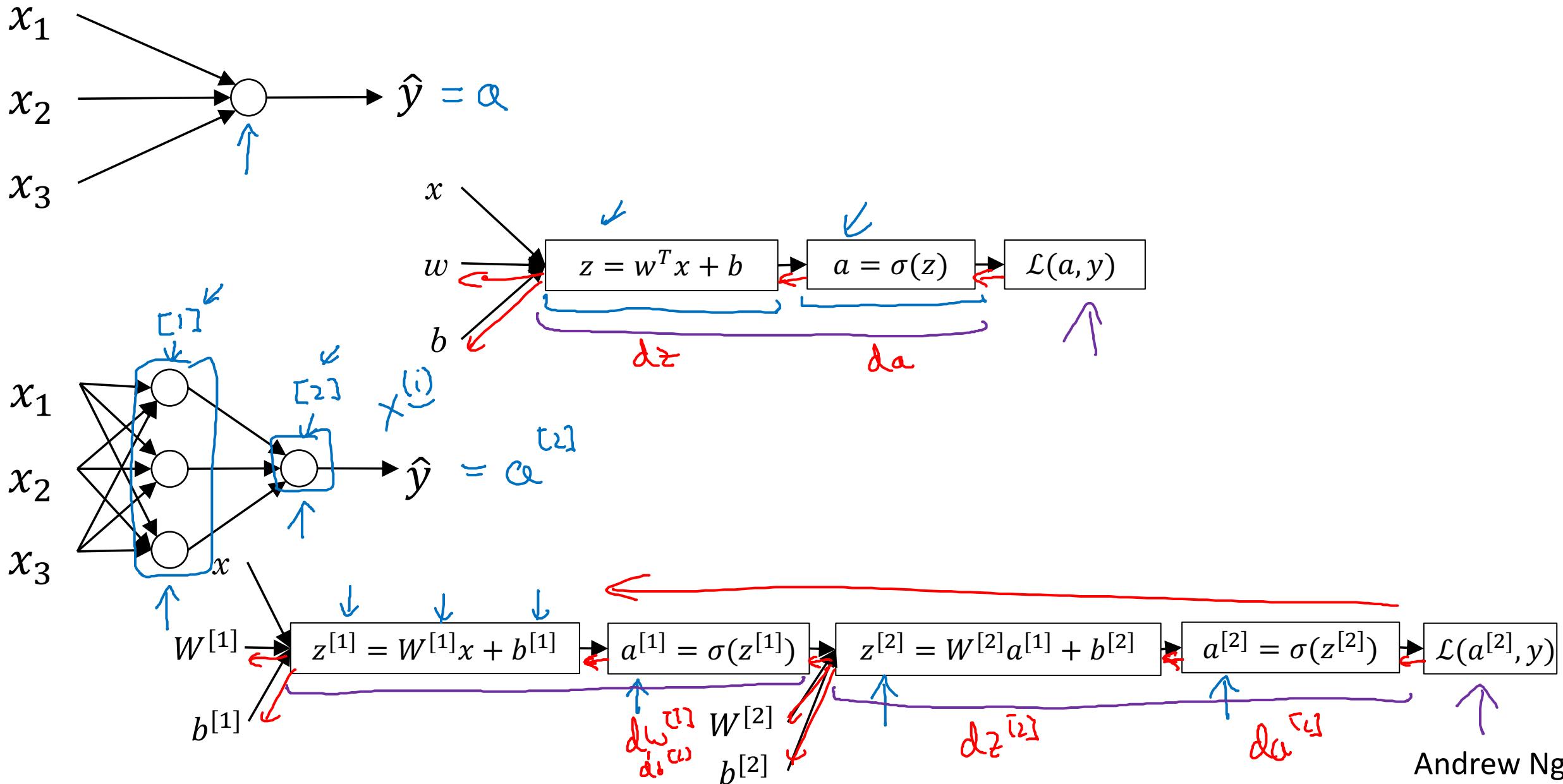


deeplearning.ai

One hidden layer
Neural Network

Neural Networks
Overview

What is a Neural Network?



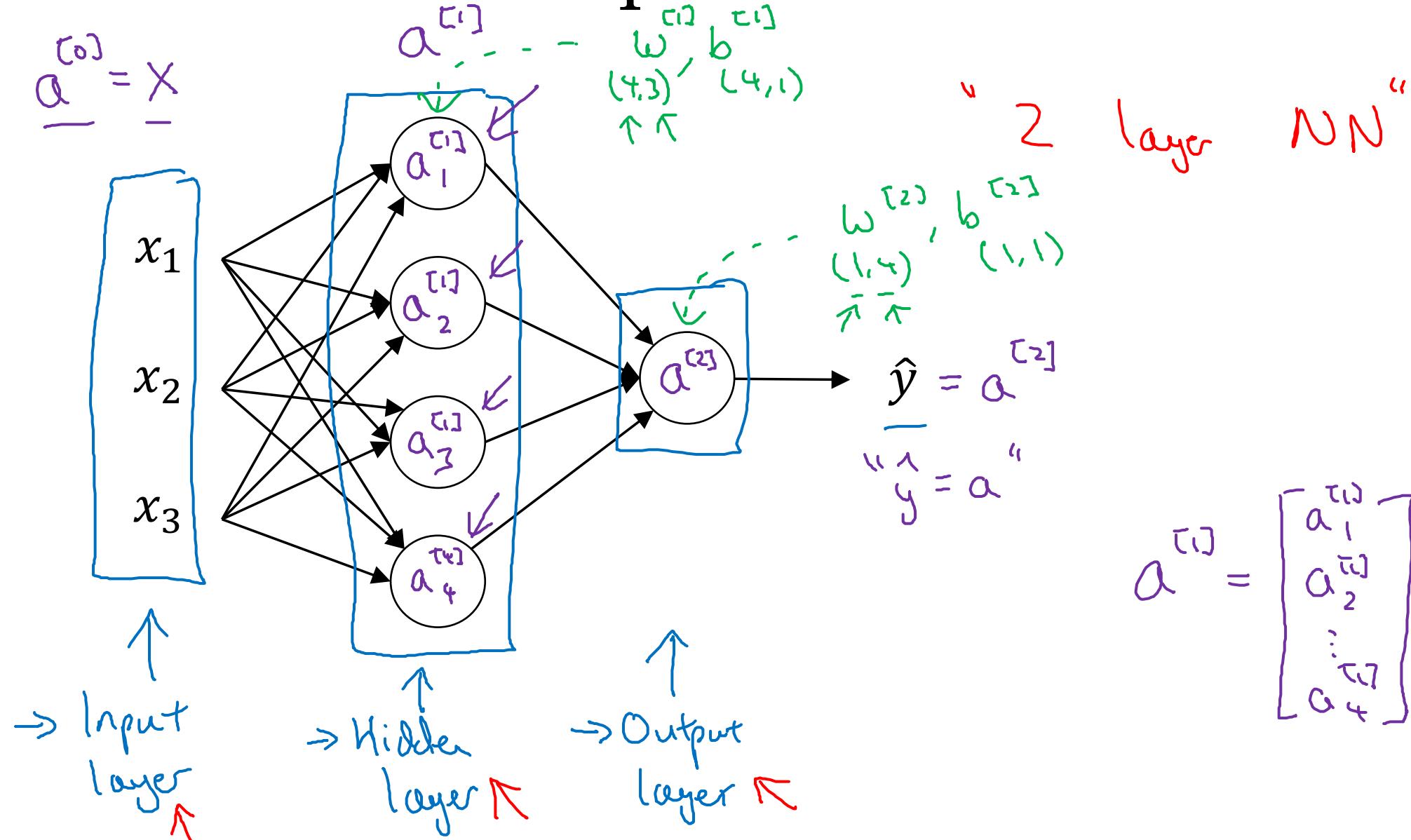


deeplearning.ai

One hidden layer
Neural Network

Neural Network
Representation

Neural Network Representation



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

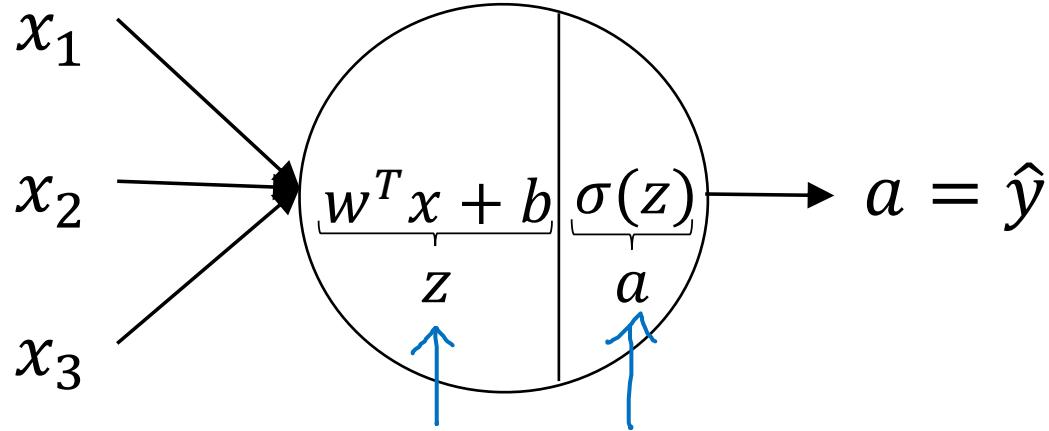


deeplearning.ai

One hidden layer Neural Network

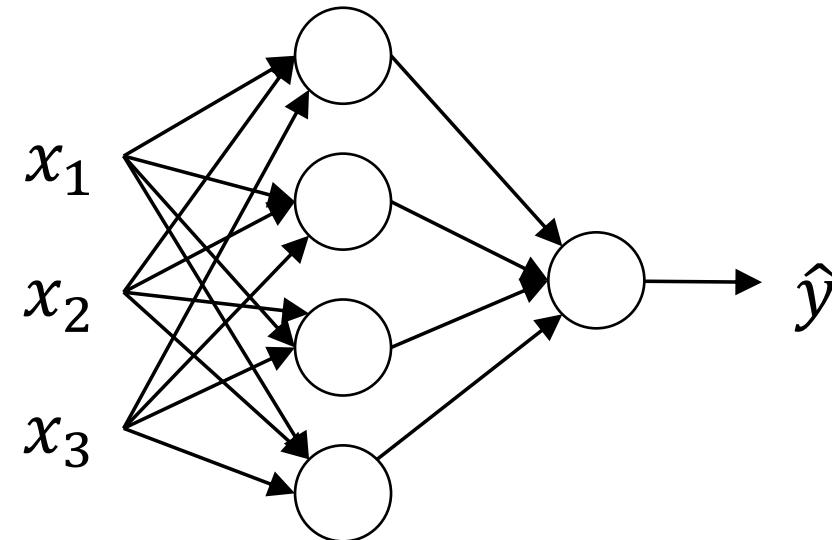
Computing a Neural Network's Output

Neural Network Representation

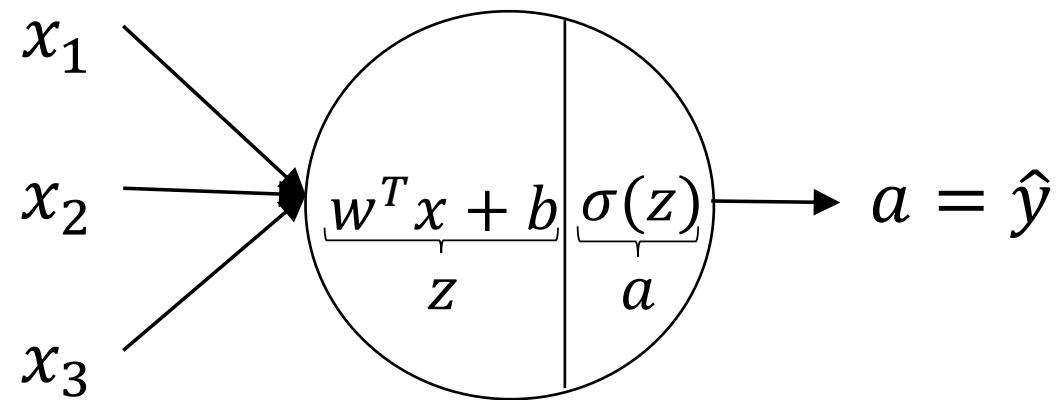


$$z = w^T x + b$$

$$a = \sigma(z)$$

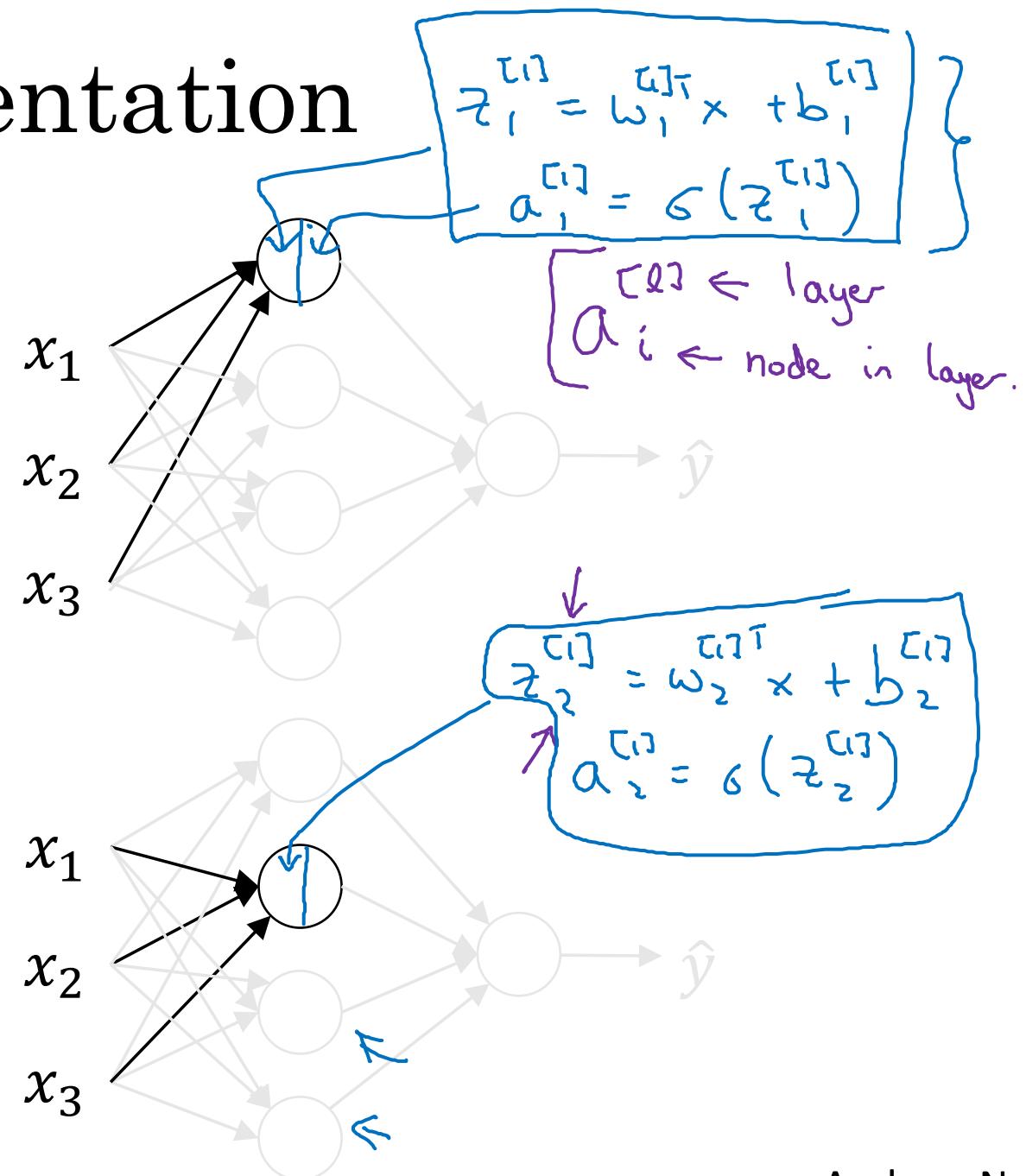


Neural Network Representation

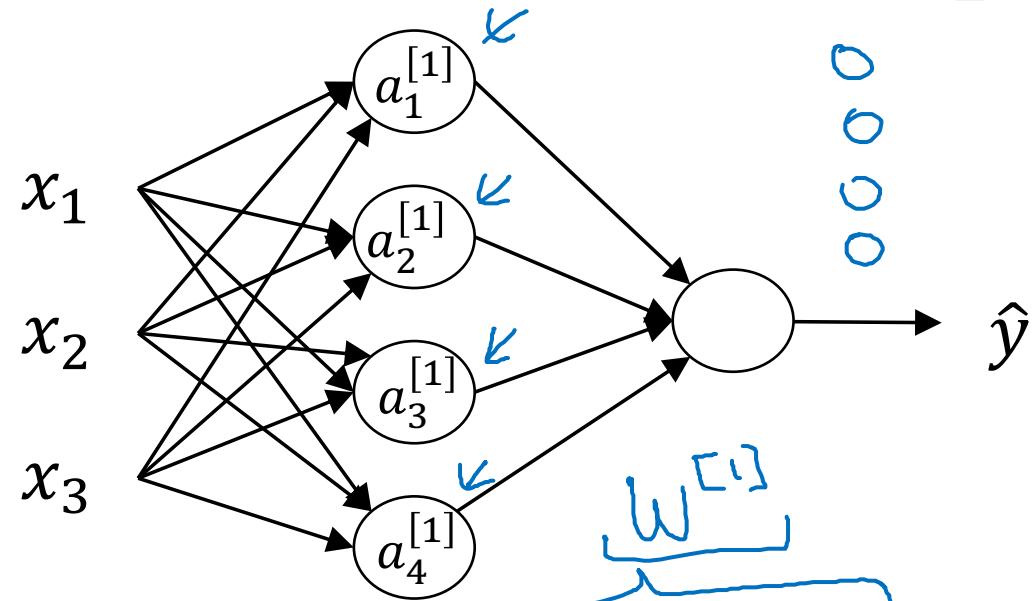


$$z = w^T x + b$$

$$a = \sigma(z)$$



Neural Network Representation



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

$$\rightarrow a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix} = g(z^{[1]})$$

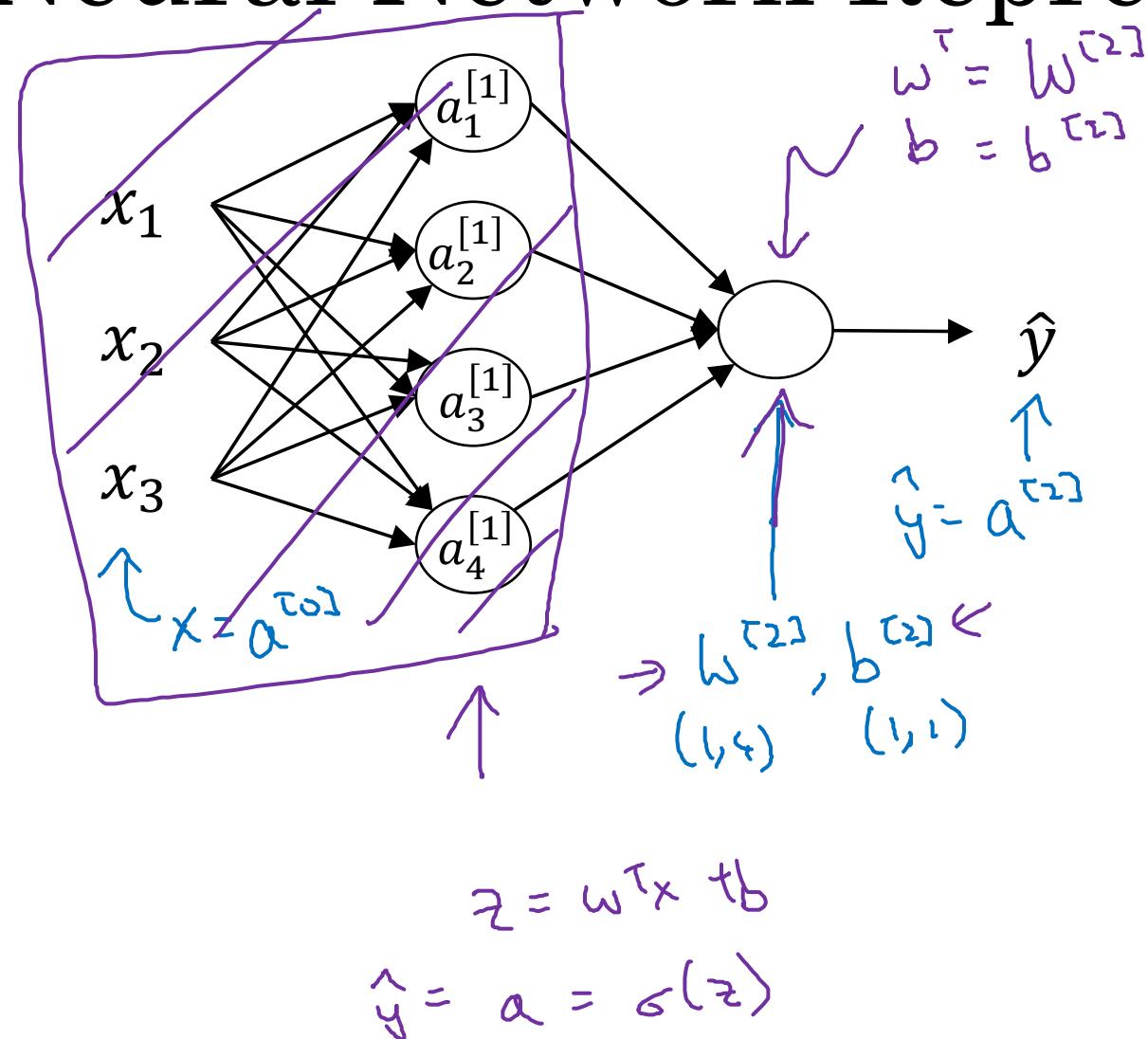
$(w_i^{[1]T} x + b_i^{[1]}) \sigma(z_i^{[1]})$

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

$$= \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

Andrew Ng

Neural Network Representation learning



Given input x :

$$\rightarrow z^{[1]} = W^{[1]} a^{[0]} + b^{[1]}$$

$(4,1) \quad (4,3) \quad (3,1) \quad (4,1)$

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

$(4,1) \quad (4,1)$

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

$(1,1) \quad (1,4) \quad (4,1) \quad (1,1)$

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

$(1,1) \quad (1,1)$

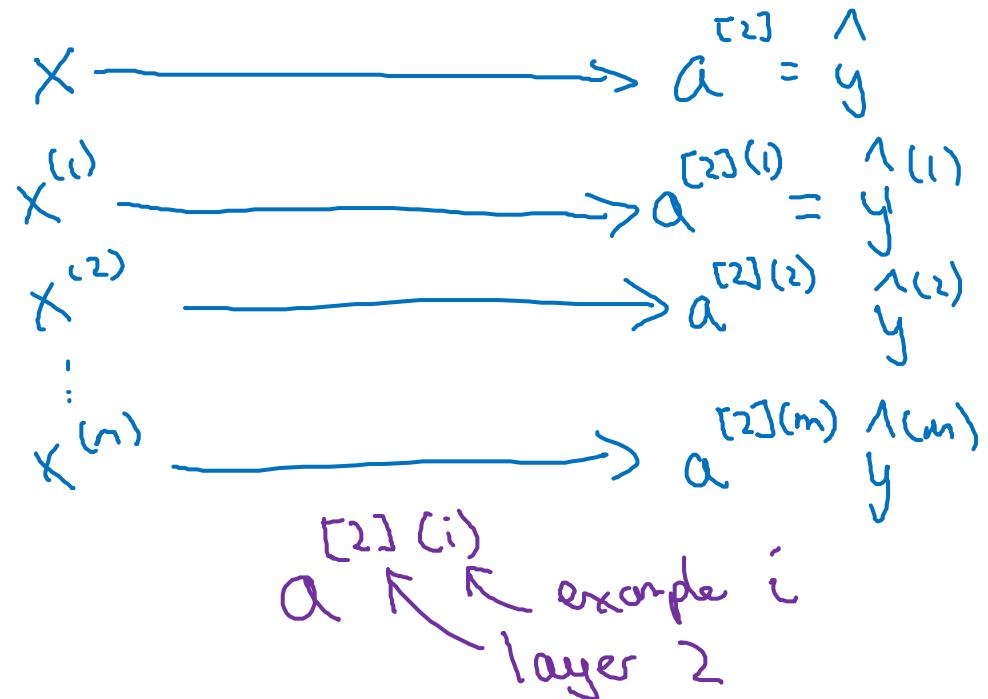
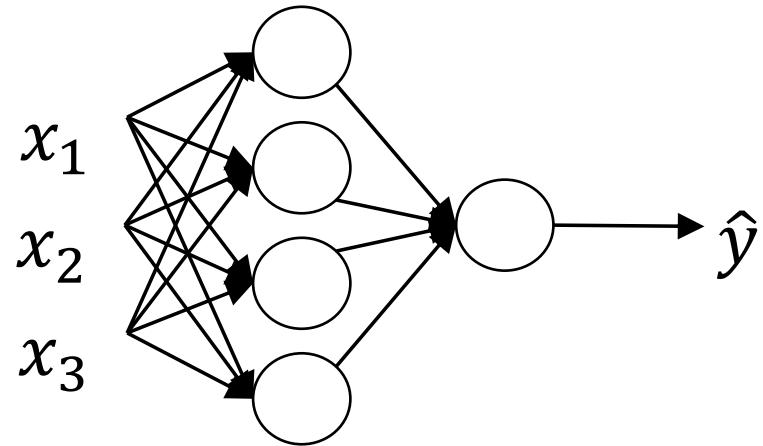


deeplearning.ai

One hidden layer Neural Network

Vectorizing across
multiple examples

Vectorizing across multiple examples



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

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:

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

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

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

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

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

(n_x, m)

truly complex

hidden units.

$$\begin{aligned} z^{[1]} &= w^{[1]} X + b^{[1]} \\ \rightarrow A^{[1]} &= c(z^{[1]}) \\ \rightarrow z^{[2]} &= w^{[2]} A^{[1]} + b^{[2]} \\ \rightarrow A^{[2]} &= c(z^{[2]}) \end{aligned}$$

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

The diagram illustrates a recurrent neural network layer $A^{[t]}$. It features a sequence of hidden states represented by circles. The first state is labeled a , followed by a_1 , a_2 , and so on. An arrow points from a to a_1 , and another arrow points from a_1 to a_2 . A dashed line indicates the sequence continues. Above the states, arrows show the flow of information: one arrow points from a to a_1 , and another arrow points from a_1 to a_2 . To the right of the sequence, a bracket groups the states under the label "hidden units".



deeplearning.ai

One hidden layer Neural Network

Explanation for vectorized implementation

Justification for vectorized implementation

$$\underline{z}^{1} = \omega^{[1]} \underline{x}^{(1)} + \cancel{v^{[1]}} ,$$

$$\underline{z}^{(1)(2)} = \underline{\omega}^{(1)(2)} \underline{x}^{(2)} + \underline{b}$$

$$\underline{z}^{(1,2,3)} = \omega^{(1,2,3)} x + b^{(1,2,3)}$$

$$\omega^{(i)} = \left[\begin{array}{c} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{array} \right]$$

$$\omega^{[1]} x^{(1)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

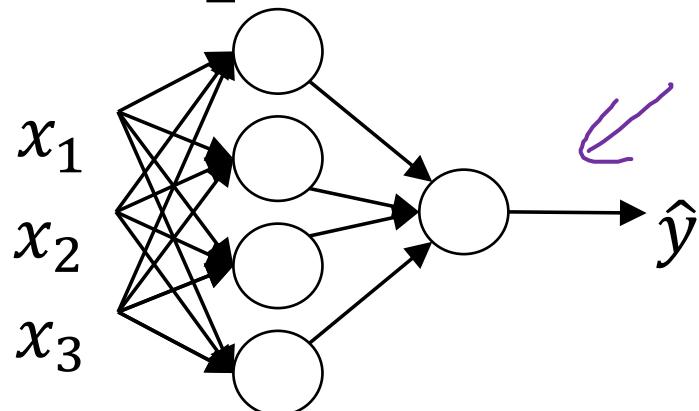
$$\omega^{(i)} x^{(i)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

$$\omega^{(1)} x^{(3)} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$z^{[i]} = \bar{w}^{[i]} X + b^{[i]}$$

$\bar{w}^{[i]}$ $\begin{bmatrix} 1 & | & | & | \\ X^{(1)} & X^{(2)} & X^{(3)} \dots & | \\ | & | & | & \dots \end{bmatrix}$ = $\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$ = $\begin{bmatrix} 1 & | & | & | \\ z^{[1]c(1)} & z^{[1]c(2)} & z^{[1]c(3)} \dots & | \\ | & | & | & \dots \end{bmatrix}$ = $\bar{z}^{[i]}$
 $\bar{w}^{[i]} X^{(i)} = z^{[i]c(i)}$

Recap of vectorizing across multiple examples



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

A vertical stack of input vectors $x^{(1)}, x^{(2)}, \dots, x^{(m)}$. 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 vertical stack of hidden layer activations $a^{1}, a^{[1](2)}, \dots, a^{[1](m)}$. 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))

```

Handwritten annotations: $x = a^{[1]}$ and $x^{(i)} = a^{[1](i)}$ are written next to the equations for $a^{[1]}$ and $a^{[2]}$ respectively. Brackets group the first two equations and the last two equations.

```

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

```

Handwritten annotations: $A^{[1]}$ is circled in blue. Brackets group the first two equations and the last two equations.

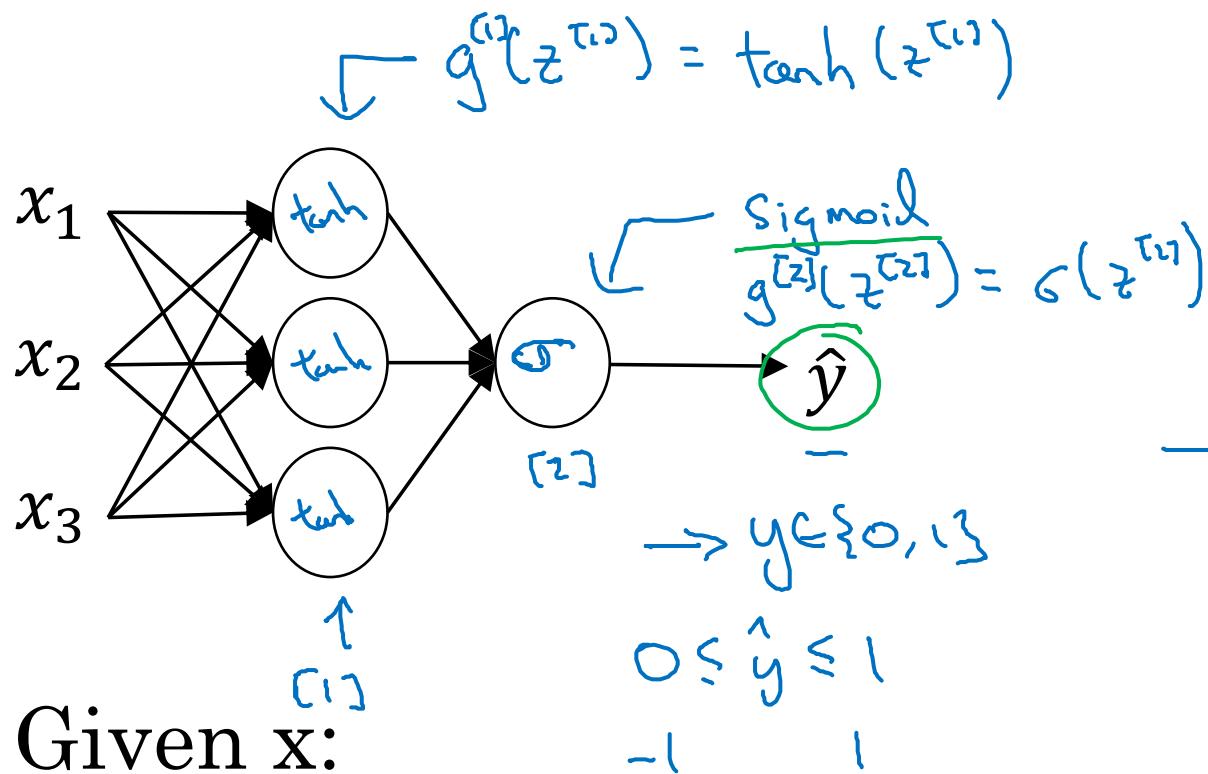


deeplearning.ai

One hidden layer Neural Network

Activation functions

Activation functions

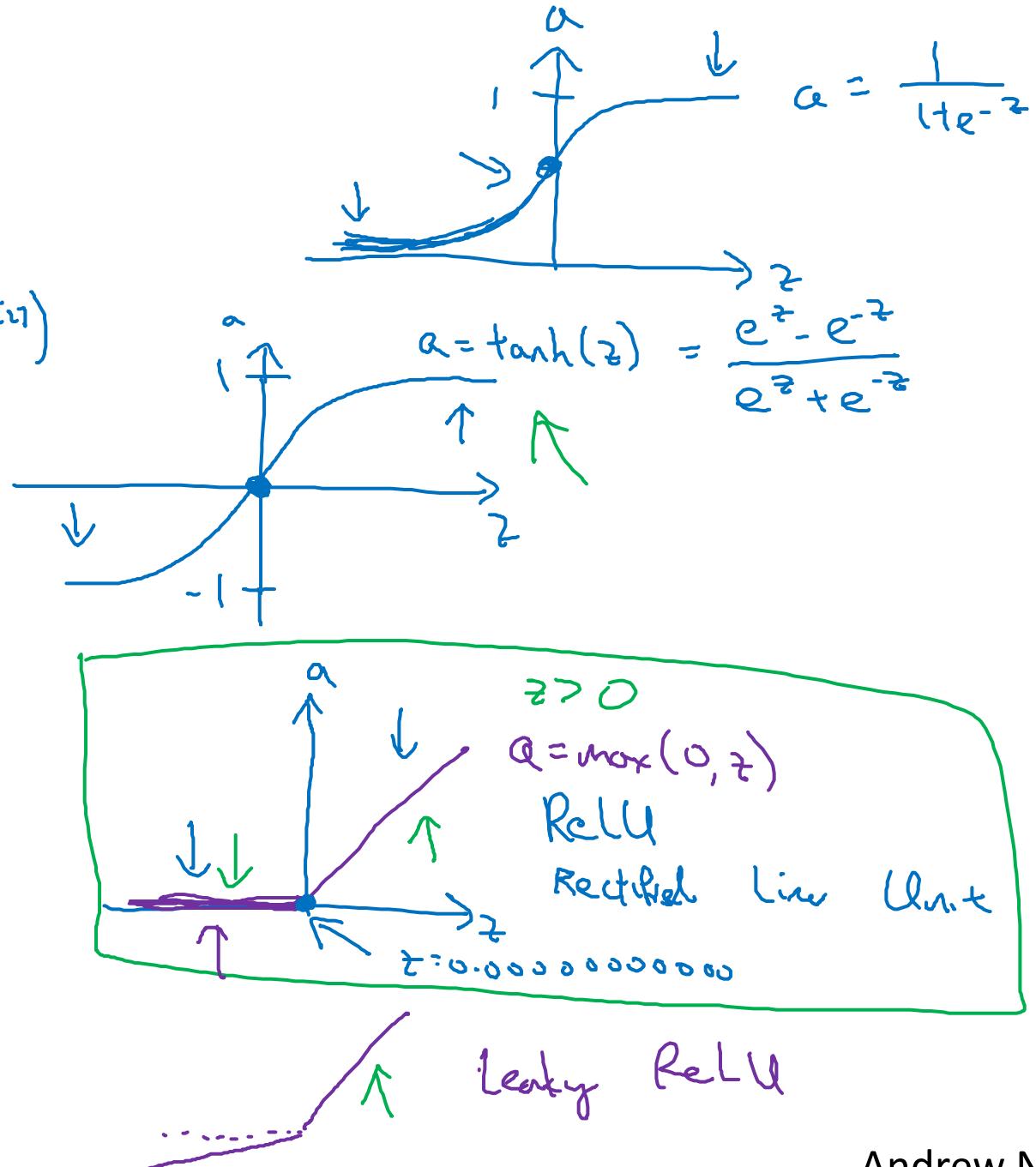


$$z^{[1]} = W^{[1]}x + b^{[1]}$$

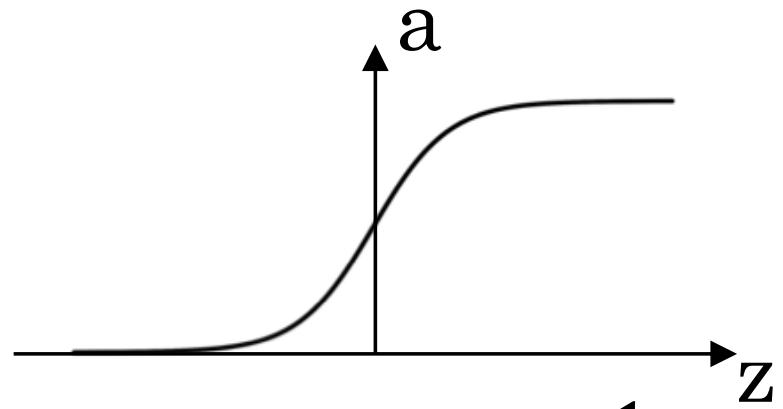
→ ~~$a^{[1]} = \sigma(z^{[1]})$~~ $\overset{[1]}{g}(z^{[1]})$

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

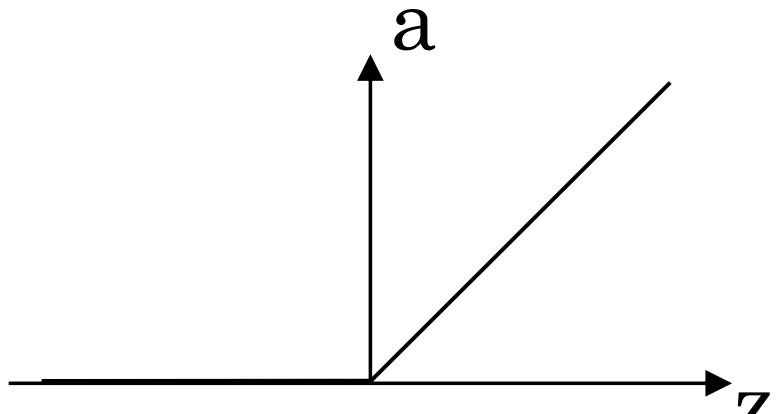
→ ~~$a^{[2]} = \sigma(z^{[2]})$~~ $\overset{[2]}{g}(z^{[2]})$



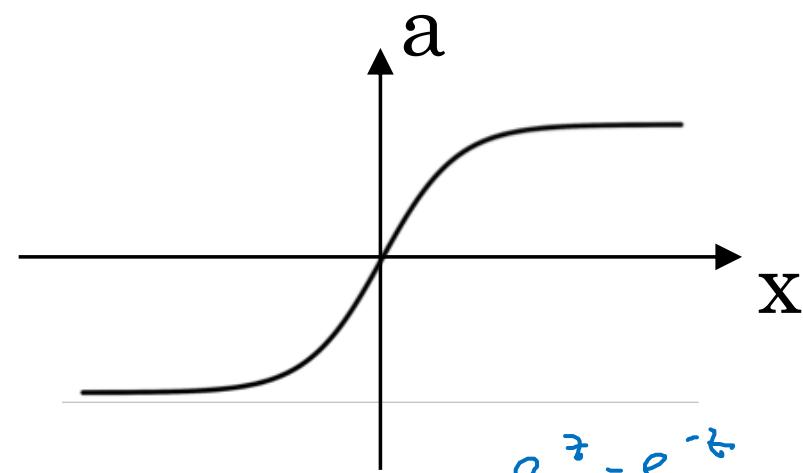
Pros and cons of activation functions



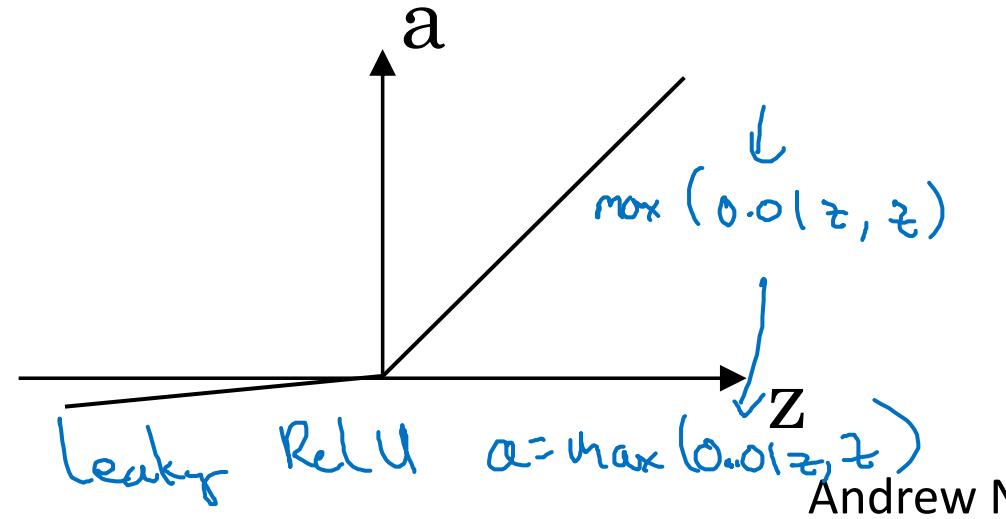
$$\text{sigmoid: } a = \frac{1}{1 + e^{-z}}$$



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



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



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

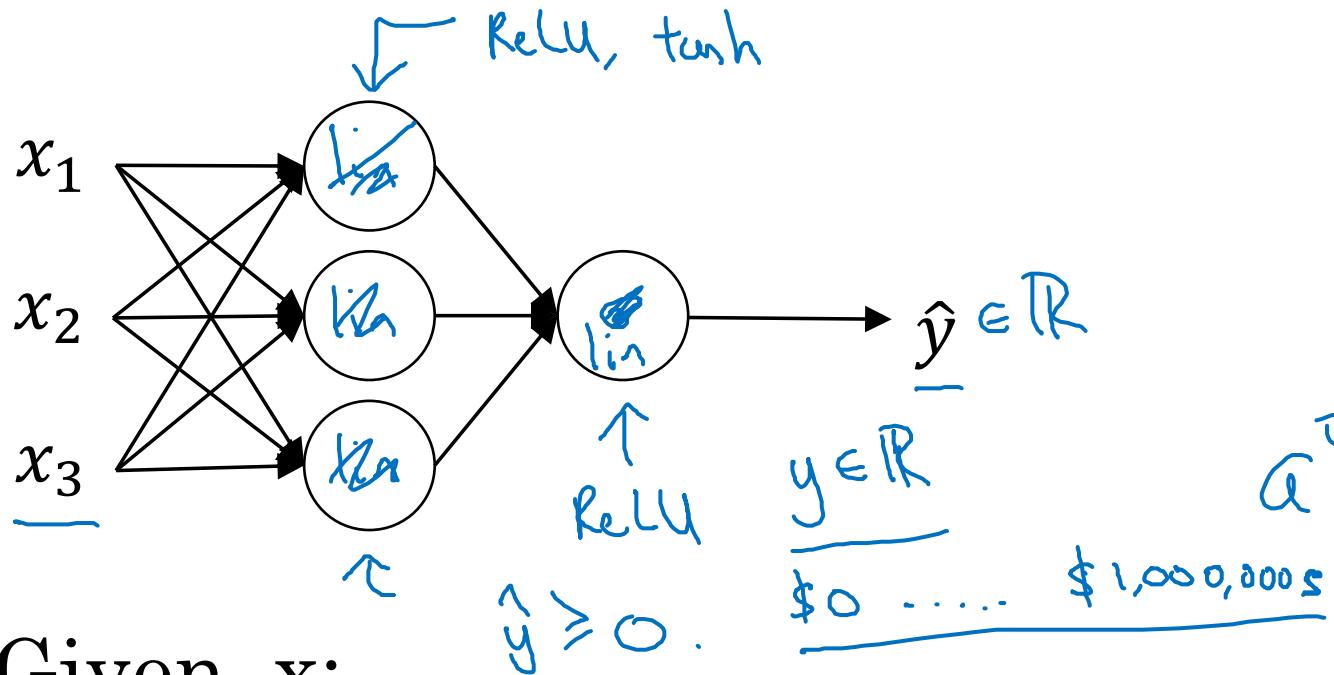


deeplearning.ai

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"

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

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

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

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

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

$$g(z) = z$$

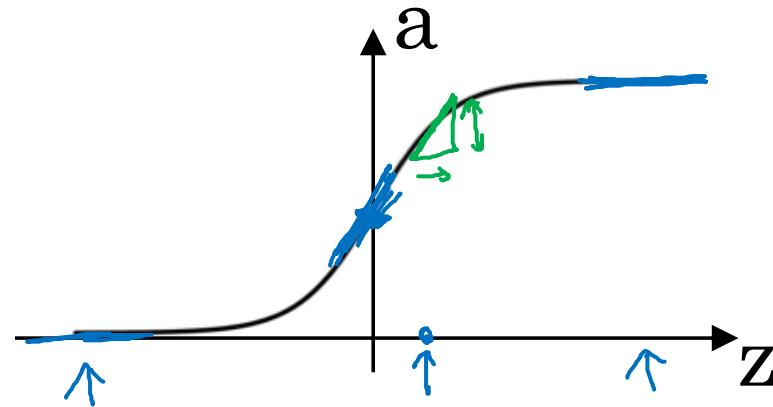


deeplearning.ai

One hidden layer Neural Network

Derivatives of activation functions

Sigmoid activation function



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

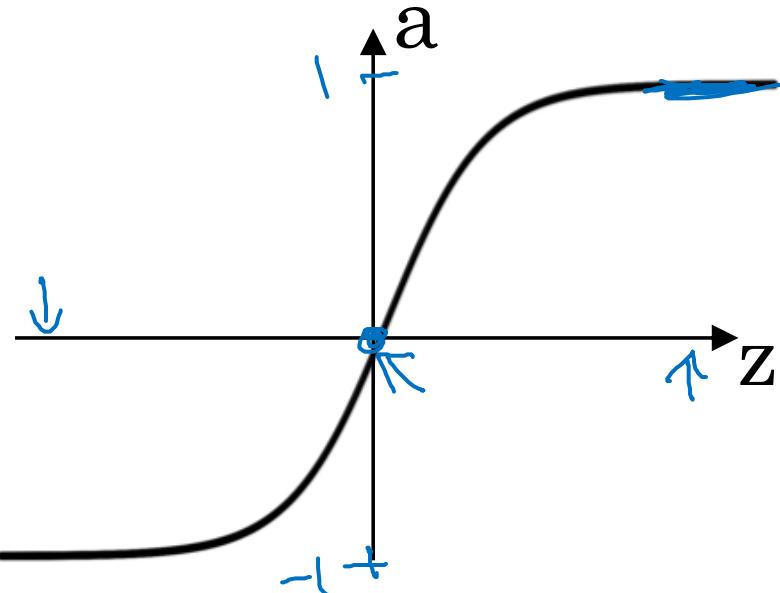
$$a = g(z) = \frac{1}{1 + e^{-z}}$$

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

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

Andrew Ng

Tanh activation function



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

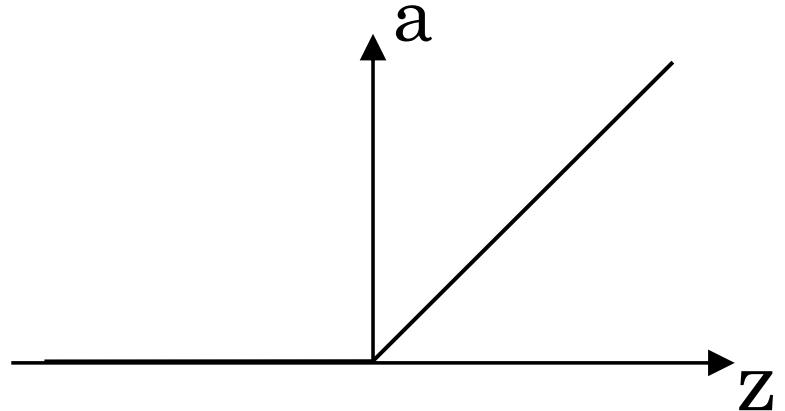
$$= \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

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

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

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

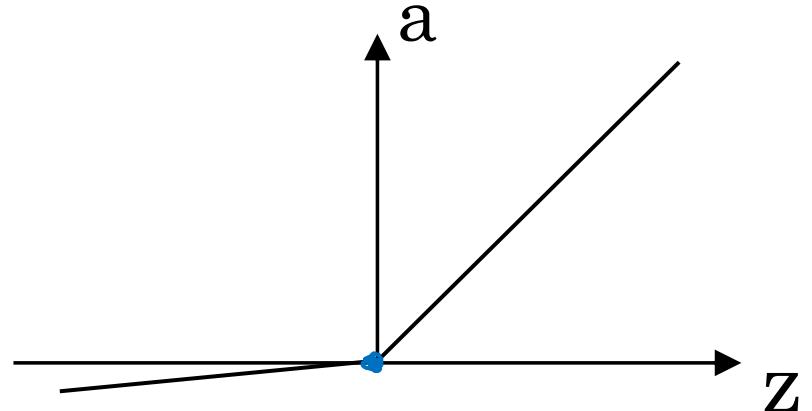
ReLU and Leaky ReLU



ReLU

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

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



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

Gradient Descent:

→ Repeat {

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

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

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

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

↳

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:

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

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

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

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

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

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

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

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

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



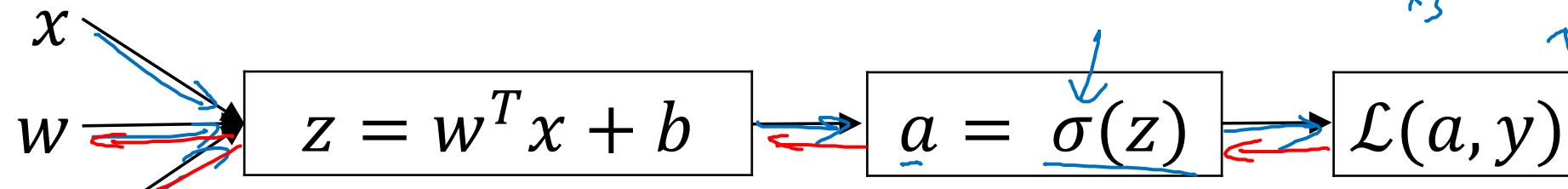
deeplearning.ai

One hidden layer
Neural Network

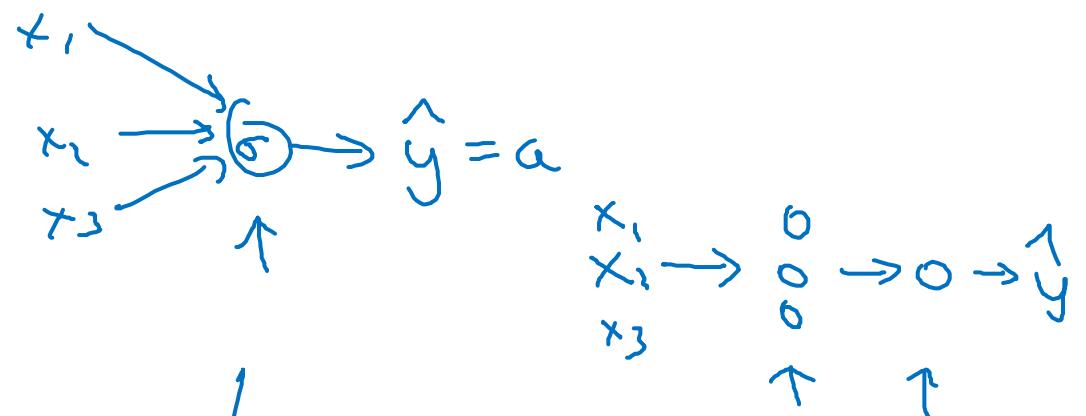
Backpropagation
intuition (Optional)

Computing gradients

Logistic regression



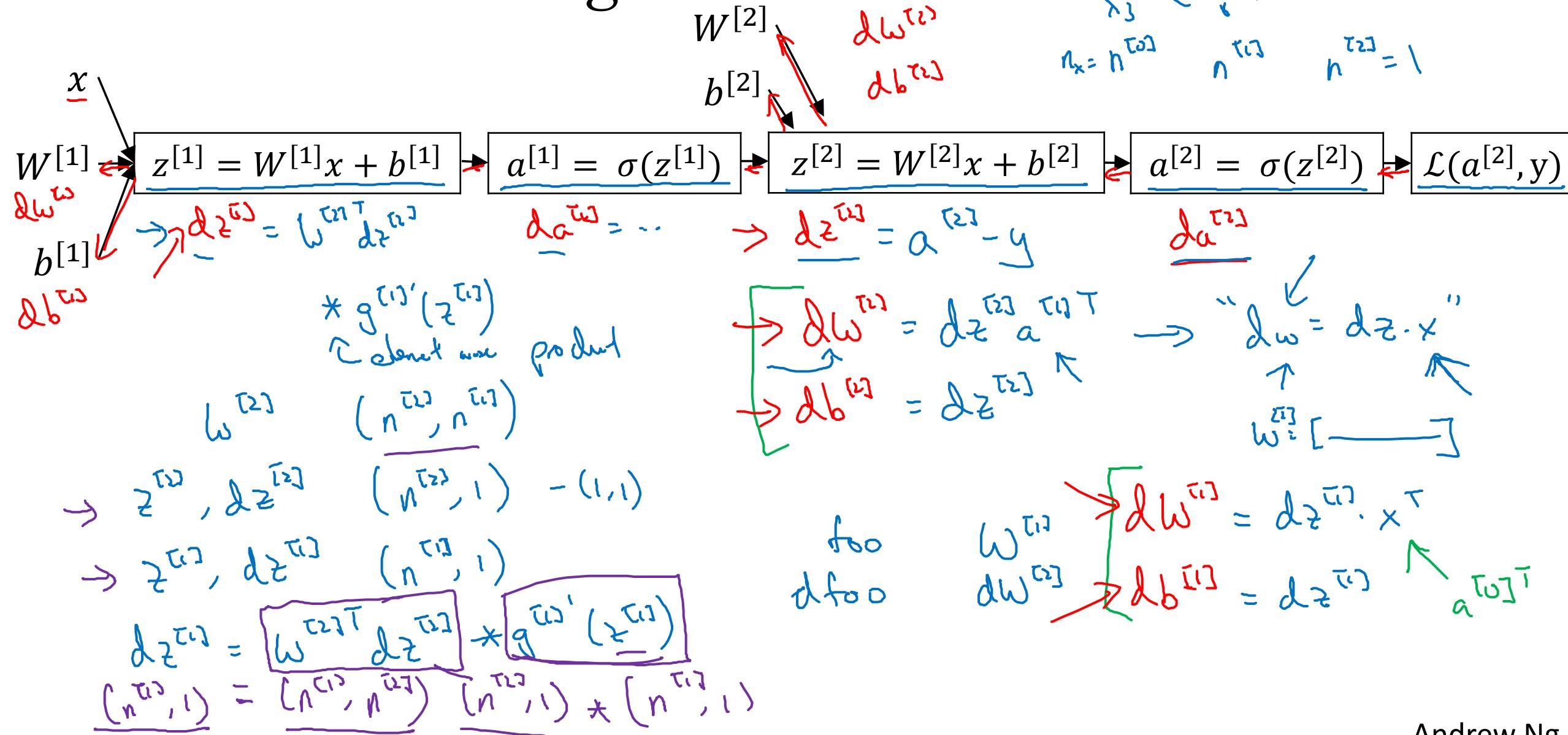
$$\begin{aligned} \frac{\partial z}{\partial w} &= x \\ \frac{\partial z}{\partial b} &= 1 \\ \frac{\partial z}{\partial a} &= g'(z) \\ g(z) &= \sigma(z) \\ \frac{\partial \mathcal{L}}{\partial z} &= \frac{\partial \mathcal{L}}{\partial a} \cdot \frac{\partial a}{\partial z} \\ "dz" &= "da" \end{aligned}$$



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

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

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:

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

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

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

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

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

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

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

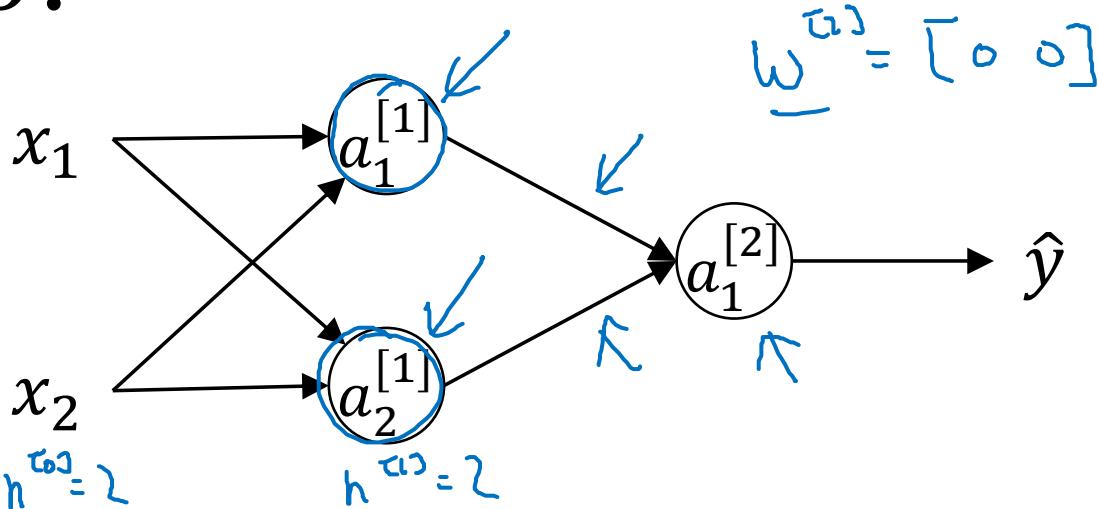


deeplearning.ai

One hidden layer
Neural Network

Random Initialization

What happens if you initialize weights to zero?



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

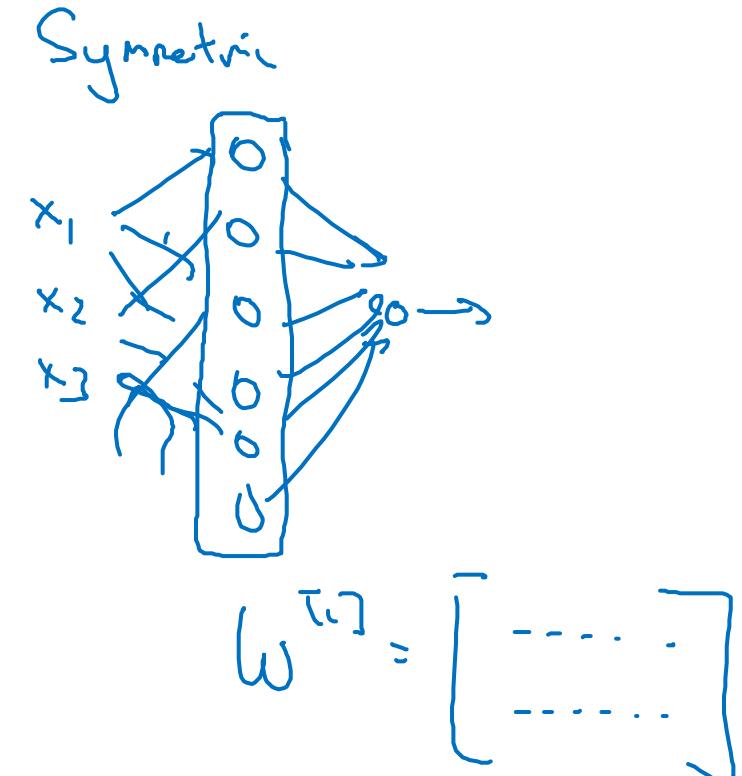
$$a_1^{[1]} = a_2^{[1]}$$

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

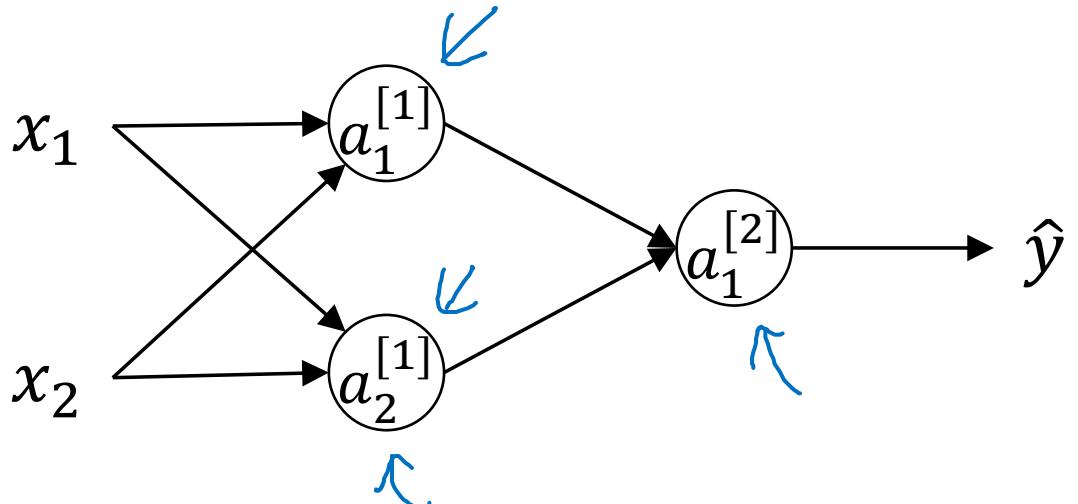
$$b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

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

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

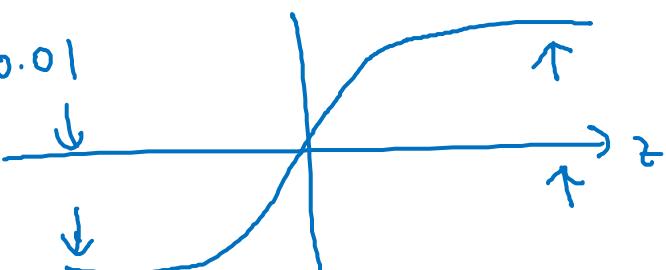


Random initialization



$$\begin{aligned} \rightarrow w^{[1]} &= \text{np.random.randn}(2, 2) \times \frac{0.01}{100?} \\ b^{[1]} &= \text{np.zeros}(2, 1) \\ w^{[2]} &= \text{np.random.randn}(1, 2) \times 0.01 \\ b^{[2]} &= 0 \end{aligned}$$

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



Copyright Notice

These slides are distributed under the Creative Commons License.

[DeepLearning.AI](#) makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite [DeepLearning.AI](#) as the source of the slides.

For the rest of the details of the license, see <https://creativecommons.org/licenses/by-sa/2.0/legalcode>

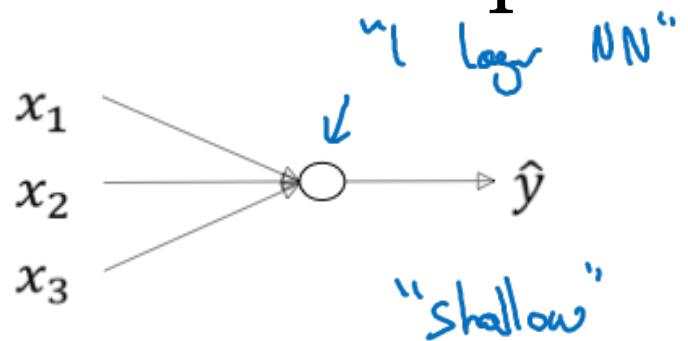


deeplearning.ai

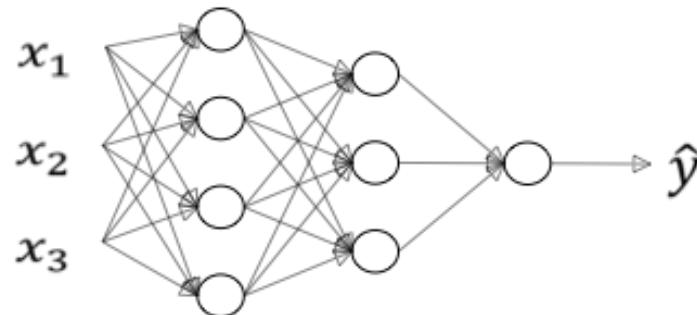
Deep Neural Networks

Deep L-layer Neural network

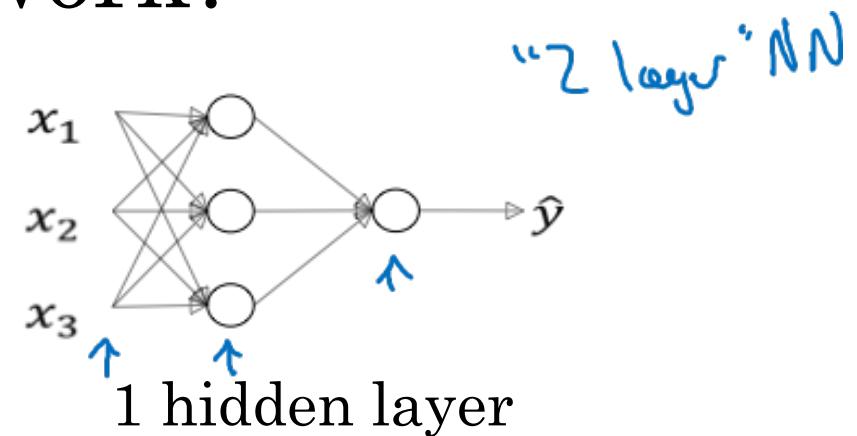
What is a deep neural network?



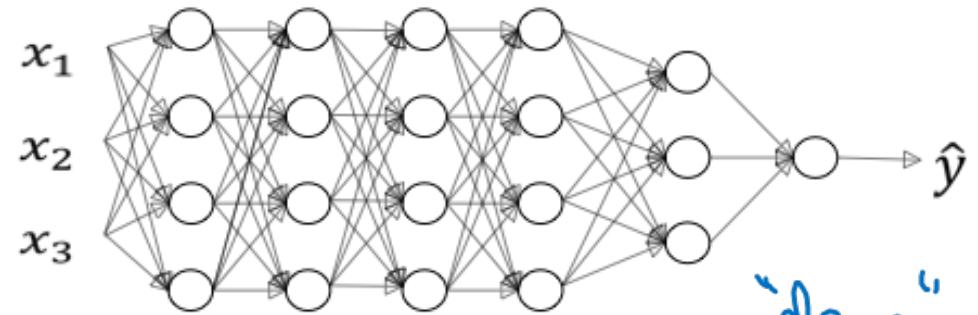
logistic regression



2 hidden layers



1 hidden layer

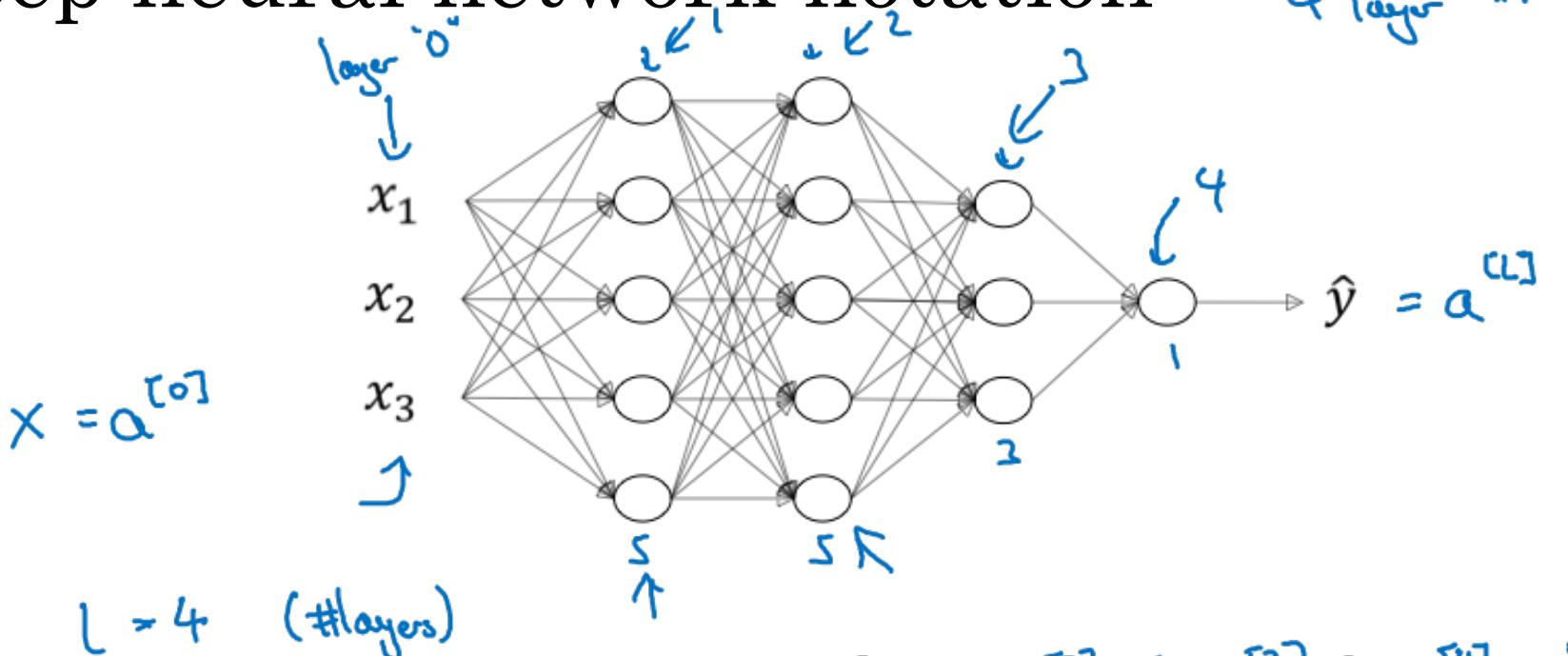


5 hidden layers

"deep"

Andrew
Ng

Deep neural network notation



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

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

Andrew
n.t

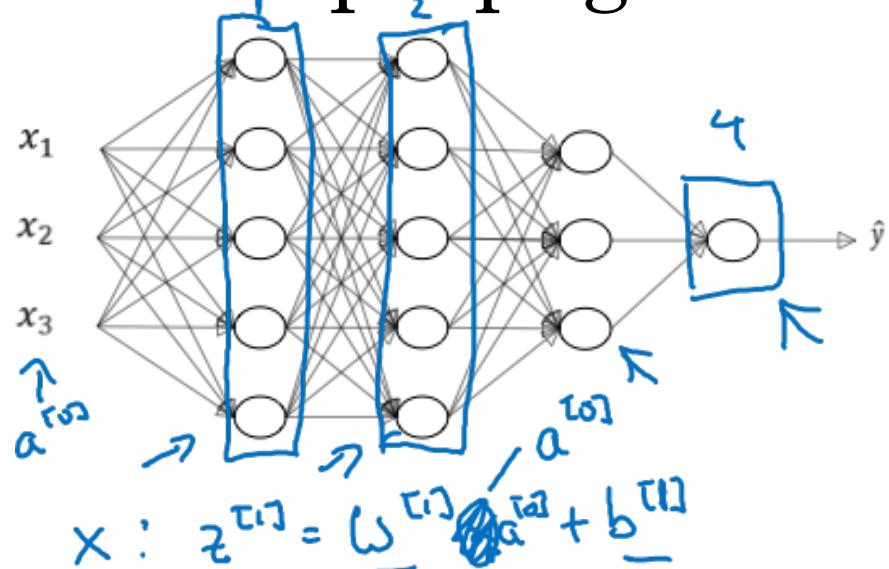


deeplearning.ai

Deep Neural Networks

Forward Propagation in a Deep Network

Forward propagation in a deep network



$A^{[0]} = X$

$\rightarrow z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$

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

Vertikal:

$\rightarrow z^{[1]} = (W^{[1]} \otimes A^{[0]} + b^{[1]})$

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

$\rightarrow z^{[2]} = (W^{[2]} \otimes A^{[1]} + b^{[2]})$

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

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

for $l=1..4$

Andrew



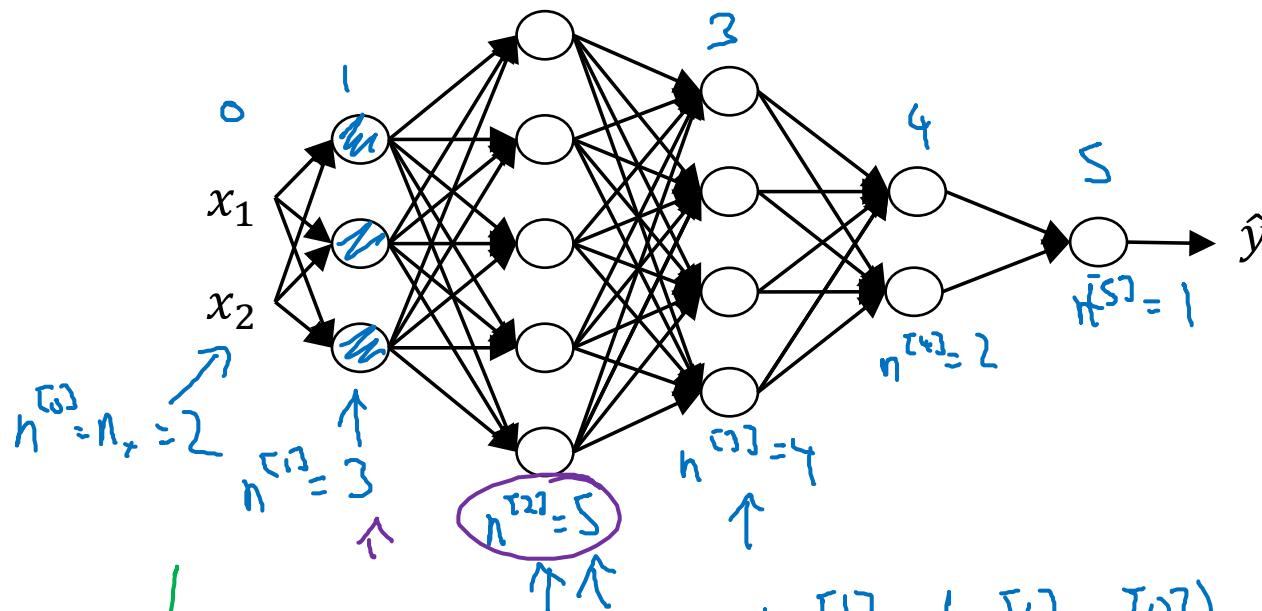
deeplearning.ai

Deep Neural Networks

Getting your matrix dimensions right

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

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



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

$$[\vdots] = [\vdots \vdots] \quad [\vdots]$$

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

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

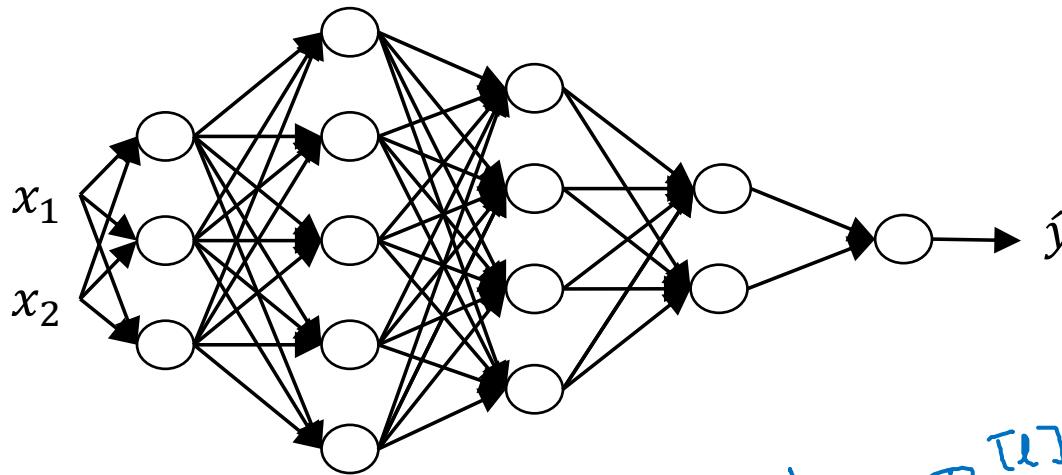
$$\begin{array}{c} z^{[2]} = \boxed{\underline{W^{[2]}} \cdot \underline{a^{[1]}}} + \boxed{\underline{b^{[2]}}} \\ \uparrow \quad \uparrow \quad \uparrow \\ (5,1) \quad (5,3) \quad (2,1) \quad (5,1) \\ (\underline{n^{[2]}}, 1) \quad (\underline{n^{[2]}}, \underline{n^{[3]}}) \quad (\underline{n^{[3]}},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

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

$L=5$

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

Vectorized implementation



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

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

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

$$\sum^{[l]} = w^{[l]} \cdot X + b^{[l]}$$

$$\underbrace{(n^{[l]}, m)}_{\uparrow} \quad \underbrace{(n^{[l]}, n^{[l+1]})}_{\uparrow} \quad \underbrace{(n^{[l]}, m)}_{\uparrow} \quad \underbrace{(n^{[l]}, 1)}_{(n^{[l]}, m)}$$

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

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

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

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

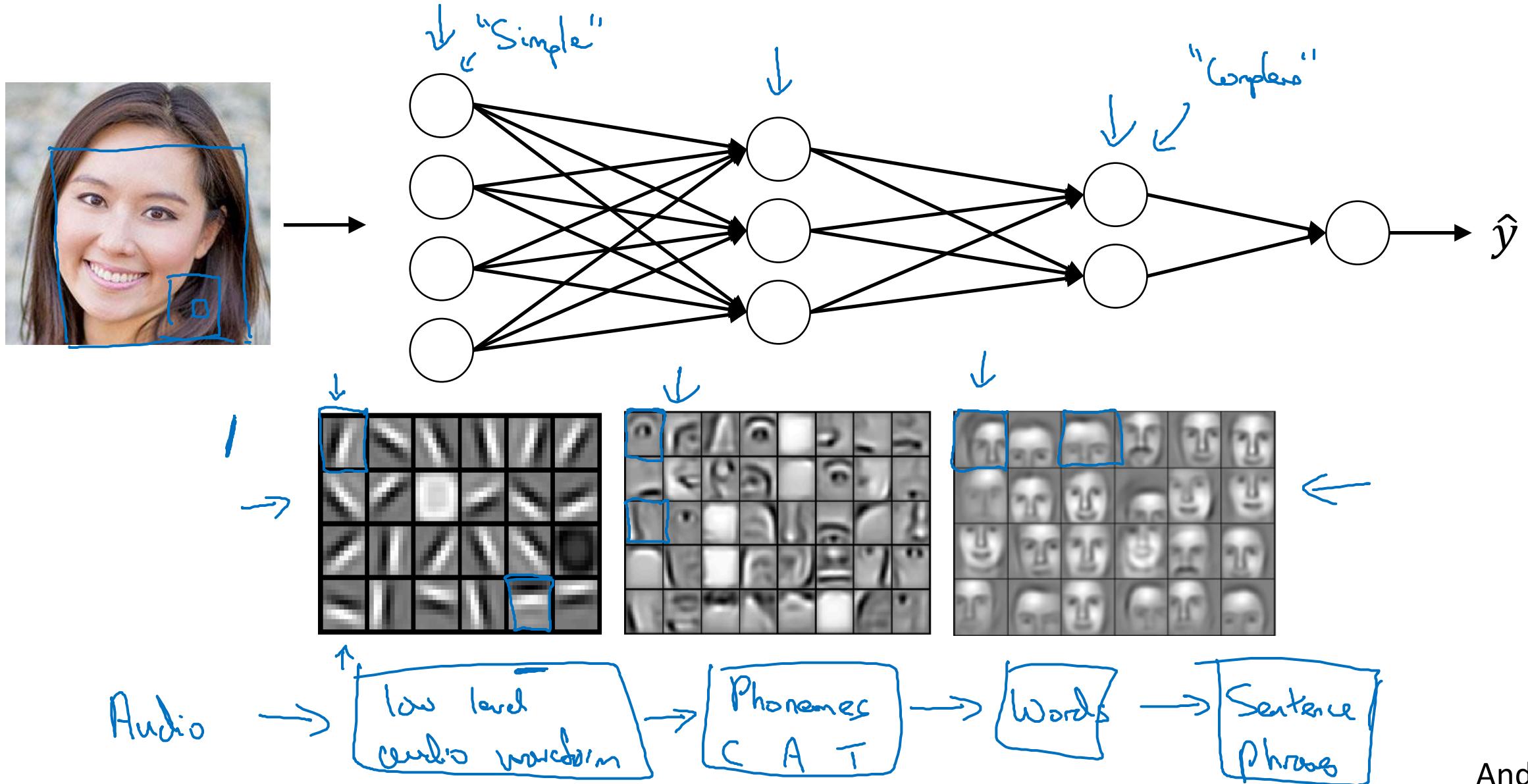


deeplearning.ai

Deep Neural Networks

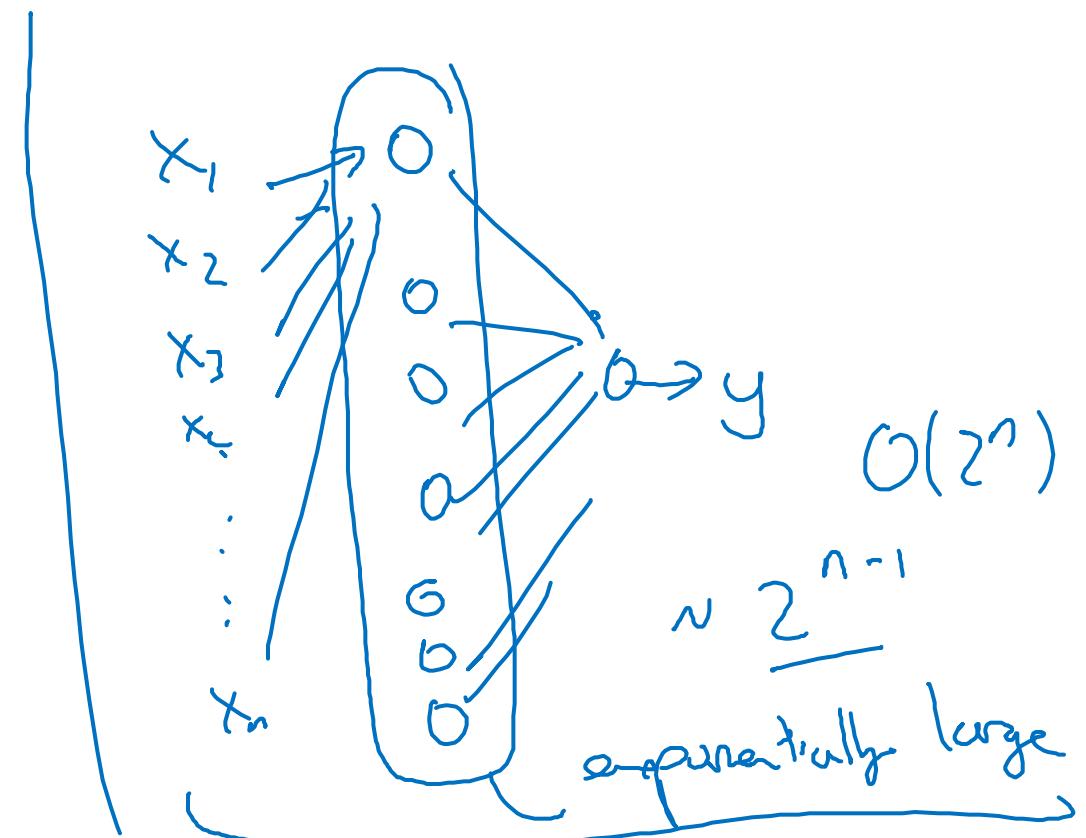
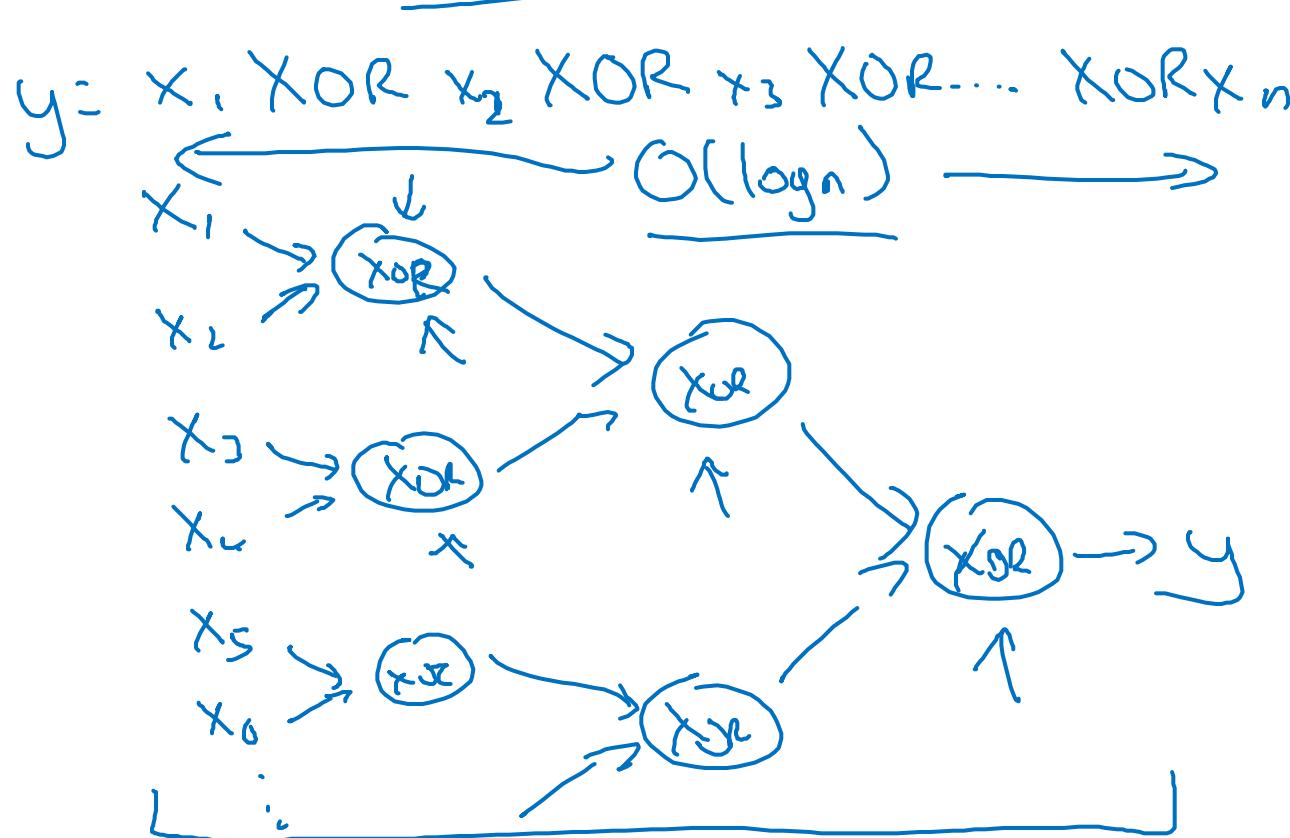
Why deep representations?

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

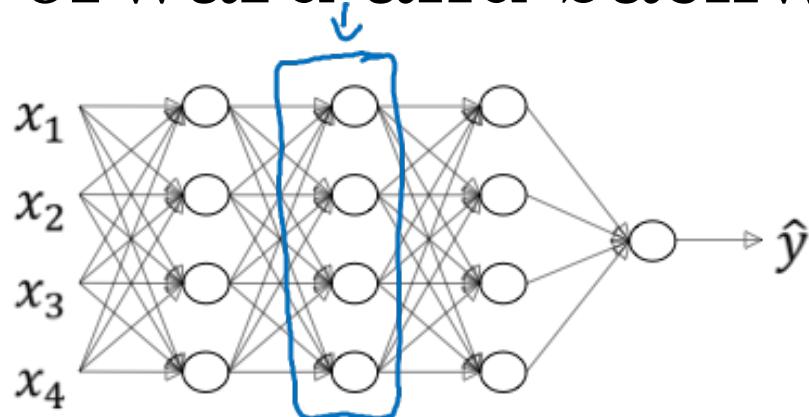


deeplearning.ai

Deep Neural Networks

Building blocks of
deep neural networks

Forward and backward functions



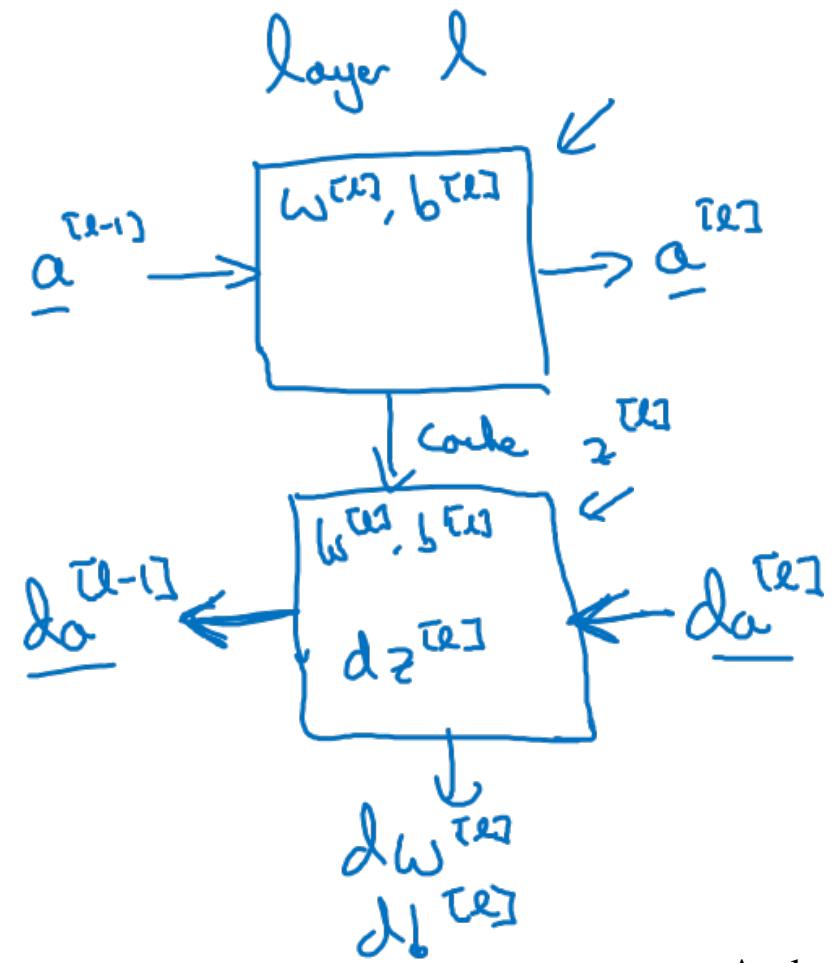
layer l : $w^{[l]}, b^{[l]}$

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

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

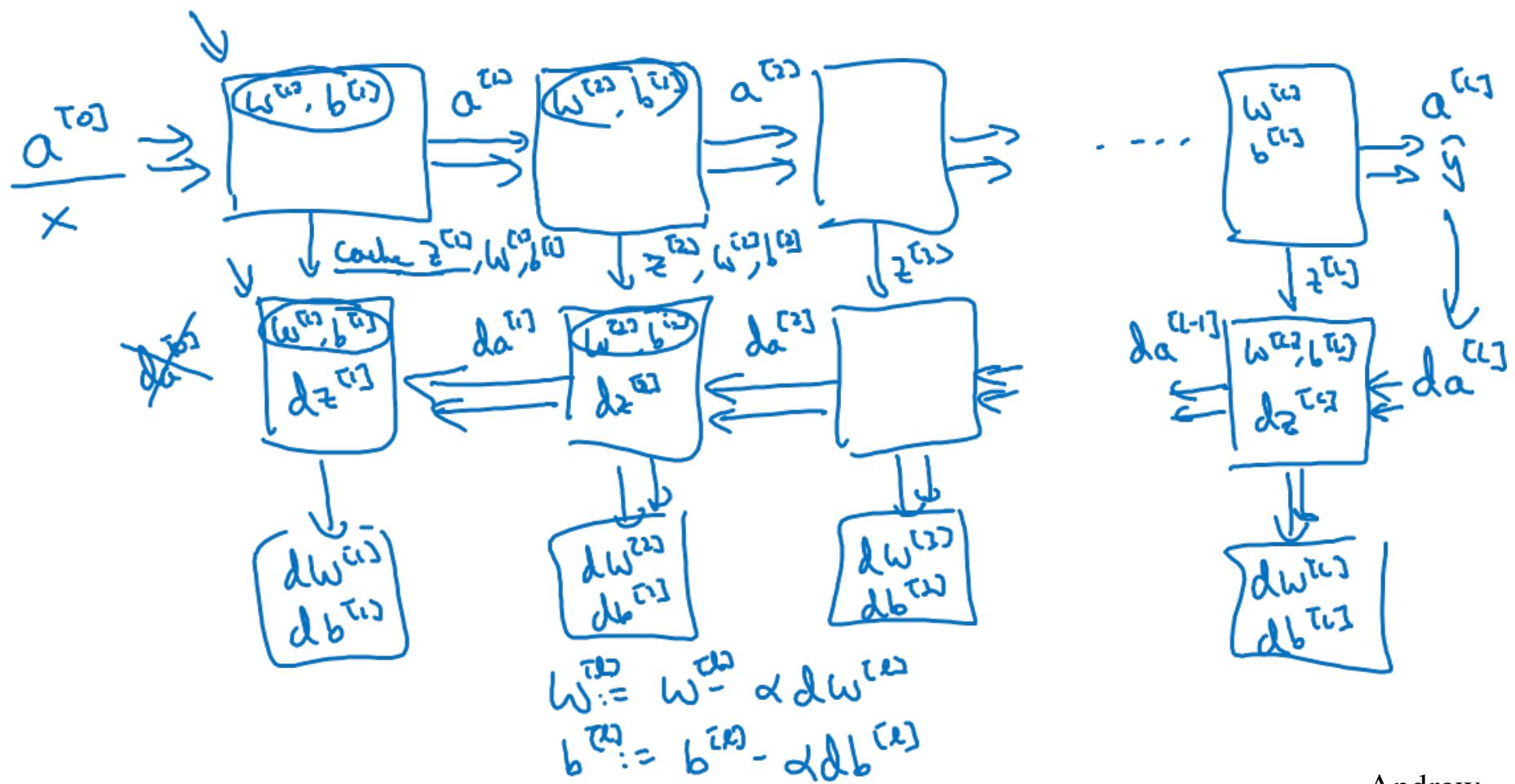
$$\underline{a}^{[l]} = g^{[l]}(\underline{z}^{[l]})$$

→ Backward: Input $\frac{da^{[l]}}{dz^{[l]}}$, output $\frac{da^{[l-1]}}{dw^{[l]}}$, $\frac{da^{[l-1]}}{db^{[l]}}$



Andrew
Ng

Forward and backward functions



Andrew
n



deeplearning.ai

Deep Neural Networks

Forward and backward propagation

Backward propagation for layer l

→ Input $da^{[l]}$

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

$$dz^{[l]} = \underbrace{da^{[l]}}_{\text{green}} * g^{[l]}(z^{[l]})$$

$$dW^{[l]} = dz^{[l]} \cdot \underbrace{a^{[l-1]}}_{\text{blue}}$$

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

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

$$dz^{[l-1]} = \underbrace{W^{[l]T}}_{\text{green}} \underbrace{dz^{[l]}}_{\text{green}} * g^{[l-1]}(z^{[l]})$$

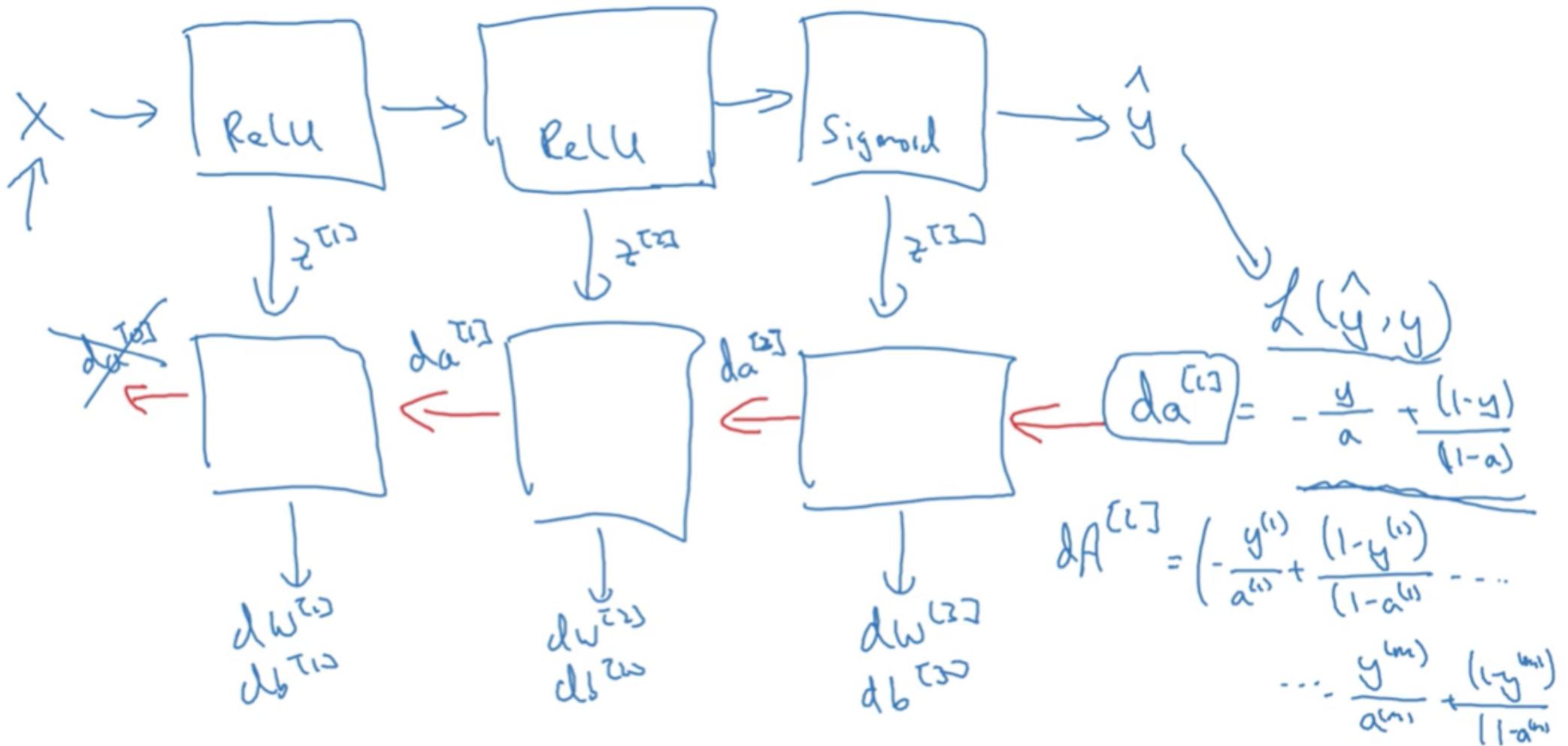
$$dz^{[l]} = dA^{[l]} * g^{[l]}(z^{[l]})$$

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

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

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

Summary





deeplearning.ai

Deep Neural Networks

Parameters vs Hyperparameters

What are hyperparameters?

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

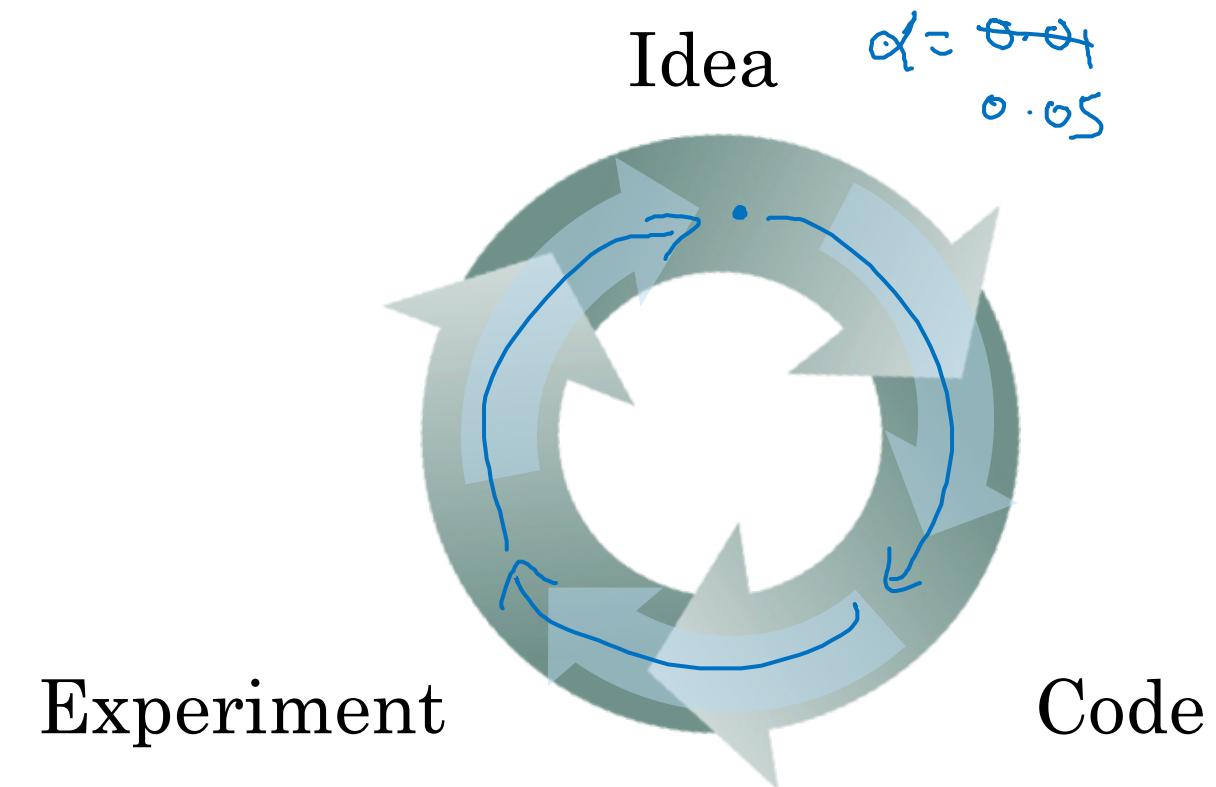
Hyperparameters:

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

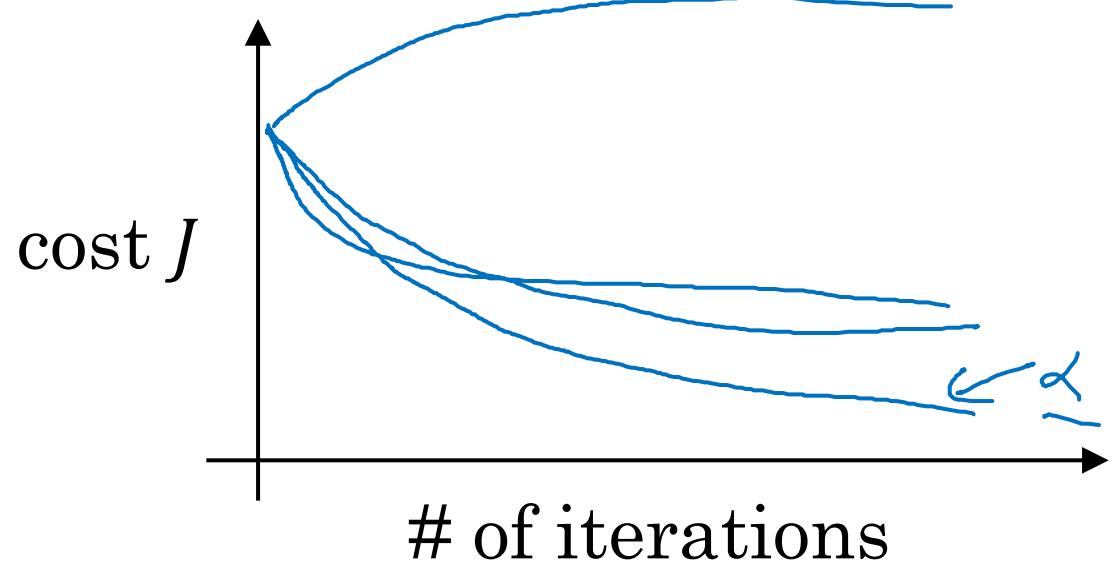
A curly brace groups the last four items.

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

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

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

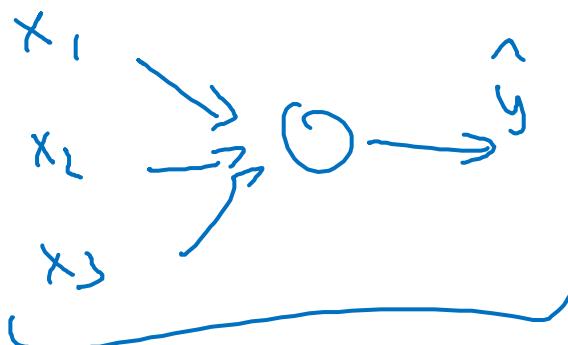
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

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

:

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

"It's like the brain"



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

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

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

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

$$\vdots$$

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

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

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

