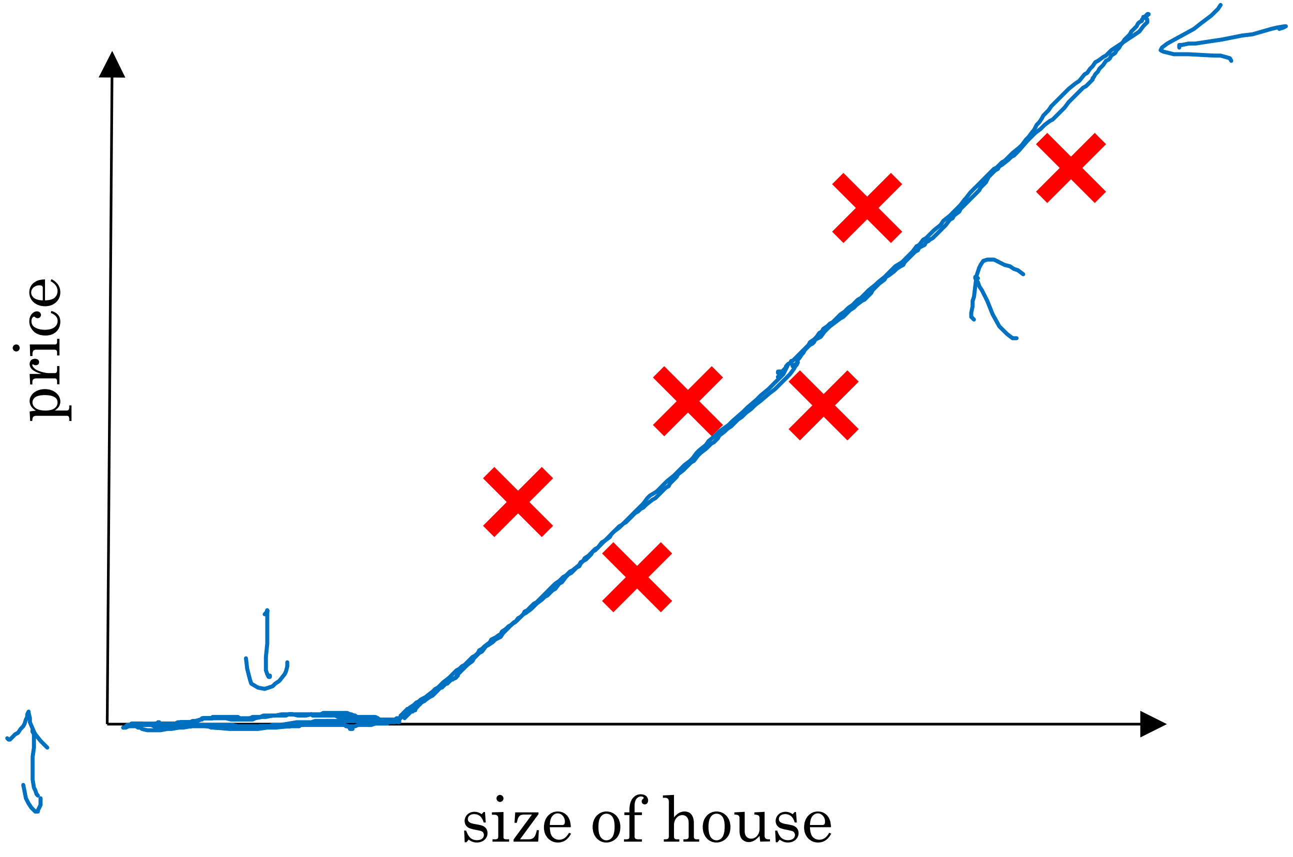
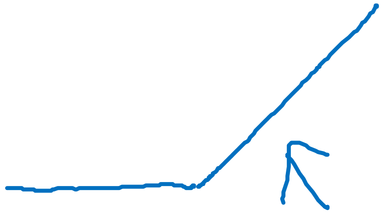
deeplearning.ai

Introduction to Deep Learning

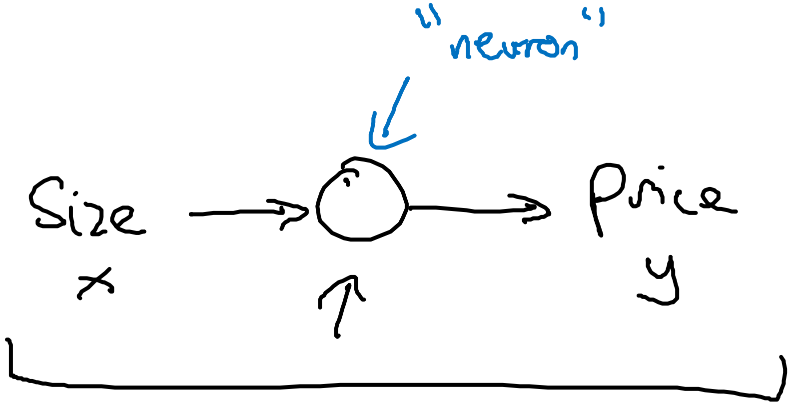
What is a Neural Network?

Housing Price Prediction

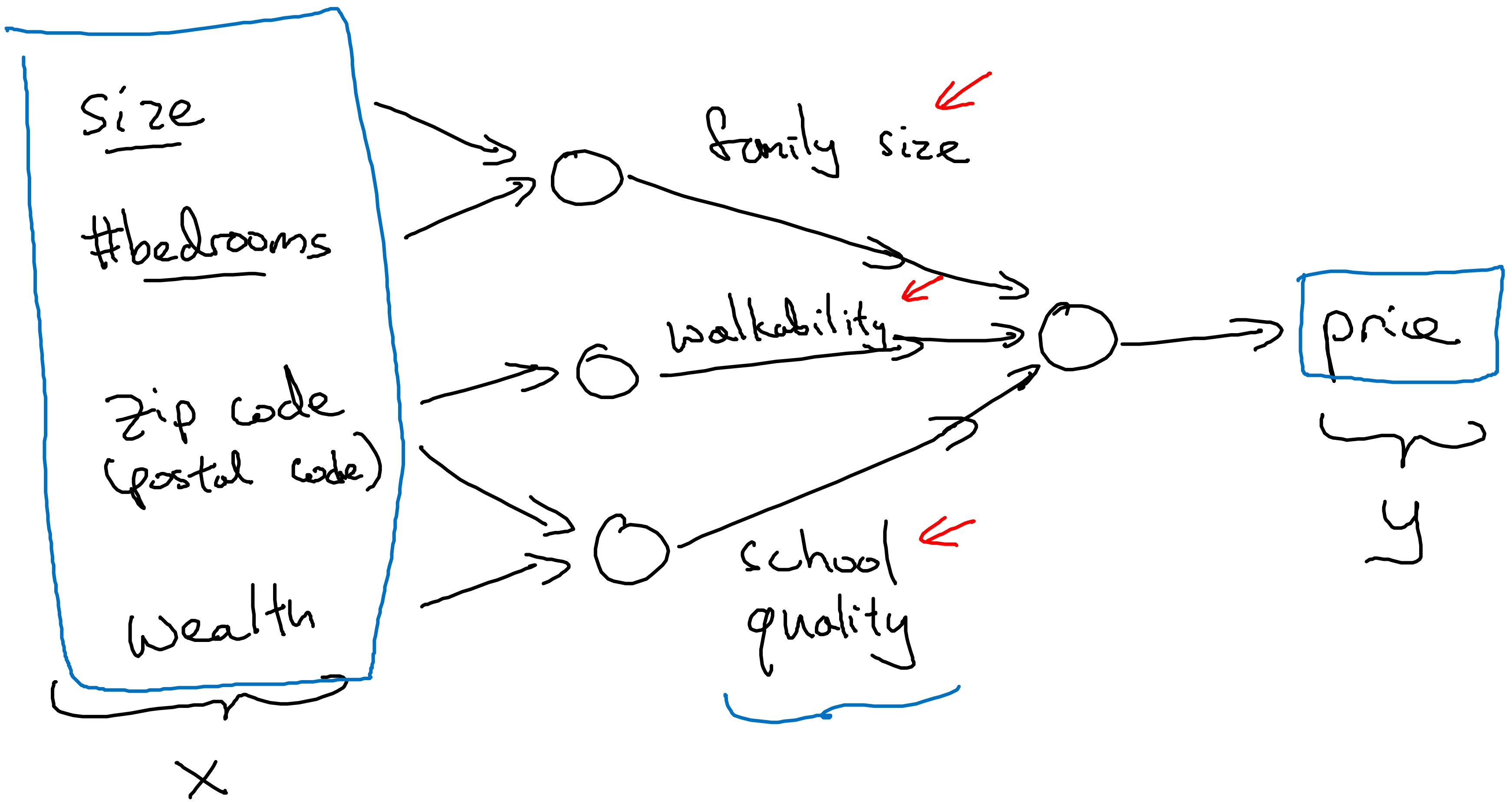




ReLU
Rectified
Linear
Unit

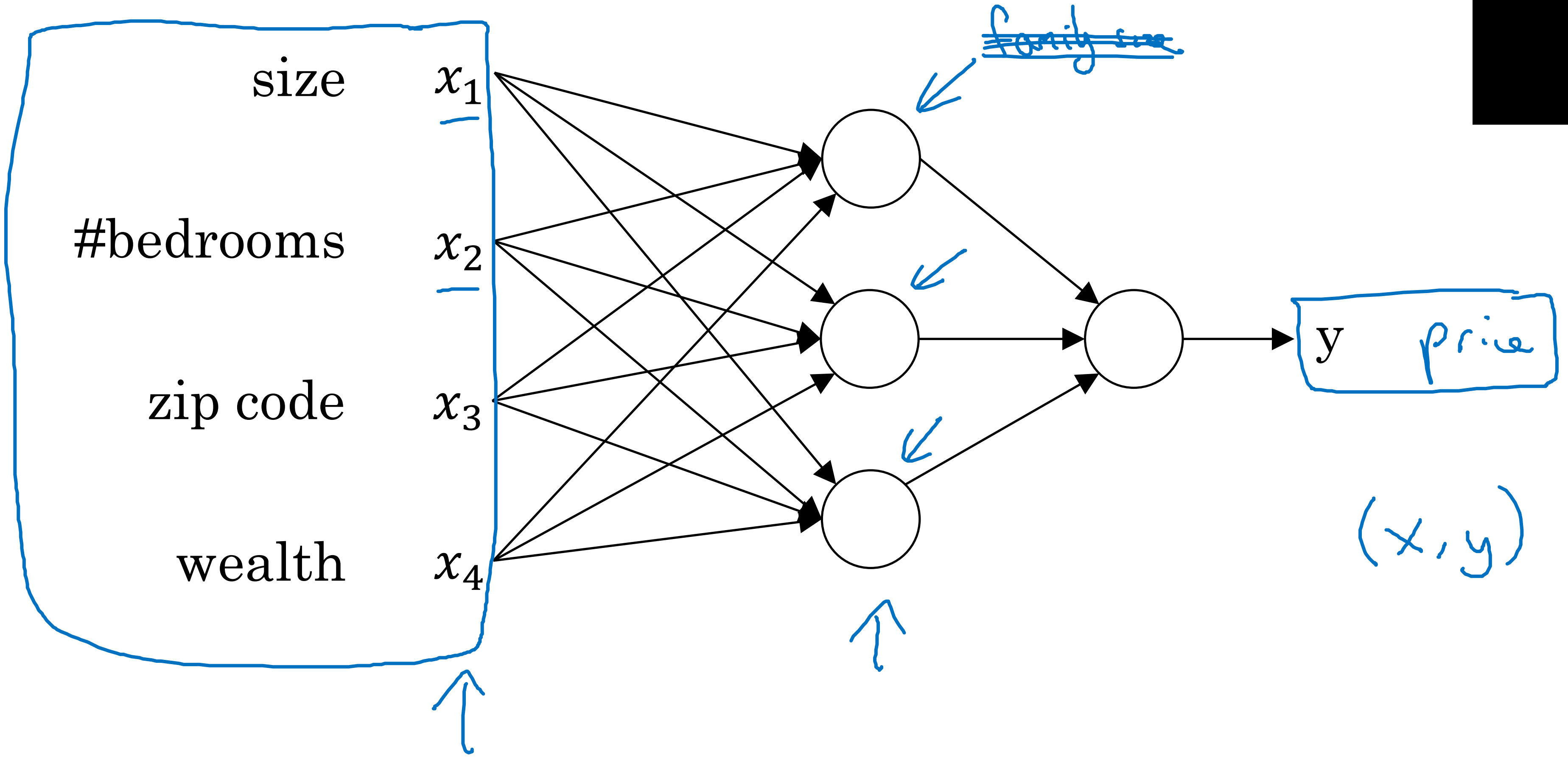


Housing Price Prediction



Housing Price Prediction

Drawing of previous Image



deeplearning.ai

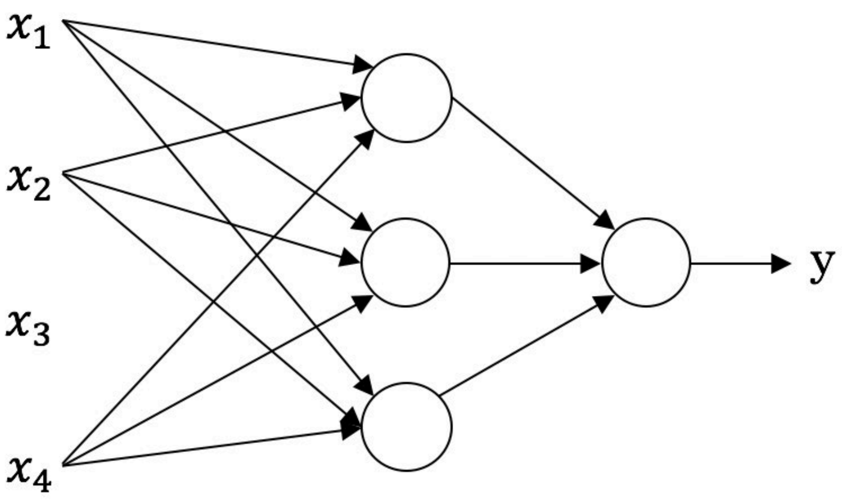
Introduction to Deep Learning

Supervised Learning with Neural Networks

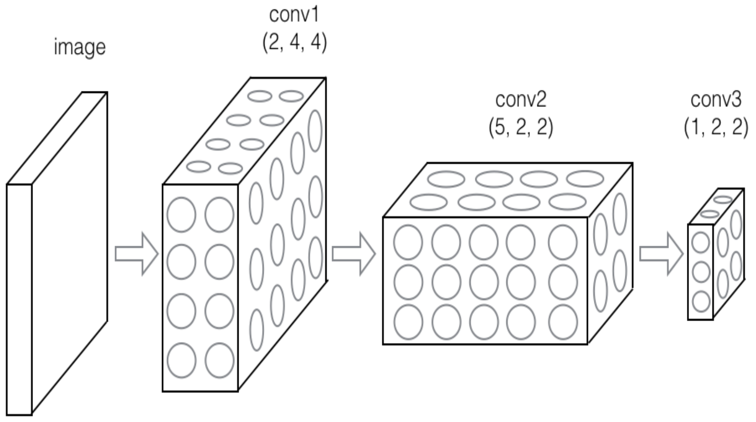
Supervised Learning

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

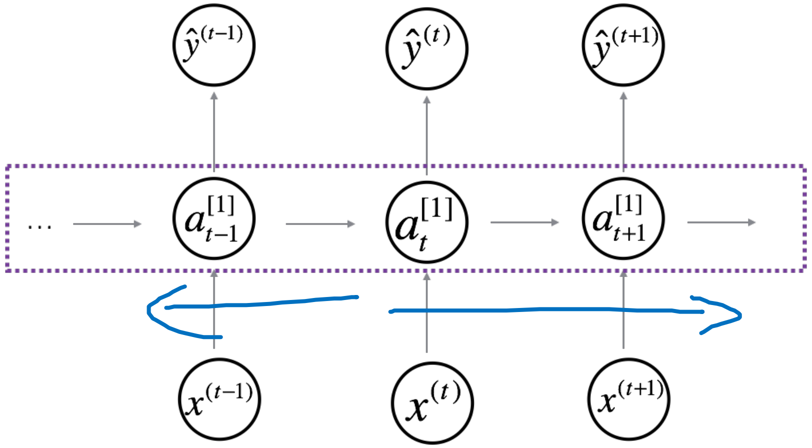
Neural Network examples



Standard NN



Convolutional NN



Recurrent NN

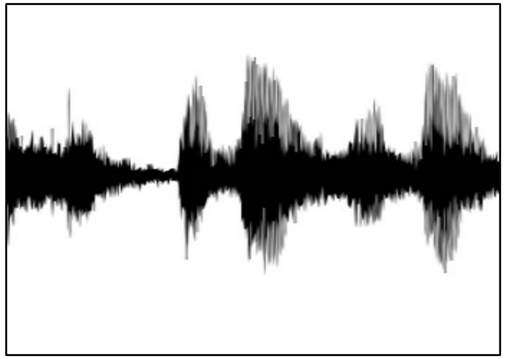
Supervised Learning

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…

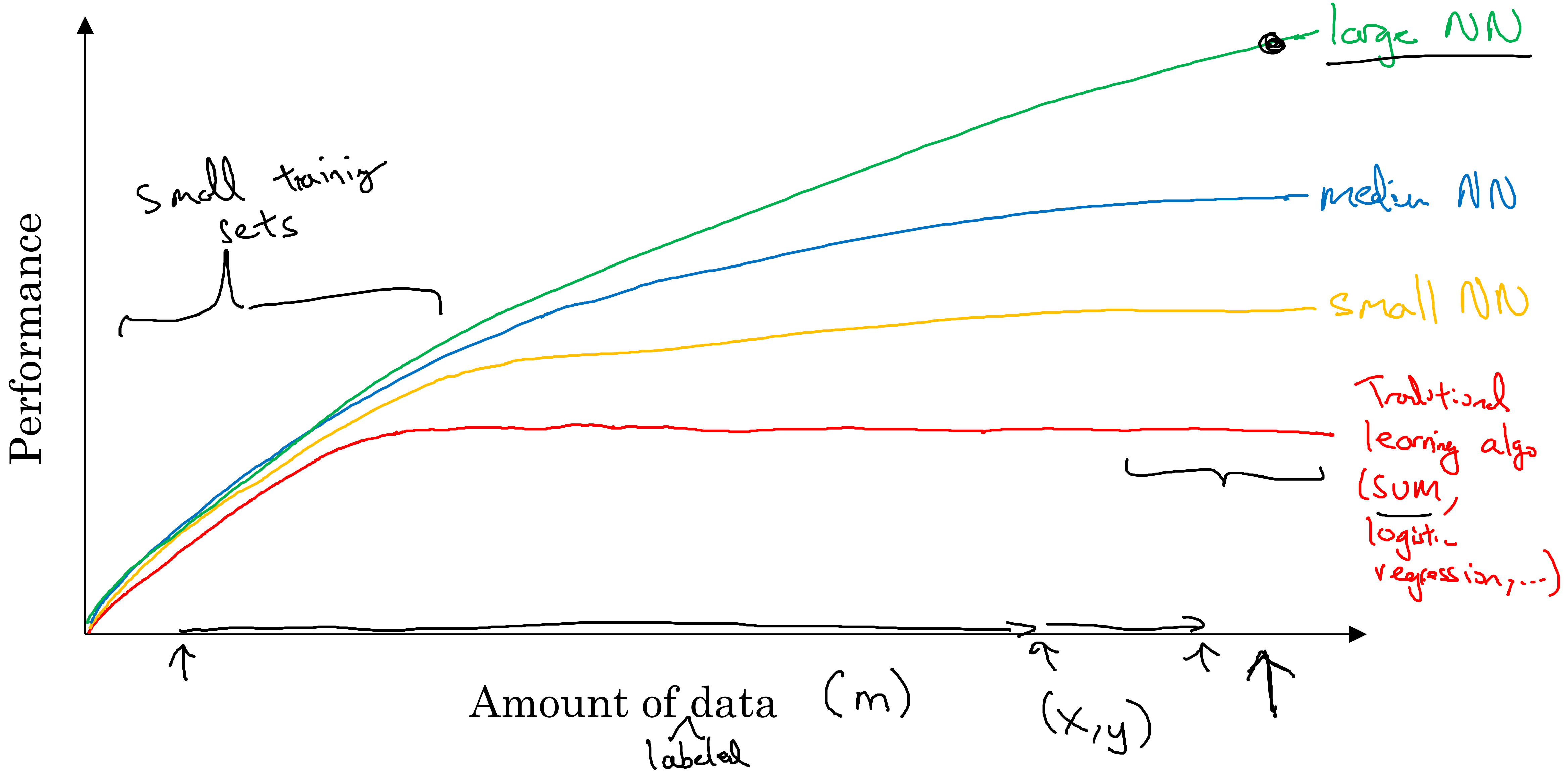
Text

deeplearning.ai

Introduction to Neural Networks

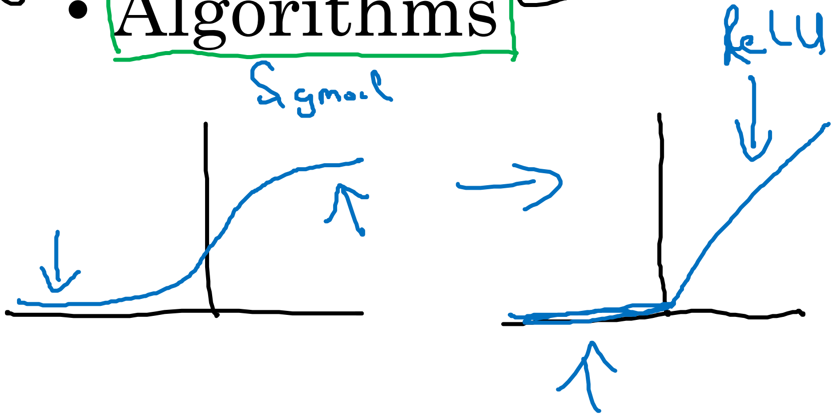
Why is Deep Learning taking off?

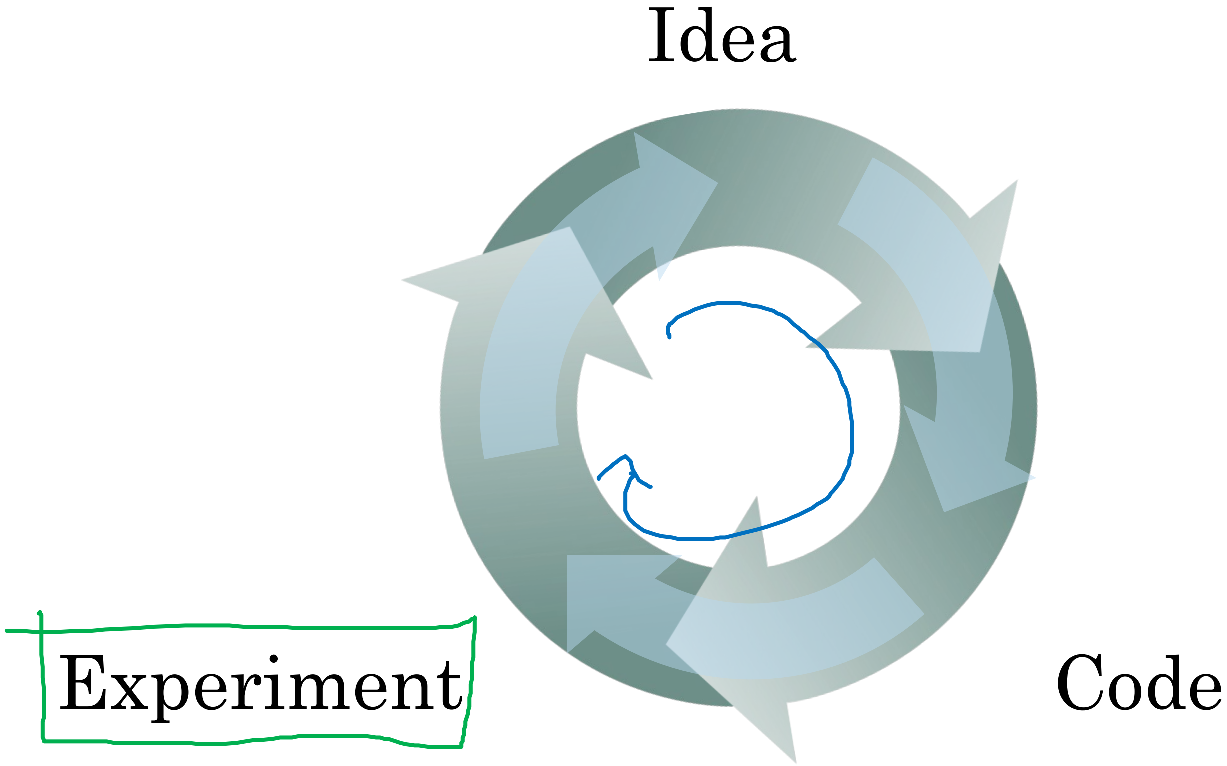
Scale drives deep learning progress



Scale drives deep learning progress

- Data
- Computation
- Algorithms





10 min
1 day
1 month

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

Basics of Neural Network Programming

Binary Classification

deeplearning.ai

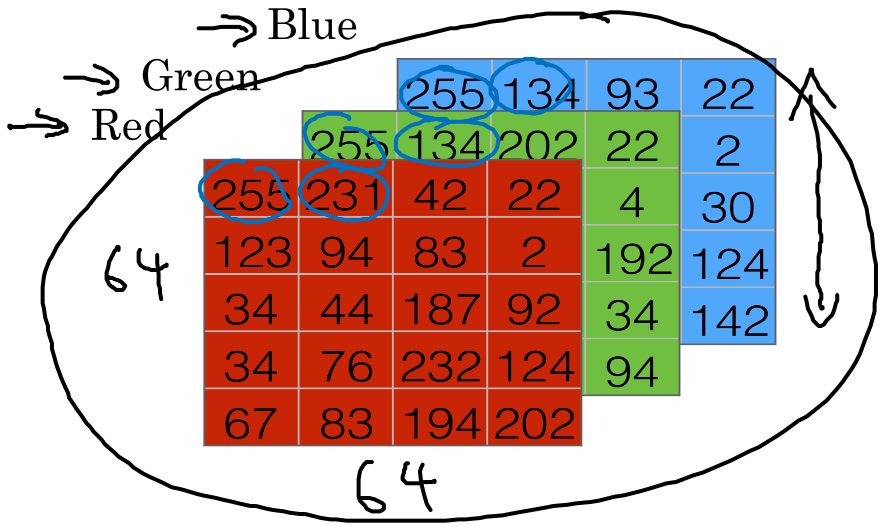
Binary Classification

64

64

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

$M = M_{train}$ $\qquad M_{test} = \#\text{test examples.}$

$$X = \underbrace{\left.\begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}\right\updownarrow n_x}_{m} \qquad \xcancel{\begin{bmatrix} — & x^{(1)T} & — \\ & \vdots & \\ — & x^{(m)T} & — \end{bmatrix}}$$

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

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

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

$Y.\text{shape} = (1, m)$

Basics of Neural Network Programming

Logistic Regression

deeplearning.ai

Logistic Regression

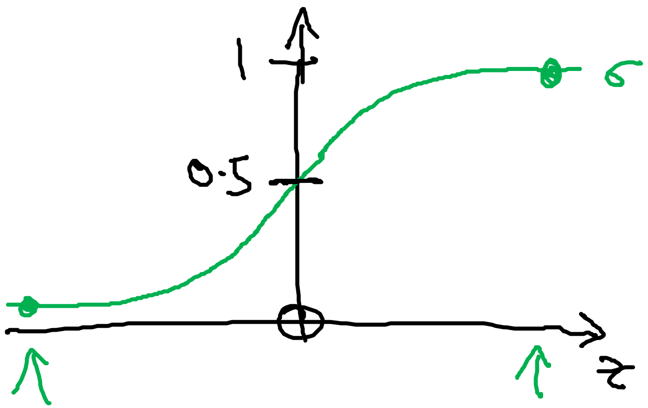
Given x, want $\hat{y} = P(y=1 \mid x)$

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

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

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

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



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

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

$$\underbrace{\Theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_{n_x} \end{bmatrix} \begin{matrix} \} \, b \leftarrow \\ \left.\begin{matrix} \\ \\ \\ \\ \end{matrix}\right\} w \leftarrow \end{matrix}}$$

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

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

If z large negative number

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

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

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

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

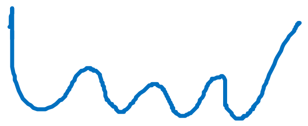
$x^{(i)}$

$y^{(i)}$

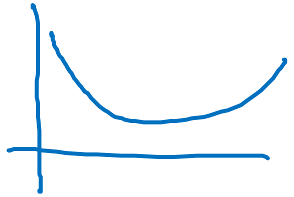
$z^{(i)}$

i-th example.

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



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



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

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

deeplearning.ai

Basics of Neural Network Programming

Logistic Regression cost function

Logistic Regression cost function

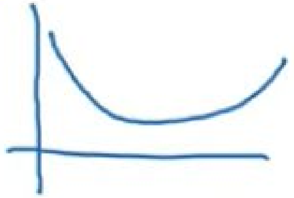
→ $\hat{y}^{(i)} = \underline{\sigma(w^T \underline{x}^{(i)} + b)}$, where $\underline{\sigma(z^{(i)}) = \frac{1}{1+e^{-z^{(i)}}}}$ $z^{(i)} = w^T x^{(i)} + b$

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

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

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

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



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

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

$\boxed{\text{Cost}}$ 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

Basics of Neural Network Programming

Gradient Descent

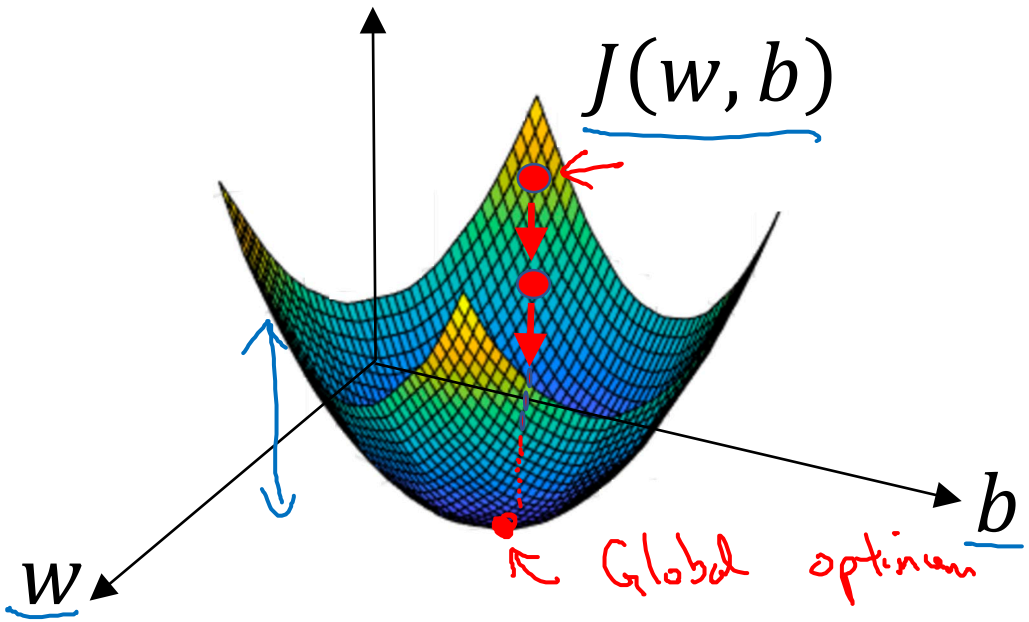
deeplearning.ai

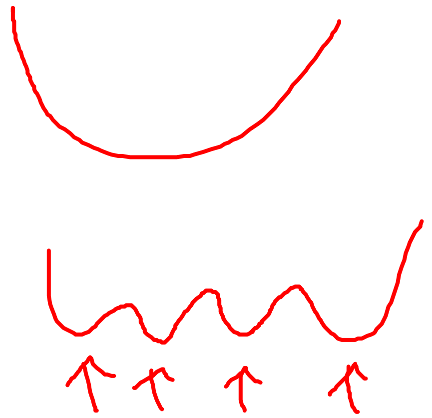
Gradient Descent

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

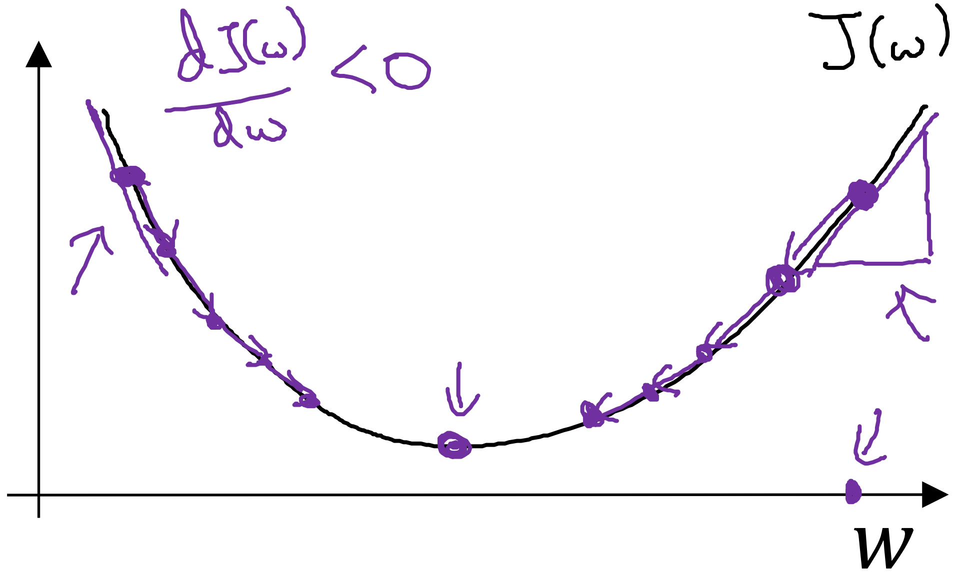
$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\boxed{\hat{y}^{(i)}}, \boxed{y^{(i)}}) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

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





Gradient Descent



learning rate

Repeat {

$$w := w - \alpha \boxed{\frac{dJ(w)}{dw}}$$

}

"dw"

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

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

$J(w,b)$

$$w := w - \alpha \boxed{\frac{dJ(w,b)}{dw}} \qquad \boxed{\frac{\partial J(w,b)}{\partial w}} \rightarrow dw$$

$$b := b - \alpha \boxed{\frac{dJ(w,b)}{db}} \qquad \boxed{\frac{\partial J(w,b)}{\partial b}} \rightarrow db$$

∂ "partial derivative"

d

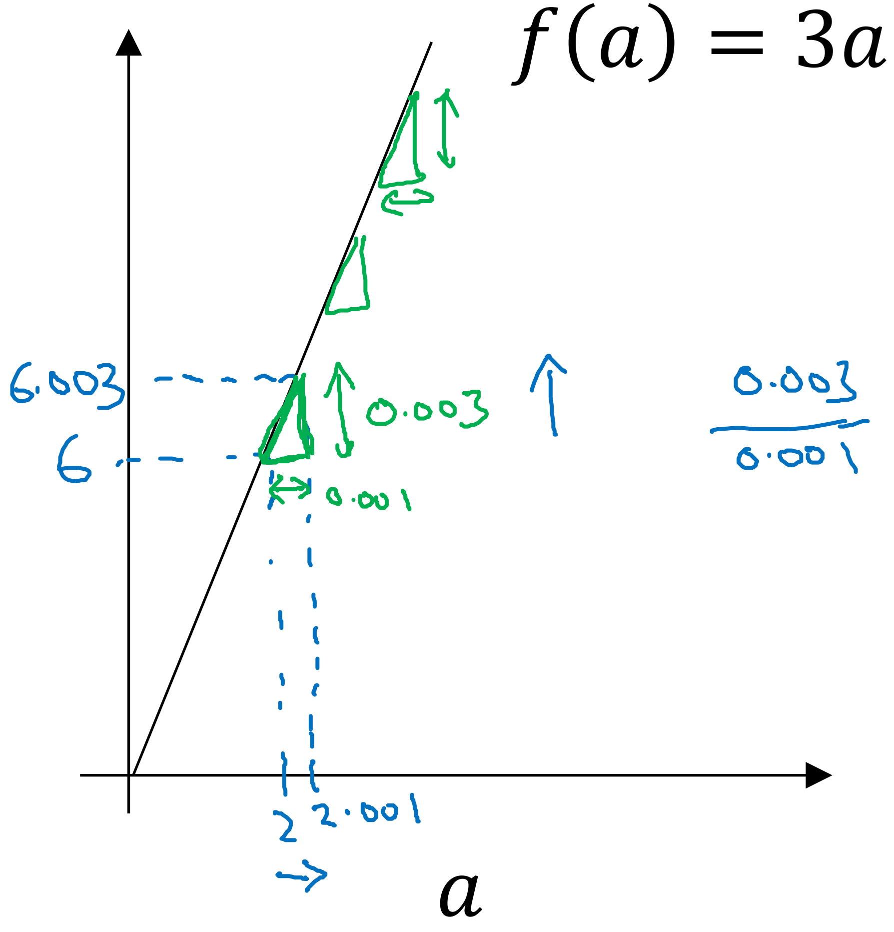
J

deeplearning.ai

Basics of Neural Network Programming

Derivatives

Intuition about derivatives



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

$a = 2$ $f(a) = 6$

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

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

$a = 5$ $f(a) = 15$

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

slope at $a=5$ is also 3

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

0.001

0.00000001

0.000000000001

Basics of Neural Network Programming

deeplearning.ai

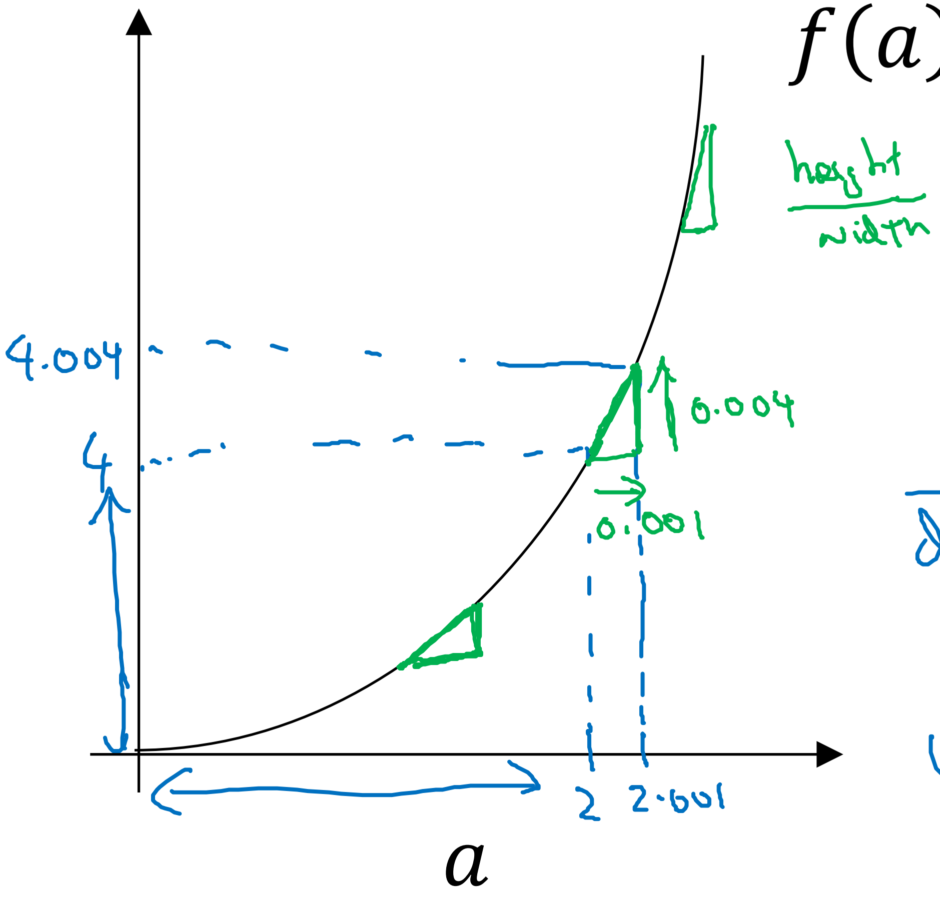
More derivatives examples

Intuition about derivatives

0.001

$0.00000\ldots01$

$f(a) = a^2$



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

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

$(4.004\boxed{001})$

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

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

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

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

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

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

0.001

$(2a) \times 0.001$

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

More derivative examples

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

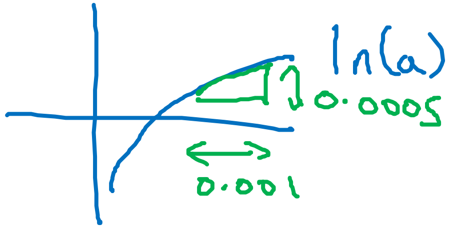
$\qquad a = 2.001 \quad f(a) \approx 4.004$

$f(a) = a^3$ $\qquad \frac{d}{da} f(a) = \underbrace{3a^2}_{3 \times 2^2 = 12}$ $\qquad a = 2 \quad f(a) = 8$

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

$f(a) = \log_e(a)$ $\qquad \frac{d}{da} f(a) = \frac{1}{a}$ $\qquad a = 2 \quad f(a) \approx 0.69315$

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



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

0.0005 $\qquad 0.0005$

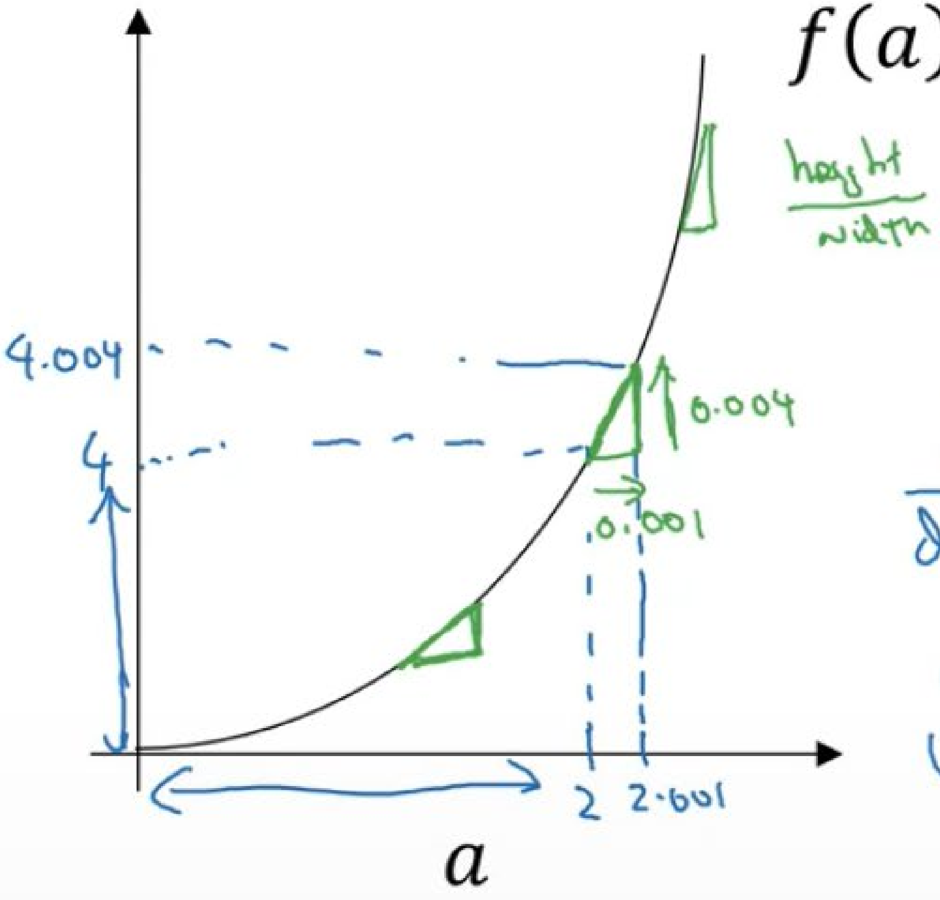
deeplearning.ai

Basics of Neural Network Programming

More derivatives examples

Intuition about derivatives

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



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

0.001

$(2a) \times 0.001$

0.001

0.00000....01

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

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

$(4.004\boxed{001})$

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

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

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

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

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

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

Andrew Ng

More derivative examples

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

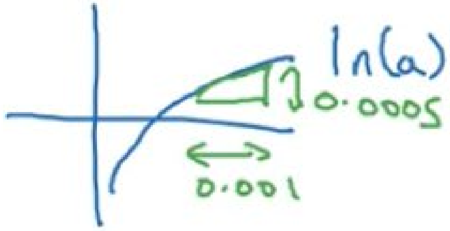
$\qquad\qquad\qquad\qquad\qquad\qquad a = 2.001 \quad f(a) \approx 4.004$

$f(a) = a^3$ $\qquad \frac{d}{da} f(a) = \underbrace{3a^2}_{3\times 2^2 = 12}$ $\qquad a = 2 \quad f(a) = 8$

$\qquad\qquad\qquad\qquad\qquad\qquad a = 2.\underline{001} \quad f(a) \approx 8.\underline{012}$

$f(a) = \log_e(a)$ $\qquad \frac{d}{da} f(a) = \frac{1}{a}$ $\qquad a = 2 \quad f(a) \approx 0.69315$

$\ln(a)$ $\qquad\qquad\qquad\qquad\qquad\qquad a = 2.\underline{001} \quad \underline{f(a)} \approx 0.69365$



$\frac{d}{da} f(a) = \boxed{\frac{1}{2}}$ $\qquad 0.0005 \longleftrightarrow \underline{0.0005}$

Andrew Ng

deeplearning.ai

Basics of Neural Network Programming

Computation Graph

Computation Graph

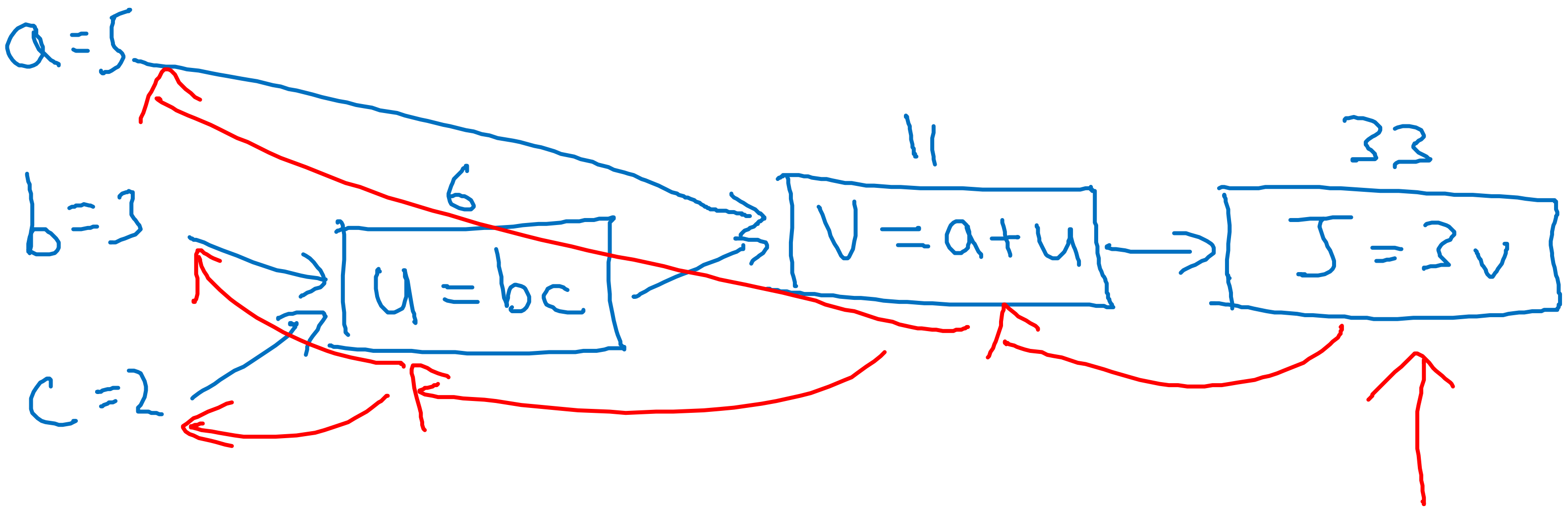
$$J(a,b,c) = \underbrace{3(\overbrace{a + \underbrace{bc}_{u}}^{})}_{J} = 3(5 + 3\times 2) = 33$$

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

$u = bc$

$V = a + u$

$J = 3v$



Basics of Neural Network Programming

Derivatives with a Computation Graph

deeplearning.ai

Computing derivatives

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

$a = 5$

$b = 3$

$c = 2$

6

$u = bc$

11

$v = a + u$

33

$J = 3v$

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

$J = 3v$

$v = 11 \rightarrow 11.001$

$J = 33 \rightarrow 33.003$

$f(a) = 3a$

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

$J = 3v$

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

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

$a \rightarrow v \rightarrow J$

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

3×1

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

$a = 5 \rightarrow 5.001$

$\rightarrow v = 11 \rightarrow 11.001$

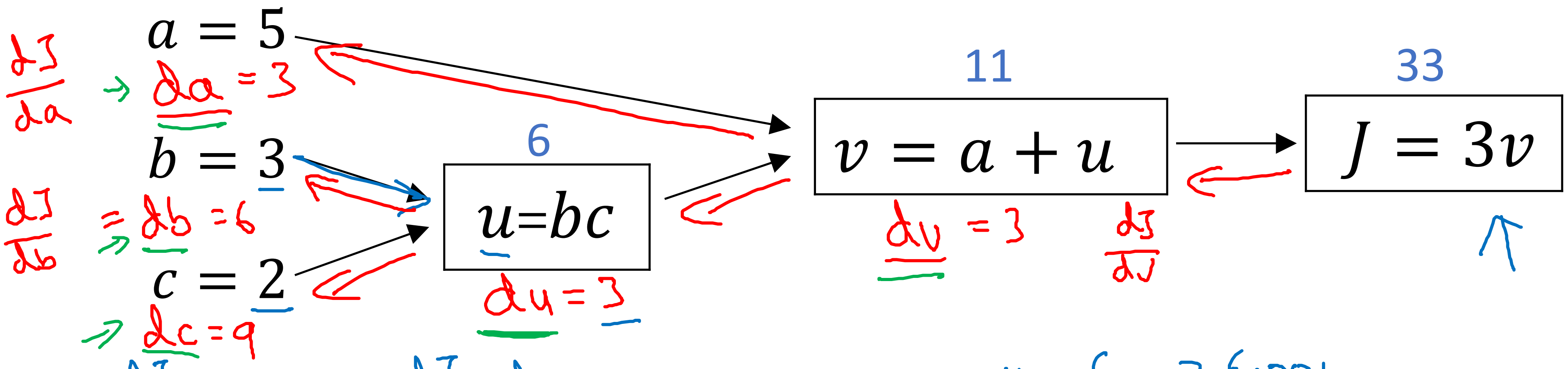
$J = 33 \rightarrow 33.003$

$\frac{d\,\mathrm{FinalOutputVar}}{d\,\mathrm{var}}$

dJdvar

"dvar"

Computing derivatives



$$\frac{dJ}{du} = 3 = \underbrace{\frac{dJ}{dv}}_{3} \cdot \underbrace{\frac{dv}{du}}_{1}$$

$$\frac{dJ}{db} = \underbrace{\boxed{\frac{dJ}{du}}}_{\to 3} \cdot \underbrace{\frac{du}{db}}_{=2} = 6$$

$$\frac{dJ}{da} = \underbrace{\boxed{\frac{dJ}{du}}}_{3} \cdot \underbrace{\frac{du}{da}}_{3} = 9 \qquad 3 \times 3$$

$u = 6 \rightarrow 6.001$

$v = 11 \rightarrow 11.001$

$J = 33 \rightarrow 33.003$

$b = 3 \rightarrow 3.001$

$u = b \cdot c = 6 \rightarrow 6.002$ $\qquad c = 2$

$J = 33.006$ $\qquad .006$

$v = 11.002$

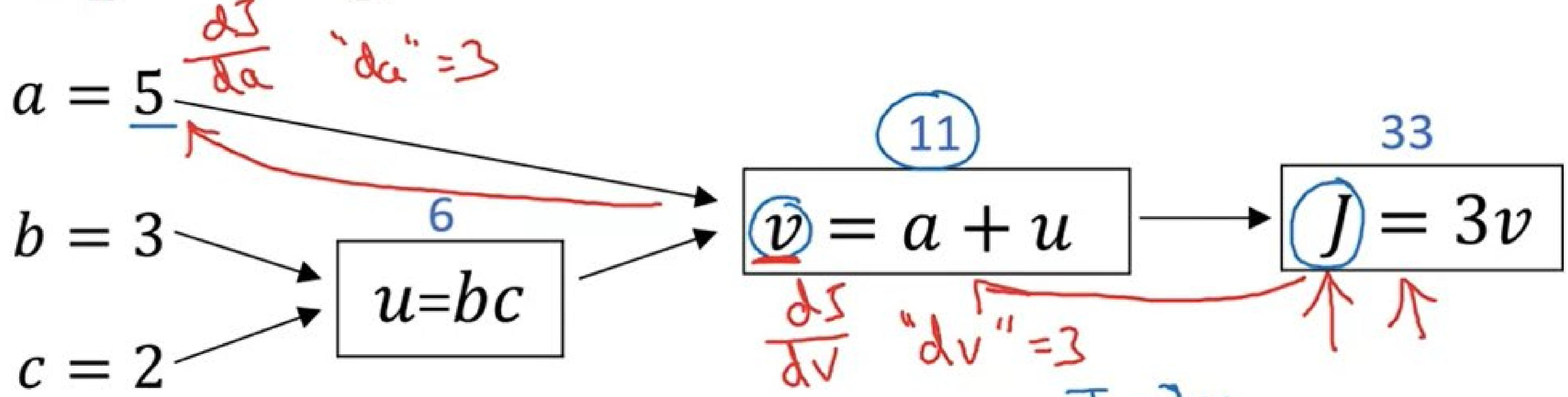
$J = 3v$

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

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

$a \to v \to J$

$J = 3v$

$v = 11 \to 11.001$

$J = 33 \to 33.003$

$a = 5 \to 5.001$

$v = 11 \to 11.001$

$J = 33 \to 33.003$

$$\frac{d\,\text{FinalOutputVar}}{d\,\text{var}}$$

dJdvar

"dvar"

$f(a) = 3a$

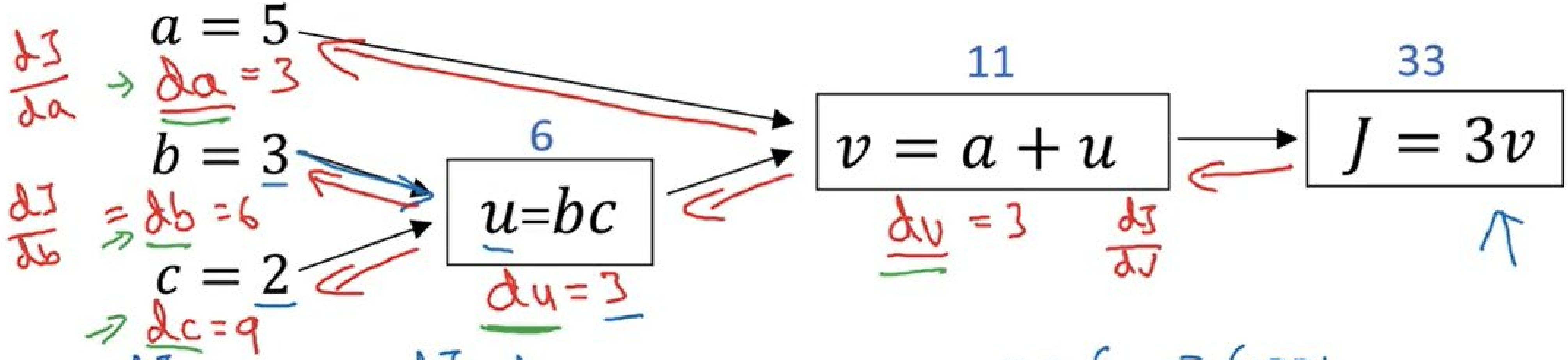
$$\frac{df(a)}{da} = \frac{df}{da} = 3$$

$J = 3v$

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

Andrew Ng

Computing derivatives



$$\frac{dJ}{du} = 3 = \underbrace{\frac{dJ}{dv}}_{3} \cdot \underbrace{\frac{dv}{du}}_{1}$$

$$\frac{dJ}{db} = \underbrace{\boxed{\frac{dJ}{du}}}_{\to 3} \cdot \underbrace{\frac{du}{db}}_{=2} = 6$$

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

$$3 \times 3$$

$u = 6 \to 6.001$

$v = 11 \to 11.001$

$J = 33 \to 33.003$

$b = 3 \to 3.001$

$u = b \cdot c = 6 \to 6.002$ $\quad c = 2$

$J = 33.006$ $\quad .006$

$v = 11.002$

$J = 3v$

Basics of Neural Network Programming

Logistic Regression Gradient descent

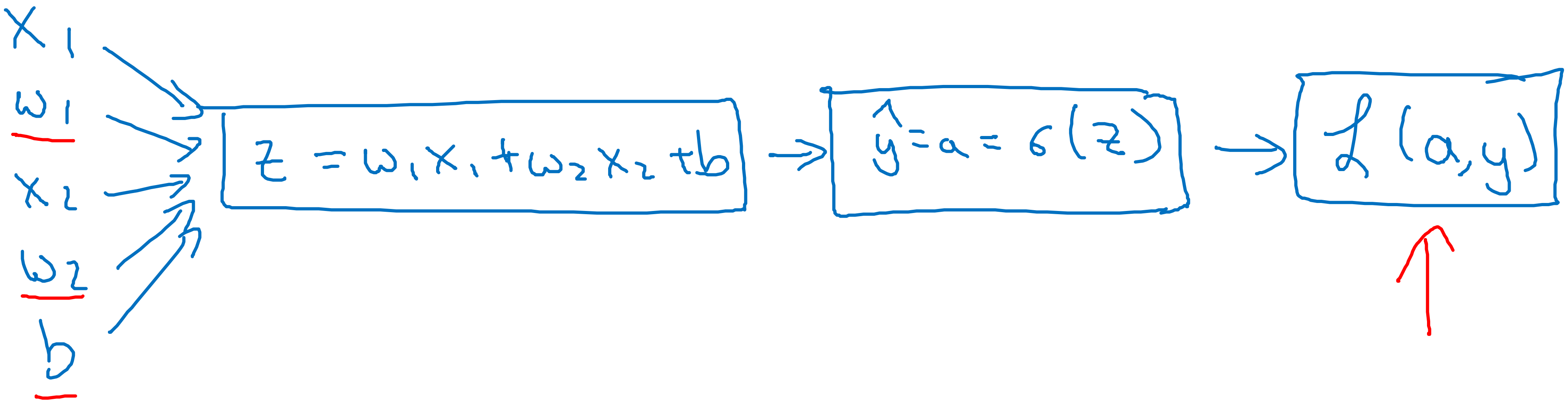
deeplearning.ai

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

x_1, w_1, x_2, w_2, b $\rightarrow$ $\boxed{z = w_1 x_1 + w_2 x_2 + b}$ $\rightarrow$ $\boxed{a = \sigma(z)}$ $\rightarrow$ $\boxed{\mathcal{L}(a, y)}$

$$dz = \frac{d\mathcal{L}}{dz} = \frac{d\mathcal{L}(a,y)}{dz}$$

$$= a - y$$

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

$$a(1-a)$$

$$\text{“}da\text{”} = \frac{d\mathcal{L}(a,y)}{da} = -\frac{y}{a} + \frac{1-y}{1-a}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \text{“}dw_1\text{”} = x_1 \cdot dz. \qquad dw_2 = x_2 \cdot dz. \qquad db = dz.$$

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

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

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

deeplearning.ai

Basics of Neural Network Programming

Gradient descent on m examples

Logistic regression on m examples

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

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

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

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

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

Logistic regression on m examples

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

For $i=1$ to m

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

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

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

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

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

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

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

$n=2$

dw_3
$\vdots$
dw_n

$J /= m$

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

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

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

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

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

Vectorization

deeplearning.ai

Basics of Neural Network Programming

Gradient descent on m examples

Logistic regression on m examples

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

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

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

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

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

Andrew Ng

Logistic regression on m examples

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

For $i = 1$ to m

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

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

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

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

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

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

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

$n = 2$

dw_3
$\vdots$
dw_n

$J /= m$

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

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

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

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

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

Vectorization

Andrew Ng

Basics of Neural Network Programming

Vectorization

deeplearning.ai

What is vectorization?

$z = \underbrace{w^T x} + b$

$w = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix}$ $x = \begin{bmatrix} \vdots \\ \vdots \end{bmatrix}$

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

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

Non-vectorized:

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

Vectorized

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

→ 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.

Neural network programming guideline

Whenever possible, avoid explicit for-loops.

$u = Av$

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

```
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
```

Logistic regression derivatives

$dw = np.zeros((n\_x, 1))$

$J = 0$, $\boxed{\cancel{dw1 = 0,\ dw2 = 0}}$, $db = 0$

for i = 1 to n:

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

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

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

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

for $j = 1 \ldots n_x$
$dw_j \mathrel{+}= \ldots$

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

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

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

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

$dw /= m.$

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)
```

Andrew Ng

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, $\boxed{\cancel{\text{dw1 = 0, dw2 = 0}}}$, db = 0

$dw = np.zeros((n\_x, 1))$

for i = 1 to m:

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

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

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

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

for $j = 1 \ldots n_x$

$dw_j \mathrel{+}= \ldots$

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

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

$n_x = 2$

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

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

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

$dw \mathrel{/}= m.$

Andrew Ng

deeplearning.ai

Basics of Neural Network Programming

Vectorizing Logistic Regression

Vectorizing Logistic Regression

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

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

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

$$[\;\text{———}\;]\; w^T \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

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

$1 \times m$

"Broadcasting"

$$\rightarrow Z = np.dot(w.T, X) + b$$

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

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

deeplearning.ai

Basics of Neural Network Programming

Vectorizing Logistic Regression's Gradient Computation

Vectorizing Logistic Regression

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

$$\boxed{dZ = [dz^{(1)} \; dz^{(2)} \; \ldots \; dz^{(m)}]}$$

$1 \times m$

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

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

$\rightarrow dw = 0$

$dw \mathrel{+}= x^{(1)} dz^{(1)}$

$dw \mathrel{+}= x^{(2)} dz^{(2)}$

$\vdots$

$dw \mathrel{/}= m$

$\cancel{dw_1}$

$\cancel{dw_2}$

$\vdots$

$db = 0$

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

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

$\vdots$

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

$db \mathrel{/}= m.$

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

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

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

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

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

$n \times 1$

Implementing Logistic Regression

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

$\boxed{\text{for } i = 1 \text{ to } m:}$

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

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

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

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

$$\left.\begin{aligned} dw_1 &\mathrel{+}= x_1^{(i)} dz^{(i)} \\ dw_2 &\mathrel{+}= x_2^{(i)} dz^{(i)} \end{aligned}\right\} \quad dw \mathrel{+}= x^{(i)} * dz^{(i)}$$

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

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

$db = db/m$

for iter in range(1000):

$$Z = w^T X + b$$

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

$$A = \sigma(Z)$$

$$dZ = A - Y$$

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

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

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

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

deeplearning.ai

Basics of Neural Network Programming

Vectorizing Logistic Regression's Gradient Computation

Vectorizing Logistic Regression

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

$$\boxed{dZ = [\, dz^{(1)} \;\; dz^{(2)} \; \cdots \; dz^{(m)} \,]} \leftarrow$$

$1 \times m$

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

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

$\rightarrow dw = 0$

$dw \mathrel{+}= x^{(1)} dz^{(1)}$

$dw \mathrel{+}= x^{(2)} dz^{(2)}$

$\vdots$

$dw \mathrel{/}= m$

$\cancel{dw_1}$

$\cancel{dw_2}$

$\vdots$

$db = 0$

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

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

$\vdots$

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

$db \mathrel{/}= m$

$$\boxed{db = \frac{1}{m} \sum_{i=1}^{m} dz^{(i)} = \frac{1}{m} \, np.sum(dZ)}$$

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

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

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

$n \times 1$

Andrew Ng

Implementing Logistic Regression

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

$\boxed{\text{for } i = 1 \text{ to } m:}$

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

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

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

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

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

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

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

$db = db/m$

for iter in range(1000):

$Z = w^T X + b$

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

$A = \sigma(Z)$

$dZ = A - Y$

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

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

$w := w - \alpha dw$

$b := b - \alpha db$

Andrew Ng

deeplearning.ai

Basics of Neural Network Programming

Broadcasting in Python

Broadcasting example

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

$$
\begin{array}{r|cccc}
 & \text{Apples} & \text{Beef} & \text{Eggs} & \text{Potatoes} \\
\hline
\text{Carb} & 56.0 & 0.0 & 4.4 & 68.0 \\
\text{Protein} & 1.2 & 104.0 & 52.0 & 8.0 \\
\text{Fat} & 1.8 & 135.0 & 99.0 & 0.9
\end{array}
= A \quad (3,4)
$$

59 cal

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

0

1

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

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

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

$(1,n) \rightsquigarrow (m,n)$ $\quad (2,3)$

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

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

General Principle

$$(m, n) \quad \begin{matrix} + \\ - \\ * \\ / \end{matrix} \quad \begin{matrix} (1, n) \rightsquigarrow (m, n) \\ (m, 1) \rightsquigarrow (m, n) \end{matrix}$$

matrix

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

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

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

Matlab/Octave: bsxfun

deeplearning.ai

Basics of Neural Network Programming

A note on python/ numpy vectors

Python Demo

Python / numpy vectors

```
import numpy as np

a = np.random.randn(5)

a = np.random.randn((5,1))

a = np.random.randn((1,5))

assert(a.shape = (5,1))
```

deeplearning.ai

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

Andrew Ng

Logistic regression cost function

$$\left.\begin{array}{l} \boxed{\text{If } \quad y = 1: \qquad p(y|x) = \hat{y}} \\ \boxed{\text{If } \quad y = 0: \qquad p(y|x) = 1 - \hat{y}} \end{array}\right\} \quad p(y|x)$$

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

$$\text{If } y=1: \quad \underline{p(y|x) = \hat{y}} \ \underbrace{(1-\hat{y})^{0}}_{=1}$$

$$\underline{\text{If } y=0}: \quad \underline{p(y|x)} = \underbrace{\hat{y}^{0}}_{=1} \ (1-\hat{y})^{(1-0)} \quad = 1 \times (1-\hat{y}) = \underline{1-\hat{y}}$$

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

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

Andrew Ng

Cost on m examples

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

$$\underline{\log p(\ldots\ldots)} = \sum_{i=1}^{m} \underbrace{\log p(y^{(i)} \mid x^{(i)})}_{-\mathcal{L}(\hat{y}^{(i)},\, y^{(i)})}$$

Maximum likelihood estimation

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

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

(minimize)

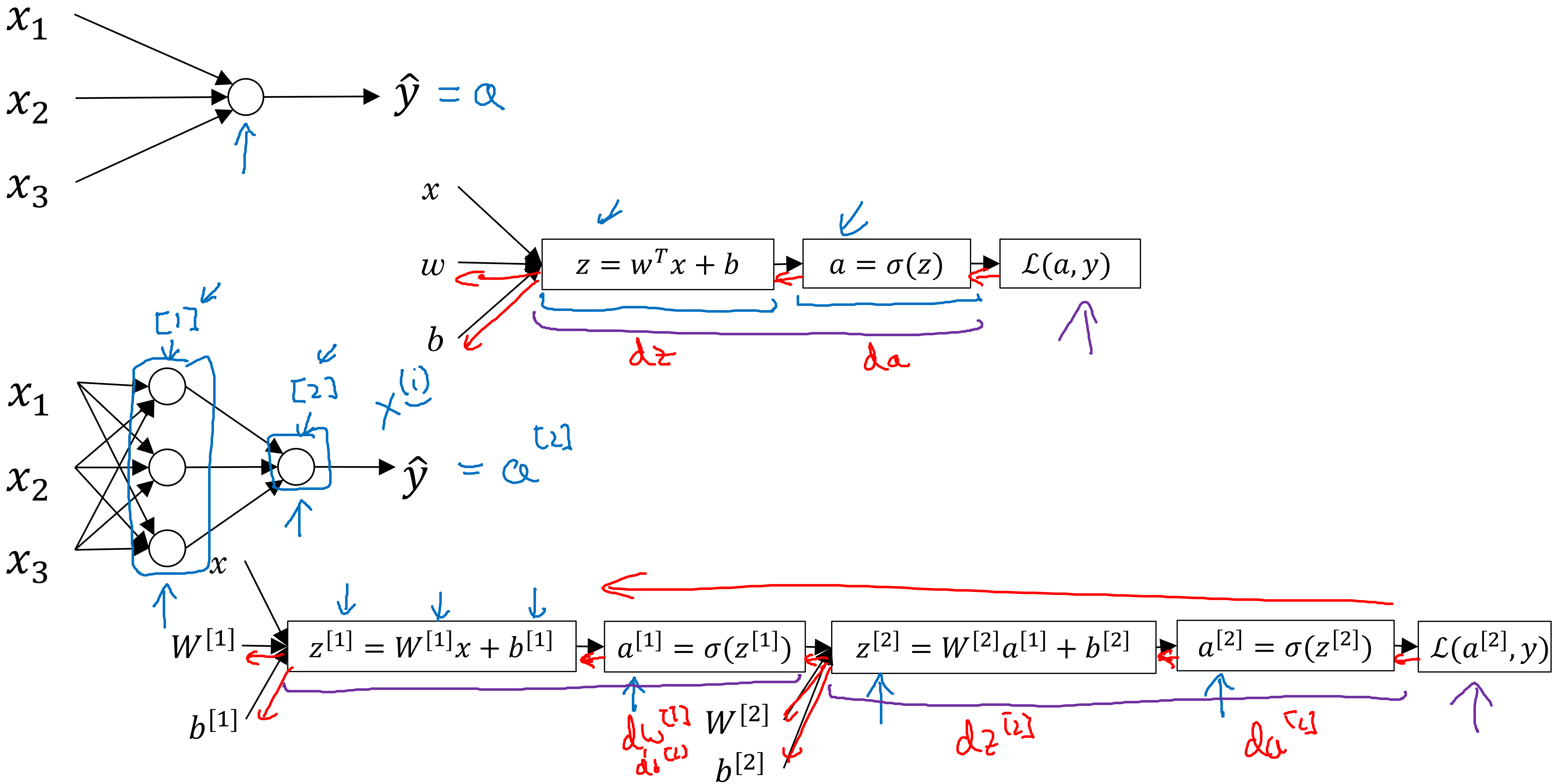
Andrew Ng

deeplearning.ai

One hidden layer Neural Network

Neural Networks Overview

What is a Neural Network?

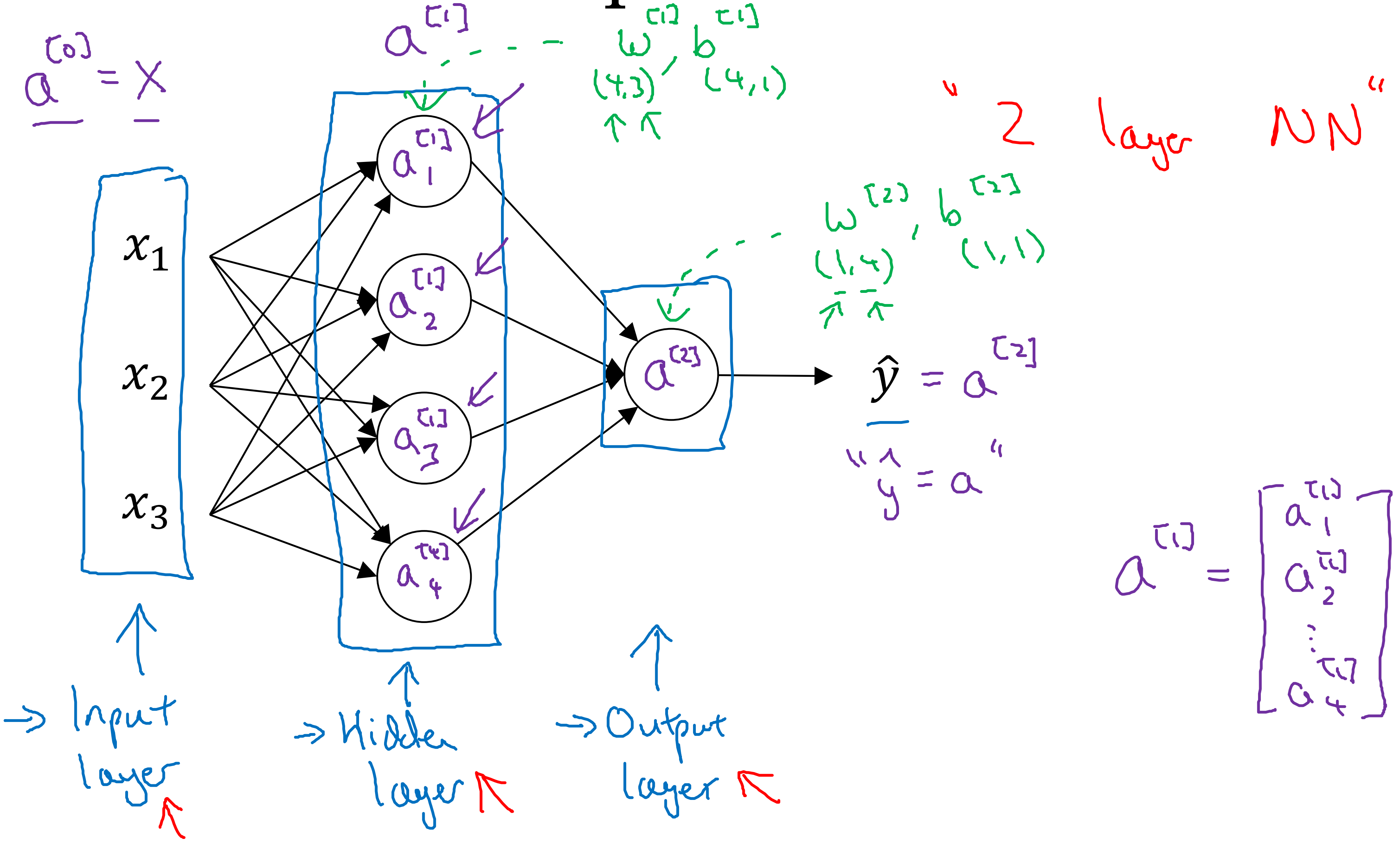


deeplearning.ai

One hidden layer Neural Network

Neural Network Representation

Neural Network Representation

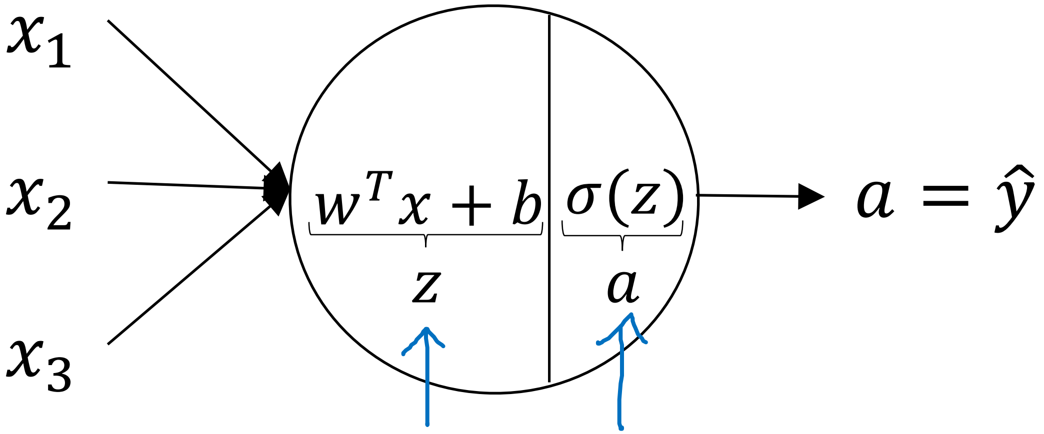


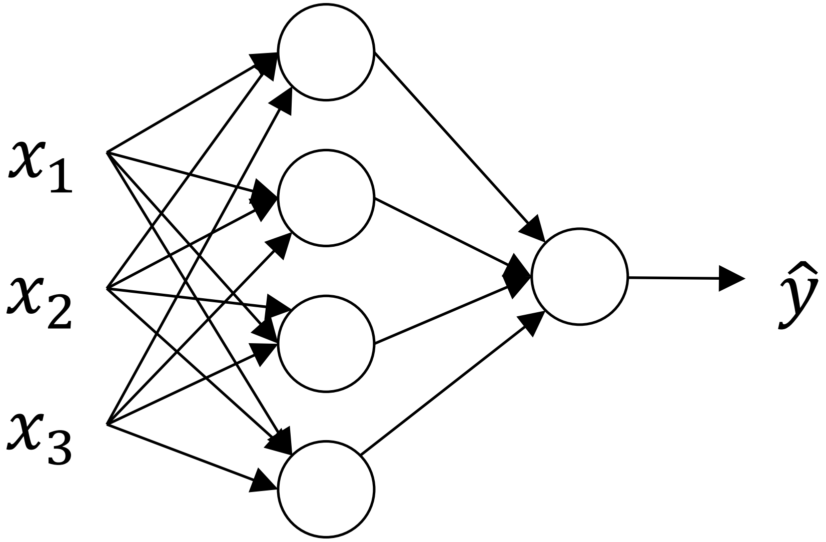
deeplearning.ai

One hidden layer Neural Network

Computing a Neural Network's Output

Neural Network Representation

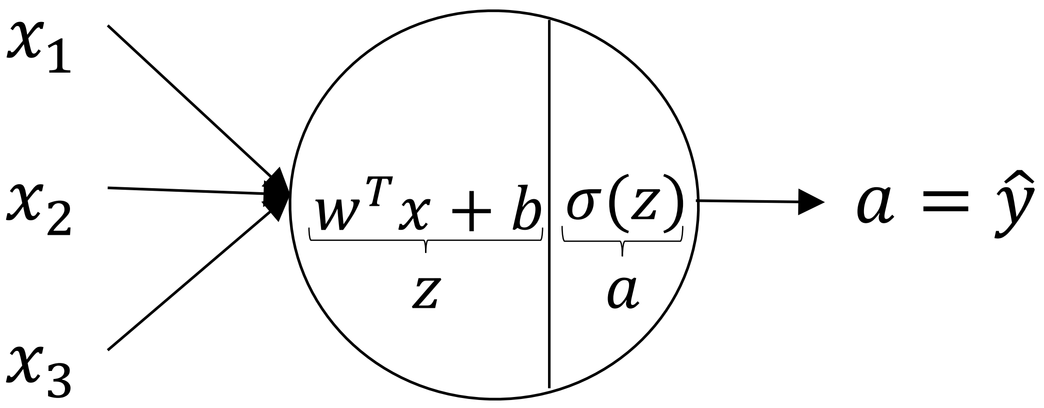




$$z = w^T x + b$$

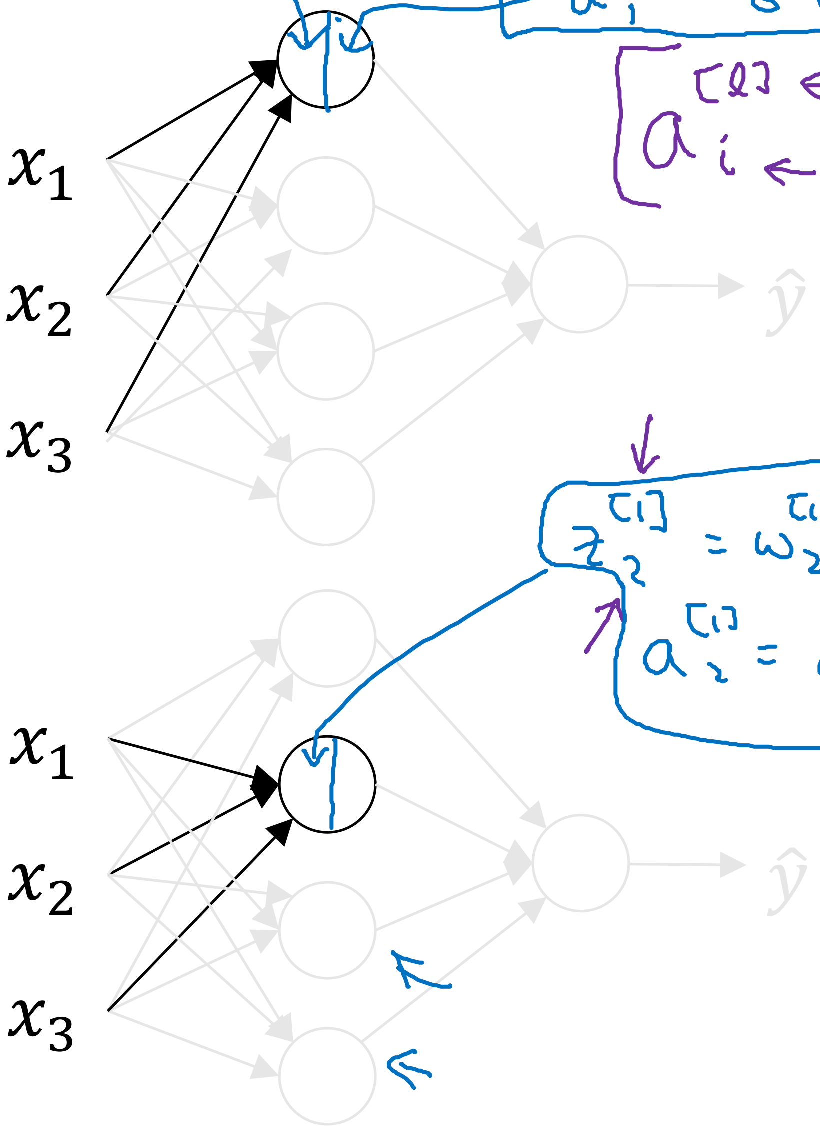
$$a = \sigma(z)$$

Neural Network Representation



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

$$a = \sigma(z)$$



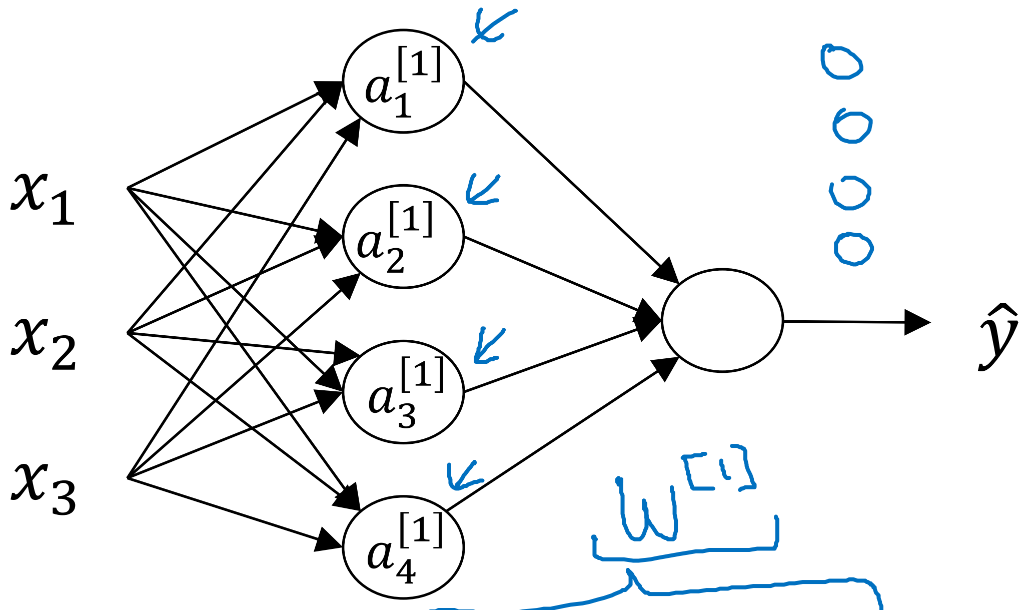
$$\boxed{\begin{aligned} z_1^{[1]} &= w_1^{[1]T} x + b_1^{[1]} \\ a_1^{[1]} &= \sigma(z_1^{[1]}) \end{aligned}}$$

$a_i^{[l]}$ ← layer

← node in layer.

$$\boxed{\begin{aligned} z_2^{[1]} &= w_2^{[1]T} x + b_2^{[1]} \\ a_2^{[1]} &= \sigma(z_2^{[1]}) \end{aligned}}$$

Neural Network Representation



$(w_1^{[1]})^T x$ $\quad a^{[1]}$

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

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

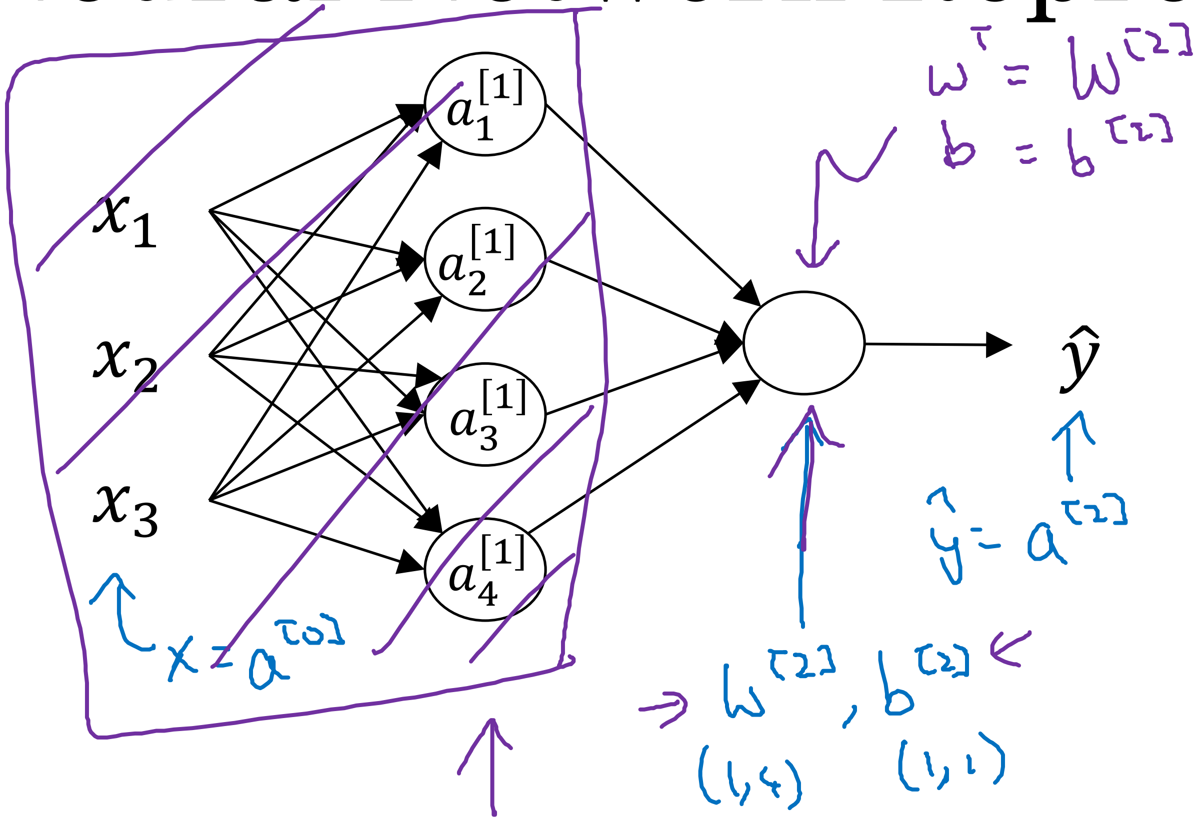
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, \quad a_3^{[1]} = \sigma(z_3^{[1]})$$

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

$$z^{[1]} = \overbrace{\begin{bmatrix} — w_1^{[1]T} — \\ — w_2^{[1]T} — \\ — w_3^{[1]T} — \\ — w_4^{[1]T} — \end{bmatrix}}^{W^{[1]}}_{(4,3)} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}}_{b^{[1]}\ (4,1)} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

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

Neural Network Representation learning



$z = w^T x + b$

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

Given input x:

$$\underset{(4,1)}{z^{[1]}} = \underset{(4,3)}{W^{[1]}}\, \underset{(3,1)}{\overset{a^{[0]}}{\cancel{x}}} + \underset{(4,1)}{b^{[1]}}$$

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

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

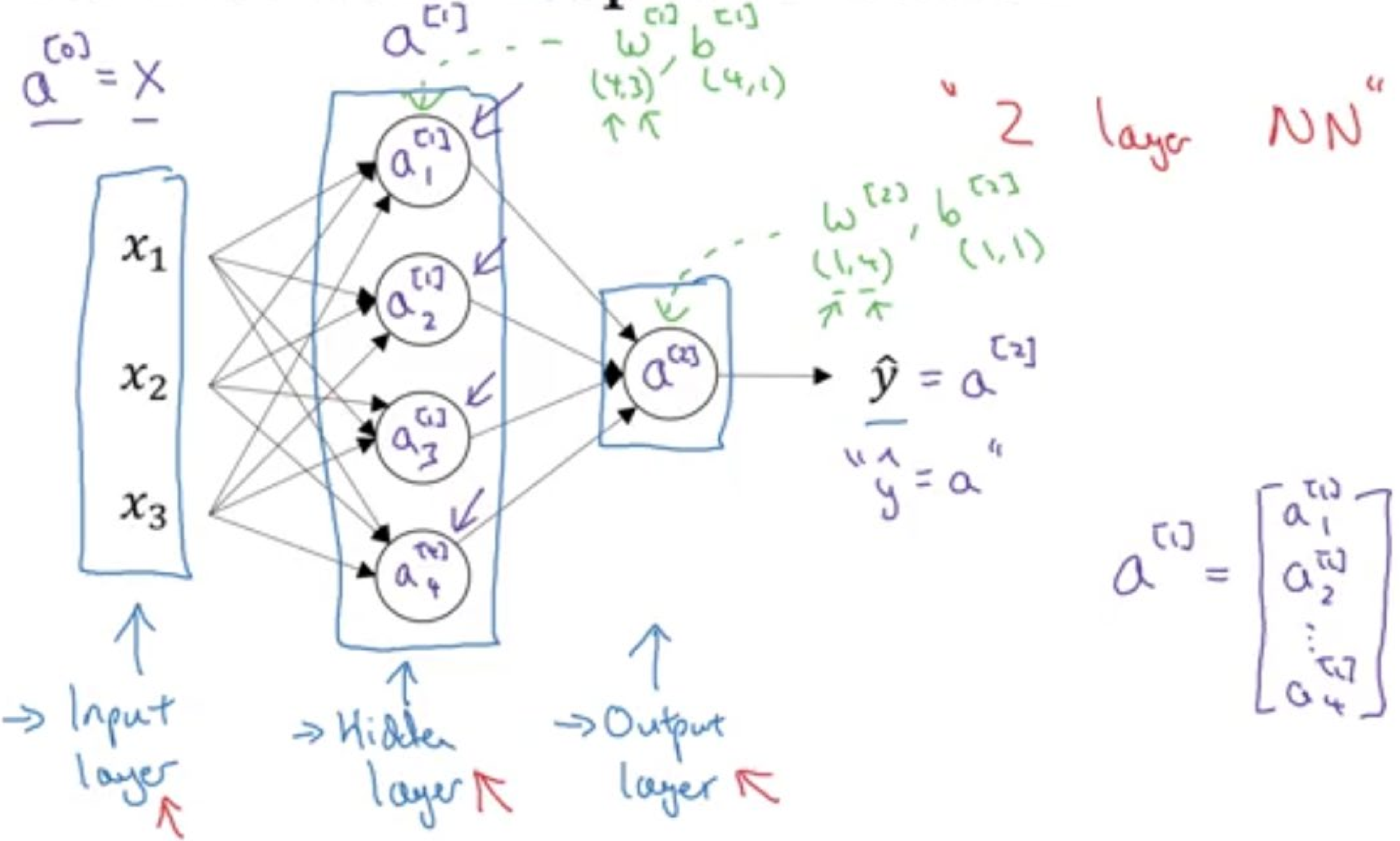
$$\underset{(1,1)}{a^{[2]}} = \sigma(\underset{(1,1)}{z^{[2]}})$$

One hidden layer Neural Network

Computing a Neural Network's Output

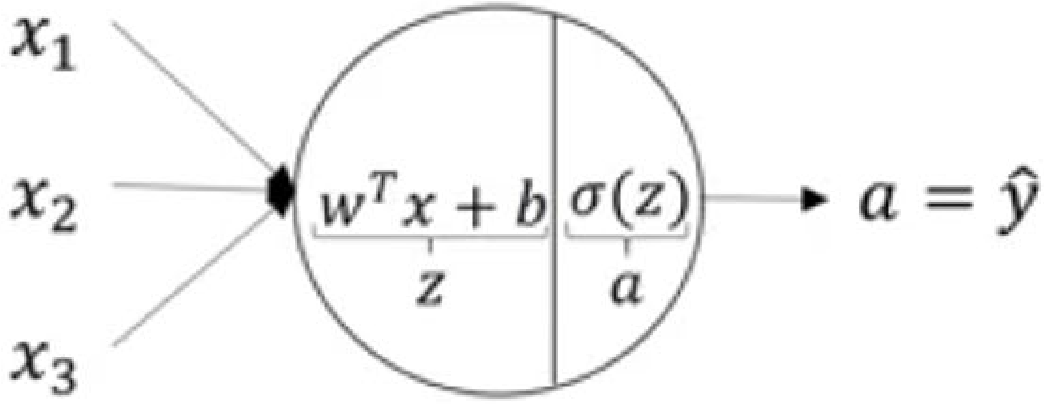
deeplearning.ai

Neural Network Representation



Andrew Ng

Neural Network Representation



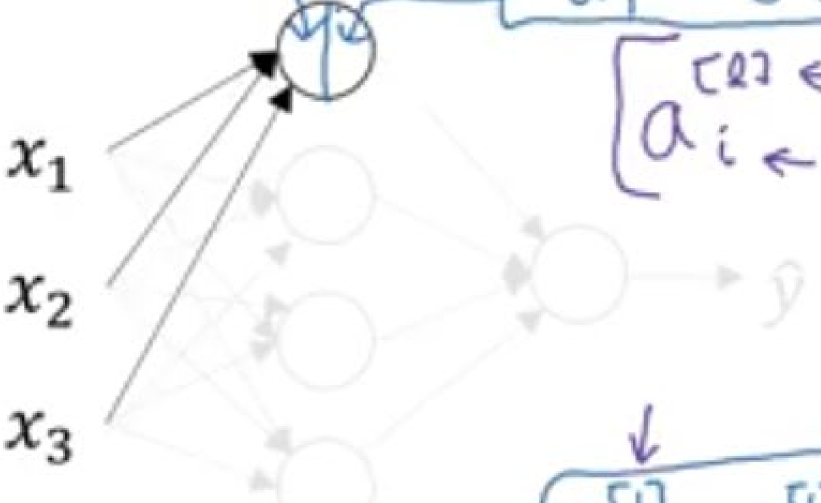
$$z = w^T x + b$$

$$a = \sigma(z)$$

$$z^{[1]}_1 = w^{[1]T}_1 x + b^{[1]}_1$$

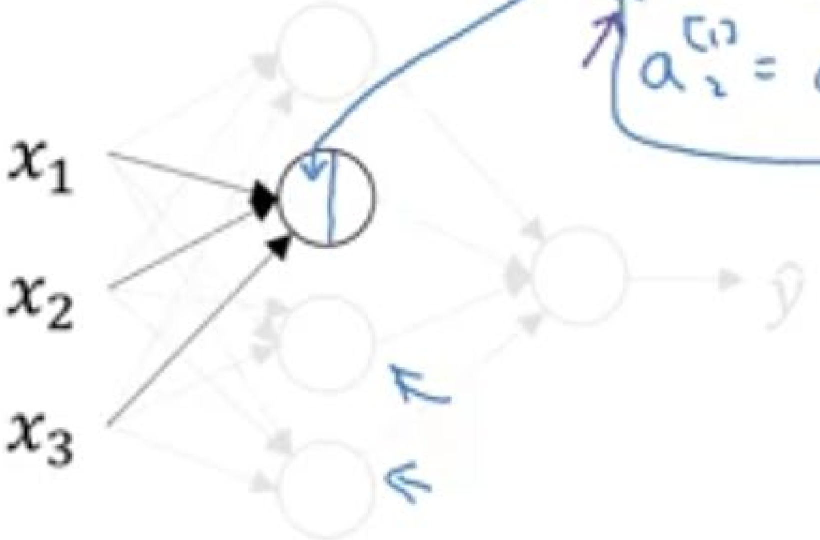
$$a^{[1]}_1 = \sigma(z^{[1]}_1)$$

$a^{[l]}_i$ ← layer, ← node in layer.



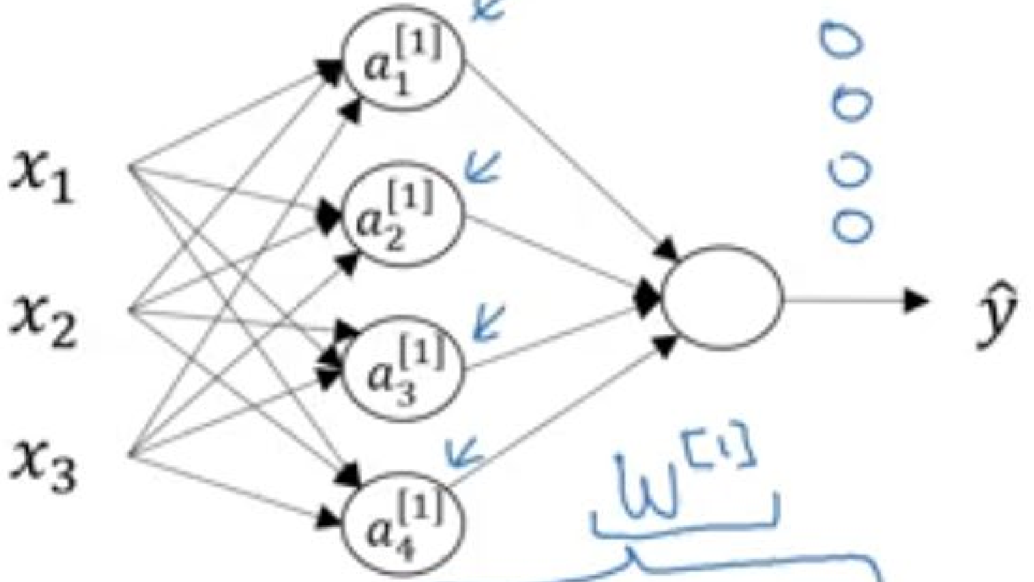
$$z^{[1]}_2 = w^{[1]T}_2 x + b^{[1]}_2$$

$$a^{[1]}_2 = \sigma(z^{[1]}_2)$$



Andrew Ng

Neural Network Representation



$(w_1^{[1]})^T x \qquad a^{[1]}$

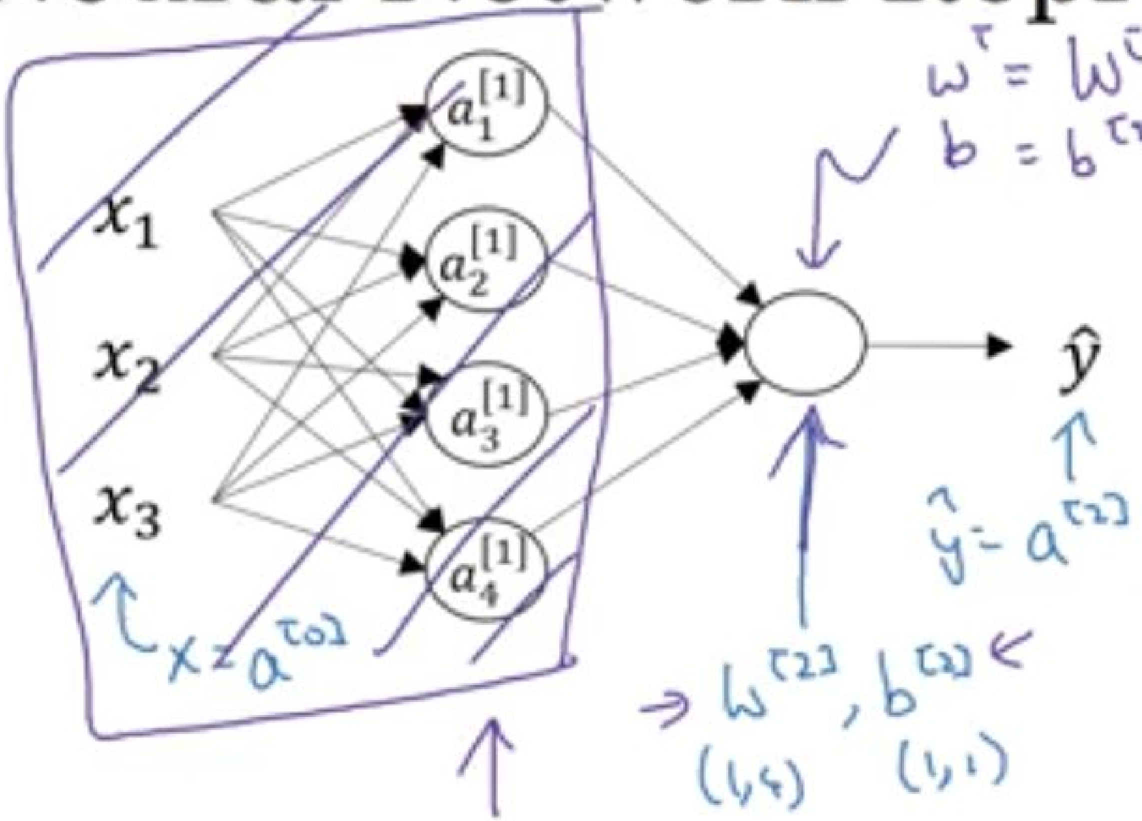
$$
\begin{aligned}
z_1^{[1]} &= w_1^{[1]T} x + b_1^{[1]}, & a_1^{[1]} &= \sigma(z_1^{[1]}) \\
z_2^{[1]} &= w_2^{[1]T} x + b_2^{[1]}, & a_2^{[1]} &= \sigma(z_2^{[1]}) \\
z_3^{[1]} &= w_3^{[1]T} x + b_3^{[1]}, & a_3^{[1]} &= \sigma(z_3^{[1]}) \\
z_4^{[1]} &= w_4^{[1]T} x + b_4^{[1]}, & a_4^{[1]} &= \sigma(z_4^{[1]})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

$$
z^{[1]} = \overbrace{\underbrace{\begin{bmatrix} — w_1^{[1]T} — \\ — w_2^{[1]T} — \\ — w_3^{[1]T} — \\ — w_4^{[1]T} — \end{bmatrix}}_{(4,3)}}^{W^{[1]}} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}}_{b^{[1]} \ (4,1)} = \begin{bmatrix} w_1^{[1]T} x + b_1^{[1]} \\ w_2^{[1]T} x + b_2^{[1]} \\ w_3^{[1]T} x + b_3^{[1]} \\ w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}
$$

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

Andrew Ng

Neural Network Representation learning



$z = w^T x + b$

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

Given input x:

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

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

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

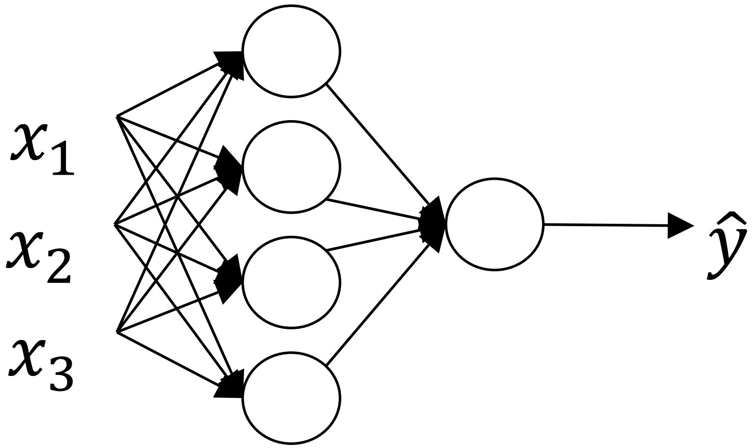
$$\underset{(1,1)}{a^{[2]}} = \sigma(\underset{(1,1)}{z^{[2]}})$$

deeplearning.ai

One hidden layer Neural Network

Vectorizing across multiple examples

Vectorizing across multiple examples



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

$X \longrightarrow a^{[2]} = \hat{y}$

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

$x^{(2)} \longrightarrow a^{[2](2)} \quad \hat{y}^{(2)}$

$\vdots$

$x^{(m)} \longrightarrow a^{[2](m)} \quad \hat{y}^{(m)}$

$a^{[2](i)}$ 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:
```

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

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

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

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

$$\boxed{\begin{aligned} Z^{[1]} &= W^{[1]}X + b^{[1]} \\ A^{[1]} &= \sigma(Z^{[1]}) \\ Z^{[2]} &= W^{[2]}A^{[1]} + b^{[2]} \\ A^{[2]} &= \sigma(Z^{[2]}) \end{aligned}}$$

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

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

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

training examples

hidden units

hidden units

deeplearning.ai

One hidden layer Neural Network

Explanation for vectorized implementation

Justification for vectorized implementation

$$z^{[1](1)} = W^{[1]}x^{(1)} + \cancel{b^{[1]}} \to 0, \quad z^{[1](2)} = W^{[1]}x^{(2)} + \cancel{b^{[1]}} \to 0, \quad z^{[1](3)} = W^{[1]}x^{(3)} + \cancel{b^{[1]}} \to 0$$

$$W^{[1]} = \begin{bmatrix} \text{---} \\ \text{---} \\ \text{---} \\ \text{---} \end{bmatrix} \qquad W^{[1]}x^{(1)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} \qquad W^{[1]}x^{(2)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} \qquad W^{[1]}x^{(3)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

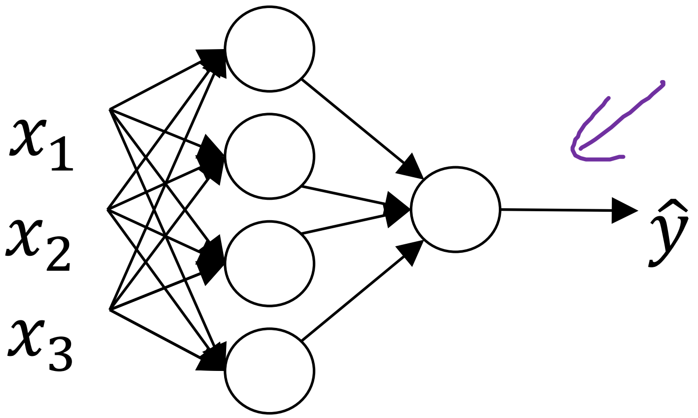
$$W^{[1]} \underbrace{\begin{bmatrix} | & | & | & \\ x^{(1)} & x^{(2)} & x^{(3)} & \cdots \\ | & | & | & \end{bmatrix}}_{X} = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} = \begin{bmatrix} | & | & | & \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} & \cdots \\ | & | & | & \end{bmatrix} = Z^{[1]}$$

$$+b^{[1]} \quad +b^{[1]} \quad +b^{[1]}$$

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

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

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

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

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

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

deeplearning.ai

One hidden layer Neural Network

Explanation for vectorized implementation

Justification for vectorized implementation

$$z^{[1](1)} = W^{[1]}x^{(1)} + \cancel{b^{[1]}} \to 0, \quad z^{[1](2)} = W^{[1]}x^{(2)} + \cancel{b^{[1]}} \to 0, \quad z^{[1](3)} = W^{[1]}x^{(3)} + \cancel{b^{[1]}} \to 0$$

$$W^{[1]} = \begin{bmatrix} \text{———} \\ \text{———} \\ \text{———} \\ \text{———} \end{bmatrix} \qquad W^{[1]}x^{(1)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} \qquad W^{[1]}x^{(2)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix} \qquad W^{[1]}x^{(3)} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

$$W^{[1]} \underbrace{\begin{bmatrix} | & | & | & \\ x^{(1)} & x^{(2)} & x^{(3)} & \cdots \\ | & | & | & \end{bmatrix}}_{X} = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} = \begin{bmatrix} | & | & | & \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} & \cdots \\ | & | & | & \end{bmatrix} = Z^{[1]}$$

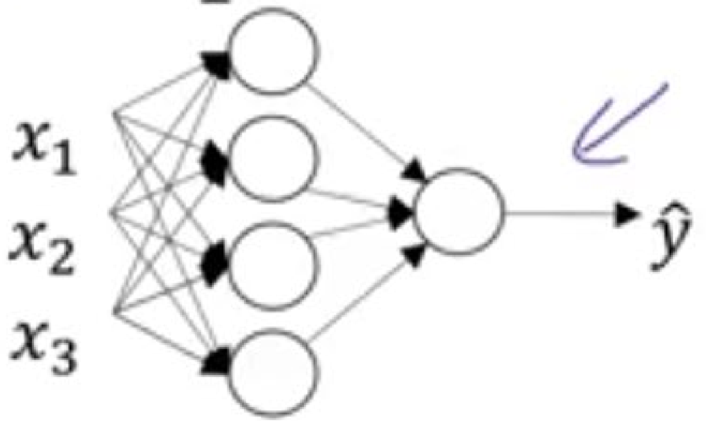
$$+b^{[1]} \quad +b^{[1]} \quad +b^{[1]}$$

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

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

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
```

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

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

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

Andrew Ng

One hidden layer Neural Network

Activation functions

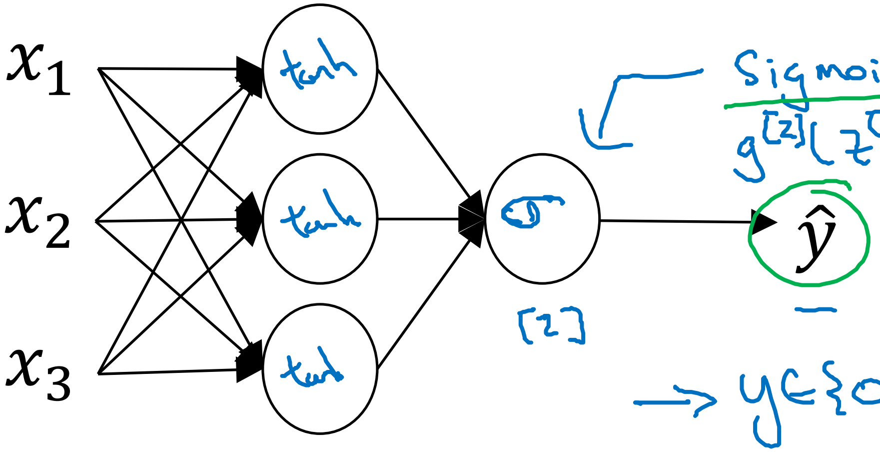
deeplearning.ai

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 1

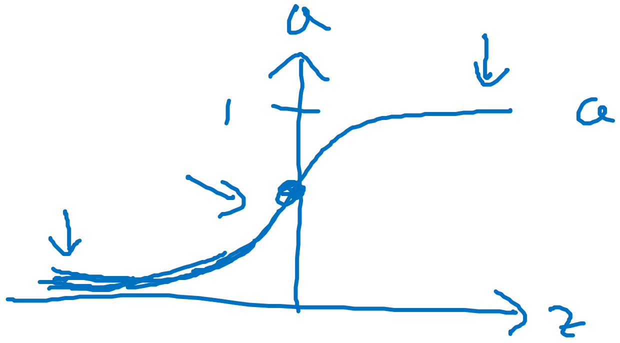
Given x:

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

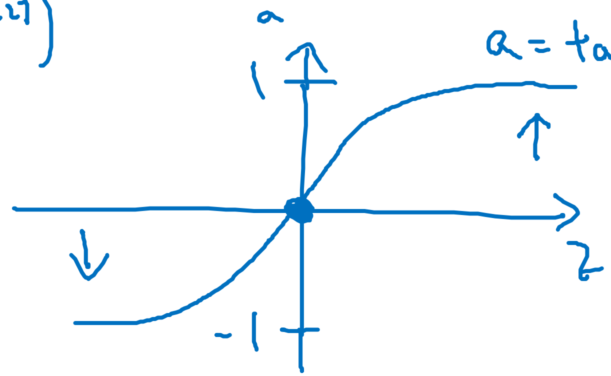
$$\rightarrow a^{[1]} = \cancel{\sigma(z^{[1]})}\ g^{[1]}(z^{[1]})$$

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

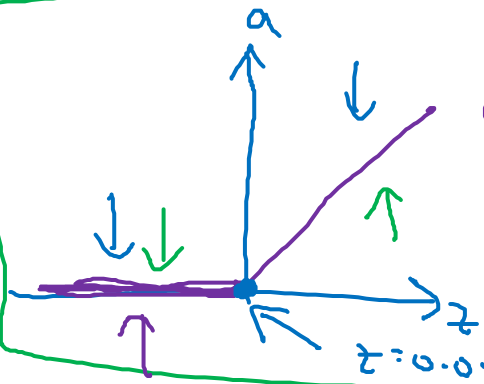
$$\rightarrow a^{[2]} = \cancel{\sigma(z^{[2]})}\ g^{[2]}(z^{[2]})$$



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



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



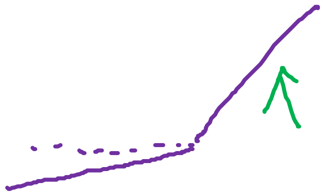
$z > 0$

$a = \max(0, z)$

ReLU

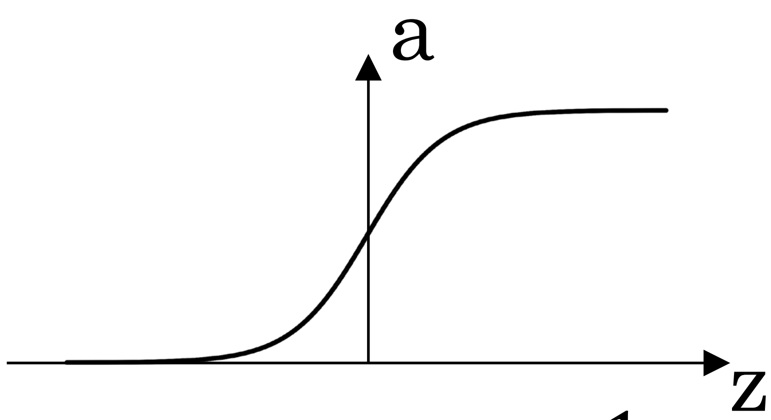
Rectified Linear Unit

$z = 0.0000000000000$

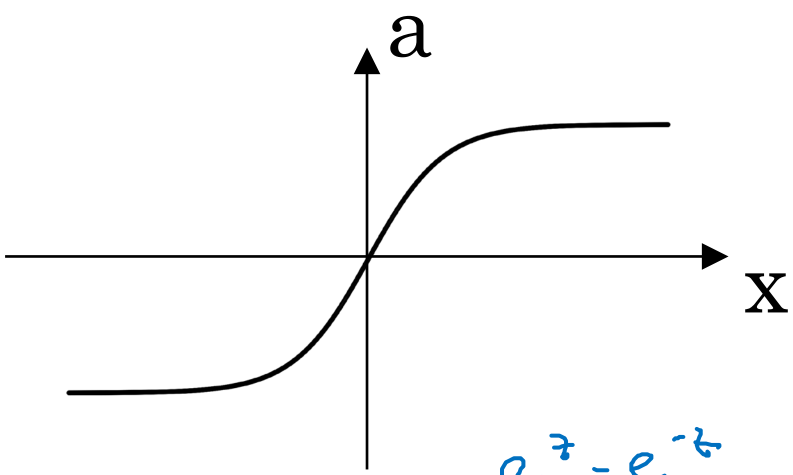


Leaky ReLU

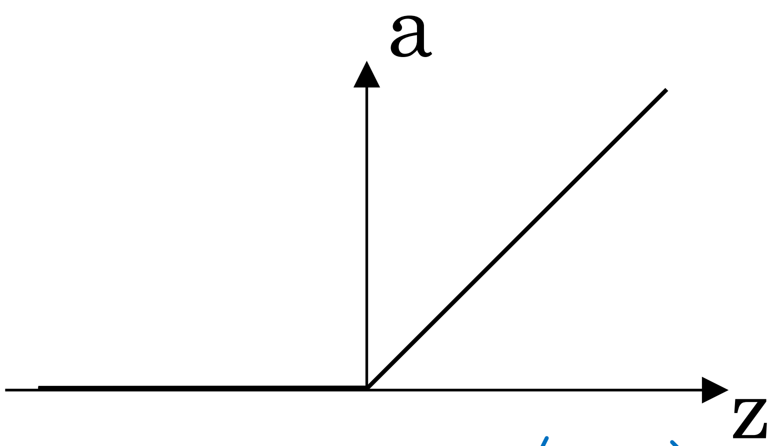
Pros and cons of activation functions



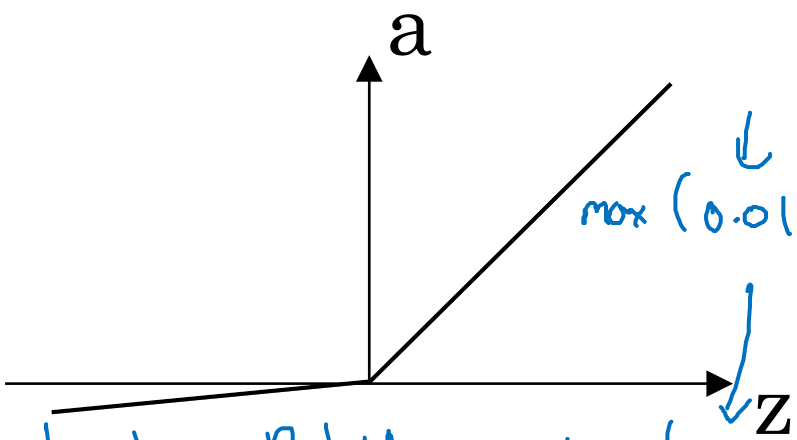
sigmoid: $a = \dfrac{1}{1 + e^{-z}}$



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



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



$\max(0.01z, z)$

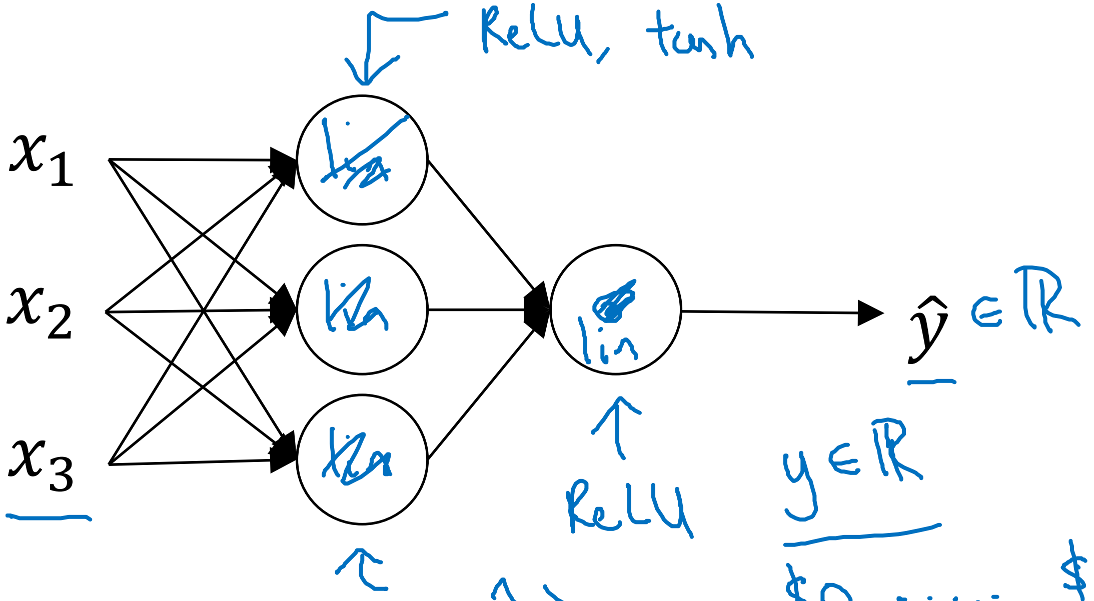
Leaky ReLU $a = \max(0.01z, z)$

deeplearning.ai

One hidden layer Neural Network

Why do you need non-linear activation functions?

Activation function



$y \in \mathbb{R}$

\$0 ..... \$1,000,000s

$\hat{y} \geq 0$.

Given x:

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

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

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

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

$g(z) = z$

"linear activation function"

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

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

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

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

$$= W'x + b'$$

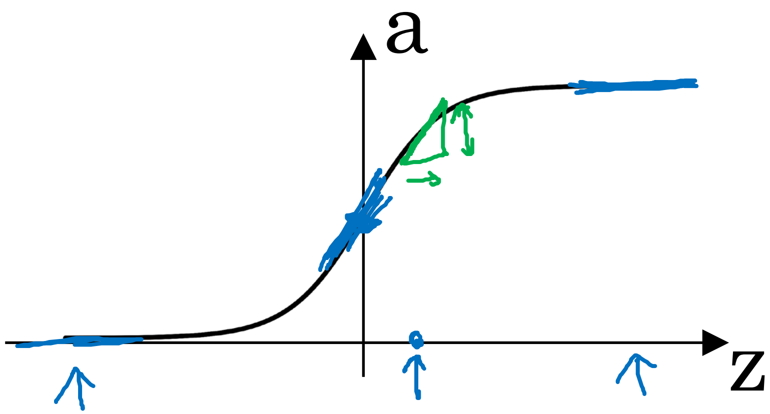
$g(z) = z$

deeplearning.ai

One hidden layer Neural Network

Derivatives of activation functions

Sigmoid activation function



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

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

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

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

$$= \underbrace{g(z)\,(1-g(z))}$$

$$= \boxed{a\,(1-a)}$$

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

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

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

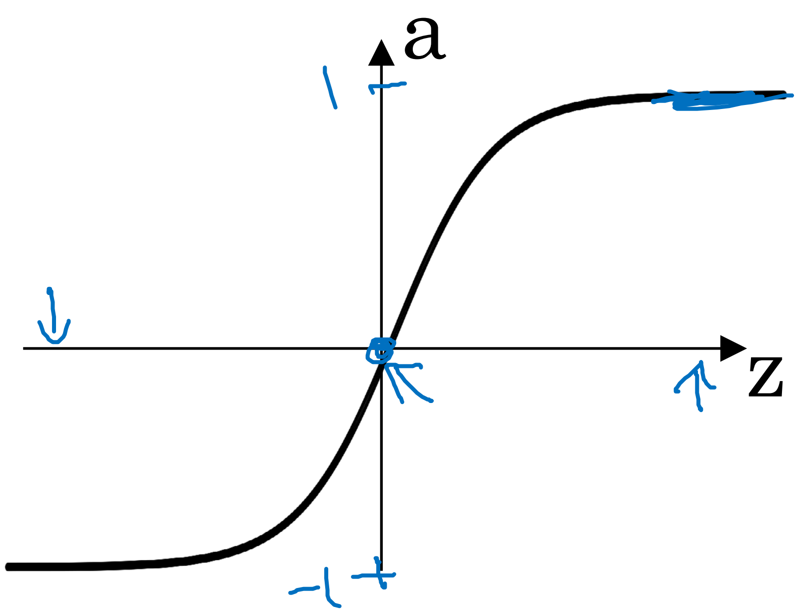
$z = -10$ $g(z) \approx 0$

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

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

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

Tanh activation function



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

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

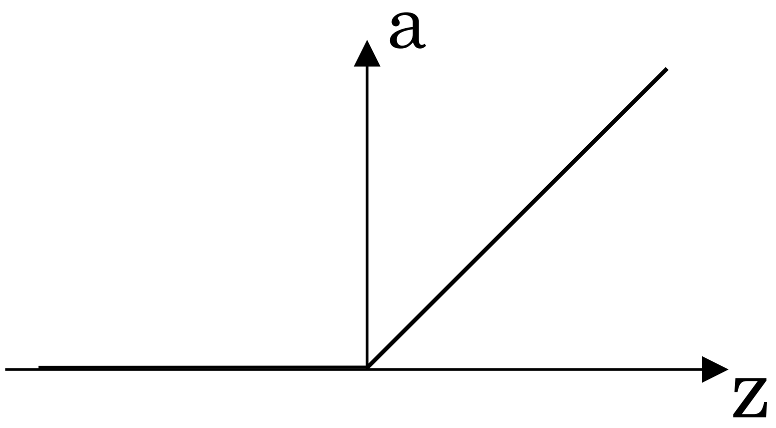
$z = 10$ $\tanh(z) \approx 1$
$g'(z) \approx 0$

$z = -10$ $\tanh(z) \approx -1$
$g'(z) \approx 0$

$z = 0$ $\tanh(z) = 0$
$g'(z) = 1$

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

ReLU and Leaky ReLU

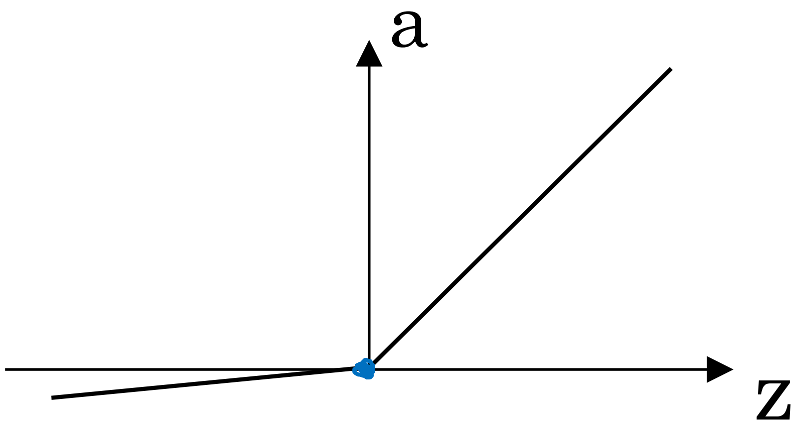


ReLU

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

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

$$z = 0.00000\ldots\ldots 0$$



Leaky ReLU

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

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

deeplearning.ai

One hidden layer Neural Network

Gradient descent for neural networks

Gradient descent for neural networks

Parameters: $\underset{(n^{[1]}, n^{[0]})}{W^{[1]}}, \underset{(n^{[1]}, 1)}{b^{[1]}}, \underset{(n^{[2]}, n^{[1]})}{W^{[2]}}, \underset{(n^{[2]}, 1)}{b^{[2]}}$ $\quad n_x = n^{[0]}, n^{[1]}, \underline{n^{[2]} = 1}$

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

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

Gradient descent:

→ Repeat {

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

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

$W^{[1]} := W^{[1]} - \alpha \, dW^{[1]}$

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

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

}

Formulas for computing derivatives

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:

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

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

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

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

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

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

$$dZ^{[1]} = \underbrace{W^{[2]T} dZ^{[2]}}_{(n^{[1]}, m)} * \underbrace{g^{[1]\prime}(Z^{[1]})}_{(n^{[1]}, m)}$$

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

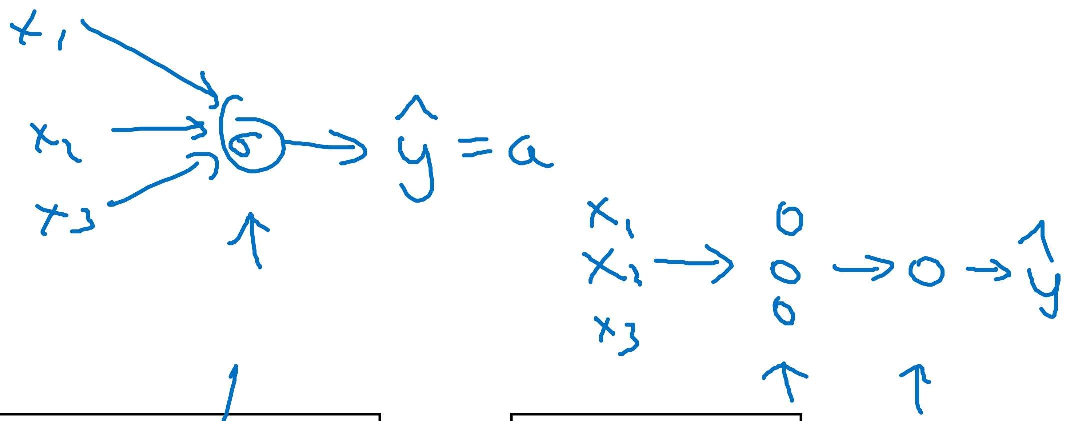
deeplearning.ai

One hidden layer Neural Network

Backpropagation intuition (Optional)

Computing gradients

Logistic regression



$$
\begin{array}{l}
x \\
w \\
b
\end{array}
\rightarrow \boxed{z = w^T x + b} \rightarrow \boxed{a = \sigma(z)} \rightarrow \boxed{\mathcal{L}(a, y)}
$$

$dz = a - y$

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

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

$dw = dz \cdot x$

$db = dz$

$\boxed{dz = da \cdot g'(z)}$

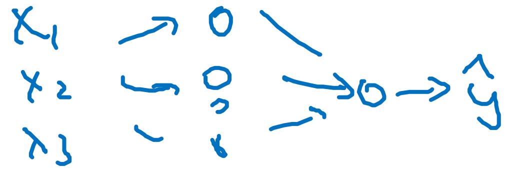
$g(z) = \sigma(z)$

$\boxed{\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial a} \cdot \boxed{\frac{da}{dz}}}$

"dz" = "da"

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

Neural network gradients



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

x

$W^{[2]}$ $\quad dW^{[2]}$

$b^{[2]}$ $\quad db^{[2]}$

$W^{[1]}$, $b^{[1]}$ → $\boxed{z^{[1]} = W^{[1]}x + b^{[1]}}$ → $\boxed{a^{[1]} = \sigma(z^{[1]})}$ → $\boxed{z^{[2]} = W^{[2]}x + b^{[2]}}$ → $\boxed{a^{[2]} = \sigma(z^{[2]})}$ → $\boxed{\mathcal{L}(a^{[2]}, y)}$

$dW^{[1]}$

$db^{[1]}$

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

$* \, g^{[1]\prime}(z^{[1]})$

element wise product

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

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

$W^{[1]}: [\;\_\_\_\_\;]$

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

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

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

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

dfoo $\quad dW^{[2]}$ $\quad db^{[1]} = dz^{[1]}$

$a^{[0]T}$

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

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

Andrew Ng

Summary of gradient descent

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

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

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

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

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

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

Vectorized Implementation:

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

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

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

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

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

Summary of gradient descent

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

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

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

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

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

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

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

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

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

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

elementwise product

$$\underbrace{dZ^{[1]}}_{(n^{[1]},m)} = \underbrace{W^{[2]T} dZ^{[2]}}_{(n^{[1]},m)} * \underbrace{g^{[1]\prime}(Z^{[1]})}_{(n^{[1]},m)}$$

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

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

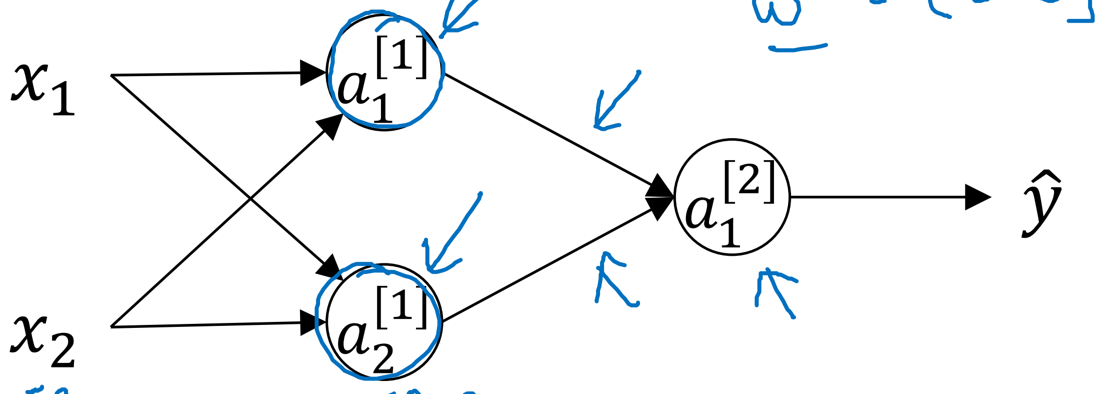
One hidden layer Neural Network

Random Initialization

deeplearning.ai

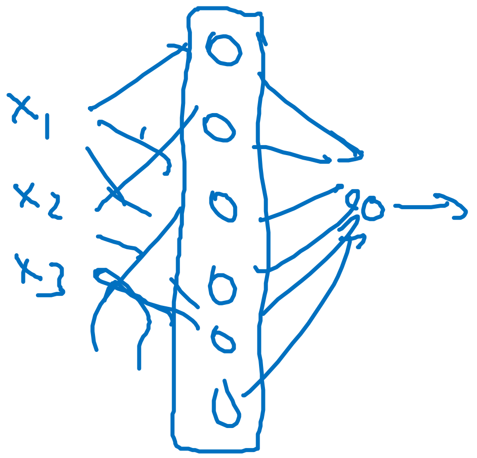
What happens if you initialize weights to zero?

$W^{[2]} = [0 \quad 0]$



$n^{[0]} = 2$ $\qquad$ $n^{[1]} = 2$

Symmetric



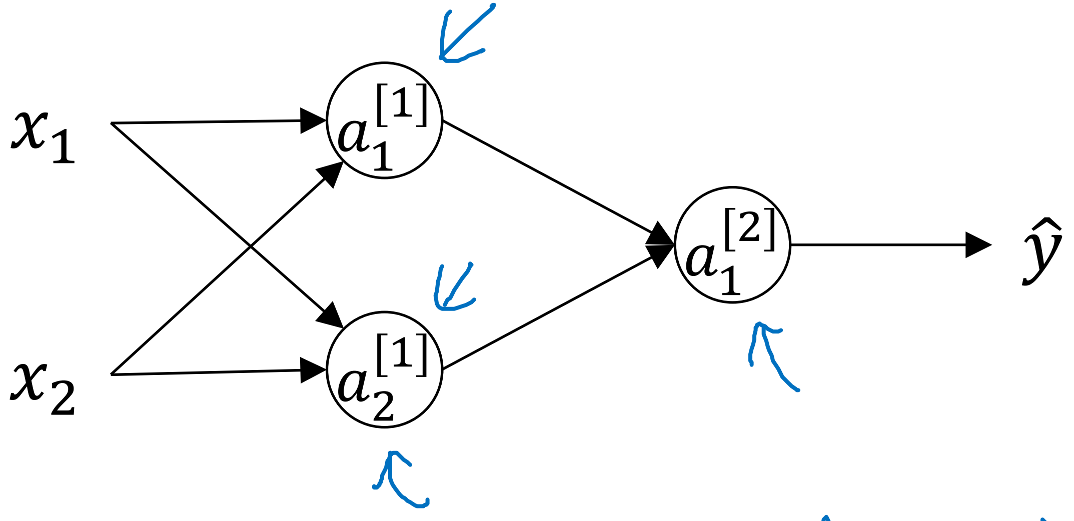
$W^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$ $\qquad$ $b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

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

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

$dW = \begin{bmatrix} u & v \\ u & v \end{bmatrix}$ $\qquad$ $W^{[1]} = W^{[1]} - \alpha \, dW$

Random initialization



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

100?

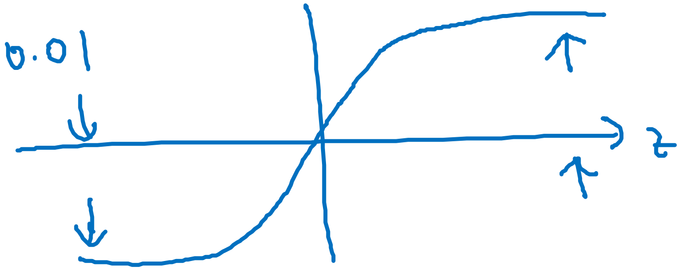
$b^{[1]} =$ np.zeros((2,1))

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

$b^{[2]} = 0$

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

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

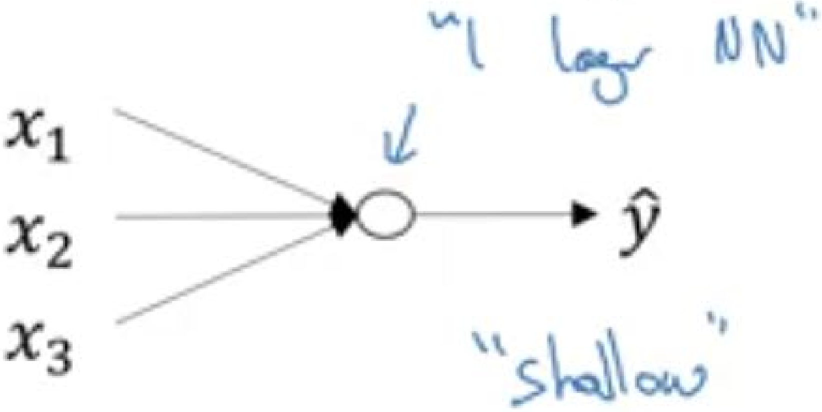


deeplearning.ai

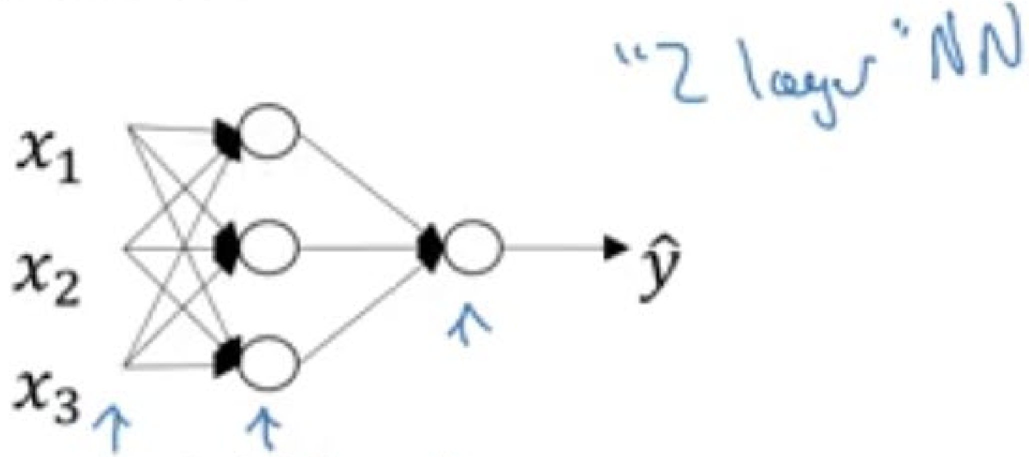
Deep Neural Networks

Deep L-layer Neural network

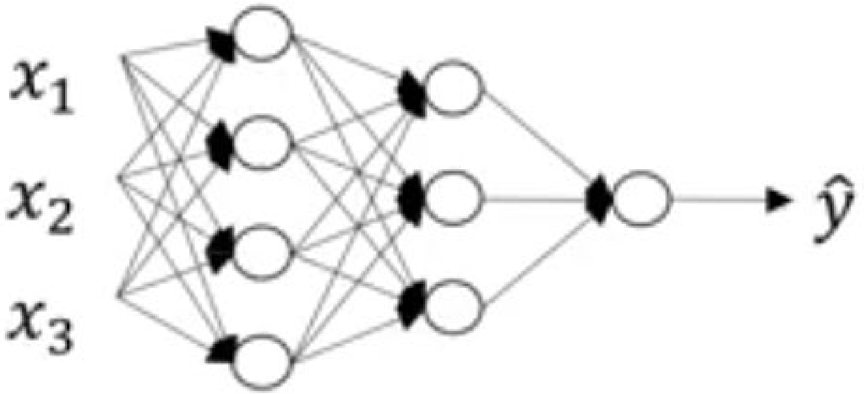
What is a deep neural network?



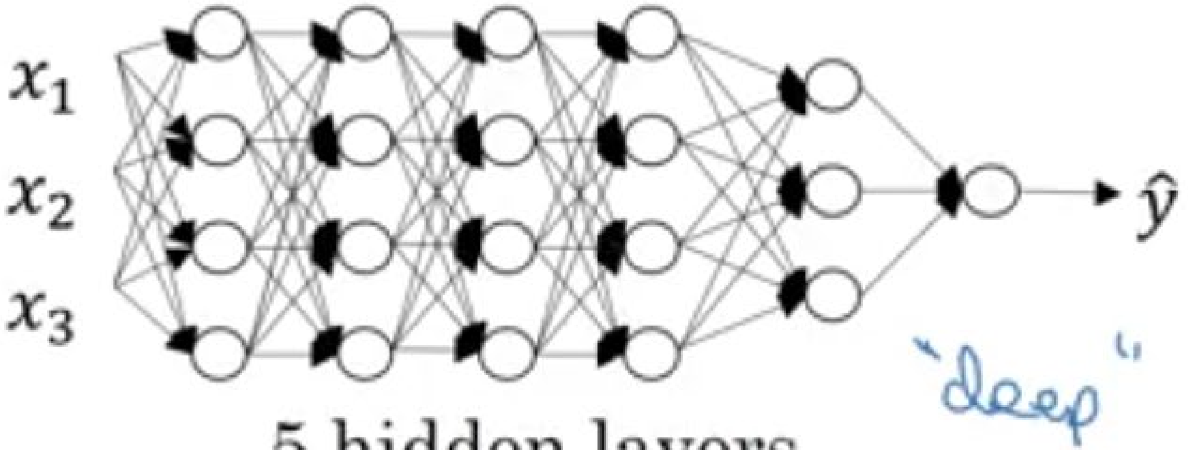
logistic regression



1 hidden layer



2 hidden layers



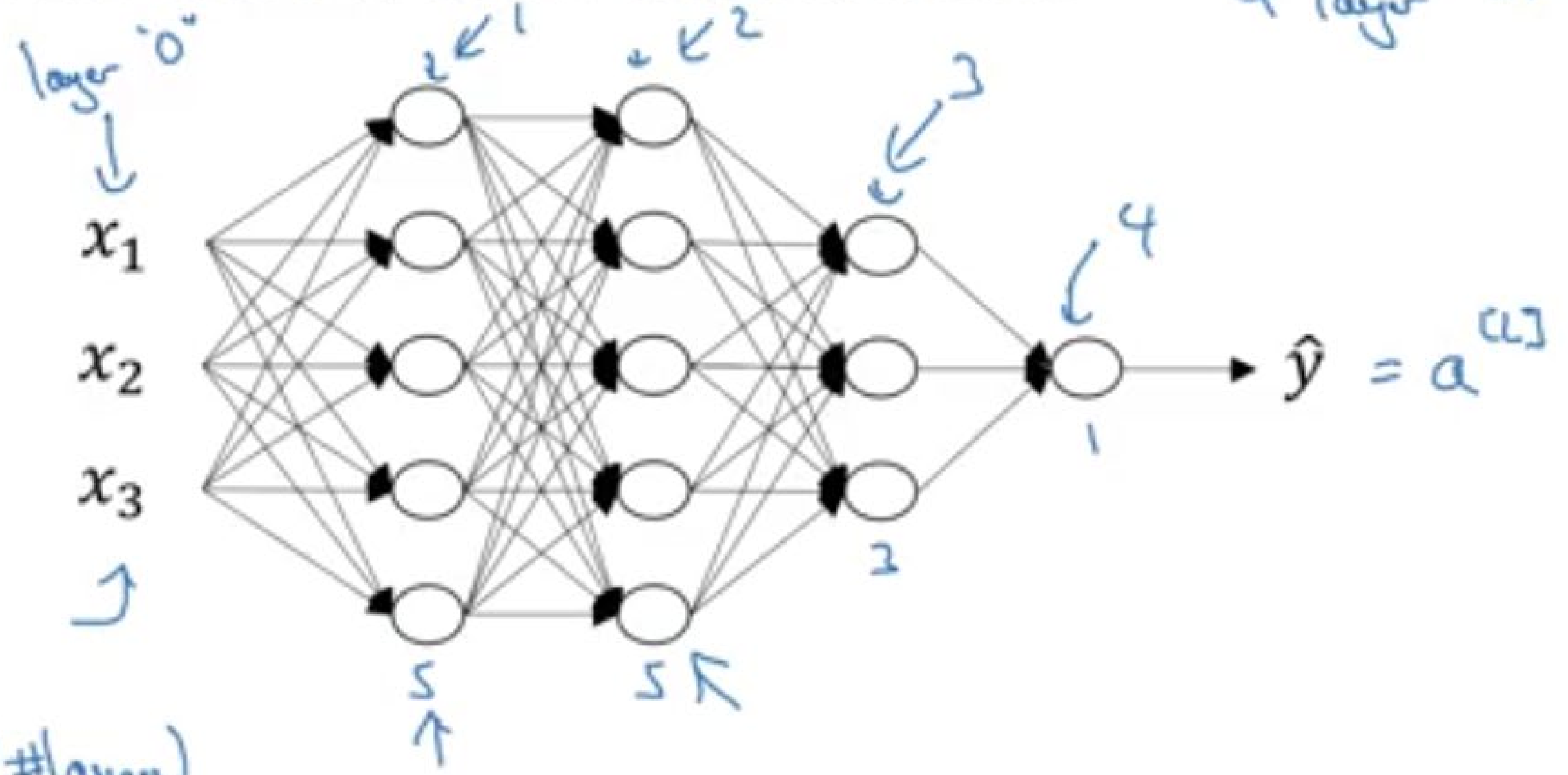
5 hidden layers

Andrew Ng

Deep neural network notation

4 layer NN

$X = a^{[0]}$



$L = 4$ (#layers)

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

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

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

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

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

$b^{[l]}$

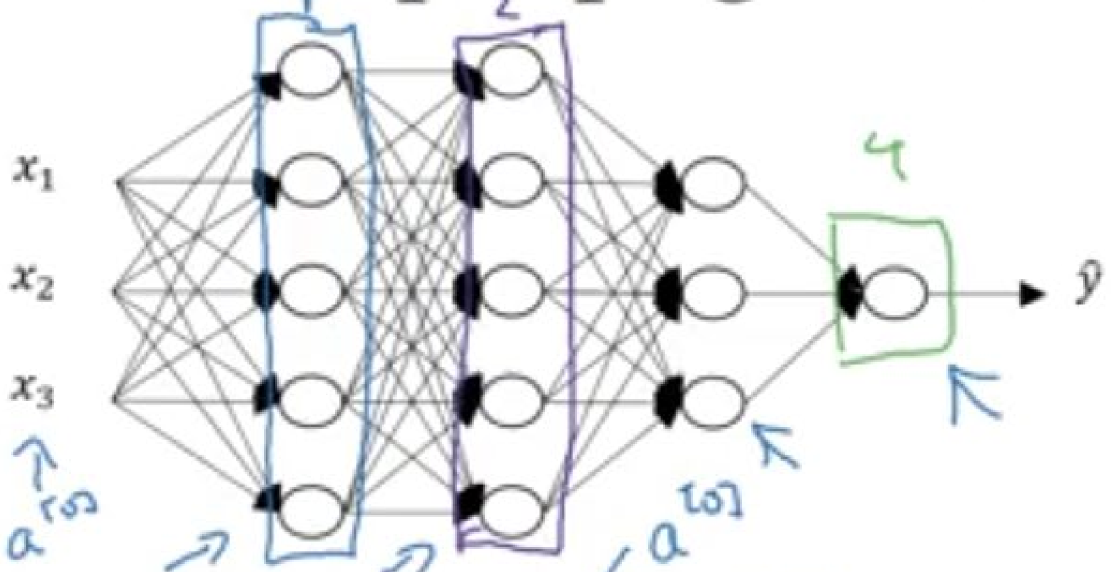
Andrew Ng

deeplearning.ai

Deep Neural Networks

Forward Propagation in a Deep Network

Forward propagation in a deep network



$a^{[0]}$

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

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

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

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

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

$\left[z^{[1](1)}, z^{[1](2)}, \ldots, z^{[1](m)} \right]$

$A^{[0]} = X$

$$\boxed{\begin{aligned} Z^{[l]} &= W^{[l]} A^{[l-1]} + b^{[l]} \\ A^{[l]} &= g^{[l]}(Z^{[l]}) \end{aligned}}$$

Vectorized:

$X = A^{[0]}$

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

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

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

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

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

for $l = 1..4$

Andrew Ng

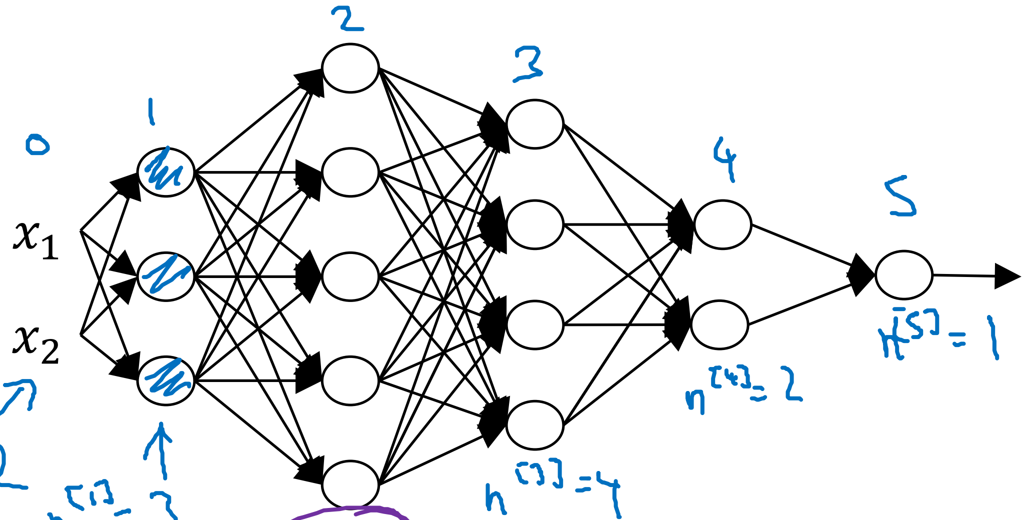
deeplearning.ai

Deep Neural Networks

Getting your matrix dimensions right

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

$L = 5$



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

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

$a^{[l]}$

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

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

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

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

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

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

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

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

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

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

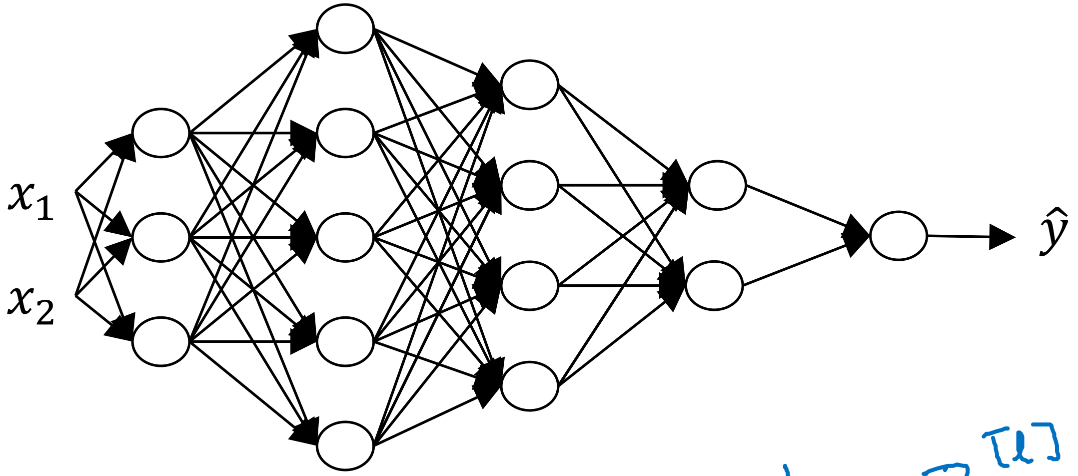
$\rightarrow W^{[l]}: (n^{[l]}, n^{[l-1]})$

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

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

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

Vectorized implementation



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

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

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

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

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

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

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

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

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

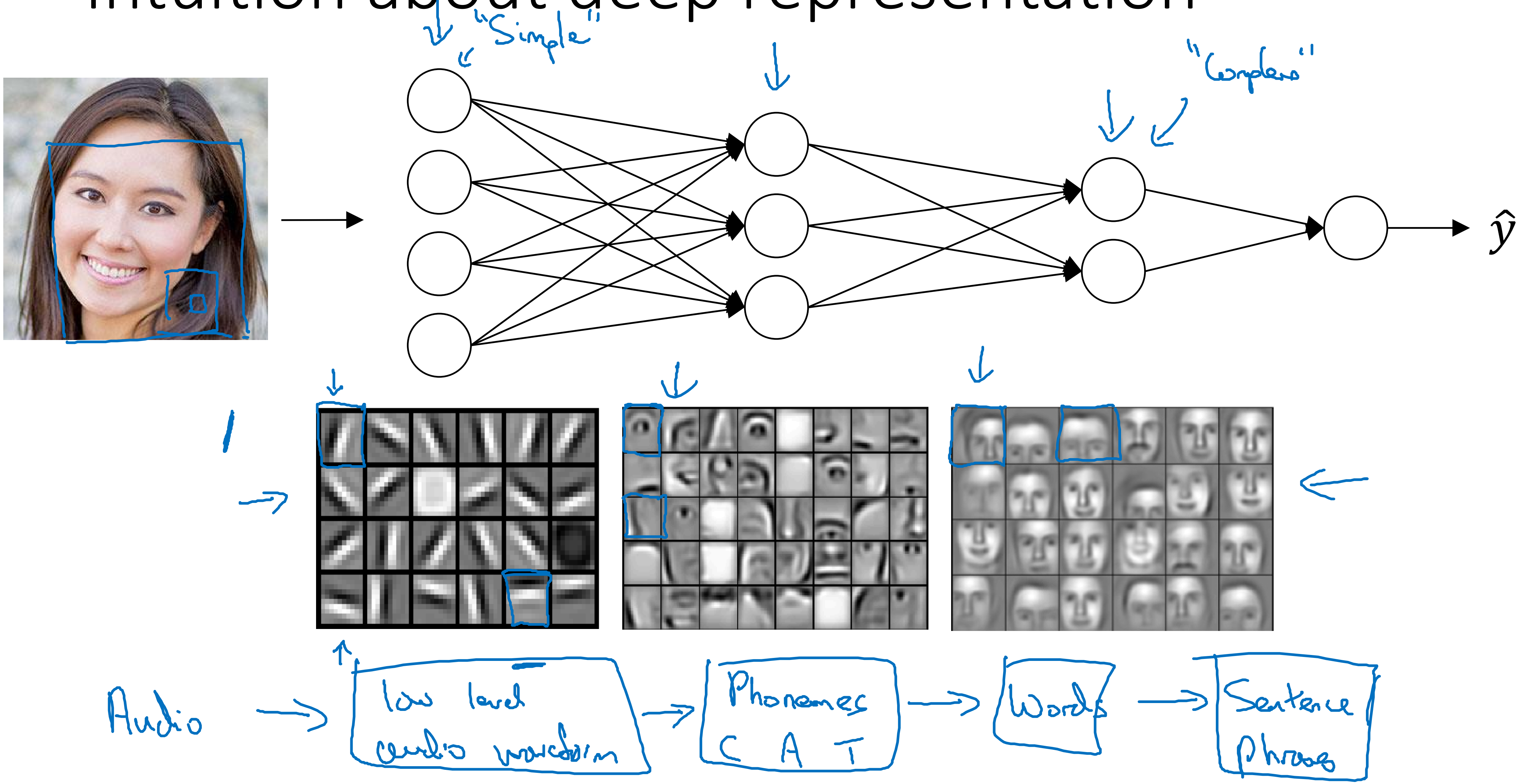
$dZ^{[l]}, dA^{[l]} : (n^{[l]},m)$

deeplearning.ai

Deep Neural Networks

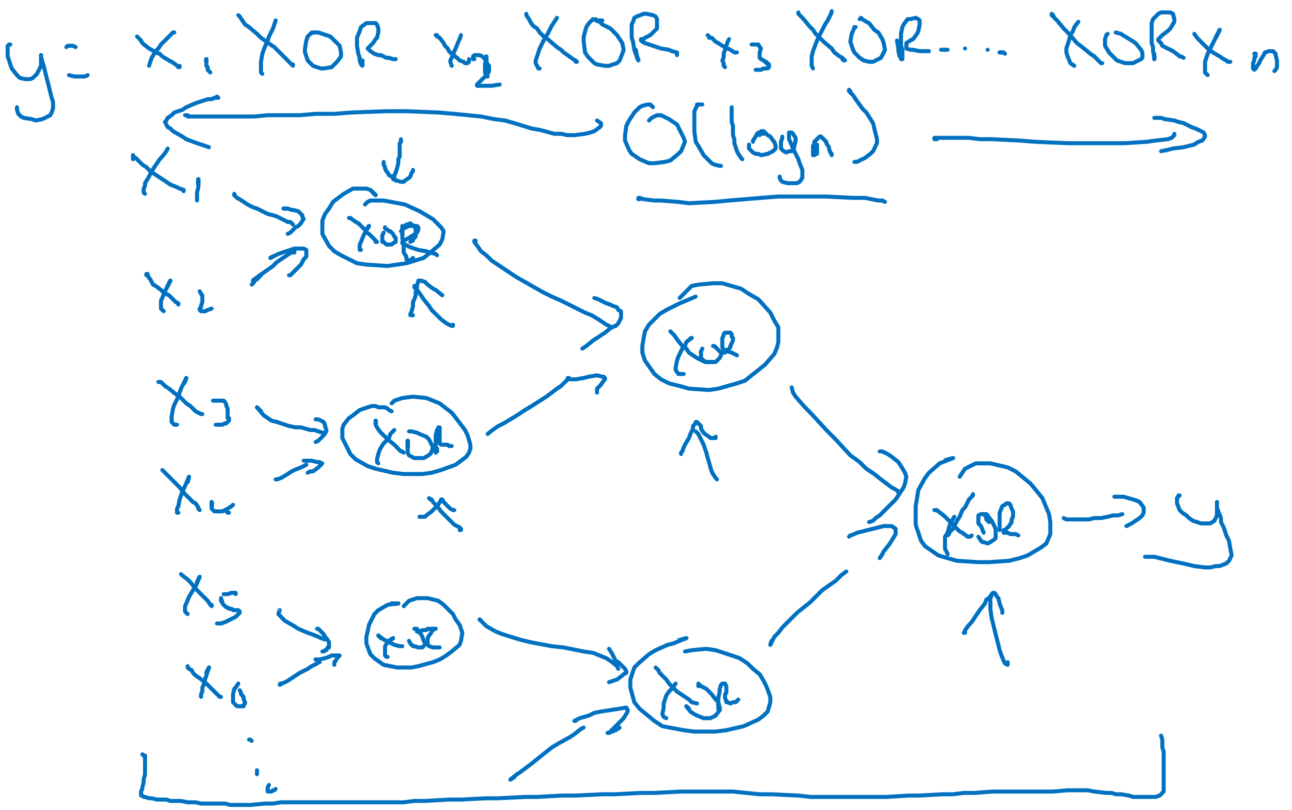
Why deep representations?

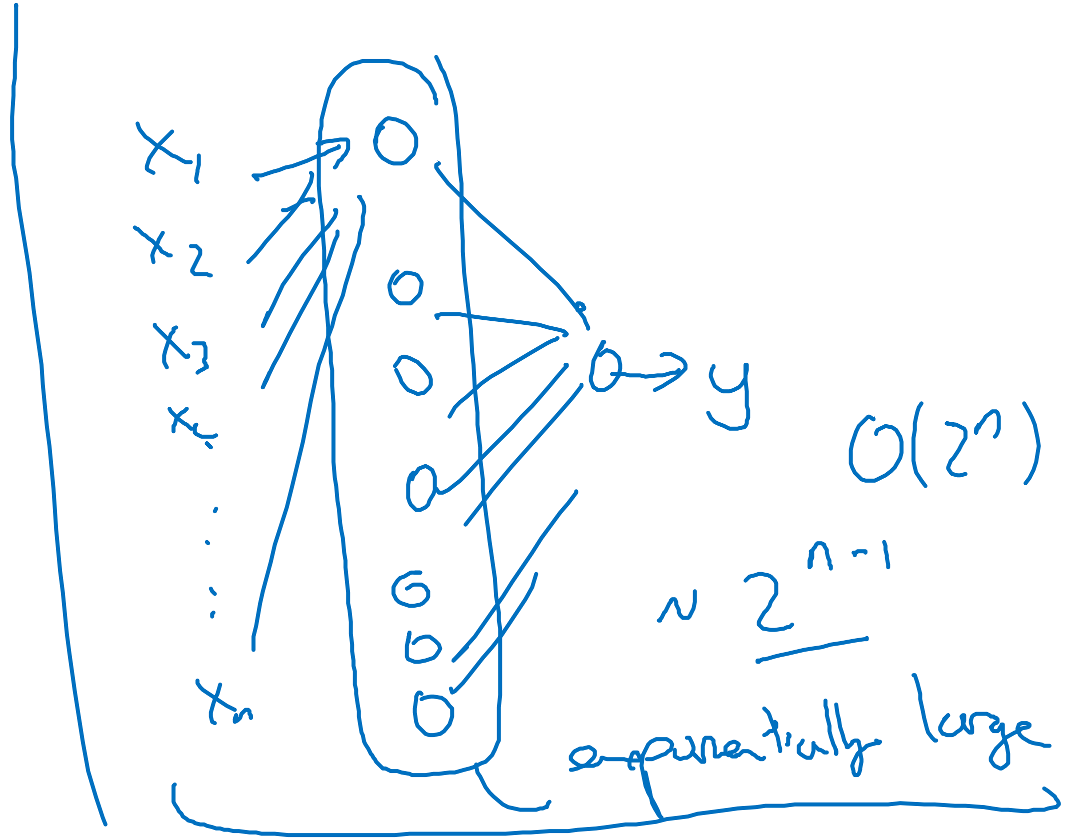
Intuition about deep representation



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.



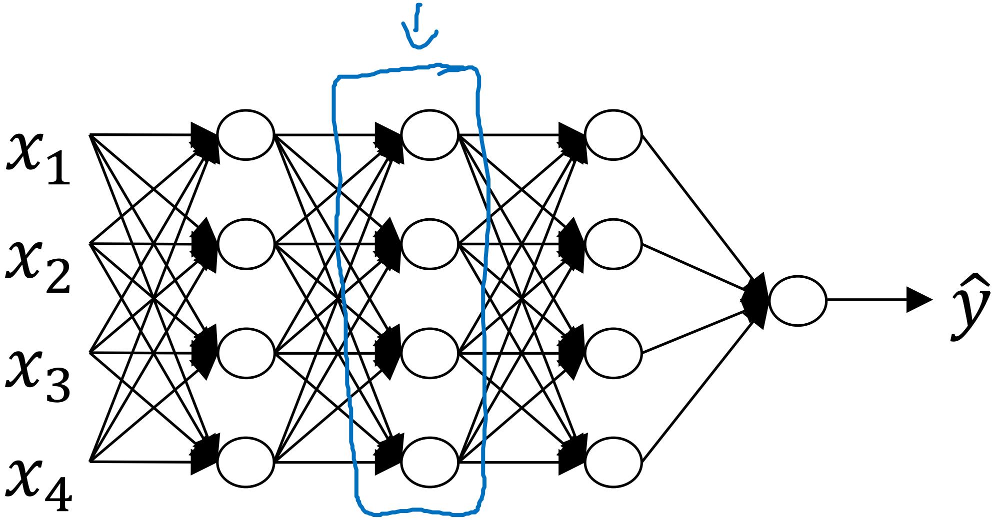


deeplearning.ai

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 $a^{[l]}$

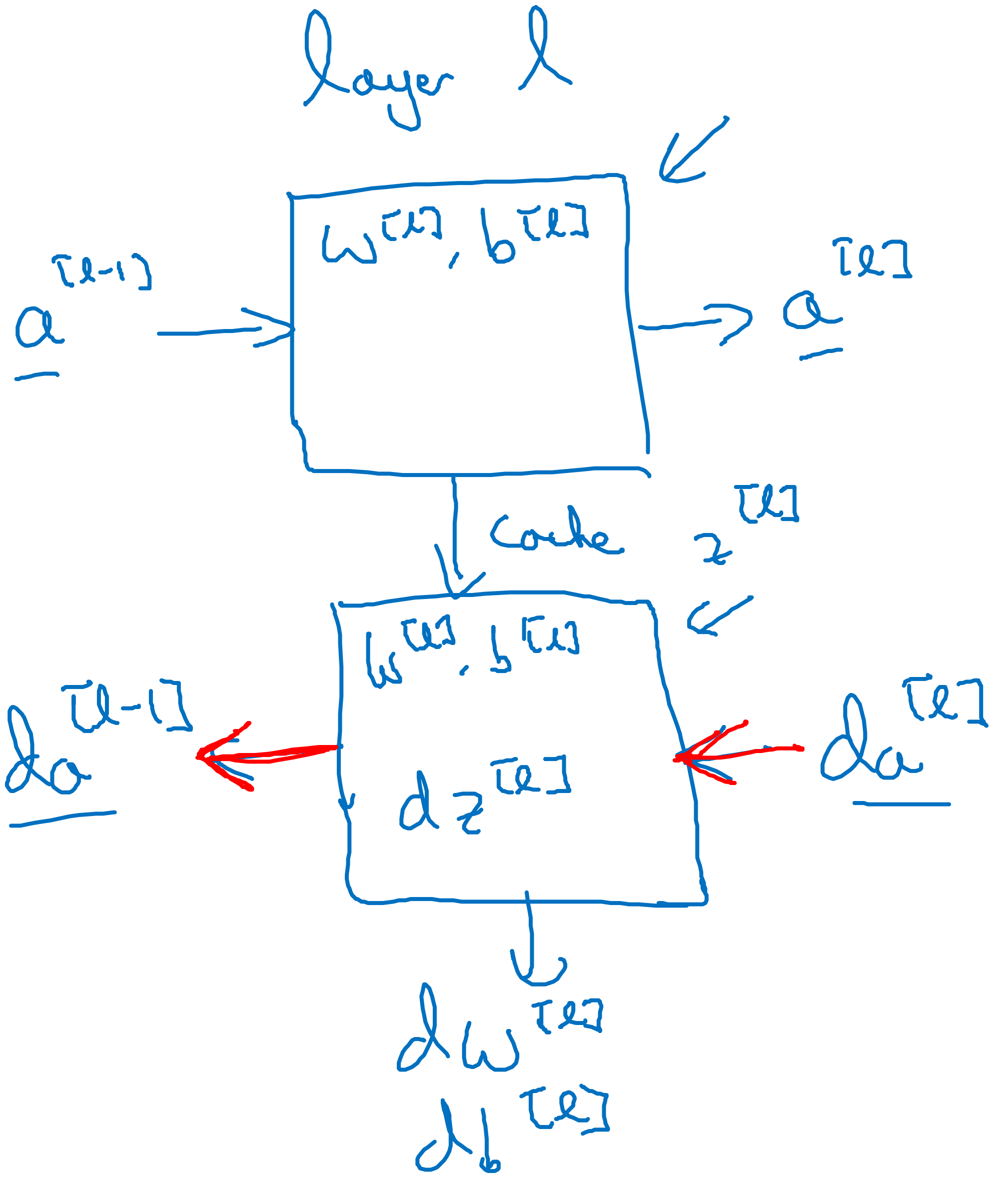
$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$ cache $z^{[l]}$

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

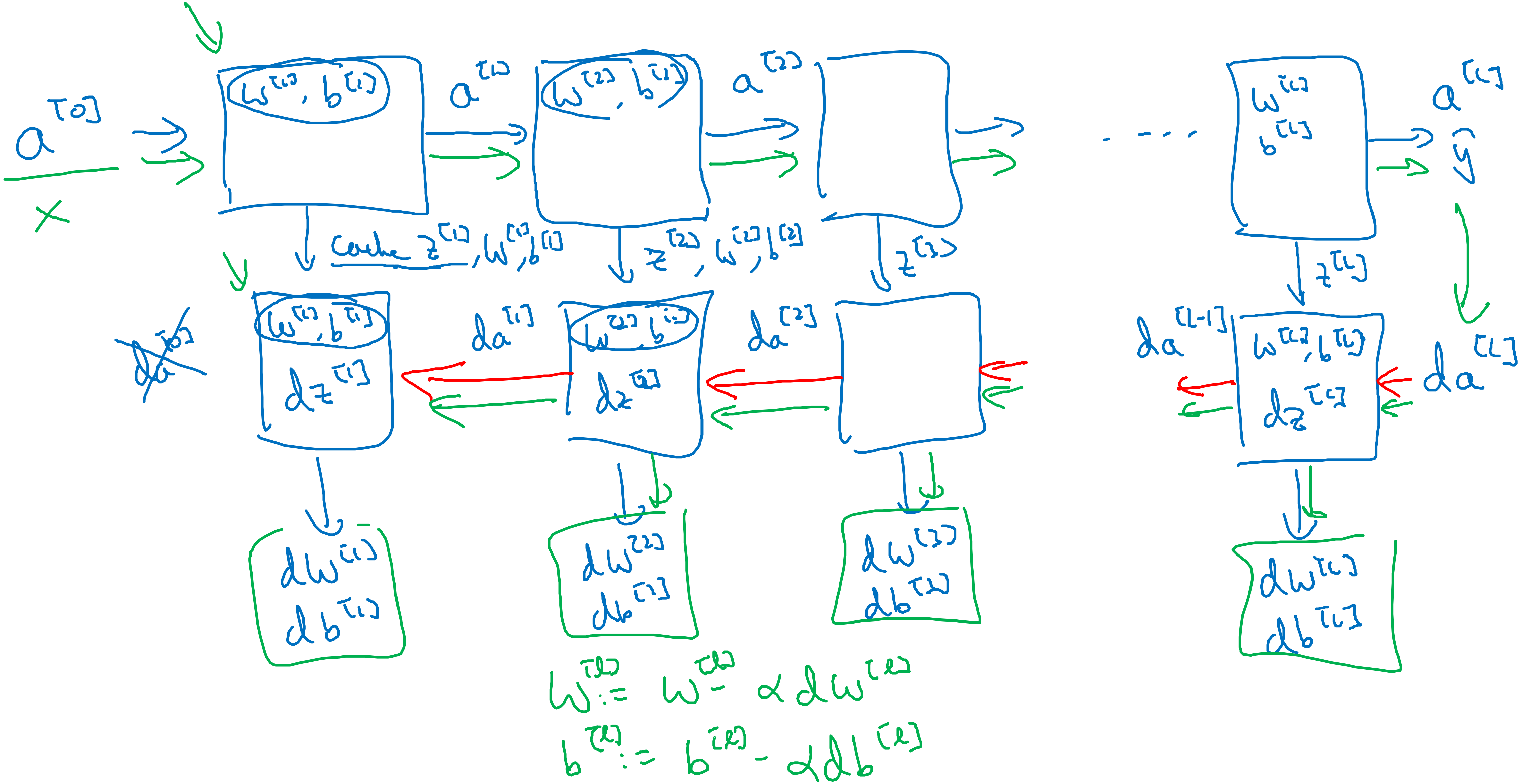
→ Backward: Input $da^{[l]}$, output $da^{[l-1]}$

cache($z^{[l]}$) $dW^{[l]}$

$db^{[l]}$



Forward and backward functions



deeplearning.ai

Deep Neural Networks

Forward and backward propagation

Forward propagation for layer l

- Input $a^{[l-1]}$ — $W^{[l]}, b^{[l]}$
- Output $a^{[l]}$, cache ($\underline{z^{[l]}}$)

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

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

Vectorized:

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

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

$a^{[0]}$

$A^{[0]}$

$X = A^{[0]} \to \square \to \square \to \square \to$

Backward propagation for layer l

→ Input $da^{[l]}$

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

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

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

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

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

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

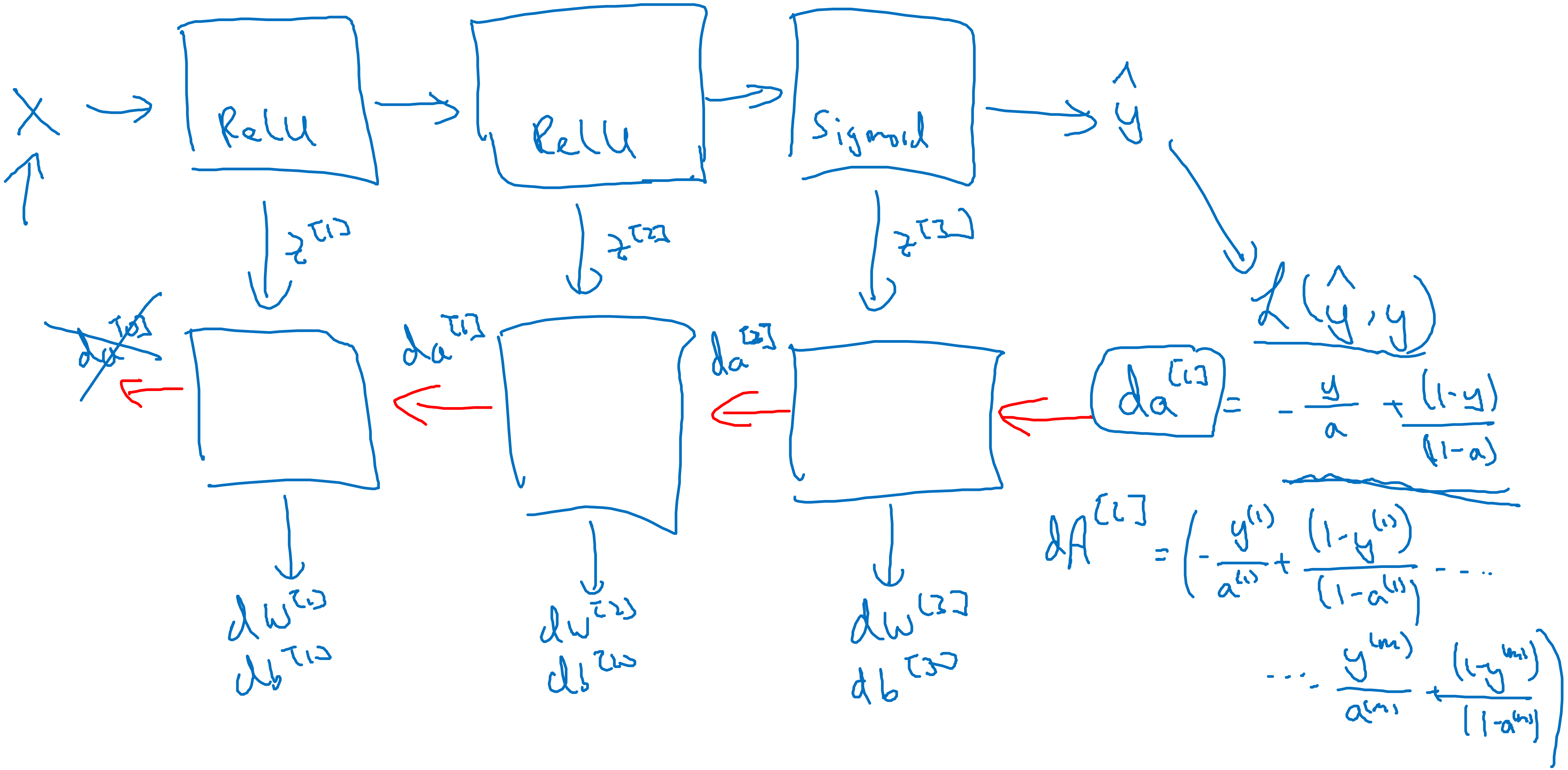
$$dZ^{[l]} = dA^{[l]} * g^{[l]\prime}(Z^{[l]})$$

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

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

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

Summary


X
ReLU
ReLU
Sigmoid
$\hat{y}$
$z^{[1]}$
$z^{[2]}$
$z^{[3]}$
$\mathcal{L}(\hat{y}, y)$
$\cancel{dx}$ $da^{[0]}$
$da^{[1]}$
$da^{[2]}$
$da^{[l]} = -\frac{y}{a} + \frac{(1-y)}{(1-a)}$
$dA^{[l]} = \left(-\frac{y^{(1)}}{a^{(1)}} + \frac{(1-y^{(1)})}{(1-a^{(1)})} \cdots \cdots -\frac{y^{(m)}}{a^{(m)}} + \frac{(1-y^{(m)})}{(1-a^{(m)})}\right)$
$dW^{[1]}$
$db^{[1]}$
$dW^{[2]}$
$db^{[2]}$
$dW^{[3]}$
$db^{[3]}$

deeplearning.ai

Deep Neural Networks

Forward and backward propagation

Forward propagation for layer l

- Input $a^{[l-1]}$
- Output $a^{[l]}$, cache ($z^{[l]}$)

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

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

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

Vectorized:

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

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

$a^{[0]}$

$A^{[0]}$

$X = A^{[0]} \rightarrow \square \rightarrow \square \rightarrow \square \rightarrow$

Andrew Ng

Backward propagation for layer l

→ Input $da^{[l]}$

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

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

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

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

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

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

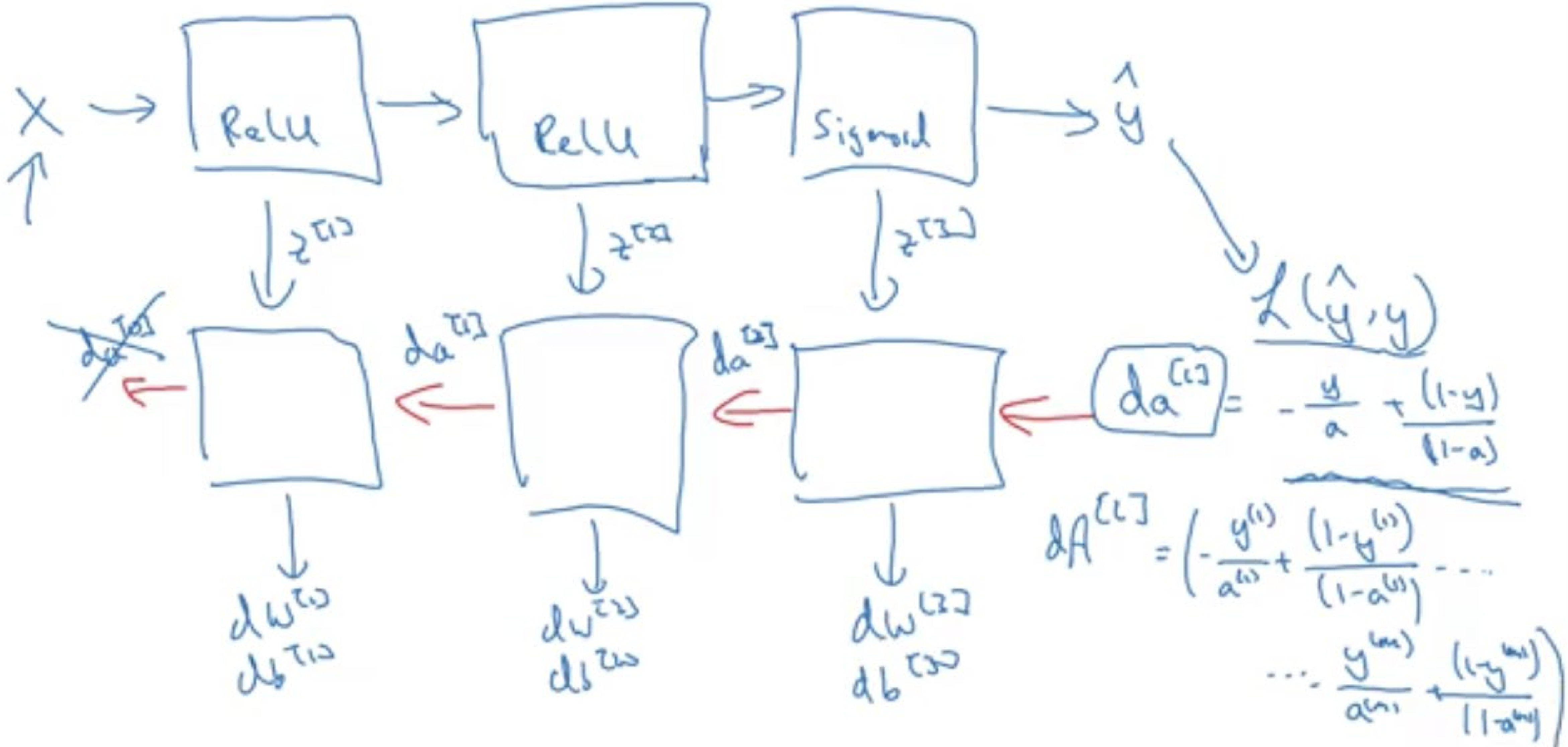
$$dZ^{[l]} = dA^{[l]} * g^{[l]\prime}(Z^{[l]})$$

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

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

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

Summary



Andrew Ng

deeplearning.ai

Deep Neural Networks

Parameters vs Hyperparameters

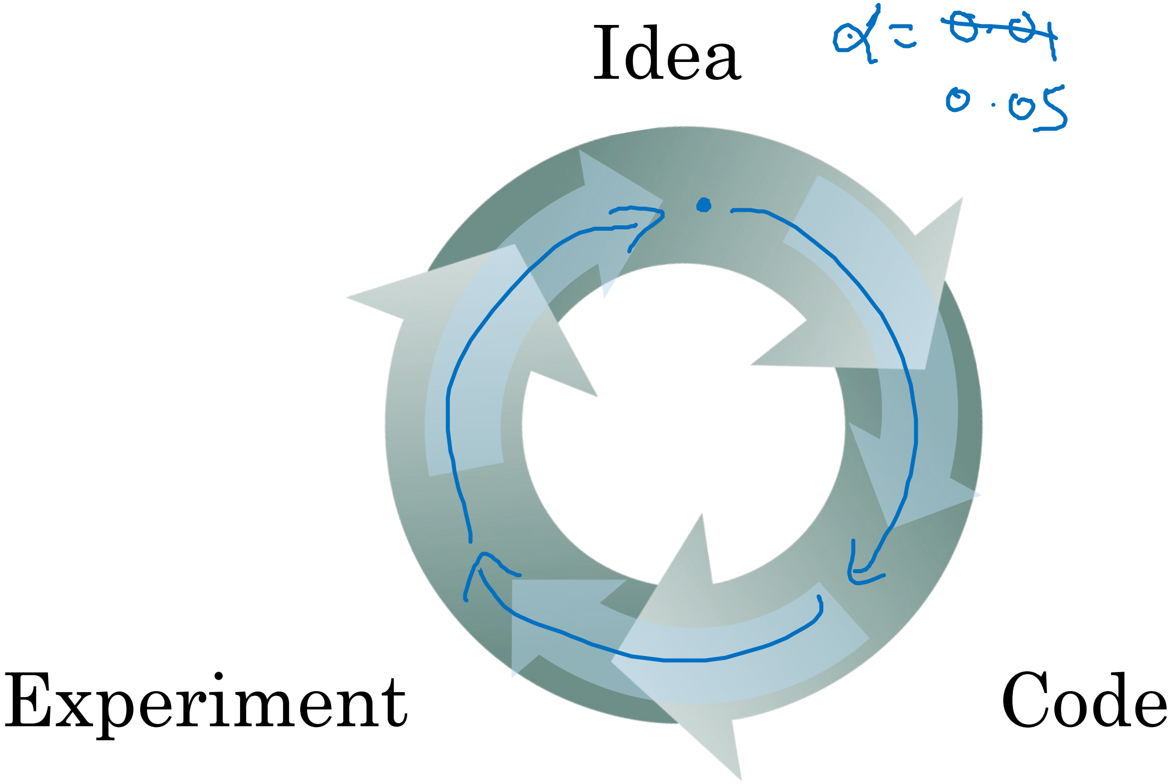
What are hyperparameters?

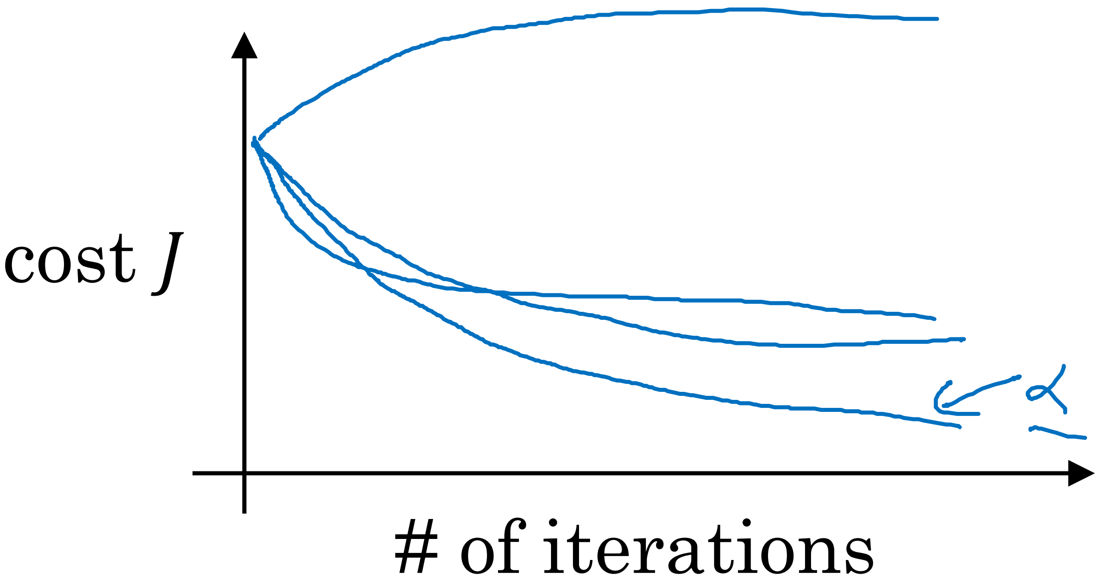
Parameters: $\underline{W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]}}$ ...

$$
\text{Hyperparameters:} \left.\begin{array}{l}
\text{learning rate } \alpha \\
\text{\#iterations} \\
\text{\#hidden layers } L \\
\text{\#hidden units } n^{[1]}, n^{[2]}, \ldots \\
\text{choice of activation function}
\end{array}\right\}
$$

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

Applied deep learning is a very empirical process





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

deeplearning.ai

Deep Neural Networks

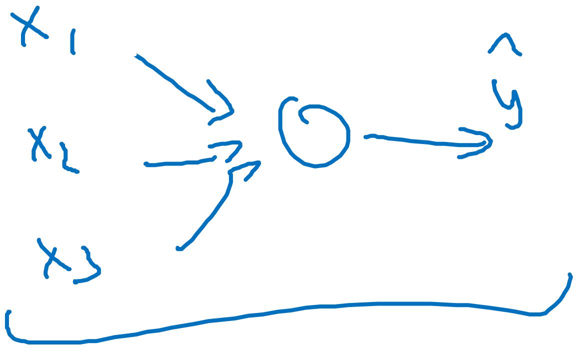
What does this have to do with the brain?

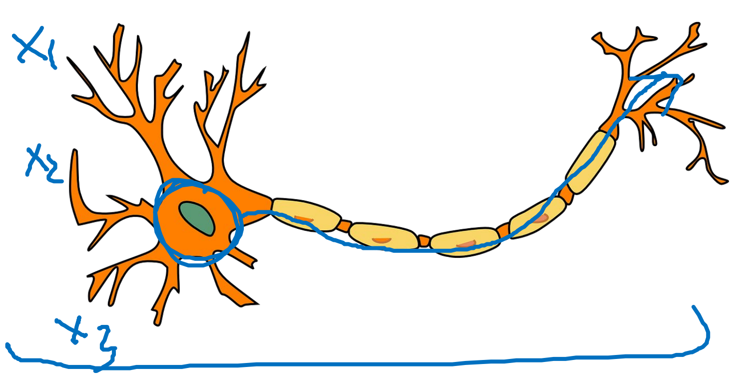
Forward and backward propagation

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

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

"It's like the brain"





$X \longrightarrow y$