Copyright Notice

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.

Andrew Ng

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
5. Natural Language Processing: Building sequence models

train/dev/test

end-to-end

CNN

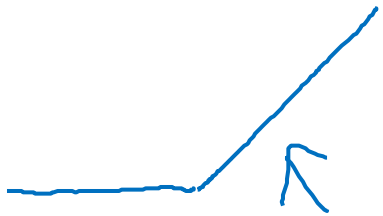
RNN, LSTM

deeplearning.ai

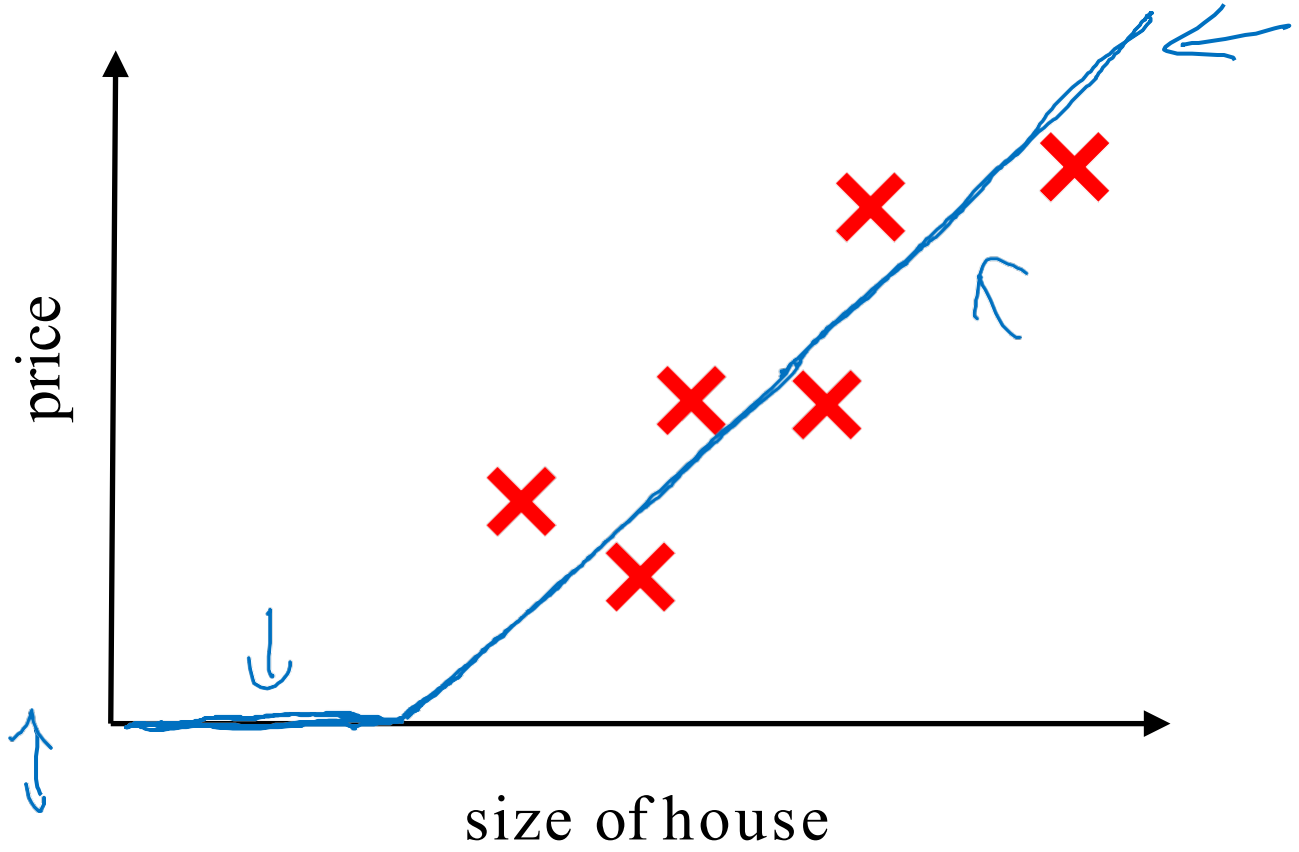
Introduction to Deep Learning

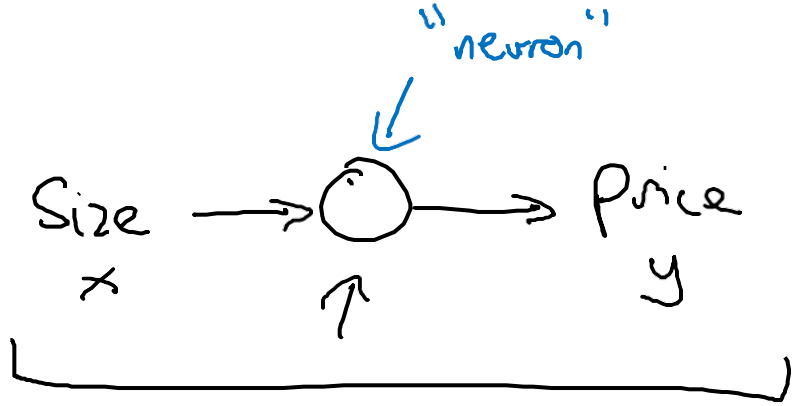
What is a Neural Network?

Housing Price Prediction

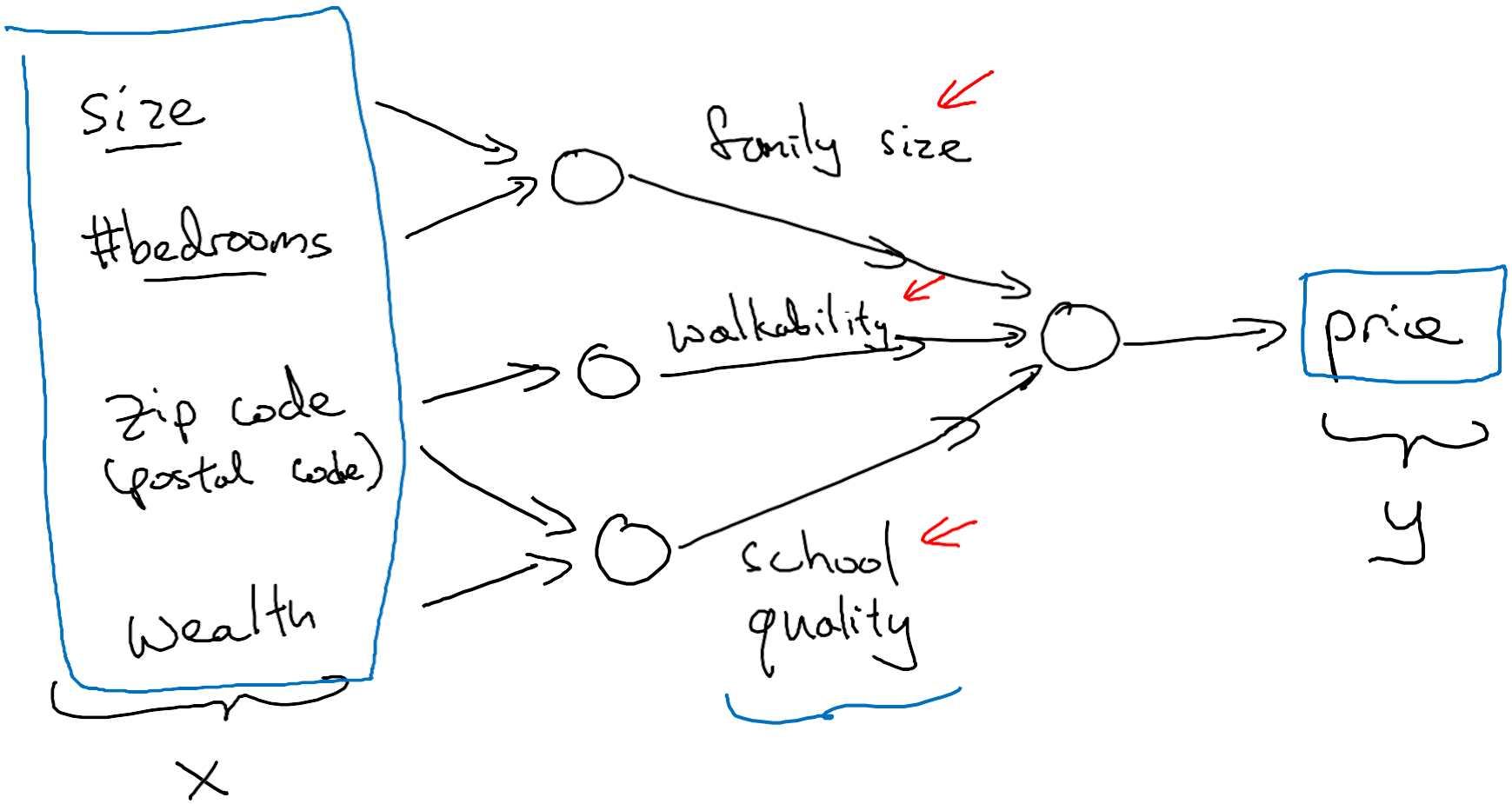


ReLU
Rectified
Linear
Unit



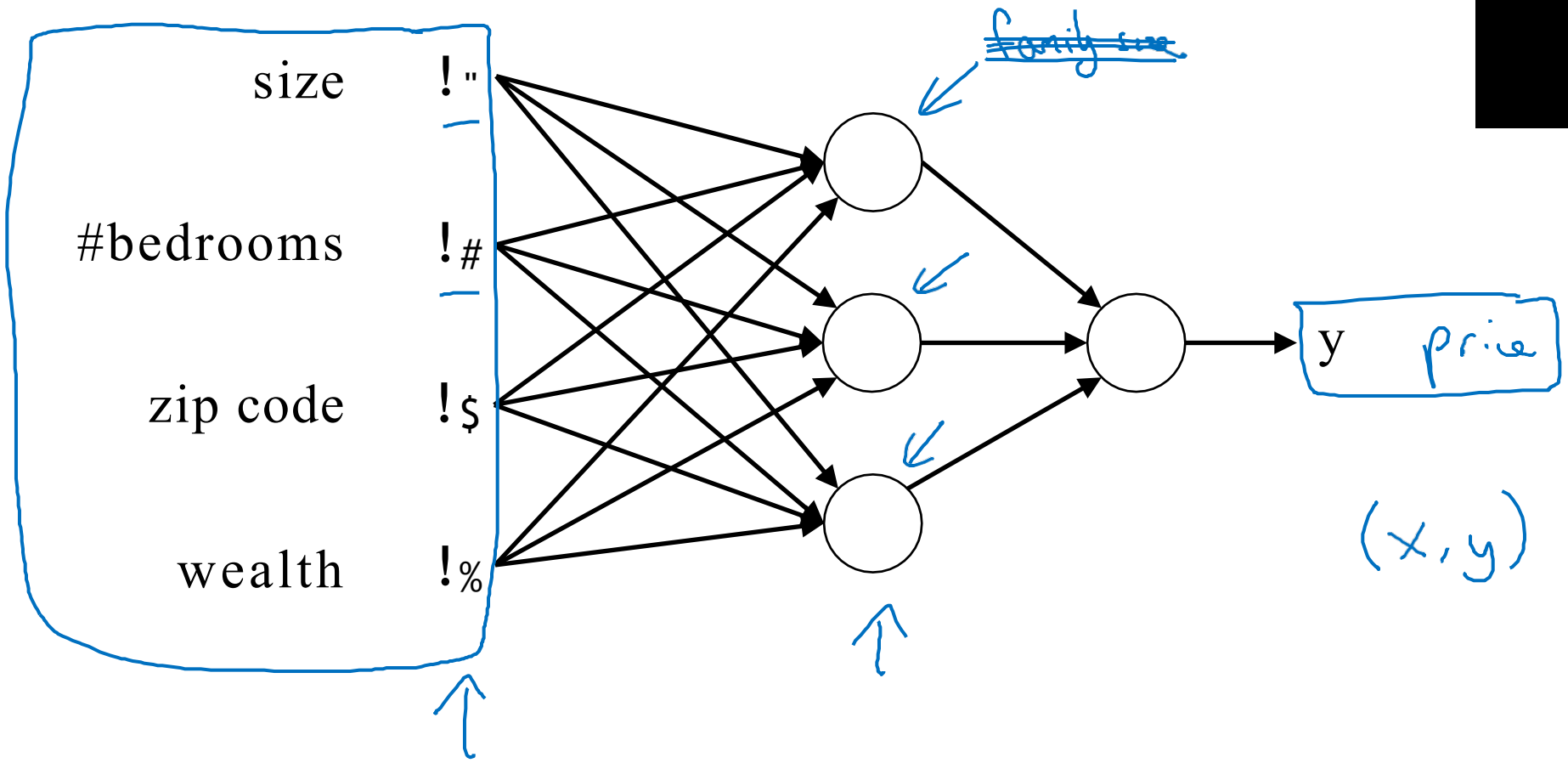


Housing Price Prediction



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

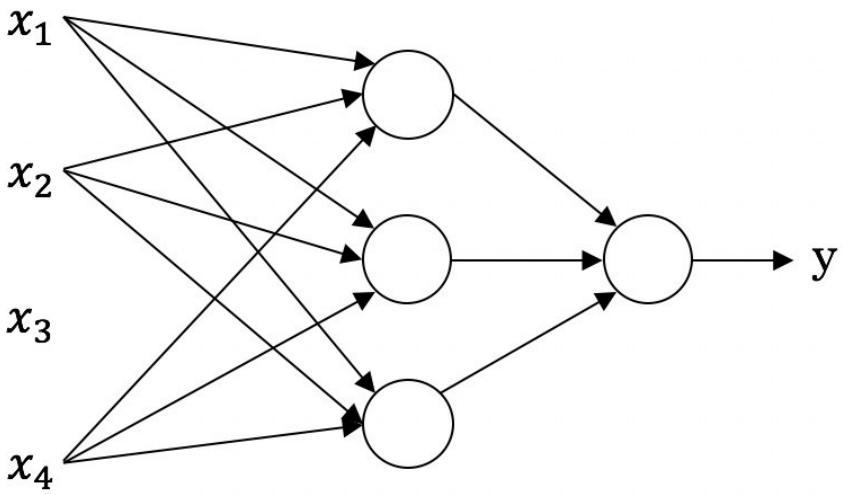
Standard NN

CNN

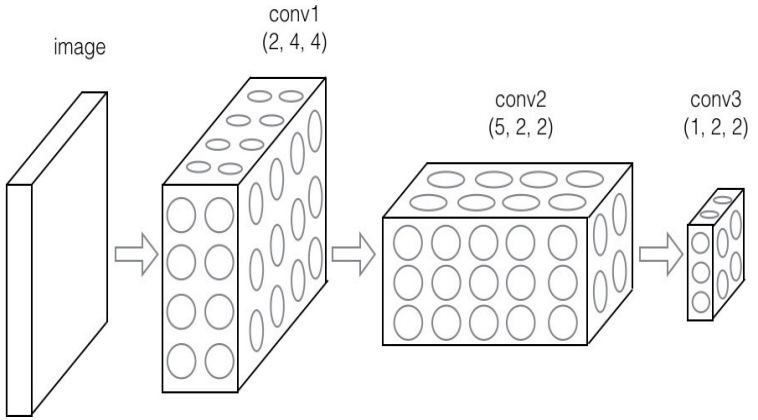
RNN

Custom/ Hybrid

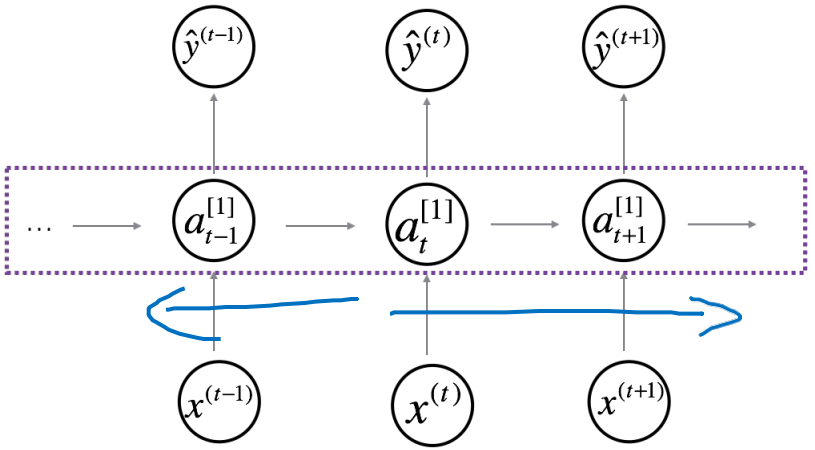
Neural Network examples



Standard NN



Convolutional NN



Recurrent NN

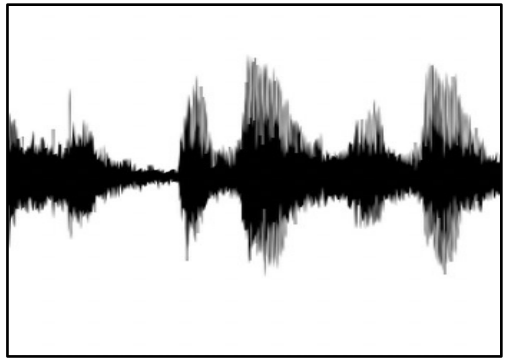
Supervised Learning

Structured Data

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

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

Unstructured Data



Audio

Image

Four scores and seven years ago…

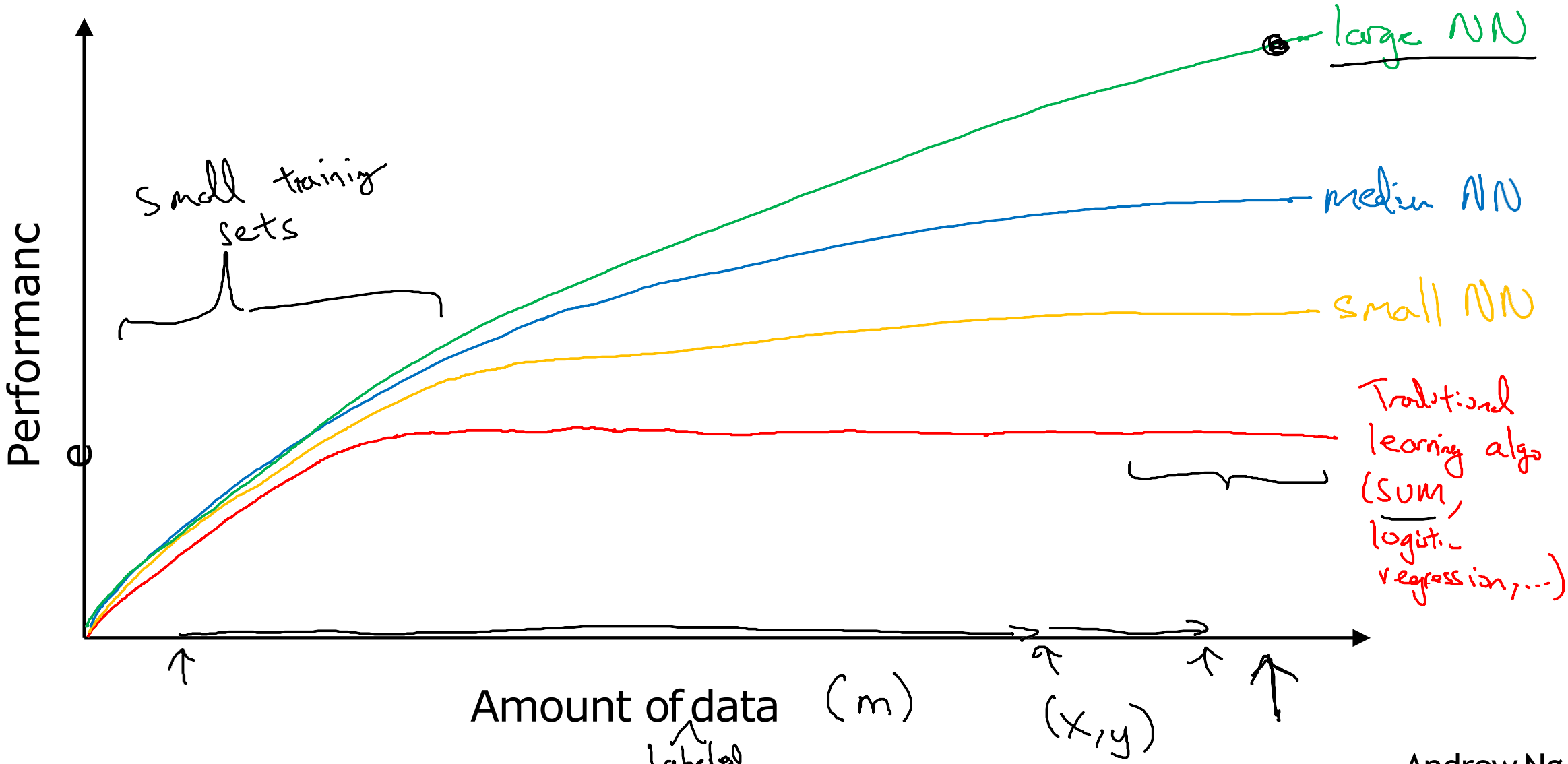
Text

deeplearning.ai

Introduction to Neural Networks

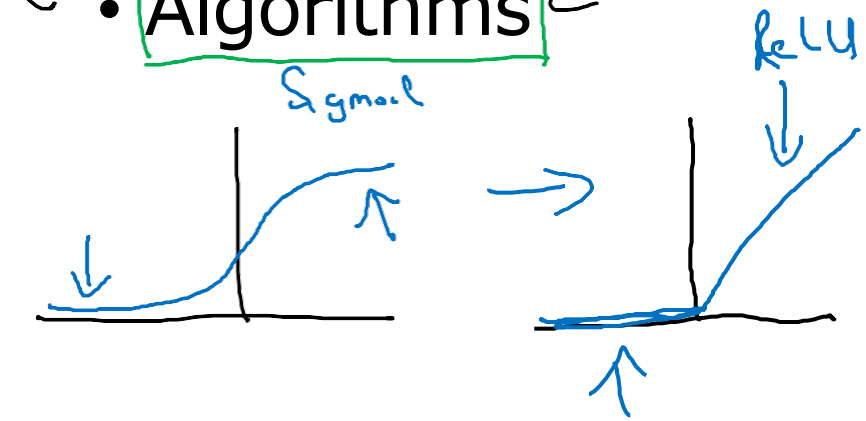
Why is Deep Learning taking off?

Scale drives deep learning progress

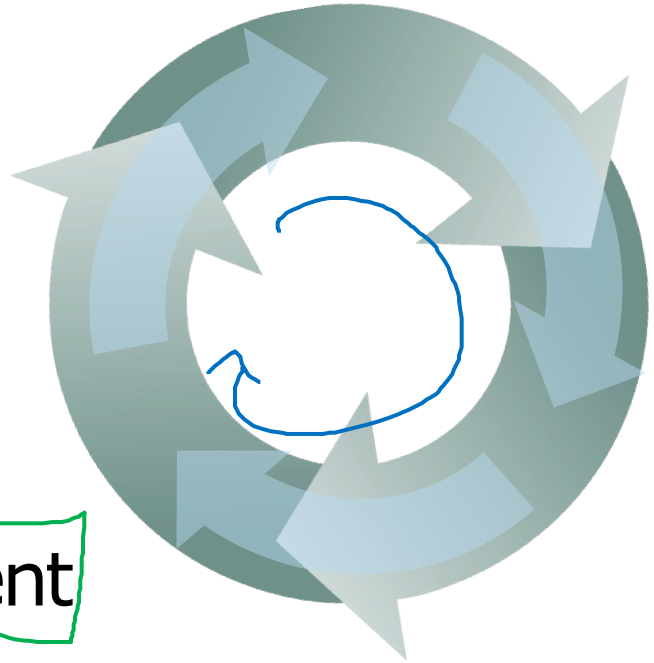


Scale drives deep learning progress

- Data
- Computation
- Algorithms



Idea



Experiment

Code

10 min
1 day
1 month

Andrew Ng

deeplearning.ai

Introduction to Neural Networks

About this Course

Courses in this 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
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

deeplearning.ai

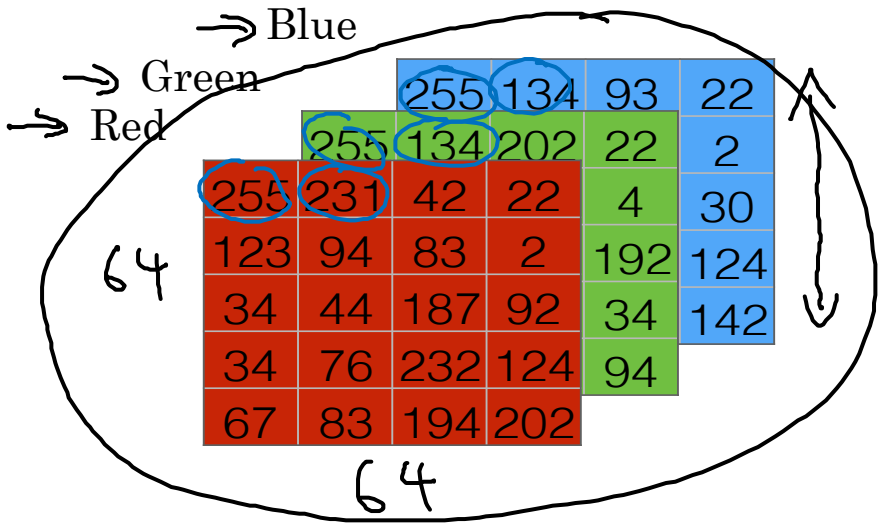
Basics of Neural Network Programming

Binary Classification

Binary Classification

$\rightarrow$ 1 (cat) vs 0 (non cat)

y



$$x = \begin{bmatrix} 255 \\ 231 \\ \vdots \\ \vdots \\ 255 \\ 134 \\ \vdots \end{bmatrix}$$

$64 \times 64 \times 3 = 12288$

$n = n_x = 12288$

$x \longrightarrow y$

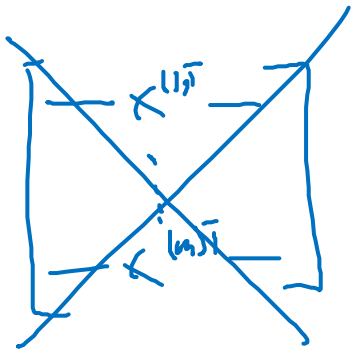
Notation

(x, y) $\quad x \in \mathbb{R}^{n_x}$, $y \in \{0, 1\}$

m training examples: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$

$m = m_{train}$ $\qquad$ m_{test} = #test examples.

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



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

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & \cdots & y^{(m)} \end{bmatrix}$$

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

$Y.shape = (1, m)$

Andrew Ng

deeplearning.ai

Basics of Neural Network Programming

Logistic Regression

Logistic Regression

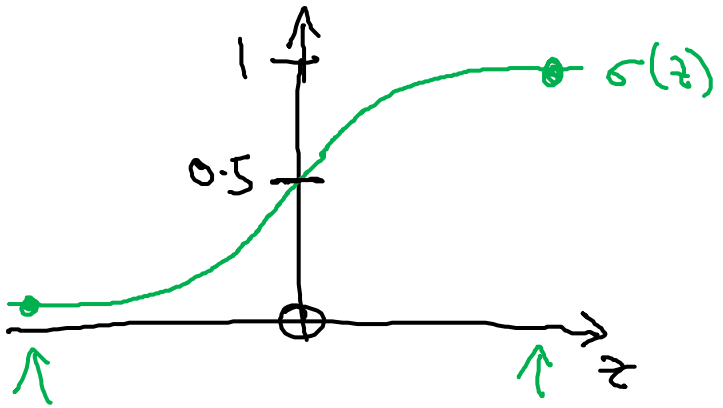
Given x, want $\hat{y} = P(y=1|x)$

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

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

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

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



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

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

$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{n_x} \end{bmatrix}$ $\begin{matrix} \} b \leftarrow \\ \\ \\ \\ \} w \leftarrow \end{matrix}$

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

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

If z large negative number

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

Andrew Ng

deeplearning.ai

Basics of Neural Network Programming

Logistic Regression cost function

Logistic Regression cost function

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

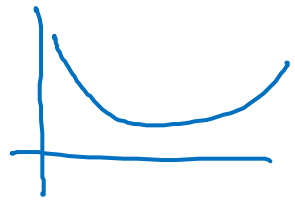
$z^{(i)} = w^T x^{(i)} + b$

$x^{(i)}$ $y^{(i)}$ $z^{(i)}$ i-th example.

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

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

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



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

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

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

Andrew Ng

deeplearning.ai

Basics of Neural Network Programming

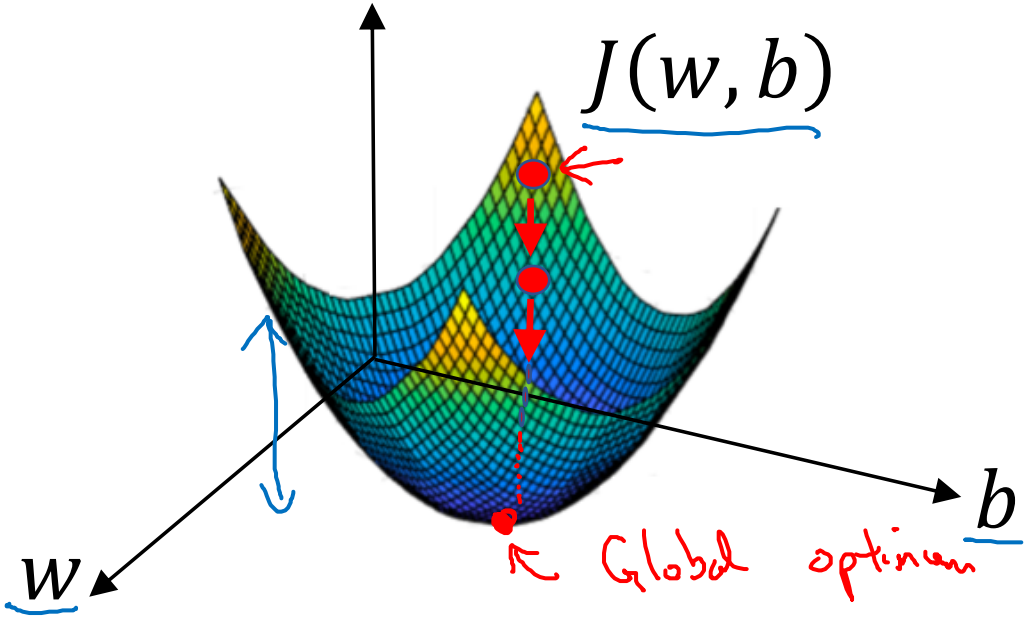
Gradient Descent

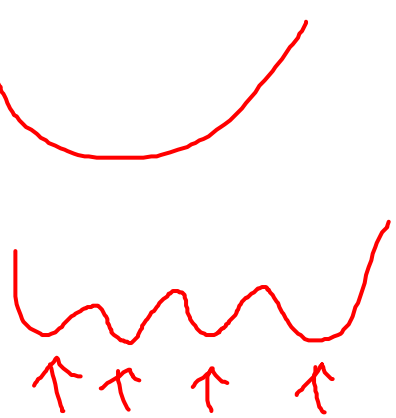
Gradient Descent

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

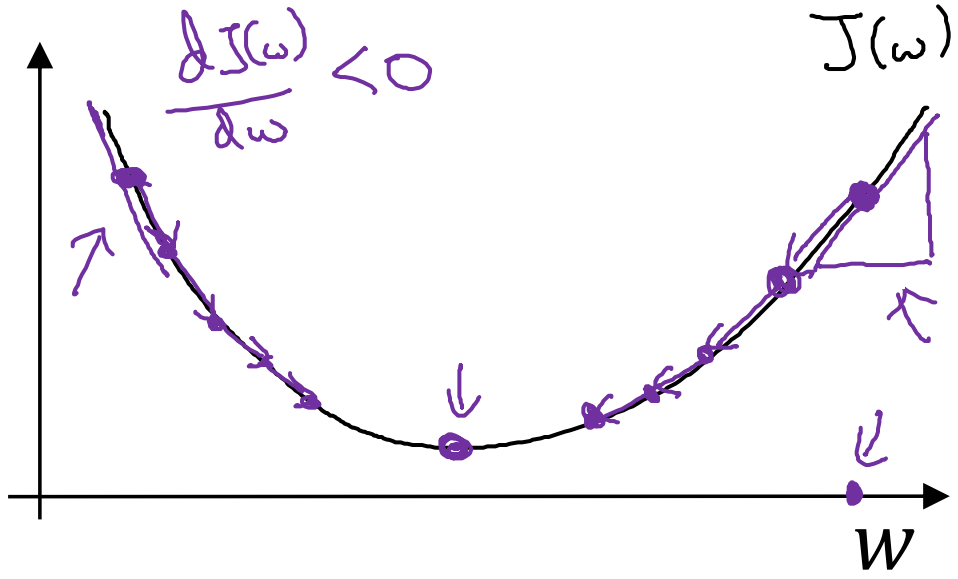
$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

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





Gradient Descent



Repeat {

$w := w - \alpha \frac{dJ(w)}{dw}$ ← learning rate

"dw"

$w := w - \alpha dw$

}

$\frac{dJ(w)}{dw} = ?$

$J(w,b)$

$w := w - \alpha \frac{\partial J(w,b)}{\partial w}$ ← $\frac{\partial J(w,b)}{\partial w}$ → dw

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

∂ "partial derivative"

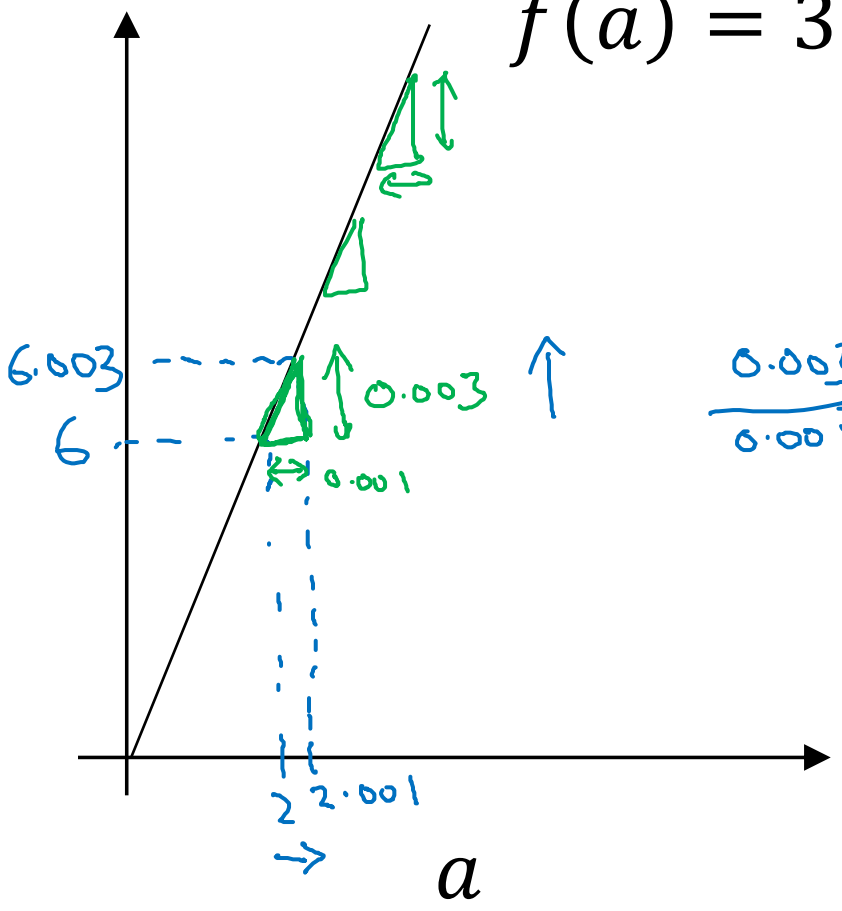
d ← J

deeplearning.ai

Basics of Neural Network Programming

Derivatives

Intuition about derivatives



$\frac{0.003}{0.001}$ $\frac{\text{height}}{\text{width}}$

→ slope (derivative) of $f(a)$ at $a=2$ is 3

→ $a = 2$ $f(a) = 6$

$a = 2.001$ $f(a) = 6.003$

→ $a = 5$ $f(a) = 15$

$a = 5.001$ $f(a) = 15.003$

slope at $a=5$ is also 3

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

0.001
0.000000001
0.000000000001

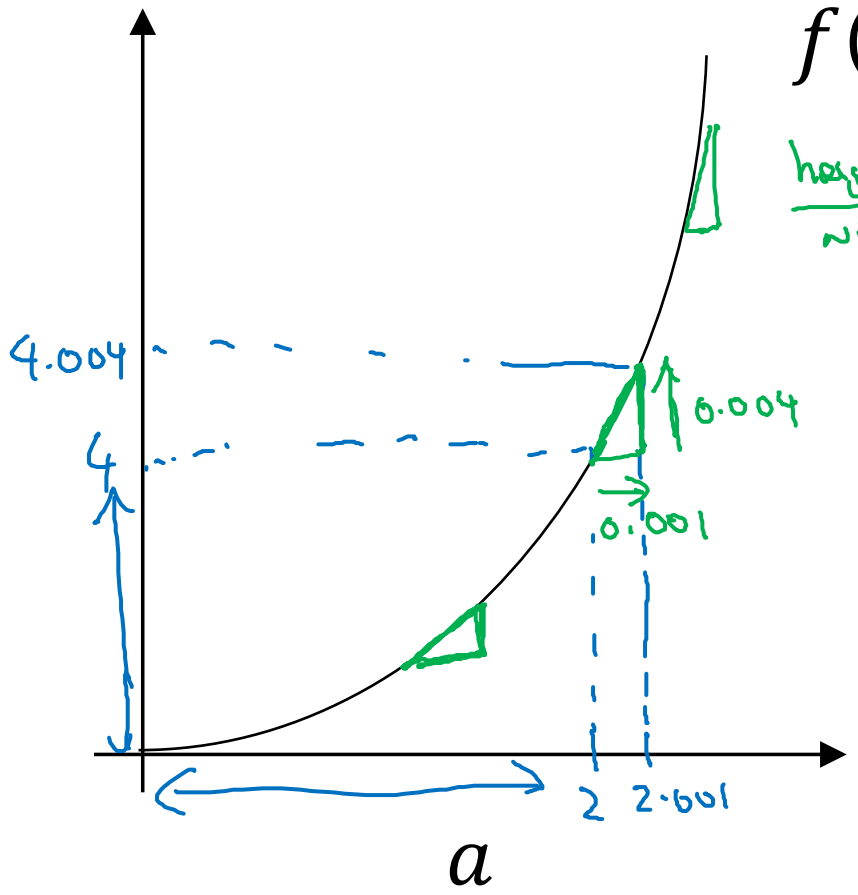
Andrew Ng

deeplearning.ai

Basics of Neural Network Programming

More derivatives examples

Intuition about derivatives



$$\frac{d}{da} a^2 = 2a$$

0.001

$(2a) \times 0.001$

0.001 ←

$0.000000\ldots01$ ←

$a = 2$

$a = 2.001$

$f(a) = 4$

$f(a) \approx 4.004$

$(4.004\underline{001})$

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

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

$a = 5$

$a = 5.001$

$f(a) = 25$

$f(a) \approx 25.010$

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

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

Andrew Ng

More derivative examples

$f(a) = a^2$

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

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

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

$f(a) = a^3$

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

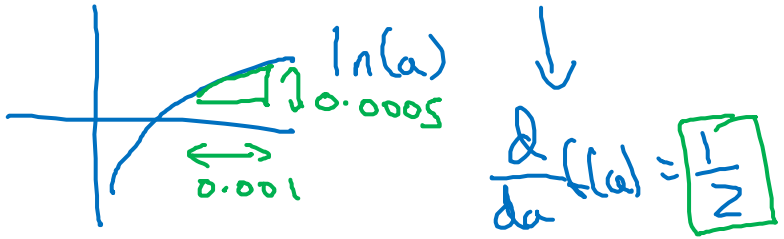
$a = 2$ $\quad f(a) = 8$

$a = 2.001$ $\quad f(a) \approx 8.012$

$f(a) = \log_e(a)$

$\ln(a)$

$\frac{d}{da} f(a) = \frac{1}{a}$



$\frac{d}{da} f(a) = \boxed{\frac{1}{2}}$

$a = 2$ $\quad f(a) \approx 0.69315$

$a = 2.001$ $\quad f(a) \approx 0.69365$

0.0005

0.0005

deeplearning.ai

Basics of Neural Network Programming

Computation Graph

Computation Graph

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

u

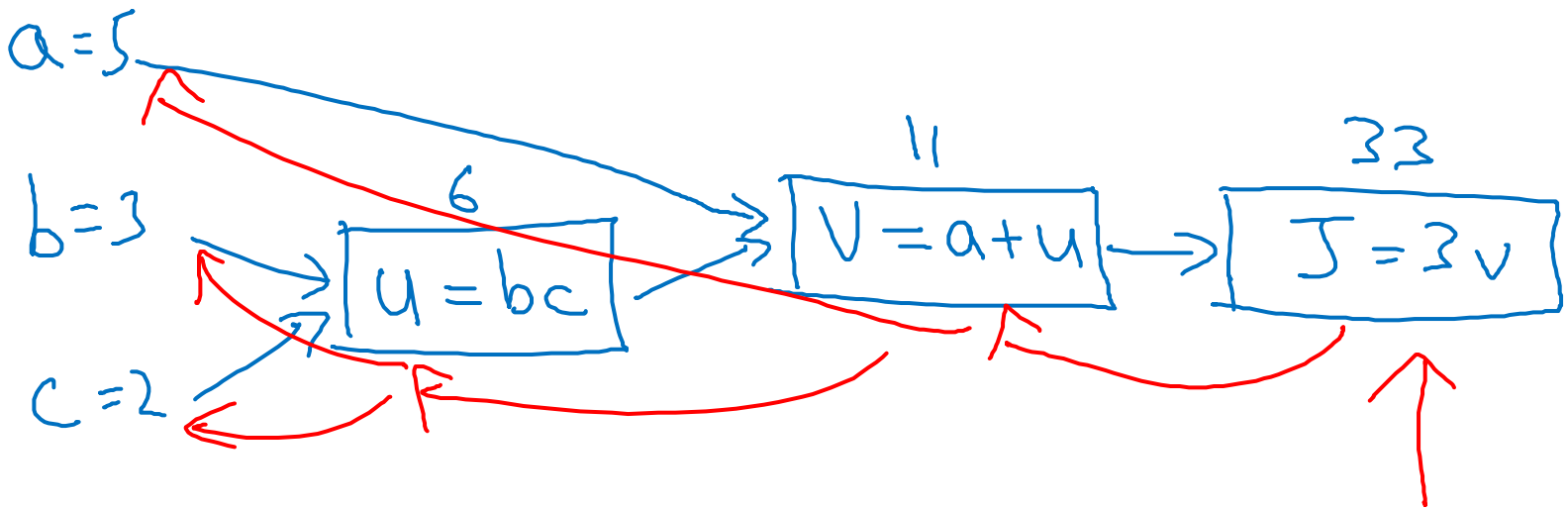
v

J

$u = bc$

$v = a+u$

$J = 3v$

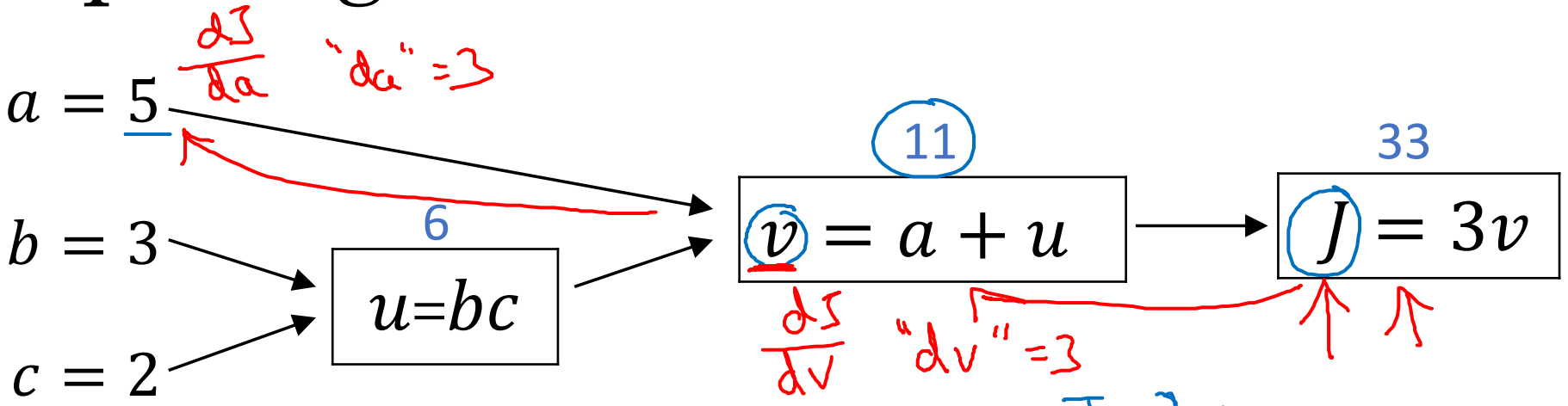


deeplearning.ai

Basics of Neural Network Programming

Derivatives with a Computation Graph

Computing derivatives



$\frac{dJ}{dv} = ? = 3$

$\frac{dJ}{da} = 3 = \frac{dJ}{dv}\frac{dv}{da}$

3×1

$\frac{dv}{da} = 1$

$a \to v \to J$

$J = 3v$

$v = 11 \to 11.001$

$J = 33 \to 33.003$

$a = 5 \to 5.001$

$\to v = 11 \to 11.001$

$J = 33 \to 33.003$

$\frac{d\,\text{FinalOutputVar}}{d\,\text{var}}$ → "dvar"

dJdvar

$f(a) = 3a$

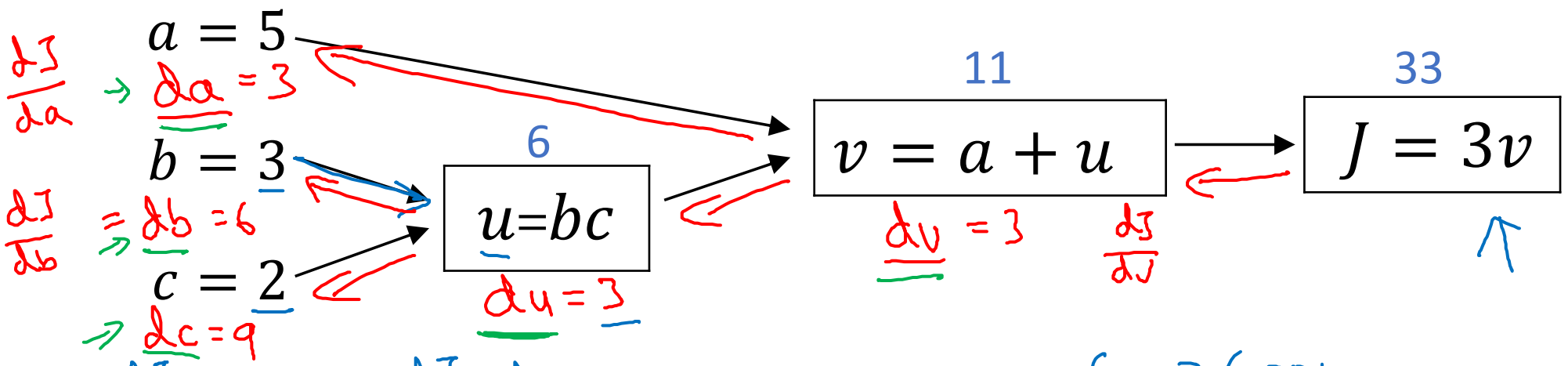
$\frac{df(a)}{da} = \frac{df}{da} = 3$

$J = 3v$

$\frac{dJ}{dv} = 3$

Andrew Ng

Computing derivatives



$$\frac{dJ}{du} = 3 = \frac{dJ}{dv} \cdot \frac{dv}{du}$$

($\frac{dJ}{dv} = 3$, $\frac{dv}{du} = 1$)

$$\frac{dJ}{db} = \frac{dJ}{du} \cdot \frac{du}{db} = 6$$

($\frac{dJ}{du} = 3$, $\frac{du}{db} = 2$)

$$\frac{dJ}{dc} = \frac{dJ}{du} \cdot \frac{du}{da} = 9$$

($\frac{dJ}{du} = 3$, $\frac{du}{da} = 3$)

$u = 6 \rightarrow 6.001$

$v = 11 \rightarrow 11.001$

$J = 33 \rightarrow 33.003$

$b = 3 \rightarrow 3.001$

$u = b \cdot c = 6 \rightarrow 6.002$

$J = 33.006$

$c = 2$

$.006$

$V = 11.002$

$J = 3V$

deeplearning.ai

Basics of Neural Network Programming

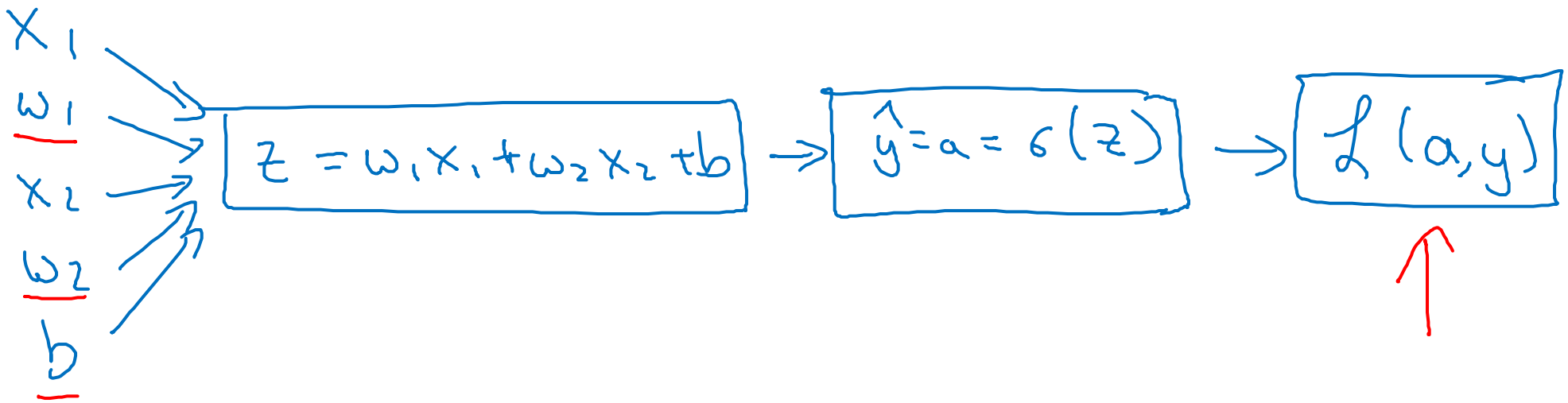
Logistic Regression Gradient descent

Logistic regression recap

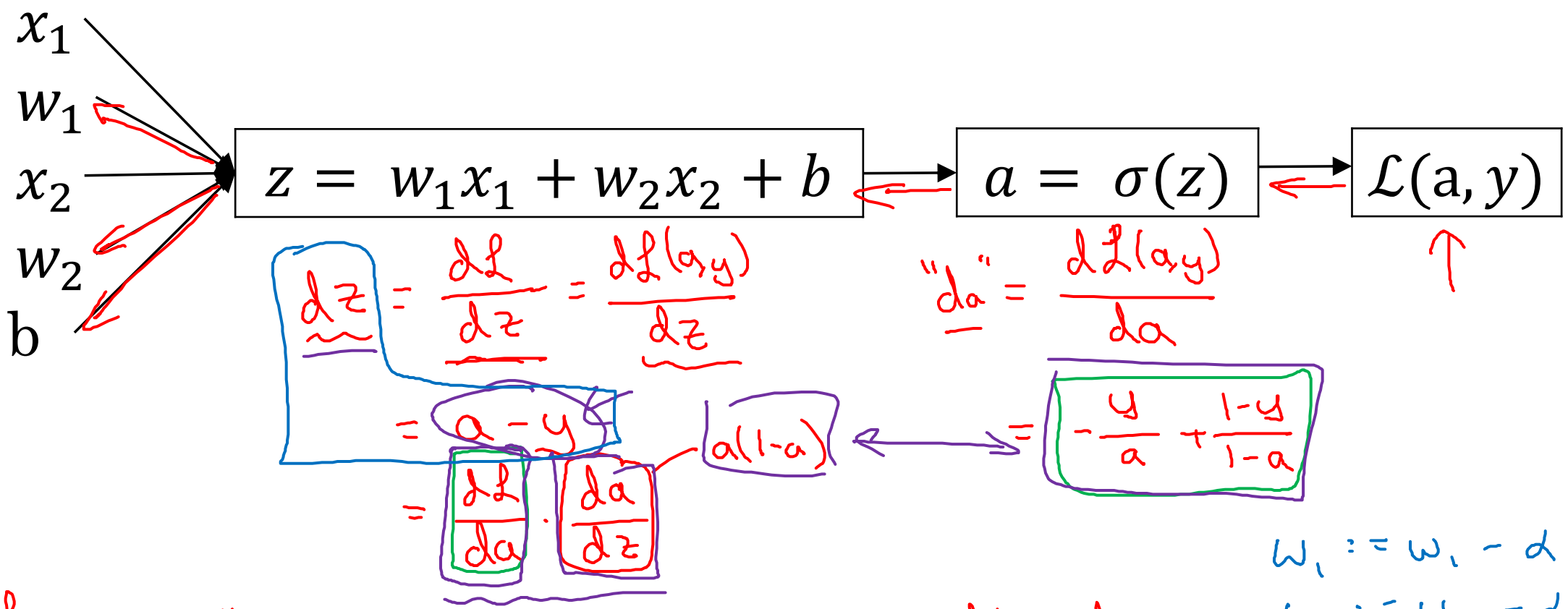
$$z = w^T x + b$$

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

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



Logistic regression derivatives



Andrew Ng

deeplearning.ai

Basics of Neural Network Programming

Gradient descent on m examples

Logistic regression on m examples

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

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

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

$dw_1^{(i)}$, $dw_2^{(i)}$, $db^{(i)}$

$$\frac{\partial}{\partial w_1} J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial w_1} \mathcal{L}(a^{(i)}, y^{(i)})$$

$dw_1^{(i)}$ — $(x^{(i)}, y^{(i)})$

Andrew Ng

Logistic regression on m examples

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

For $i=1$ to m

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

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

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

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

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

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

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

dw_3 ... dw_n

$n=2$

$J /= m$

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

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

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

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

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

Vectorization

deeplearning.ai

Basics of Neural Network Programming

Vectorization

What is vectorization?

$z = w^T x + b$

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

Non-vectorized:

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

Vectorized

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

$w^T x$

→ GPU
→ CPU

} SIMD - single instruction multiple data.

deeplearning.ai

Basics of Neural Network Programming

More vectorization examples

Neural network programming guideline

Whenever possible, avoid explicit for-loops.

$u = Av$

$u_i = \sum_j A_{ij} v_j$

$u = np.zeros((n,1))$

for i ...

for j ...

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

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

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
1/v
```

Andrew Ng

Logistic regression derivatives

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

$dw = np.zeros((n_x, 1))$

for $i = 1$ to n:

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

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

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

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

for $i = 1 \ldots n_x$:

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

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

$n_x = 2$

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

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

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

$dw /= m$.

deeplearning.ai

Basics of Neural Network Programming

Vectorizing Logistic Regression

Vectorizing Logistic Regression

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

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

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

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

$w^T = \begin{bmatrix} - & x^{(1)} & - \\ - & x^{(2)} & - \\ & \vdots & \\ - & x^{(n_x)} & - \end{bmatrix}$

$Z = \begin{bmatrix} z^{(1)} & z^{(2)} & \cdots & z^{(m)} \end{bmatrix} = w^T X + \underbrace{[b \; b \cdots b]}_{1 \times m} = \begin{bmatrix} w^T x^{(1)} + b & w^T x^{(2)} + b & \cdots & w^T x^{(m)} + b \end{bmatrix}$

$Z = np.dot(w.T, X) + b$ $b \in \mathbb{R}$ $(1,1)$ "Broadcasting"

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

Andrew Ng

deeplearning.ai

Basics of Neural Network Programming

Vectorizing Logistic Regression's Gradient Computation

Vectorizing Logistic Regression

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

$dZ = [dz^{(1)}\ dz^{(2)} \cdots dz^{(m)}]$ ←

$1 \times m$

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

→ $dZ = A - Y = [a^{(1)} - y^{(1)} \quad a^{(2)} - y^{(2)} \quad \cdots]$

→ $dw = 0$

$dw += X^{(1)} dz^{(1)}$

$dw += X^{(2)} dz^{(2)}$

⋮

$dw /= m$

$db = 0$

$db += dz^{(1)}$

$db += dz^{(2)}$

⋮

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

$db /= m$.

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

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

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

$$= \frac{1}{m} \begin{bmatrix} | & & | \\ x^{(1)} & \cdots & x^{(m)} \\ | & & | \end{bmatrix} \begin{bmatrix} dz^{(1)} \\ \vdots \\ dz^{(m)} \end{bmatrix}$$

$$= \frac{1}{m} \left[x^{(1)} dz^{(1)} + \cdots + x^{(m)} dz^{(m)} \right]$$

$n \times 1$

Andrew Ng

Implementing Logistic Regression

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

for i = 1 to m:

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

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

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

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

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

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

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

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

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

$db = db/m$

for iter in range(1000):

$Z = w^T X + b$

$= np.dot(w.T, X) + b$ 1xM

$A = \sigma(Z)$ 1xM

$dZ = A - Y$ 1xM

$dw = \frac{1}{m} X dZ^T$ Nx1

$db = \frac{1}{m} np.sum(dZ)$ scalar

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

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

Andrew Ng

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

$= A$ (3,4)

59 cal

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

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))
```

(3,4) / (1,4)

Broadcasting example

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

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix} + \begin{bmatrix}100 & 200 & 300\\100 & 200 & 300\end{bmatrix}$$

(m, n) (2,3) (1, n) ⇝ (m, n) (2,3)

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix} + \begin{bmatrix}100\\200\end{bmatrix} \begin{bmatrix}100 & 100\\200 & 100\end{bmatrix} =$$

(m, n) (m, 1) ⇝ (m, n)

General Principle

(m, n) matrix

$+ \; - \; * \; /$

$(1, n) \rightsquigarrow (m, n)$

$(m, 1) \rightsquigarrow (m, n)$

$(m, 1) + \mathbb{R}$

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

$[1 \; 2 \; 3] + 100 = [101 \quad 102 \quad 103]$

Matlab/Octave: bsxfun

deeplearning.ai

Basics of Neural Network Programming

Explanation of logistic regression cost function (Optional)

Logistic regression cost function

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

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

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

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

Logistic regression cost function

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

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

$p(y|x)$

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

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

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

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

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

Andrew Ng

Cost on m examples

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

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

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

Maximum likelihood estimation

Cost: (minimize)

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

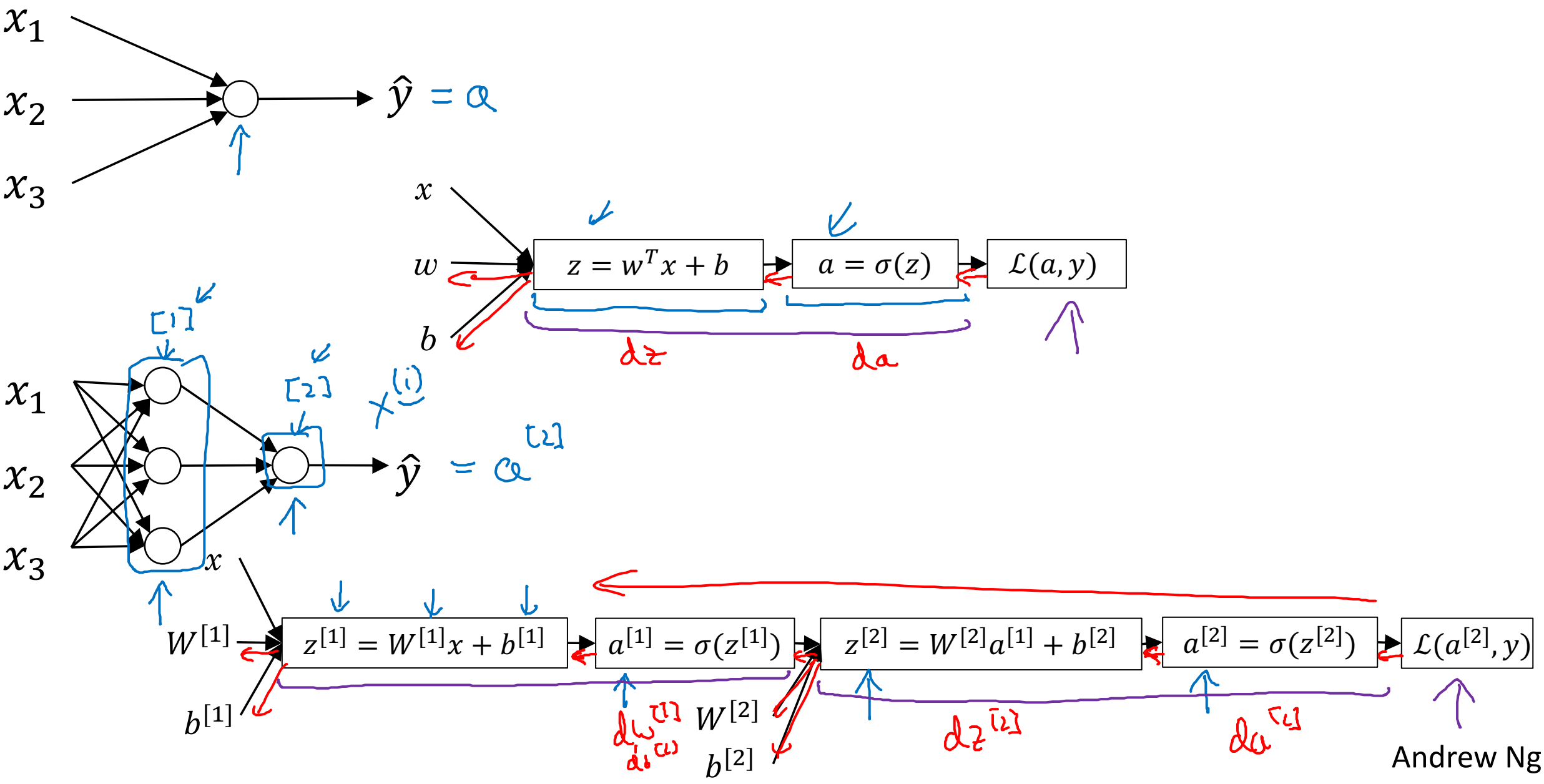
Copyright Notice

deeplearning.ai

One hidden layer Neural Network

Neural Networks Overview

What is a Neural Network?



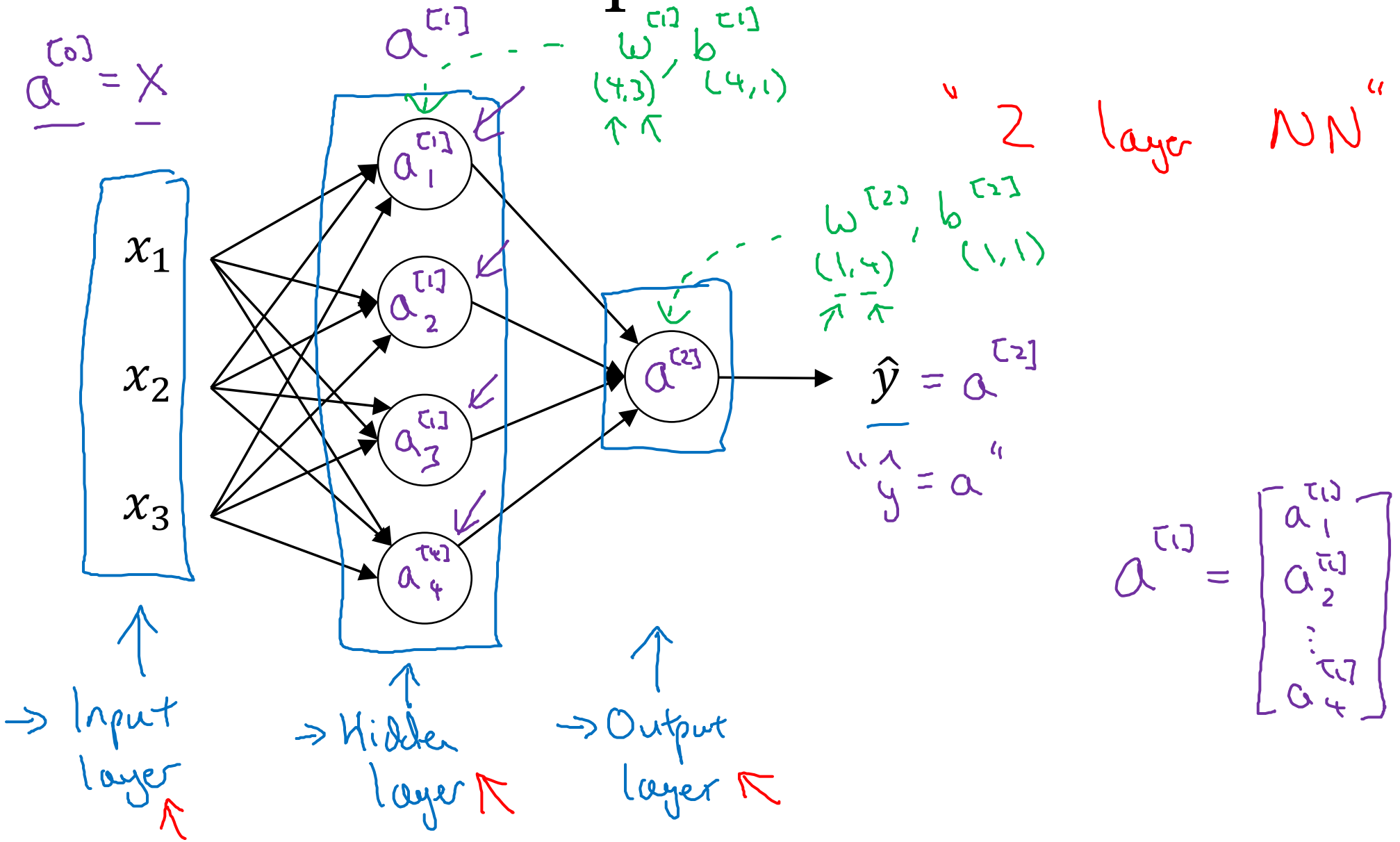
Andrew Ng

deeplearning.ai

One hidden layer Neural Network

Neural Network Representation

Neural Network Representation



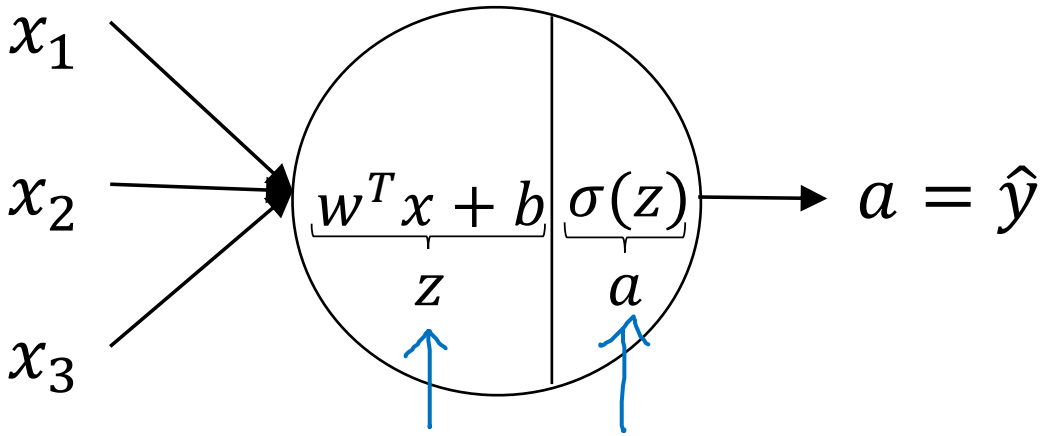
Andrew Ng

deeplearning.ai

One hidden layer Neural Network

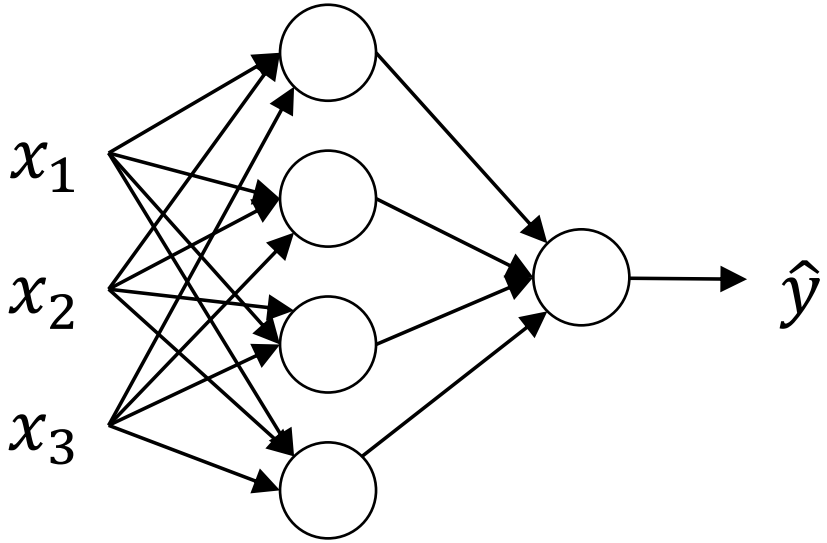
Computing a Neural Network's Output

Neural Network Representation

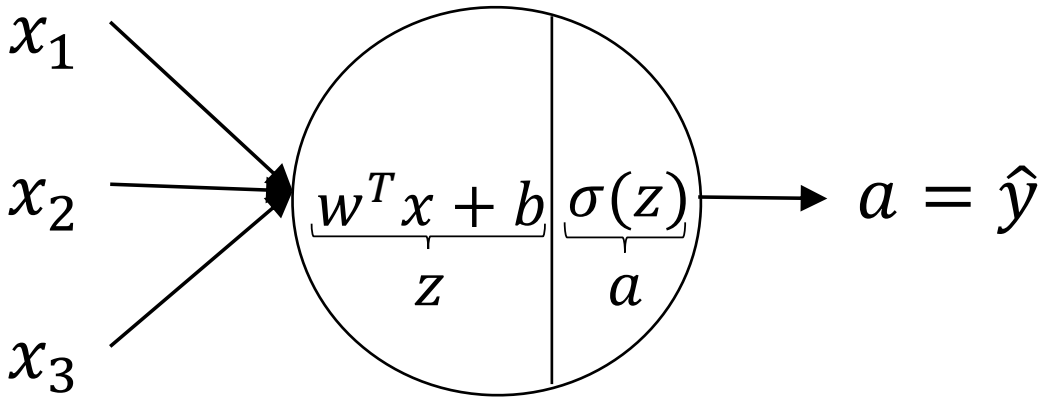


$$z = w^T x + b$$

$$a = \sigma(z)$$

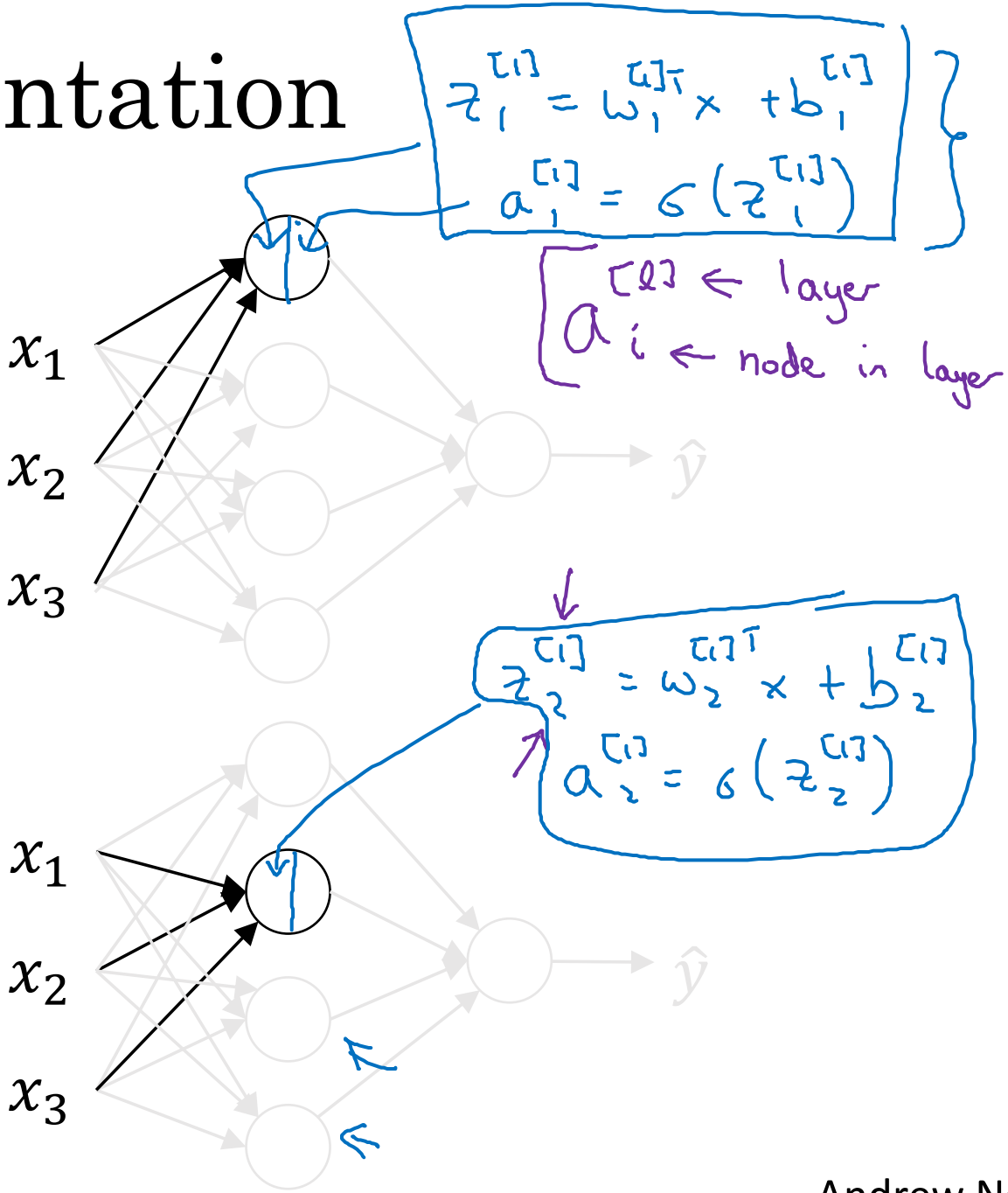


Neural Network Representation



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

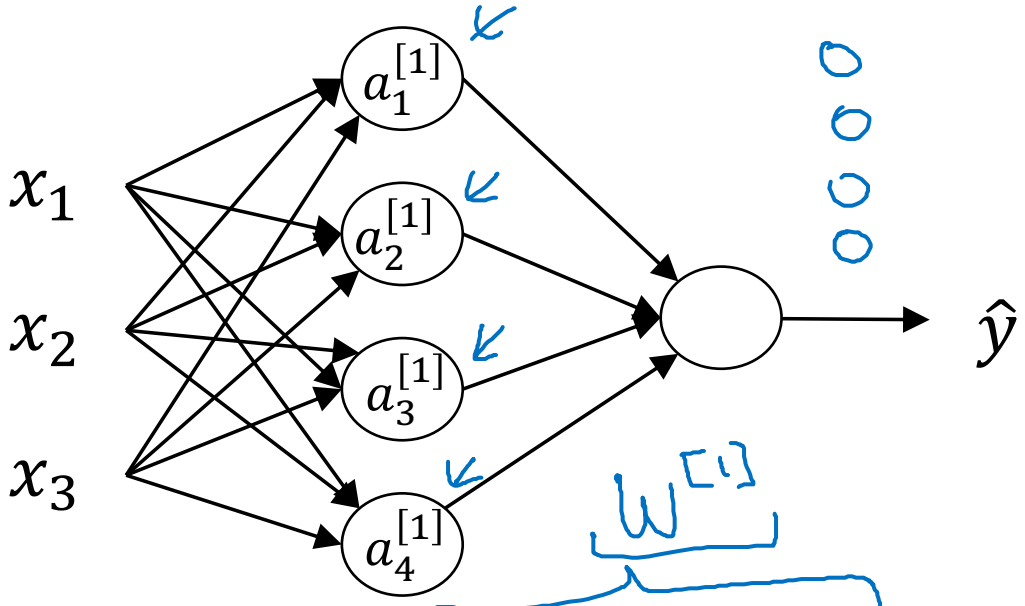
$$a = \sigma(z)$$



Andrew Ng

W is a matrix of transposed 'w' column vectors

Neural Network Representation



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

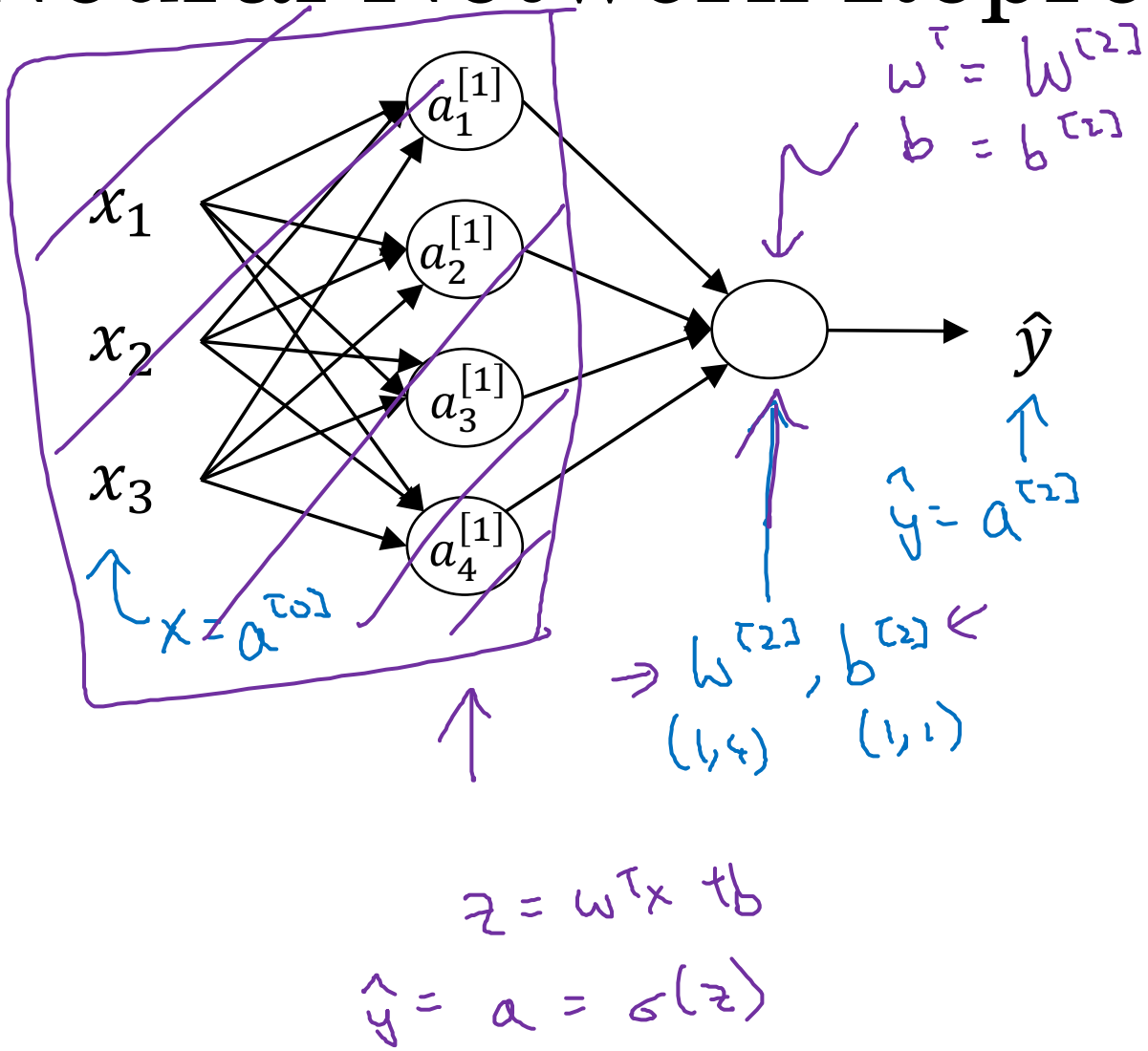
$$z^{[1]} = \begin{bmatrix} — w_1^{[1]T} — \\ — w_2^{[1]T} — \\ — w_3^{[1]T} — \\ — w_4^{[1]T} — \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

$W^{[1]}$ is $(4, 3)$; $b^{[1]}$ is $(4, 1)$

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

Andrew Ng

Neural Network Representation learning



Changed:
Transpose(W) to W

Given input x:

$$z^{[1]} = W^{[1]} a^{[0]} + b^{[1]}$$
(4,1) (4,3) (3,1) (4,1)

$$a^{[1]} = \sigma(z^{[1]})$$
(4,1) (4,1)

$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$
(1,1) (1,4) (4,1) (1,1)

$$a^{[2]} = \sigma(z^{[2]})$$
(1,1) (1,1)

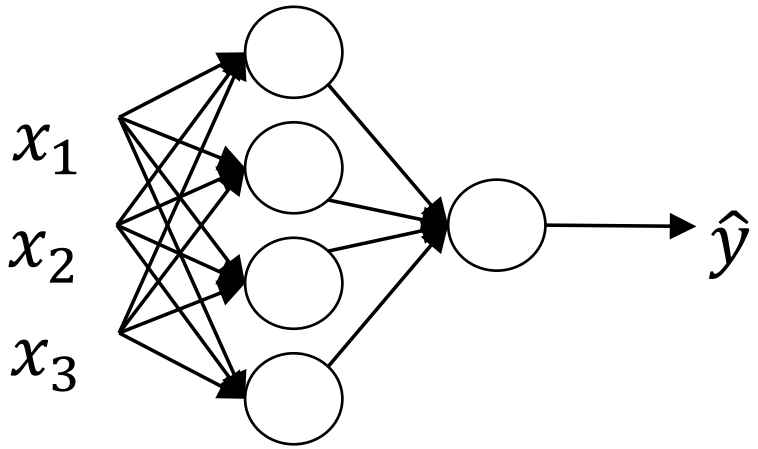
Andrew Ng

deeplearning.ai

One hidden layer Neural Network

Vectorizing across multiple examples

Vectorizing across multiple examples



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

$x \rightarrow a^{[2]} = \hat{y}$

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

$x^{(2)} \rightarrow a^{[2](2)} = \hat{y}^{(2)}$

$\vdots$

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

$a^{[2](i)}$ ← example i, layer 2

for $i = 1$ to m,

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

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

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

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

Vectorizing across multiple examples

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

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

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

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

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

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

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

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

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

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

(n_x, m)

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

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

training examples

hidden units

hidden units

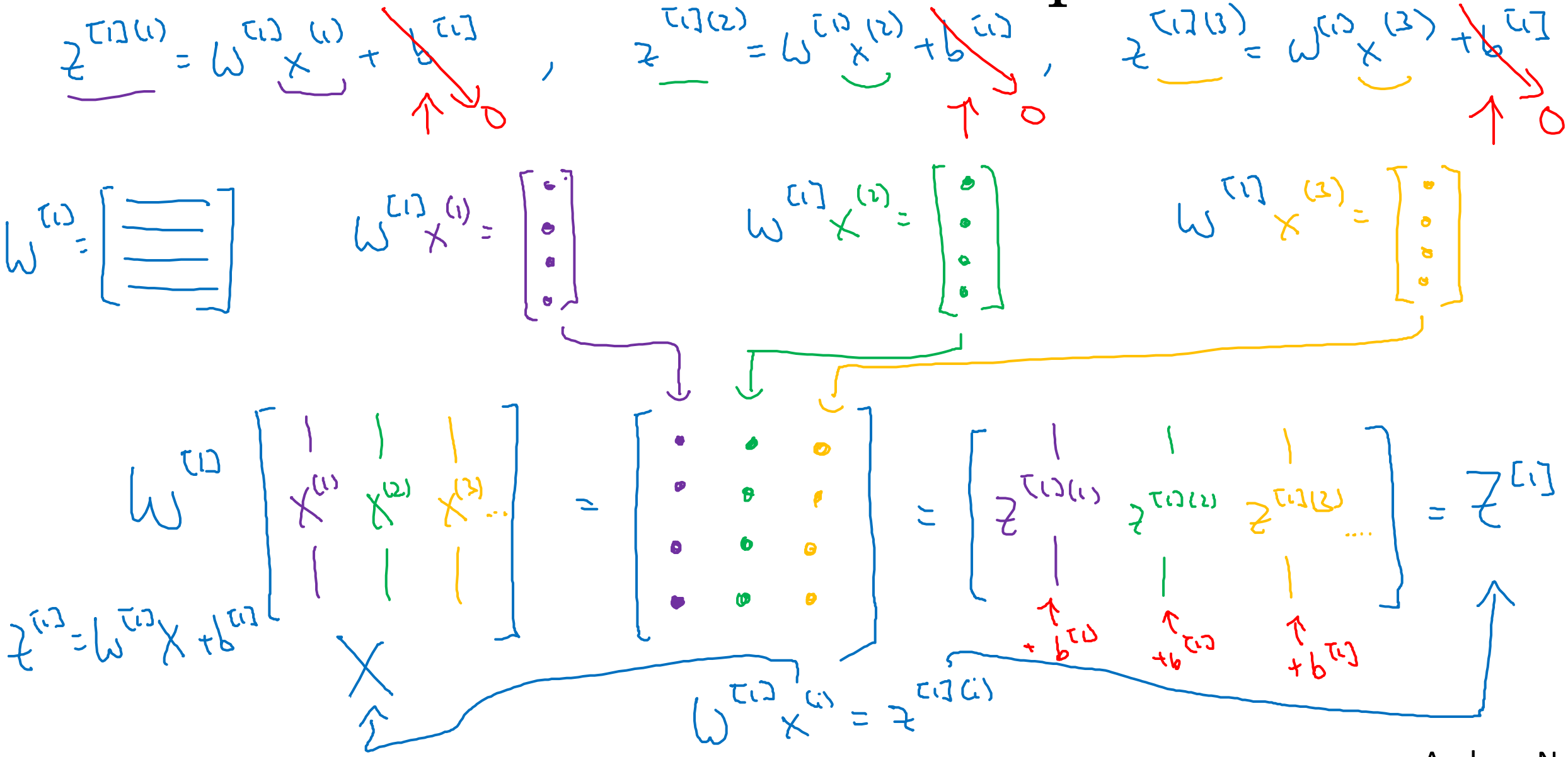
Andrew Ng

deeplearning.ai

One hidden layer Neural Network

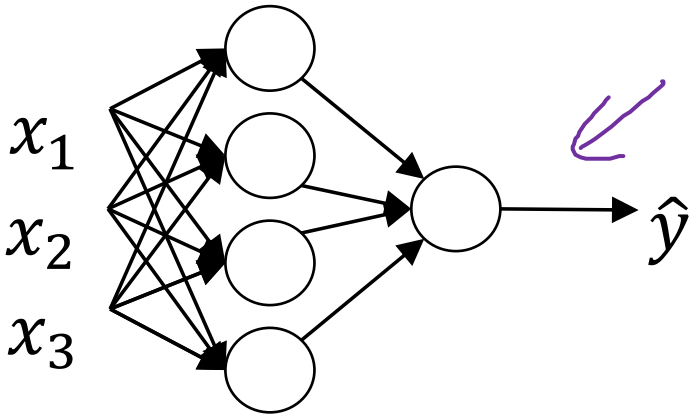
Explanation for vectorized implementation

Justification for vectorized implementation



Andrew Ng

Recap of vectorizing across multiple examples



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

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

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

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

$A^{[0]} = X = a^{[0]}$ $\quad x^{(i)} = a^{[0](i)}$

$$Z^{[1]} = W^{[1]}X + b^{[1]} \quad \leftarrow \quad W^{[1]}A^{[0]} + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

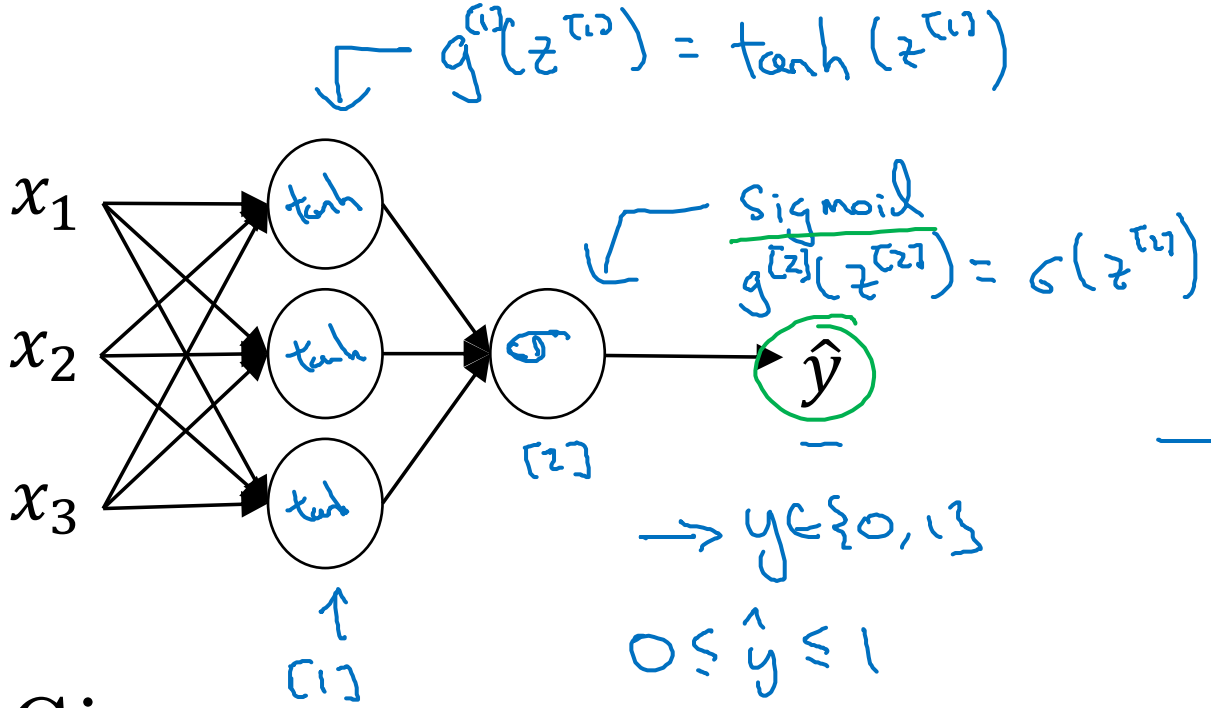
Andrew Ng

deeplearning.ai

One hidden layer Neural Network

Activation functions

Activation functions



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

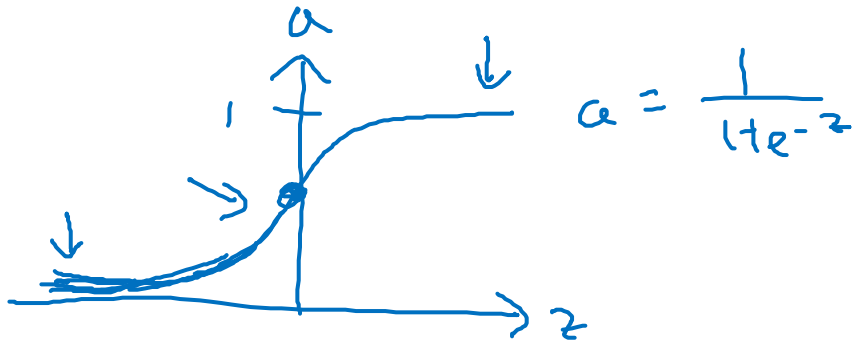
Sigmoid

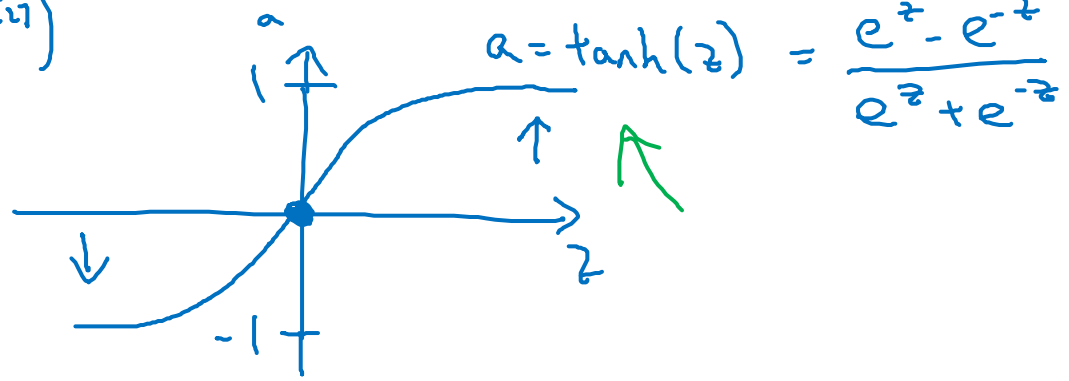
$g^{[2]}(z^{[2]}) = \sigma(z^{[2]})$

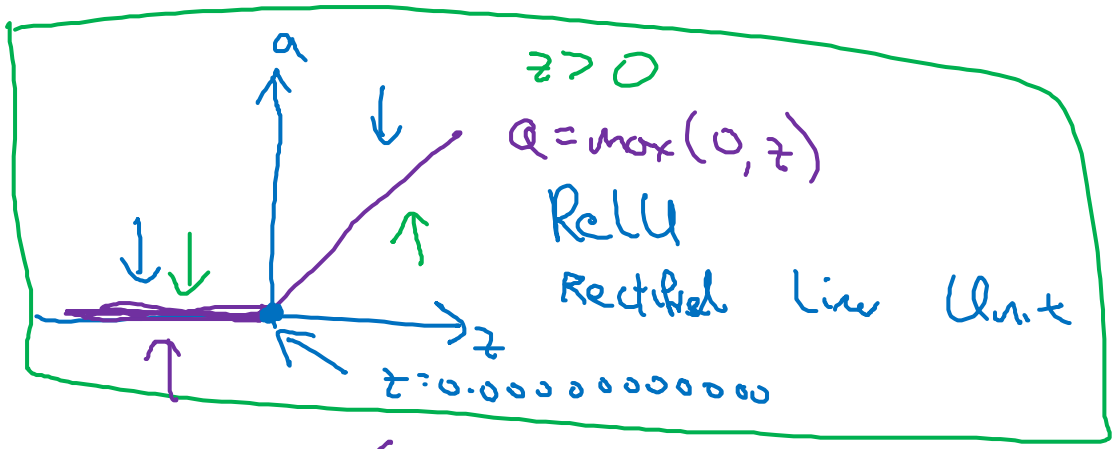
$\rightarrow y \in \{0, 1\}$

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

$-1 \quad 1$







Given x:

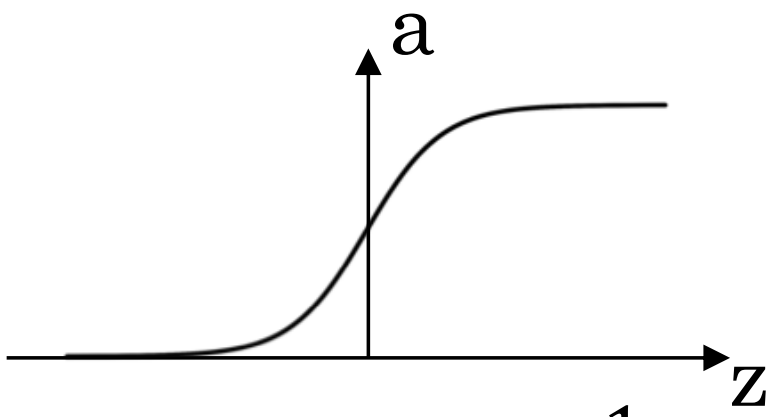
$$z^{[1]} = W^{[1]}x + b^{[1]}$$

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

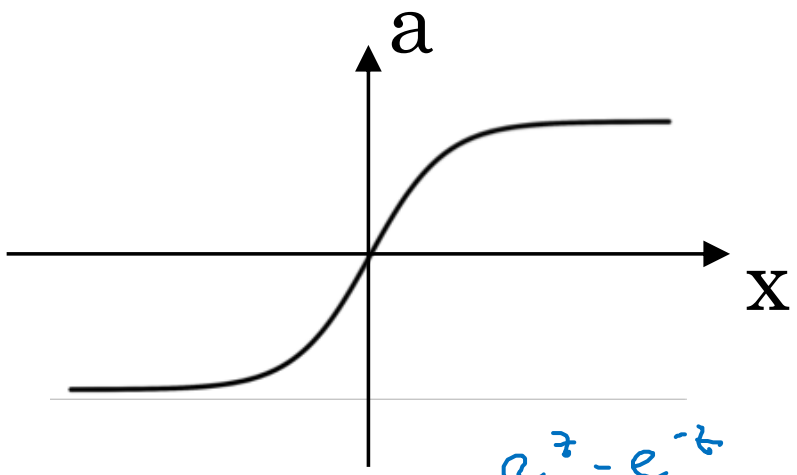
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

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

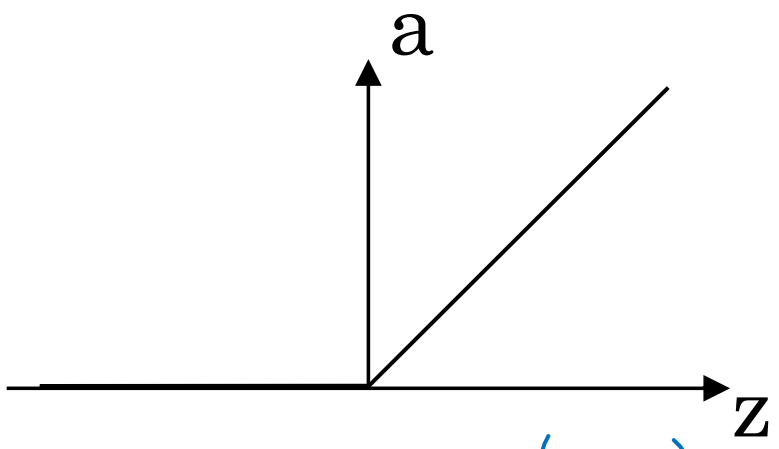
Pros and cons of activation functions



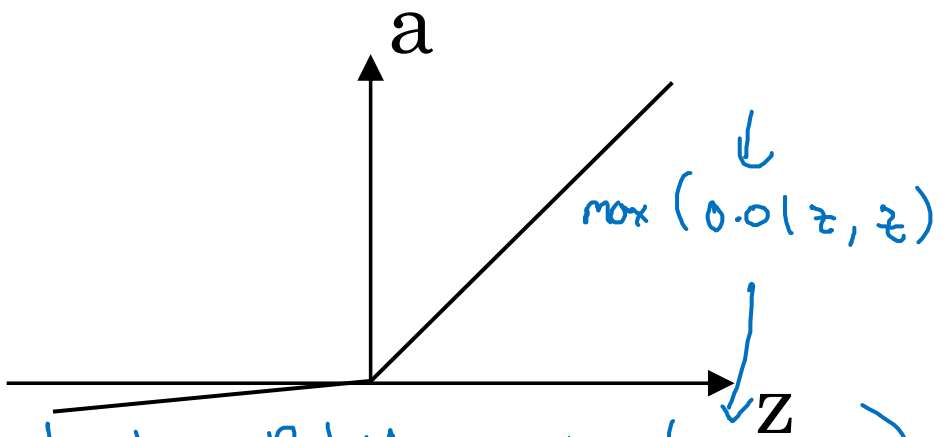
sigmoid: $a = \frac{1}{1+e^{-z}}$



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



ReLU $a = \max(0, z)$



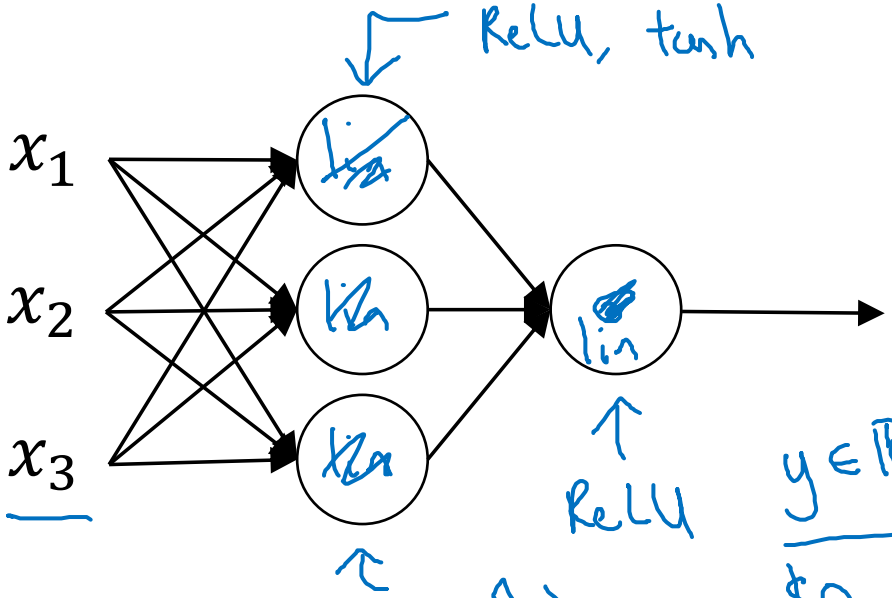
Leaky ReLU $a = \max(0.01z, z)$

deeplearning.ai

One hidden layer Neural Network

Why do you need non-linear activation functions?

Activation function



Given x:

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

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

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

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

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

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

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

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

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

$$= W'x + b'$$

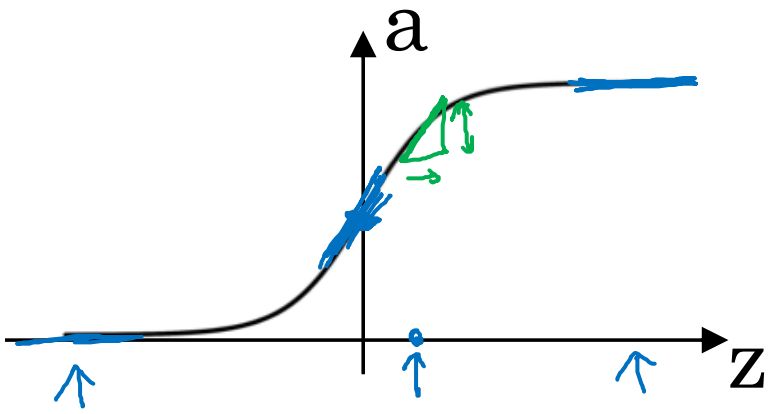
$g(z) = z$

deeplearning.ai

One hidden layer Neural Network

Derivatives of activation functions

Sigmoid activation function



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

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

$g'(z) = \frac{d}{dz} g(z)$ = slope of $g(x)$ at z

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

$= g(z)\left(1 - g(z)\right)$ ←

$= a(1-a)$

$g'(z) = a(1-a)$ ↑

$z = 10$. $g(z) \approx 1$

$\frac{d}{dz} g(z) \approx 1\,(1-1) \approx 0$

$z = -10$ $g(z) \approx 0$

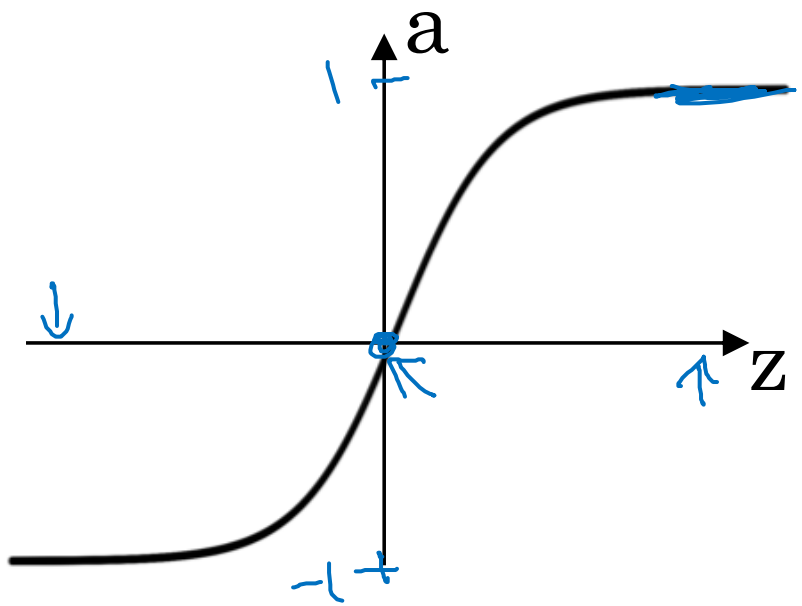
$\frac{d}{dz} g(z) \approx 0 \cdot (1-0) \approx 0$

$z = 0$ $g(z) = \frac{1}{2}$

$\frac{d}{dz} g(z) = \frac{1}{2}\left(1 - \frac{1}{2}\right) = \frac{1}{4}$

Andrew Ng

Tanh activation function



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

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

$$g'(z) = \frac{d}{dz} g(z) = \text{slope of } g(z) \text{ at } z$$

$$= 1 - (\tanh(z))^2$$

$z = 10 \quad \tanh(z) \approx 1$

$g'(z) \approx 0$

$z = -10 \quad \tanh(z) \approx -1$

$g'(z) \approx 0$

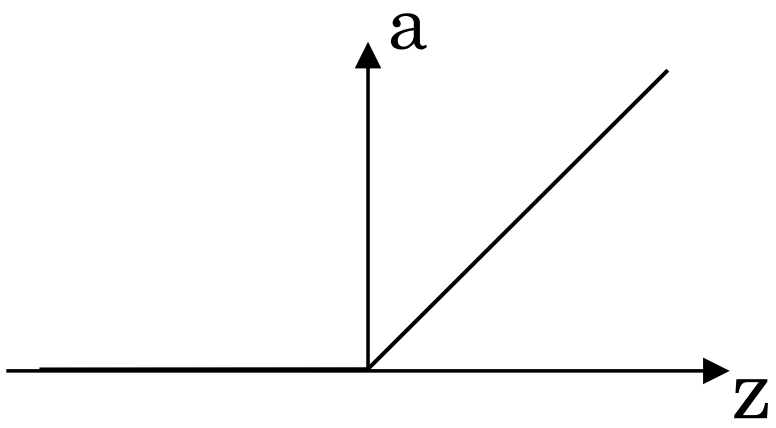
$z = 0 \quad \tanh(z) = 0$

$g'(z) = 1$

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

Andrew Ng

ReLU and Leaky ReLU



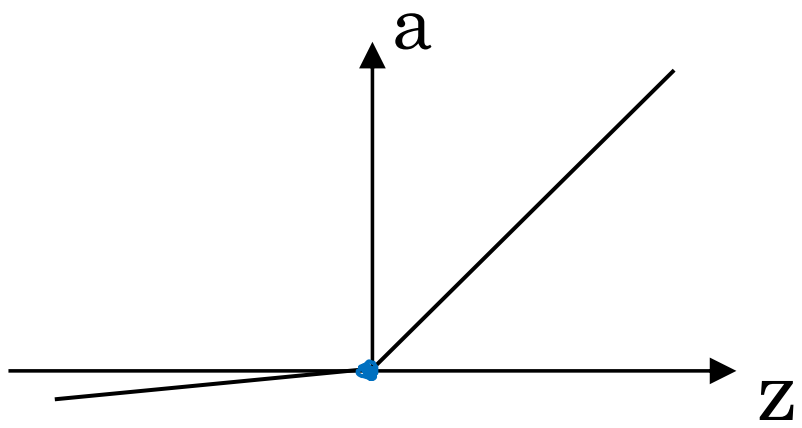
ReLU

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

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

~~undef if $z = 0$~~

$z = 0.000\ldots0$



Leaky ReLU

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

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

Andrew Ng

deeplearning.ai

One hidden layer Neural Network

Gradient descent for neural networks

Gradient descent for neural networks

Parameters: $w^{[1]}$, $b^{[1]}$, $w^{[2]}$, $b^{[2]}$

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

$n_x = n^{[0]}$, $n^{[1]}$, $n^{[2]} = 1$

Cost function: $J(w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}, y)$

($w^{[2]}$, $b^{[2]}$; $\hat{y} = a^{[2]}$)

Gradient descent:

→ Repeat {

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

$dw^{[1]} = \frac{\partial J}{\partial w^{[1]}}$, $db^{[1]} = \frac{\partial J}{\partial b^{[1]}}$, ...

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

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

$w^{[2]} := \ldots$ $b^{[2]} := \ldots$

}

Andrew Ng

Formulas for computing derivatives

A-Y: derivation shown in Logistic case

Forward propagation:

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

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

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

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

Back propagation:

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

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

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

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

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

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

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

$(n^{[1]}, m)$ $(n^{[1]}, m)$ element-wise product

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

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

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

Andrew Ng

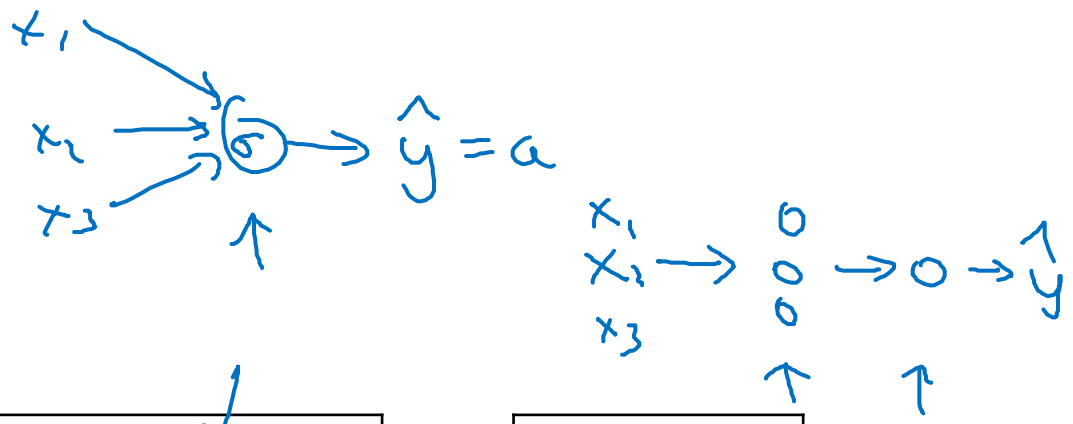
deeplearning.ai

One hidden layer Neural Network

Backpropagation intuition (Optional)

Computing gradients

Logistic regression



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

$dz = a - y$

$dw = dz \cdot x$

$db = dz$

$da = \frac{d}{da}\mathcal{L}(a,y) = -y\log a - (1-y)\log(1-a)$

$= -\frac{y}{a} + \frac{1-y}{1-a}$

$dz = da \cdot g'(z)$

$g(z) = \sigma(z)$

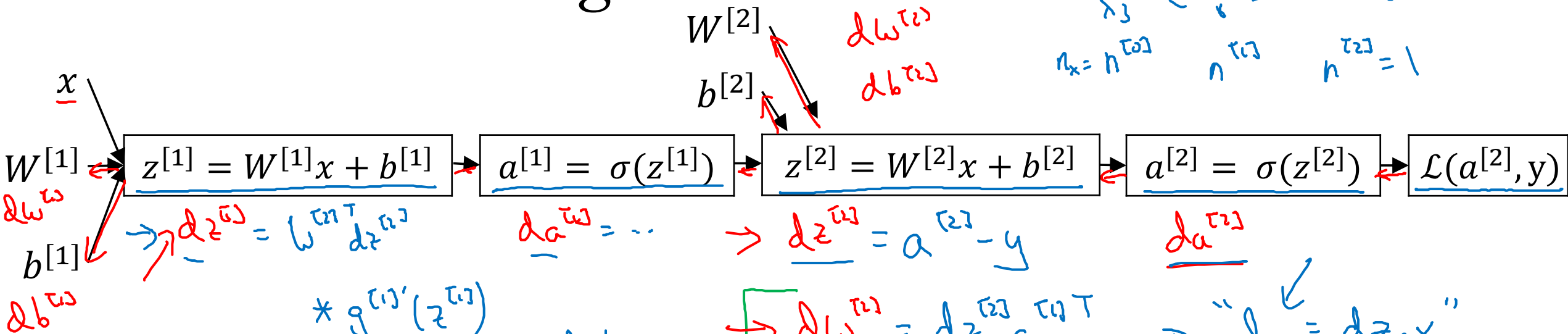
$\frac{d\mathcal{L}}{dz} = \frac{d\mathcal{L}}{da} \cdot \frac{da}{dz}$

"$dz = da$"

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

for single example

Neural network gradients



$dW^{[2]}$ $db^{[2]}$ $dW^{[1]}$ $db^{[1]}$

$dz^{[1]} = W^{[2]T} dz^{[2]}$ $\quad da^{[1]} = \ldots$ $\quad dz^{[2]} = a^{[2]} - y$ $\quad da^{[2]}$

$* g^{[1]'}(z^{[1]})$ ← element wise product

$dW^{[2]} = dz^{[2]} a^{[1]T}$ $\quad db^{[2]} = dz^{[2]}$ $\quad \rightarrow$ "$dw = dz \cdot x$"

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

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

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

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

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

foo

dfoo

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

$dW^{[2]}$ $db^{[1]} = dz^{[1]}$ $\quad a^{[0]T}$

var & d(var) have same dimension

Andrew Ng

Summary of gradient descent

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

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

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

$$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$$

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

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

Vectorized Implementation:

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

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

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

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

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

Andrew Ng

Summary of gradient descent

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

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

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

$$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$$

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

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

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

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

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

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

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

elementwise product

$$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$$

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

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

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

Andrew Ng

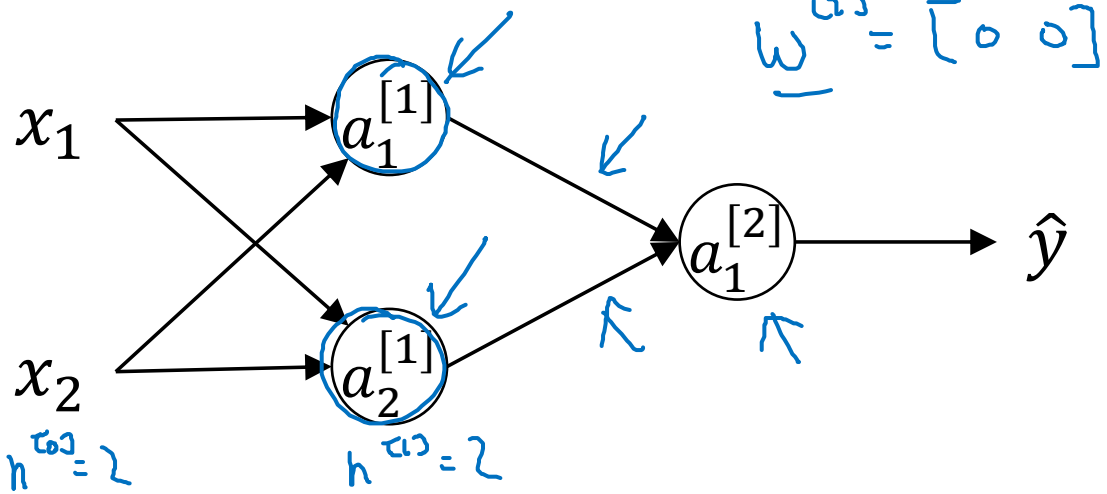
deeplearning.ai

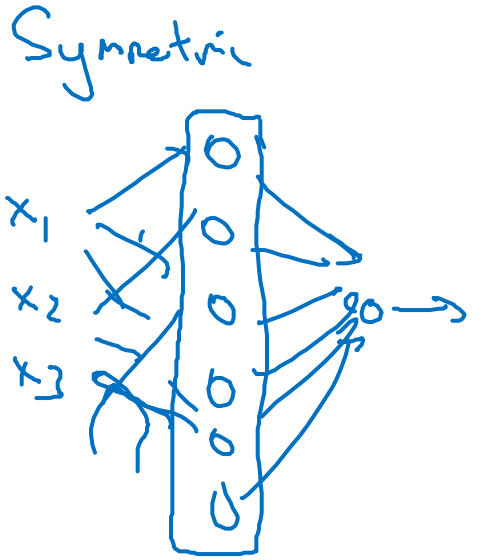
One hidden layer Neural Network

Random Initialization

What happens if you initialize weights to zero?

Every node in a layer becomes symmetric and changes together at each iteration.





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

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

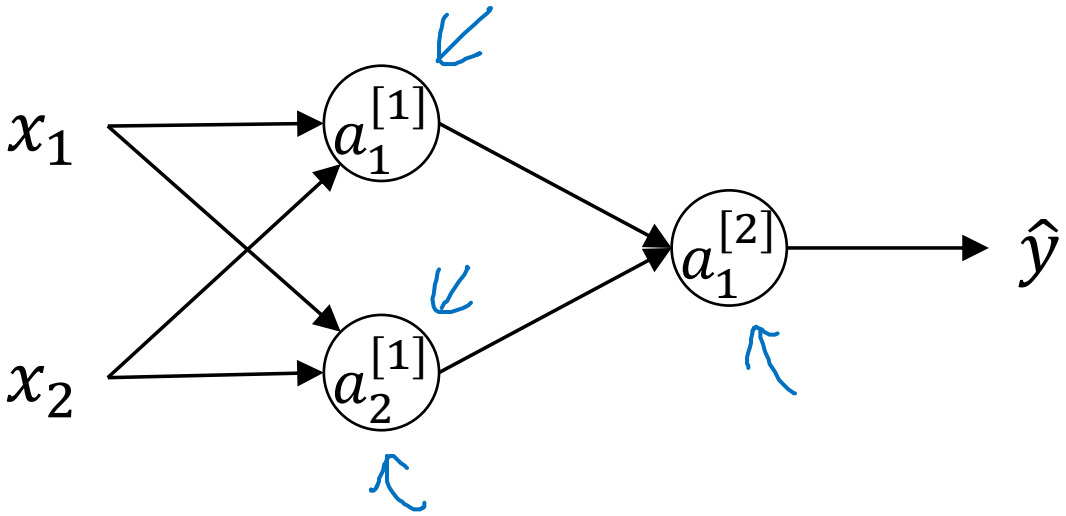
$$dw = \begin{bmatrix} u & v \\ u & v \end{bmatrix}$$

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

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

Andrew Ng

Random initialization



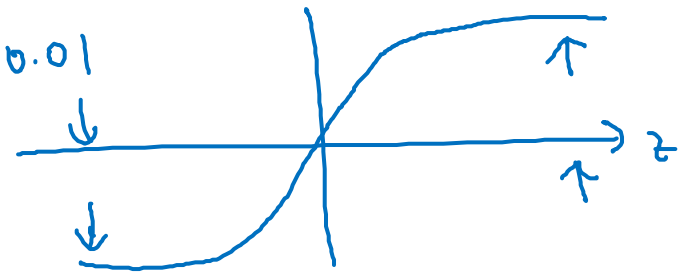
$W^{[1]} = np.random.randn((2,2)) * 0.01$

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

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

$b^{[2]} = 0$

0.01 / 100 ?



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

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

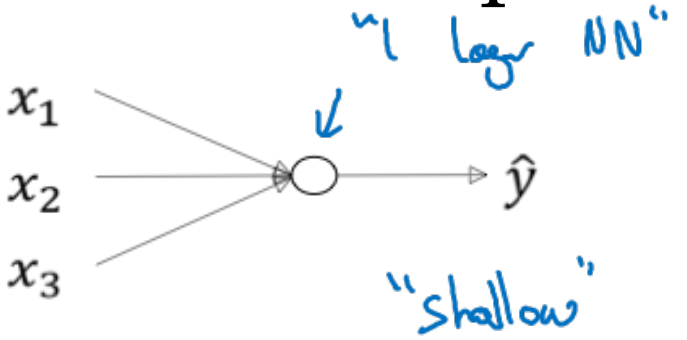
Copyright Notice

deeplearning.ai

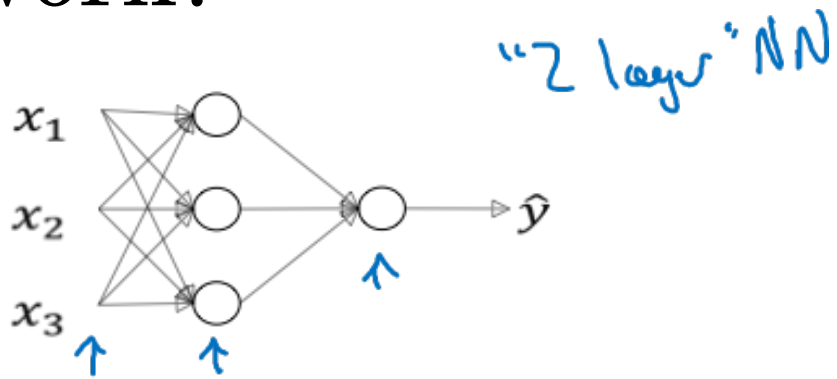
Deep Neural Networks

Deep L-layer Neural network

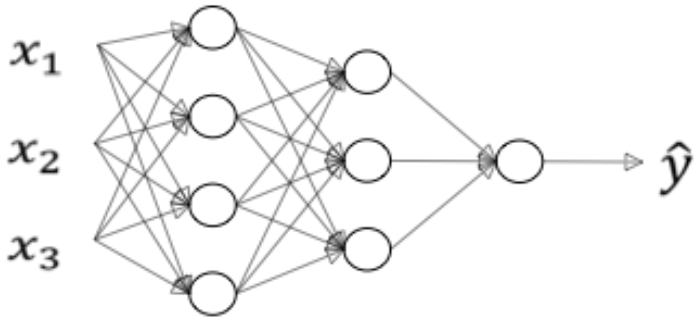
What is a deep neural network?



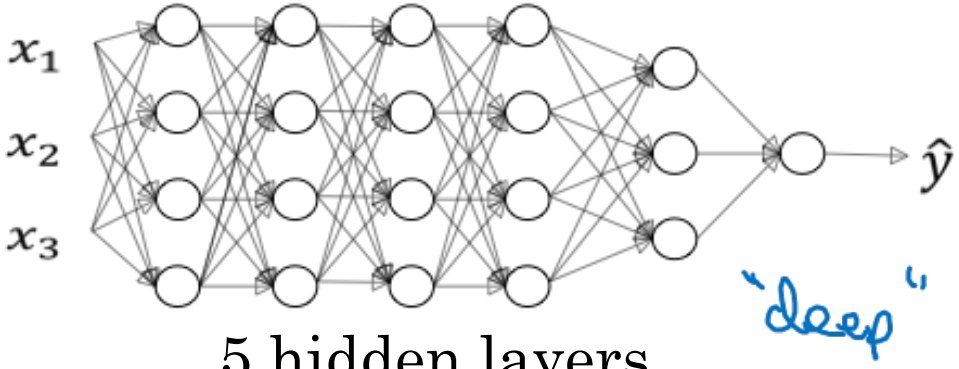
logistic regression



1 hidden layer



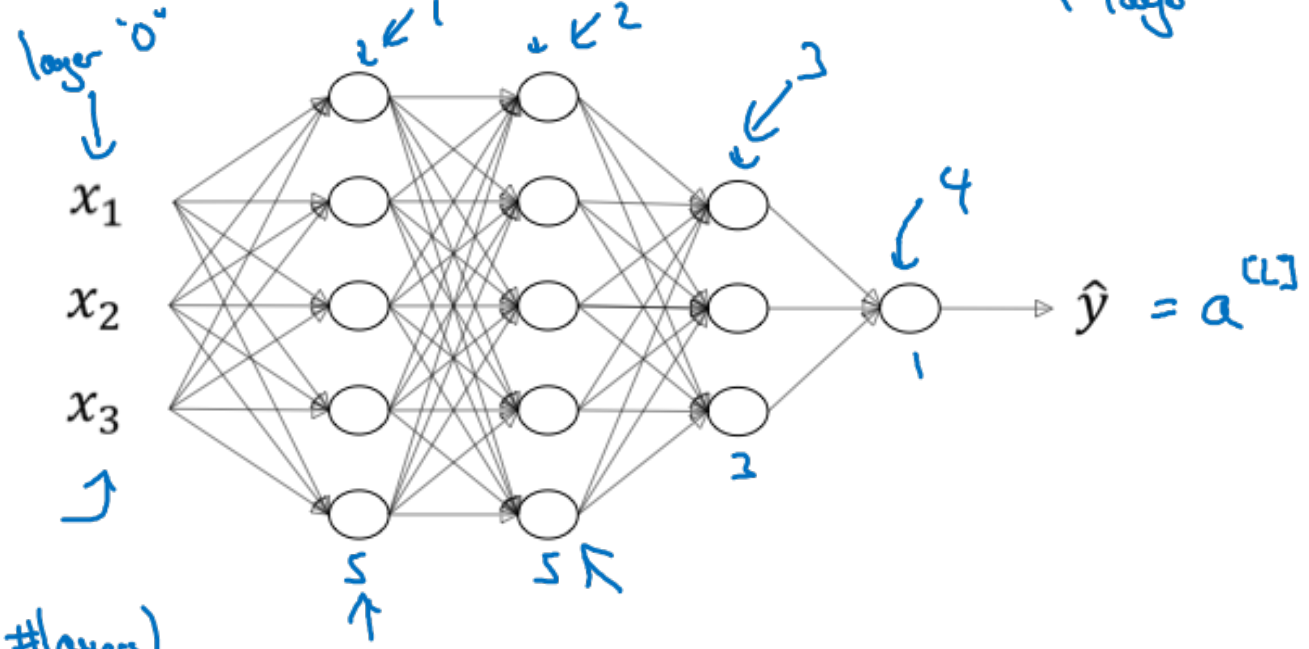
2 hidden layers



5 hidden layers

Andrew

Deep neural network notation



$L = 4$ (#layers)

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

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

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

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

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

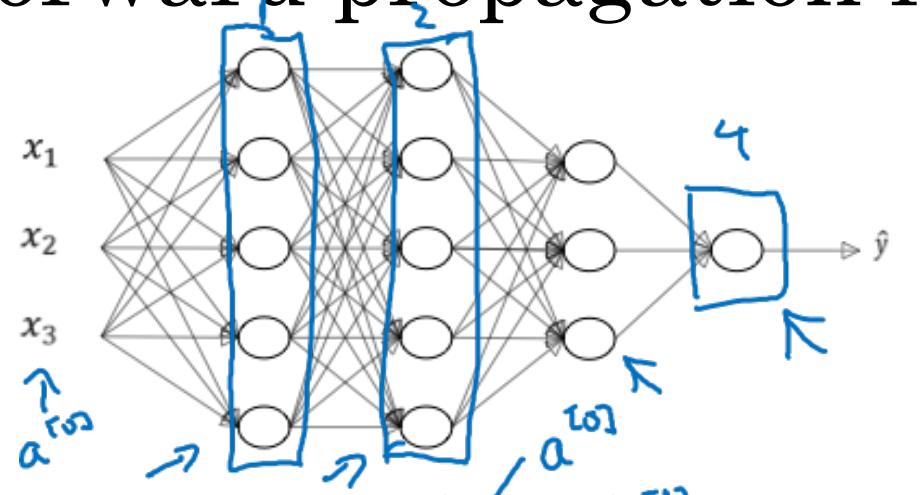
Andrew

deeplearning.ai

Deep Neural Networks

Forward Propagation in a Deep Network

Forward propagation in a deep network



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

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

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

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

$$z^{[4]} = W^{[4]} a^{[3]} + b^{[4]}, \quad a^{[4]} = g^{[4]}(z^{[4]}) = \hat{y}$$

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

$A^{[0]} = X$

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

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

Vectorized:

$$Z^{[1]} = W^{[1]} A^{[0]} + b^{[1]}$$

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

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

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

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

$X = A^{[0]}$

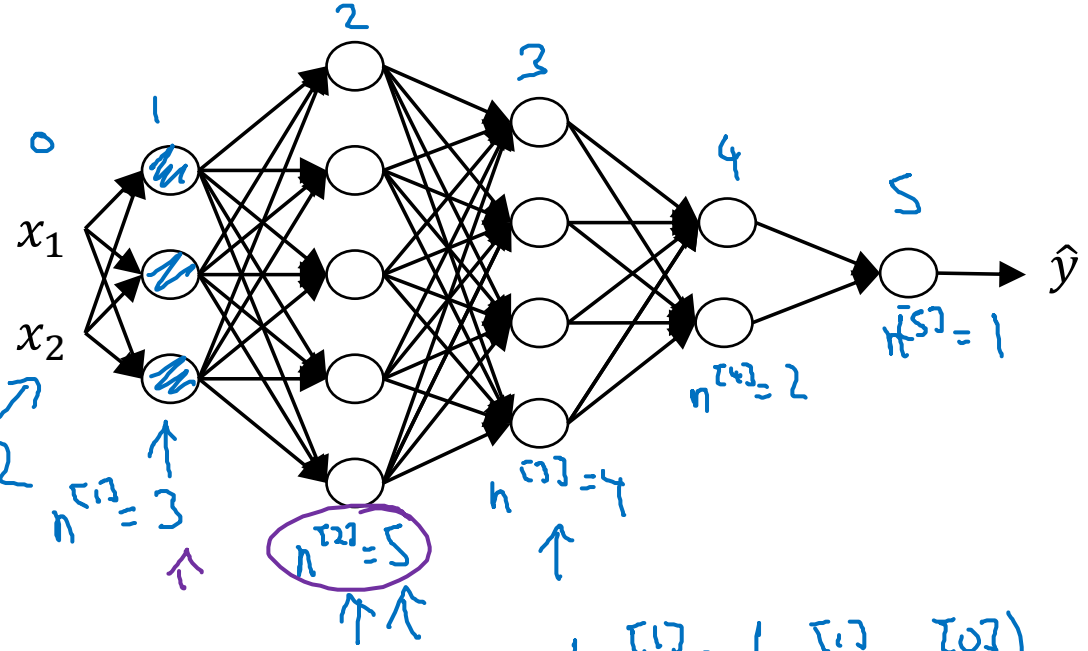
for $l = 1 \ldots 4$

deeplearning.ai

Deep Neural Networks

Getting your matrix dimensions right

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



$L = 5$

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

$a^{[l]}$

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

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

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

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

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

$(3,1) \leftarrow (3,2) \quad (2,1) \quad (3,1)$

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

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

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

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

$\rightarrow (5,1) \quad (5,3) \quad (3,1) \quad (5,1)$

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

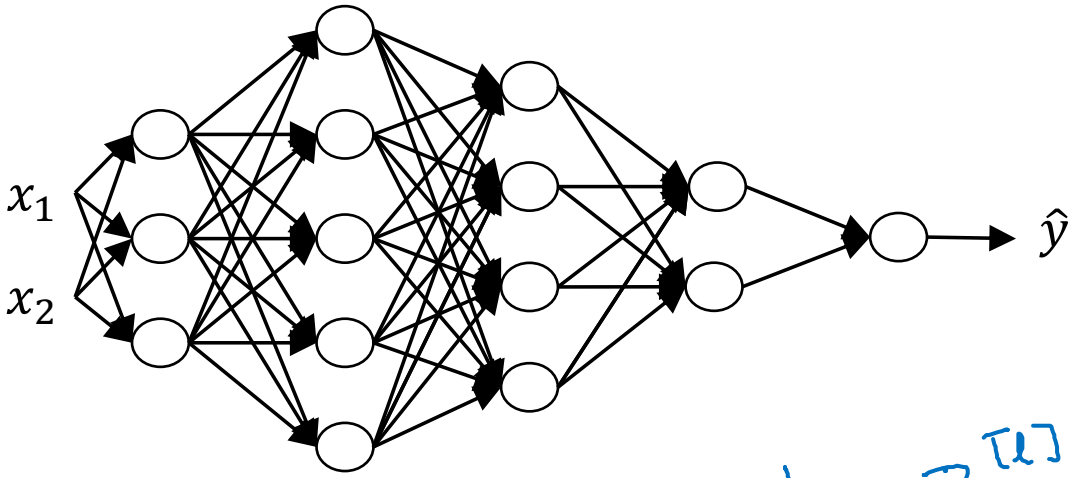
$W^{[3]} : (4, 5)$

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

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

Andrew Ng

Vectorized implementation



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

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

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

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

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

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

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

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

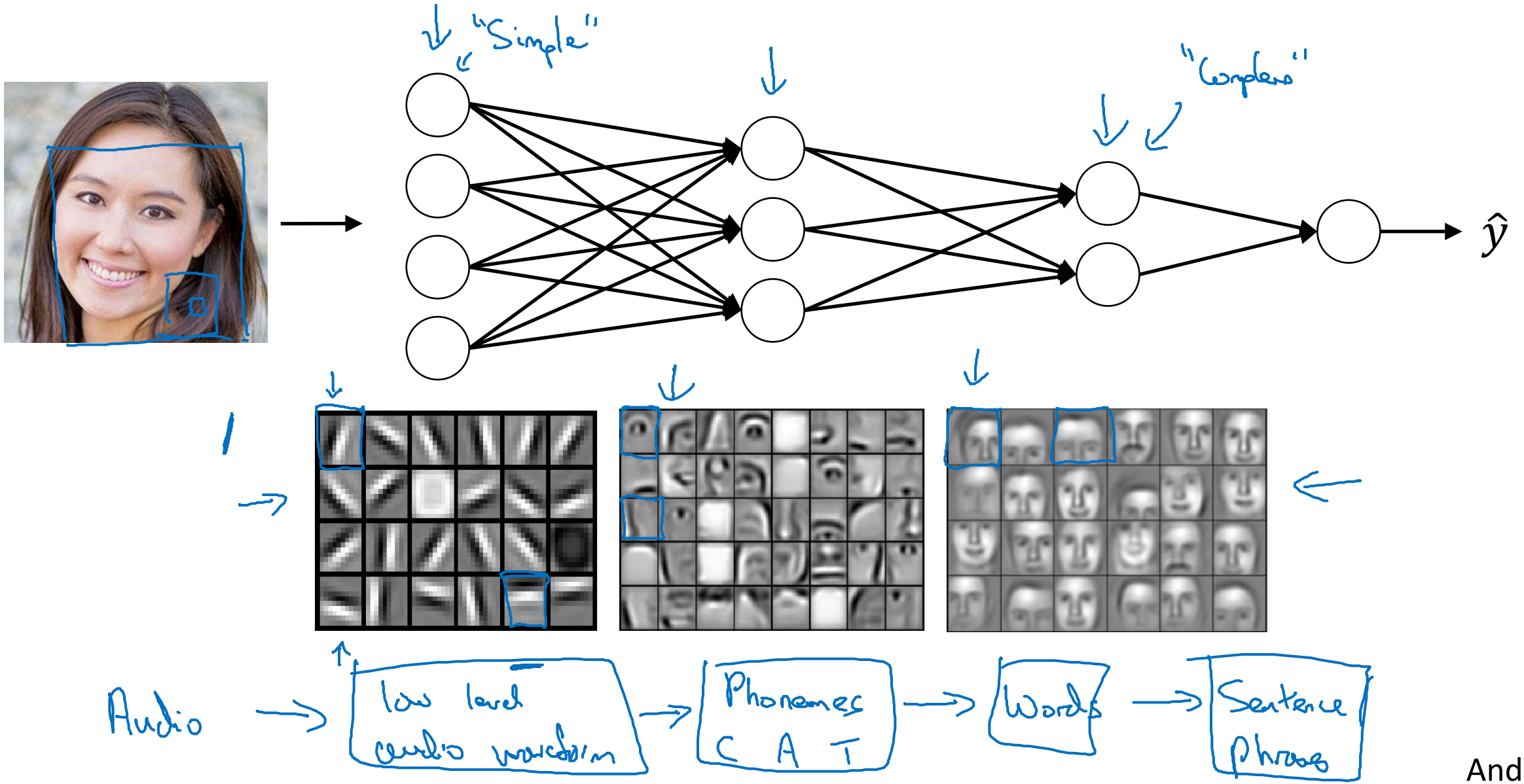
$dZ^{[l]}, dA^{[l]} : (n^{[l]}, m)$

deeplearning.ai

Deep Neural Networks

Why deep representations?

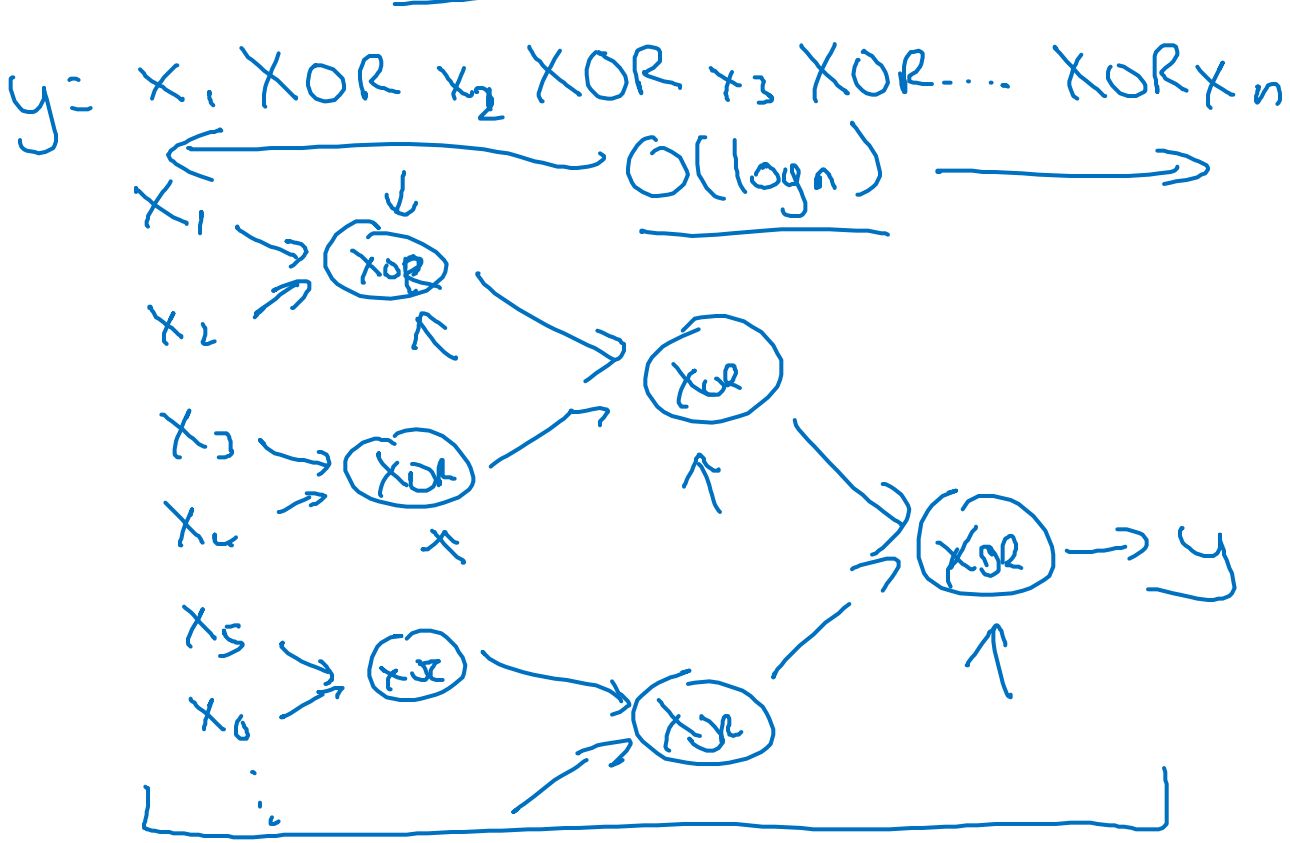
Intuition about deep representation

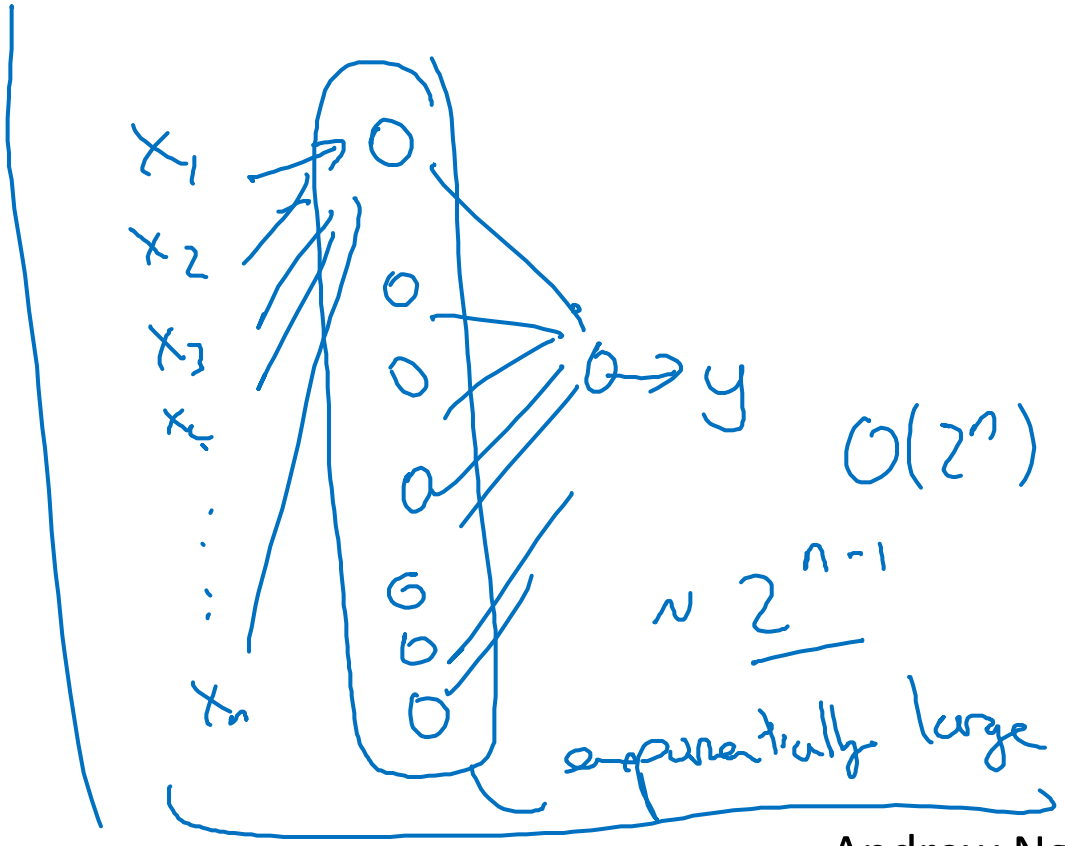


Andrew Ng

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.



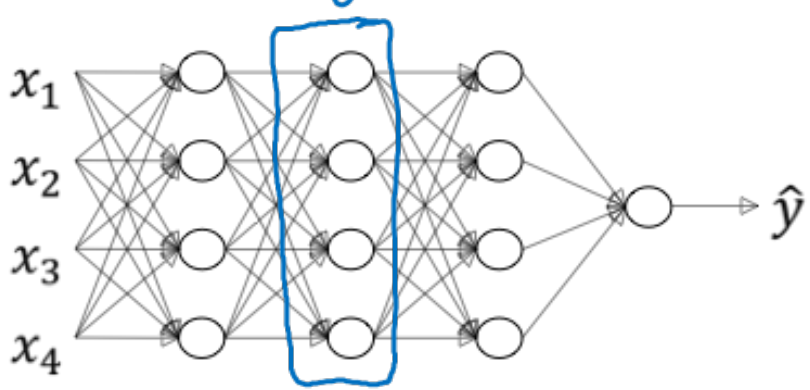


deeplearning.ai

Deep Neural Networks

Building blocks of deep neural networks

Forward and backward functions



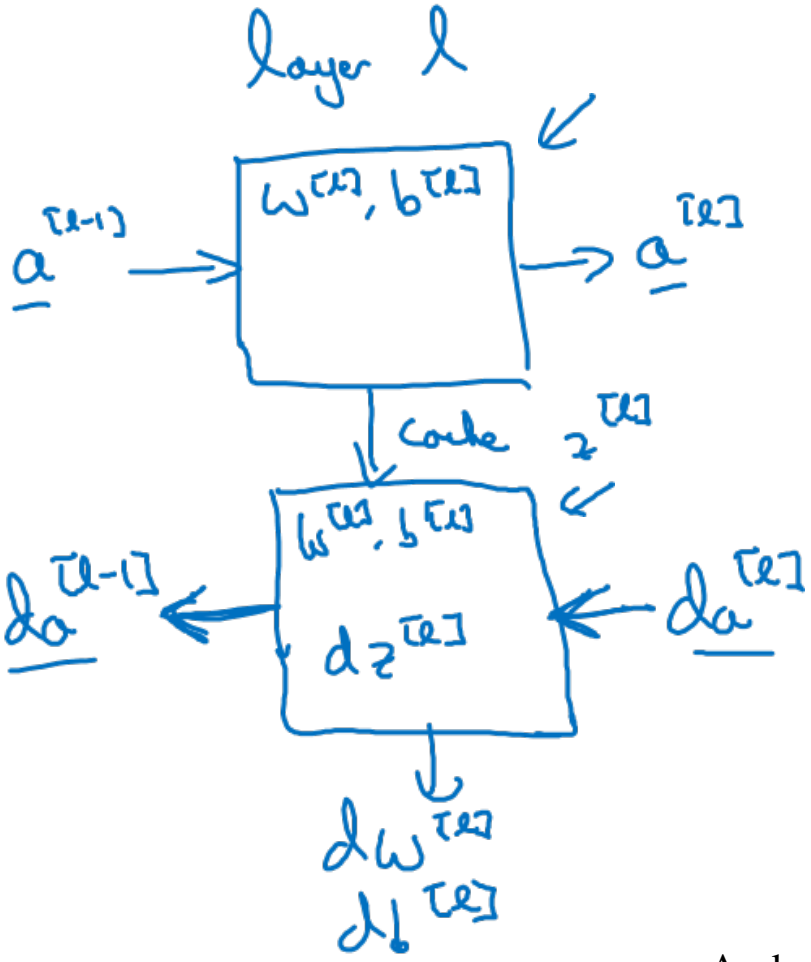
Layer l: $W^{[l]}, b^{[l]}$

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

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

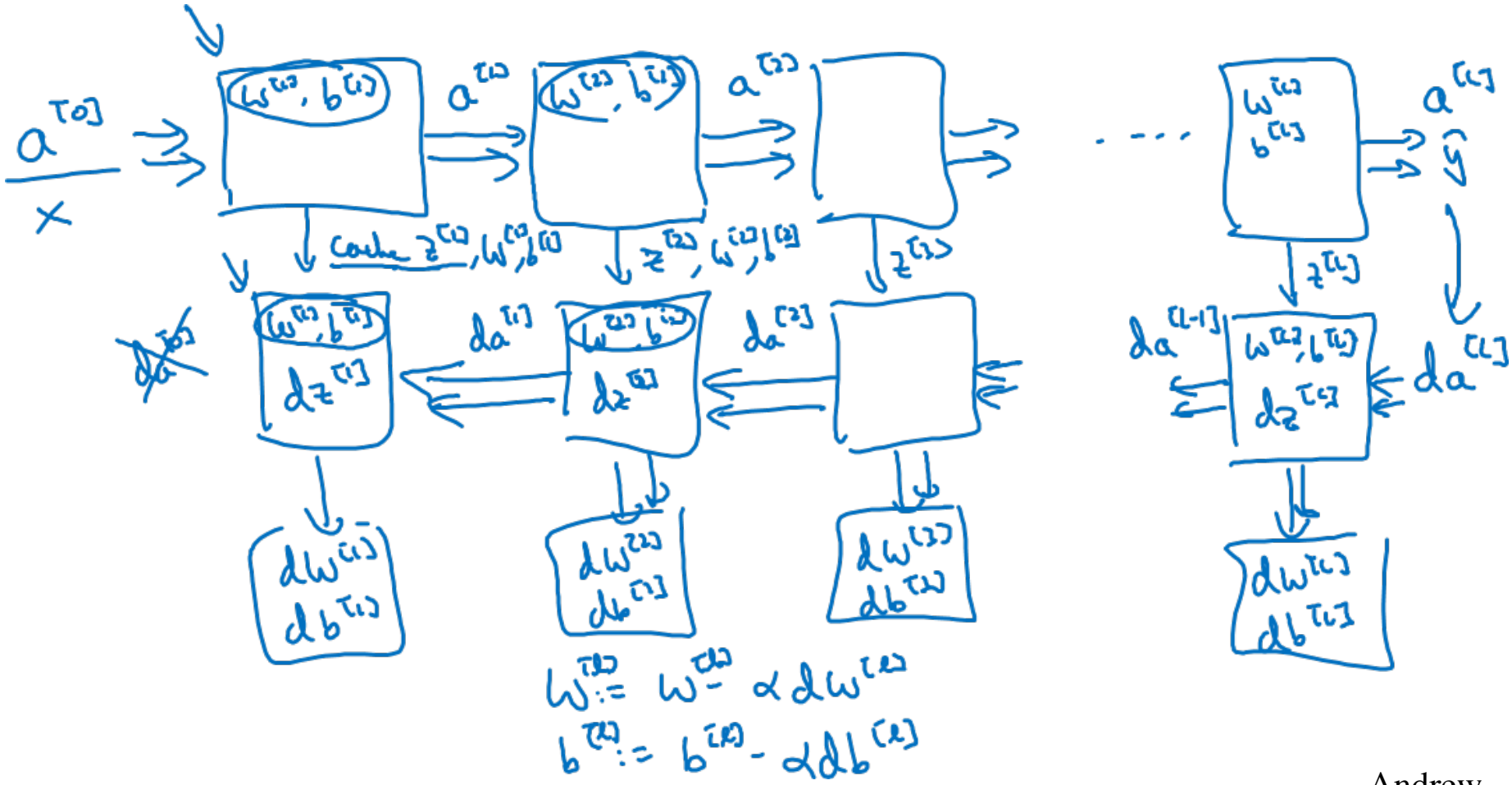
$a^{[l]} = g^{[l]}(z^{[l]})$

→ Backward: Input $da^{[l]}$, cache($z^{[l]}$), output $da^{[l-1]}$, $dW^{[l]}$, $db^{[l]}$



Andrew

Forward and backward functions



Andrew

deeplearning.ai

Deep Neural Networks

Forward and backward propagation

Slide on forward propagation: going from A[L-1] input to A[L] output (along with Z[L] and W and B, is missing

Backward propagation for layer l

→ Input $da^{[l]}$

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

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

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

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

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

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

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

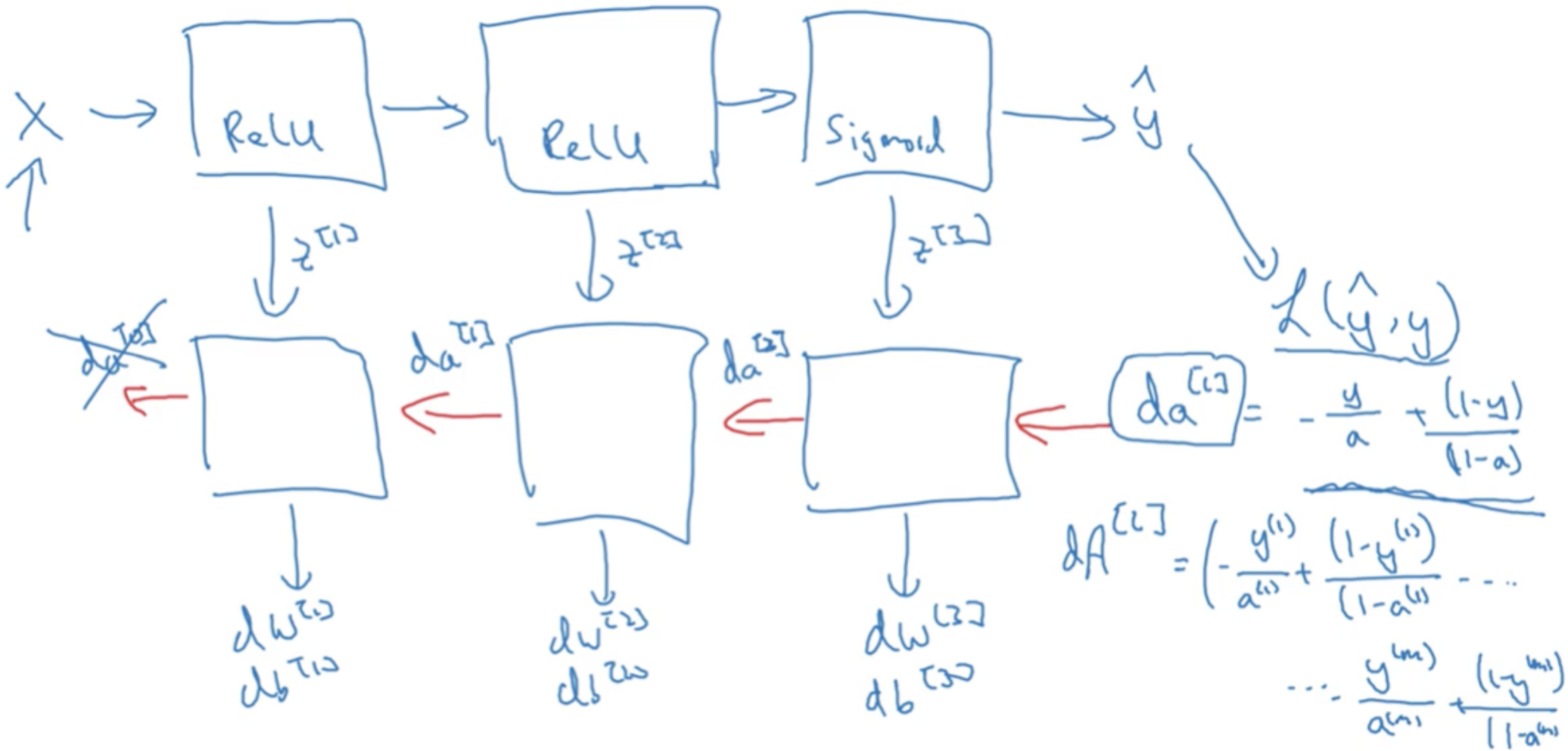
$$dW^{[l]} = \frac{1}{m} dz^{[l]} \cdot A^{[l-1]T}$$

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

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

Red arrows show dA(s) - however, in shallow network, instead of dA(s) we worked directly with dZ(s), they have a 1x1 relationship based on activation function

Summary



deeplearning.ai

Deep Neural Networks

Parameters vs Hyperparameters

What are hyperparameters?

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

Hyperparameters: learning rate α

#iterations

#hidden layers L

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

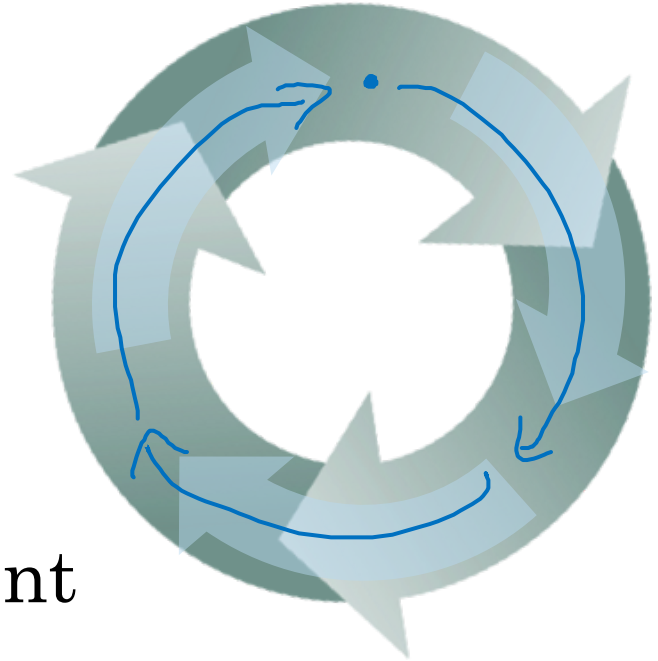
choice of activation function

Later: Momentum, mini-batch size, regularizations, ...

Applied deep learning is a very empirical process

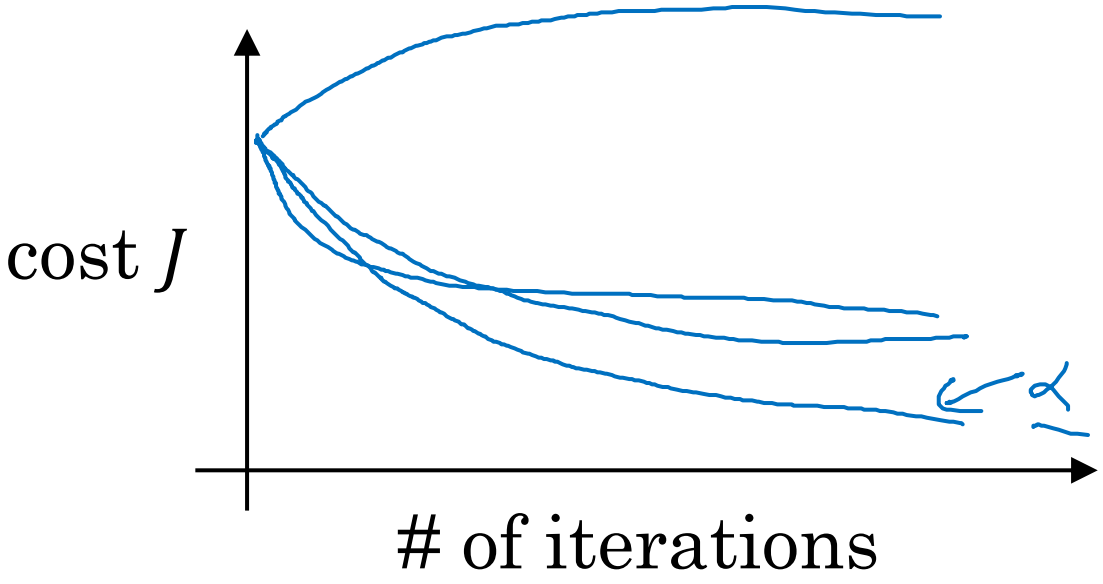
Idea

$\alpha = 0.04$
0.05



Experiment

Code



Vision, Speech, NLP, Ad, Search, Recommendation.

deeplearning.ai

Deep Neural Networks

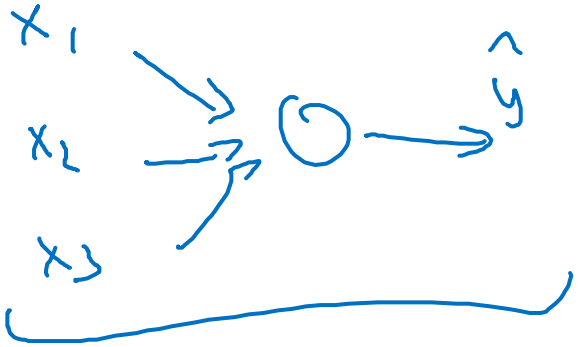
What does this have to do with the brain?

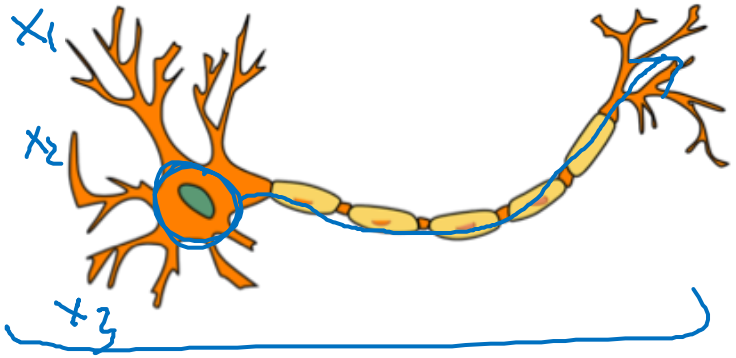
Forward and backward propagation

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

"It's like the brain."

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





Andrew Ng

Standard notations for Deep Learning

This document has the purpose of discussing a new standard for deep learning mathematical notations.

1 Neural Networks Notations.

General comments:

· superscript (i) will denote the i^{th} training example while superscript [l] will denote the l^{th} layer

Sizes:

·m : number of examples in the dataset

·n_x : input size

·n_y : output size (or number of classes)

·$n_h^{[l]}$: number of hidden units of the l^{th} layer

In a for loop, it is possible to denote $n_x = n_h^{[0]}$ and $n_y = n_h{}^{[\text{number of layers } +1]}$.

·L : number of layers in the network.

Objects:

·$X \in \mathbb{R}^{n_x \times m}$ is the input matrix

·$x^{(i)} \in \mathbb{R}^{n_x}$ is the i^{th}example represented as a column vector

·$Y \in \mathbb{R}^{n_y \times m}$ is the label matrix

·$y^{(i)} \in \mathbb{R}^{n_y}$ is the output label for the i^{th} example

·$W^{[l]} \in \mathbb{R}^{\text{number of units in next layer} \times \text{number of units in the previous layer}}$ is the weight matrix,superscript [l] indicates the layer

·$b^{[l]} \in \mathbb{R}^{\text{number of units in next layer}}$ is the bias vector in the l^{th} layer

·$\hat{y} \in \mathbb{R}^{n_y}$ is the predicted output vector. It can also be denoted $a^{[L]}$ where L is the number of layers in the network.

Common forward propagation equation examples:

$a = g^{[l]}(W_x x^{(i)} + b_1) = g^{[l]}(z_1)$ where $g^{[l]}$ denotes the l^{th} layer activation function

$\hat{y}^{(i)} = softmax(W_h h + b_2)$

· General Activation Formula: $a_j^{[l]} = g^{[l]}(\sum_k w_{jk}^{[l]} a_k^{[l-1]} + b_j^{[l]}) = g^{[l]}(z_j^{[l]})$

· $J(x, W, b, y)$ or $J(\hat{y}, y)$ denote the cost function.

Examples of cost function:

· $J_{CE}(\hat{y}, y) = -\sum_{i=0}^{m} y^{(i)} \log \hat{y}^{(i)}$

· $J_1(\hat{y}, y) = \sum_{i=0}^{m} \mid y^{(i)} - \hat{y}^{(i)} \mid$

2 Deep Learning representations

For representations:

· nodes represent inputs, activations or outputs

· edges represent weights or biases

Here are several examples of Standard deep learning representations

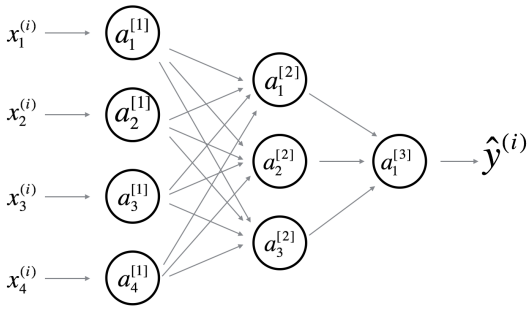


Figure 1: Comprehensive Network: representation commonly used for Neural Networks. For better aesthetic, we omitted the details on the parameters ($w_{ij}^{[l]}$ and $b_i^{[l]}$ etc...) that should appear on the edges

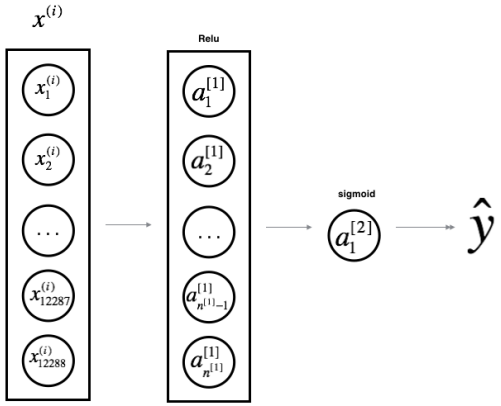


Figure 2: Simplified Network: a simpler representation of a two layer neural network, both are equivalent.

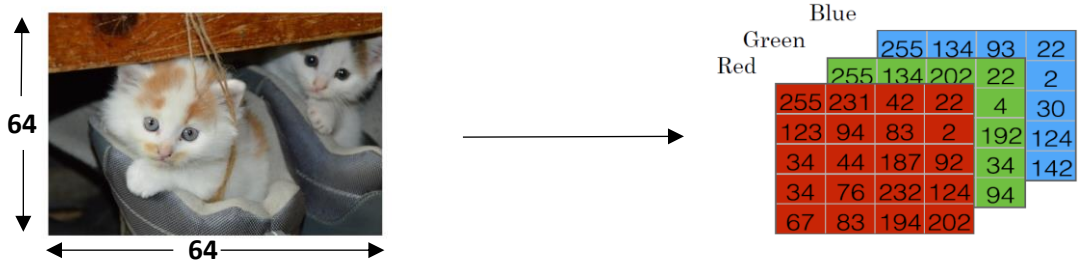
Binary Classification

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

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

Example: Cat vs Non-Cat

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



An image is store in the computer in three separate matrices corresponding to the Red, Green, and Blue color channels of the image. The three matrices have the same size as the image, for example, the resolution of the cat image is 64 pixels X 64 pixels, the three matrices (RGB) are 64 X 64 each.

The value in a cell represents the pixel intensity which will be used to create a feature vector of n-dimension. In pattern recognition and machine learning, a feature vector represents an object, in this case, a cat or no cat.

To create a feature vector, x, the pixel intensity values will be "unroll" or "reshape" for each color. The dimension of the input feature vector x is $n_x = 64\ x\ 64\ x\ 3 = 12\ 288$.

$$x = \begin{bmatrix} 255 \\ 231 \\ 42 \\ \vdots \\ 255 \\ 134 \\ 202 \\ \vdots \\ 255 \\ 134 \\ 93 \\ \vdots \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \\ \\ \text{red} \\ \\ \\ \\ \text{green} \\ \\ \\ \\ \text{blue} \\ \\ \end{matrix}$$

Logistic Regression

Logistic regression is a learning algorithm used in a supervised learning problem when the output y are all either zero or one. The goal of logistic regression is to minimize the error between its predictions and training data.

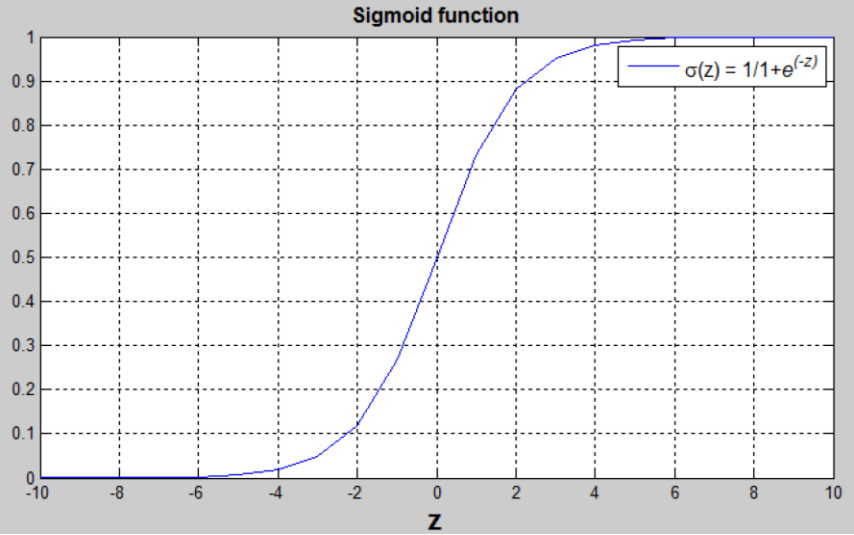
Example: Cat vs No - cat

Given an image represented by a feature vector x, the algorithm will evaluate the probability of a cat being in that image.

$$Given\ x\ ,\ \hat{y} = P(y = 1|x), \text{where } 0\ \leq \hat{y} \leq 1$$

The parameters used in Logistic regression are:

- The input features vector: $x\ \in \mathbb{R}^{n_x}$, where n_x is the number of features
- The training label: $y \in 0,1$
- The weights: $w\ \in \mathbb{R}^{n_x}$, where n_x is the number of features
- The threshold: $b\ \in \mathbb{R}$
- The output: $\hat{y} = \sigma(w^T x + b)$
- Sigmoid function: s = $\sigma(w^T x + b)$ = $\sigma(z)$= $\frac{1}{1+ e^{-z}}$



$(w^T x + b)$ is a linear function $(ax + b)$, but since we are looking for a probability constraint between [0,1], the sigmoid function is used. The function is bounded between [0,1] as shown in the graph above.

Some observations from the graph:

- If z is a large positive number, then $\sigma(z)$ = 1
- If z is small or large negative number, then $\sigma(z)$ = 0
- If $z = 0$, then $\sigma(z)$ = 0.5

Logistic Regression: Cost Function

To train the parameters w and b, we need to define a cost function.

Recap:

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

$x^{(i)}$ the i-th training example

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

Loss (error) function:

The loss function measures the discrepancy between the prediction ($\hat{y}^{(i)}$) and the desired output ($y^{(i)}$). In other words, the loss function computes the error for a single training example.

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

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

- If $y^{(i)} = 1$: $L(\hat{y}^{(i)}, y^{(i)}) = -\log(\hat{y}^{(i)})$ where $\log(\hat{y}^{(i)})$ and $\hat{y}^{(i)}$ should be close to 1
- If $y^{(i)} = 0$: $L(\hat{y}^{(i)}, y^{(i)}) = -\log(1 - \hat{y}^{(i)})$ where $\log(1 - \hat{y}^{(i)})$ and $\hat{y}^{(i)}$ should be close to 0

Cost function

The cost function is the average of the loss function of the entire training set. We are going to find the parameters $w\ and\ b$ that minimize the overall cost function.

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