



deeplearning.ai

# Introduction to Deep Learning

---

## Welcome



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

# What you'll learn



Courses in this sequence (Specialization):

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

## What is neural network?

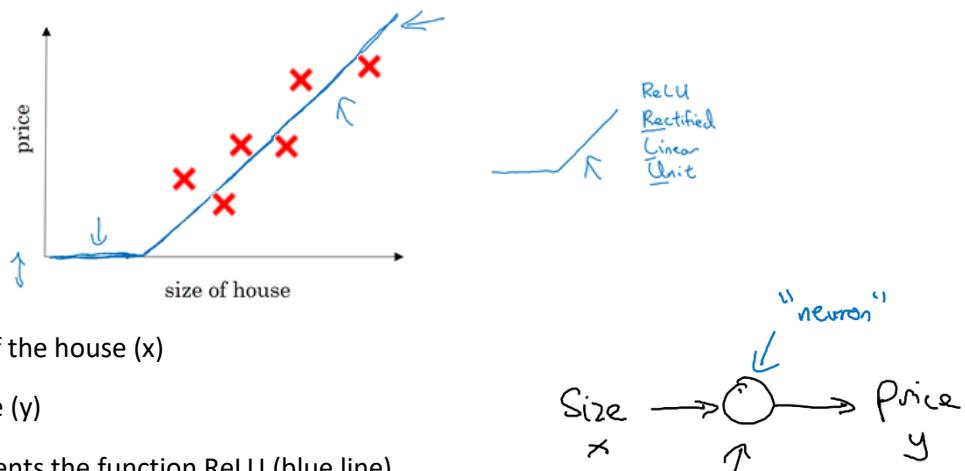
It is a powerful learning algorithm inspired by how the brain works.

### Example 1 – single neural network

Given data about the size of houses on the real estate market and you want to fit a function that will predict their price. It is a linear regression problem because the price as a function of size is a continuous output.

We know the prices can never be negative so we are creating a function called Rectified Linear Unit (ReLU) which starts at zero.

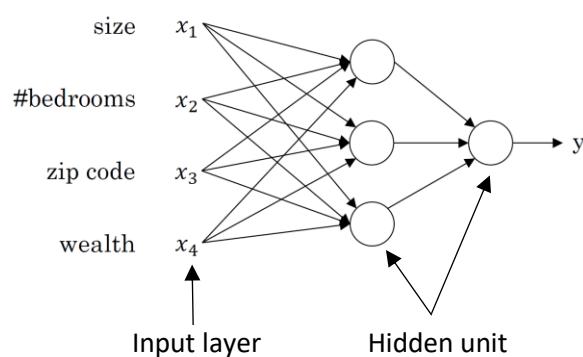
### Housing Price Prediction



### Example 2 – Multiple neural network

The price of a house can be affected by other features such as size, number of bedrooms, zip code and wealth. The role of the neural network is to predict the price and it will automatically generate the hidden units. We only need to give the inputs x and the output y.

### Housing Price Prediction





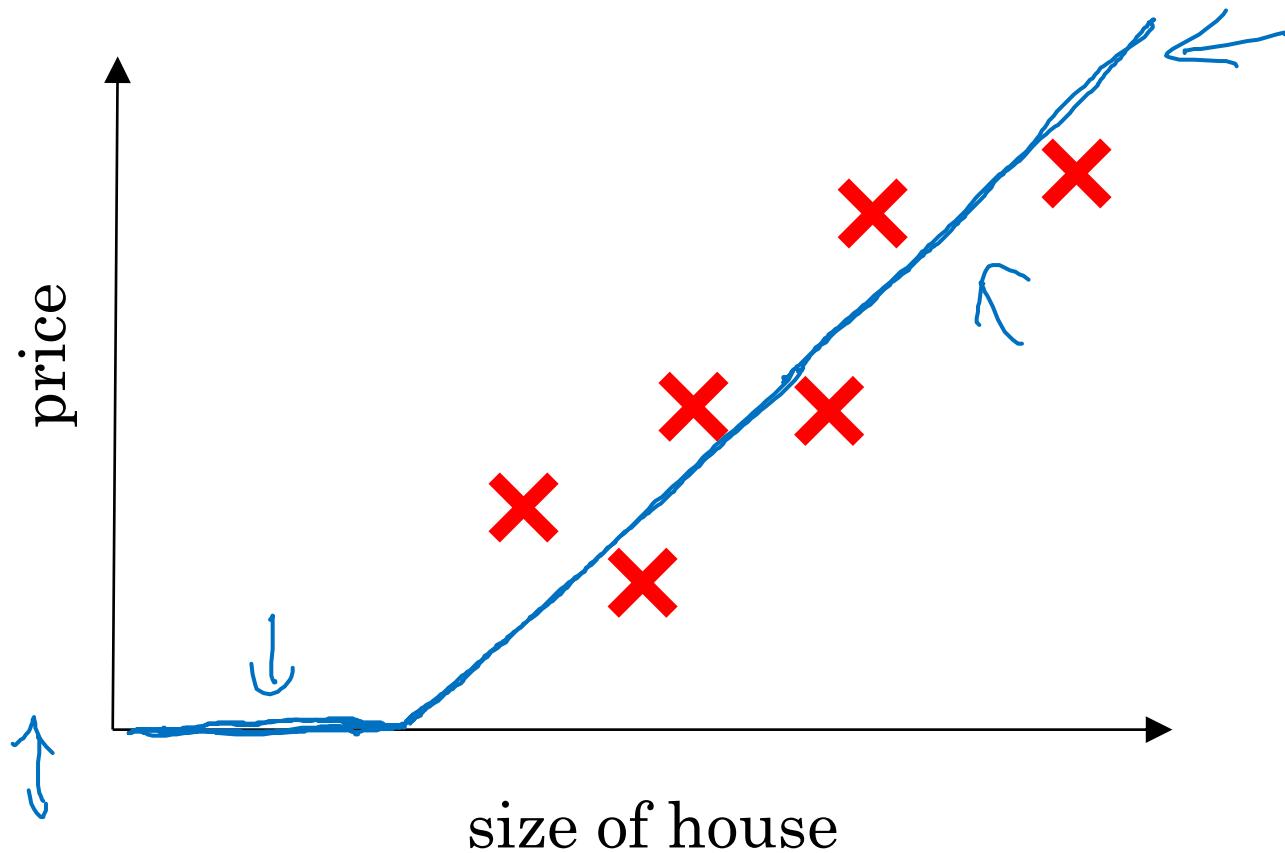
deeplearning.ai

# Introduction to Deep Learning

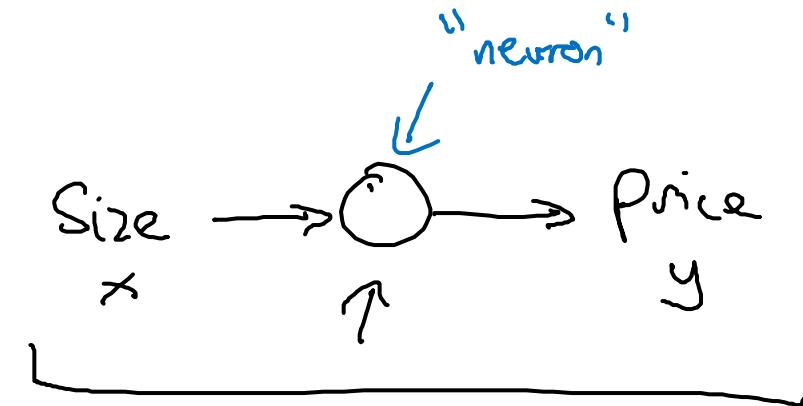
---

## What is a Neural Network?

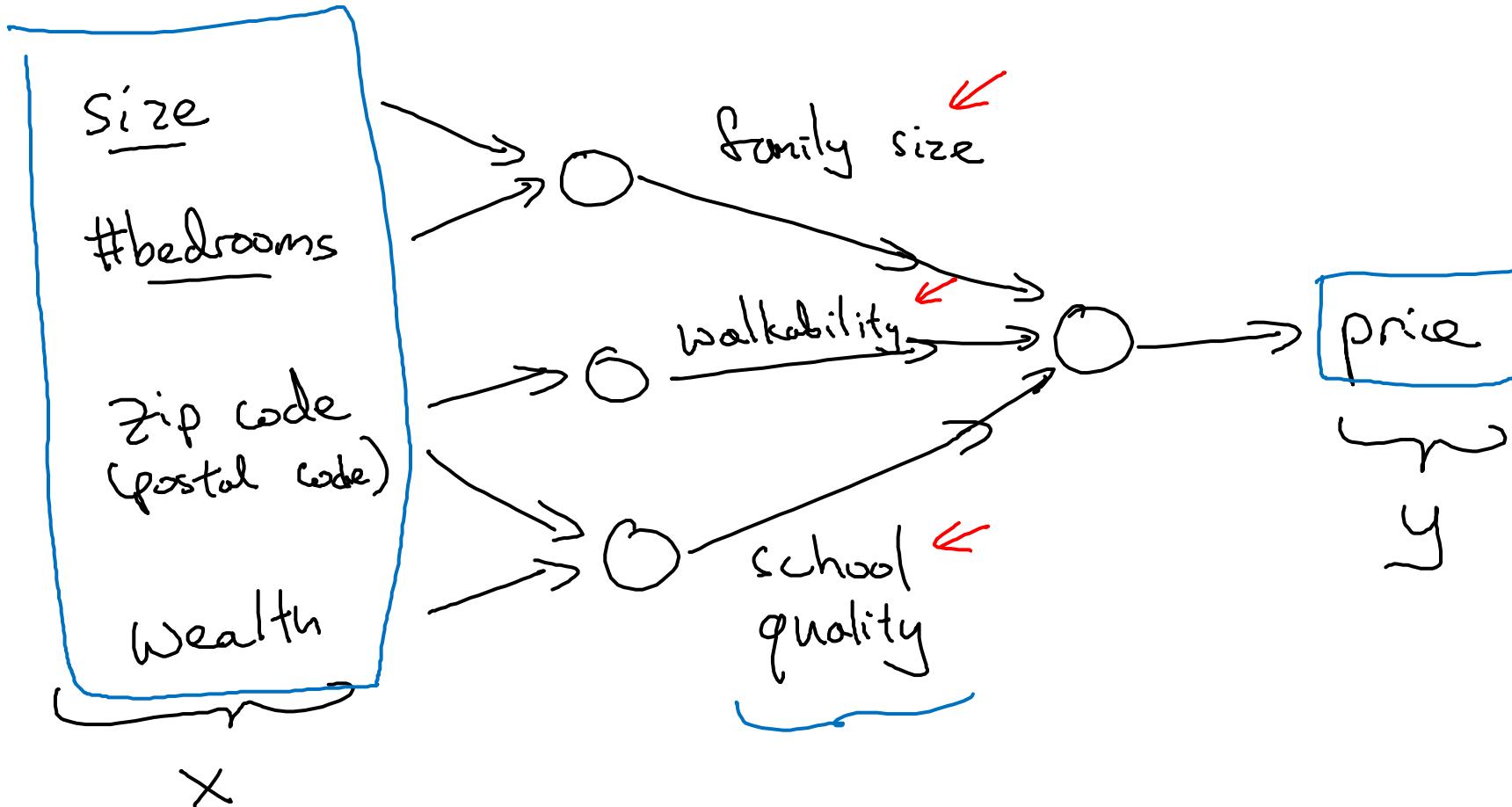
# Housing Price Prediction



ReLU  
Rectified  
Linear  
Unit

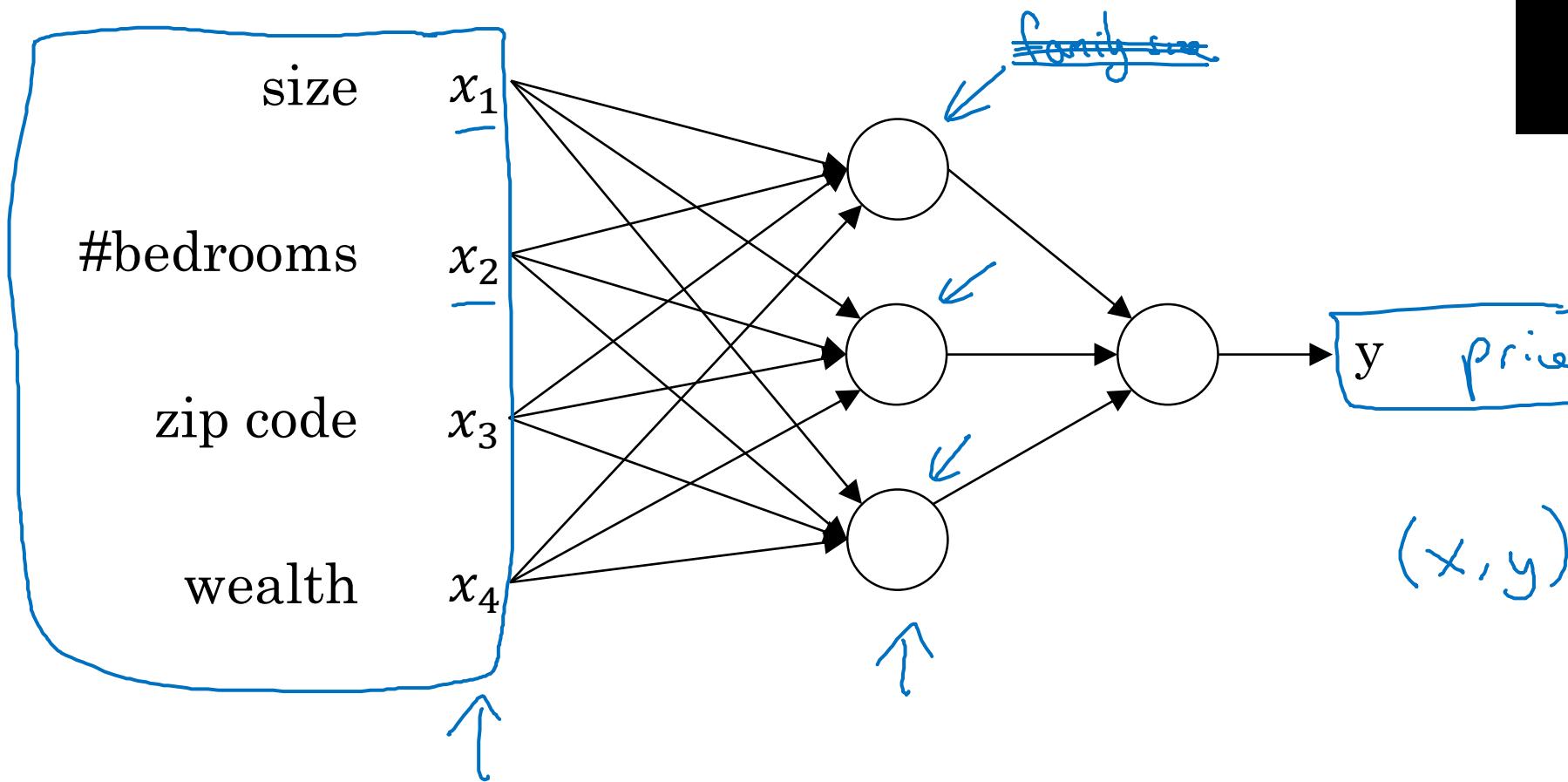


# Housing Price Prediction



# Housing Price Prediction

Drawing of  
previous Image



## Supervised learning for Neural Network

In supervised learning, we are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output.

Supervised learning problems are categorized into "regression" and "classification" problems. In a regression problem, we are trying to predict results within a continuous output, meaning that we are trying to map input variables to some continuous function. In a classification problem, we are instead trying to predict results in a discrete output. In other words, we are trying to map input variables into discrete categories.

Here are some examples of 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

There are different types of neural network, for example Convolution Neural Network (CNN) used often for image application and Recurrent Neural Network (RNN) used for one-dimensional sequence data such as translating English to Chinese or a temporal component such as text transcript. As for the autonomous driving, it is a hybrid neural network architecture.

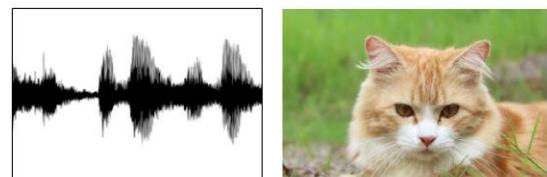
Structured vs unstructured data

Structured data refers to things that has a defined meaning such as price, age whereas unstructured data refers to things like pixel, raw audio, text.

Structured Data

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

Unstructured Data



Audio

Image

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

Four scores and seven years ago...

Text



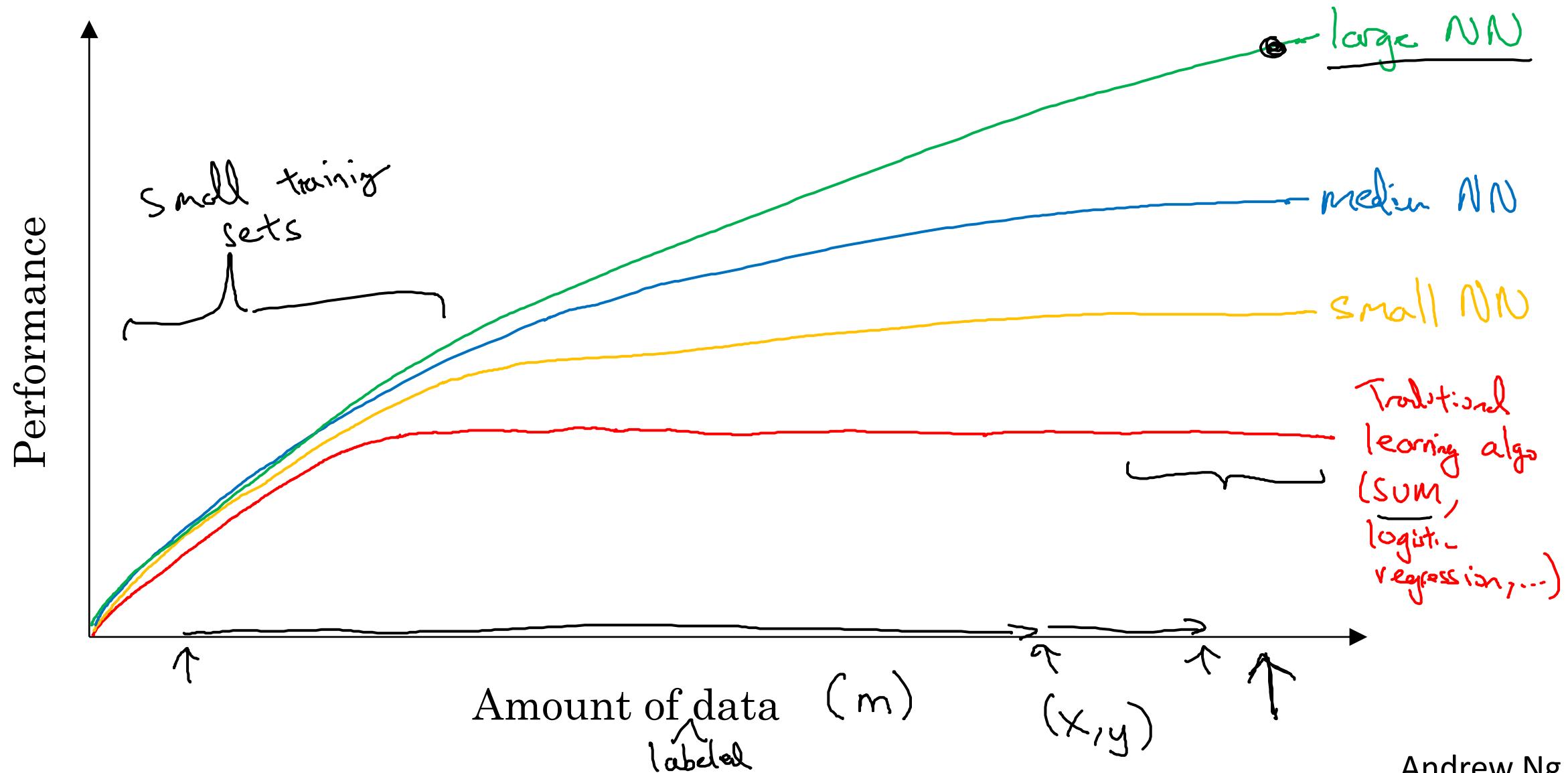
deeplearning.ai

# Introduction to Neural Networks

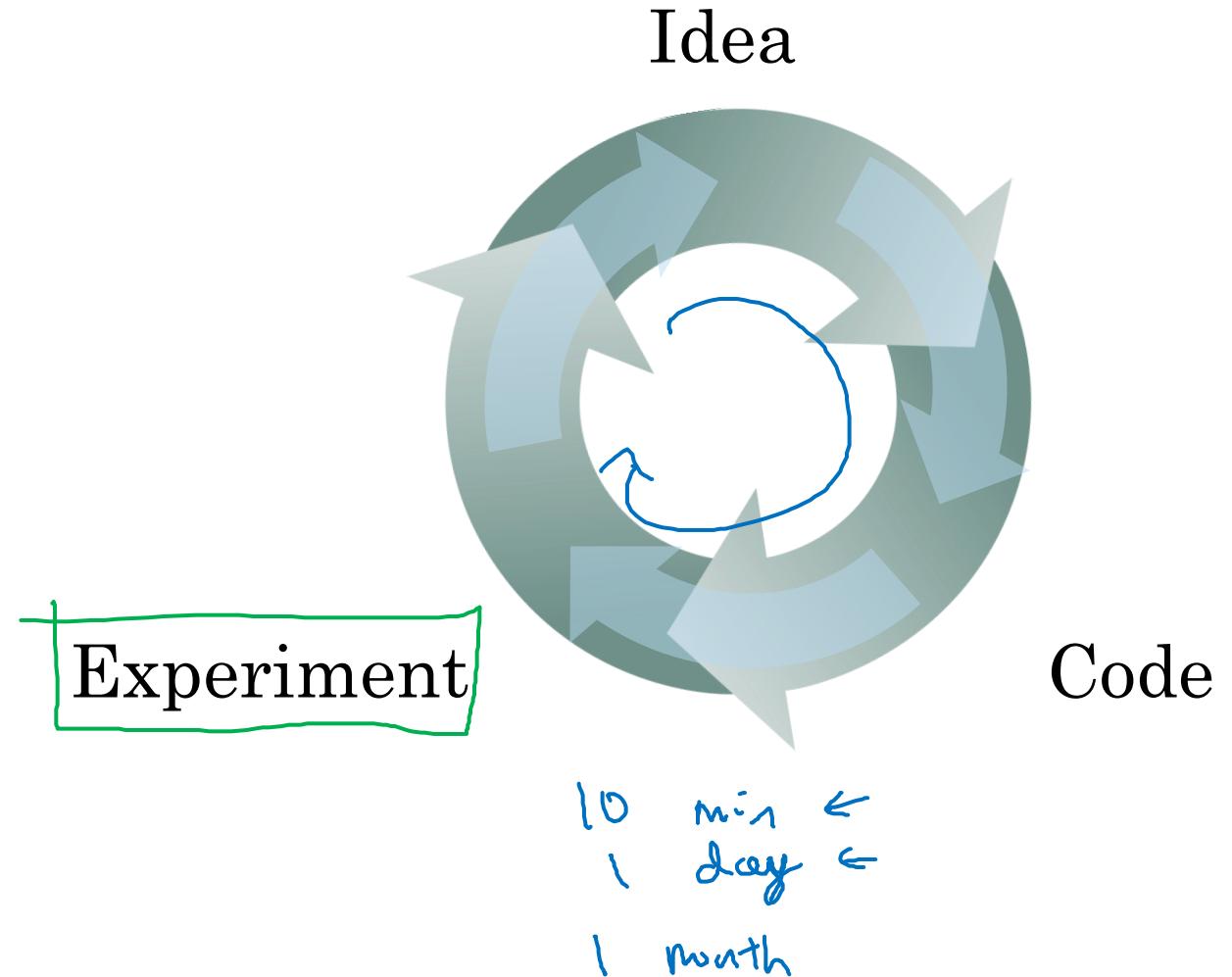
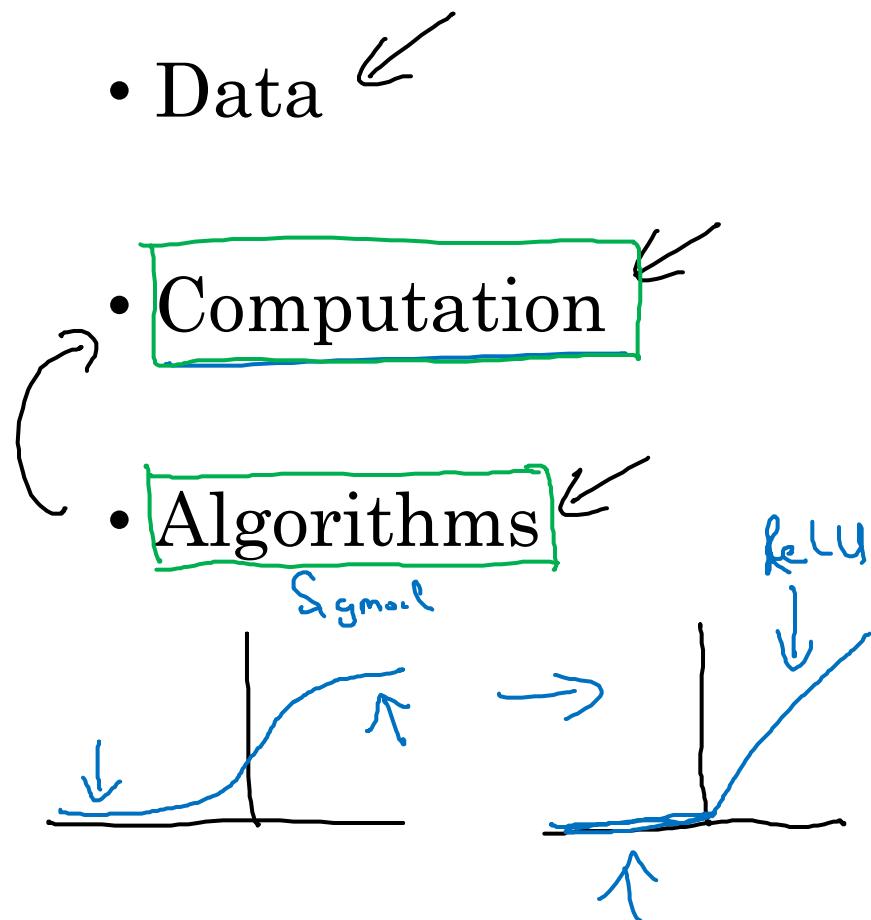
---

## Why is Deep Learning taking off?

# Scale drives deep learning progress



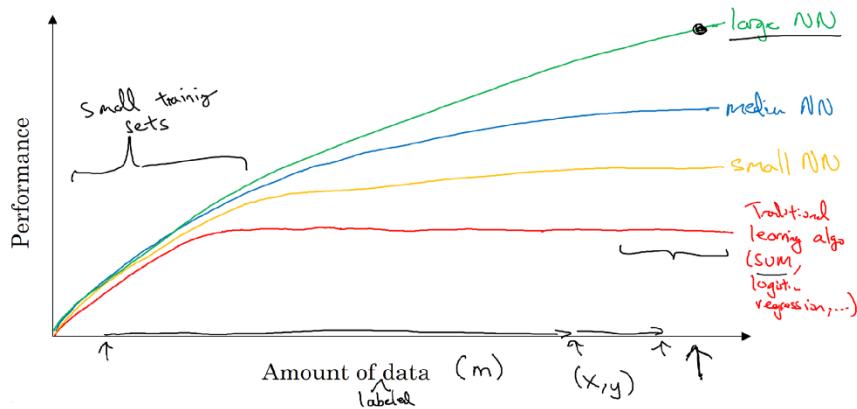
# Scale drives deep learning progress



## Why is deep learning taking off?

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

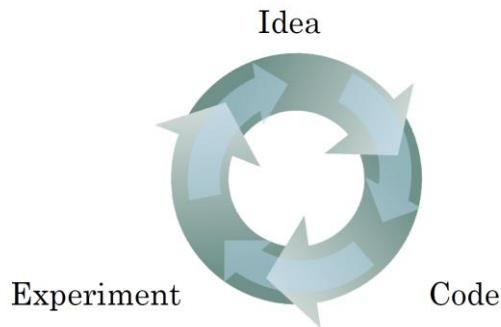
### Scale drives deep learning progress



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

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

The process of training a neural network is iterative.



It could take a good amount of time to train a neural network, which affects your productivity. Faster computation helps to iterate and improve new algorithm.



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



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



deeplearning.ai

# Introduction to Neural Networks

---

## Course resources

# Course Resources

Discussion forum

- Questions, technical discussions, bug reports, etc.

Wiki (in-progress lecture notes)

- [deeplearning.ai/wiki](https://deeplearning.ai/wiki)

Contact us: [feedback@deeplearning.ai](mailto:feedback@deeplearning.ai)

Companies: [enterprise@deeplearning.ai](mailto:enterprise@deeplearning.ai)

Universities: [academic@deeplearning.ai](mailto:academic@deeplearning.ai)

# Course Resources

Discussion forum

- Questions, technical discussions, bug reports, etc.

Contact us: [feedback@deeplearning.ai](mailto:feedback@deeplearning.ai)

Companies: [enterprise@deeplearning.ai](mailto:enterprise@deeplearning.ai)

Universities: [academic@deeplearning.ai](mailto:academic@deeplearning.ai)

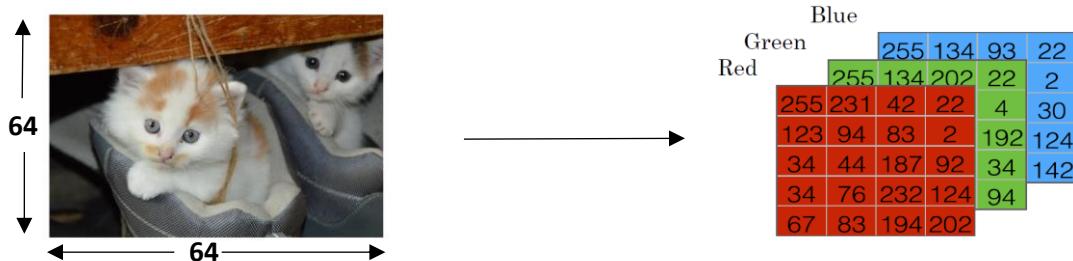
## Binary Classification

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 stored 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  $n_x = 64 \times 64 \times 3 = 12,288$ .

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



deeplearning.ai

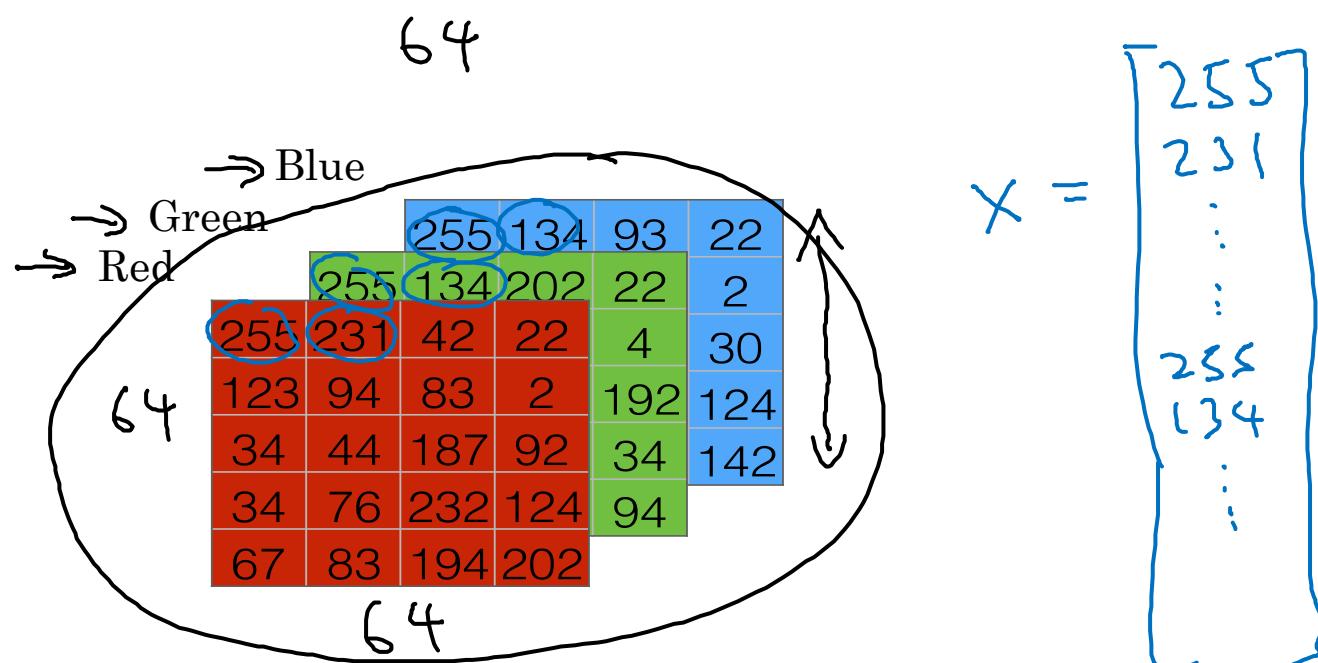
# Basics of Neural Network Programming

---

## Binary Classification

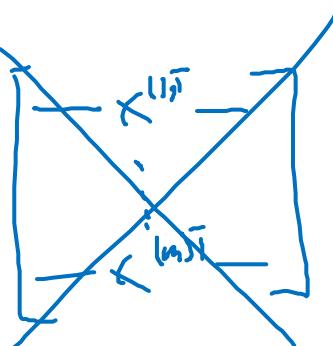
# Binary Classification

64 →  → 1 (cat) vs 0 (non cat)



# Notation

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

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

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

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

## 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.

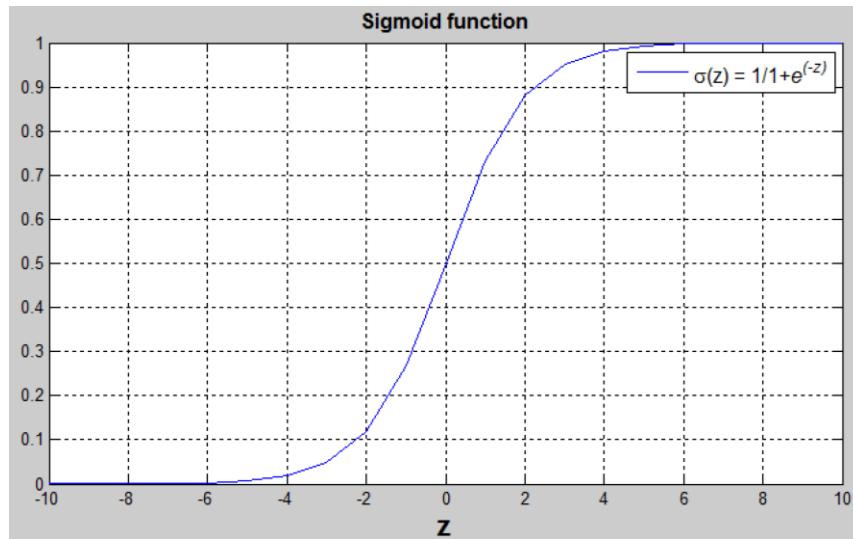
Example: Cat vs No - cat

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

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

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  $\sigma(z) = 0.5$



deeplearning.ai

# Basics of Neural Network Programming

---

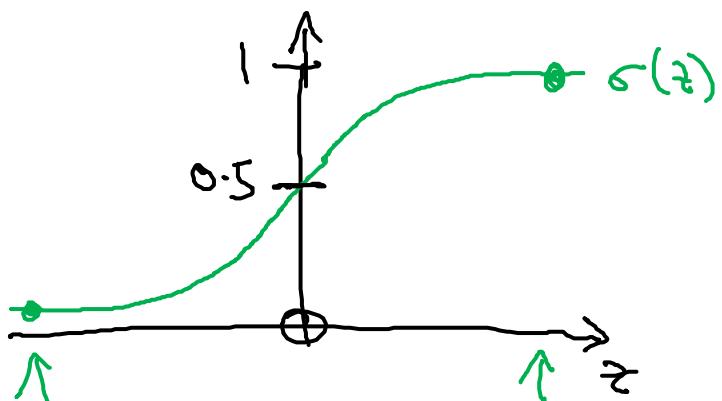
## Logistic Regression

# Logistic Regression

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

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

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



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

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

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

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

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

If  $z$  large negative number

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



deeplearning.ai

# Basics of Neural Network Programming

---

## Logistic Regression cost function

# Logistic Regression cost function

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

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

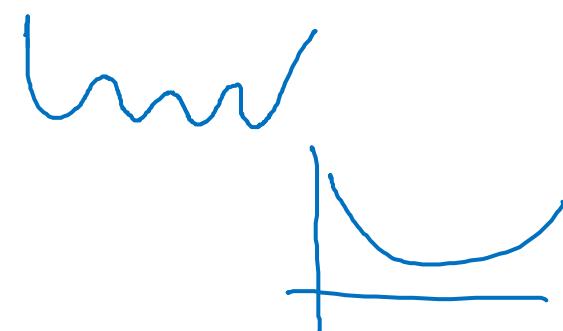
**Loss** (error) function:

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

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

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

*i-th example.*



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

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

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

## Logistic Regression: Cost Function

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

Recap:

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

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

Given  $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$ , 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.

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



deeplearning.ai

# Basics of Neural Network Programming

---

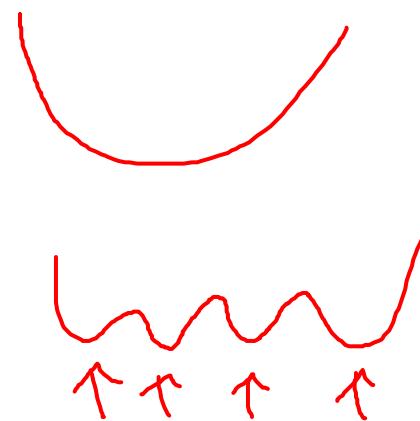
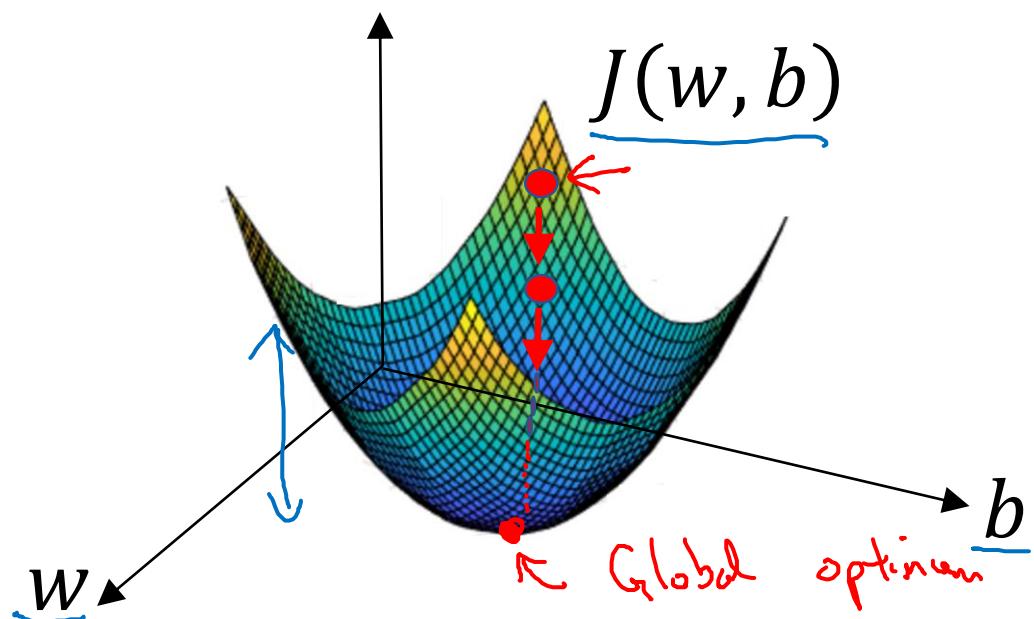
## Gradient Descent

# Gradient Descent

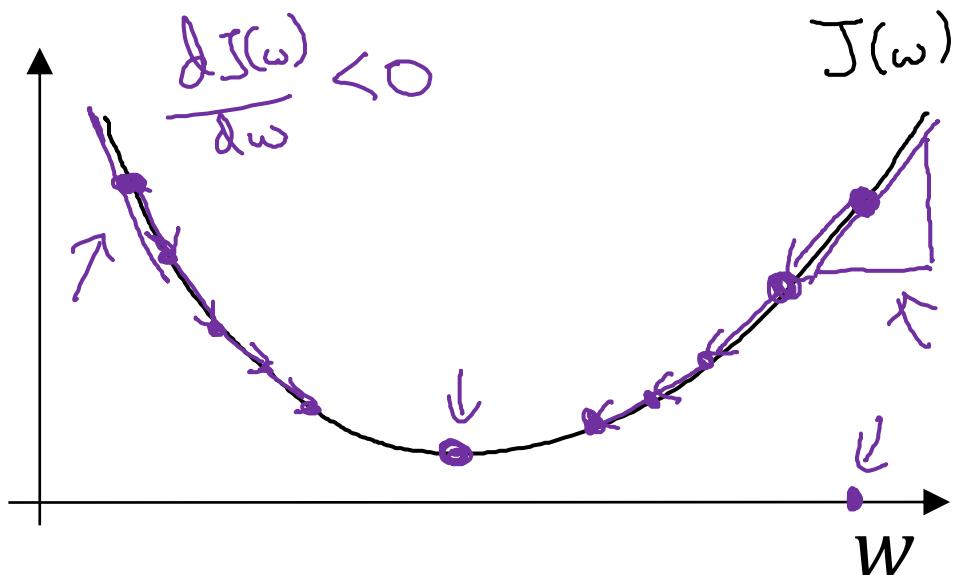
Recap:  $\hat{y} = \sigma(w^T x + b)$ ,  $\sigma(z) = \frac{1}{1+e^{-z}}$  

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

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



# Gradient Descent



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

"learning rate"

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

---

$J(w, b)$

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

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

$\frac{dJ(w, b)}{dw}$  "partial derivative"  $J$

$\frac{dJ(w, b)}{db}$

$\frac{dJ(w, b)}{dw}$

$\frac{dJ(w, b)}{db}$

Andrew Ng



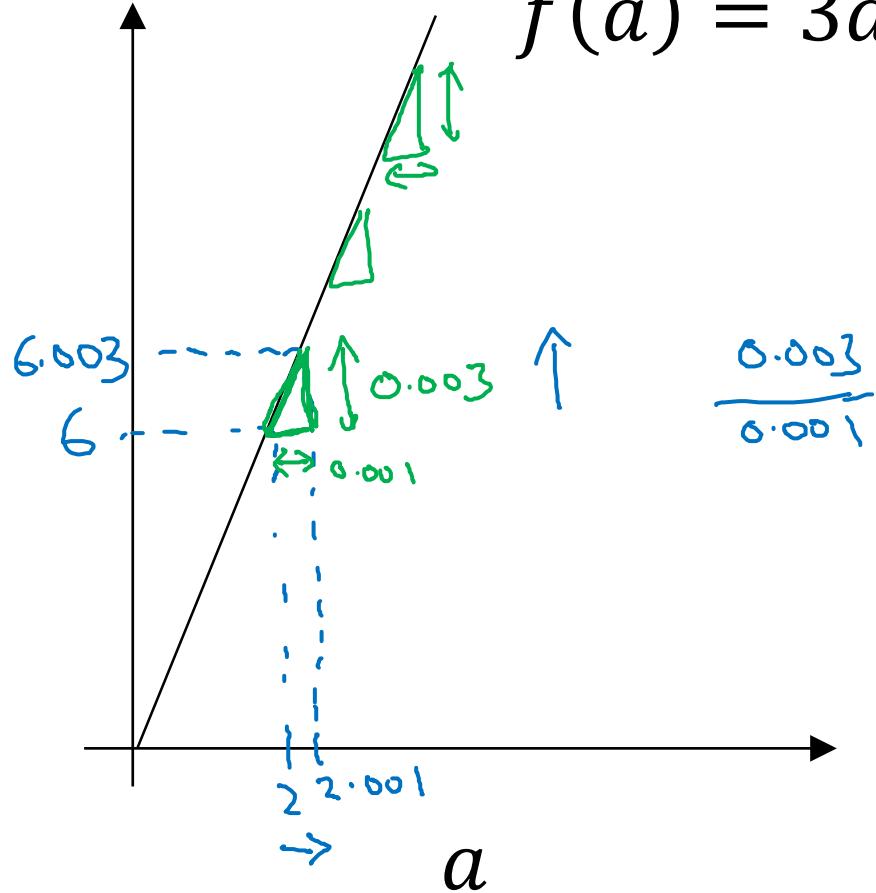
deeplearning.ai

# Basics of Neural Network Programming

---

## Derivatives

# Intuition about derivatives



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

height  
width

$$\rightarrow a = 2$$

$$a = 2.001$$

$$f(a) = 6$$

$$f(a) = 6.003$$

slope (derivative) of  $f(a)$

at  $a=2$  is 3

$$\rightarrow a = 5$$

$$a = 5.001$$

$$f(a) = 15$$

$$f(a) = 15.003$$

slope at  $a=5$  is also 3

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

0.001  
0.00000001  
0.0000000001



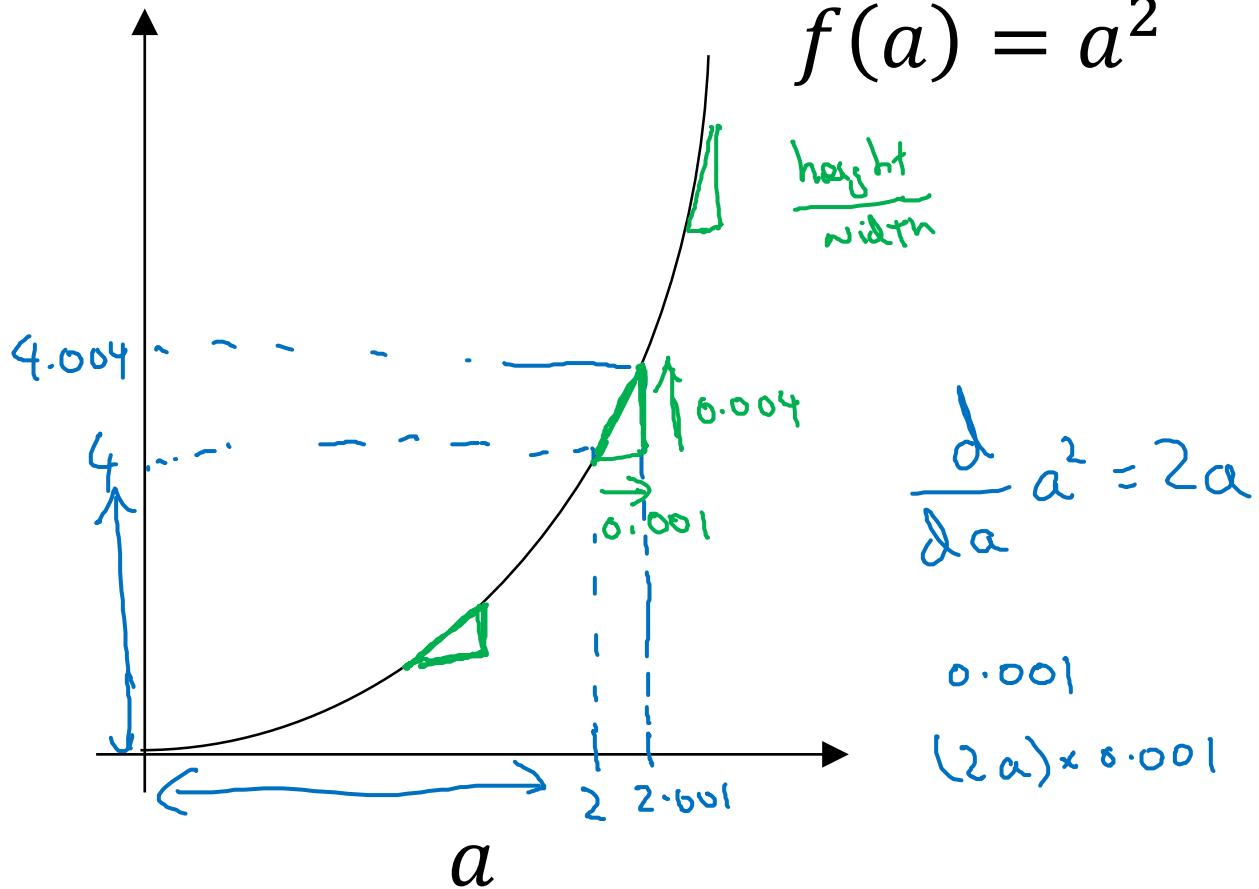
deeplearning.ai

# Basics of Neural Network Programming

---

More derivatives  
examples

# Intuition about derivatives



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

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

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

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

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

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

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

# More derivative examples

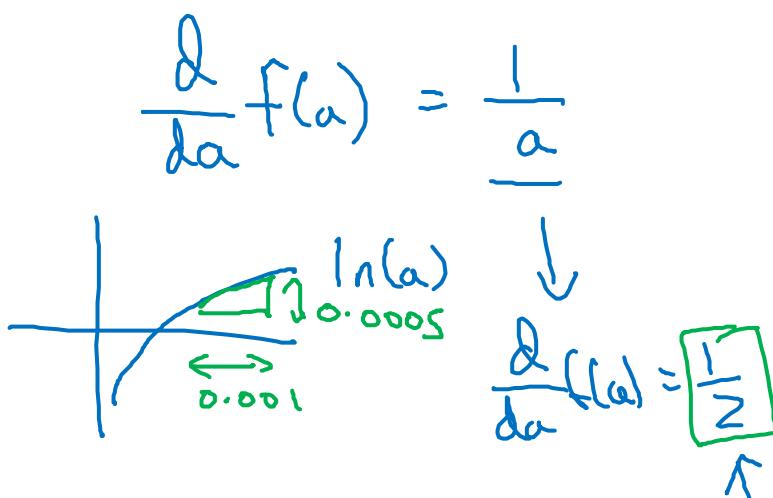
$$f(a) = a^2$$

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

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

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

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



$$a = 2$$

$$a = 2.001$$

$$f(a) = 4$$

$$f(a) \approx 4.004$$

$$a = 2$$

$$a = \underline{2.001}$$

$$f(a) = 8$$

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

$$a = 2$$

$$a = \underline{2.001}$$

$$f(a) \approx 0.69315$$

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

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



deeplearning.ai

# Basics of Neural Network Programming

---

## Computation Graph

# Computation Graph

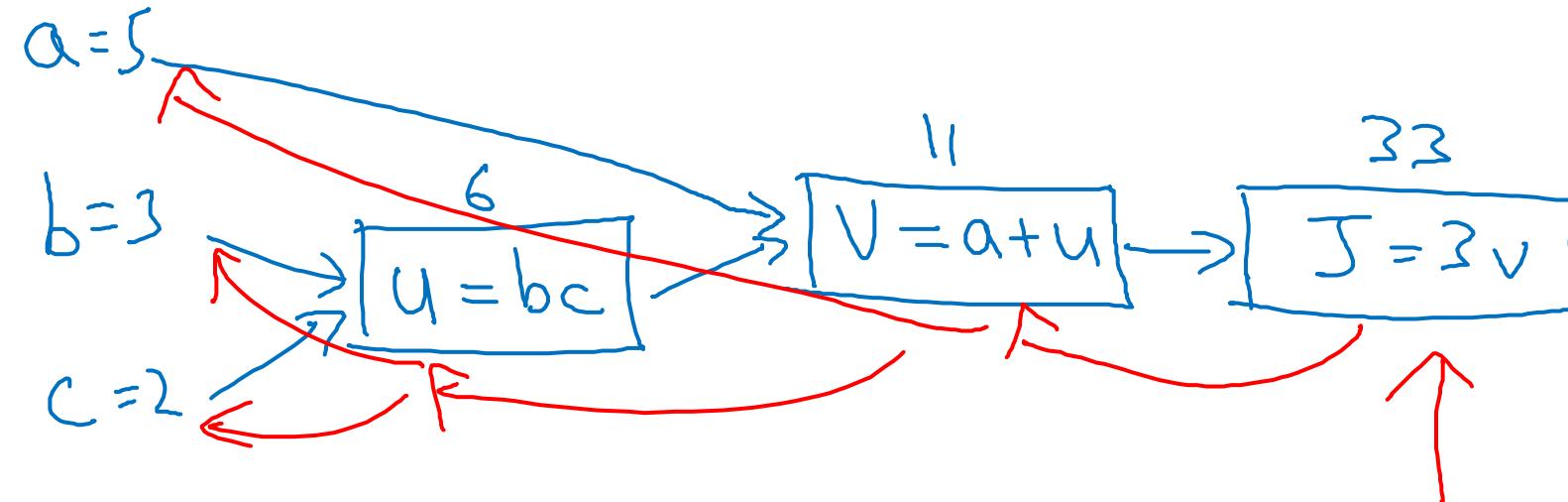
$$J(a, b, c) = 3(u + bc) = 3(5 + 3 \times 2) = 33$$

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

$$u = bc$$

$$v = a + u$$

$$J = 3v$$





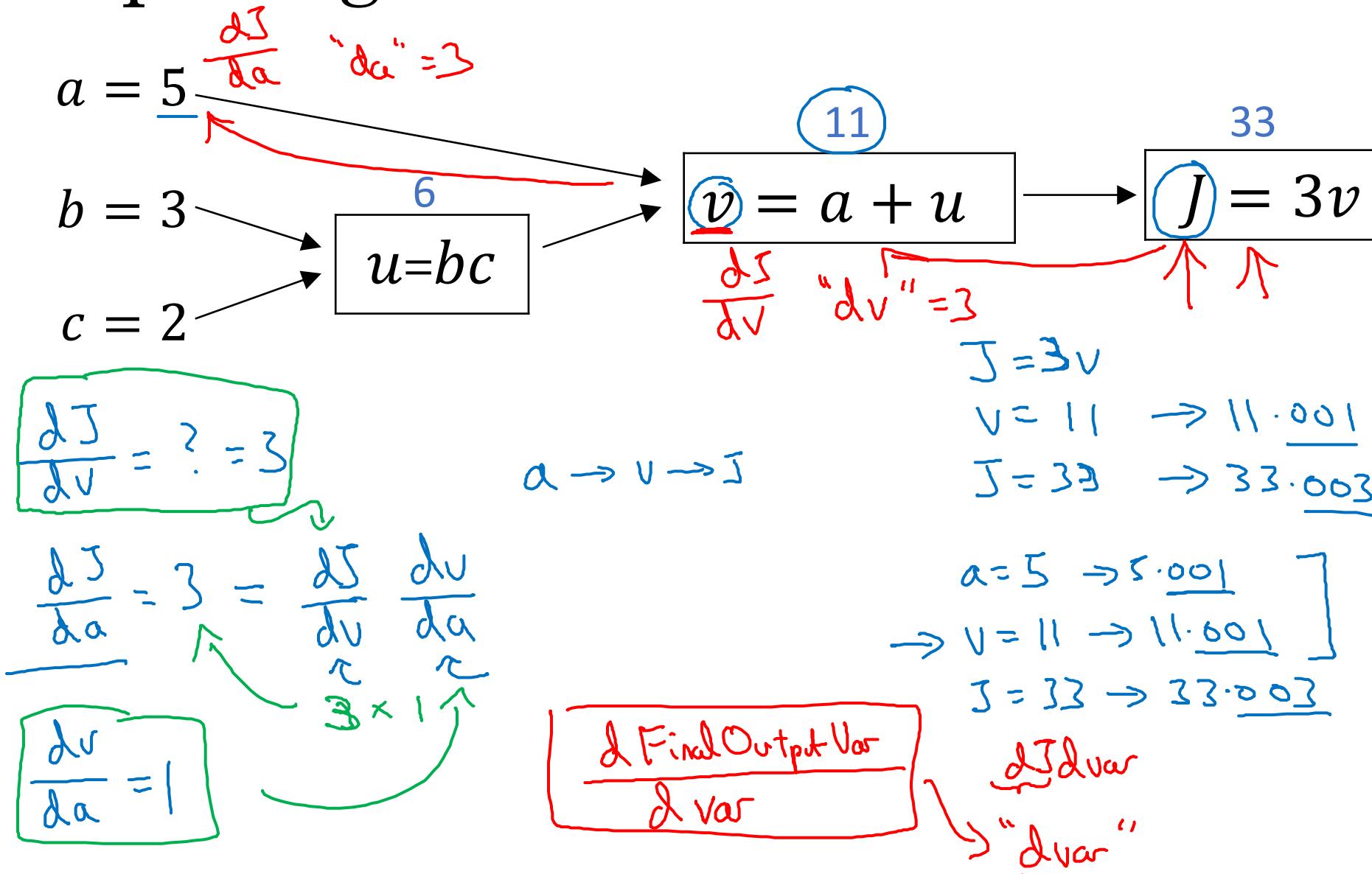
deeplearning.ai

# Basics of Neural Network Programming

---

## Derivatives with a Computation Graph

# Computing derivatives



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

# Computing derivatives

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

$a = 5$  →  $v = a + u$  →  $J = 3v$   
 $b = 3$  →  $v = a + u$   
 $c = 2$  →  $u = bc$   
 $u = 6$  →  $v = a + u$   
 $\frac{\partial u}{\partial v} = 3$   
 $\frac{\partial u}{\partial a} = 1$   
 $\frac{\partial u}{\partial b} = 6$   
 $\frac{\partial u}{\partial c} = 2$   
 $\frac{\partial J}{\partial u} = 3 = \frac{\partial J}{\partial v} \cdot \frac{\partial v}{\partial u}$   
 $\frac{\partial J}{\partial b} = \frac{\partial J}{\partial u} \cdot \frac{\partial u}{\partial b} = 3 \cdot 6 = 18$   
 $\frac{\partial J}{\partial a} = \frac{\partial J}{\partial u} \cdot \frac{\partial u}{\partial a} = 3 \cdot 1 = 3$

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

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

$v = 11.002$   
 $J = 33.006$



deeplearning.ai

# Basics of Neural Network Programming

---

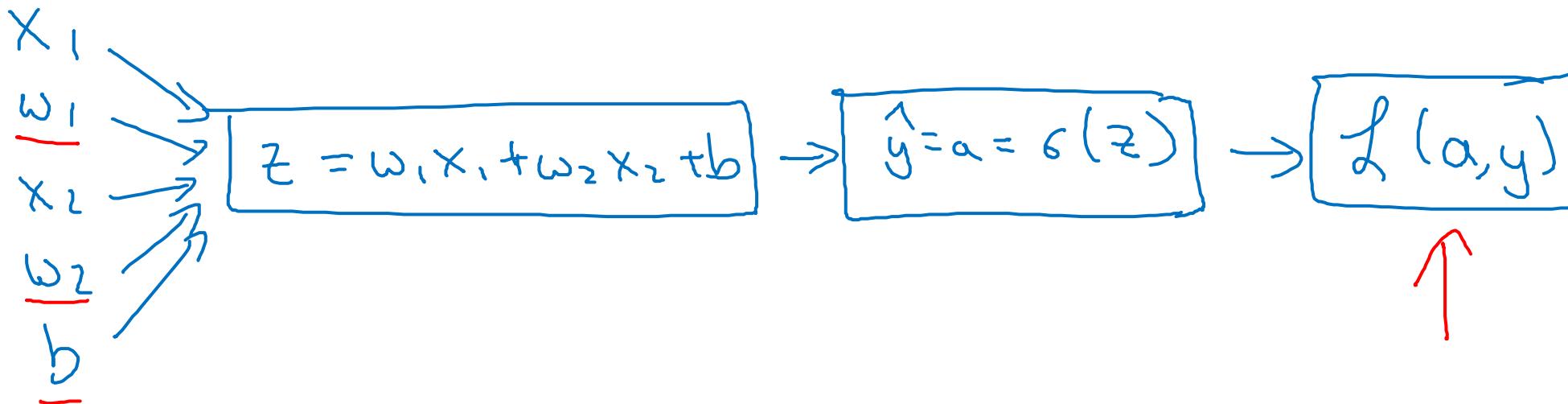
## Logistic Regression Gradient descent

# Logistic regression recap

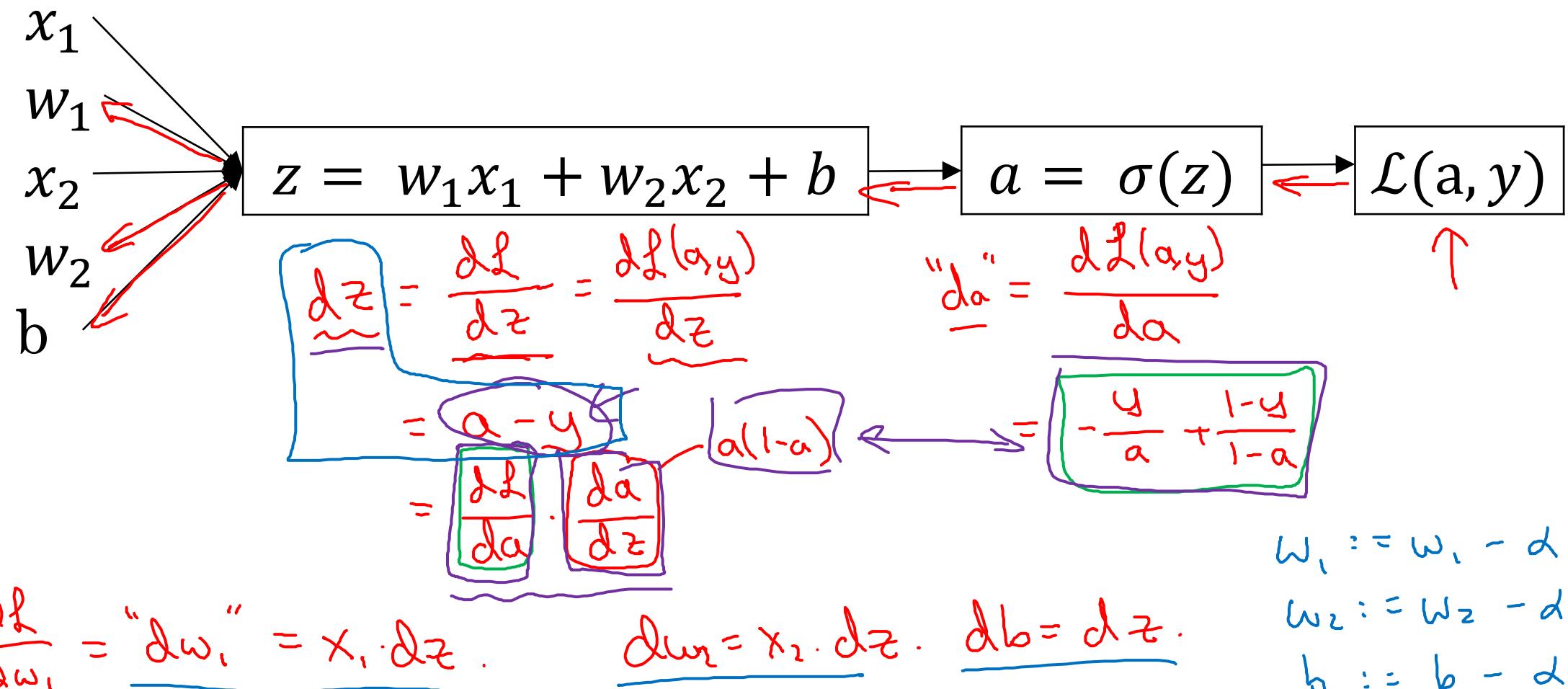
$$\rightarrow z = w^T x + b$$

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

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



# Logistic regression derivatives



$$w_1 := w_1 - \alpha \frac{\partial \mathcal{L}}{\partial w_1}$$

$$w_2 := w_2 - \alpha \frac{\partial \mathcal{L}}{\partial w_2}$$

$$b := b - \alpha \frac{\partial \mathcal{L}}{\partial b}$$



deeplearning.ai

# Basics of Neural Network Programming

---

## Gradient descent on $m$ examples

# Logistic regression on $m$ examples

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

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

# Logistic regression on $m$ examples

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

→ For  $i = 1$  to  $m$

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

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

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

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

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

$$\begin{aligned} \Delta w_3 &= \dots \\ \Delta w_n &= \dots \end{aligned}$$

$$J / m \leftarrow$$

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

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

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

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

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

Vectorization



deeplearning.ai

# Basics of Neural Network Programming

---

## Vectorization

# What is vectorization?

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

Non-vectorized:

$$z = 0$$

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

$$z += b$$

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

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

Vectorized

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

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



deeplearning.ai

# Basics of Neural Network Programming

---

## More vectorization examples

# Neural network programming guideline

Whenever possible, avoid explicit for-loops.

# Neural network programming guideline

Whenever possible, avoid explicit for-loops.

$$u = Av$$

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

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

for i ...

    for j ...

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

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

# Vectors and matrix valued functions

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

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

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

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

# Logistic regression derivatives

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

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

for i = 1 to n:

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

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

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

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

for j=1..n<sub>x</sub>  
 $dw_j += \frac{\partial J}{\partial w_j}$

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

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

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

$$d\omega /= m.$$



deeplearning.ai

# Basics of Neural Network Programming

---

## Vectorizing Logistic Regression

# Vectorizing Logistic Regression

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

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

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

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

$\xrightarrow{\quad}$

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

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

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

$$\dots w^T x^{(m)} + b$$

$$\rightarrow \underline{\underline{Z}} = np.\text{dot}(w.T, X) + \underbrace{b}_{(1, 1)}$$

R

"Broadcasting"

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



deeplearning.ai

# Basics of Neural Network Programming

---

## Vectorizing Logistic Regression's Gradient Computation

# Vectorizing Logistic Regression

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

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

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

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

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

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

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

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

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

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

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

$$= \frac{1}{m} \left[ \underline{x^{(1)} dz^{(1)}} + \dots + \underline{x^{(m)} dz^{(m)}} \right]_{n \times 1}$$

# Implementing Logistic Regression

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

for i = 1 to m:

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

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

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

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

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

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

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

$$db = db/m$$

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



deeplearning.ai

# Basics of Neural Network Programming

---

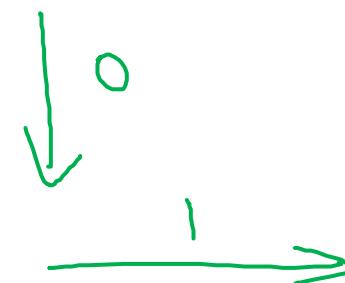
## Broadcasting in Python

# Broadcasting example

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

	Apples	Beef	Eggs	Potatoes	
Carb	56.0	0.0	4.4	68.0	
Protein	1.2	104.0	52.0	8.0	
Fat	1.8	135.0	99.0	0.9	
	59 cal	$\frac{56}{59} \approx 94.9\%$			

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



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

cal = A.sum(axis = 0)

percentage =  $100 * A / (\text{cal} / \text{A.reshape}(1, 4))$

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

# Broadcasting example

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

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

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

# General Principle

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

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

Matlab/Octave: bsxfun



deeplearning.ai

# Basics of Neural Network Programming

---

A note on python/  
numpy vectors

# Python Demo

# Python / numpy vectors

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



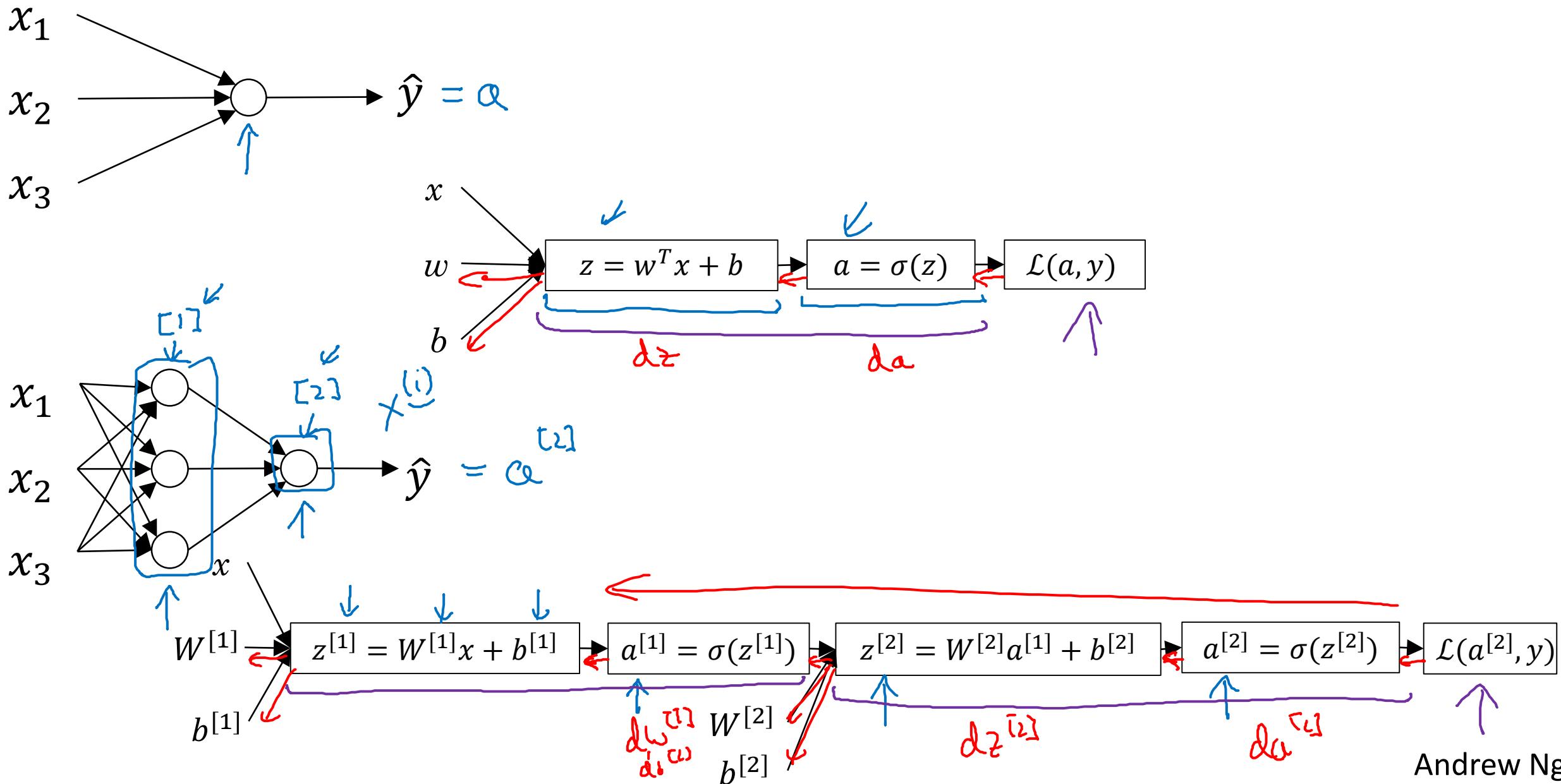
deeplearning.ai

One hidden layer  
Neural Network

---

Neural Networks  
Overview

# What is a Neural Network?





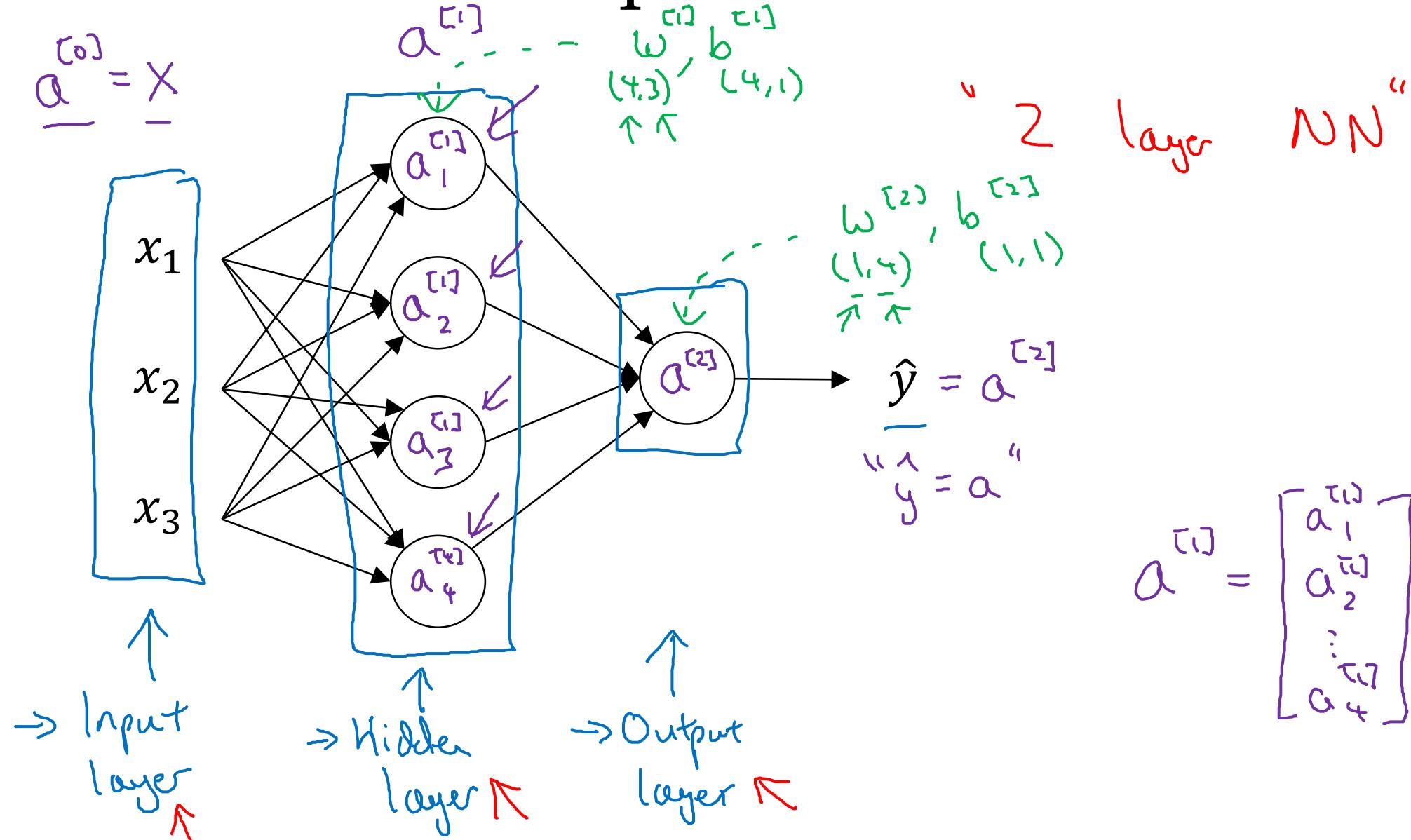
deeplearning.ai

One hidden layer  
Neural Network

---

Neural Network  
Representation

# Neural Network Representation





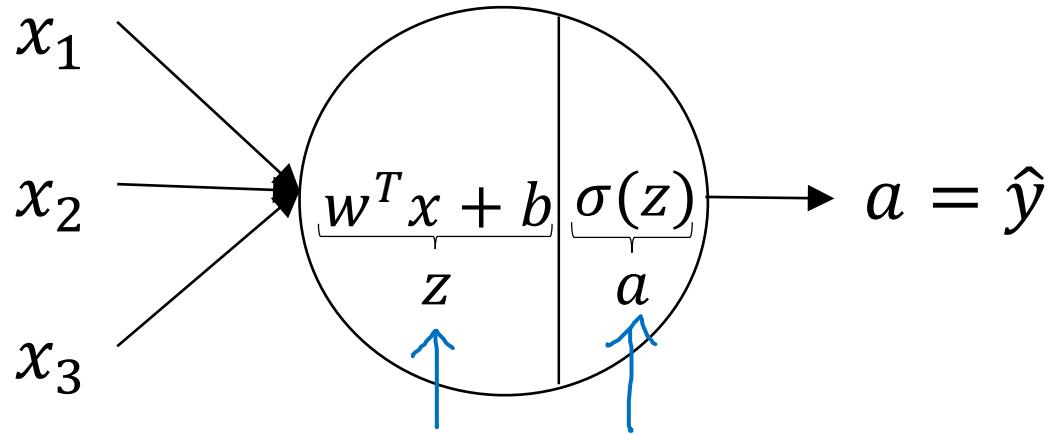
deeplearning.ai

# One hidden layer Neural Network

---

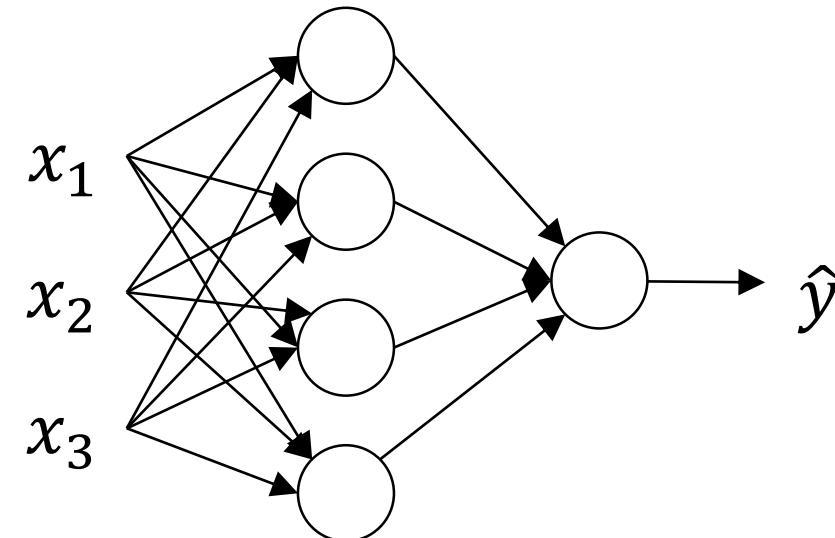
## Computing a Neural Network's Output

# Neural Network Representation

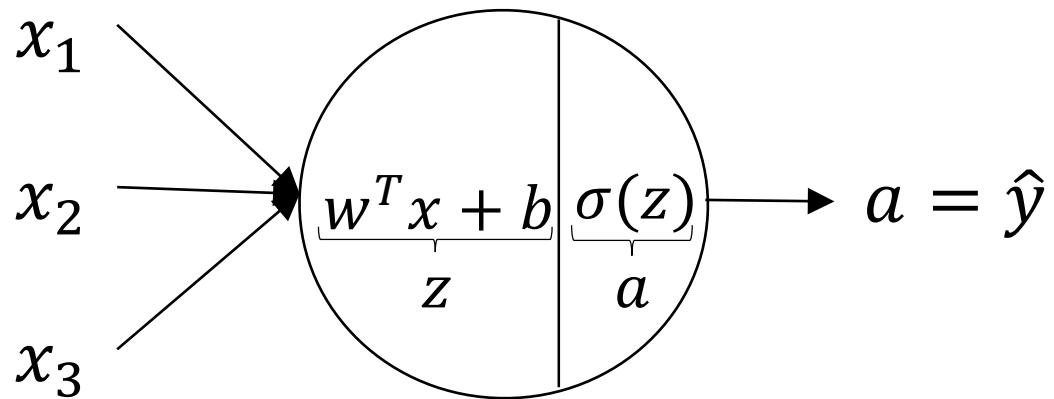


$$z = w^T x + b$$

$$a = \sigma(z)$$

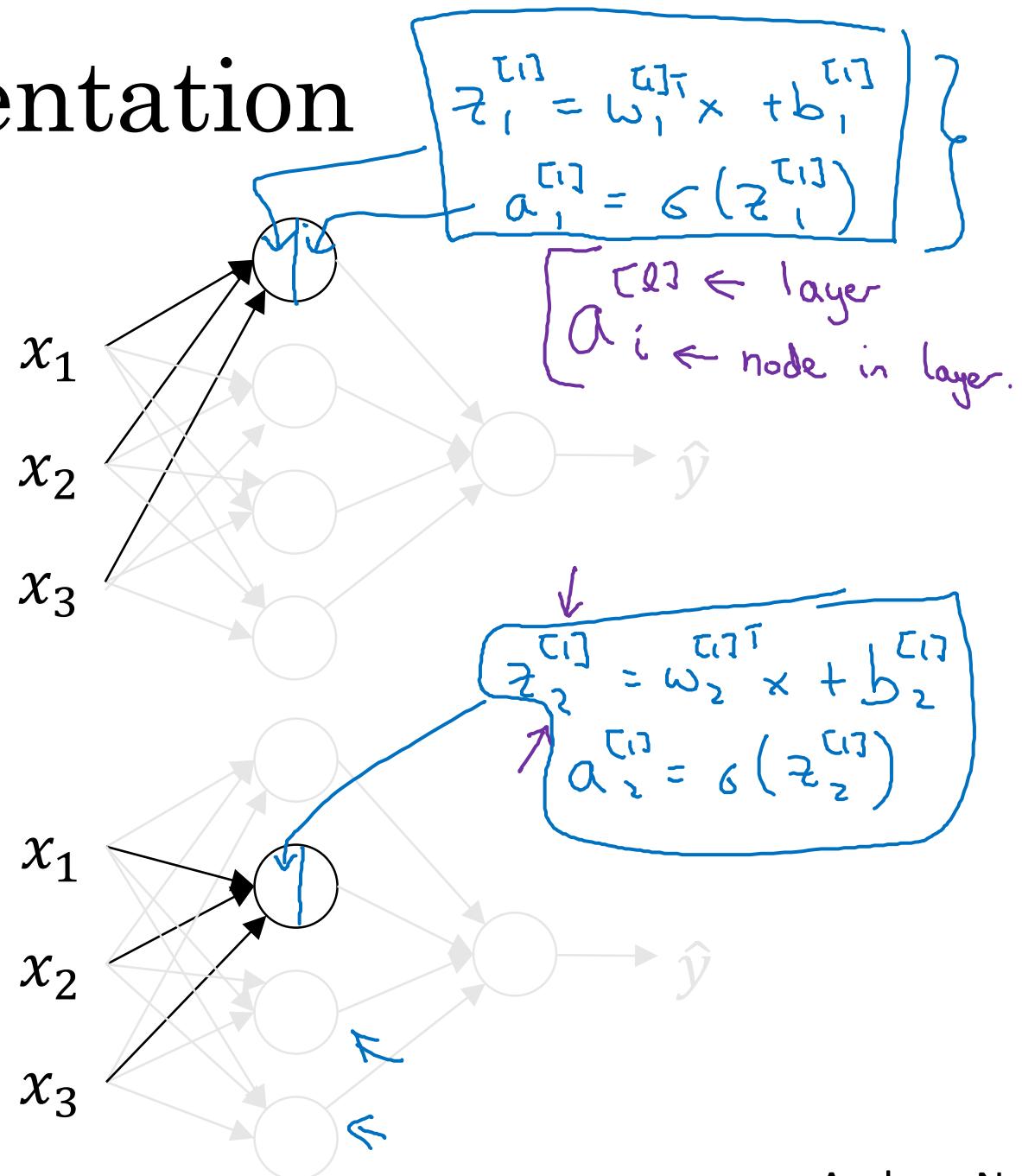


# Neural Network Representation

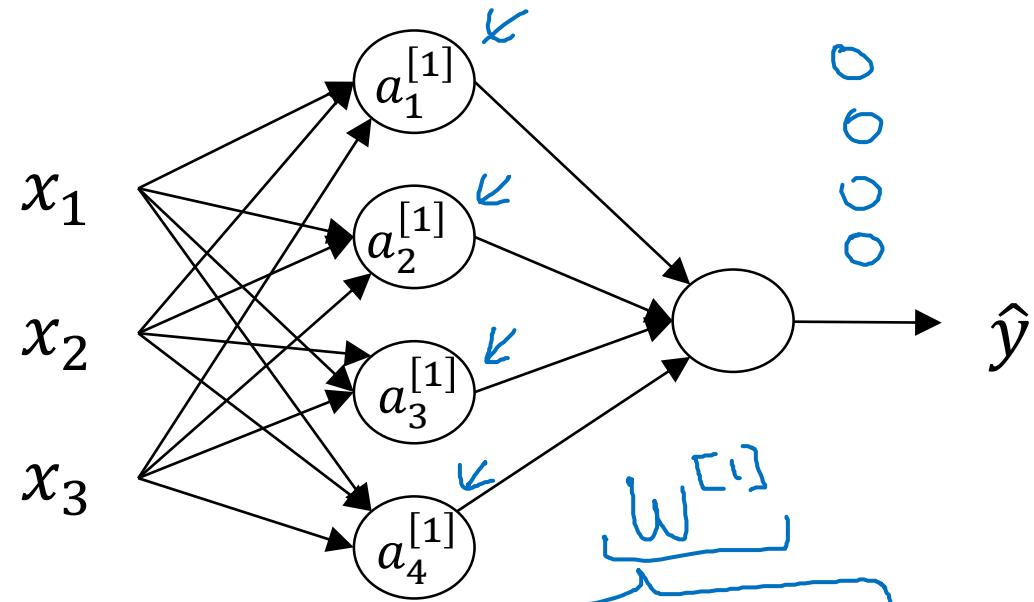


$$z = w^T x + b$$

$$a = \sigma(z)$$



# Neural Network Representation



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

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

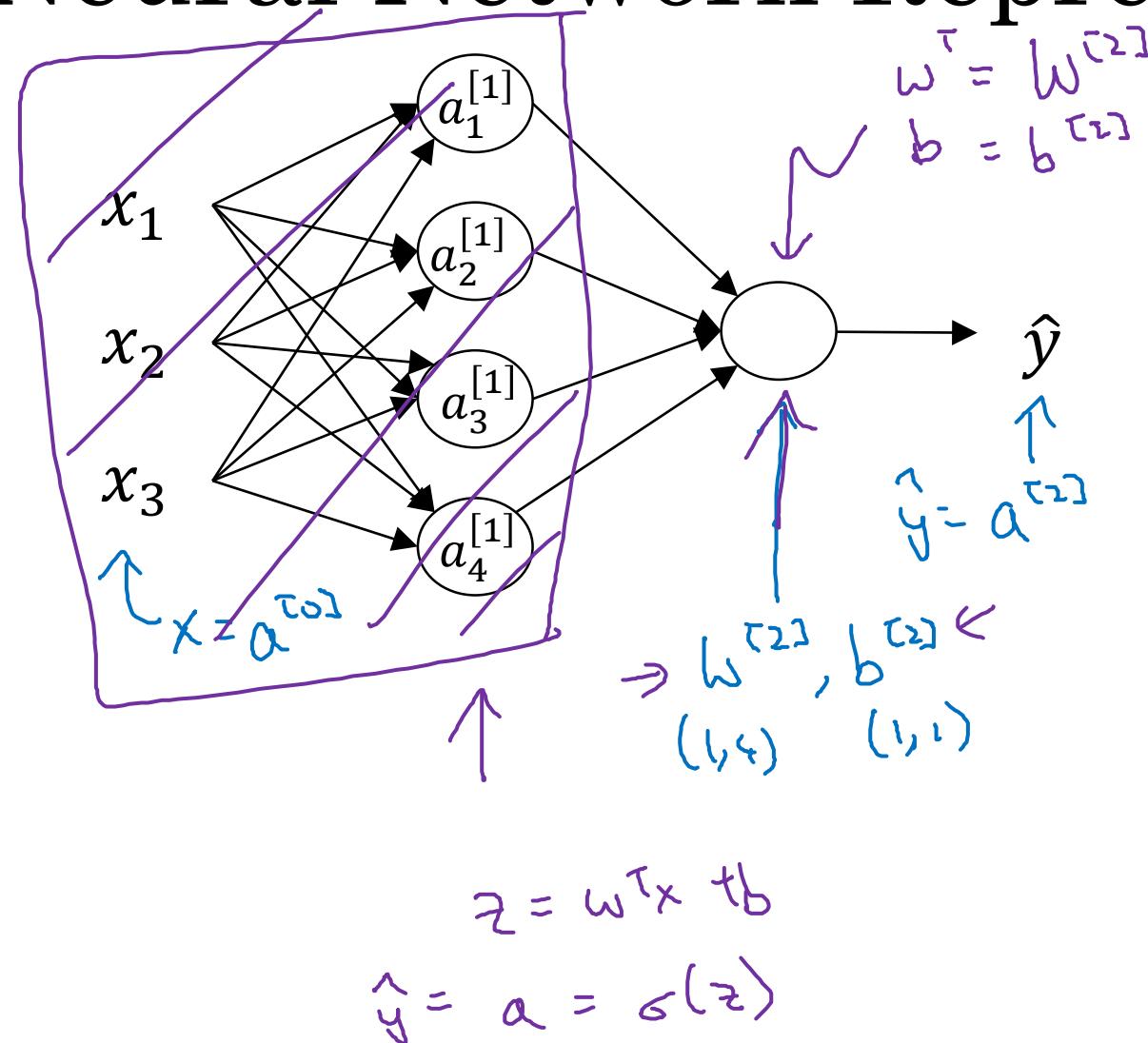
Diagram illustrating the mathematical representation of the neural network layers:

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

Annotations:

- $(w_1^{[1]T} x + b_1^{[1]})$  is circled in blue.
- $a^{[1]}$  is circled in red.
- $z^{[1]}$  is circled in purple.
- $b^{[1]}$  is circled in green.
- $w^{[1]}$  is circled in blue.
- $\sigma(z)$  is circled in red.

# Neural Network Representation learning



Given input  $x$ :

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



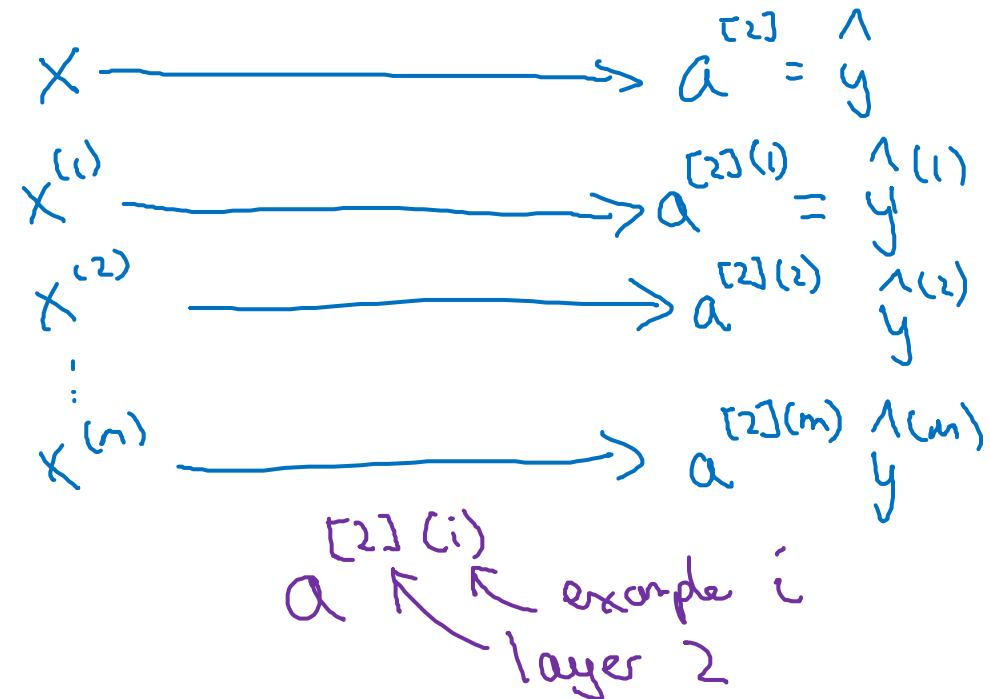
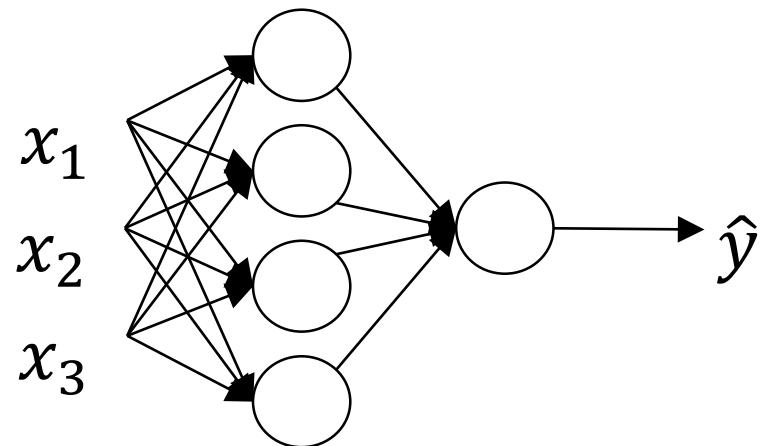
deeplearning.ai

# One hidden layer Neural Network

---

Vectorizing across  
multiple examples

# Vectorizing across multiple examples



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

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

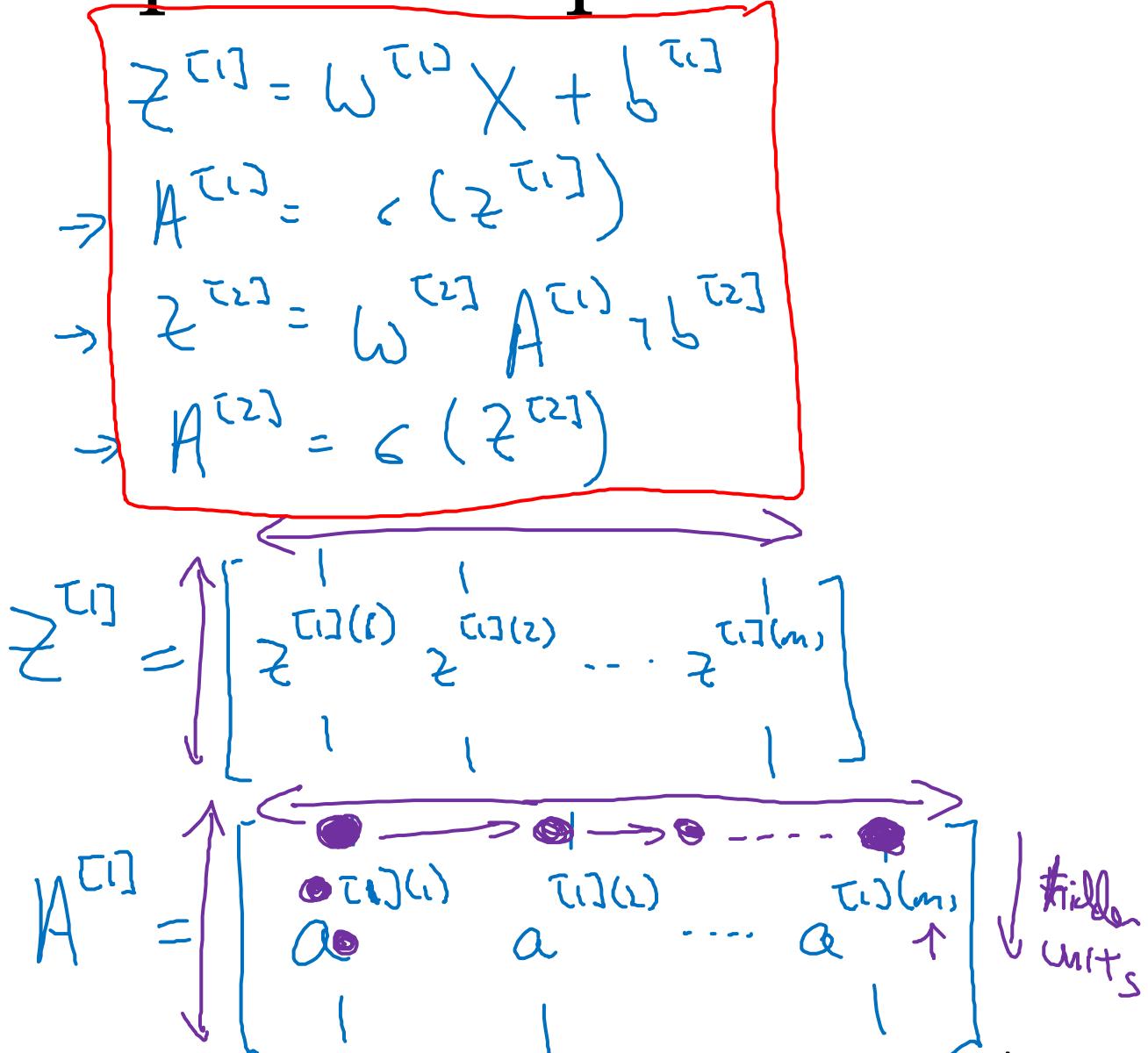
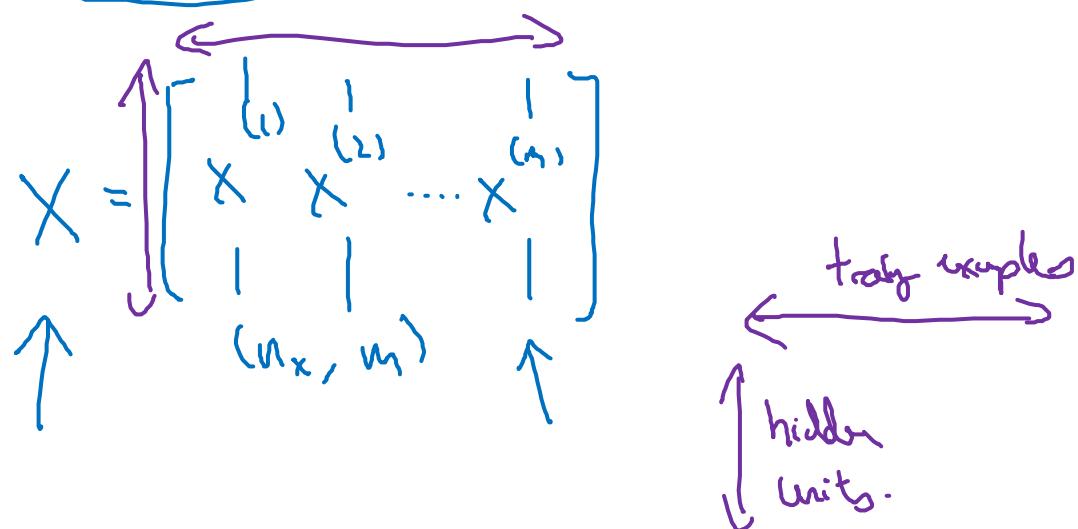
for  $i = 1$  to  $m$ :

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

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

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

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





deeplearning.ai

# One hidden layer Neural Network

---

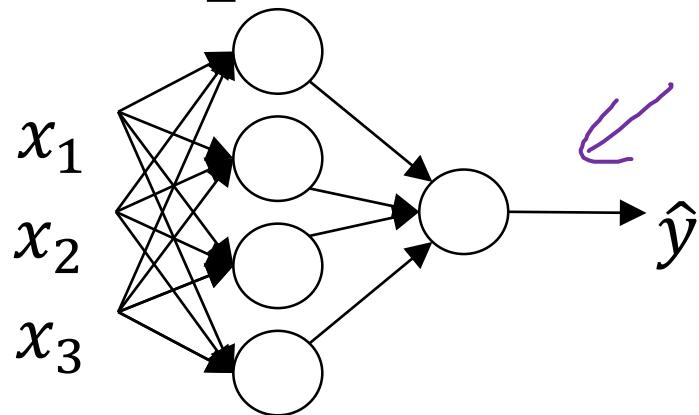
## Explanation for vectorized implementation

# Justification for vectorized implementation

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

Andrew Ng

# Recap of vectorizing across multiple examples



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

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

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

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

```

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

```

$A^{[1]} = a^{[1]}(1) \quad A^{[1]} = a^{[1]}(2) \quad \dots \quad A^{[1]} = a^{[1]}(m)$

$Z^{[1]} = W^{[1]}X + b^{[1]} \quad \leftarrow \quad w^{[1], j} A^{[1], j} + b^{[1]}$

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

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

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



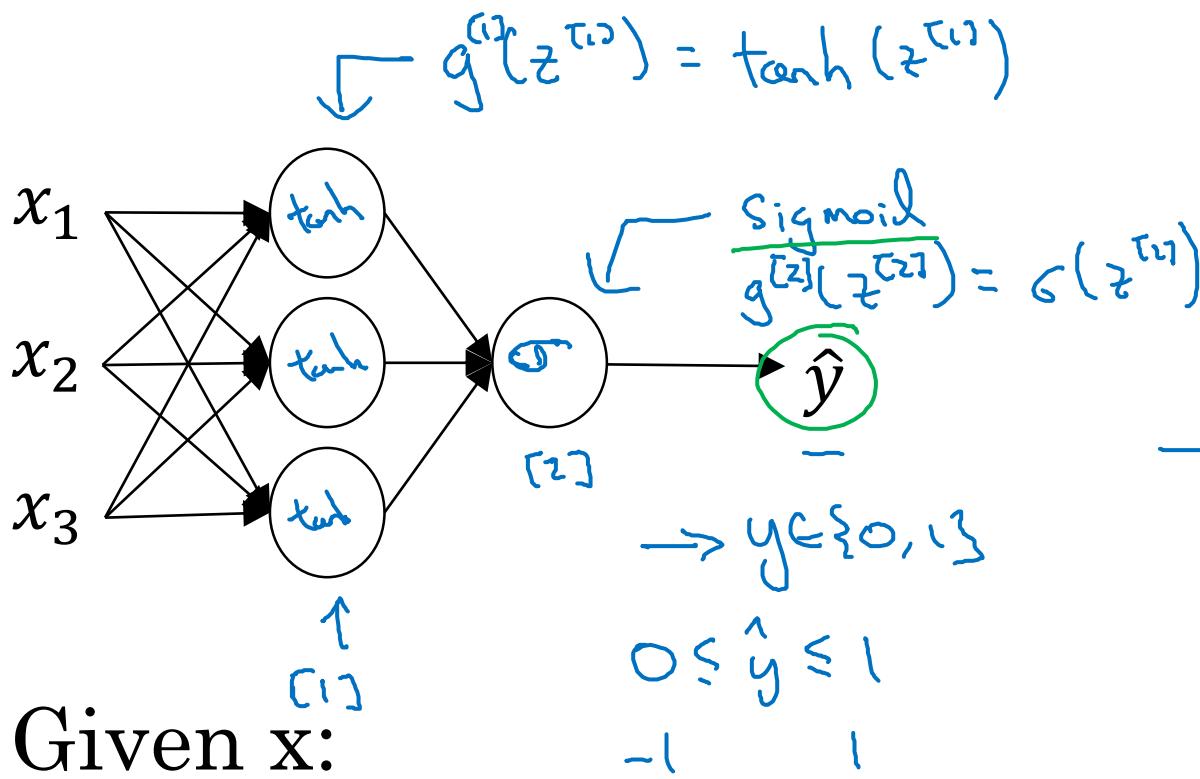
deeplearning.ai

One hidden layer  
Neural Network

---

Activation functions

# Activation functions

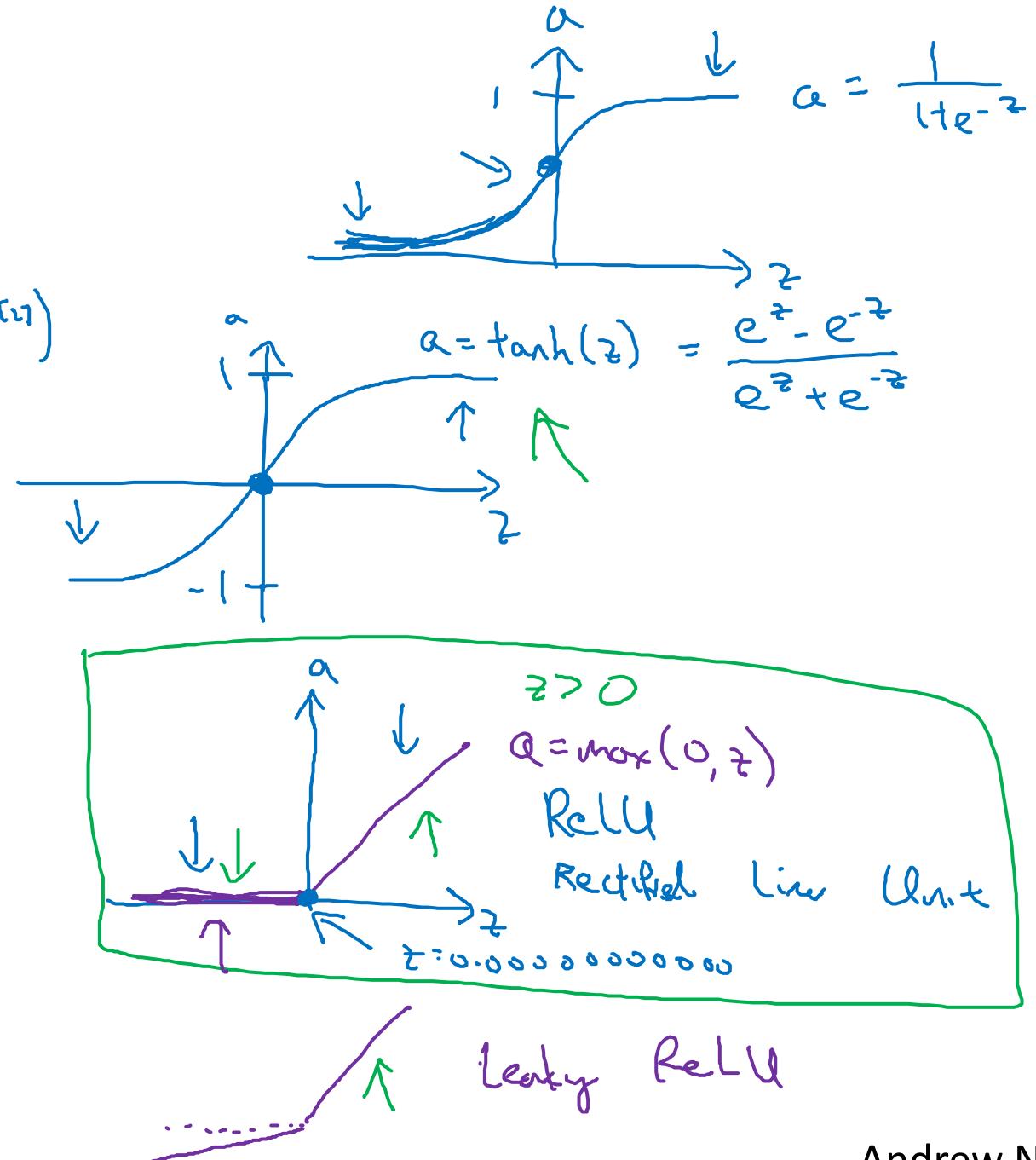


$$z^{[1]} = W^{[1]}x + b^{[1]}$$

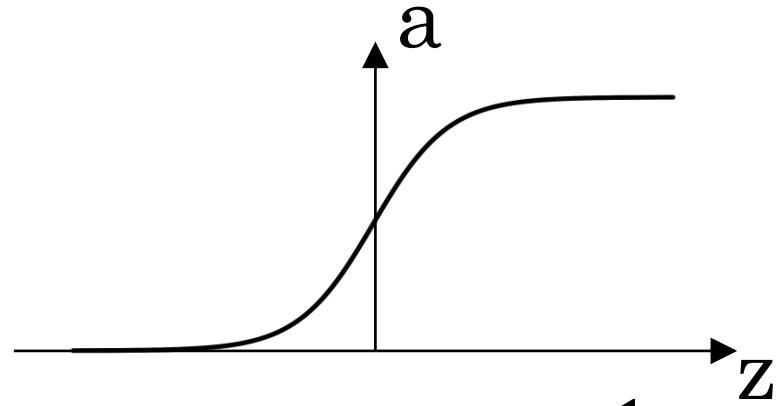
$$\rightarrow a^{[1]} = \cancel{\sigma(z^{[1]})} g^{(1)}(z^{(1)})$$

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

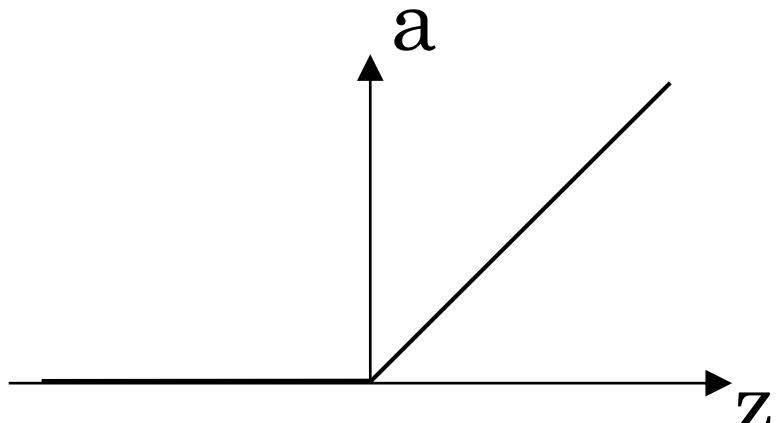
$$\rightarrow a^{[2]} = \cancel{\sigma(z^{[2]})} g^{(2)}(z^{(2)})$$



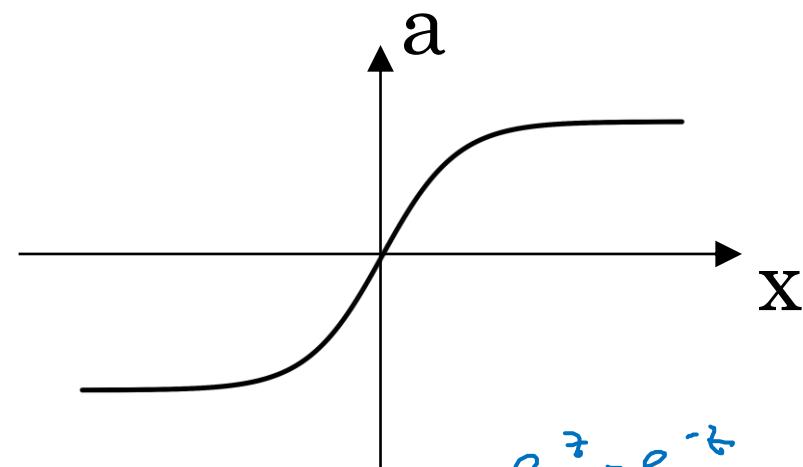
# Pros and cons of activation functions



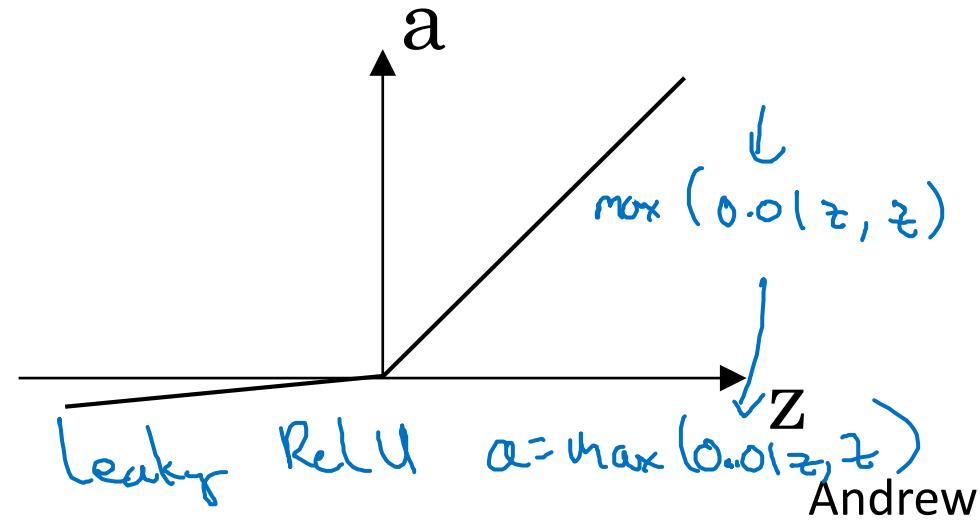
$$\text{sigmoid: } a = \frac{1}{1 + e^{-z}}$$



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



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



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



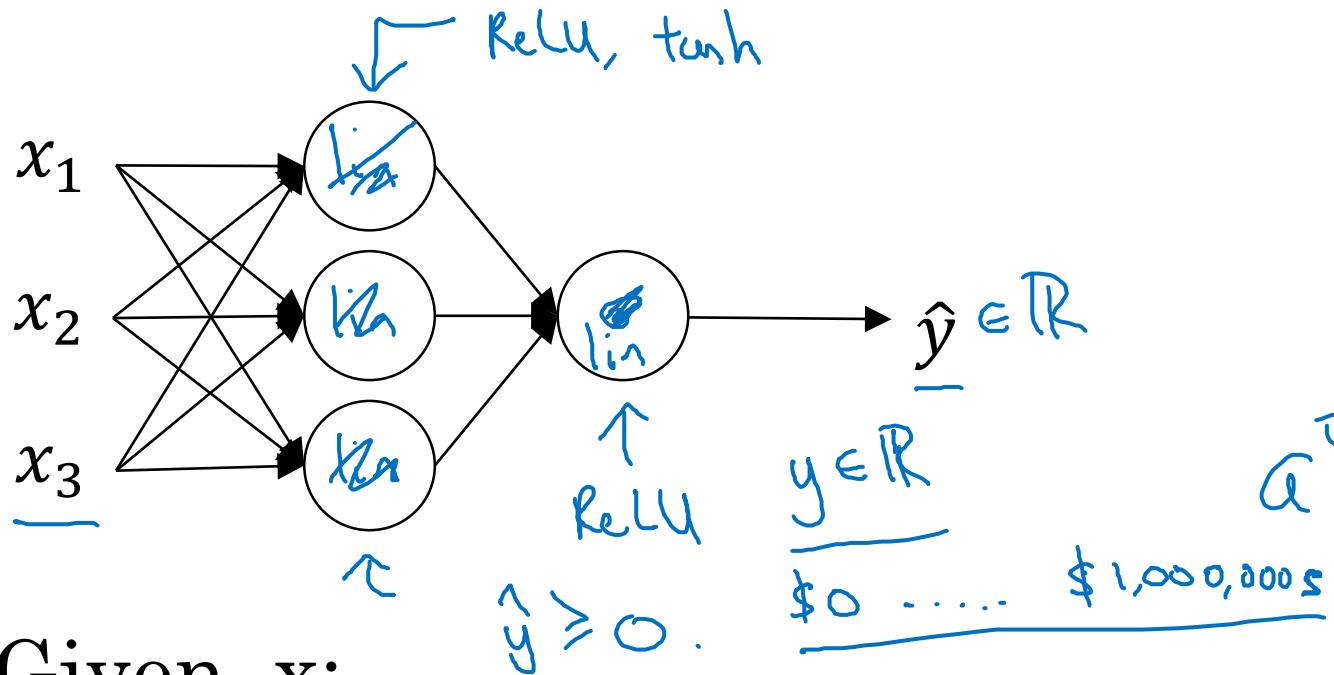
deeplearning.ai

# One hidden layer Neural Network

---

Why do you  
need non-linear  
activation functions?

# Activation function



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

$$\begin{aligned}
 a^{[1]} &= z^{[1]} = \underbrace{W^{[1]}x + b^{[1]}}_{a^{[1]}} \\
 a^{[2]} &= z^{[2]} = \underbrace{W^{[2]}a^{[1]} + b^{[2]}}_{a^{[2]}} \\
 a^{[2]} &= W^{[2]} \left( \underbrace{W^{[1]}x + b^{[1]}}_{a^{[1]}} \right) + b^{[2]} \\
 &= \underbrace{(W^{[2]}W^{[1]})}_{w'} x + \underbrace{(W^{[2]}b^{[1]} + b^{[2]})}_{b'} \\
 &= \underbrace{w'x + b'}_{g(z)} = z
 \end{aligned}$$



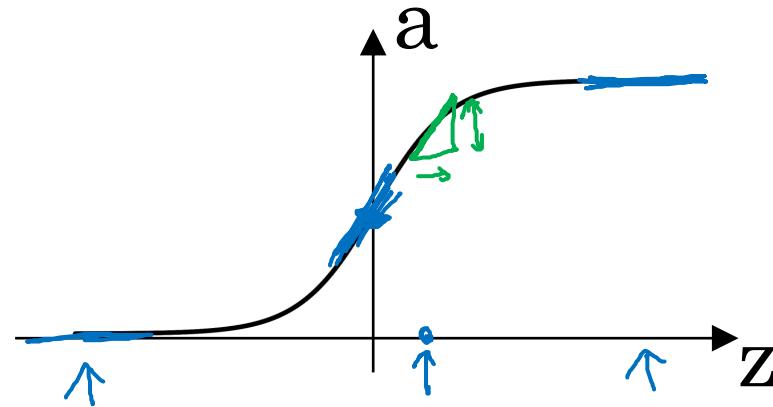
deeplearning.ai

One hidden layer  
Neural Network

---

Derivatives of  
activation functions

# Sigmoid activation function



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

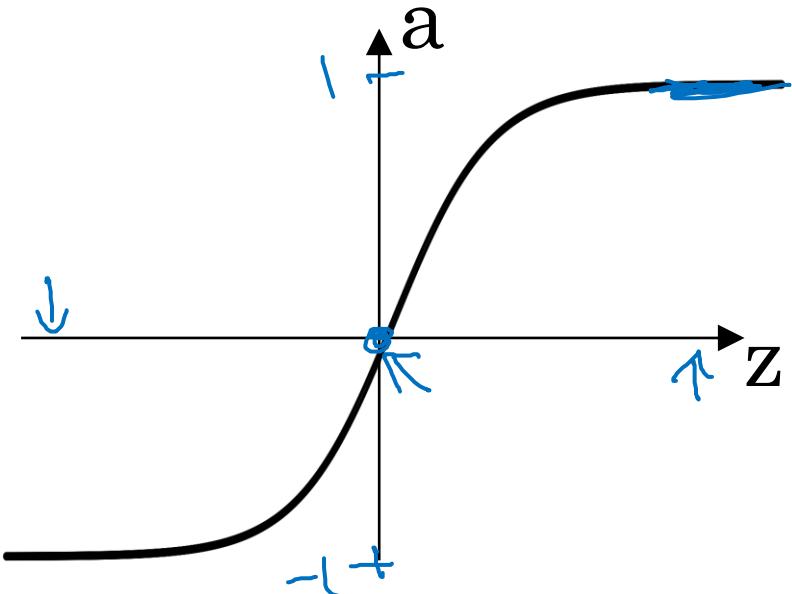
$$a = g(z) = \frac{1}{1 + e^{-z}}$$

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

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

Andrew Ng

# Tanh activation function



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

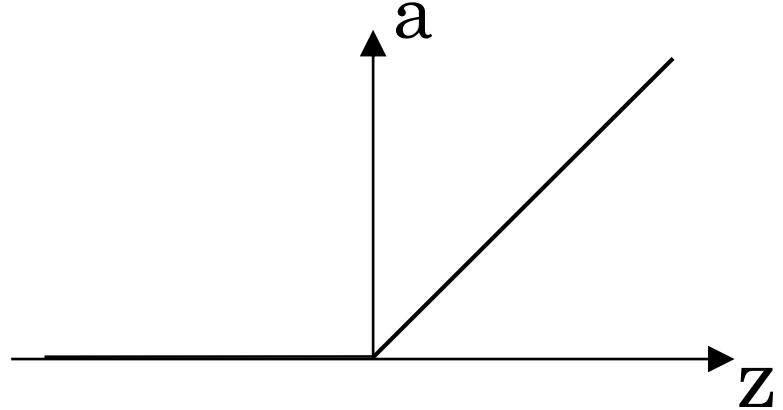
$$= \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

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

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

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

# ReLU and Leaky ReLU



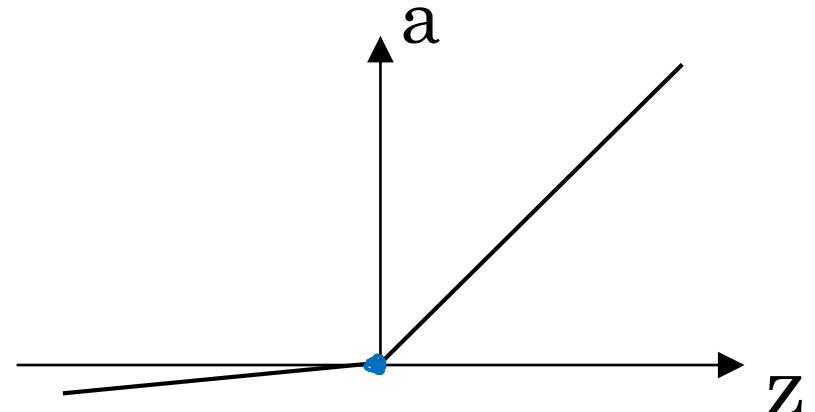
ReLU

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

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

~~undefined if  $z=0$~~

$\bar{z} = 0.0000\ldots 0$



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

Cost function:  $J(\underbrace{\omega^{[1]}, b^{[1]}, \dots, \omega^{[L]}, b^{[L]}}_{\nabla \omega^{[2]}}, \nabla b^{[2]}) = \frac{1}{m} \sum_{i=1}^m l(\hat{y}_i, y_i)$

Gradient Descent:

→ Repeat {

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

$$\frac{\partial J}{\partial \omega^{[1]}} = \frac{\partial J}{\partial \omega^{[1]}} , \quad \frac{\partial J}{\partial b^{[1]}} = \frac{\partial J}{\partial b^{[1]}} , \dots$$

$$\omega^{[1]} := \omega^{[1]} - \alpha \frac{\partial J}{\partial \omega^{[1]}}$$

$$b^{[1]} := b^{[1]} - \alpha \frac{\partial J}{\partial b^{[1]}}$$

↳

# Formulas for computing derivatives

Forward propagation:

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

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

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

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

Back propagation:

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

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

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

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

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

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

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

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

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



deeplearning.ai

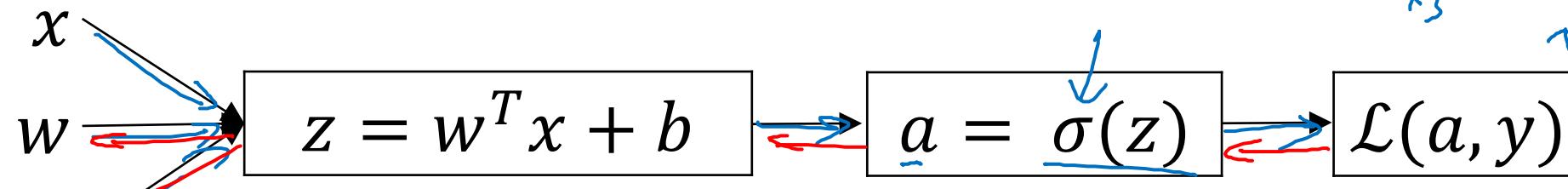
One hidden layer  
Neural Network

---

Backpropagation  
intuition (Optional)

# Computing gradients

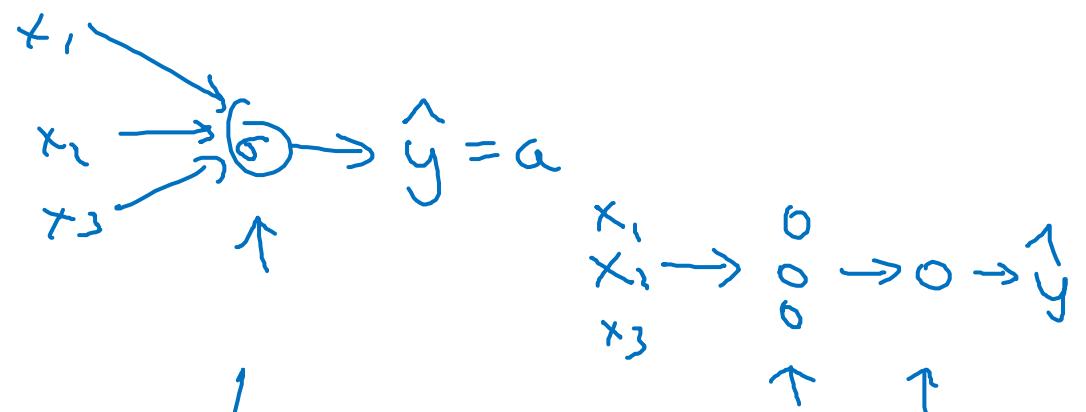
## Logistic regression



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

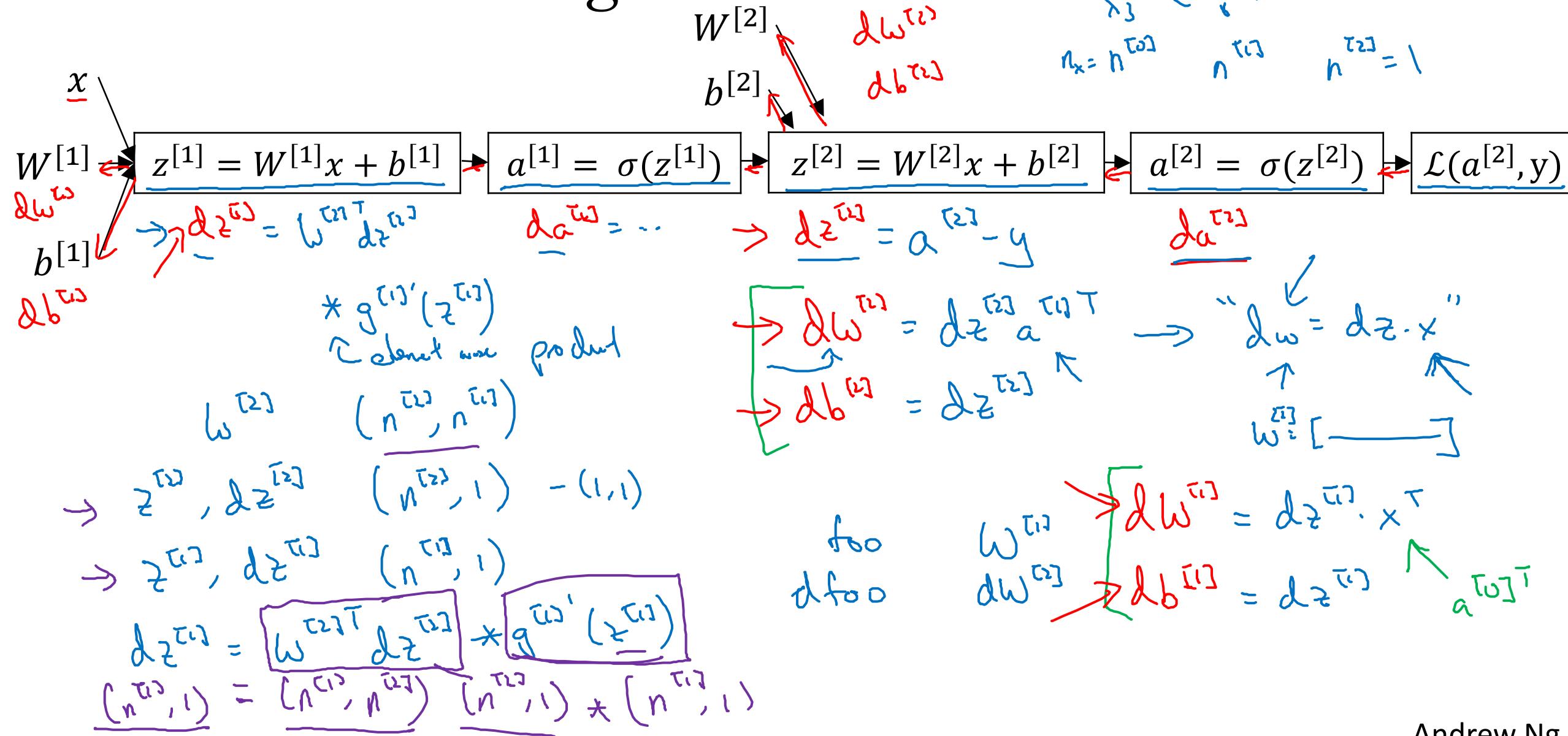
Handwritten annotations:

- $\frac{\partial z}{\partial w} = x$
- $\frac{\partial z}{\partial b} = 1$
- $\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial a} \cdot g'(z)$
- $g(z) = \sigma(z)$
- $\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial a} \cdot \frac{\partial a}{\partial z}$
- " $\frac{\partial z}{\partial w}$ " = " $\frac{\partial a}{\partial z}$ "
- $\frac{\partial}{\partial z} g(z) = g'(z)$



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

# Neural network gradients



# Summary of gradient descent

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

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

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

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

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

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

Vectorized implementation:

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

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

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

# Summary of gradient descent

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

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

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

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

(n<sup>T<sub>2</sub></sup>, 1)

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

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

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

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

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

$$dZ^{[1]} = \underbrace{W^{[2]T} dZ^{[2]}}_{(n^{T_2}, m)} * \underbrace{g^{[1]'}(Z^{[1]})}_{\text{elementwise product}} \quad (n^{T_1}, m)$$

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

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

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



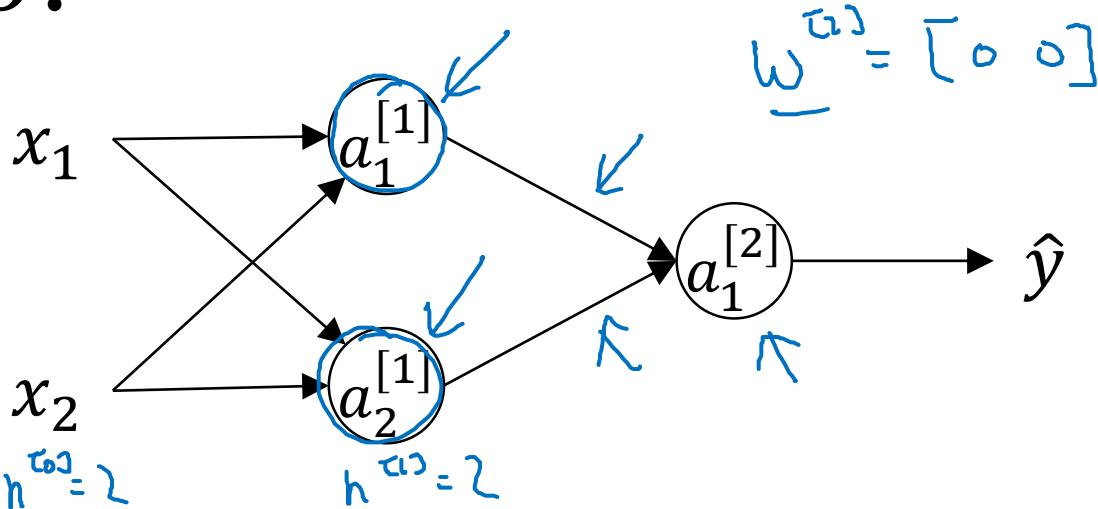
deeplearning.ai

One hidden layer  
Neural Network

---

Random Initialization

# What happens if you initialize weights to zero?



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

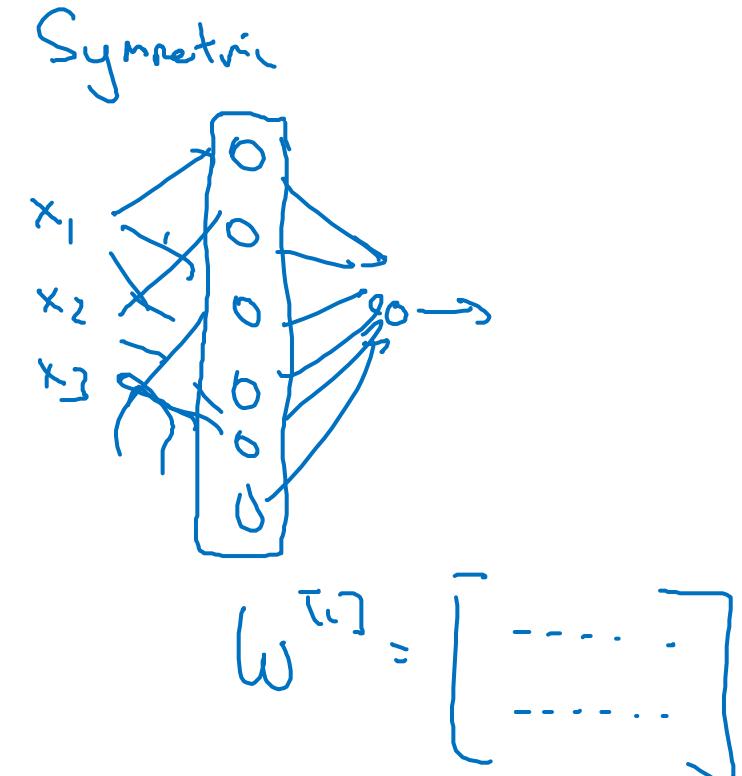
$$a_1^{[1]} = a_2^{[1]}$$

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

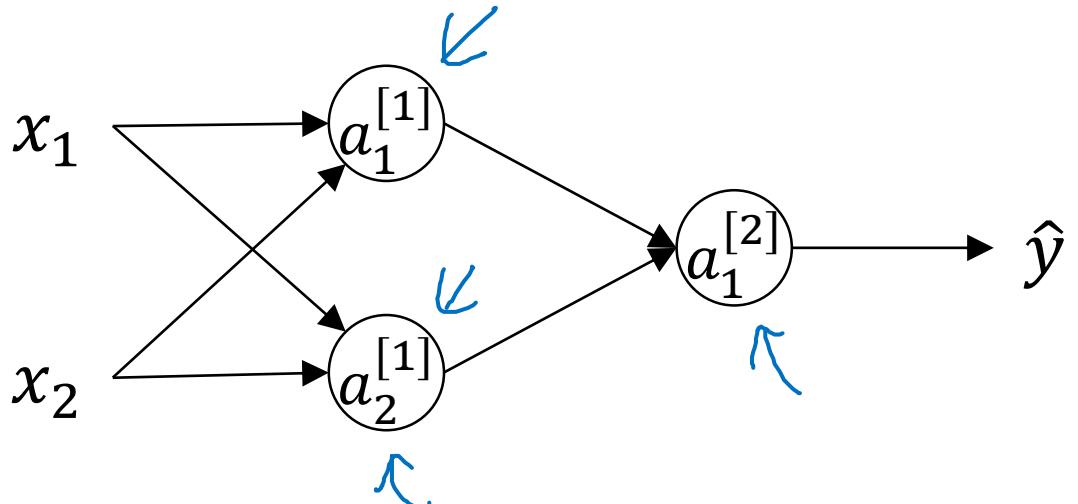
$$b^{[1]} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

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

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

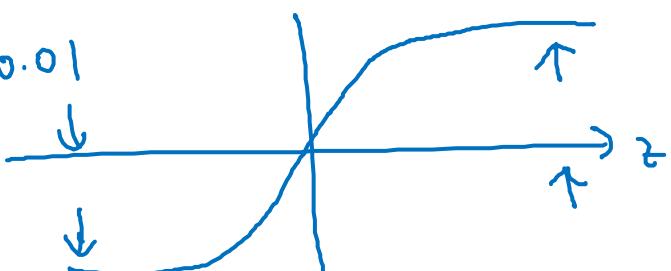


# Random initialization



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

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





deeplearning.ai

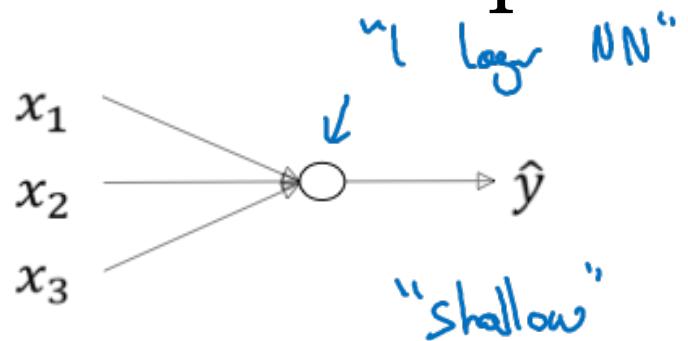
# Deep Neural Networks

---

## Deep L-layer Neural network

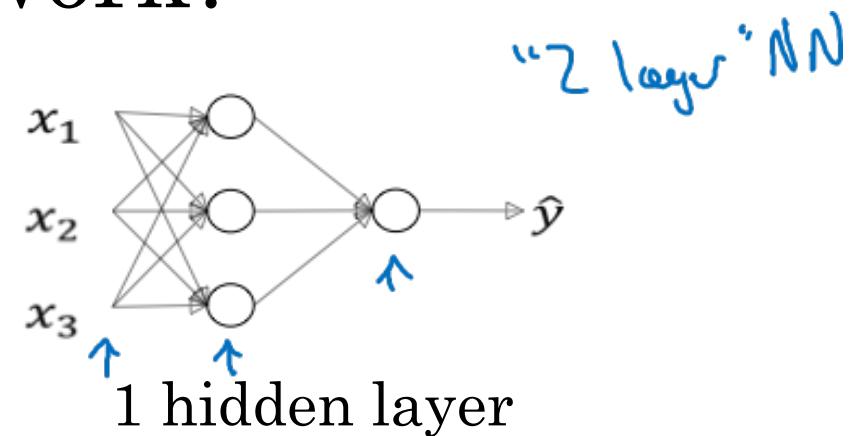
---

# What is a deep neural network?

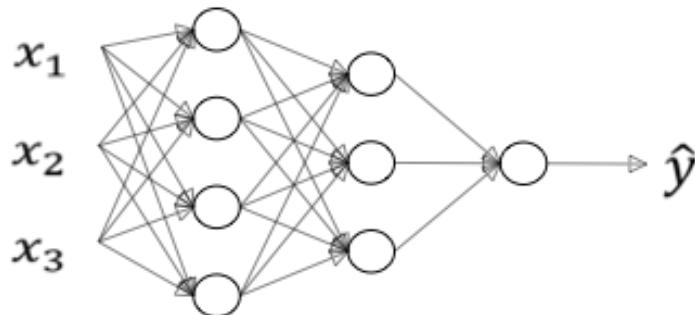


*"Shallow"*

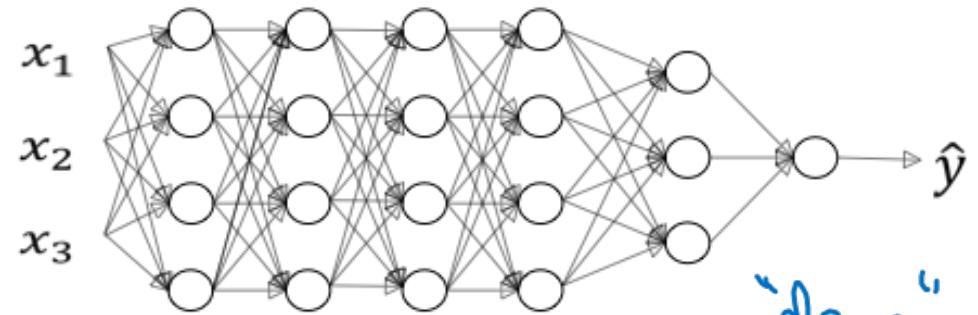
logistic regression



1 hidden layer



2 hidden layers

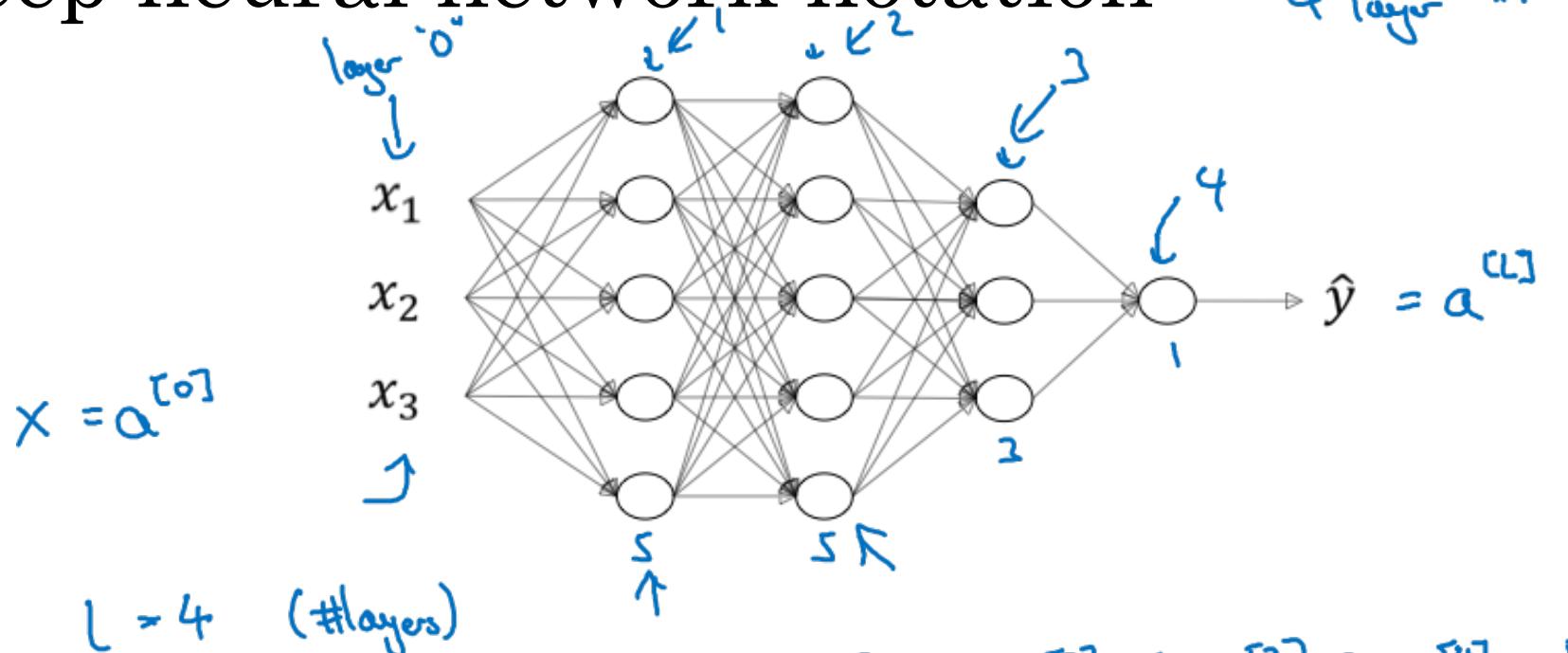


5 hidden layers

*"deep"*

Andrew

# Deep neural network notation



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

$$n^{[4]}=n_y=1$$

Andrew  
Ng



deeplearning.ai

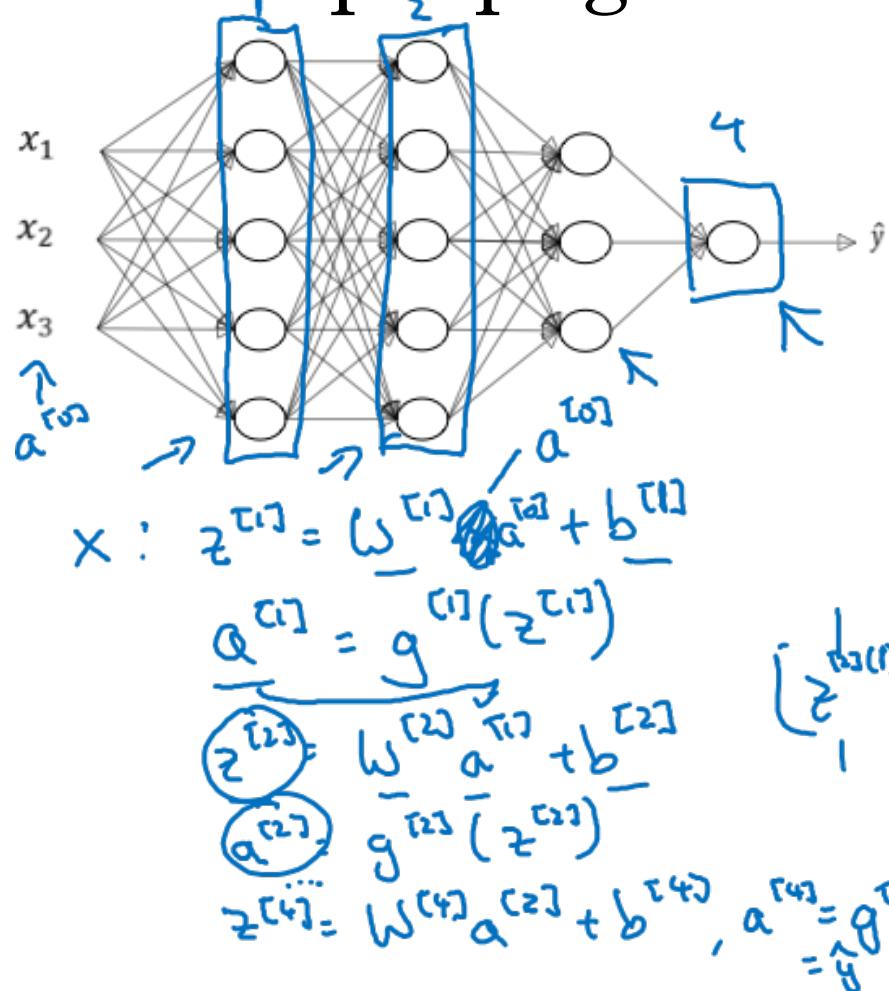
# Deep Neural Networks

---

## Forward Propagation in a Deep Network

---

# Forward propagation in a deep network



$A^{[0]} = X$

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

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

Vertikal:

$\rightarrow z^{[1]} = (W^{[1]} \otimes A^{[0]} + b^{[1]})$  for  $l=1..4$

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

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

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

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

Andrew



deeplearning.ai

# Deep Neural Networks

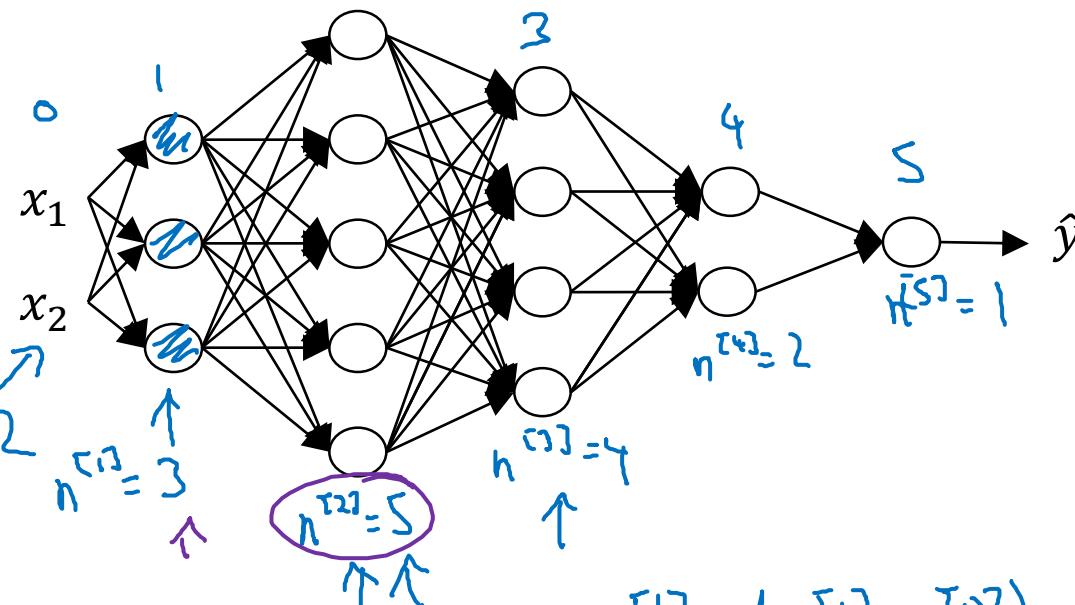
---

## Getting your matrix dimensions right

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

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

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



$$z^{[1]} = \underbrace{W^{[1]} \cdot x}_{(3,1) \leftarrow (3,2) \frac{(2,1)}{(n^{[1]}, 1)}} + \underbrace{b^{[1]}_{(3,1)}}_{(n^{[1]}, 1)}$$

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

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

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

$$z^{[2]} = \underbrace{W^{[2]} \cdot a^{[1]}}_{\rightarrow (5,1)} + \underbrace{b^{[2]}_{(5,1)}}_{(n^{[2]}, 1)}$$

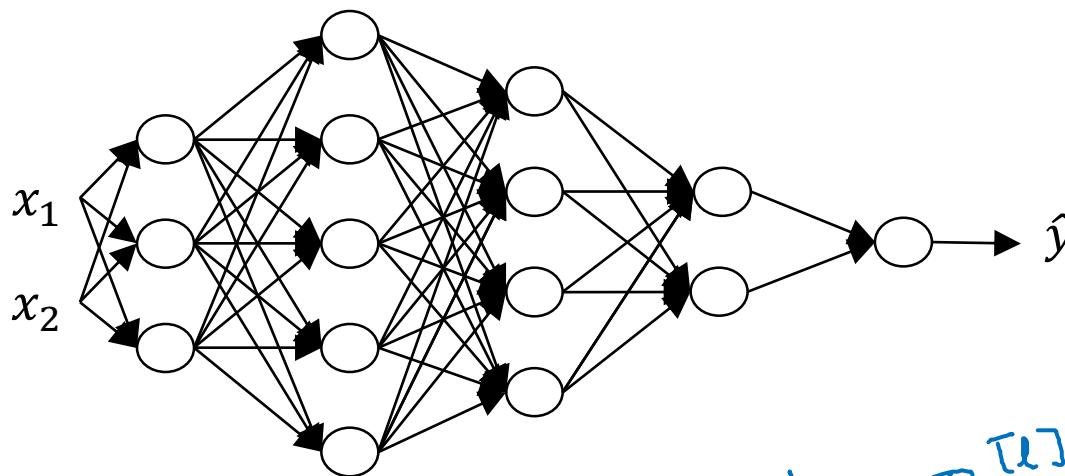
$$W^{[3]}: (4, 5)$$

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

$$l = 5$$

$$\begin{cases} W^{[l]}: (n^{[l]}, n^{[l-1]}) \\ b^{[l]}: (n^{[l]}, 1) \\ dW^{[l]}: (n^{[l]}, n^{[l-1]}) \\ db^{[l]}: (n^{[l]}, 1) \end{cases}$$

# Vectorized implementation



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

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

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

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

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

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

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

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

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



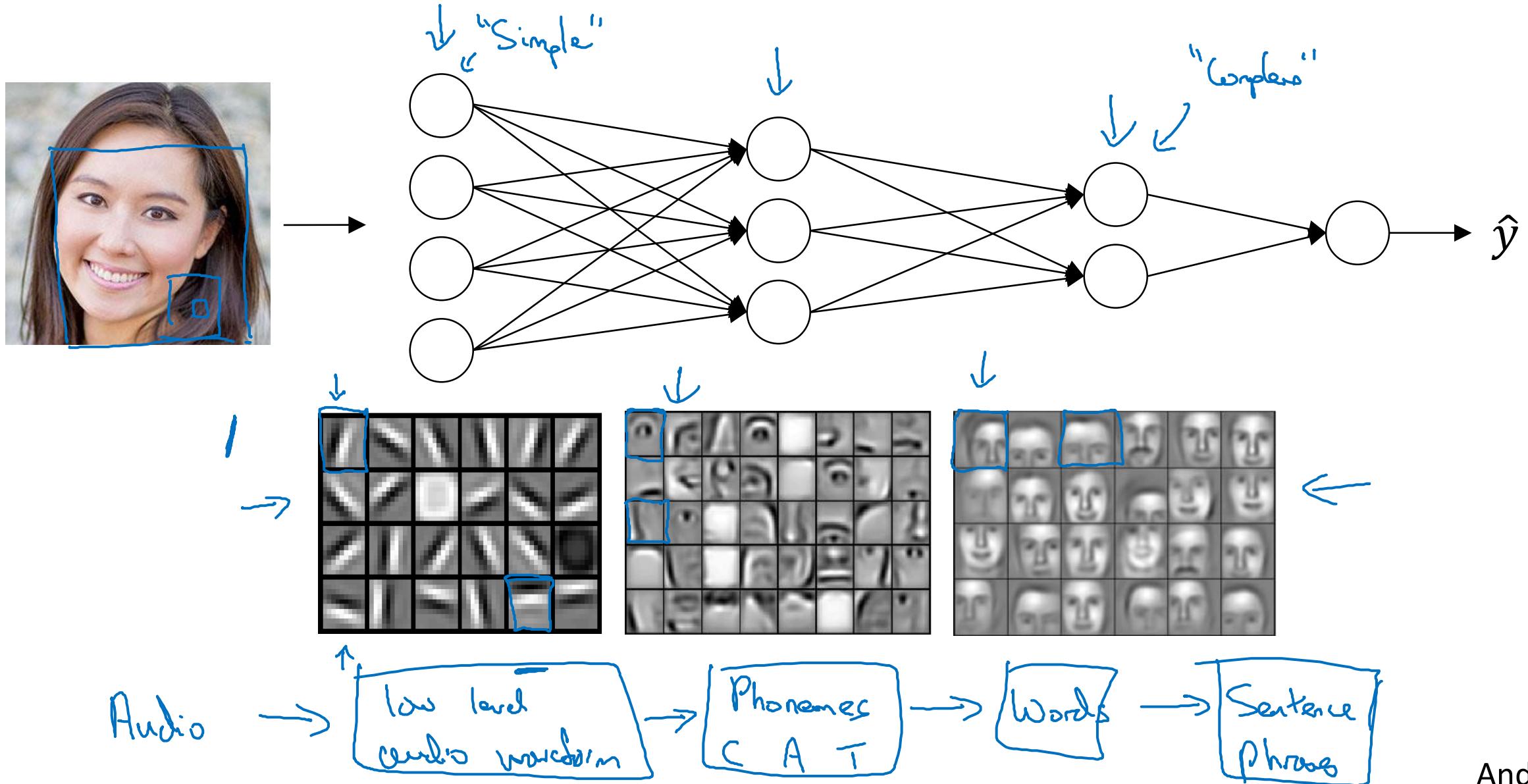
deeplearning.ai

# Deep Neural Networks

---

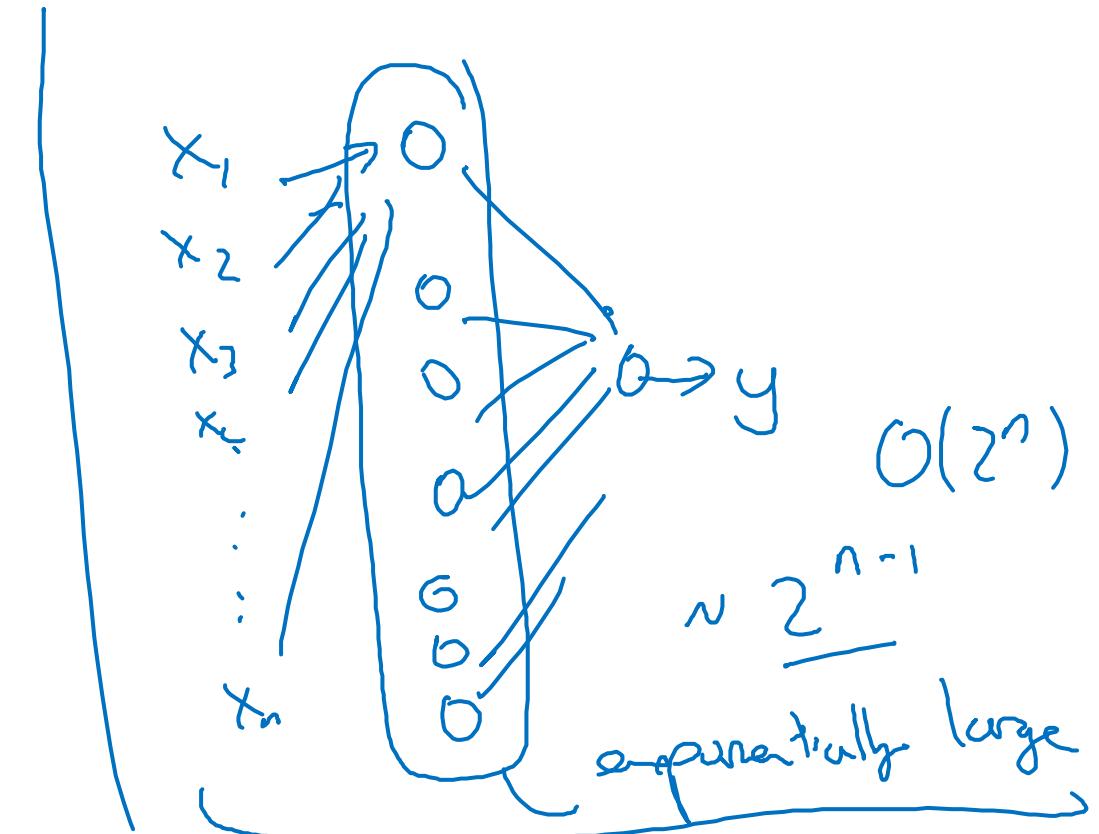
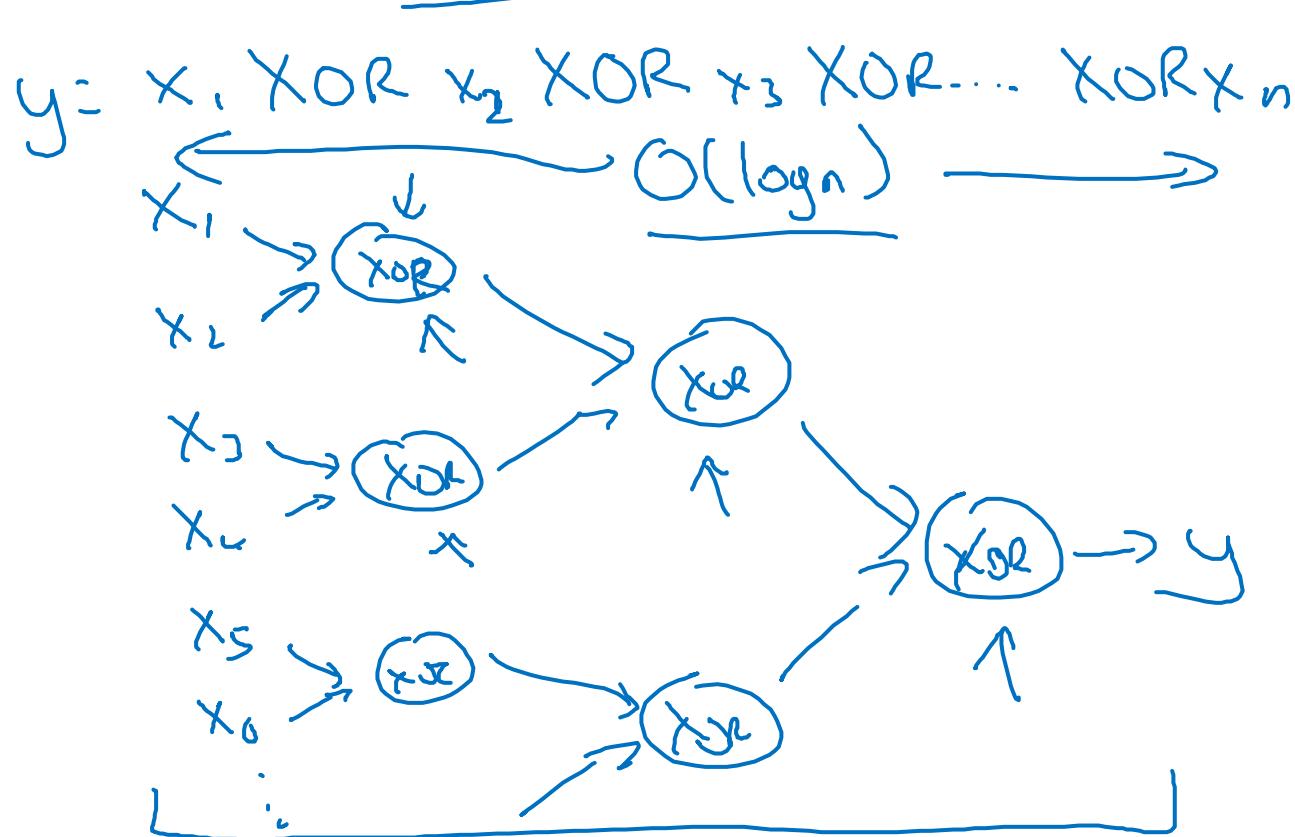
## Why deep representations?

# Intuition about deep representation



# Circuit theory and deep learning

Informally: There are functions you can compute with a “small” L-layer deep neural network that shallow networks require exponentially more hidden units to compute.





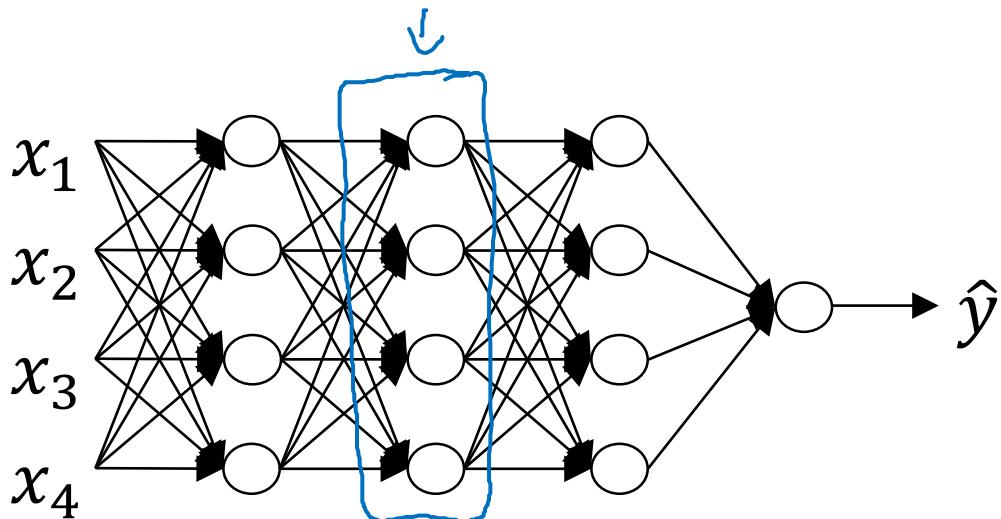
deeplearning.ai

# Deep Neural Networks

---

Building blocks of  
deep neural networks

# Forward and backward functions



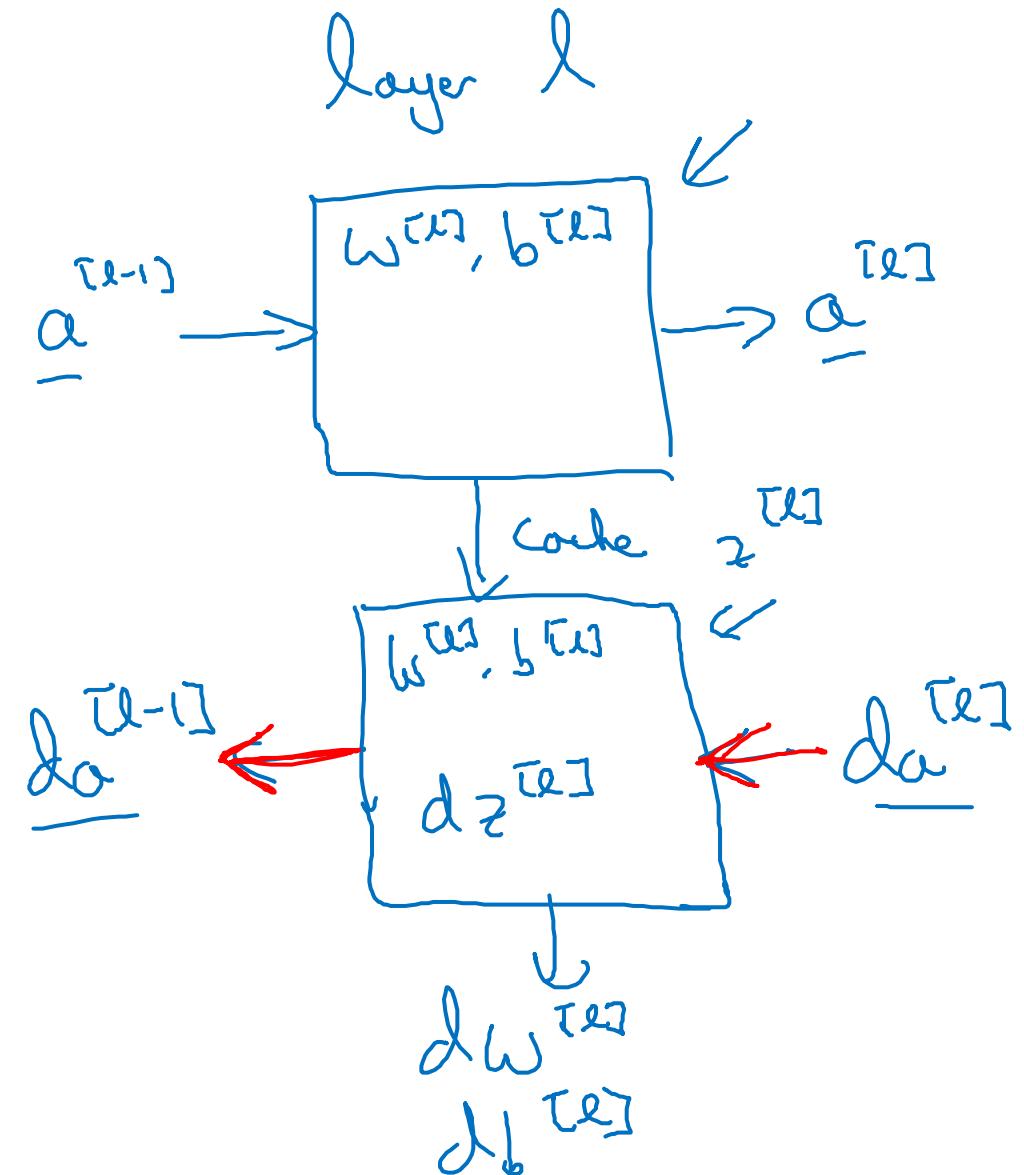
Layer  $l$ :  $w^{[l]}, b^{[l]}$

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

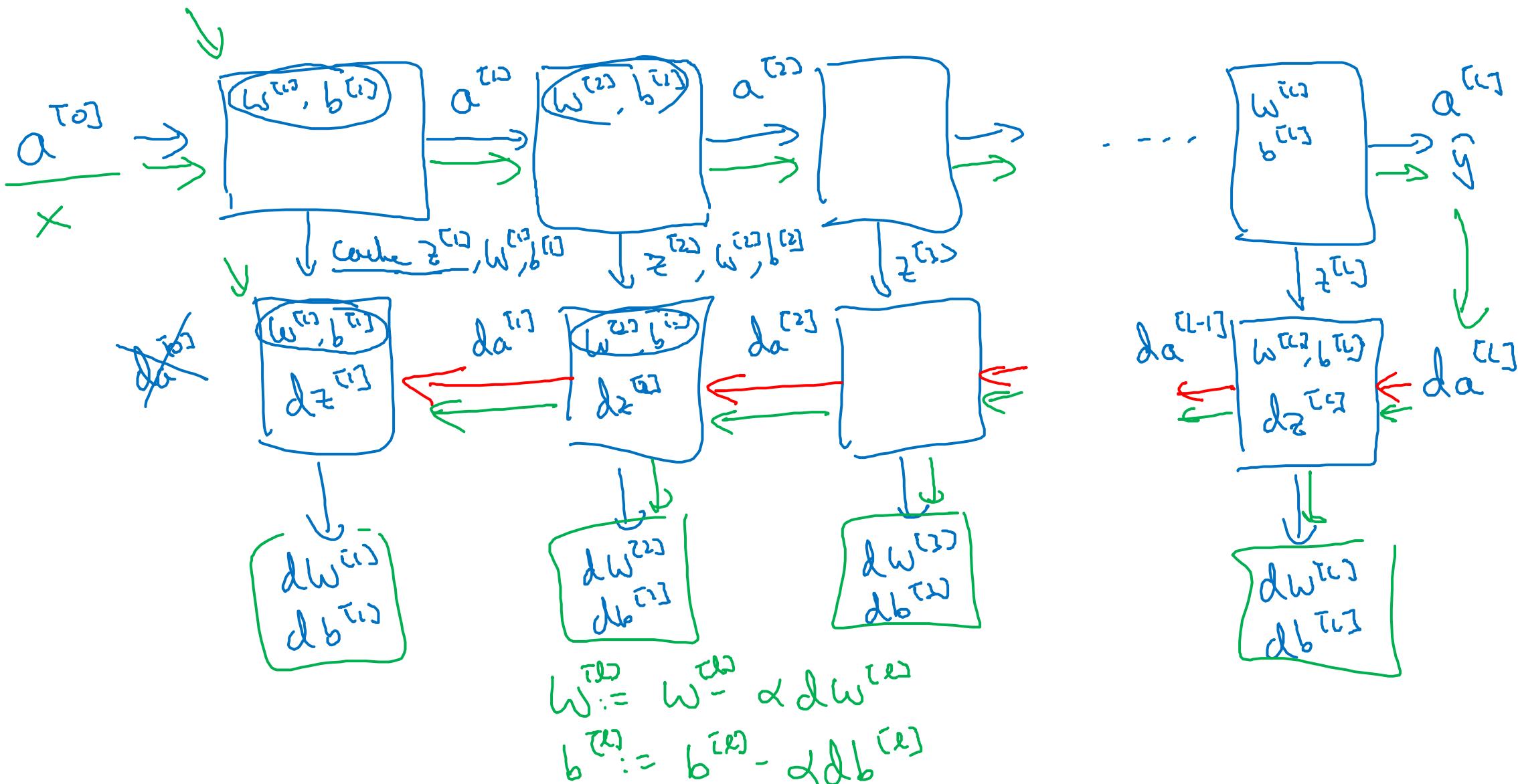
$$\underline{z}^{[l]} = w^{[l]} \underline{a}^{[l-1]} + b^{[l]}$$

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

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



# Forward and backward functions





deeplearning.ai

# Deep Neural Networks

---

## Forward and backward propagation

# Forward propagation for layer $l$

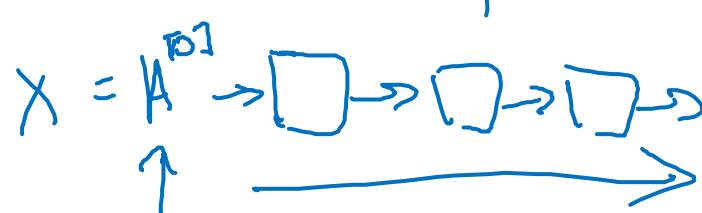
→ Input  $a^{[l-1]} \leftarrow$

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

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

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

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



Vertwijf:

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

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

# Backward propagation for layer $l$

→ Input  $da^{[l]}$

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

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

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

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

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

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

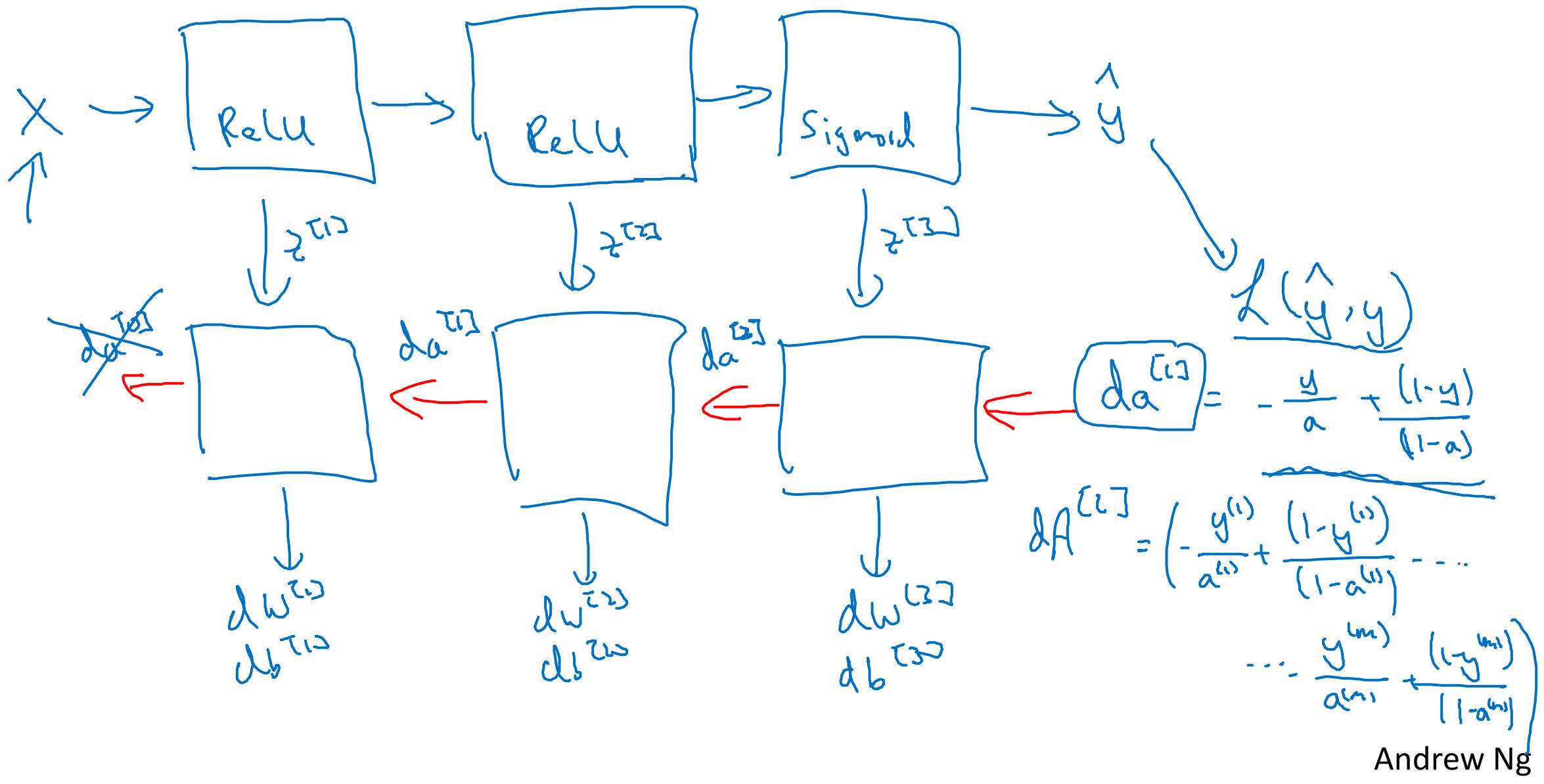
$$dz^{[l]} = da^{[l]} * g^{[l]'}(z^{[l]})$$

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

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

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

# Summary





deeplearning.ai

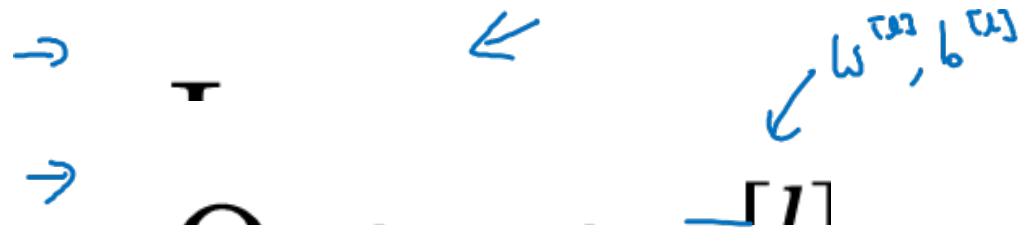
# Deep Neural Networks

---

## Forward and backward propagation

---

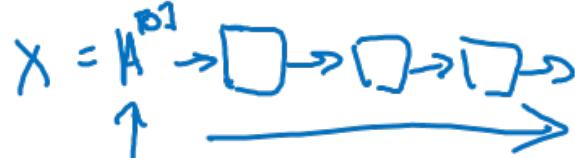
# Forward propagation for layer $l$



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

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

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



Verteilg:

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

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

# Backward propagation for layer $l$

→      ↘

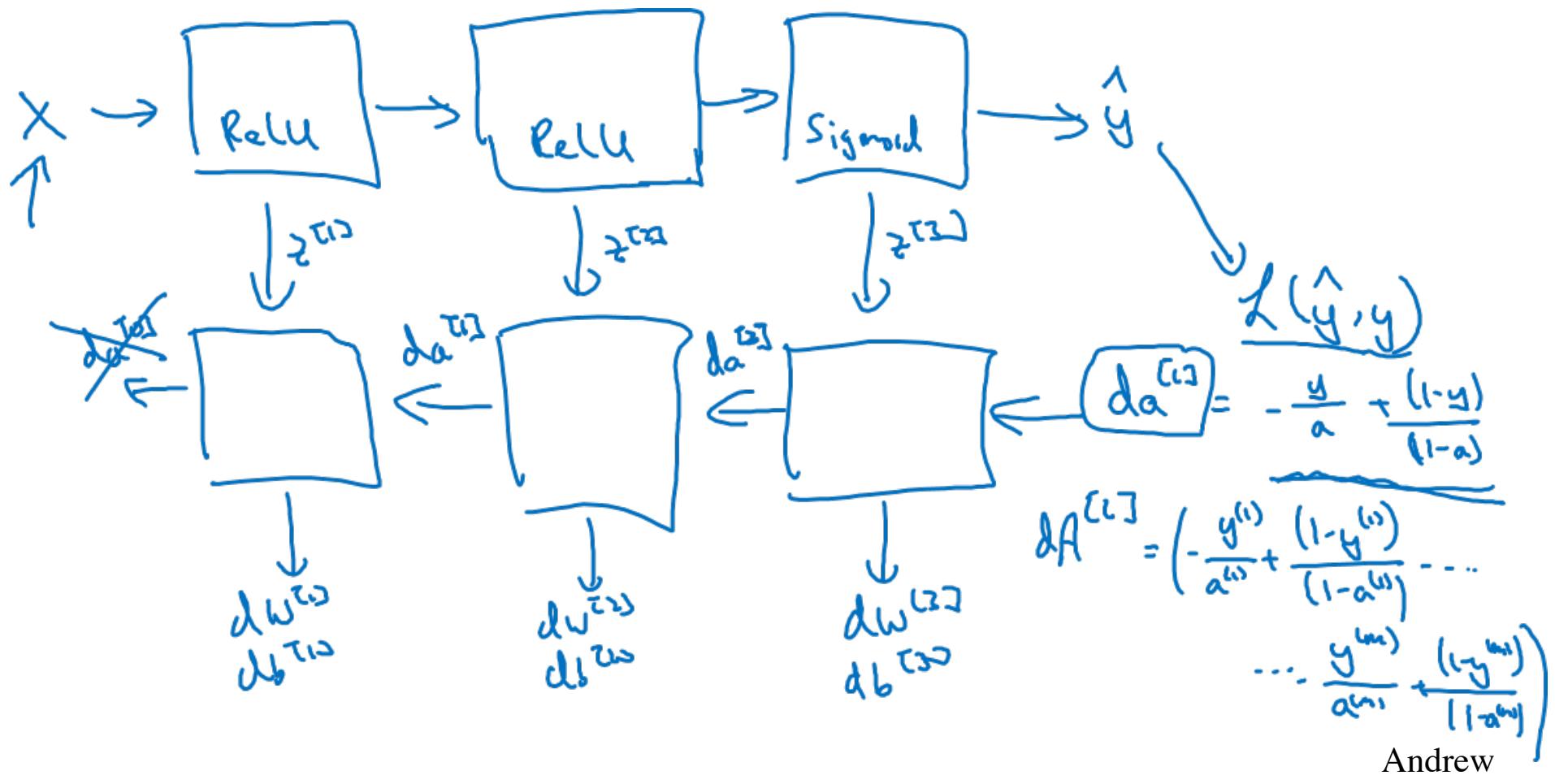
$$\begin{aligned}
 dz^{[l]} &= \text{do} \circ g^{[l+1]'}(z^{[l+1]}) \\
 dw^{[l]} &= dz^{[l]} \cdot a^{[l-1]} \\
 db^{[l]} &= dz^{[l]} \\
 da^{[l-1]} &= w^{[l]T} \cdot dz^{[l]} \\
 dz^{[l]} &= w^{[l+1]T} dz^{[l+1]} \star g^{[l+1]'}(z^{[l+1]}) \\
 \end{aligned}$$

↓

$$\begin{aligned}
 dz^{[l]} &= dA^{[l+1]} \star g^{[l+1]'}(z^{[l+1]}) \\
 dw^{[l]} &= \frac{1}{m} dz^{[l]} \cdot A^{[l-1]T} \\
 db^{[l]} &= \frac{1}{m} np \cdot \text{sum}(dz^{[l]}, \text{axis}=1, \text{keepdims=True}) \\
 dA^{[l-1]} &= w^{[l]T} \cdot dz^{[l]}
 \end{aligned}$$

Andrew

# Summary



Andrew



deeplearning.ai

# Deep Neural Networks

---

## Parameters vs Hyperparameters

# What are hyperparameters?

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

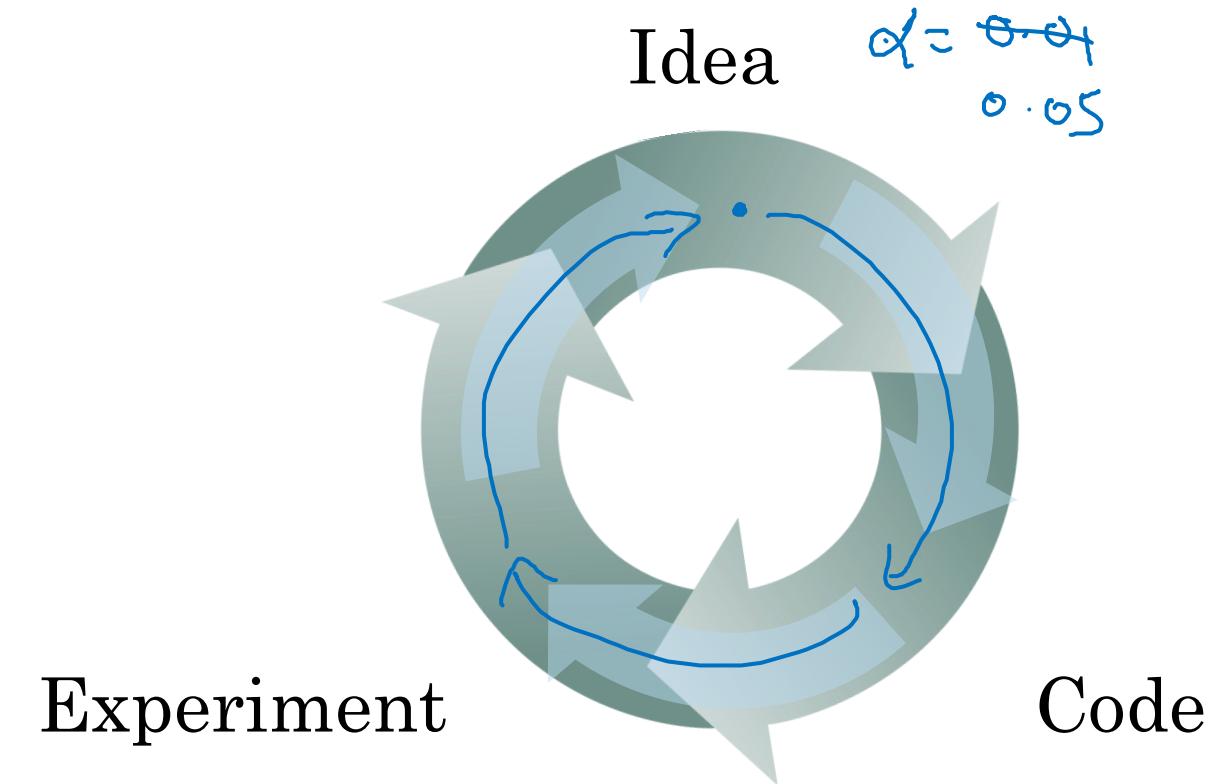
Hyperparameters:

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

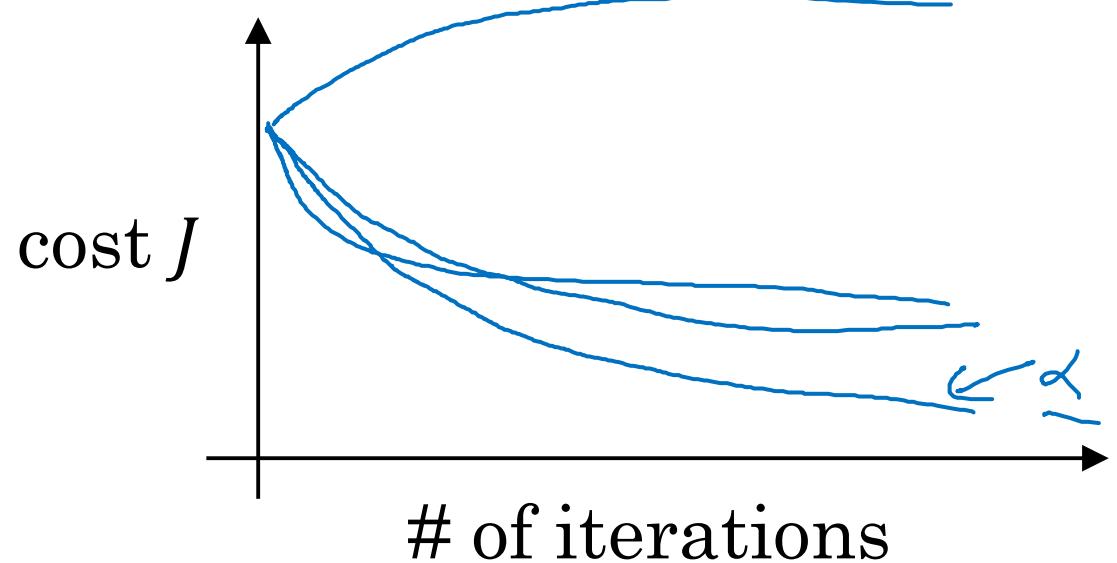
Curly brace groups the last four items.

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

# Applied deep learning is a very empirical process



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





deeplearning.ai

# Deep Neural Networks

---

What does this  
have to do with  
the brain?

# Forward and backward propagation

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

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

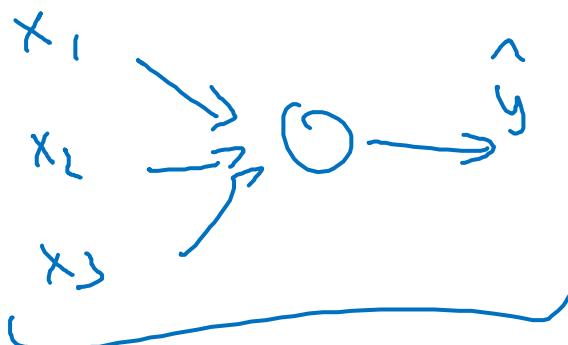
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

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

:

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

"It's like the brain"



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

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

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

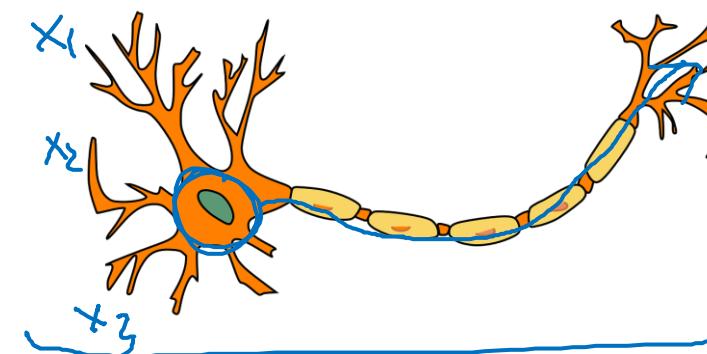
$$dZ^{[L-1]} = dW^{[L]T} dZ^{[L]} g'^{[L]}(Z^{[L-1]})$$

:

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

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

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





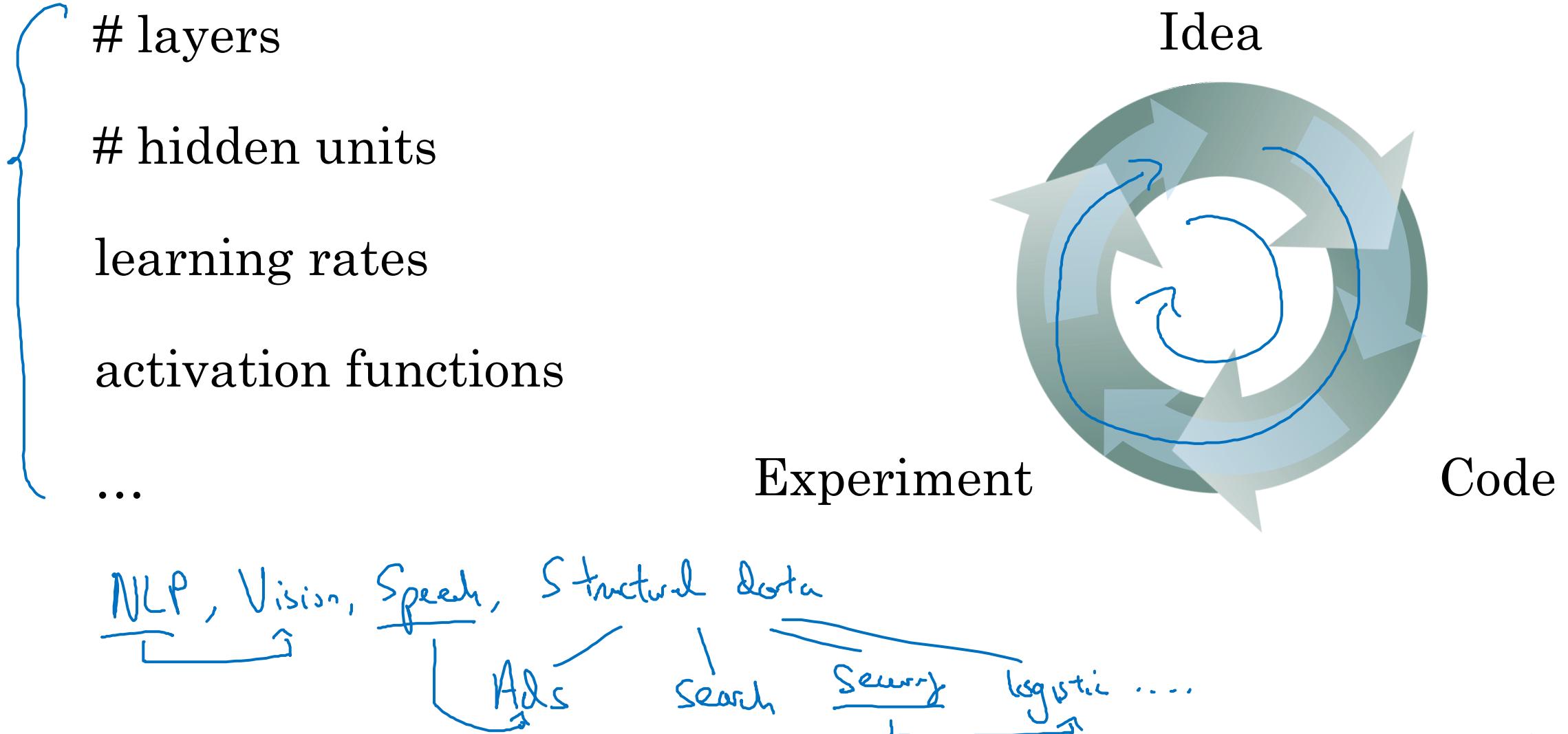
deeplearning.ai

Setting up your  
ML application

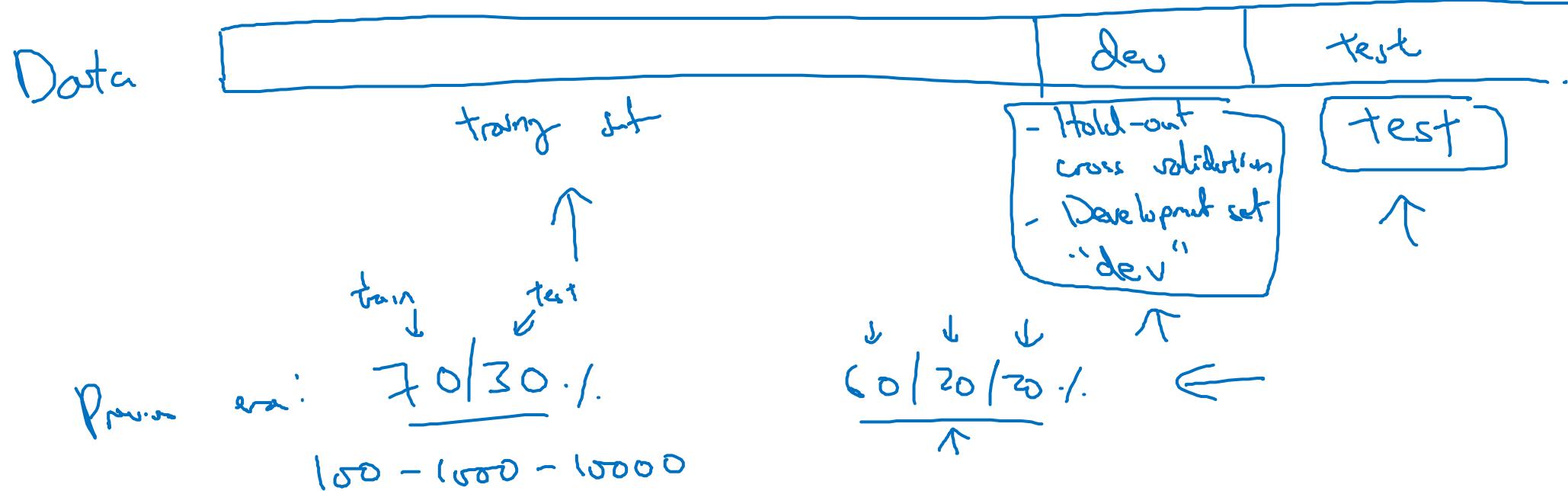
---

Train/dev/test  
sets

# Applied ML is a highly iterative process



# Train/dev/test sets



Big data! 1,000,000

10,000 10,000

98 / 1 / 1 %.

99.5 { 25 / 25 %.

·4 { -1 -1 .

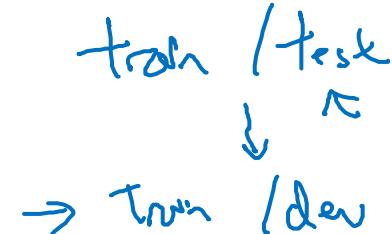
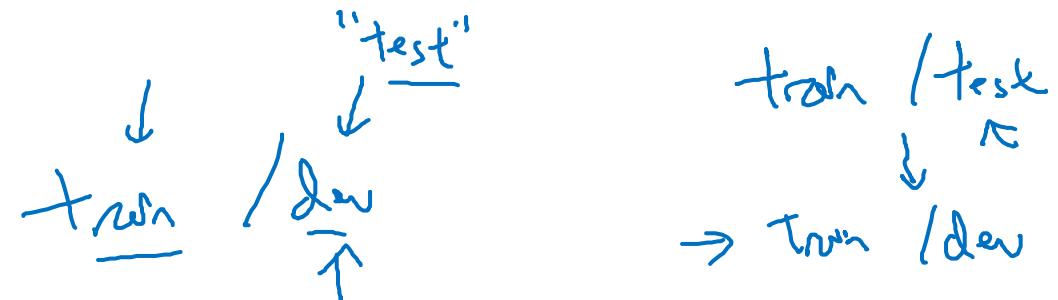
# Mismatched train/test distribution

Conts

Training set:  
Cat pictures from }  
webpages

Dev/test sets:  
Cat pictures from }  
users using your app

→ Make sure dev and test come from same distribution.



Not having a test set might be okay. (Only dev set.)



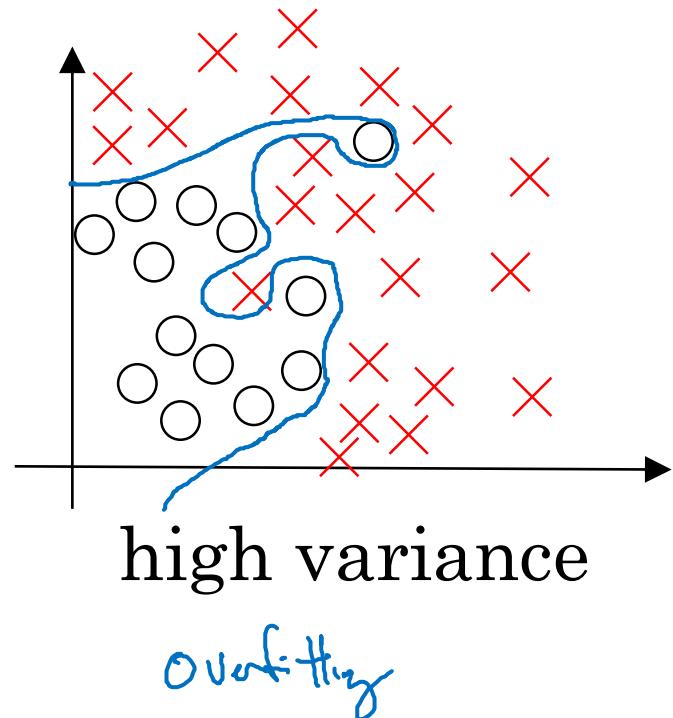
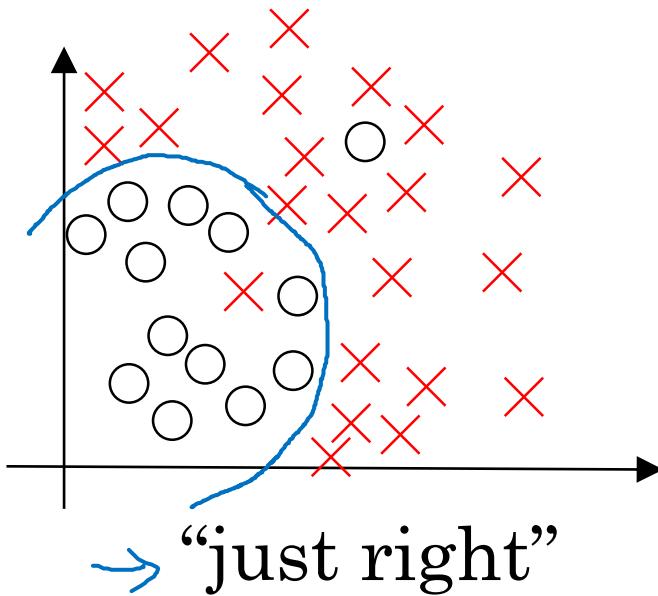
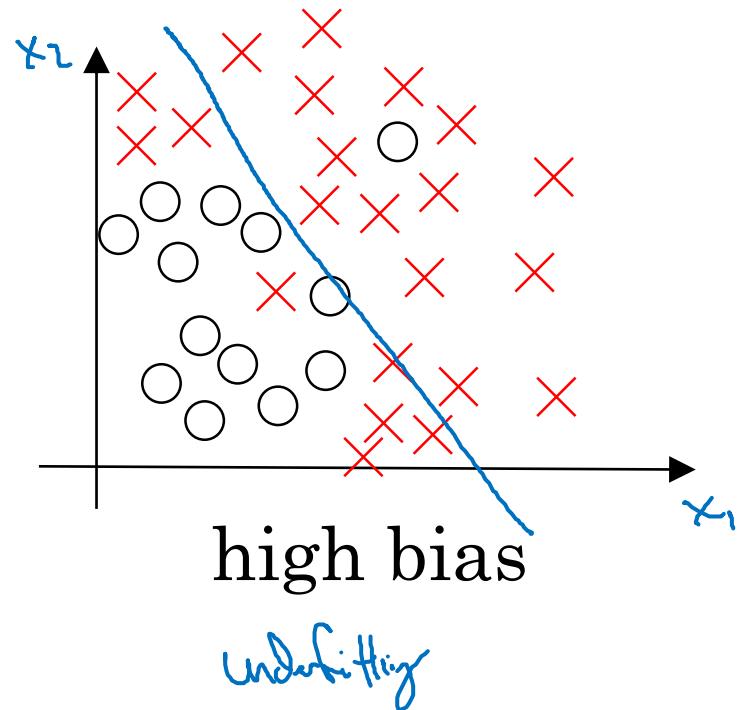
deeplearning.ai

Setting up your  
ML application

---

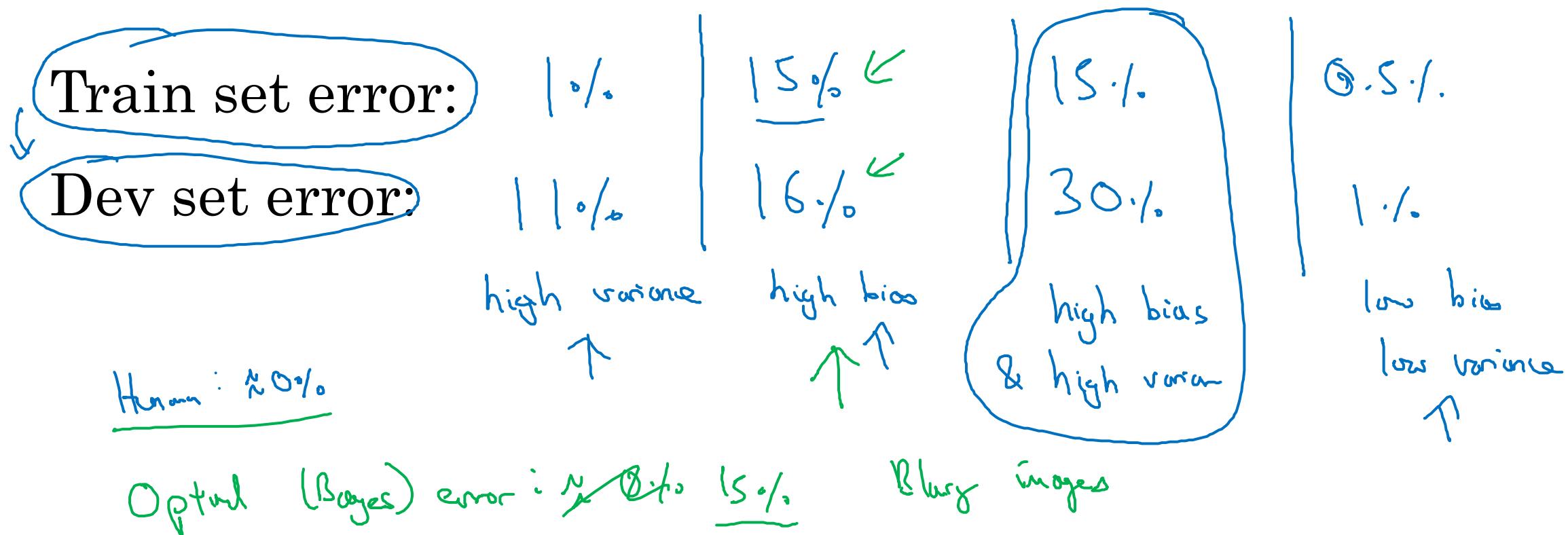
Bias/Variance

# Bias and Variance

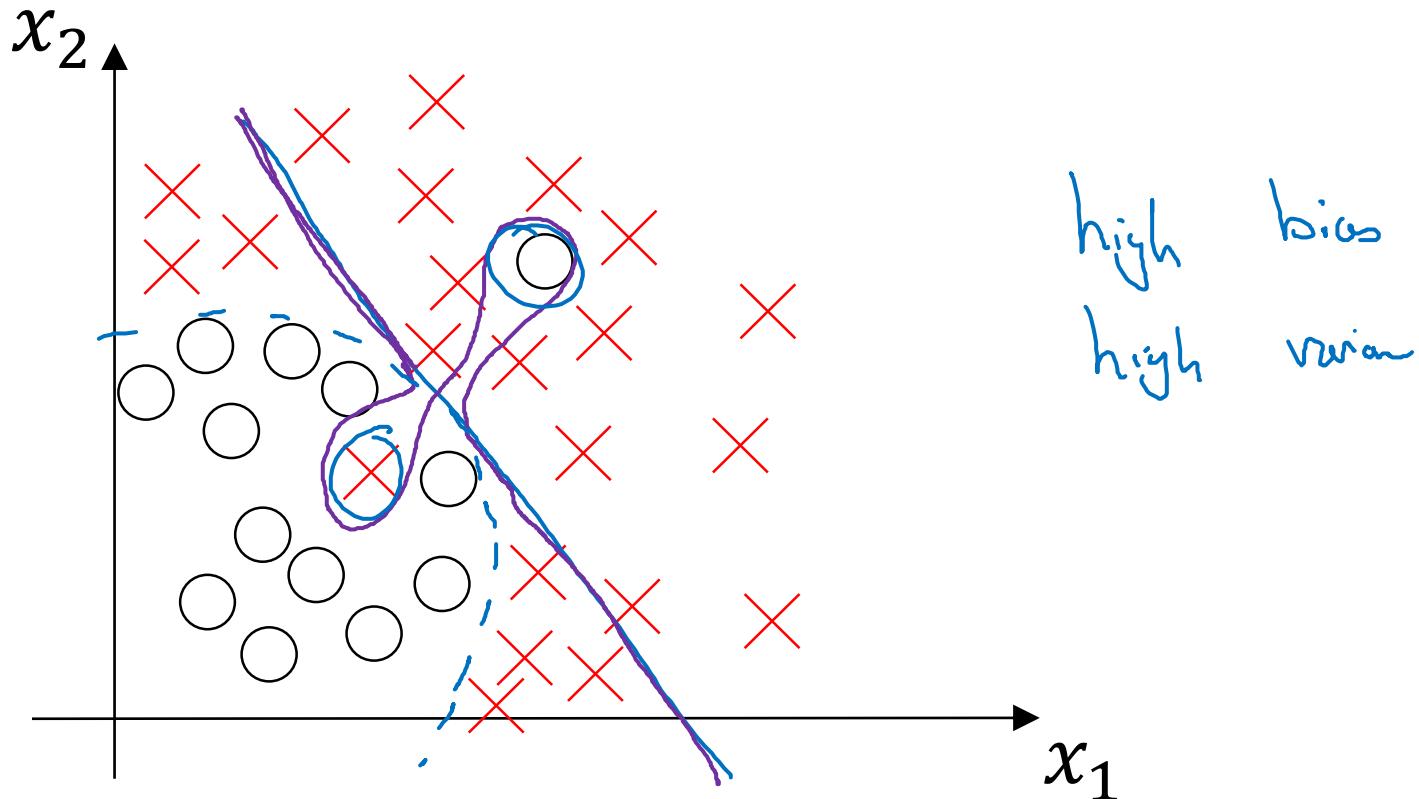


# Bias and Variance

Cat classification



# High bias and high variance





deeplearning.ai

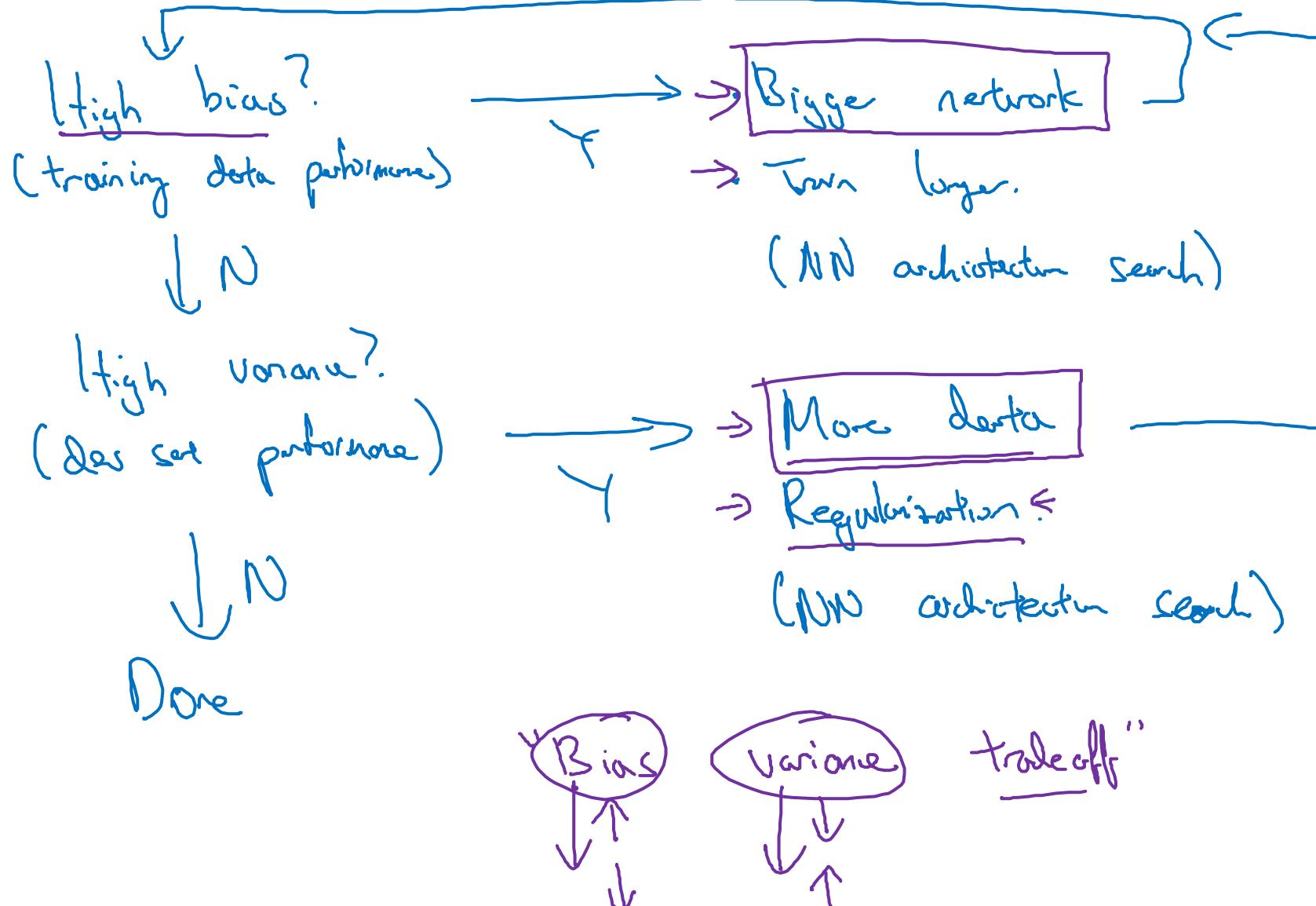
# Setting up your ML application

---

## Basic “recipe” for machine learning

# Basic “recipe” for machine learning

# Basic recipe for machine learning





deeplearning.ai

## Setting up your ML application

---

Basic “recipe”  
for machine learning

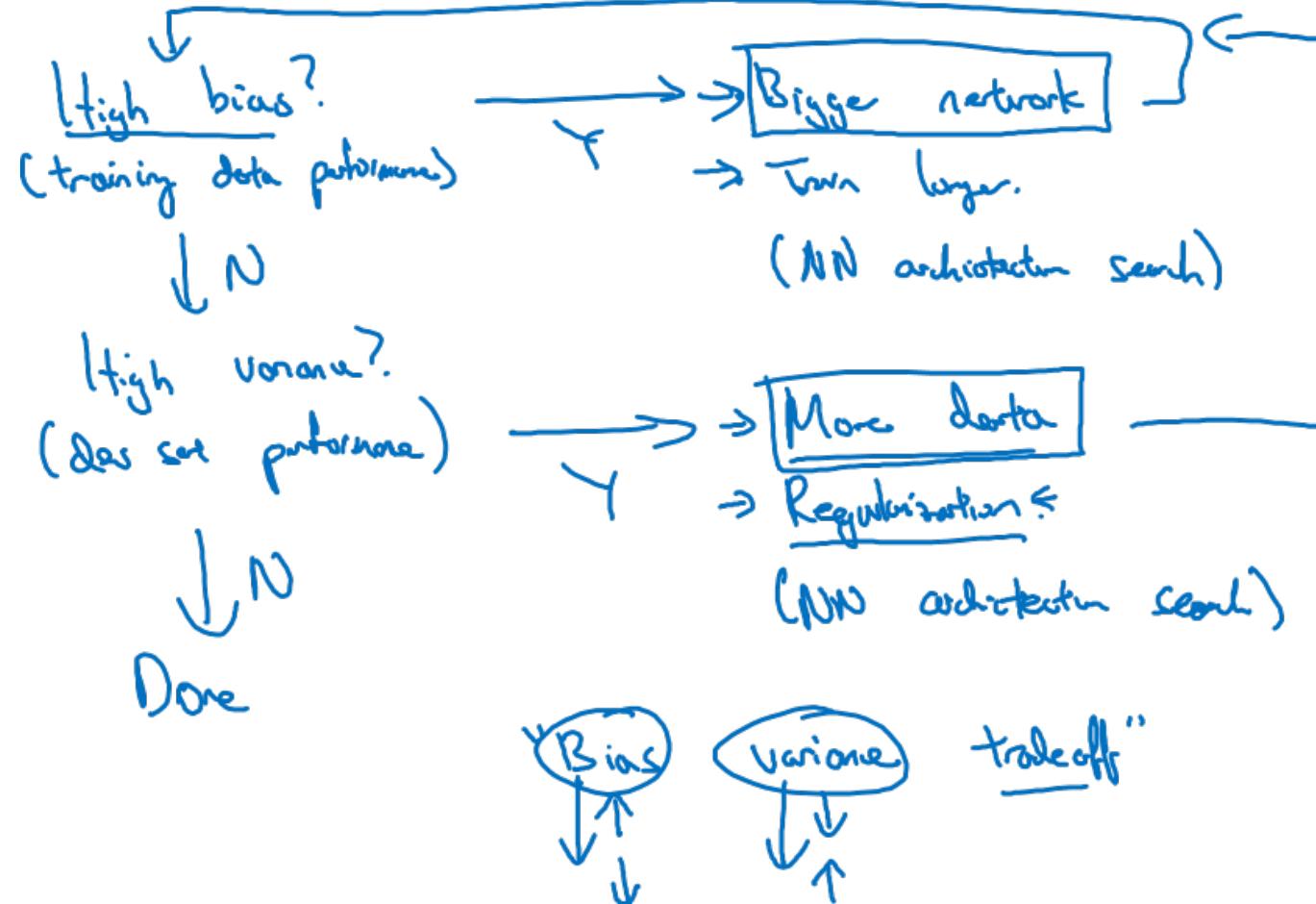
---

# Basic “recipe” for machine learning

Andrew  
Ng

---

# Basic recipe for machine learning



Andrew  
x



deeplearning.ai

Regularizing your  
neural network

---

Regularization

# Logistic regression

$$\min_{w,b} J(w, b)$$

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

$\lambda$  = regularization parameter  
lambda lambd

$$J(w, b) = \underbrace{\frac{1}{m} \sum_{i=1}^m \ell(\hat{y}^{(i)}, y^{(i)})}_{\text{L}_2 \text{ regularization}} + \frac{\lambda}{2m} \|w\|_2^2$$

$$+ \cancel{\frac{\lambda}{2m} b^2}$$

omit

$$\|w\|_2^2 = \sum_{j=1}^{n_x} w_j^2 = w^T w \leftarrow$$

L<sub>1</sub> regularization

$$\frac{\lambda}{2m} \sum_{j=1}^{n_x} |w_j| = \frac{\lambda}{2m} \|w\|_1$$

w will be sparse

# Neural network

$$\rightarrow J(w^{(1)}, b^{(1)}, \dots, w^{(L)}, b^{(L)}) = \underbrace{\frac{1}{m} \sum_{i=1}^m f(\hat{y}^{(i)}, y^{(i)})}_{n^{(1)} \times n^{(L-1)}} + \underbrace{\frac{\lambda}{2m} \sum_{l=1}^L \|w^{(l)}\|_F^2}_{\text{regularization}}$$

$$\|w^{(l)}\|_F^2 = \sum_{i=1}^{n^{(l)}} \sum_{j=1}^{n^{(l-1)}} (w_{ij}^{(l)})^2$$

$w^{(l)}: (n^{(l)}, n^{(l-1)})$

"Frobenius norm"

$$\|\cdot\|_2^2$$

$$\|\cdot\|_F^2$$

$$dW^{(l)} = \boxed{(\text{from backprop}) + \frac{\lambda}{m} w^{(l)}}$$

$$\rightarrow w^{(l)} := w^{(l)} - \alpha dW^{(l)}$$

$$\frac{\partial J}{\partial w^{(l)}} = dw^{(l)}$$

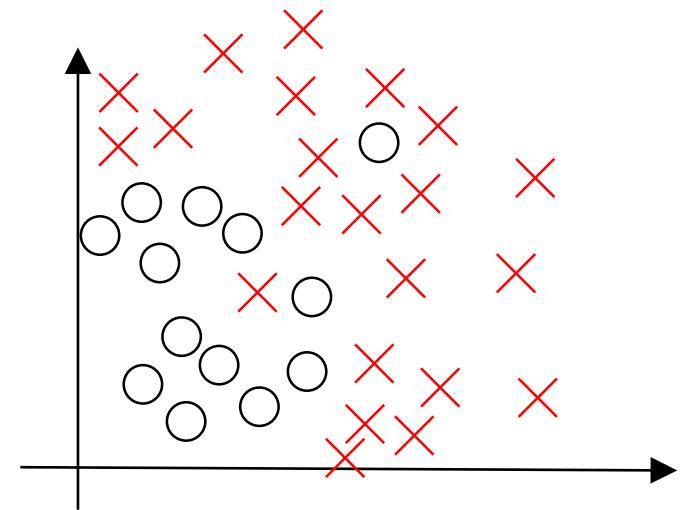
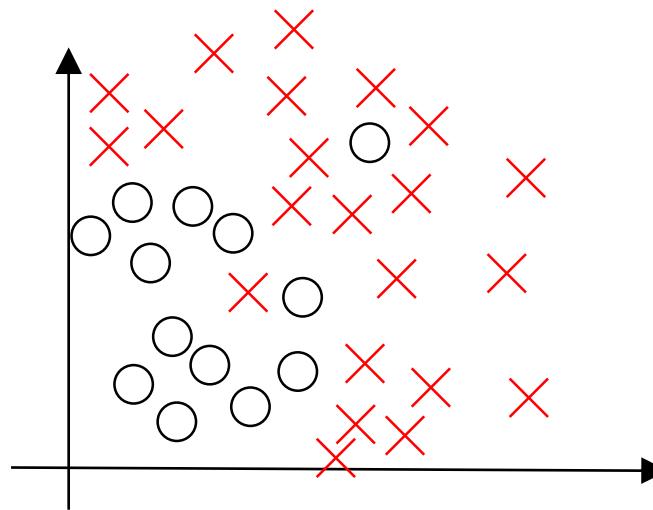
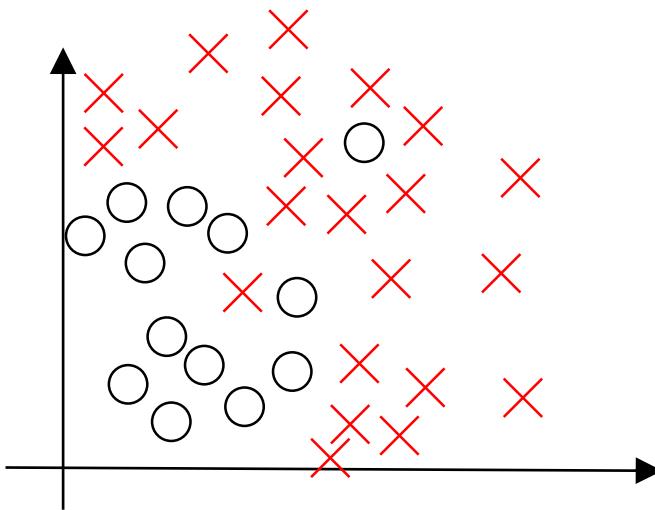
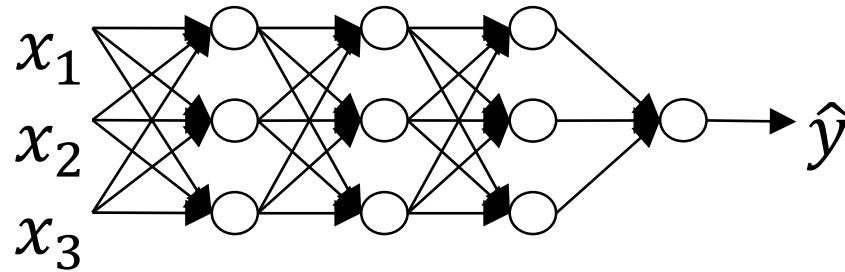
"Weight decay"

$$w^{(l)} := w^{(l)} - \alpha \left[ (\text{from backprop}) + \frac{\lambda}{m} w^{(l)} \right]$$

$$= w^{(l)} - \frac{\alpha \lambda}{m} w^{(l)} - \alpha (\text{from backprop})$$

$$= \underbrace{\left(1 - \frac{\alpha \lambda}{m}\right) w^{(l)}}_{<1} - \alpha (\text{from backprop})$$

# How does regularization prevent overfitting?



# How does regularization prevent overfitting?



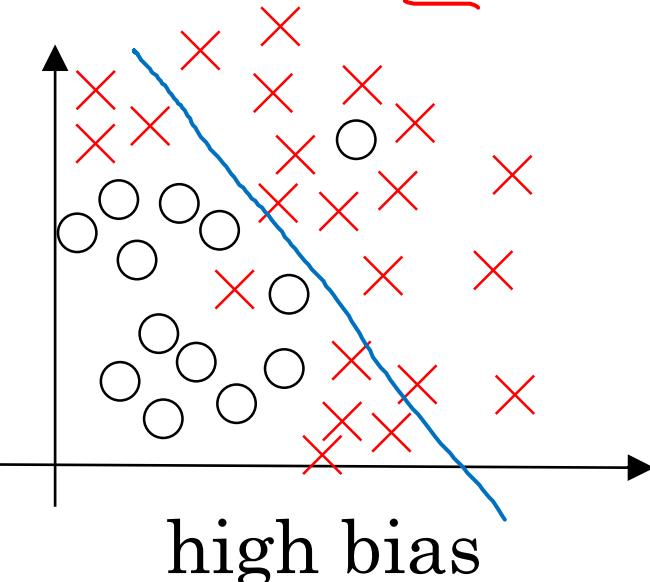
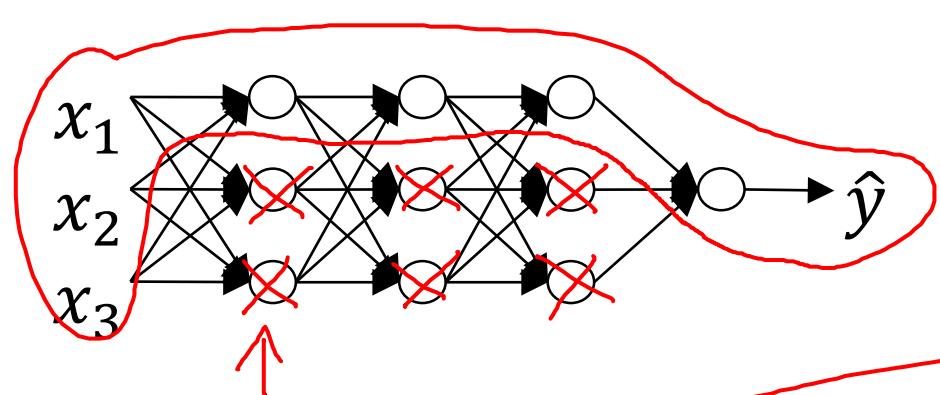
deeplearning.ai

# Regularizing your neural network

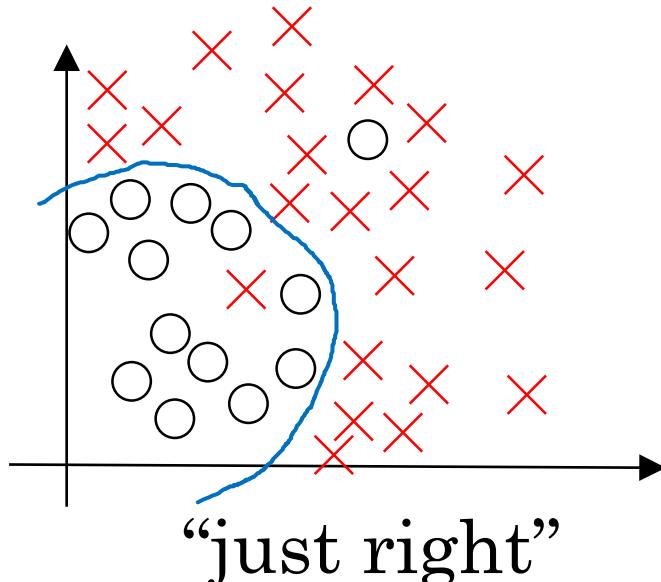
---

## Why regularization reduces overfitting

# How does regularization prevent overfitting?



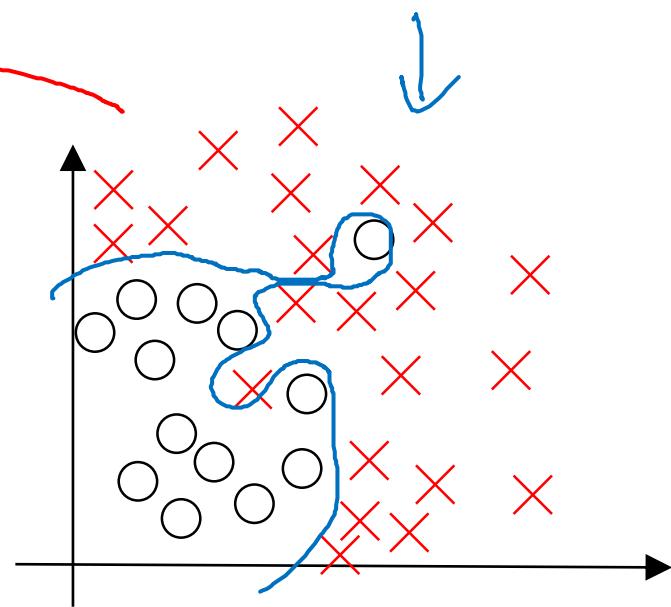
high bias



“just right”

$$J(\boldsymbol{\omega}^{(u)}, \boldsymbol{b}^{(u)}) = \frac{1}{m} \sum_{i=1}^m \ell(y^{(i)}, \hat{y}^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^L \|\boldsymbol{w}^{(l)}\|_F^2$$

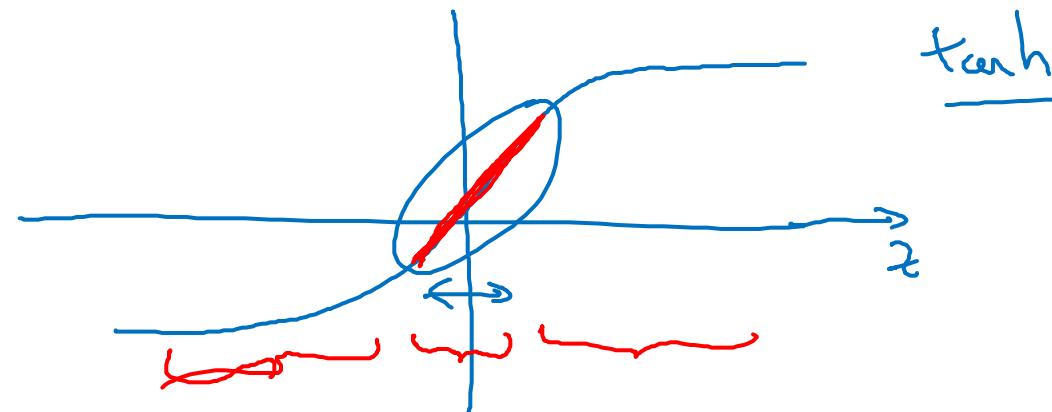
$\boldsymbol{w}^{(u)} \approx 0$



high variance



# How does regularization prevent overfitting?



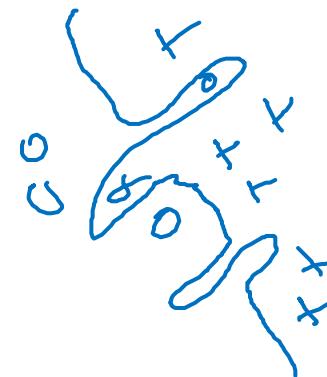
$$\lambda \uparrow$$

$$\underline{w^{[l]}} \downarrow$$

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

Every layer  $\approx$  linear.

$$J(\dots) = \boxed{\sum_i L(\hat{y}^{(i)}, y^{(i)})} + \lambda \sum_l \|\underline{w^{[l]}}\|_F^2$$





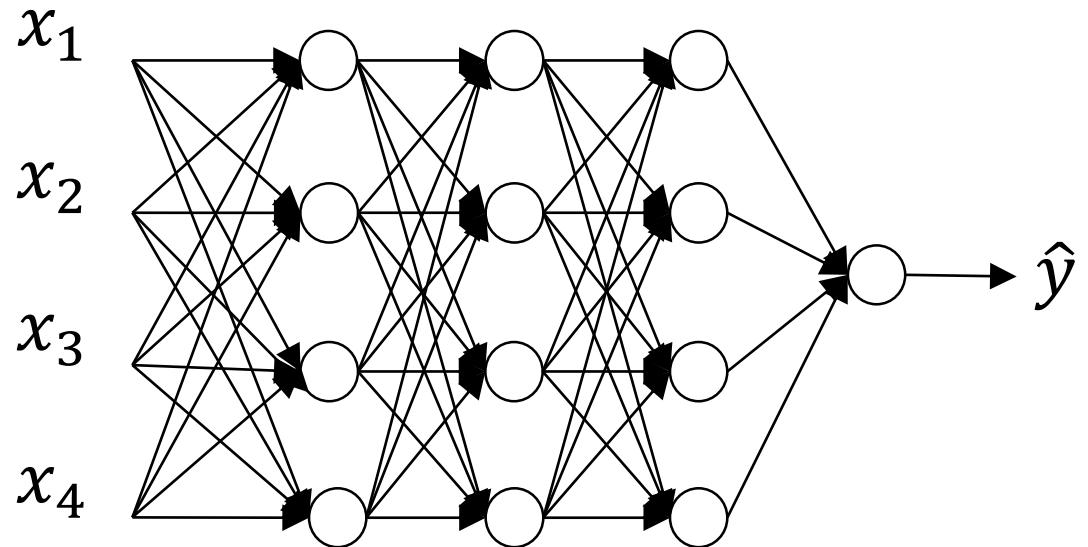
deeplearning.ai

Regularizing your  
neural network

---

Dropout  
regularization

# Dropout regularization



$\uparrow$   
0.5       $\uparrow$   
0.5       $\uparrow$   
0.5

# Implementing dropout (“Inverted dropout”)

Illustrate with layer  $l=3$ .  $\text{keep-prob} = \frac{0.8}{x}$   $\underline{\underline{0.2}}$

$\rightarrow d_3 = \underline{\underline{\text{np.random.rand}(a3.shape[0], a3.shape[1]) < \text{keep-prob}}}$

$\underline{\underline{a3}} = \text{np.multiply}(a3, d3)$   $\# a3 * d3$ .

$\rightarrow a3 /= \cancel{\text{keep-prob}}$   $\leftarrow$   
 $\uparrow$

50 units.  $\rightsquigarrow$  10 units shut off

$$z^{(4)} = w^{(4)} \cdot \underline{\underline{a^{(3)}}} + b^{(4)}$$

$\downarrow$  reduced by  $\underline{20\%}$ .

Test

$$1 = \underline{\underline{0.8}}$$

# Making predictions at test time

$$a^{(0)} = X$$

No drop out.

$$\uparrow z^{(1)} = w^{(1)} \underline{a^{(0)}} + b^{(1)}$$

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

$$z^{(2)} = w^{(2)} \underline{a^{(1)}} + b^{(2)}$$

$$a^{(2)} = \dots$$

$$\downarrow \hat{y}$$

$\lambda$  = keep-prob



deeplearning.ai

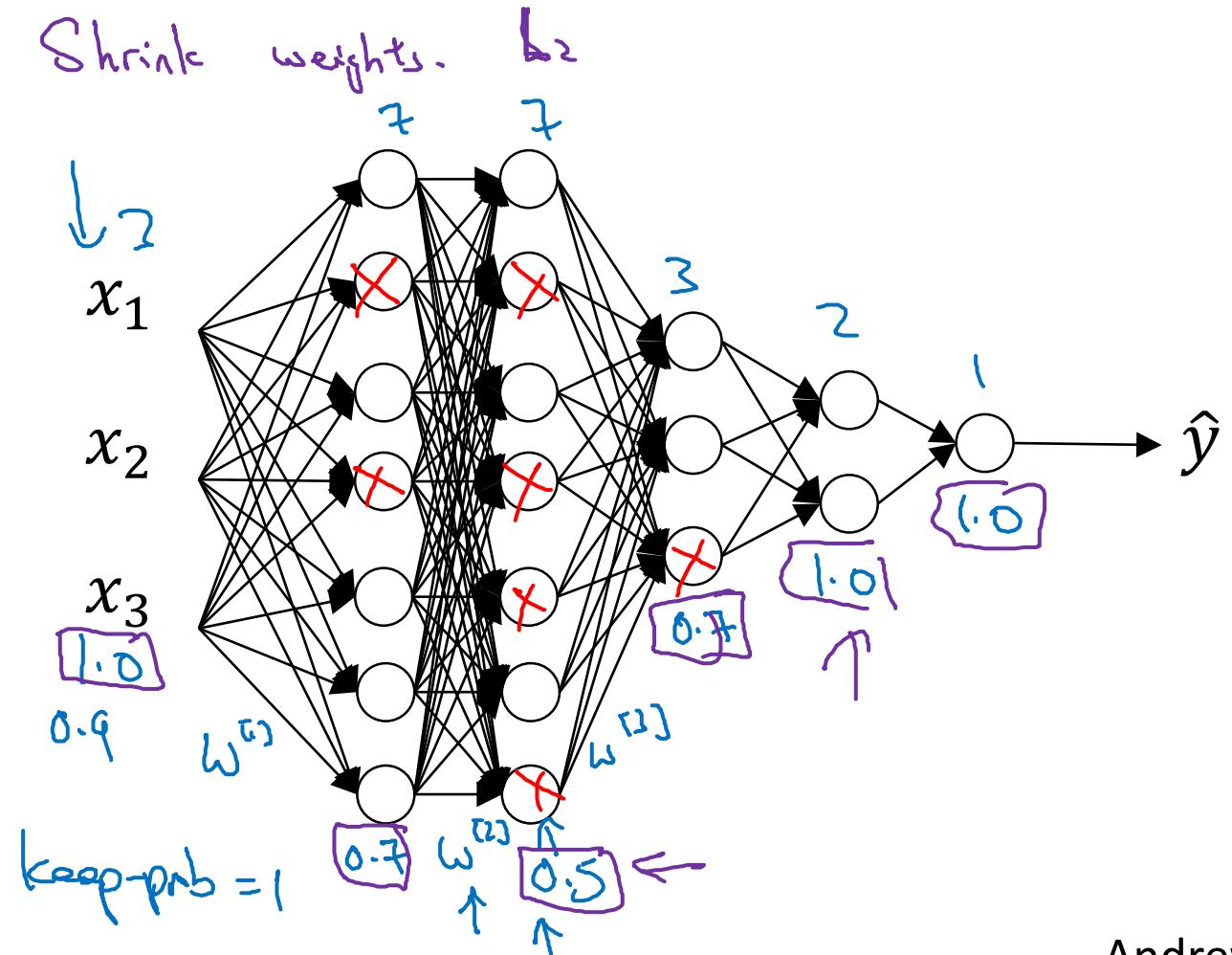
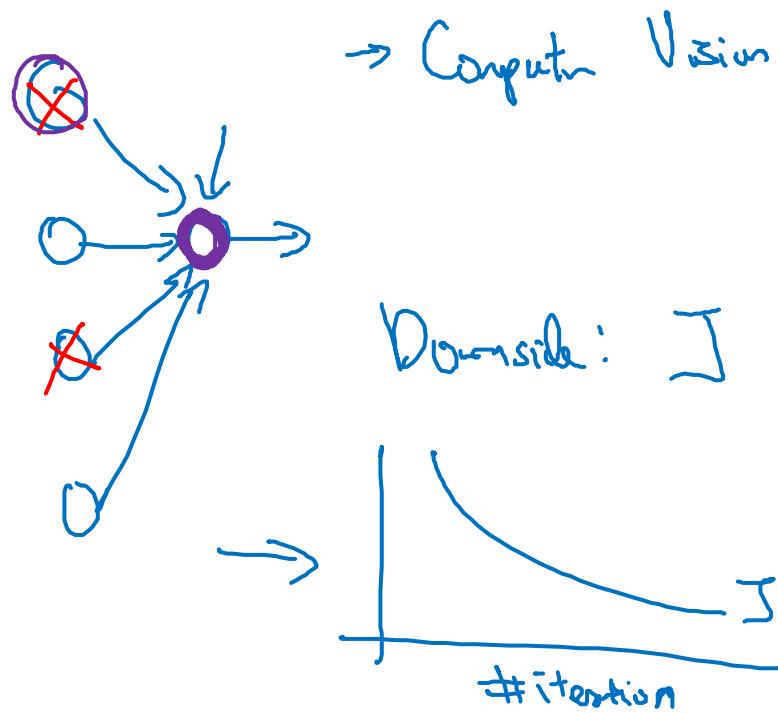
Regularizing your  
neural network

---

Understanding  
dropout

# Why does drop-out work?

Intuition: Can't rely on any one feature, so have to spread out weights.  $\rightsquigarrow$  Shrink weights.





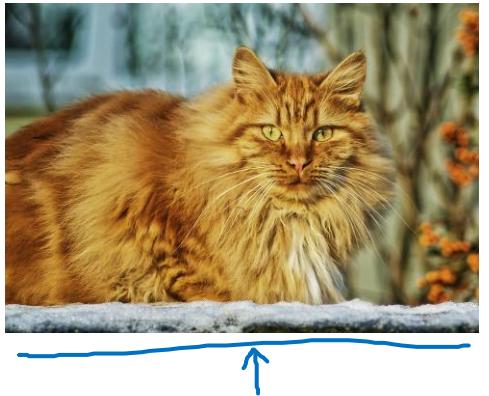
deeplearning.ai

# Regularizing your neural network

---

## Other regularization methods

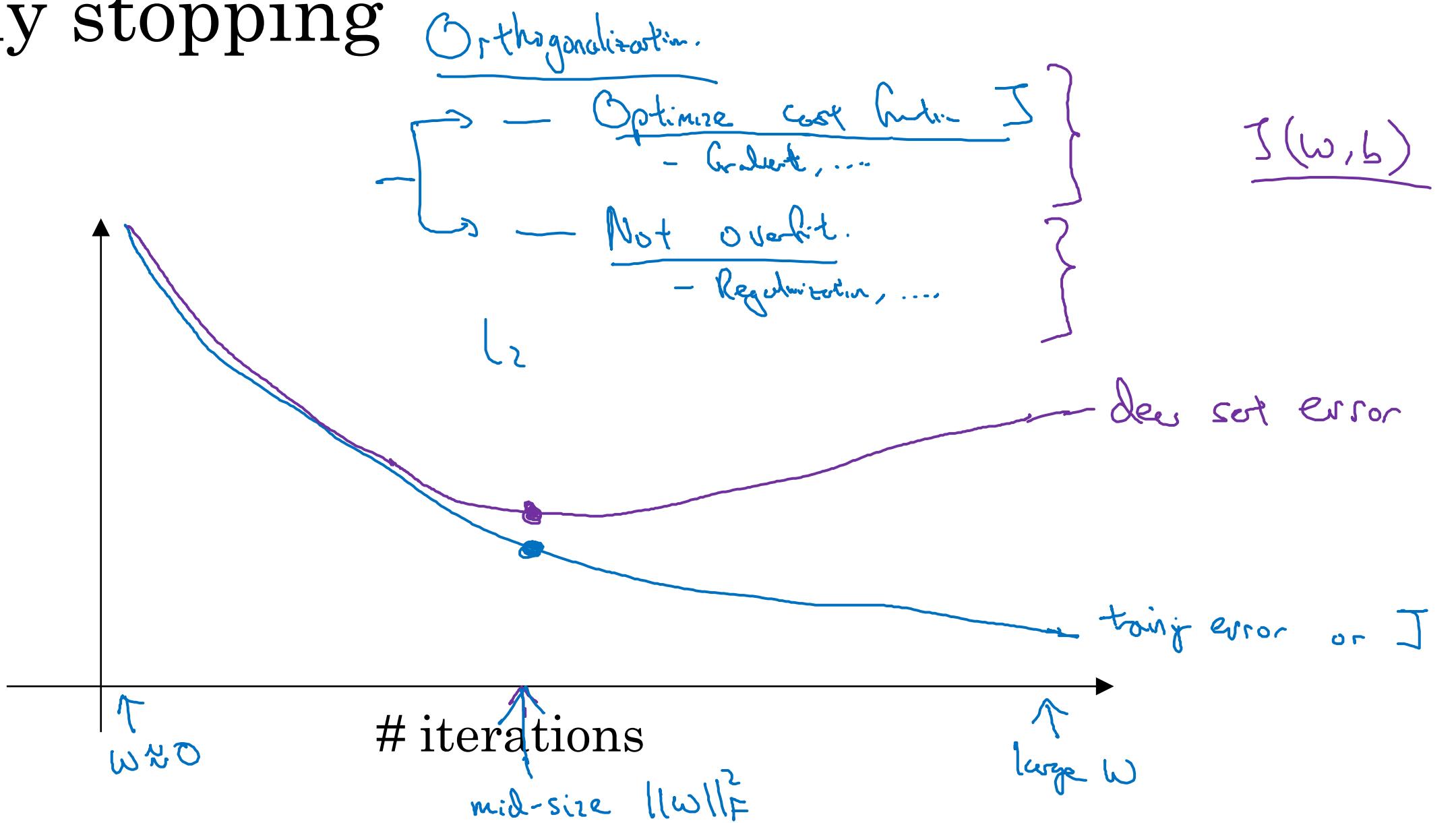
# Data augmentation



4

A large black digit '4' centered on the page.

# Early stopping





deeplearning.ai

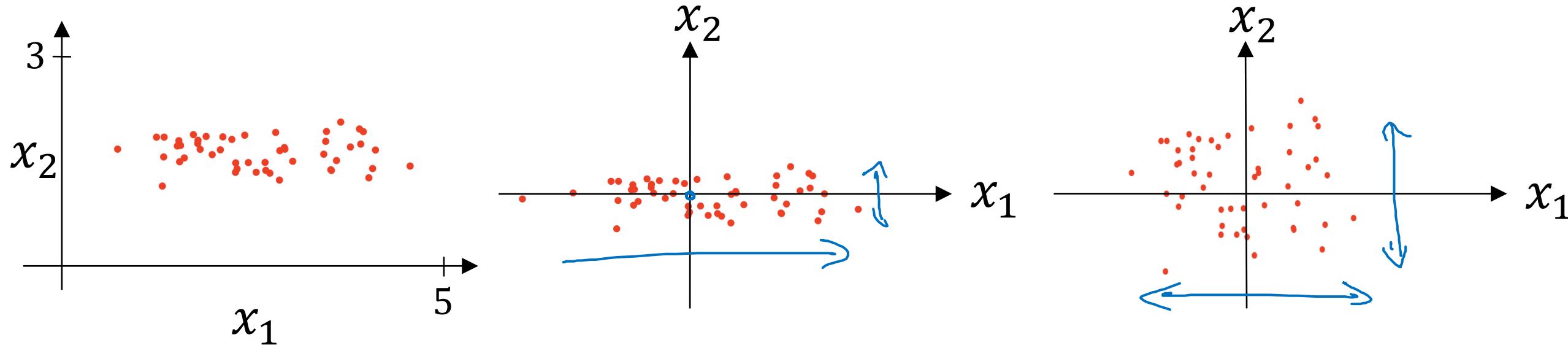
Setting up your  
optimization problem

---

Normalizing inputs

# Normalizing training sets

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$



Subtract mean:

$$\mu = \frac{1}{m} \sum_{i=1}^m x^{(i)}$$

$$\underline{x := x - \mu}$$

Normalize variance

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^m x^{(i)} * x^{(i)}$$

~ element-wise

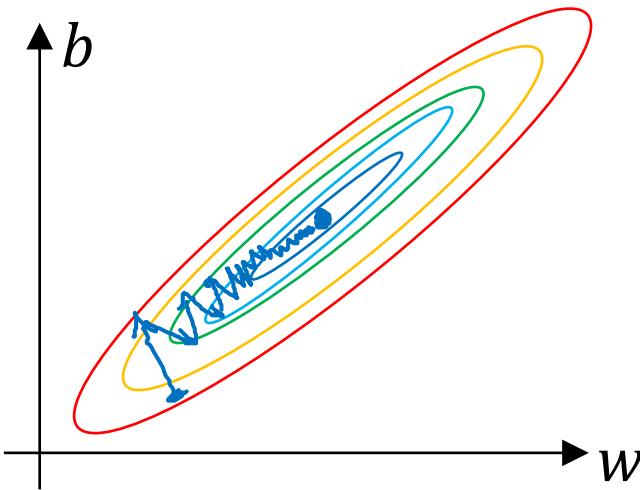
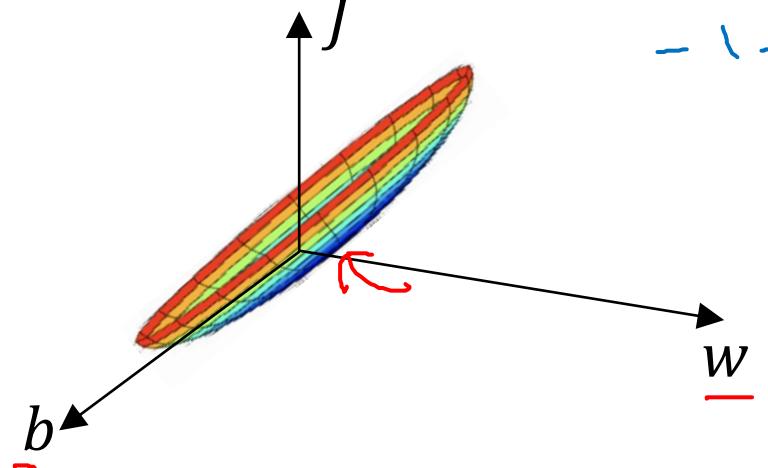
$$\underline{x / \sigma^2}$$

Use same  $\mu, \sigma^2$  to normalize test set.

# Why normalize inputs?

$w_1 \quad x_1: \frac{1 \dots 1000}{0 \dots 1} \leftarrow$   
 $w_2 \quad x_2: \frac{0 \dots 1}{-1 \dots 1} \leftarrow$

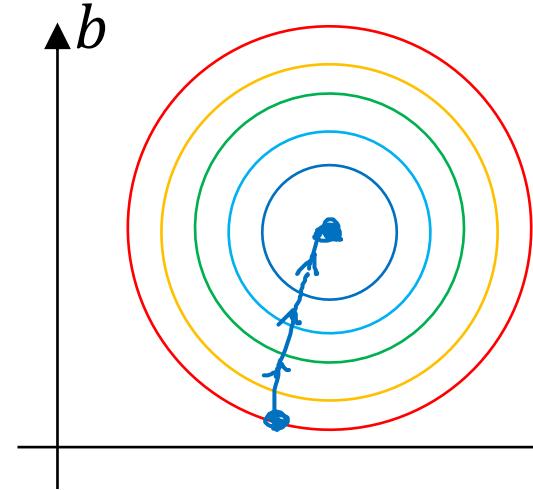
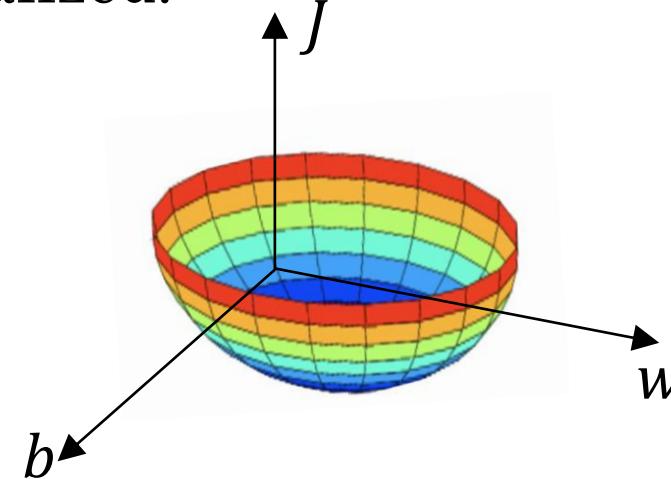
Unnormalized:



$x_1: 0 \dots 1$   
 $x_2: -1 \dots 1$   
 $x_3: 1 \dots 2$

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

Normalized:





deeplearning.ai

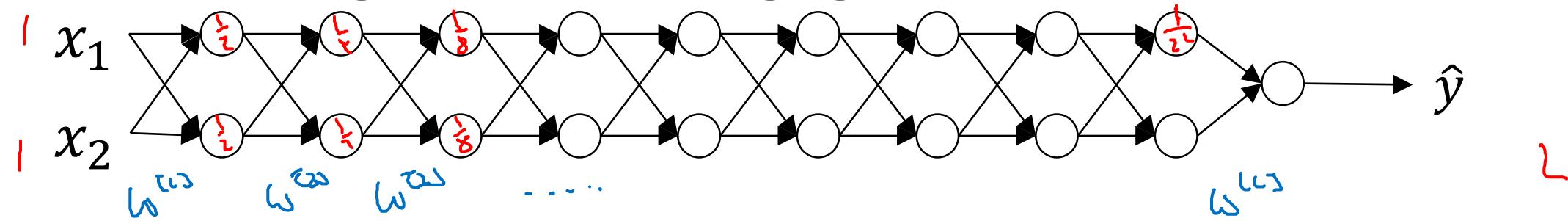
Setting up your  
optimization problem

---

Vanishing/exploding  
gradients

# Vanishing/exploding gradients

$L=150$



$$\underline{g(z) = z} . \quad b^{[L]} = 0 .$$



$$w^{[1]} > I$$

$$w^{[2]} < I \quad \begin{bmatrix} 0.9 & \\ & 0.9 \end{bmatrix}$$

$$w^{[L]} = \begin{bmatrix} 1.5 & 0 \\ 0 & 6.5 \end{bmatrix}$$

$$z^{[1]} = \underline{w^{[1]} x}$$

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

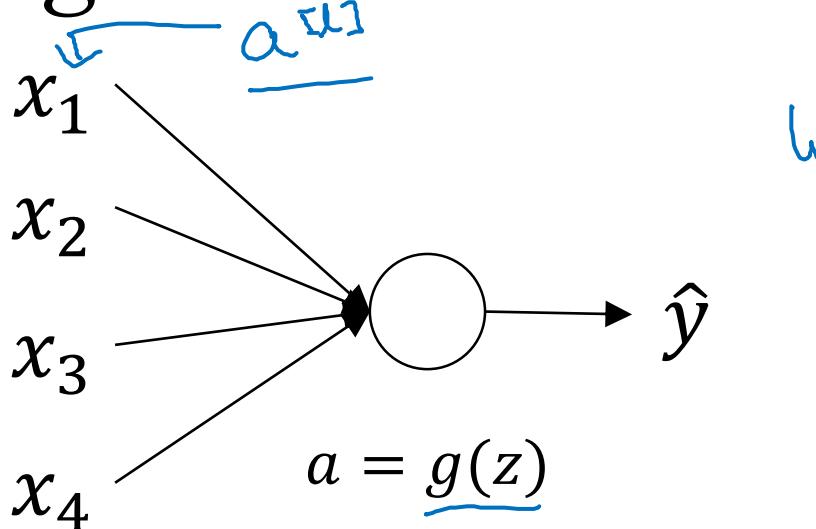
$$a^{[2]} = g(z^{[1]}) = g(w^{[2]} a^{[1]})$$

$$\hat{y} = w^{[1]} \begin{bmatrix} 0.5 & \\ & 1.5 \end{bmatrix}^{L-1} x$$

$$1.5^{L-1} x$$

$$6.5^{L-1} x$$

# Single neuron example



$$z = \underline{w_1 x_1 + w_2 x_2 + \dots + w_n x_n} \quad \cancel{\text{if } n \text{ is large}}$$

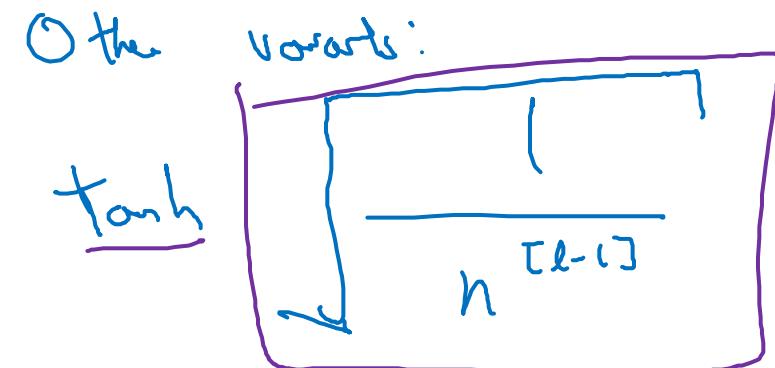
Large  $n \rightarrow$  Smaller  $w_i$

$$\text{Var}(w_i) = \frac{1}{n} \frac{2}{n}$$

$$\underline{w^{[l]}} = \text{np.random.randn}(\text{shape}) * \text{np.sqrt}\left(\frac{2}{n^{[l-1]}}\right)$$

ReLU

$g^{[l]}(z) = \text{ReLU}(z)$



$$\frac{2}{n^{[l-1]} + n^{[l]}}$$

↑



deeplearning.ai

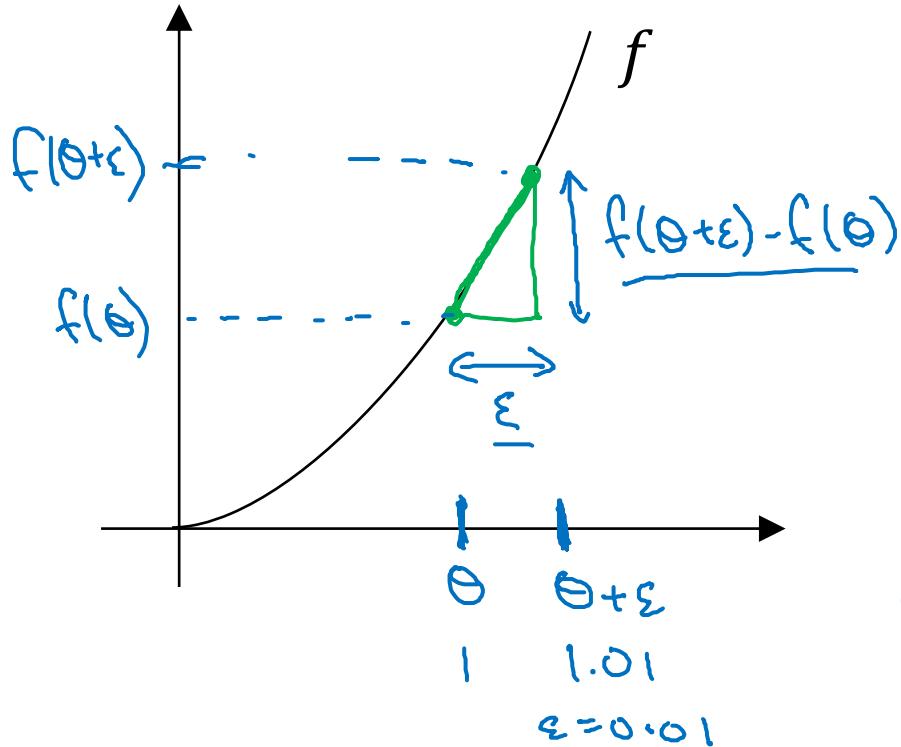
# Setting up your optimization problem

---

## Numerical approximation of gradients

# Checking your derivative computation

$$\begin{aligned} f(\theta) &= \underline{\theta^3} \\ \theta &\in \mathbb{R}. \\ \text{I} \end{aligned}$$



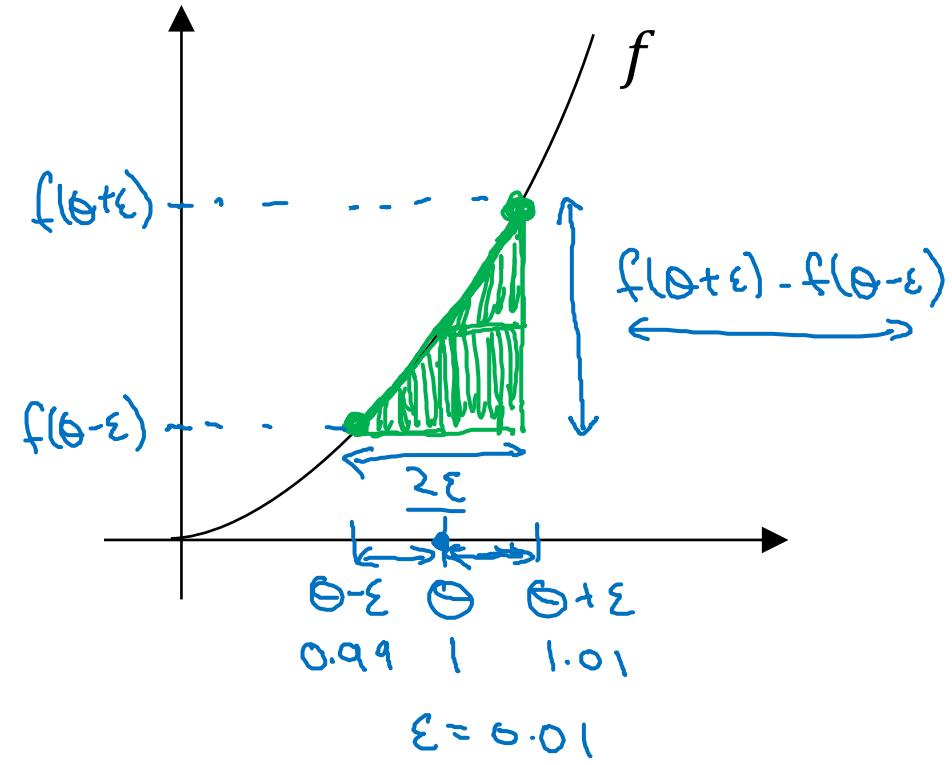
$$\begin{aligned} g(\theta) &= \frac{d}{d\theta} f(\theta) = f'(\theta) \\ g(\theta) &= 3\theta^2. \\ g(1) &= 3 \cdot (1)^2 = 3 \\ \text{when } \theta &= 1 \\ \frac{dw}{db} \end{aligned}$$

$$\begin{aligned} \frac{f(\theta+\epsilon) - f(\theta)}{\epsilon} &\approx g(\theta) \\ \frac{(1.01)^3 - 1^3}{0.01} &= 3.0301 \\ \frac{3.0301}{0.0301} &\approx 3 \end{aligned}$$

$$\begin{aligned} \theta &= 1 \\ \theta + \epsilon &= 1.01 \end{aligned}$$

# Checking your derivative computation

$$\underline{f(\theta) = \theta^3}$$



$f'(\theta) = \lim_{\epsilon \rightarrow 0} \frac{f(\theta + \epsilon) - f(\theta - \epsilon)}{2\epsilon}$	$\frac{f(\theta + \epsilon) - f(\theta - \epsilon)}{2\epsilon} = \frac{\mathcal{O}(\epsilon^2)}{\epsilon} = \frac{0.01}{0.0001}$	$\frac{f(\theta + \epsilon) - f(\theta)}{\epsilon} = \text{error: } \mathcal{O}(\epsilon) = 0.01$
--	--	---



deeplearning.ai

Setting up your  
optimization problem

---

Gradient Checking

# Gradient check for a neural network

Take  $W^{[1]}, b^{[1]}, \dots, W^{[L]}, b^{[L]}$  and reshape into a big vector  $\underline{\theta}$ .

$$J(w^{[1]}, b^{[1]}, \dots, w^{[L]}, b^{[L]}) = J(\underline{\theta})$$

Take  $dW^{[1]}, db^{[1]}, \dots, dW^{[L]}, db^{[L]}$  and reshape into a big vector  $\underline{d\theta}$ .

Is  $d\theta$  the gradient of  $J(\underline{\theta})$ ?

# Gradient checking (Grad check)

$$J(\theta) = J(\theta_0, \theta_1, \theta_2, \dots)$$

for each  $i$ :

$$\rightarrow \underline{d\theta_{\text{approx}}[i]} = \frac{J(\theta_0, \theta_1, \dots, \theta_i + \varepsilon, \dots) - J(\theta_0, \theta_1, \dots, \theta_i - \varepsilon, \dots)}{\varepsilon}$$

$$\approx \underline{d\theta[i]} = \frac{\partial J}{\partial \theta_i}$$

$$d\theta_{\text{approx}} \stackrel{?}{\approx} d\theta$$

Check

$$\rightarrow \frac{\|d\theta_{\text{approx}} - d\theta\|_2}{\|d\theta_{\text{approx}}\|_2 + \|d\theta\|_2}$$

$$\varepsilon = 10^{-7}$$

$$\approx \boxed{10^{-7} - \text{great!}} \leftarrow$$

$$\rightarrow 10^{-3} - \text{worry.} \leftarrow$$



deeplearning.ai

# Setting up your optimization problem

---

## Gradient Checking implementation notes

# Gradient checking implementation notes

- Don't use in training – only to debug

$$\frac{\partial \theta_{\text{approx}}^{[i]}}{\uparrow} \longleftrightarrow \frac{\partial \theta^{[i]}}{\uparrow}$$

- If algorithm fails grad check, look at components to try to identify bug.

$$\frac{\partial b^{[l]}}{\uparrow} \quad \frac{\partial w^{[l]}}{\uparrow}$$

- Remember regularization.

$$J(\theta) = \frac{1}{m} \sum_i f(y^{(i)}, \hat{y}^{(i)}) + \frac{\lambda}{2m} \sum_l \|w^{(l)}\|_F^2$$

$\frac{\partial \theta}{\uparrow} = \text{gradt of } J \text{ wrt. } \theta$

- Doesn't work with dropout.

$$J \quad \underline{\text{keep-prob} = 1.0}$$

- Run at random initialization; perhaps again after some training.

$$\underline{w, b \text{ no}}$$



deeplearning.ai

# Optimization Algorithms

---

## Mini-batch gradient descent

# Batch vs. mini-batch gradient descent

$$\underline{X}, \underline{Y} \quad \underline{X^{\{t\}}, Y^{\{t\}}}$$

Vectorization allows you to efficiently compute on  $m$  examples.

$$X = [x^{(1)} \ x^{(2)} \ x^{(3)} \ \dots \ x^{(500)} \ | \ x^{(1001)} \ \dots \ x^{(2000)} \ | \ \dots \ | \ \dots \ x^{(m)}]_{(n_x, m)}$$

$\underbrace{x^{(1)}}_{\{1\}} \quad (n_x, 1000) \quad \underbrace{x^{(2)}}_{\{2\}} \quad (n_x, 1000) \quad \dots \quad \underbrace{x^{(5000)}}_{\{5,000\}} \quad (n_x, 1000)$

$$Y = [y^{(1)} \ y^{(2)} \ y^{(3)} \ \dots \ y^{(1000)} \ | \ y^{(1001)} \ \dots \ y^{(2000)} \ | \ \dots \ | \ \dots \ y^{(m)}]_{(1, m)}$$

$\underbrace{y^{(1)}}_{\{1\}} \quad (1, 1000) \quad \underbrace{y^{(2)}}_{\{2\}} \quad (1, 1000) \quad \dots \quad \underbrace{y^{(5000)}}_{\{5,000\}} \quad (1, 1000)$

What if  $m = \underline{5,000,000}$ ?

$5,000$  mini-batches of  $1,000$  each

Mini-batch  $t$ :  $\underline{X^{\{t\}}, Y^{\{t\}}}$

$$\begin{aligned} & x^{(i)} \\ & z^{[l]} \\ & \underline{X^{\{t\}}, Y^{\{t\}}}. \end{aligned}$$

# Mini-batch gradient descent

repeat {  
for  $t = 1, \dots, 5000$  {

Forward prop on  $X^{\{t\}}$ .

$$Z^{(l)} = W^{(l)} X^{\{t\}} + b^{(l)}$$

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

:

$$A^{(L)} = g^{(L)}(Z^{(L)})$$

Compute cost  $J^{\{t\}} = \frac{1}{1000} \sum_{i=1}^L f(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2 \cdot 1000} \sum_l \|W^{(l)}\|_F^2$ .

Vectorized implementation  
(1000 examples)

$\downarrow$  for  $X^{\{t\}}, Y^{\{t\}}$ .

Backprop to compute gradients wrt  $J^{\{t\}}$  (using  $(X^{\{t\}}, Y^{\{t\}})$ )

$$W^{(l)} := W^{(l)} - \alpha \delta W^{(l)}, \quad b^{(l)} := b^{(l)} - \alpha \delta b^{(l)}$$

3 } 3 }

"1 epoch"  
└ pass through training set.

1 step of gradient descent  
using  $\frac{X^{\{t+1\}}}{Y^{\{t+1\}}}$   
(as if  $t=5000$ )

$X, Y$



deeplearning.ai

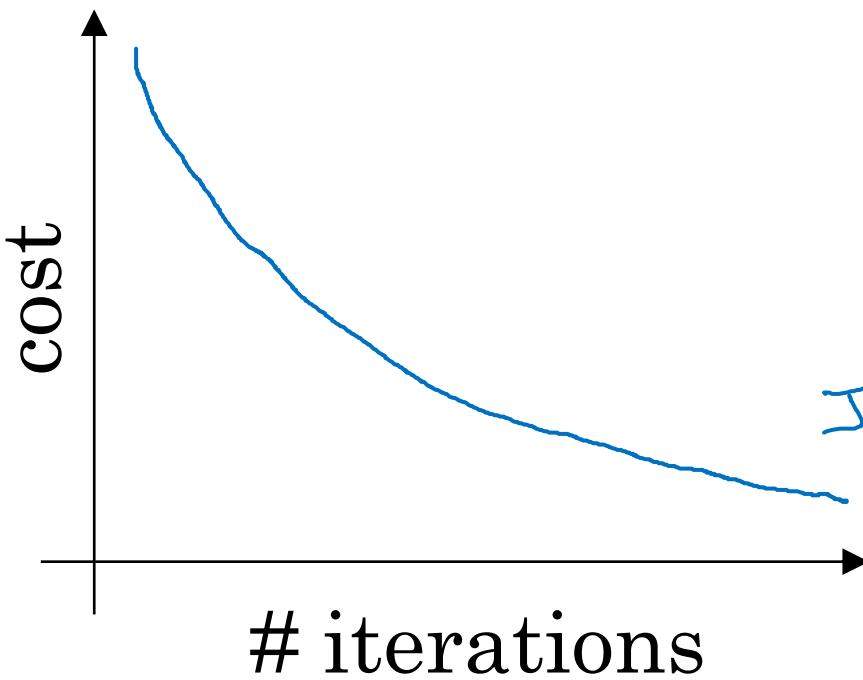
# Optimization Algorithms

---

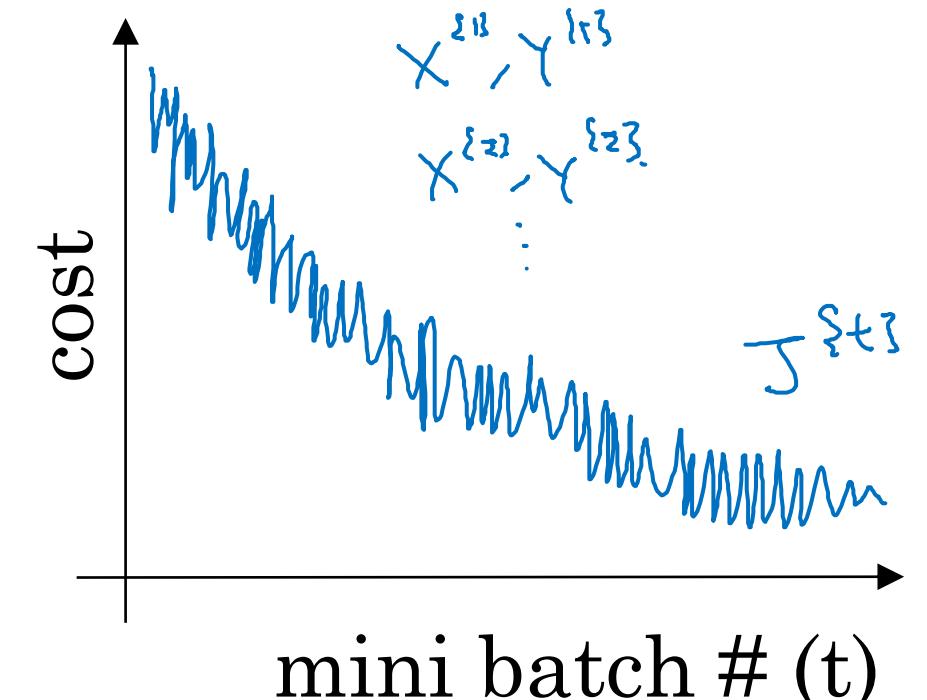
## Understanding mini-batch gradient descent

# Training with mini batch gradient descent

Batch gradient descent



Mini-batch gradient descent



Plot  $J^{st}$  computed using  $X^{\{st\}}, Y^{\{st\}}$

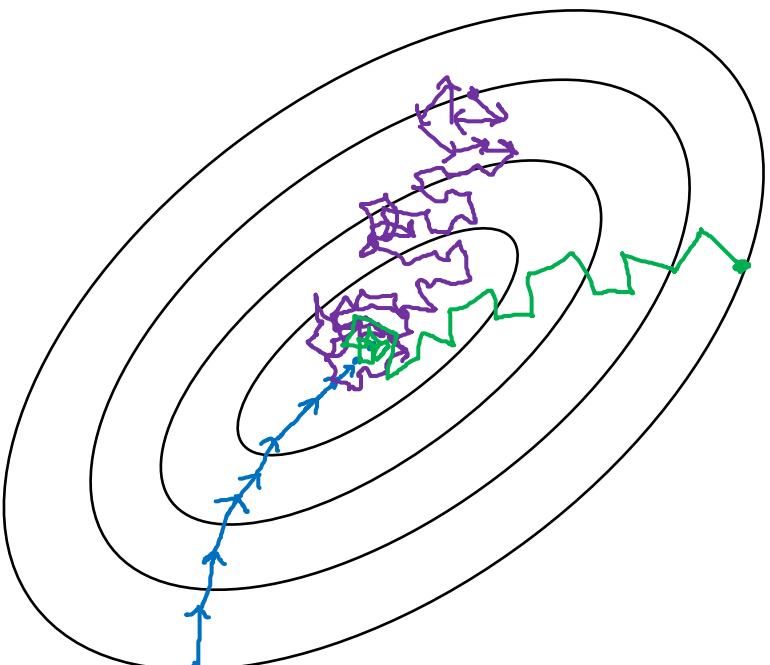
# Choosing your mini-batch size

→ If mini-batch size =  $m$  : Batch gradient descent.

$$(X^{\{1\}}, Y^{\{1\}}) = (X, Y)$$

→ If mini-batch size = 1 : Stochastic gradient descent. Every example is its own  
 $(X^{(1)}, Y^{(1)}) = (x^{(1)}, y^{(1)}) \dots (x^{(n)}, y^{(n)})$  mini-batch.

In practice: Somehw in-between 1 and  $m$



Stochastic  
gradient  
descent

{  
Use speedup  
from vectorization

In-between  
(mini-batch size  
not too big/small)

{  
Faster learning.

- Vectorization.  
( $n \approx 1000$ )
- Make passes without  
processing entire training set.

Batch  
gradient descent  
(mini-batch size =  $m$ )

{  
Two long  
per iteration

# Choosing your mini-batch size

If small training set : Use batch gradient descent.  
 $(m \leq 2000)$

Typical mini-batch sizes:

$$\rightarrow 64, 128, 256, 512 \quad \frac{1024}{2^{10}}$$

$2^6 \quad 2^7 \quad 2^8 \quad 2^9$



Make sure mini-batch fits in CPU/GPU memory.

$$X^{\{t\}}, Y^{\{t\}}$$



deeplearning.ai

# Optimization Algorithms

---

## Exponentially weighted averages

# Temperature in London

$$\theta_1 = 40^{\circ}\text{F} \quad 4^{\circ}\text{C} \quad \leftarrow$$

$$\theta_2 = 49^{\circ}\text{F} \quad 9^{\circ}\text{C}$$

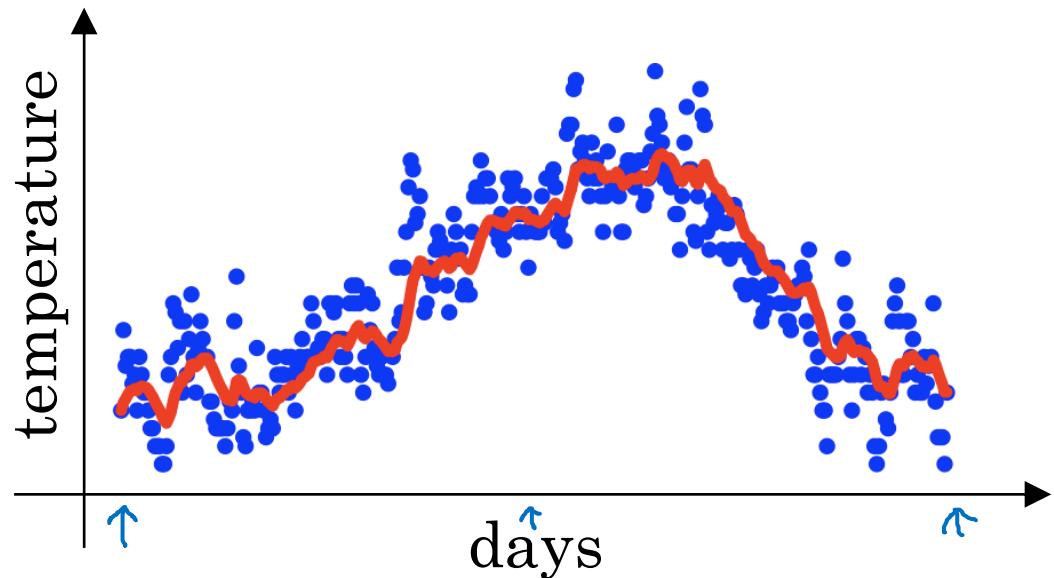
$$\theta_3 = 45^{\circ}\text{F} \quad \vdots$$

⋮

$$\theta_{180} = 60^{\circ}\text{F} \quad 15^{\circ}\text{C}$$

$$\theta_{181} = 56^{\circ}\text{F} \quad \vdots$$

⋮



$$V_0 = 0$$

$$V_1 = 0.9 V_0 + 0.1 \theta_1$$

$$V_2 = 0.9 V_1 + 0.1 \theta_2$$

$$V_3 = 0.9 V_2 + 0.1 \theta_3$$

⋮

$$V_t = 0.9 V_{t-1} + 0.1 \theta_t$$

# Exponentially weighted averages <sup>Moving</sup>

$$V_t = \beta V_{t-1} + (1-\beta) \theta_t \leftarrow$$

$\beta = 0.9$  :  $\approx 10$  days' temperatur.

$\beta = 0.98$  :  $\approx 50$  days

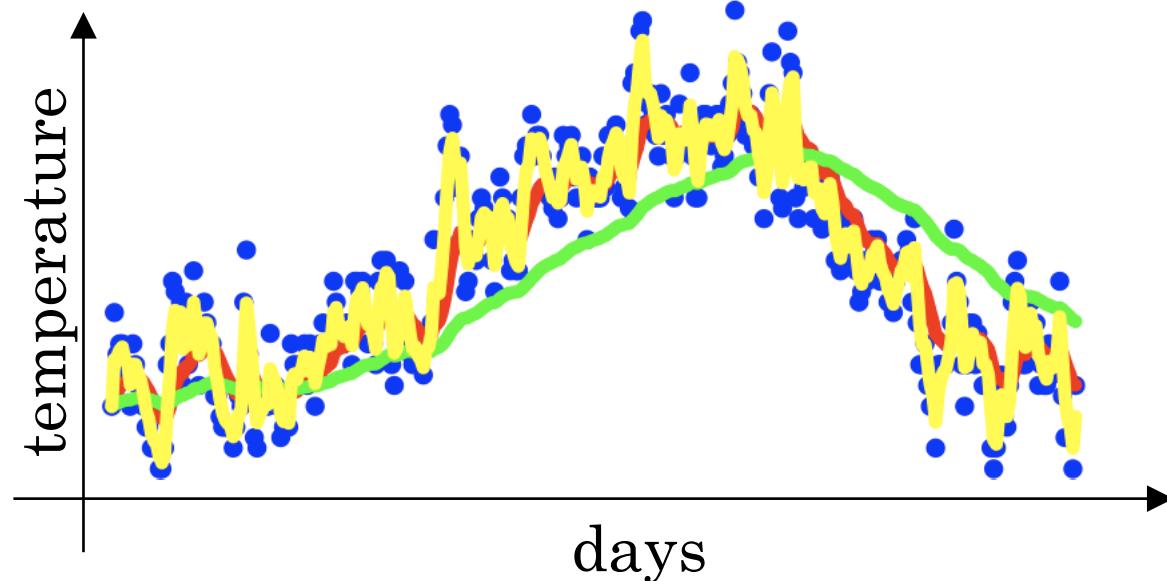
$\beta = 0.5$  :  $\approx 2$  days

$V_t$  is approximately

average over

$\rightarrow \approx \frac{1}{1-\beta}$  days'  
temperature.

$$\frac{1}{1-0.98} = 50$$





deeplearning.ai

# Optimization Algorithms

---

## Understanding exponentially weighted averages

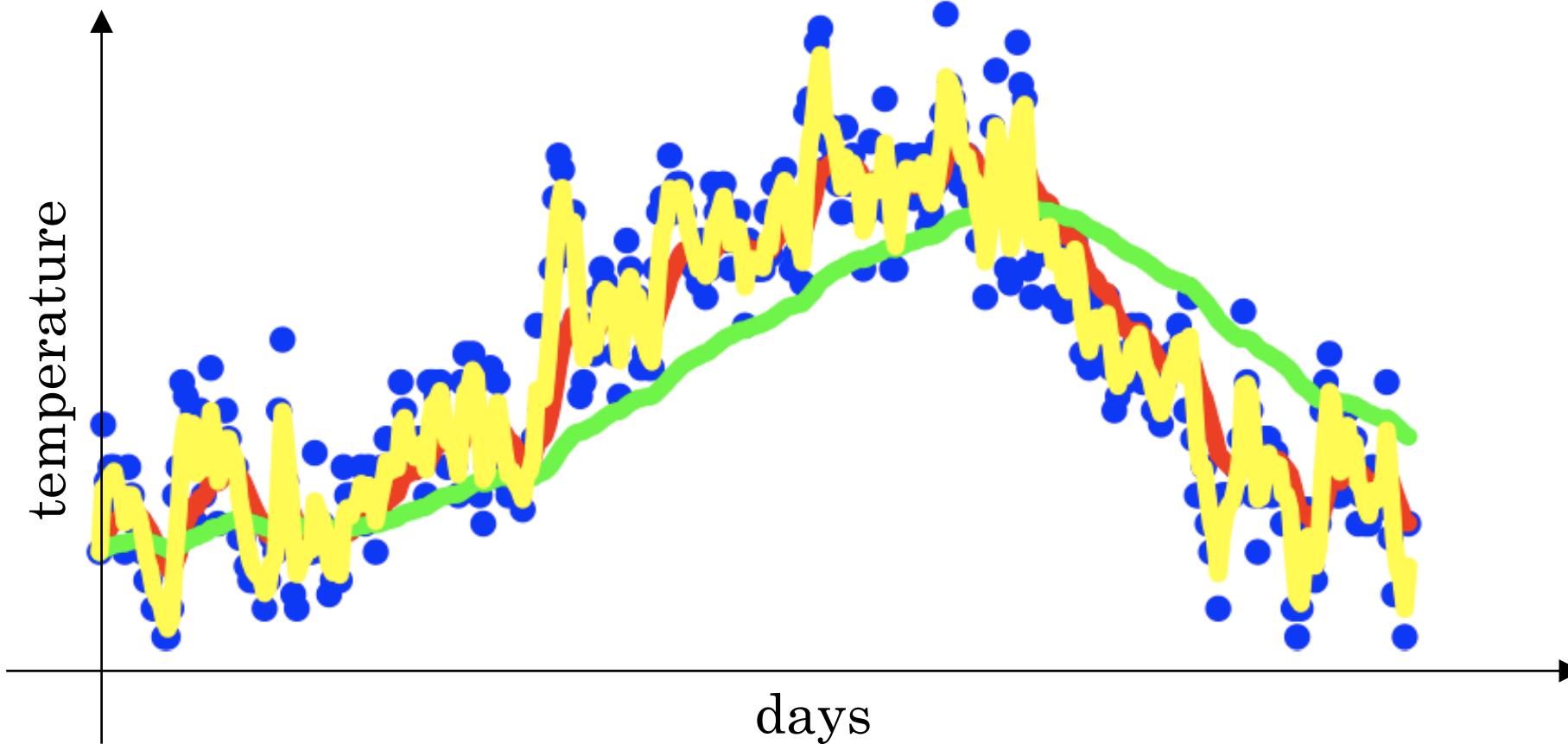
# Exponentially weighted averages

$$v_t = \beta v_{t-1} + (1 - \beta) \theta_t$$

$$\beta = 0.9$$

$$0.98$$

$$0.5$$



# Exponentially weighted averages

$$v_t = \beta v_{t-1} + (1 - \beta) \theta_t$$

$$v_{100} = 0.9v_{99} + 0.1\theta_{100}$$

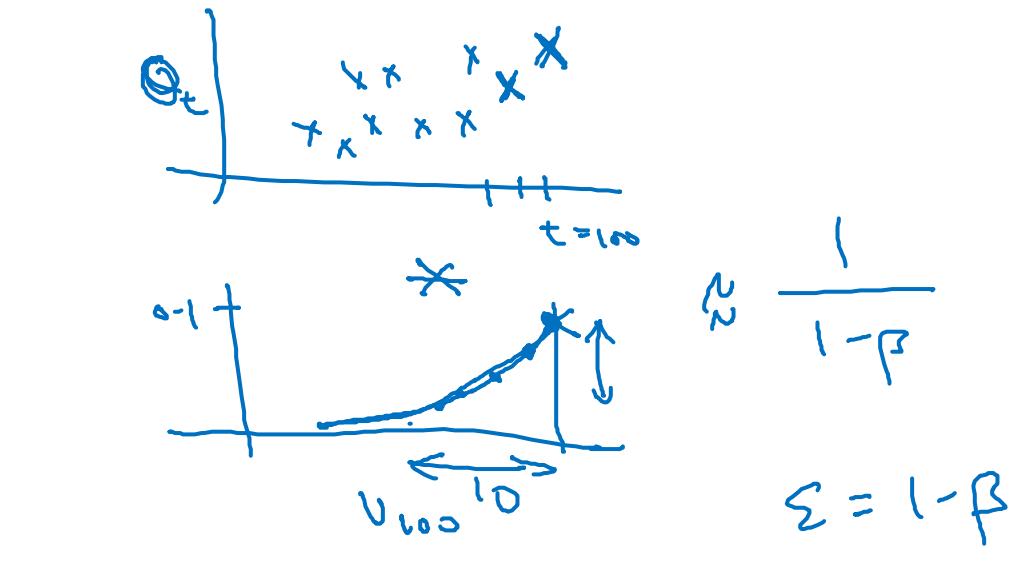
$$v_{99} = 0.9v_{98} + 0.1\theta_{99}$$

$$v_{98} = 0.9v_{97} + 0.1\theta_{98}$$

...

$$\begin{aligned} \underline{v_{100}} &= 0.1 \underline{\theta_{100}} + 0.9 \cancel{(0.1 \underline{\theta_{99}})} + 0.9 \cancel{(0.1 \underline{\theta_{98}})} \\ &= 0.1 \underline{\theta_{100}} + \underline{0.1 \times 0.9 \cdot \underline{\theta_{99}}} + \underline{0.1 (0.9)^2 \underline{\theta_{98}}} + \underline{0.1 (0.9)^3 \underline{\theta_{97}}} + \underline{0.1 (0.9)^4 \underline{\theta_{96}}} + \dots \end{aligned}$$

$$\underline{0.9^{10}} \approx \underline{0.35} \approx \frac{1}{e}$$



$$\frac{1}{1-\beta}$$

$$\Sigma = 1 - \beta$$

$$0.1 \underline{\theta_{98}} + 0.9 \underline{v_{97}}$$

$$\frac{(1-\epsilon)^{1/\epsilon}}{0.9} = \frac{1}{e}$$

$$0.98 ?$$

$$\epsilon = 0.02 \rightarrow \underline{0.98^{50}} \approx \frac{1}{e}$$

# Implementing exponentially weighted averages

$$v_0 = 0$$

$$v_1 = \beta v_0 + (1 - \beta) \theta_1$$

$$v_2 = \beta v_1 + (1 - \beta) \theta_2$$

$$v_3 = \beta v_2 + (1 - \beta) \theta_3$$

...

$$V_0 := 0$$

$$V_0 := \beta V + (1-\beta) \theta_1$$

$$V_0 := \beta V + (1-\beta) \theta_2$$

:

---

$$\rightarrow V_0 = 0$$

Repeat {

Get next  $\theta_t$

$$V_0 := \beta V_0 + (1-\beta) \theta_t \leftarrow$$

}



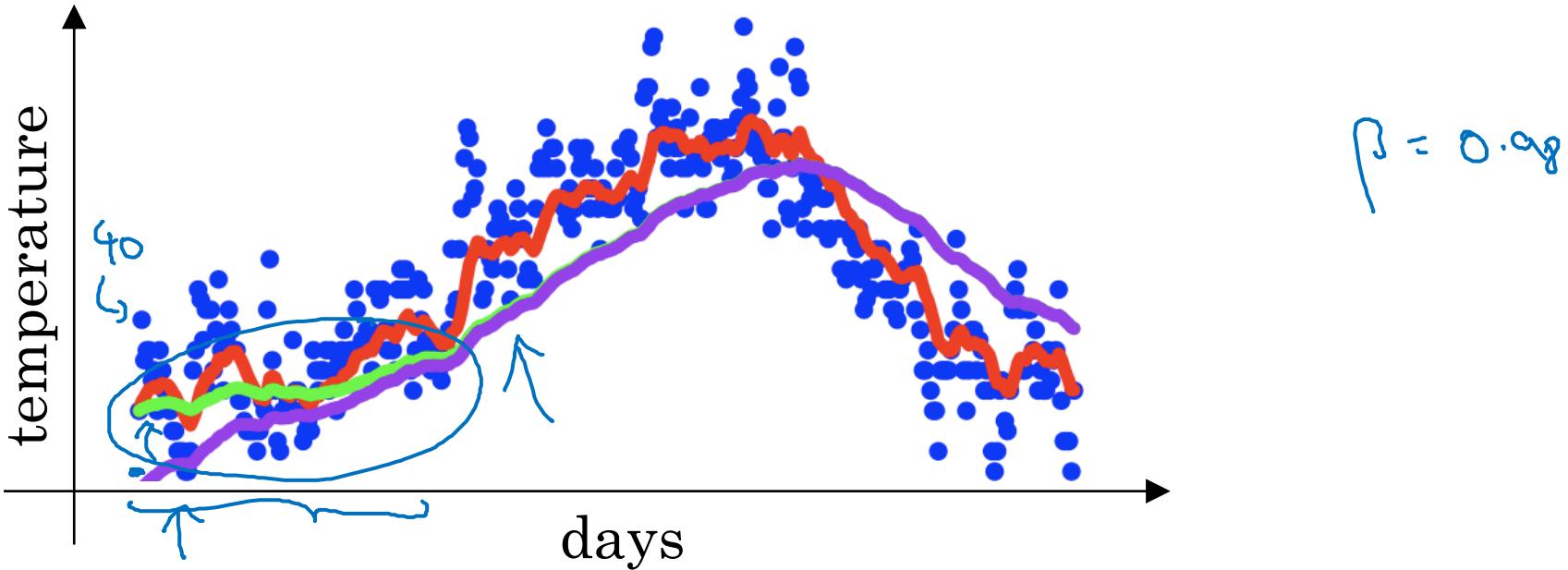
deeplearning.ai

# Optimization Algorithms

---

Bias correction  
in exponentially  
weighted average

# Bias correction



$$\rightarrow v_t = \beta v_{t-1} + (1 - \beta) \theta_t$$

$$v_0 = 0$$

$$v_1 = \cancel{0.98v_0} + \underline{0.02\theta_1}$$

$$\begin{aligned} v_2 &= 0.98 v_1 + 0.02 \theta_2 \\ &= 0.98 \times 0.02 \times \theta_1 + 0.02 \theta_2 \\ &= \underline{0.0196\theta_1} + \underline{0.02\theta_2} \end{aligned}$$

$$\frac{v_t}{1 - \beta^t}$$

$$t=2: 1 - \beta^t = 1 - (0.98)^2 = 0.0396$$

$$\frac{v_2}{0.0396} =$$

$$\frac{\underline{0.0196\theta_1} + \underline{0.02\theta_2}}{0.0396}$$



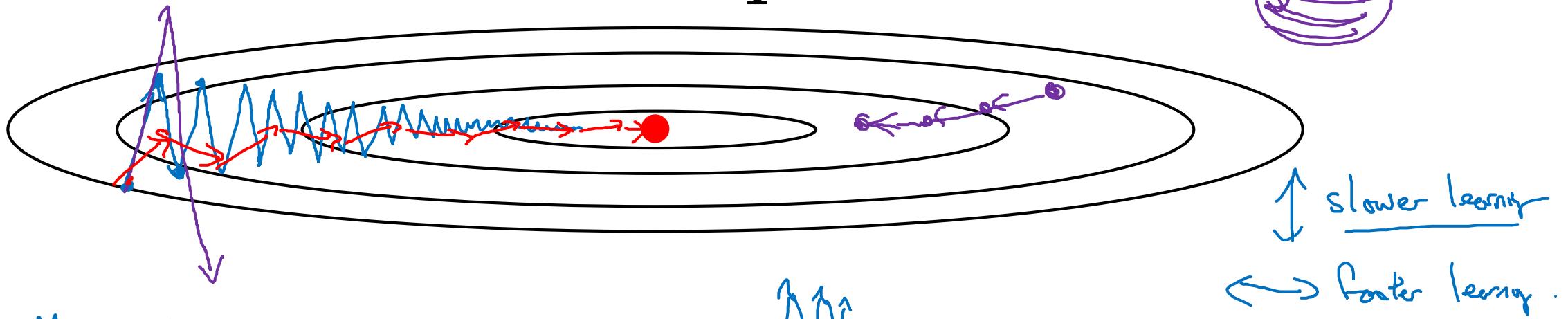
deeplearning.ai

# Optimization Algorithms

---

## Gradient descent with momentum

# Gradient descent example



Momentum:

On iteration  $t$ :

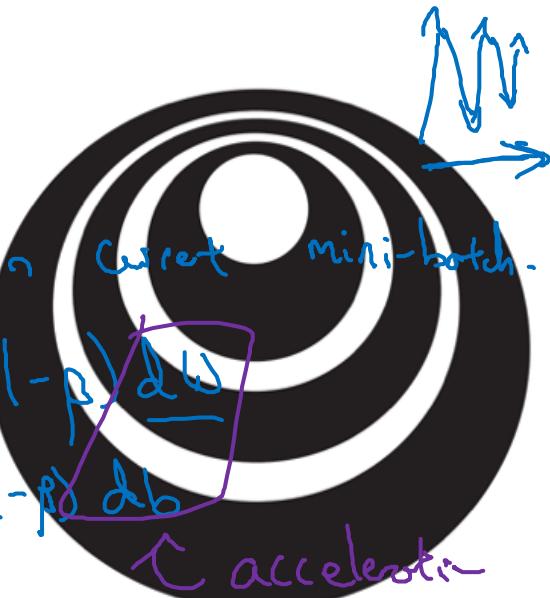
Compute  $\Delta w, \Delta b$  on current mini-batch.

$$v_{dw} = \beta v_{dw} + (1-\beta) \Delta w$$

$$v_{db} = \beta v_{db} + (1-\beta) \Delta b$$

Friction ↑ velocity

$$w := w - \alpha v_{dw}$$



$$v_{\theta} = \beta v_{\theta} + (1-\beta) \theta_t$$

deeplearning.ai  
by Andrew Ng

# Implementation details

$$v_{dw} = 0, v_{db} = 0$$

On iteration  $t$ :

Compute  $dW, db$  on the current mini-batch

$$\begin{aligned} \rightarrow v_{dw} &= \beta v_{dw} + (1 - \beta) dW \\ \rightarrow v_{db} &= \beta v_{db} + (1 - \beta) db \end{aligned} \quad \left| \begin{array}{l} v_{dw} = \beta v_{dw} + dW \leftarrow \\ v_{db} = \beta v_{db} + db \end{array} \right.$$
$$W = W - \underbrace{\alpha v_{dw}}, b = \underbrace{b - \alpha v_{db}}$$

$$\frac{v_{dw}}{1 - \beta^t}$$

Hyperparameters:  $\alpha, \beta$

$$\beta = 0.9$$

average over last  $\approx 10$  gradients



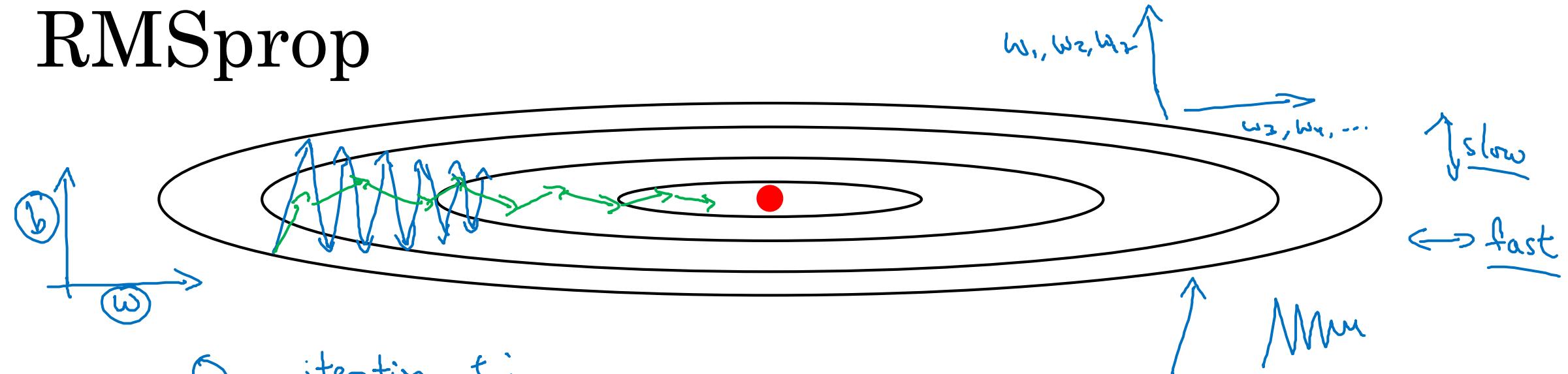
deeplearning.ai

# Optimization Algorithms

---

## RMSprop

# RMSprop



On iteration  $t$ :

Compute  $d\mathbf{w}, d\mathbf{b}$  on current mini-batch

$$\underline{S_{d\mathbf{w}}} = \beta_2 \underline{S_{d\mathbf{w}}} + (1-\beta_2) \underline{d\mathbf{w}^2} \quad \begin{matrix} \text{element-wise} \\ \leftarrow \text{small} \end{matrix}$$

$$\rightarrow \underline{S_{d\mathbf{b}}} = \beta_2 \underline{S_{d\mathbf{b}}} + (1-\beta_2) \underline{d\mathbf{b}^2} \quad \leftarrow \text{large}$$

$$\mathbf{w} := \mathbf{w} - \frac{\alpha}{\sqrt{\underline{S_{d\mathbf{w}}} + \epsilon}} \underline{d\mathbf{w}}$$

$$\mathbf{b} := \mathbf{b} - \frac{\alpha}{\sqrt{\underline{S_{d\mathbf{b}}} + \epsilon}} \underline{d\mathbf{b}}$$

$$\epsilon = 10^{-8}$$



deeplearning.ai

# Optimization Algorithms

---

## Adam optimization algorithm

# Adam optimization algorithm

$$V_{dw} = 0, S_{dw} = 0. \quad V_{db} = 0, S_{db} = 0$$

On iteration  $t$ :

Compute  $\delta w, \delta b$  using current mini-batch

$$V_{dw} = \beta_1 V_{dw} + (1 - \beta_1) \delta w, \quad V_{db} = \beta_1 V_{db} + (1 - \beta_1) \delta b \quad \leftarrow \text{"moment"} \beta_1$$

$$S_{dw} = \beta_2 S_{dw} + (1 - \beta_2) \delta w^2, \quad S_{db} = \beta_2 S_{db} + (1 - \beta_2) \delta b \quad \leftarrow \text{"RMSprop"} \beta_2$$

$yhat = np.array([.9, 0.2, 0.1, .4, .9])$

$$V_{dw}^{corrected} = V_{dw} / (1 - \beta_1^t), \quad V_{db}^{corrected} = V_{db} / (1 - \beta_1^t)$$

$$S_{dw}^{corrected} = S_{dw} / (1 - \beta_2^t), \quad S_{db}^{corrected} = S_{db} / (1 - \beta_2^t)$$

$$w := w - \alpha \frac{V_{dw}^{corrected}}{\sqrt{S_{dw}^{corrected} + \epsilon}}$$

$$b := b - \alpha \frac{V_{db}^{corrected}}{\sqrt{S_{db}^{corrected} + \epsilon}}$$

# Hyperparameters choice:

- $\alpha$  : needs to be tune
- $\beta_1$  : 0.9       $\rightarrow (\underline{dw})$
- $\beta_2$  : 0.999     $\rightarrow (\underline{dw^2})$
- $\epsilon$  :  $10^{-8}$

Adam: Adaptive moment estimation



Adam Coates



deeplearning.ai

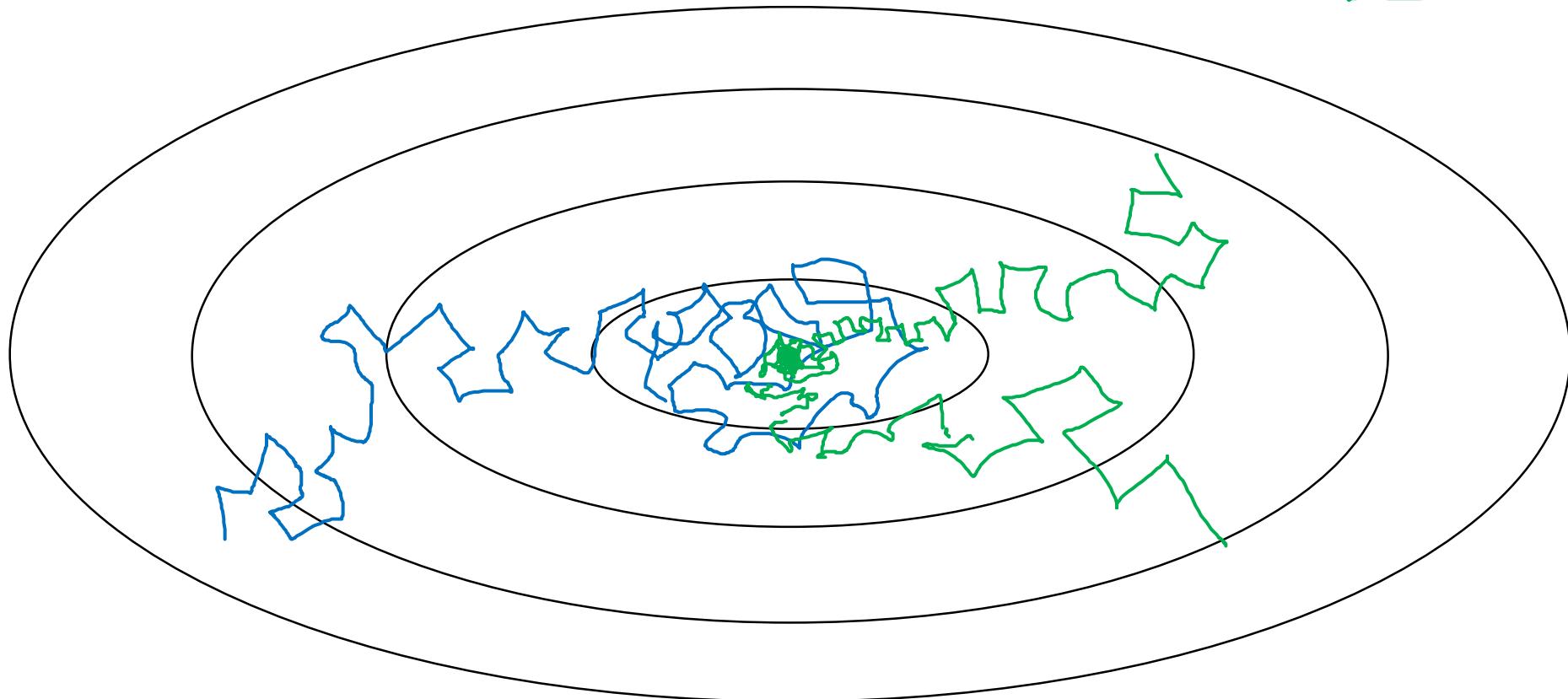
# Optimization Algorithms

---

## Learning rate decay

# Learning rate decay

Slowly reduce  $\lambda$

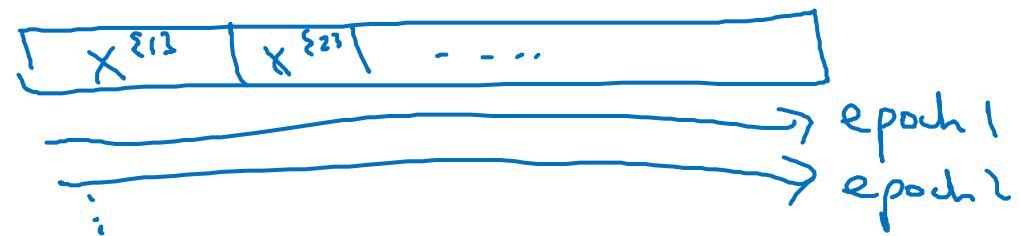


# Learning rate decay

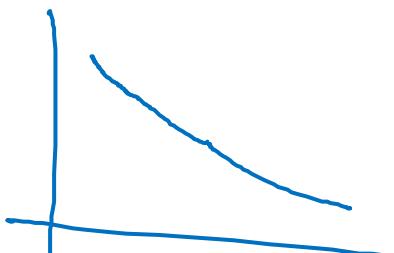
1 epoch = 1 pass through data.

$$\alpha = \frac{\alpha_0}{1 + \text{decay-rate} * \text{epoch-num}}$$

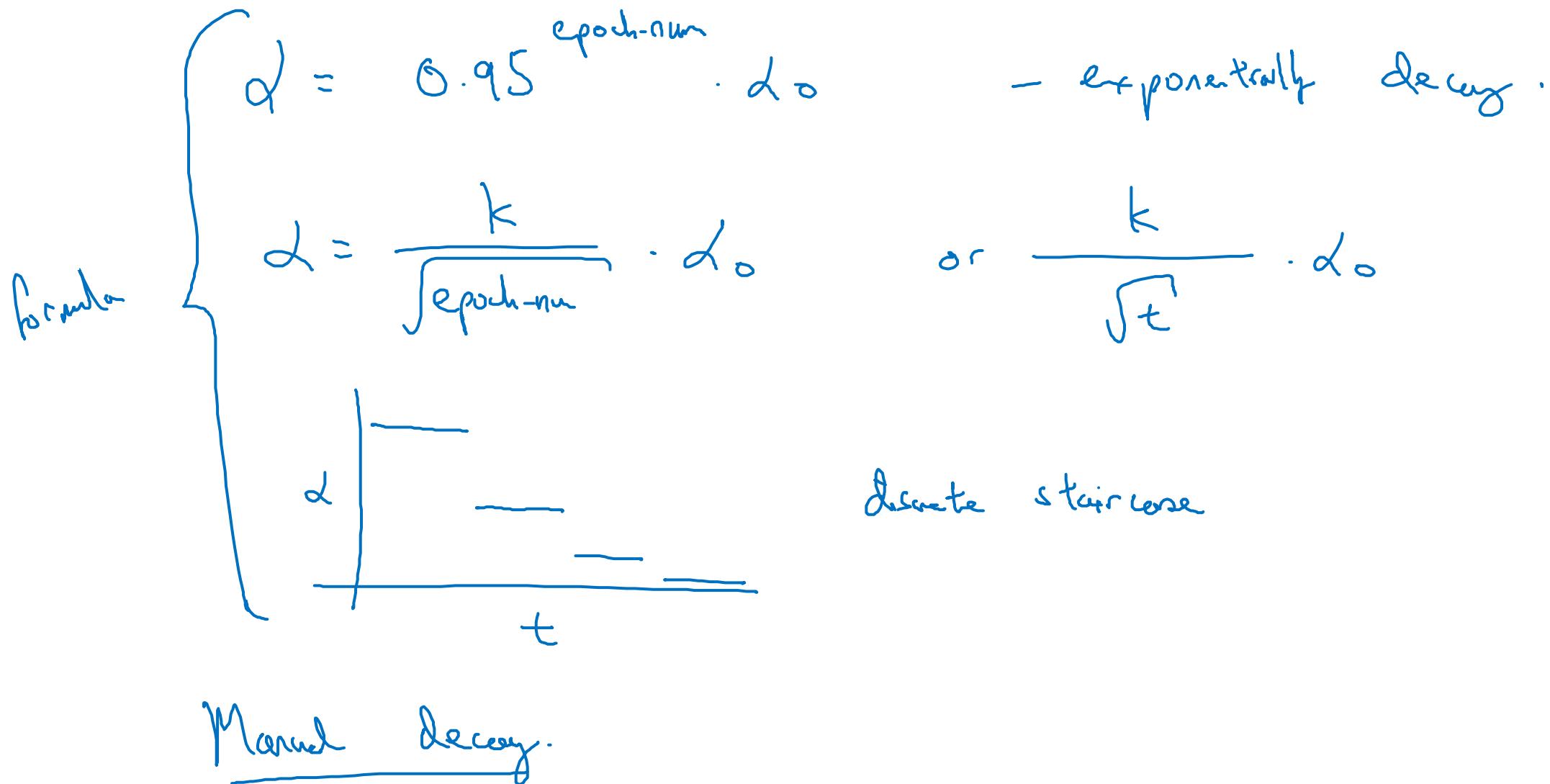
Epoch	$\alpha$
1	0.1
2	0.67
3	0.5
4	0.4
:	:



$$\alpha_0 = 0.2$$
$$\text{decay-rate} = 1$$



# Other learning rate decay methods





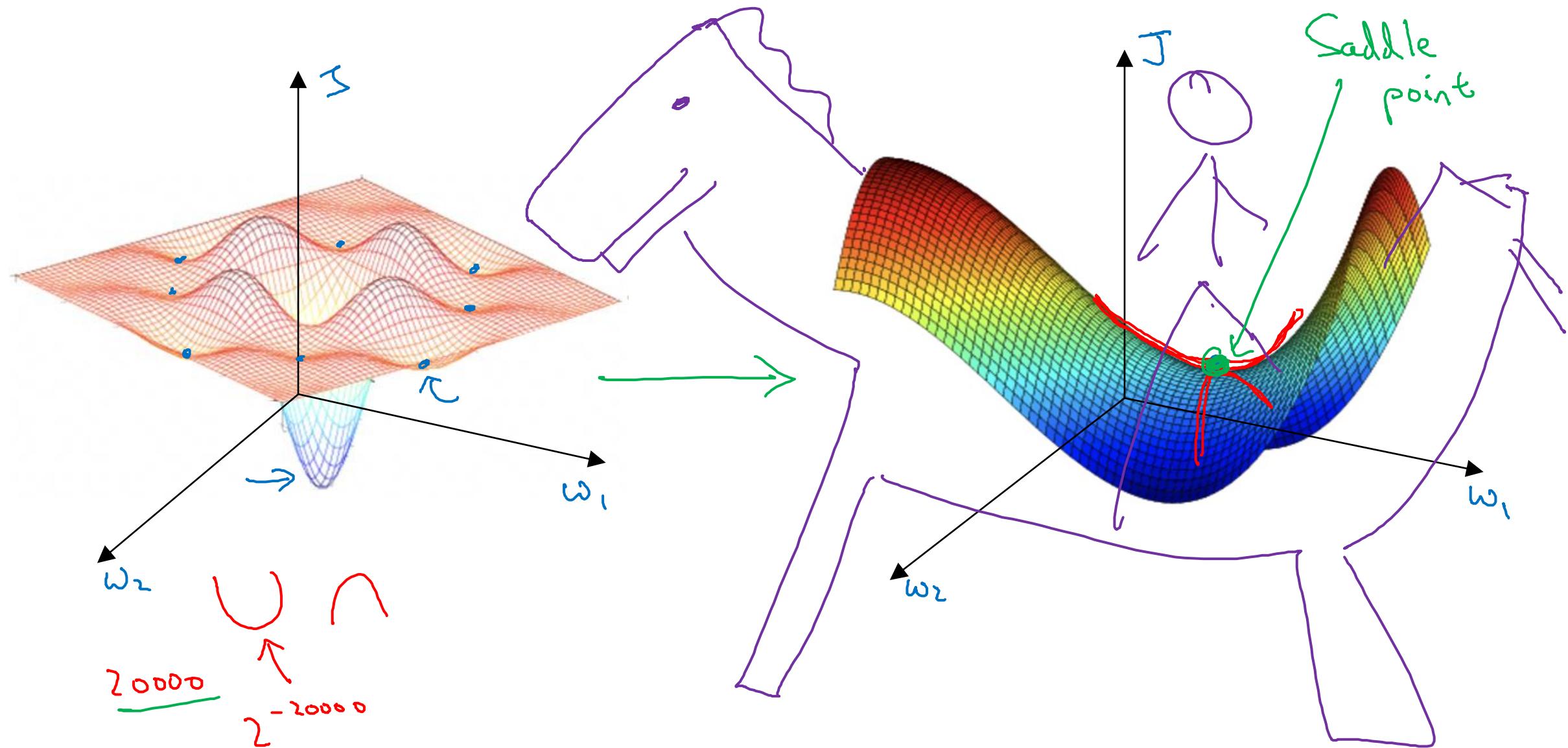
deeplearning.ai

# Optimization Algorithms

---

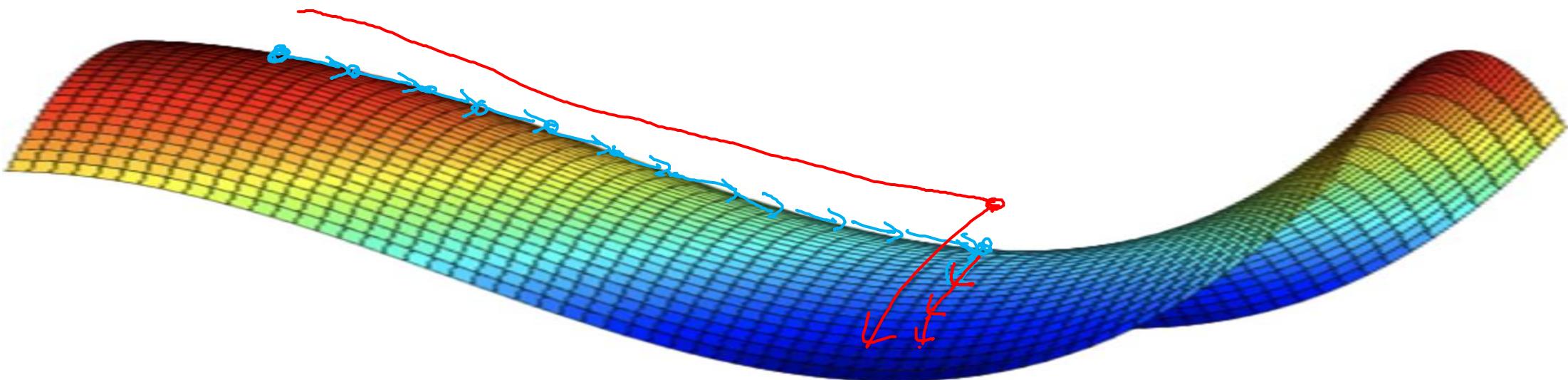
## The problem of local optima

# Local optima in neural networks



Andrew Ng

# Problem of plateaus



- Unlikely to get stuck in a bad local optima
- Plateaus can make learning slow



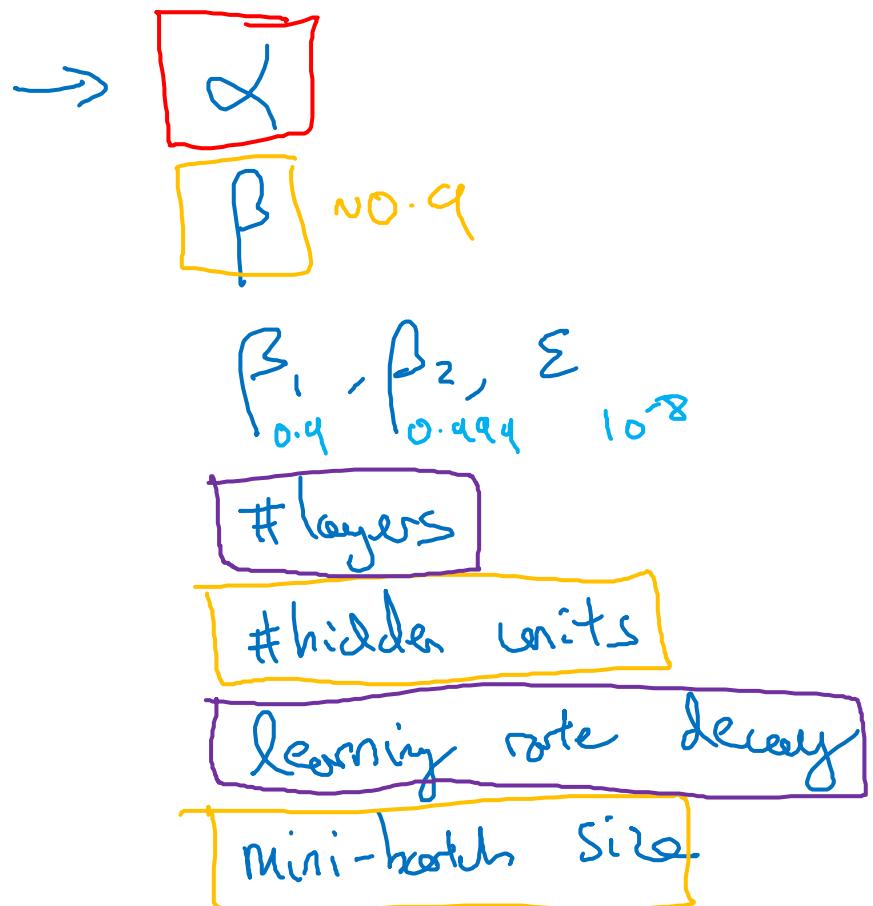
deeplearning.ai

# Hyperparameter tuning

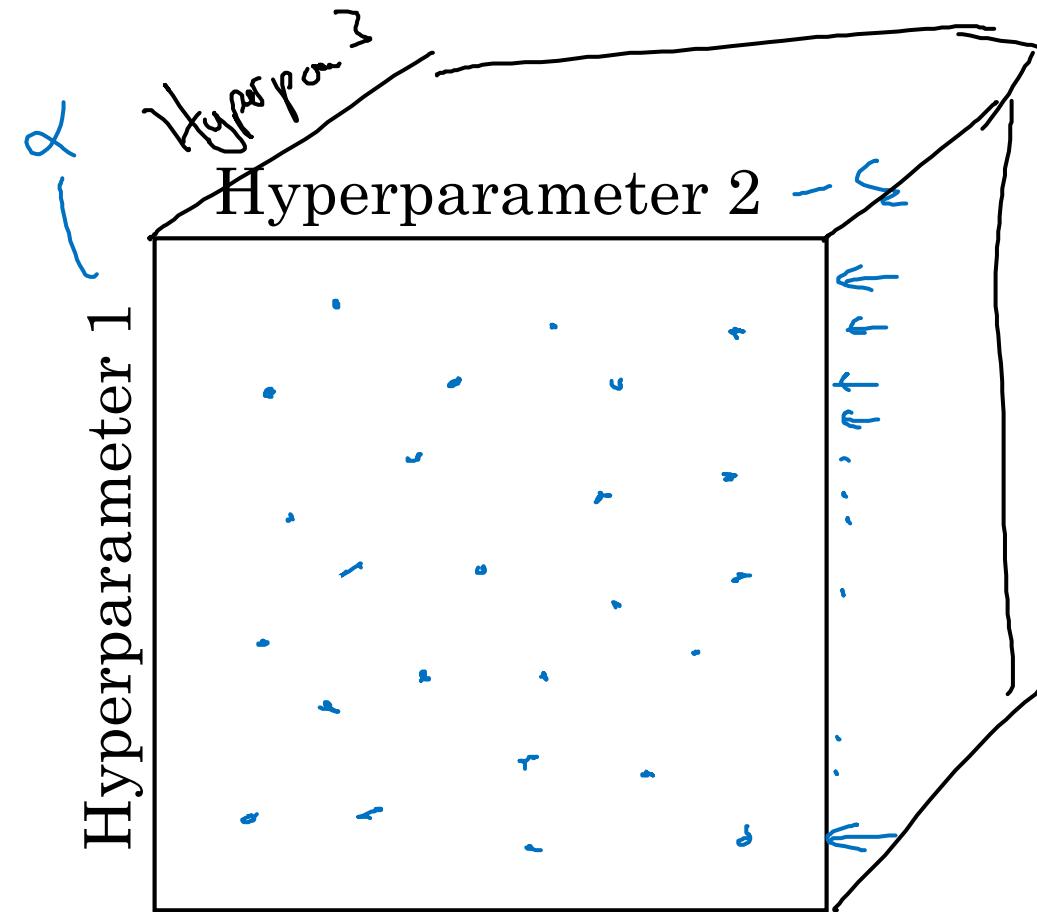
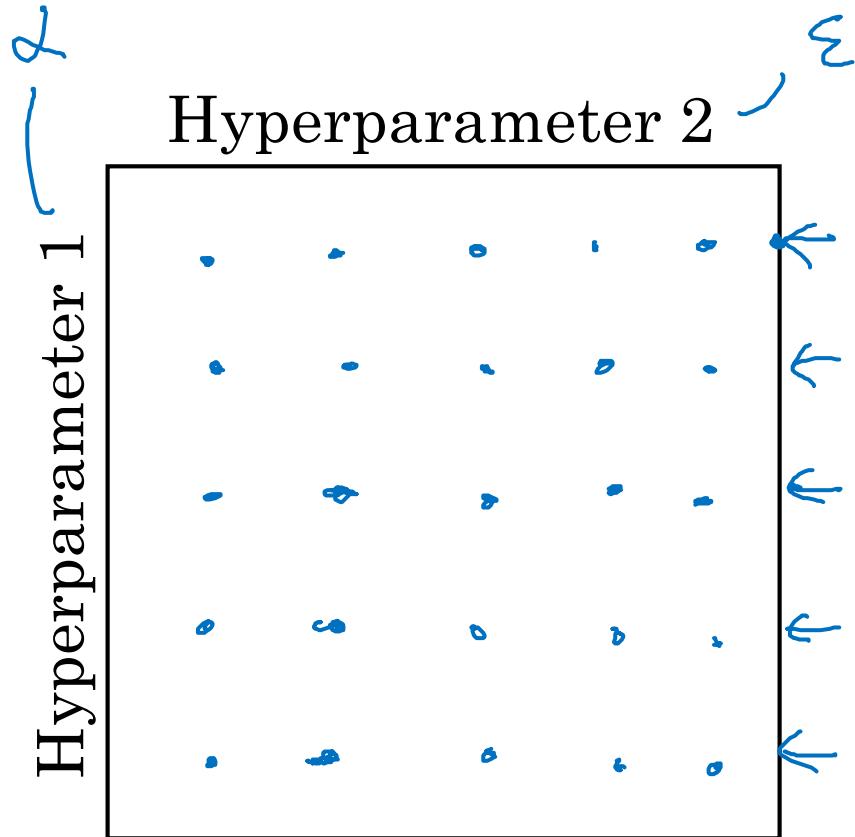
---

## Tuning process

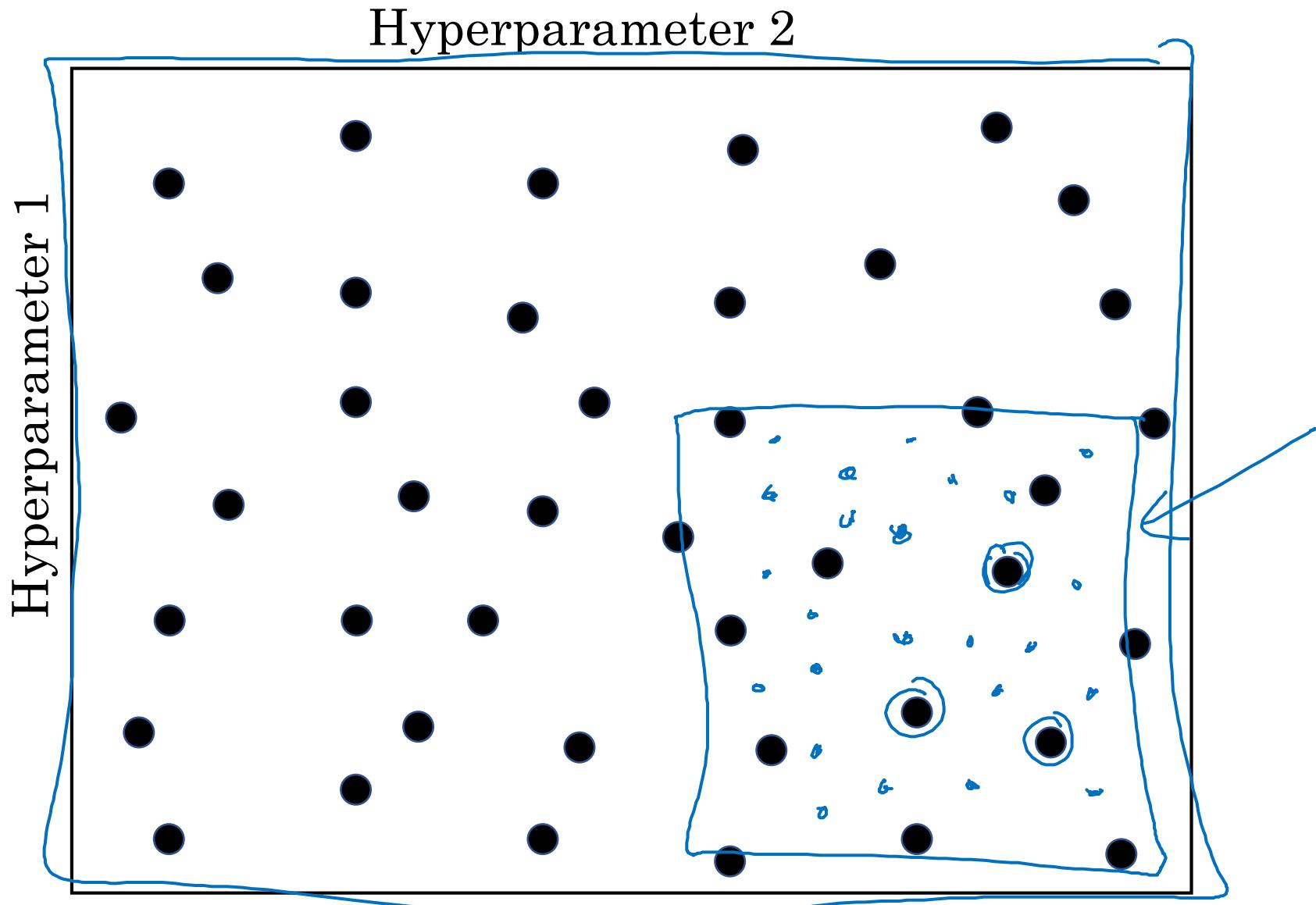
# Hyperparameters



# Try random values: Don't use a grid



# Coarse to fine





deeplearning.ai

# Hyperparameter tuning

---

Using an appropriate  
scale to pick  
hyperparameters

# Picking hyperparameters at random

→  $n^{[l]} = 50, \dots, 100$

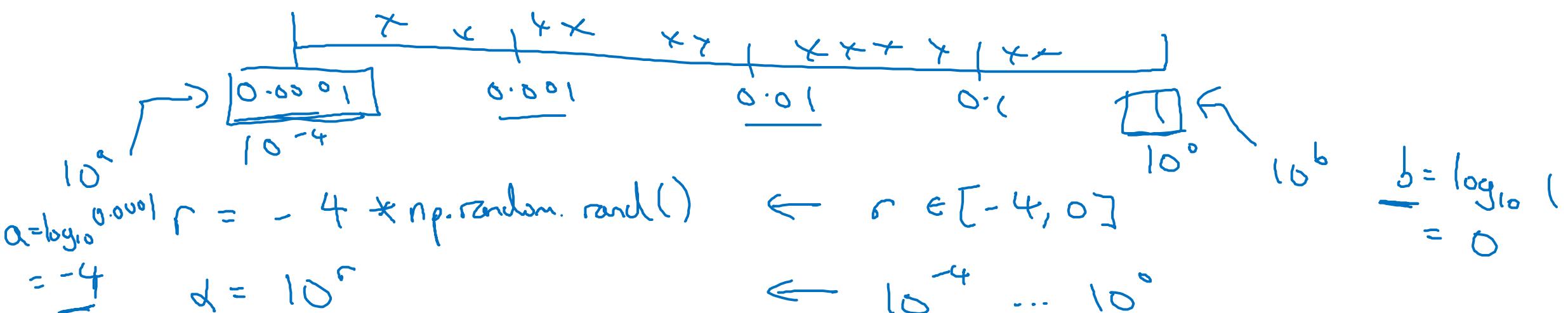
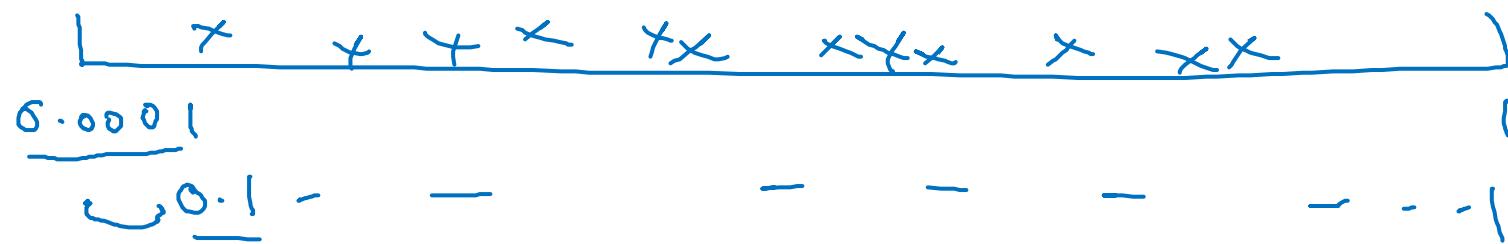


→ #layers  $L : 2 - 4$

2, 3, 4

# Appropriate scale for hyperparameters

$$\lambda = 0.0001, \dots, 1$$



$$10^a \dots 10^b$$

$$\frac{r \in [a, b]}{[-4, 0]}$$

$$\lambda = 10^r$$

# Hyperparameters for exponentially weighted averages

$$\beta = 0.9 \dots 0.999$$

$\downarrow$                      $\downarrow$   
 $10$                      $1000$

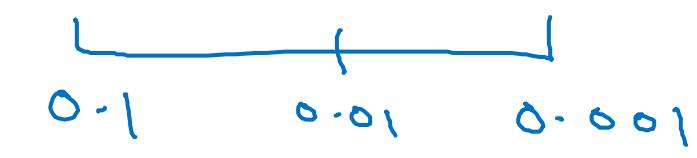
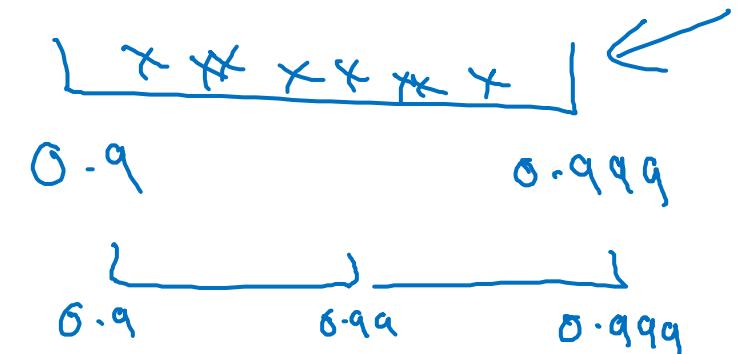
$$1-\beta = 0.1 \dots 0.001$$

$$\beta: 0.900 \rightarrow 0.9005 \quad \} \sim 10$$

$$\beta: 0.999 \rightarrow 0.9995$$

$\sim 1000$                      $\sim 2000$

$$\frac{1}{1-\beta}$$



$$\frac{10^{-1}}{1-\beta} \quad \frac{10^{-3}}{\beta}$$

$r \in [-3, -1]$

$$1-\beta = 10^r$$

$$\beta = 1 - 10^r$$



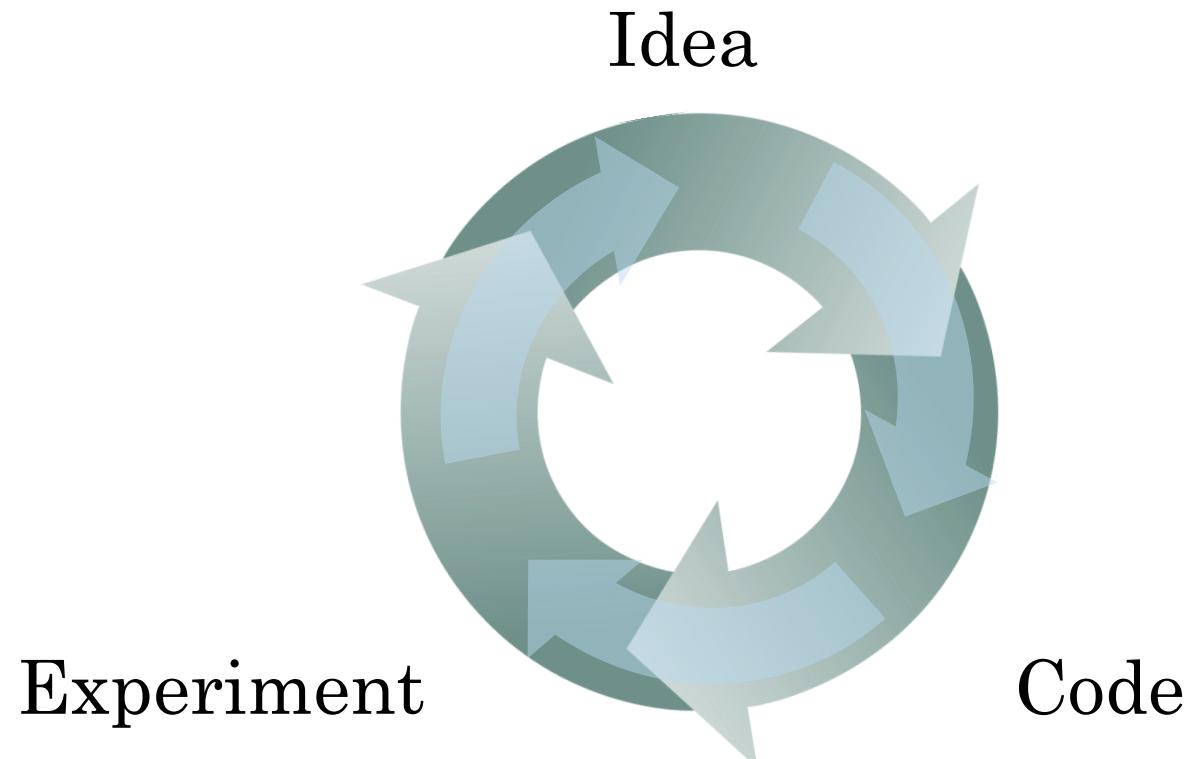
deeplearning.ai

# Hyperparameters tuning

---

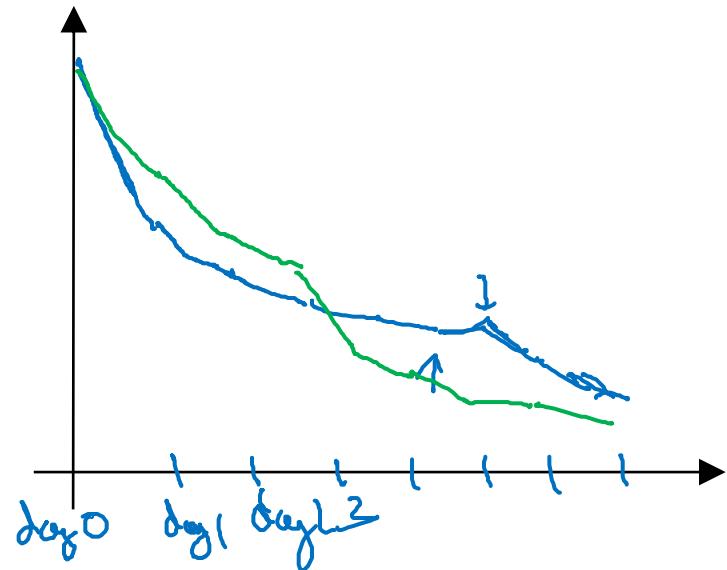
## Hyperparameters tuning in practice: Pandas vs. Caviar

# Re-test hyperparameters occasionally



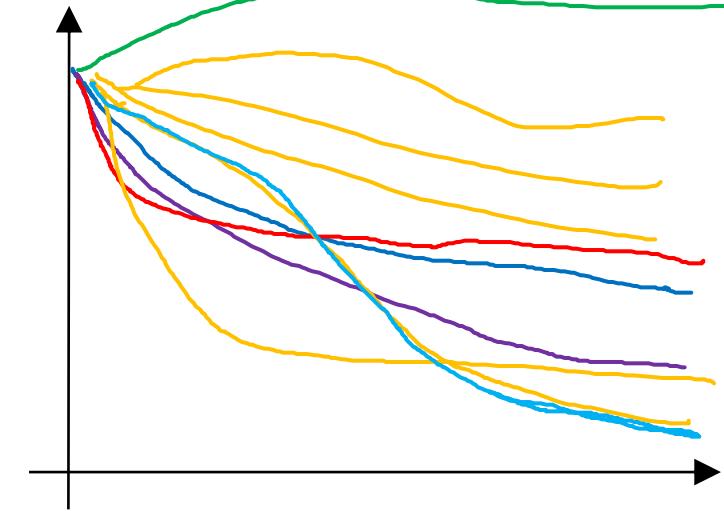
- NLP, Vision, Speech,  
Ads, logistics, ....
- Intuitions do get stale.  
Re-evaluate occasionally.

# Babysitting one model



Panda ↵

# Training many models in parallel



Caviar ↵

Andrew Ng



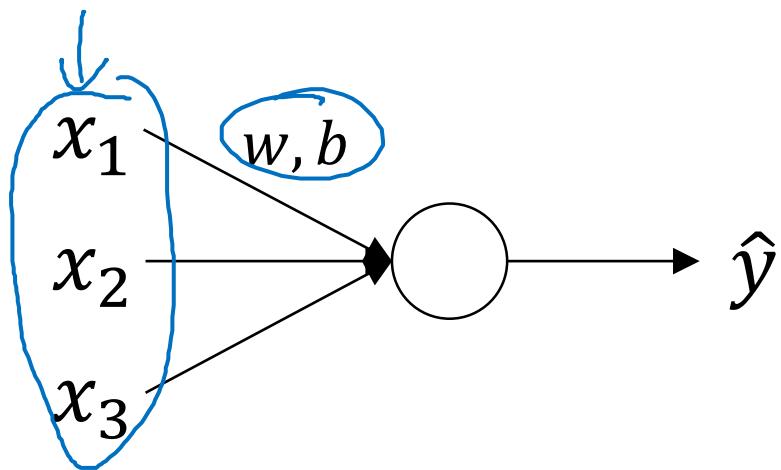
deeplearning.ai

# Batch Normalization

---

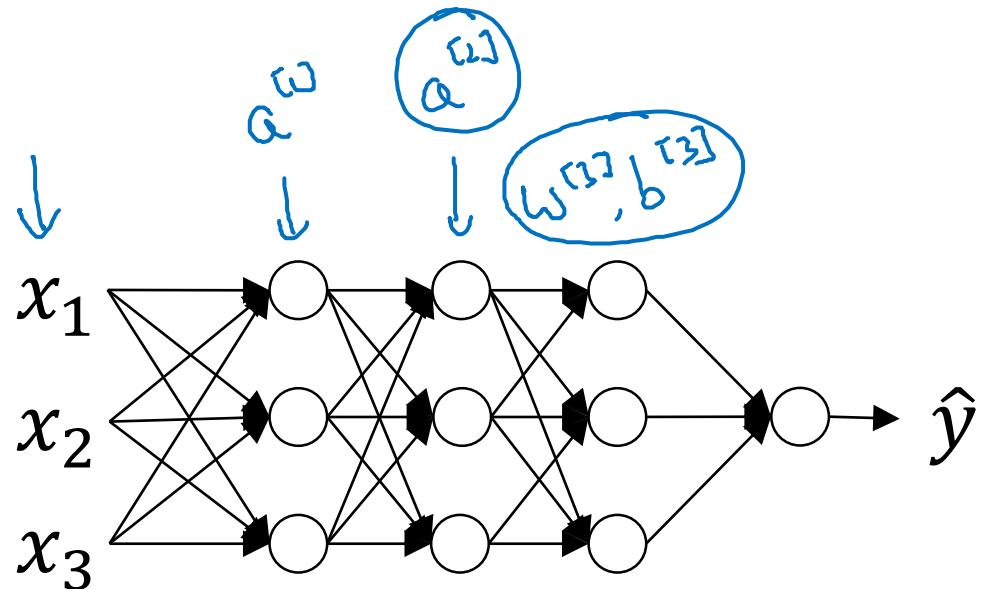
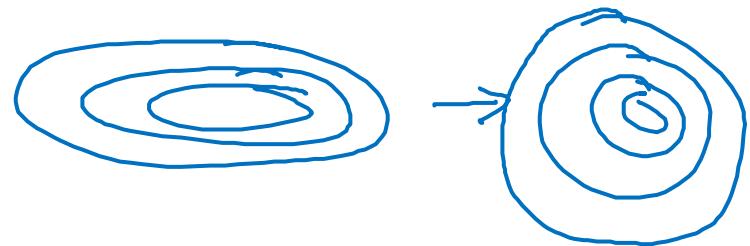
## Normalizing activations in a network

# Normalizing inputs to speed up learning



$$\mu = \frac{1}{m} \sum_i x^{(i)}$$
$$X = X - \mu$$
$$\sigma^2 = \frac{1}{m} \sum_i (x^{(i)} - \mu)^2$$
$$X = X / \sigma^2$$

element-wise



Can we normalize  $\frac{a^{[2]}}{w^{[2]}, b^{[2]}}$  so  
as to train  $w^{[2]}, b^{[2]}$  faster

Normalize  $\frac{z^{[2]}}{\uparrow}$

# Implementing Batch Norm

Given some intermediate values in NN

$$\mu = \frac{1}{m} \sum_i z^{(i)}$$

$$\sigma^2 = \frac{1}{m} \sum_i (z^{(i)} - \mu)^2$$

$$z_{\text{norm}}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

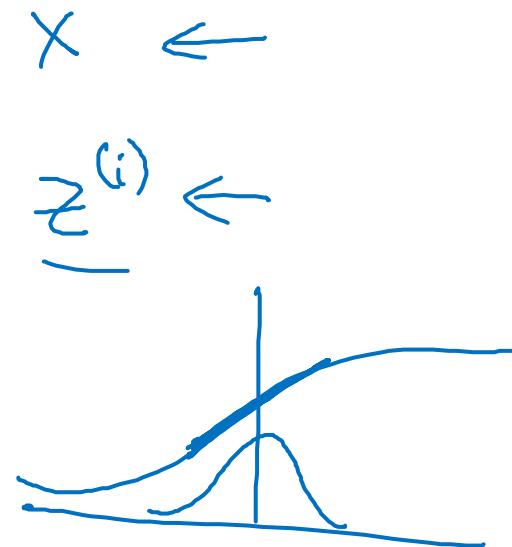
$$\hat{z}^{(i)} = \gamma z_{\text{norm}}^{(i)} + \beta$$

Use  $\hat{z}^{(i)}$  instead of  $z^{(i)}$ .

If  $\gamma = \sqrt{\sigma^2 + \epsilon}$  ←  
then  $\hat{z}^{(i)} = z^{(i)}$  ←

learnable parameters  
of model.

$$\begin{aligned} & \downarrow \quad \downarrow \\ & z^{(1)}, \dots, z^{(m)} \\ & \quad \quad \quad z^{[l]}(:) \end{aligned}$$





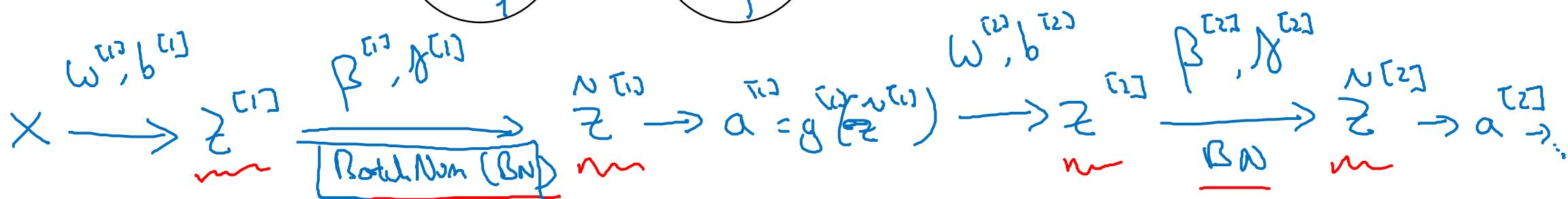
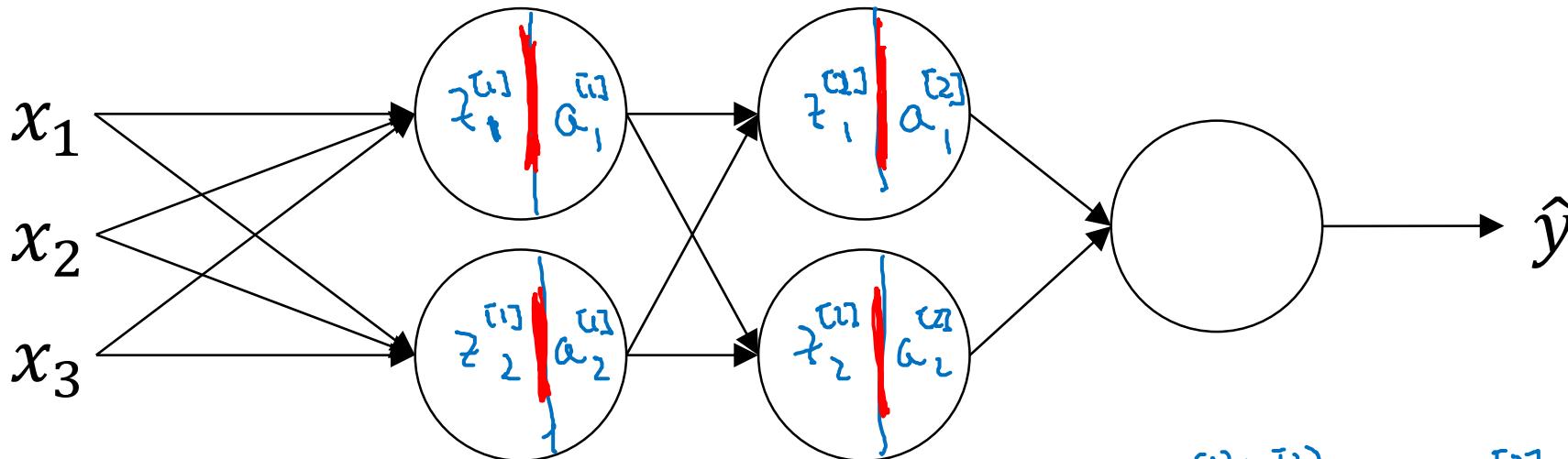
deeplearning.ai

# Batch Normalization

---

Fitting Batch Norm  
into a neural network

# Adding Batch Norm to a network



Parameters:  $w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}, \dots, w^{[L]}, b^{[L]}, \{ \beta^{[1]}, \gamma^{[1]}, \beta^{[2]}, \gamma^{[2]}, \dots, \beta^{[L]}, \gamma^{[L]} \}$

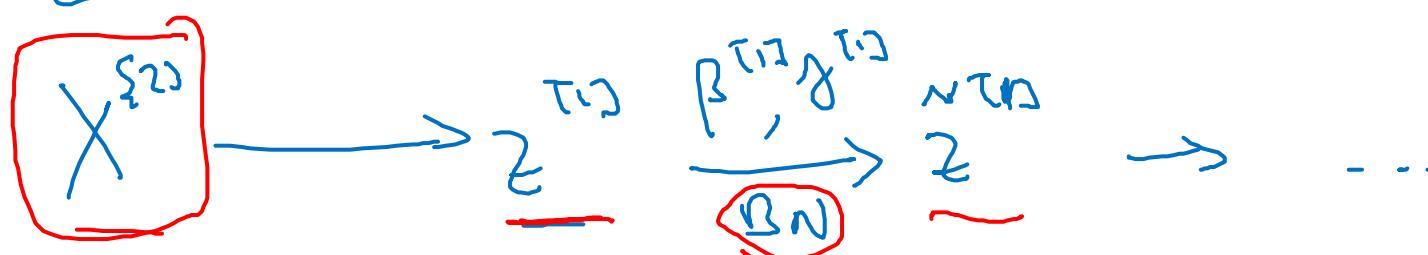
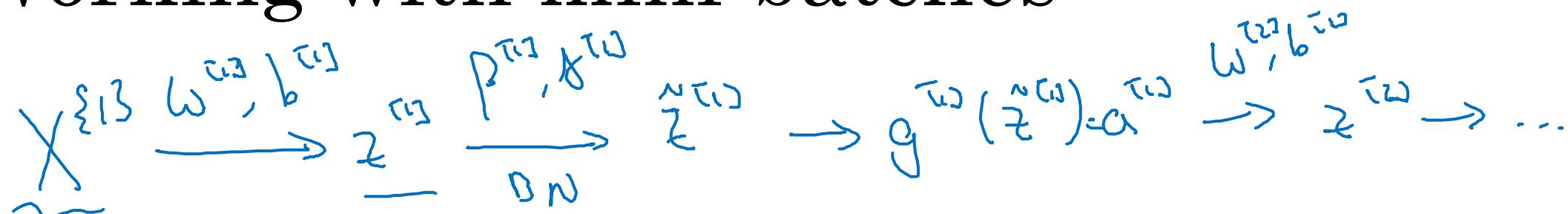
$\rightarrow \beta^{[1]}, \gamma^{[1]}, \beta^{[2]}, \gamma^{[2]}, \dots, \beta^{[L]}, \gamma^{[L]}$

$\rightarrow \beta$

$$\beta = \bar{\beta} - d \delta \beta^{[L]}$$

`tf.nn.batch_normalization` ←

# Working with mini-batches



$X^{[2]} \quad \longrightarrow \quad \dots$

Parameters:  $W^{[1]}, \cancel{b^{[1]}}, \beta^{[1]}, \gamma^{[1]}$ .

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

$$\begin{matrix} W^{[1]} \\ (n^{[1]}, n^{[2]}) \end{matrix} \quad \begin{matrix} \cancel{b^{[1]}} \\ (n^{[1]}, 1) \end{matrix} \quad \begin{matrix} \beta^{[1]} \\ (n^{[1]}, 1) \end{matrix} \quad \begin{matrix} \gamma^{[1]} \\ (n^{[1]}, 1) \end{matrix}$$

$$\begin{aligned} \cancel{b^{[1]}} &= W^{[1]} a^{[1]} + \cancel{b^{[1]}} \\ Z^{[1]} &= W^{[1]} a^{[1]} \\ Z^{[2]} &= W^{[2]} a^{[2]} \\ \cancel{b^{[2]}} &= \gamma^{[2]} Z^{[2]} + \beta^{[2]} \end{aligned}$$

Andrew Ng

# Implementing gradient descent

for  $t = 1 \dots \text{num MiniBatches}$   
Compute forward prop on  $X^{[t]}$ .

In each hidden layer, use BN to replace  $\underline{z}^{[l]}$  with  $\hat{\underline{z}}^{[l]}$ .

Use backprop to compute  $\underline{dw}^{[l]}$ ,  ~~$\underline{db}^{[l]}$~~ ,  $\underline{d\beta}^{[l]}$ ,  $\underline{dg}^{[l]}$

Update parameters  $\left. \begin{array}{l} w^{[l]} := w^{[l]} - \alpha \underline{dw}^{[l]} \\ \beta^{[l]} := \beta^{[l]} - \alpha \underline{d\beta}^{[l]} \\ g^{[l]} := \dots \end{array} \right\} \leftarrow$

Works w/ momentum, RMSprop, Adam.



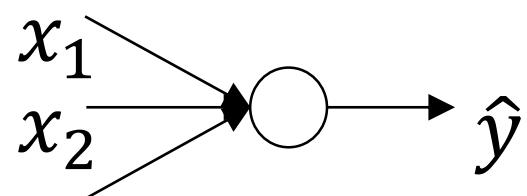
deeplearning.ai

# Batch Normalization

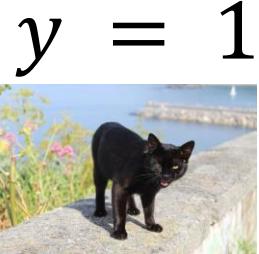
---

Why does  
Batch Norm work?

# Learning on shifting input distribution



Cat



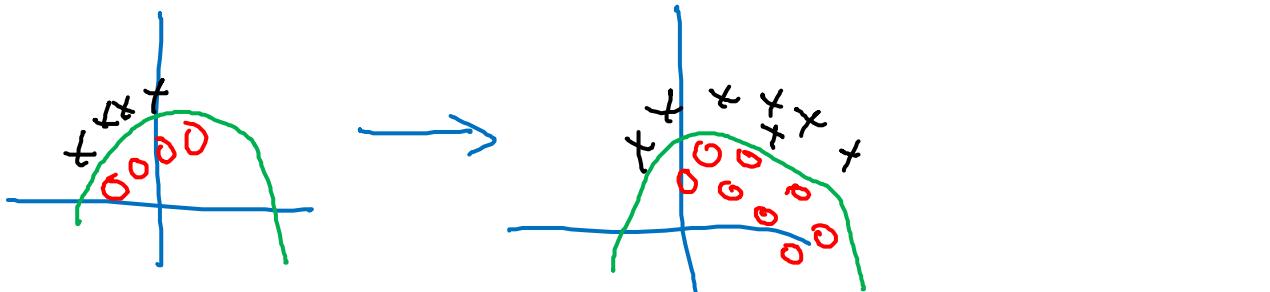
Non-Cat



$$y = 1$$

↙

$$y = 0$$



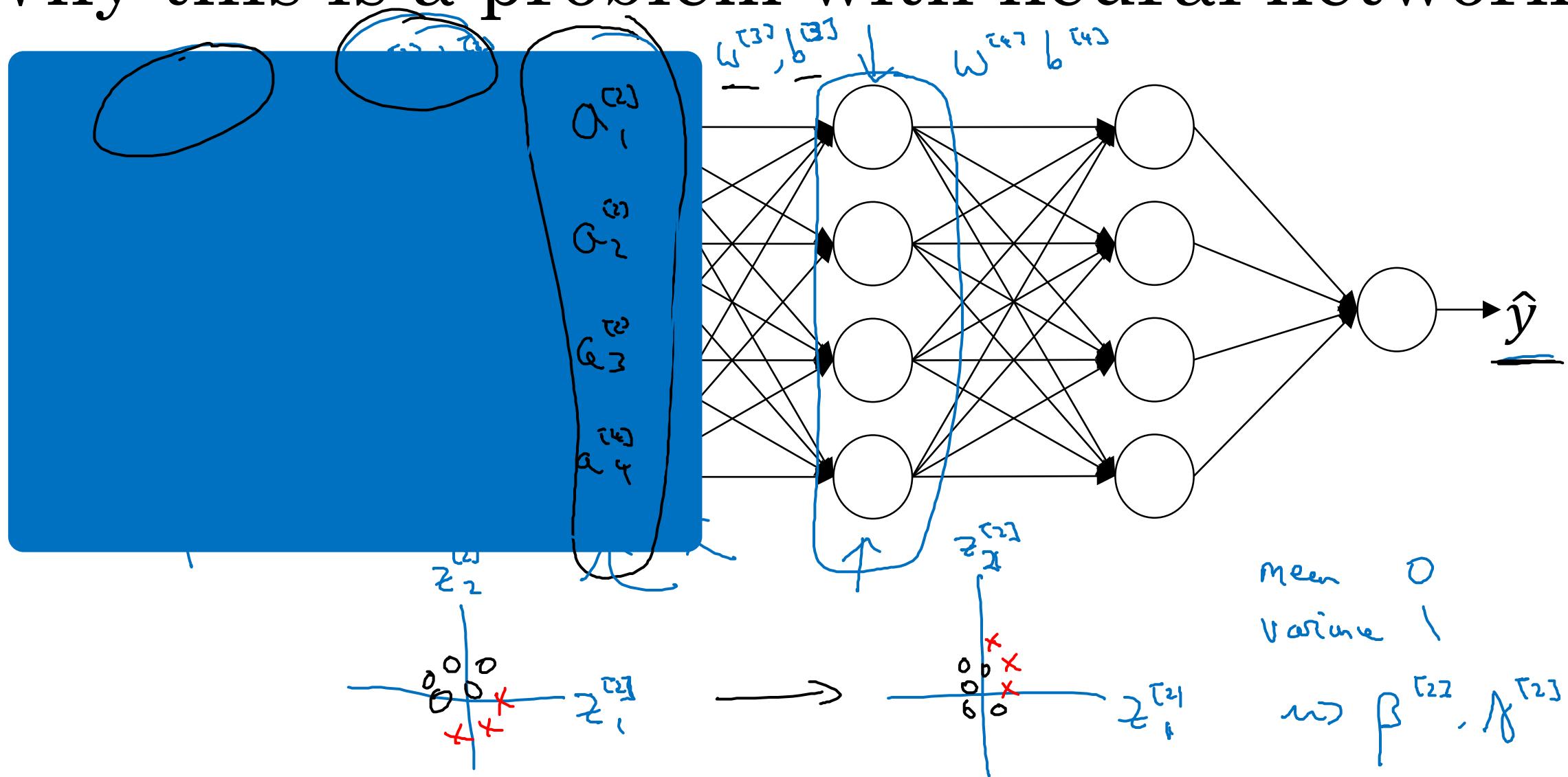
$$y = 1 \quad \swarrow \quad y = 0$$



"Covariate shift"

X → Y

# Why this is a problem with neural networks?



# Batch Norm as regularization

X

- Each mini-batch is scaled by the mean/variance computed on just that mini-batch.  
 $\xrightarrow{\hat{z}^{[l]}}$   $\mu, \sigma^2$   $\{z^{[l]}\}$   
 $\hat{z}^{[l]}$   $\underline{64}, \underline{128}$
- This adds some noise to the values  $z^{[l]}$  within that minibatch. So similar to dropout, it adds some noise to each hidden layer's activations.  
 $\mu, \sigma^2$
- This has a slight regularization effect.

mini-batch : 64  $\longrightarrow$  512



deeplearning.ai

# Batch Normalization

---

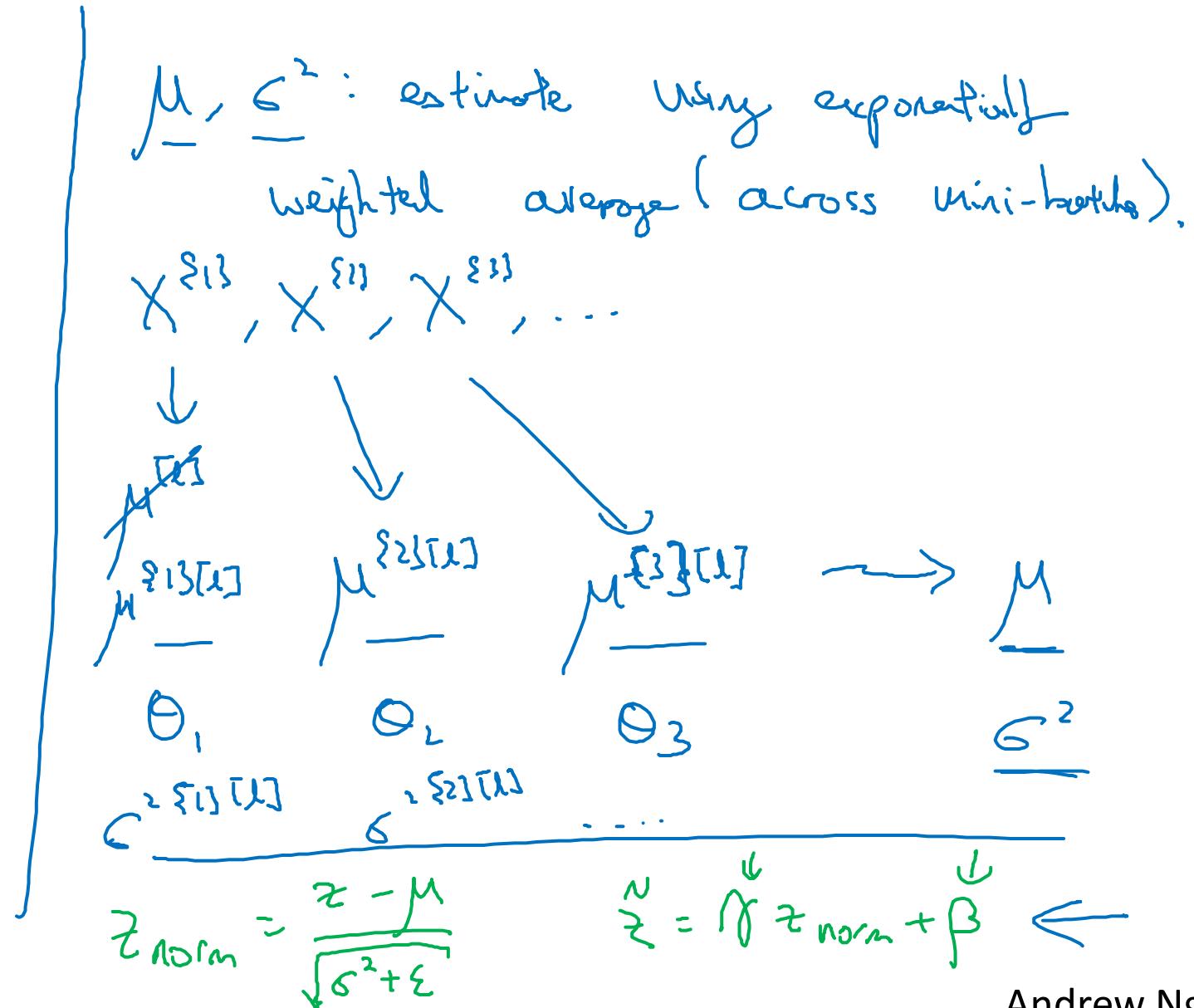
## Batch Norm at test time

# Batch Norm at test time

$$\mu = \frac{1}{m} \sum_i z^{(i)}$$
$$\sigma^2 = \frac{1}{m} \sum_i (z^{(i)} - \mu)^2$$

$$z_{\text{norm}}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

$$\tilde{z}^{(i)} = \gamma z_{\text{norm}}^{(i)} + \beta$$





deeplearning.ai

Multi-class  
classification

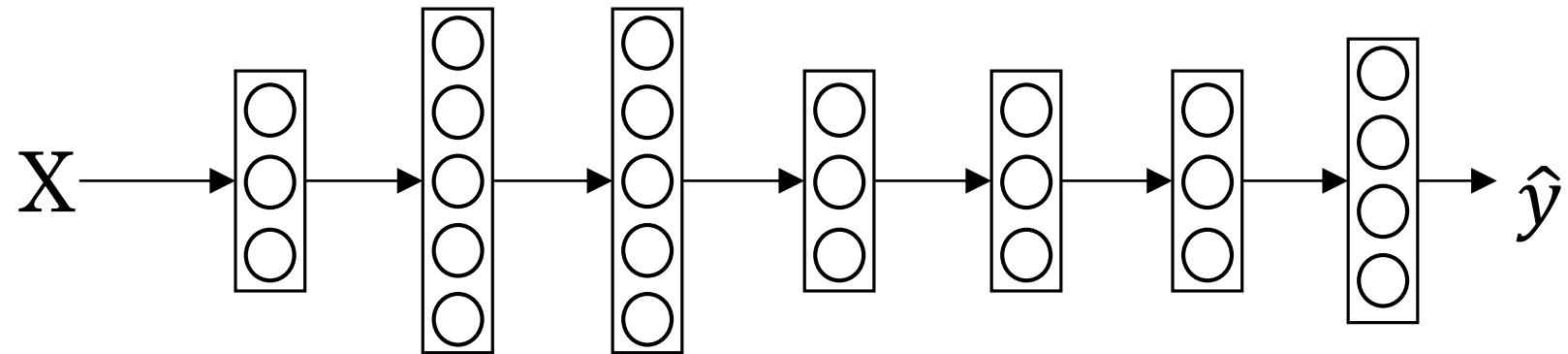
---

Softmax regression

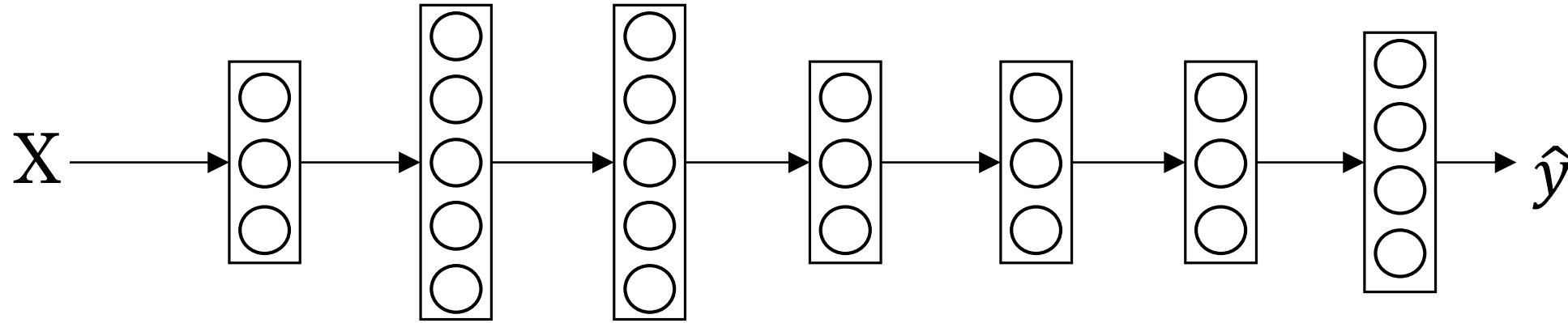
# Recognizing cats, dogs, and baby chicks



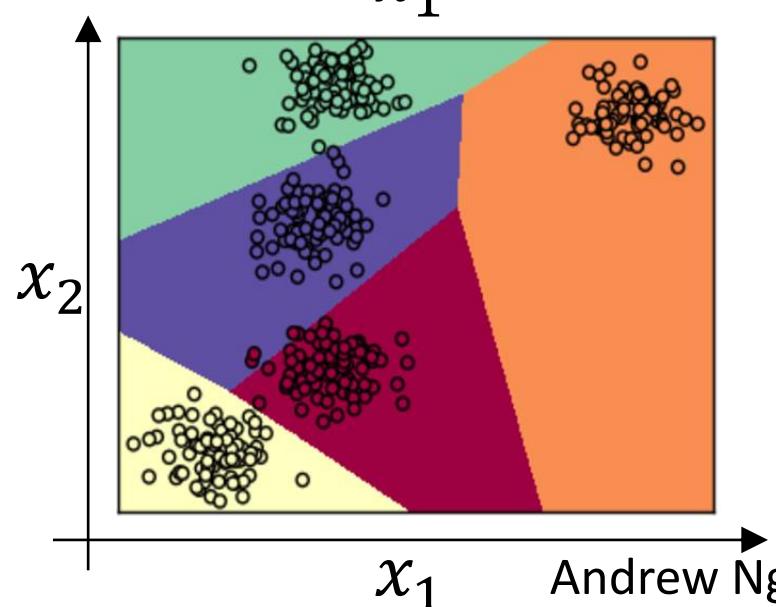
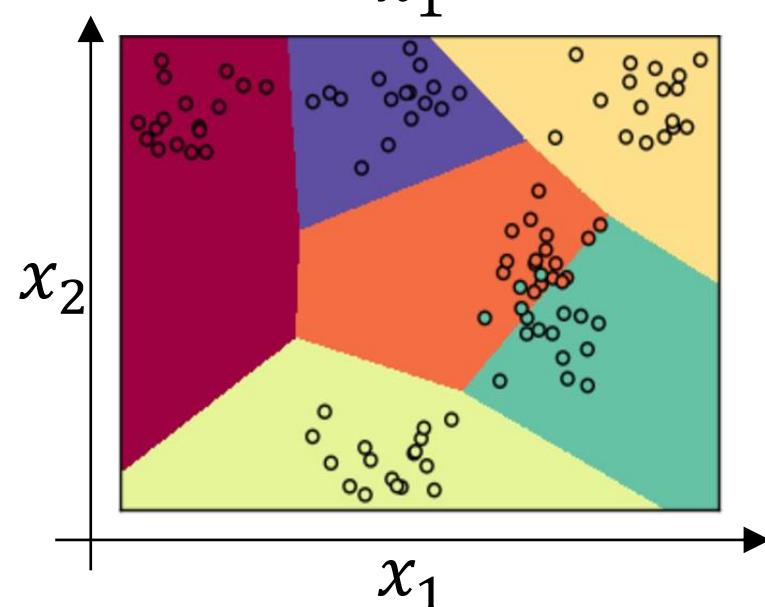
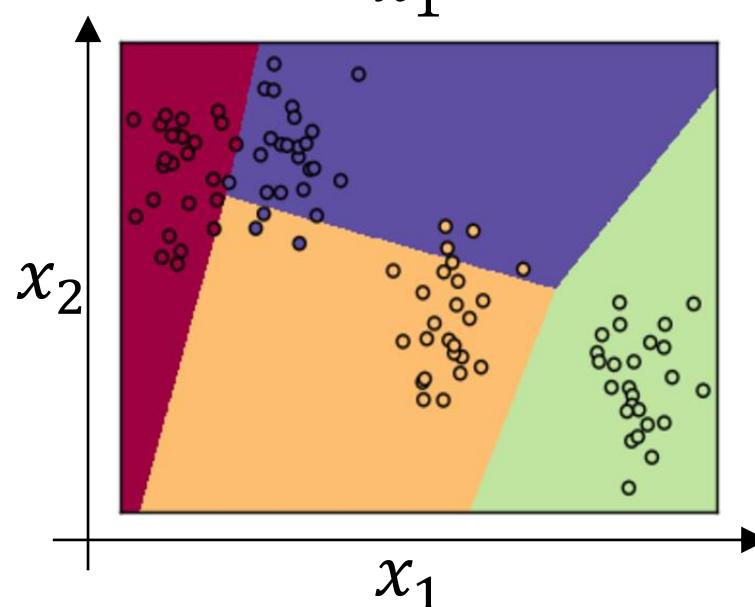
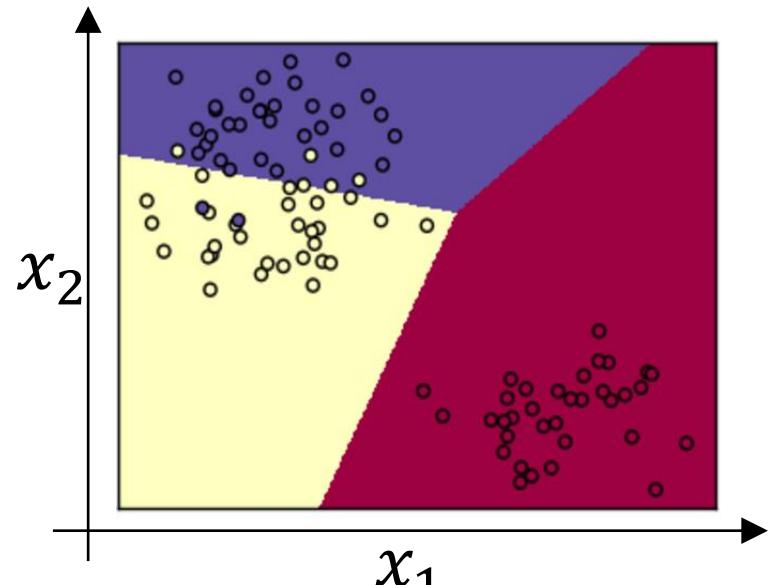
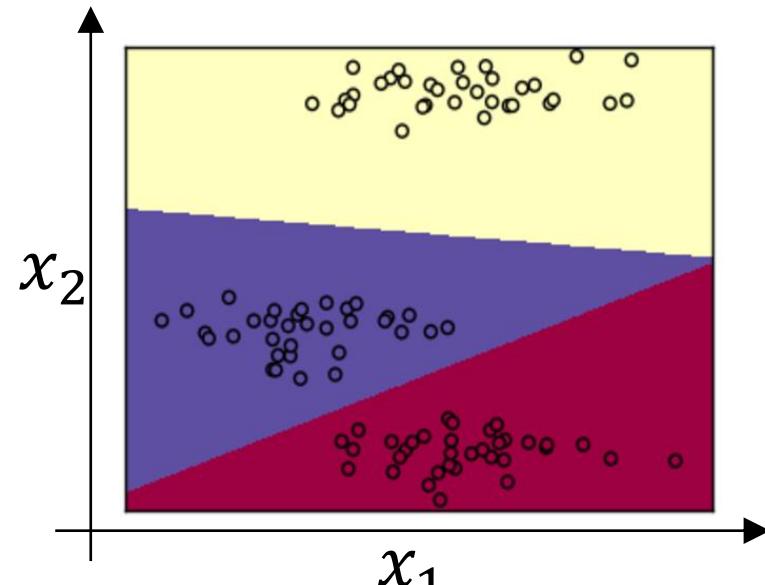
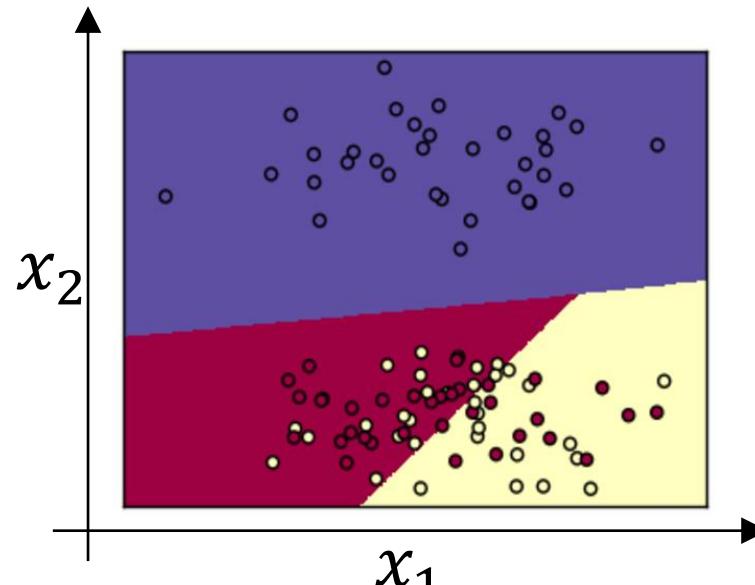
3            1            2            0            3            2            0            1



# Softmax layer



# Softmax examples



Andrew Ng



deeplearning.ai

Multi-class  
classification

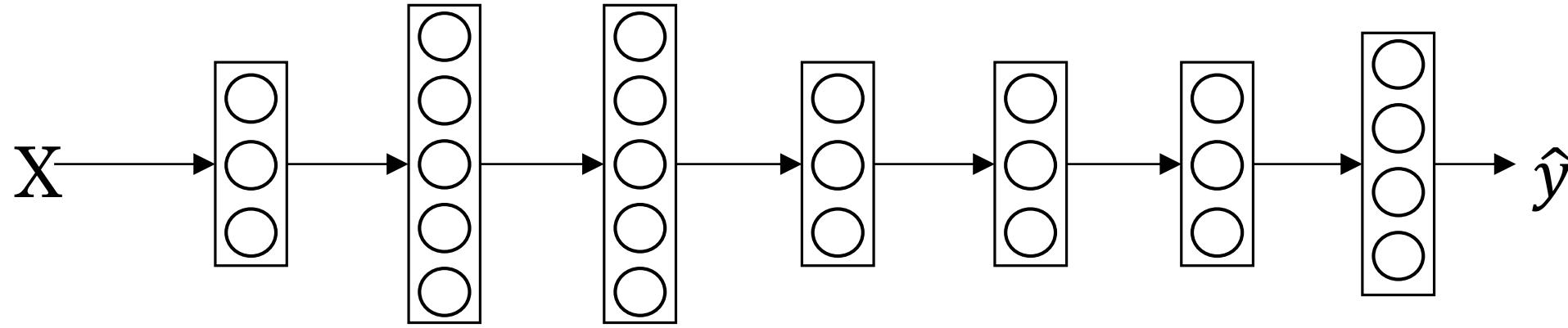
---

Trying a softmax  
classifier

# Understanding softmax

# Loss function

# Summary of softmax classifier





deeplearning.ai

# Programming Frameworks

---

# Deep Learning frameworks

# Deep learning frameworks

- Caffe/Caffe2
- CNTK
- DL4J
- Keras
- Lasagne
- mxnet
- PaddlePaddle
- TensorFlow
- Theano
- Torch

Choosing deep learning frameworks

- Ease of programming (development and deployment)
  - Running speed
- - Truly open (open source with good governance)



deeplearning.ai

# Programming Frameworks

---

## TensorFlow

# Motivating problem

$$J(\omega) = \frac{(\omega^2 - 10\omega + 25)}{(\omega - 5)^2}$$

$\omega = 5$

$J(\omega, b)$

↑ ↑

# Code example

```
import numpy as np  
import tensorflow as tf
```

```
coefficients = np.array([[1], [-20], [25]])
```

```
w = tf.Variable([0], dtype=tf.float32)
```

```
x = tf.placeholder(tf.float32, [3,1])
```

```
cost = x[0][0]*w**2 + x[1][0]*w + x[2][0] # (w-5)**2
```

```
train = tf.train.GradientDescentOptimizer(0.01).minimize(cost) ←
```

```
init = tf.global_variables_initializer()
```

```
session = tf.Session()
```

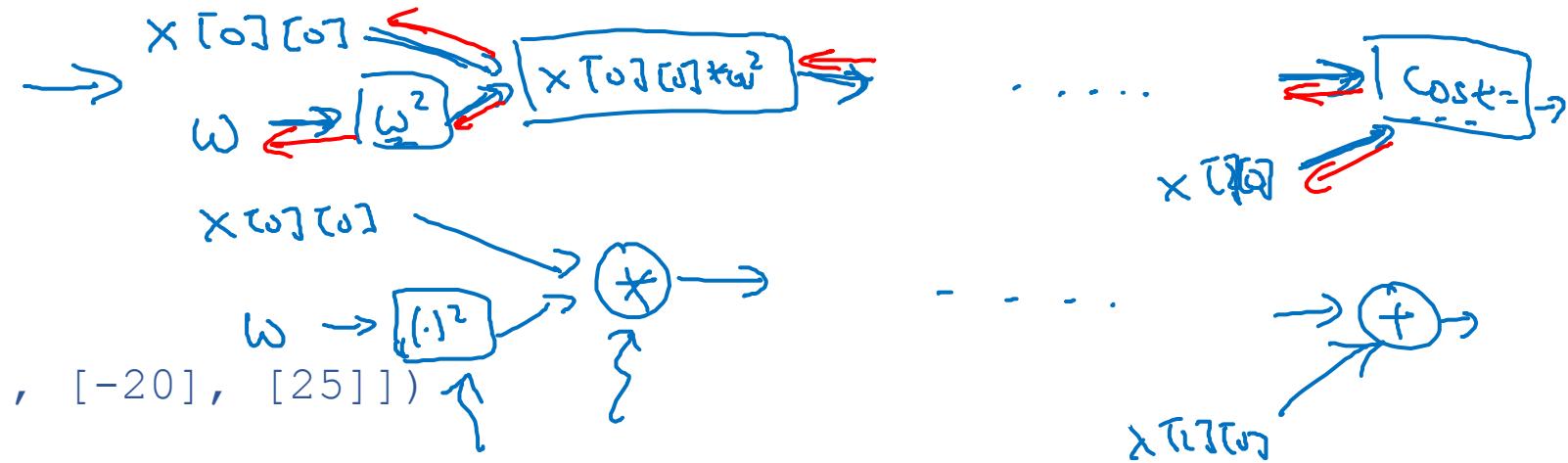
```
session.run(init)
```

```
print(session.run(w))
```

```
for i in range(1000):
```

```
    session.run(train, feed_dict={x:coefficients})
```

```
print(session.run(w))
```



```
with tf.Session() as session:  
    session.run(init) ←  
    print(session.run(w)) ←
```



deeplearning.ai

# Introduction to ML strategy

---

## Why ML Strategy?

# Motivating example



90%.

Ideas:

- Collect more data ←
- Collect more diverse training set
- Train algorithm longer with gradient descent
- Try Adam instead of gradient descent
- Try bigger network
- Try smaller network
- Try dropout
- Add  $L_2$  regularization
- Network architecture
  - Activation functions
  - # hidden units
  - ...



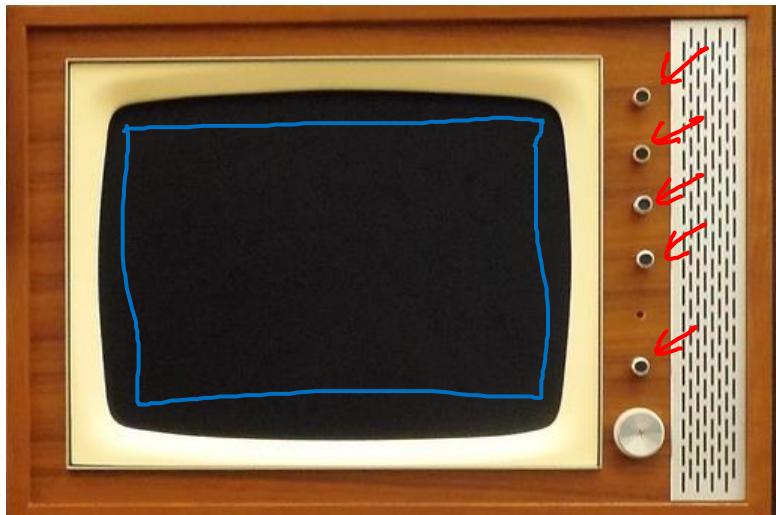
deeplearning.ai

# Introduction to ML strategy

---

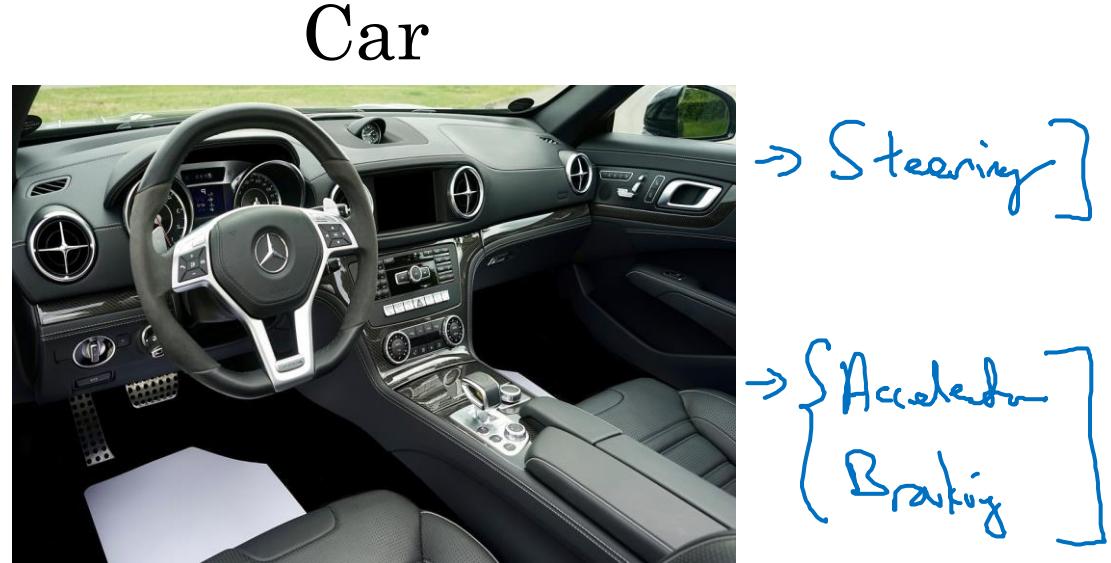
## Orthogonalization

# TV tuning example



Orthogonalization

$$\begin{aligned}
 & 0.1 \times \begin{array}{c} \uparrow \\ \square \end{array} \\
 + & 0.3 \times \begin{array}{c} \leftarrow \\ \square \end{array} \\
 - & 1.7 \times \begin{array}{c} \searrow \\ \square \end{array} \\
 + & 0.8 \times \begin{array}{c} \leftarrow \\ \square \end{array} \\
 + \dots & \vdots
 \end{aligned}$$



$$\rightarrow \underline{0.3 \times \text{angle}} - 0.8 \times \text{speed}$$

$$\rightarrow 2 \times \text{angle} + 0.9 \times \text{speed}.$$

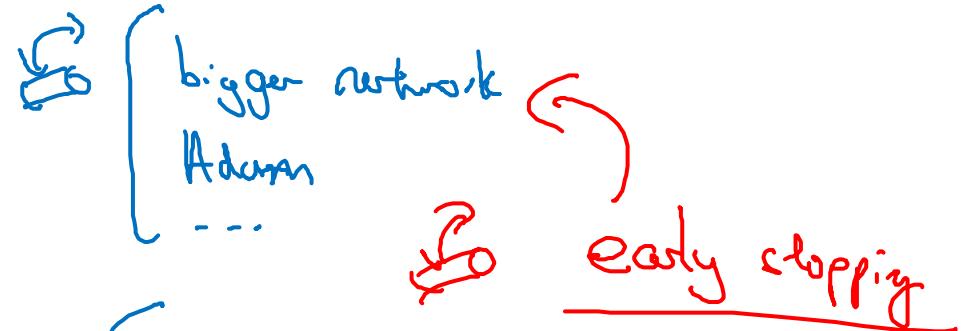


# Chain of assumptions in ML

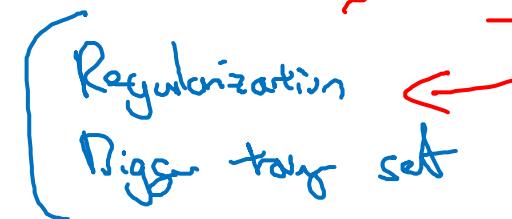
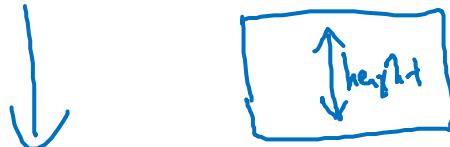
→ Fit training set well on cost function



( $\approx$  human-level performance)



→ Fit dev set well on cost function



→ Fit test set well on cost function



Bigger dev set

→ Performs well in real world

(Happy cat pic off users.)

Change dev set or  
cost function

## Orthogonalization

Orthogonalization or orthogonality is a system design property that assures that modifying an instruction or a component of an algorithm will not create or propagate side effects to other components of the system. It becomes easier to verify the algorithms independently from one another, it reduces testing and development time.

When a supervised learning system is design, these are the 4 assumptions that needs to be true and orthogonal.

1. Fit training set well in cost function
  - If it doesn't fit well, the use of a bigger neural network or switching to a better optimization algorithm might help.
2. Fit development set well on cost function
  - If it doesn't fit well, regularization or using bigger training set might help.
3. Fit test set well on cost function
  - If it doesn't fit well, the use of a bigger development set might help
4. Performs well in real world
  - If it doesn't perform well, the development test set is not set correctly or the cost function is not evaluating the right thing.



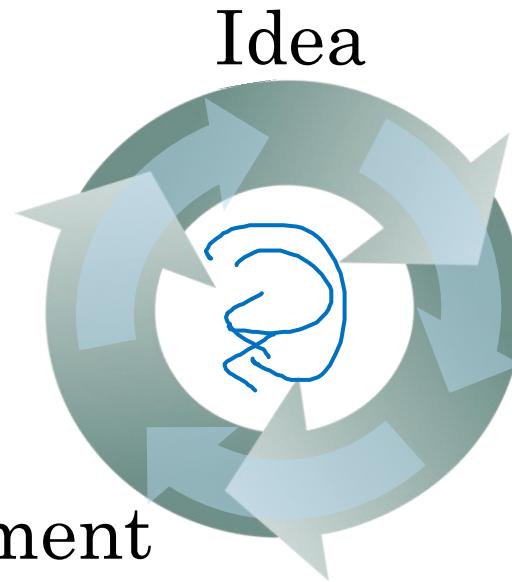
deeplearning.ai

Setting up  
your goal

---

Single number  
evaluation metric

# Using a single number evaluation metric



Code

- Of examples recognized as cont, what % actually are cont?
- what % of actual cont are correctly recognized

Classifier	Precision	Recall
A	95%	90%
B	98%	85%

$F_1$  Score = "Average" of P and R.

$$\left( \underbrace{\frac{2}{\frac{1}{P} + \frac{1}{R}}}_{\text{Harmonic mean}} \cdot \right)$$

Dev set + Single number evaluation metric  
real      Speel up iterating

# Another example

Algorithm	US	China	India	Other
A	<u>3%</u>	7%	5%	9%
B	5%	6%	5%	10%
C	2%	3%	4%	5%
D	5%	8%	7%	2%
E	4%	5%	2%	4%
F	7%	11%	8%	12%

## Single number evaluation metric

To choose a classifier, a well-defined development set and an evaluation metric speed up the iteration process.

Example : Cat vs Non- cat

$y = 1$ , cat image detected

Predict class $\hat{y}$	Actual class $y$	
	1	0
1	True positive	False positive
0	False negative	True negative

### Precision

Of all the images we predicted  $y=1$ , what fraction of it have cats?

$$\text{Precision (\%)} = \frac{\text{True positive}}{\text{Number of predicted positive}} \times 100 = \frac{\text{True positive}}{(\text{True positive} + \text{False positive})} \times 100$$

### Recall

Of all the images that actually have cats, what fraction of it did we correctly identifying have cats?

$$\text{Recall (\%)} = \frac{\text{True positive}}{\text{Number of predicted actually positive}} \times 100 = \frac{\text{True positive}}{(\text{True positive} + \text{True negative})} \times 100$$

Let's compare 2 classifiers A and B used to evaluate if there are cat images:

Classifier	Precision (p)	Recall (r)
A	95%	90%
B	98%	85%

In this case the evaluation metrics are precision and recall.

For classifier A, there is a 95% chance that there is a cat in the image and a 90% chance that it has correctly detected a cat. Whereas for classifier B there is a 98% chance that there is a cat in the image and a 85% chance that it has correctly detected a cat.

The problem with using precision/recall as the evaluation metric is that you are not sure which one is better since in this case, both of them have a good precision et recall. F1-score, a harmonic mean, combine both precision and recall.

$$\text{F1-Score} = \frac{2}{\frac{1}{p} + \frac{1}{r}}$$

Classifier	Precision (p)	Recall (r)	F1-Score
A	95%	90%	92.4 %
B	98%	85%	91.0%

Classifier A is a better choice. F1-Score is not the only evaluation metric that can be use, the average, for example, could also be an indicator of which classifier to use.



deeplearning.ai

Setting up  
your goal

---

Satisficing and  
optimizing metrics

# Another cat classification example

Classifier	Accuracy	Running time
A	90%	80ms
B	92%	95ms
C	95%	1,500ms

optimizing



satisficing



$$\text{Cost} = \underline{\text{accuracy}} - 0.5 \times \underline{\text{running Time}}$$

Maximize accuracy

Subject to running Time  $\leq \underline{100 \text{ ms.}}$

N metrics : 1 optimizing

N-1 satisficing

Wakewords / trigger words

Alexa, OK Google,

Hey Siri, nihao baidu

你好 百度

accuracy.

#false positive

Maximize accuracy.

s.t.  $\leq 1$  false positive  
every 24 hours.

## Satisficing and optimizing metric

There are different metrics to evaluate the performance of a classifier, they are called evaluation matrices. They can be categorized as satisficing and optimizing matrices. It is important to note that these evaluation matrices must be evaluated on a training set, a development set or on the test set.

Example: Cat vs Non-cat

Classifier	Accuracy	Running time
A	90%	80 ms
B	92%	95 ms
C	95%	1 500 ms

In this case, accuracy and running time are the evaluation matrices. Accuracy is the optimizing metric, because you want the classifier to correctly detect a cat image as accurately as possible. The running time which is set to be under 100 ms in this example, is the satisficing metric which mean that the metric has to meet expectation set.

The general rule is:

$$N_{metric} : \begin{cases} 1 & \text{Optimizing metric} \\ N_{metric} - 1 & \text{Satisficing metric} \end{cases}$$



deeplearning.ai

Setting up  
your goal

---

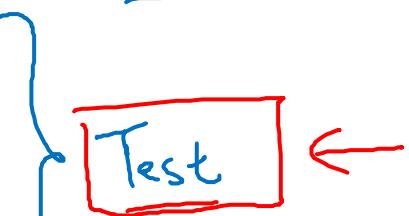
Train/dev/test  
distributions

# Cat classification dev/test sets

↳ development set, hold out cross validation set

Regions:

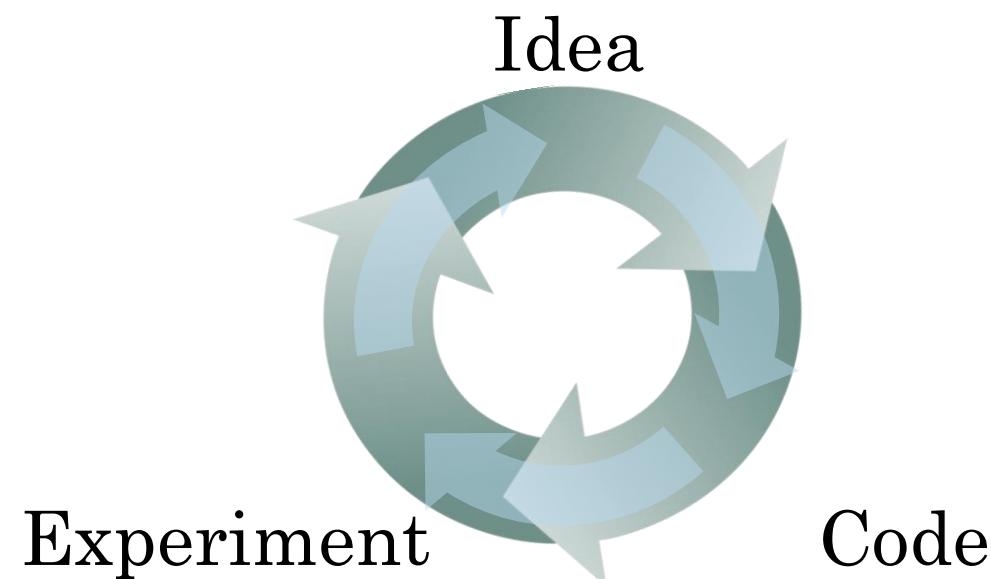
- US
- UK
- Other Europe
- South America
- India
- China
- Other Asia
- Australia



Randomly shuffle into dev/test



dev set  
+  
Metric



# True story (details changed)

[ Optimizing on dev set on loan approvals for  
medium income zip codes

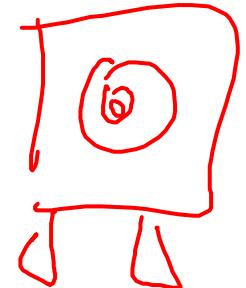


$x \rightarrow y$  (repay loan?)



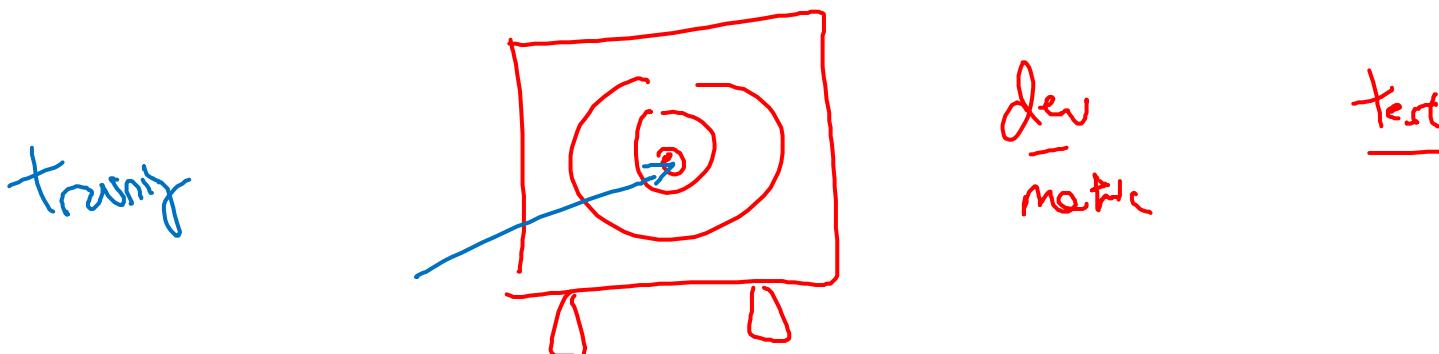
[ Tested on low income zip codes

$\sim 3$  month



# Guideline

Choose a dev set and test set to reflect data you expect to get in the future and consider important to do well on.



## Training, development and test distributions

Setting up the training, development and test sets have a huge impact on productivity. It is important to choose the development and test sets from the same distribution and it must be taken randomly from all the data.

### Guideline

Choose a development set and test set to reflect data you expect to get in the future and consider important to do well.



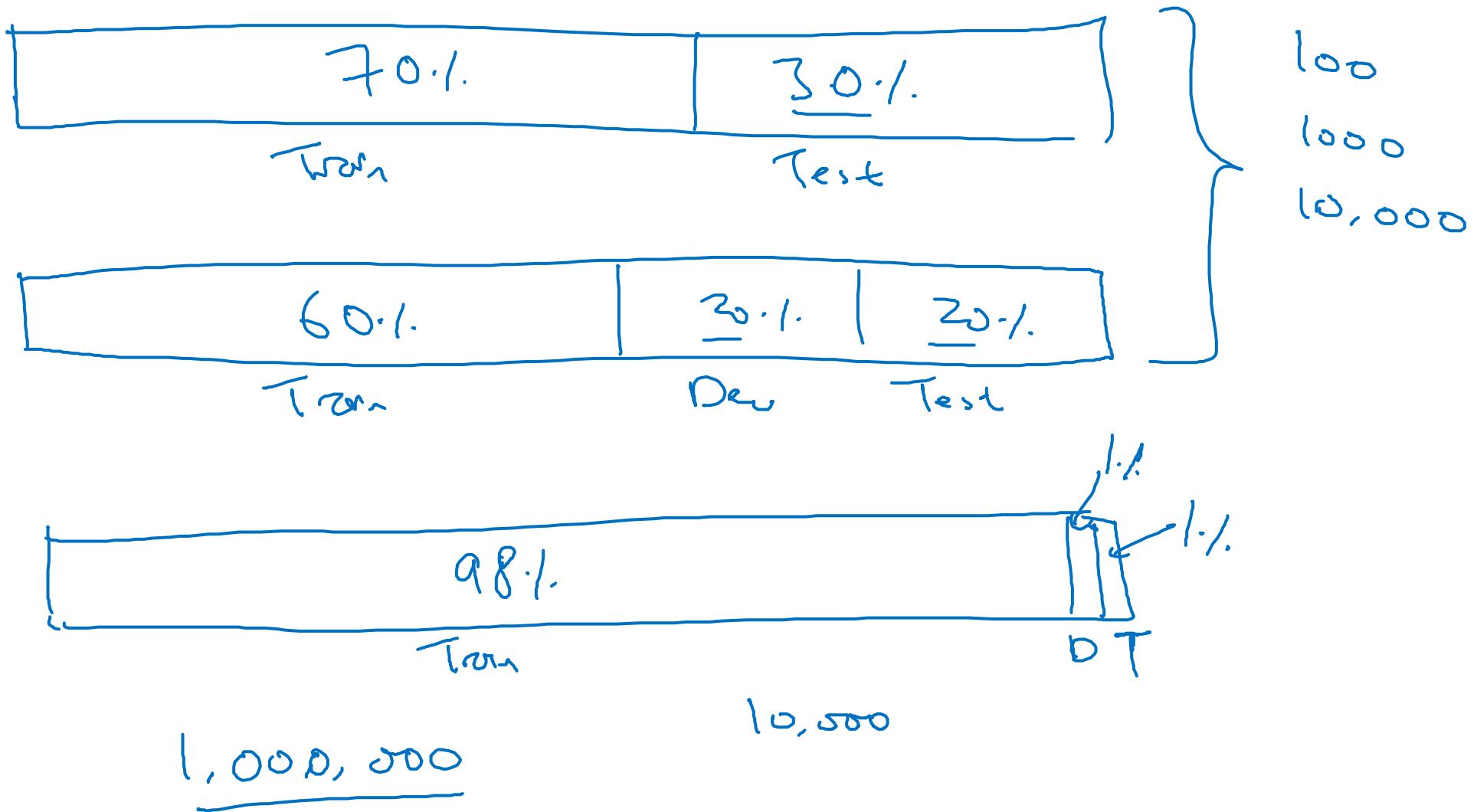
deeplearning.ai

Setting up  
your goal

---

Size of dev  
and test sets

# Old way of splitting data



# Size of dev set

A    B

Set your dev set to be big enough to detect differences in  
algorithm/models you're trying out.

100: small  
10%

A                          B  
97% → 97.1%  
0.1%  
10%

1,000

10,000

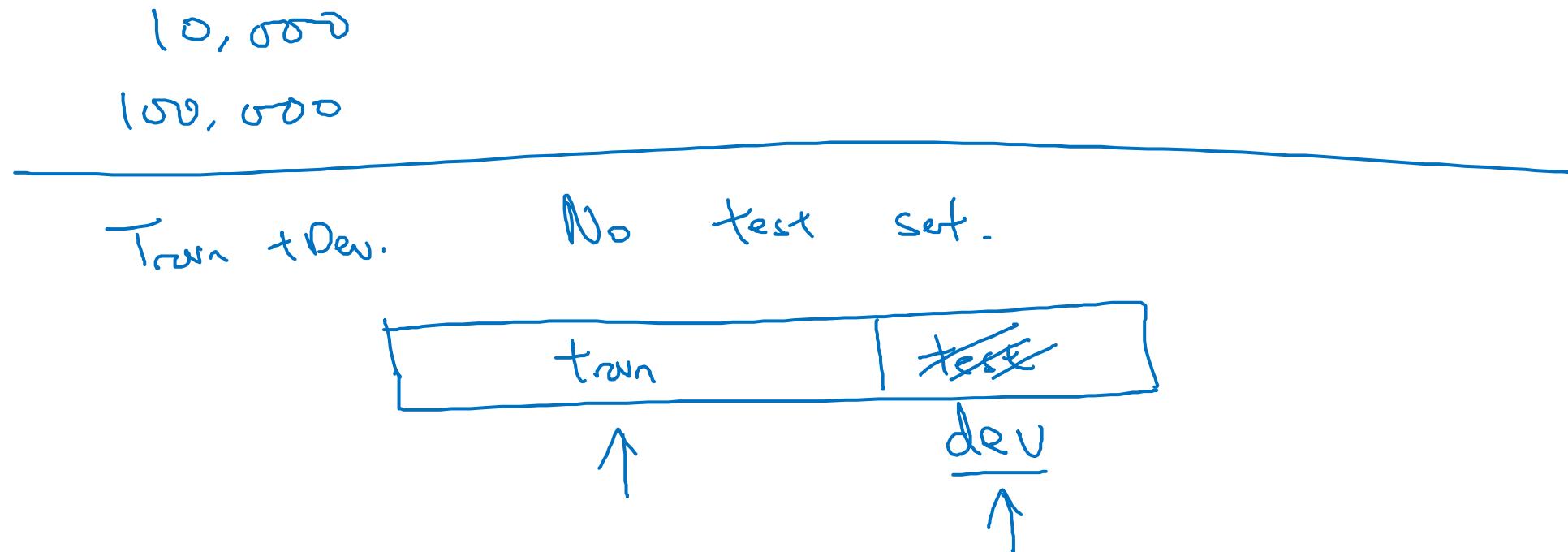
100,000

0.01%  
0.001%

Online advertising

# Size of test set

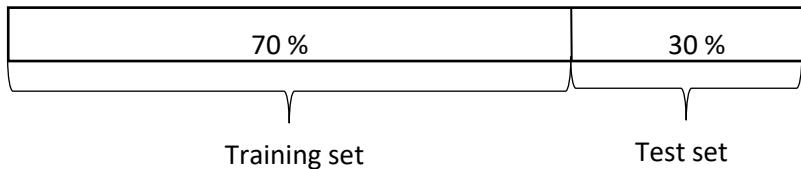
→ Set your test set to be big enough to give high confidence in the overall performance of your system.



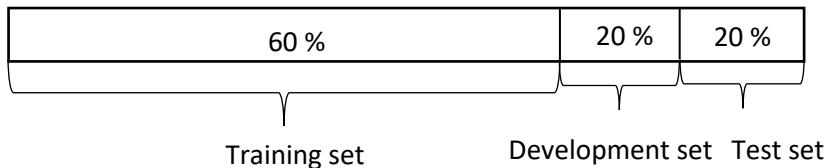
## Size of the development and test sets

Old way of splitting data

We had smaller data set therefore we had to use a greater percentage of data to develop and test ideas and models.

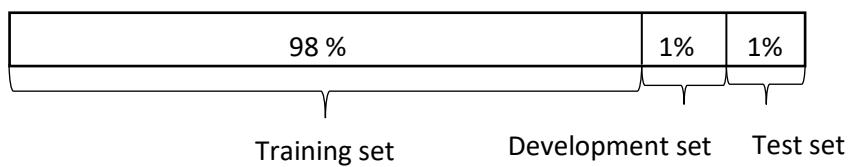


Or



Modern era – Big data

Now, because a large amount of data is available, we don't have to compromise as much and can use a greater portion to train the model.



Guidelines

- Set up the size of the test set to give a high confidence in the overall performance of the system.
- Test set helps evaluate the performance of the final classifier which could be less 30% of the whole data set.
- The development set has to be big enough to evaluate different ideas.



deeplearning.ai

Setting up  
your goal

---

When to change  
dev/test sets and  
metrics

# Cat dataset examples

Metric + Dev : Prefer A  
You/users : Prefer B.

→ Metric: classification error

Algorithm A: 3% error → Pornographic

✓ Algorithm B: 5% error

$$\left\{ \begin{array}{l} \text{Error: } \frac{1}{\sum_i w^{(i)}} - \cancel{\frac{m_{\text{dev}}}{m_{\text{dev}}}} \\ \sum_{i=1}^{m_{\text{dev}}} \underline{w^{(i)}} \downarrow \{ y_{\text{pred}}^{(i)} + y^{(i)} \} \\ \text{predval value (0/1)} \end{array} \right.$$

$\rightarrow w^{(i)} = \begin{cases} 1 & \text{if } x^{(i)} \text{ is non-porn} \\ 10 & \text{if } x^{(i)} \text{ is porn} \end{cases}$

# Orthogonalization for cat pictures: anti-porn

- 1. So far we've only discussed how to define a metric to evaluate classifiers. ← Place target 
- 2. Worry separately about how to do well on this metric. 

An (shoot at target)

$$\rightarrow J = \frac{1}{\sum w^{(i)}} \sum_{i=1}^m w^{(i)} \ell(\hat{y}^{(i)}, y^{(i)})$$



# Another example

Algorithm A: 3% error

✓ Algorithm B: 5% error ←

→ Dev/test



→ User images



If doing well on your metric + dev/test set does not correspond to doing well on your application, change your metric and/or dev/test set.

## When to change development/test sets and metrics

Example: Cat vs Non-cat

A cat classifier tries to find a great amount of cat images to show to cat loving users. The evaluation metric used is a classification error.

Algorithm	Classification error [%]
A	3%
B	5%

It seems that Algorithm A is better than Algorithm B since there is only a 3% error, however for some reason, Algorithm A is letting through a lot of the pornographic images.

Algorithm B has 5% error thus it classifies fewer images but it doesn't have pornographic images. From a company's point of view, as well as from a user acceptance point of view, Algorithm B is actually a better algorithm. The evaluation metric fails to correctly rank order preferences between algorithms. The evaluation metric or the development set or test set should be changed.

The misclassification error metric can be written as a function as follow:

$$\text{Error} : \frac{1}{m_{dev}} \sum_{i=1}^{m_{dev}} \mathcal{L}\{\hat{y}^{(i)} \neq y^{(i)}\}$$

This function counts up the number of misclassified examples.

The problem with this evaluation metric is that it treats pornographic vs non-pornographic images equally. One way to change this evaluation metric is to add the weight term  $w^{(i)}$ .

$$w^{(i)} = \begin{cases} 1 & \text{if } x^{(i)} \text{ is non-pornographic} \\ 10 & \text{if } x^{(i)} \text{ is pornographic} \end{cases}$$

The function becomes:

$$\text{Error} : \frac{1}{\sum w^{(i)}} \sum_{i=1}^{m_{dev}} w^{(i)} \mathcal{L}\{\hat{y}^{(i)} \neq y^{(i)}\}$$

Guideline

1. Define correctly an evaluation metric that helps better rank order classifiers
2. Optimize the evaluation metric



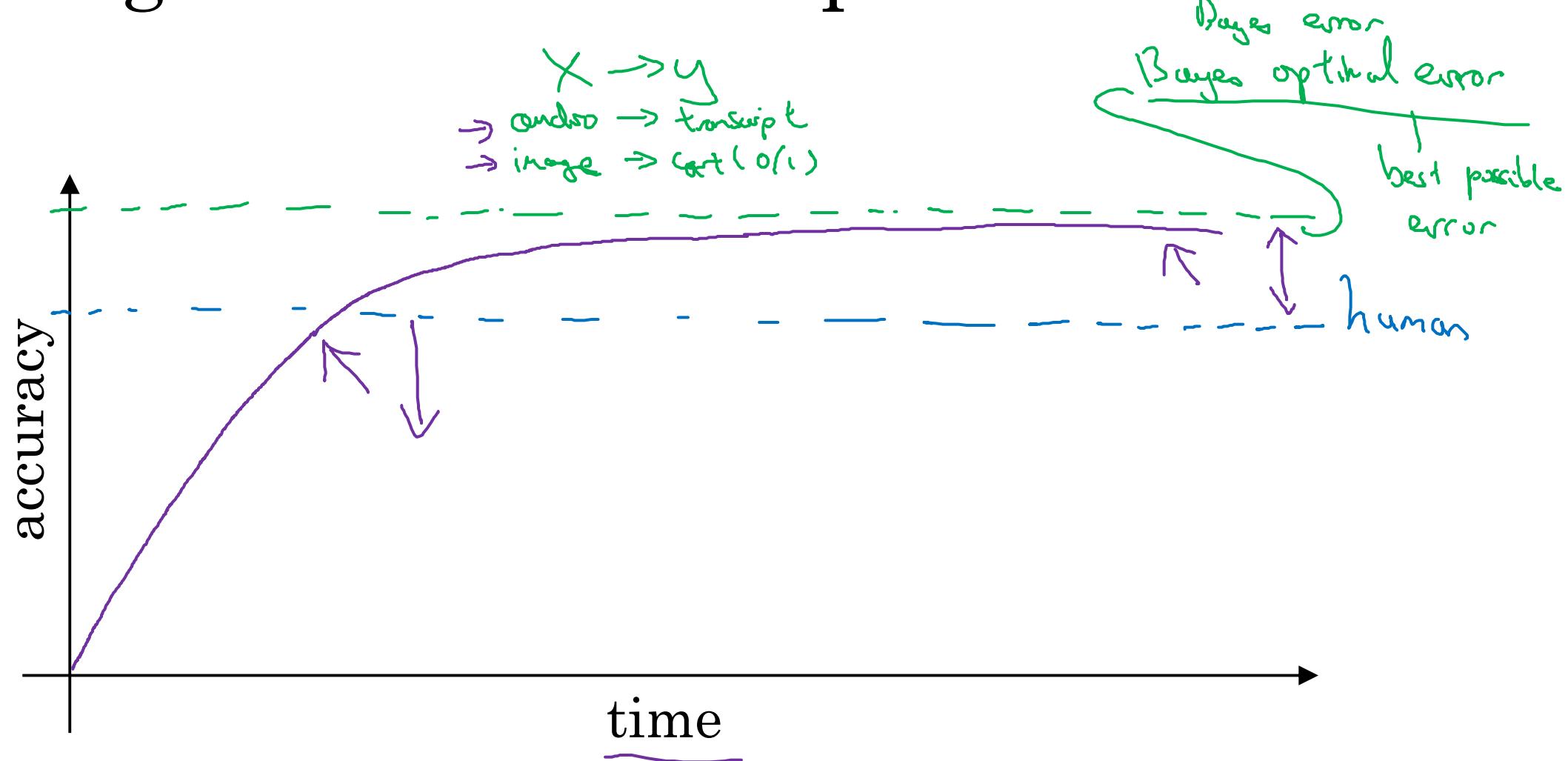
deeplearning.ai

Comparing to human-level performance

---

Why human-level performance?

# Comparing to human-level performance



# Why compare to human-level performance

Humans are quite good at a lot of tasks. So long as ML is worse than humans, you can:

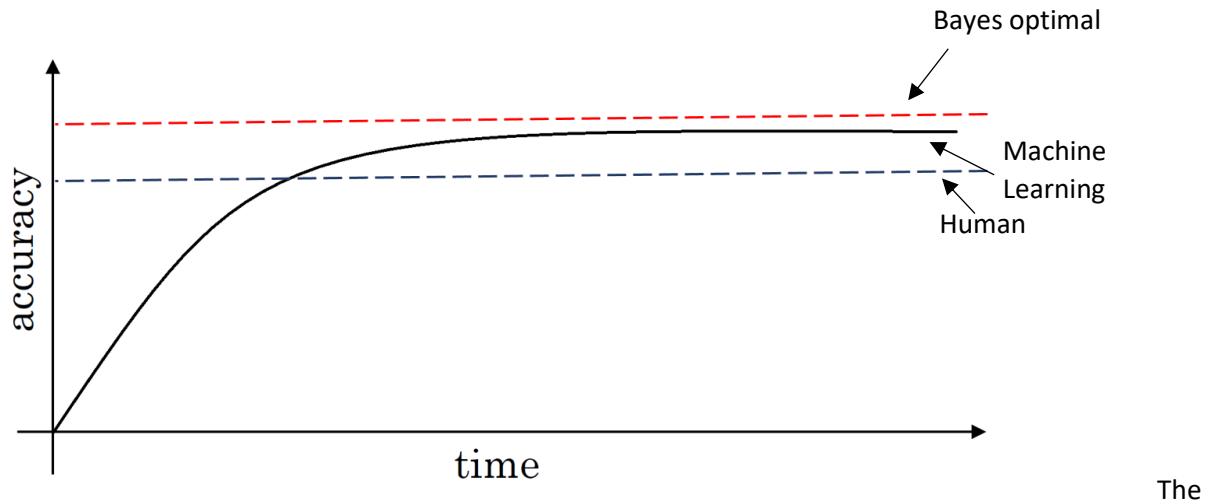
- - Get labeled data from humans.  $(x, y)$
- - Gain insight from manual error analysis:  
Why did a person get this right?
- - Better analysis of bias/variance.

## Why human-level performance?

Today, machine learning algorithms can compete with human-level performance since they are more productive and more feasible in a lot of application. Also, the workflow of designing and building a machine learning system, is much more efficient than before.

Moreover, some of the tasks that humans do are close to "perfection", which is why machine learning tries to mimic human-level performance.

The graph below shows the performance of humans and machine learning over time.



Machine learning progresses slowly when it surpasses human-level performance. One of the reason is that human-level performance can be close to Bayes optimal error, especially for natural perception problem.

Bayes optimal error is defined as the best possible error. In other words, it means that any functions mapping from  $x$  to  $y$  can't surpass a certain level of accuracy.

Also, when the performance of machine learning is worse than the performance of humans, you can improve it with different tools. They are harder to use once its surpasses human-level performance.

These tools are:

- Get labeled data from humans
- Gain insight from manual error analysis: Why did a person get this right?
- Better analysis of bias/variance.



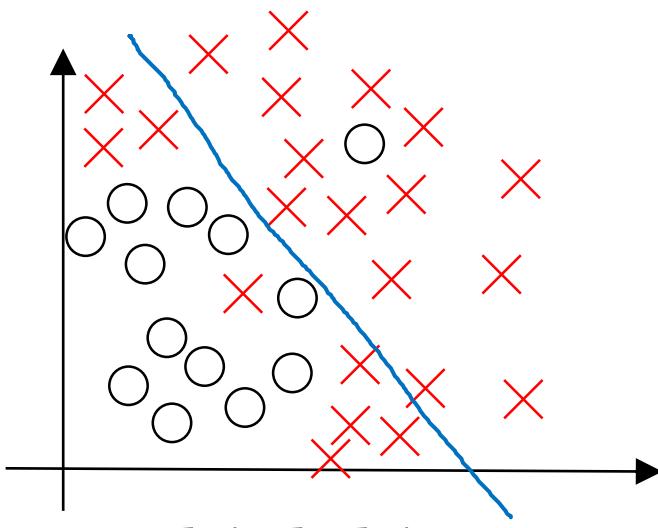
deeplearning.ai

Comparing to human-level performance

---

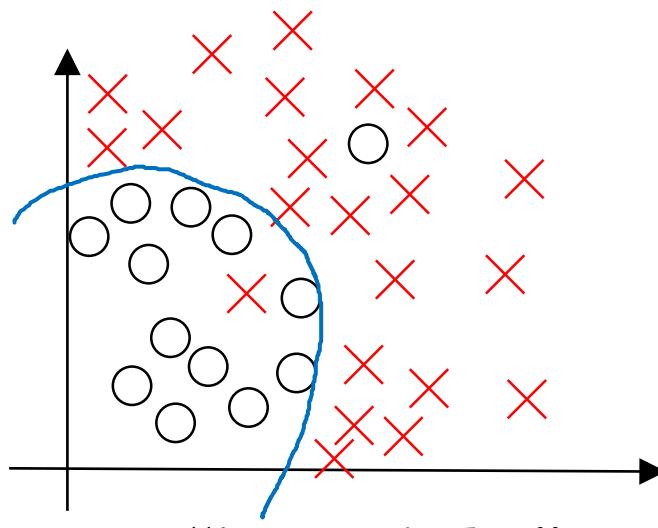
Avoidable bias

# Bias and Variance

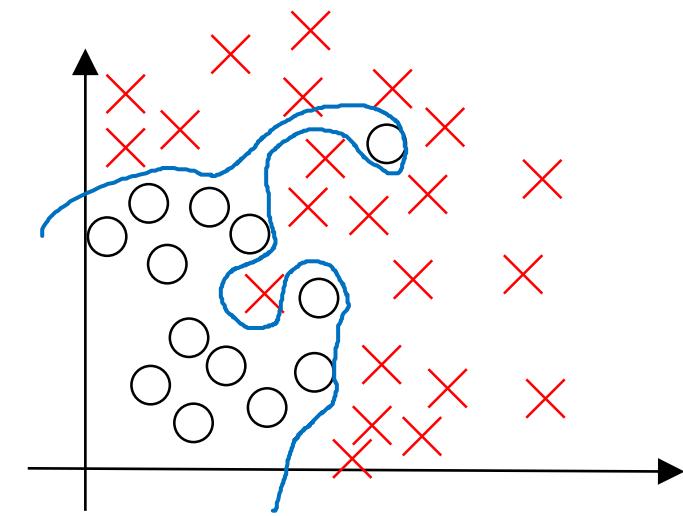


high bias

*Underfitting*



“just right”



high variance

*Overfitting*

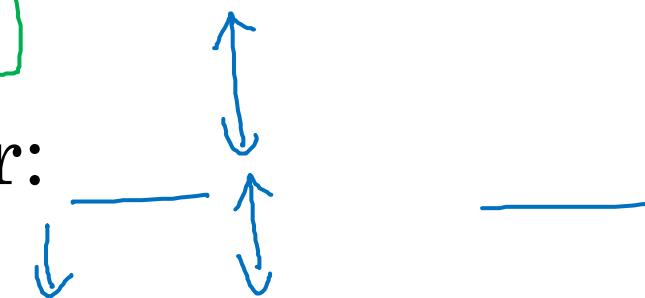
# Bias and Variance

Cat classification

Human-level  $\approx 0\%$

Training set error:

Dev set error:



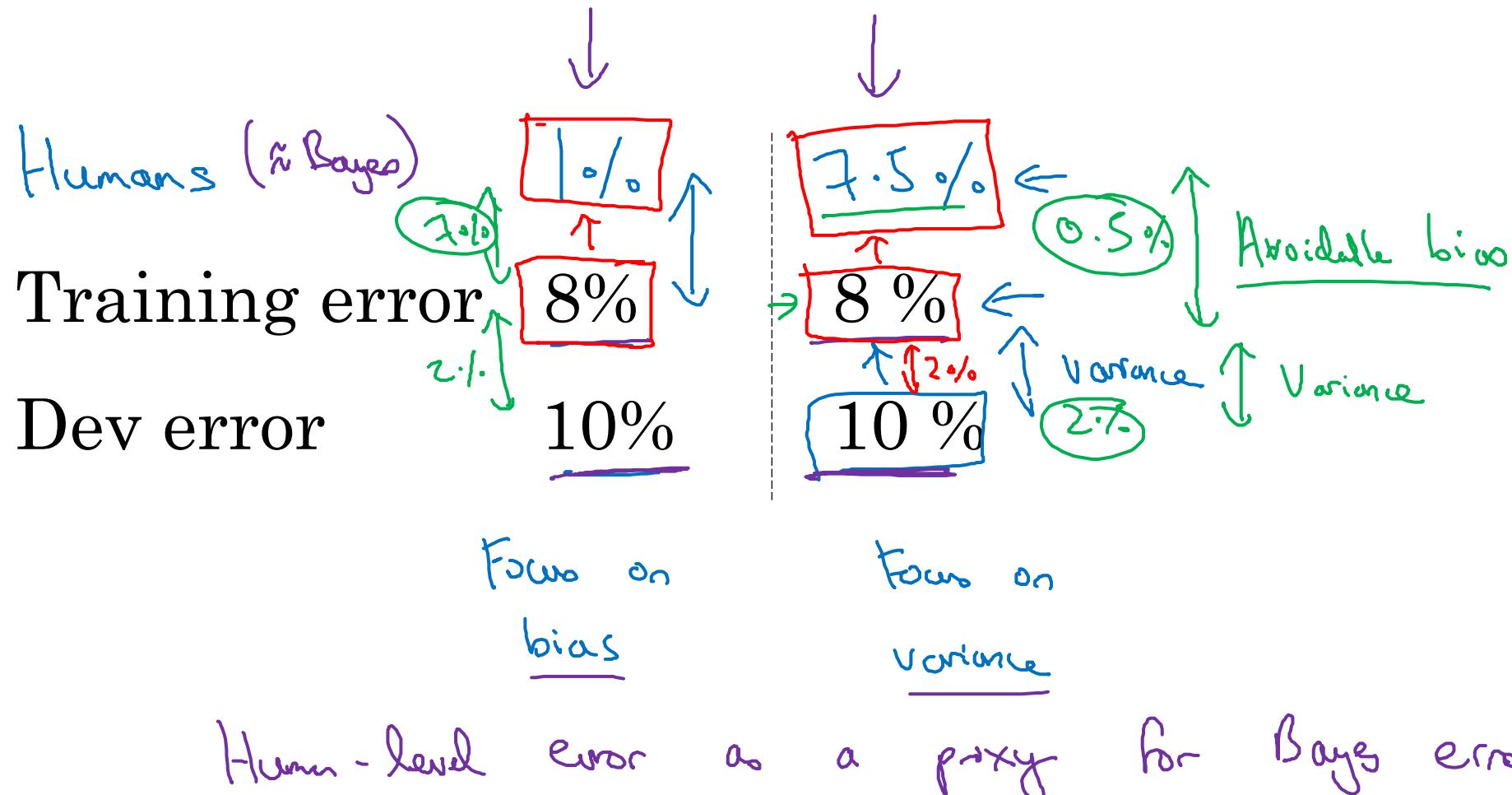
high variance

high bias

high bias  
high variance

low bias  
low variance

# Cat classification example



## Avoidable bias

By knowing what the human-level performance is, it is possible to tell when a training set is performing well or not.

Example: Cat vs Non-Cat

	Classification error (%)	
	Scenario A	Scenario B
Humans	1	7.5
Training error	8	8
Development error	10	10

In this case, the human level error as a proxy for Bayes error since humans are good to identify images. If you want to improve the performance of the training set but you can't do better than the Bayes error otherwise the training set is overfitting. By knowing the Bayes error, it is easier to focus on whether bias or variance avoidance tactics will improve the performance of the model.

### Scenario A

There is a 7% gap between the performance of the training set and the human level error. It means that the algorithm isn't fitting well with the training set since the target is around 1%. To resolve the issue, we use bias reduction technique such as training a bigger neural network or running the training set longer.

### Scenario B

The training set is doing good since there is only a 0.5% difference with the human level error. The difference between the training set and the human level error is called avoidable bias. The focus here is to reduce the variance since the difference between the training error and the development error is 2%. To resolve the issue, we use variance reduction technique such as regularization or have a bigger training set.



deeplearning.ai

Comparing to human-level performance

---

Understanding  
human-level  
performance

# Human-level error as a proxy for Bayes error

Medical image classification example:

Suppose:

- (a) Typical human ..... 3 % error
- (b) Typical doctor ..... 1 % error
- (c) Experienced doctor ..... 0.7 % error
- (d) Team of experienced doctors .. 0.5 % error



What is “human-level” error?

$$\text{Baye error} \leq \underline{0.5\%}$$

# Error analysis example

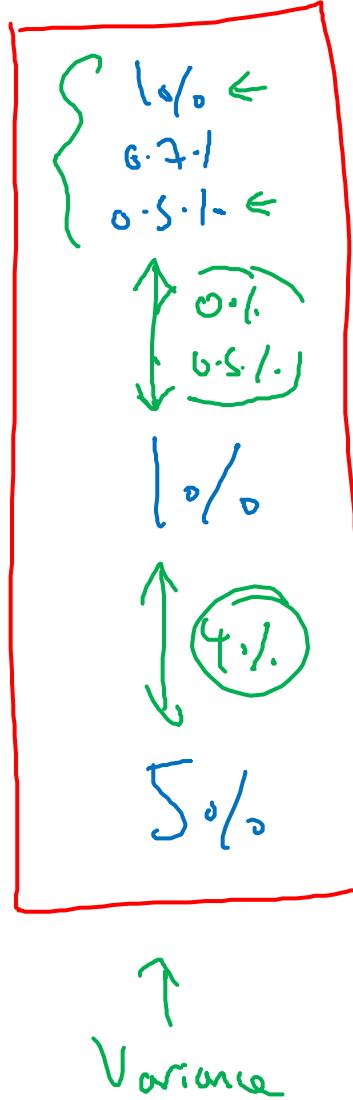
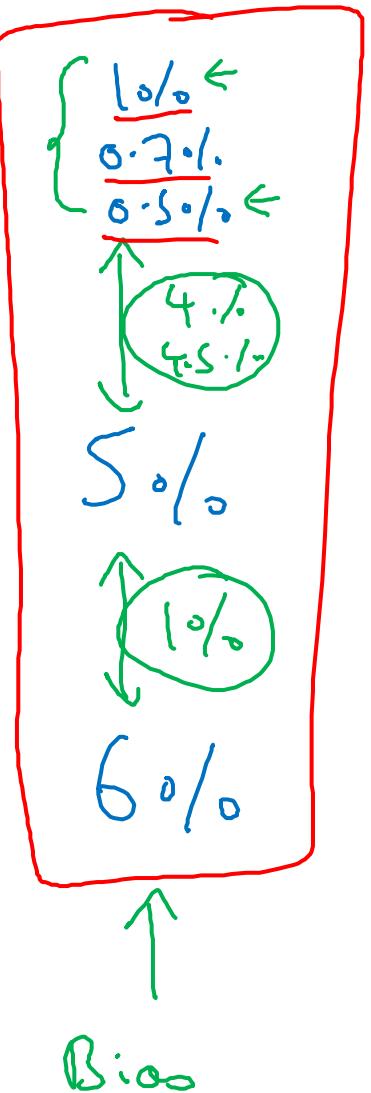
Human (proxy for Bayes error)



Training error

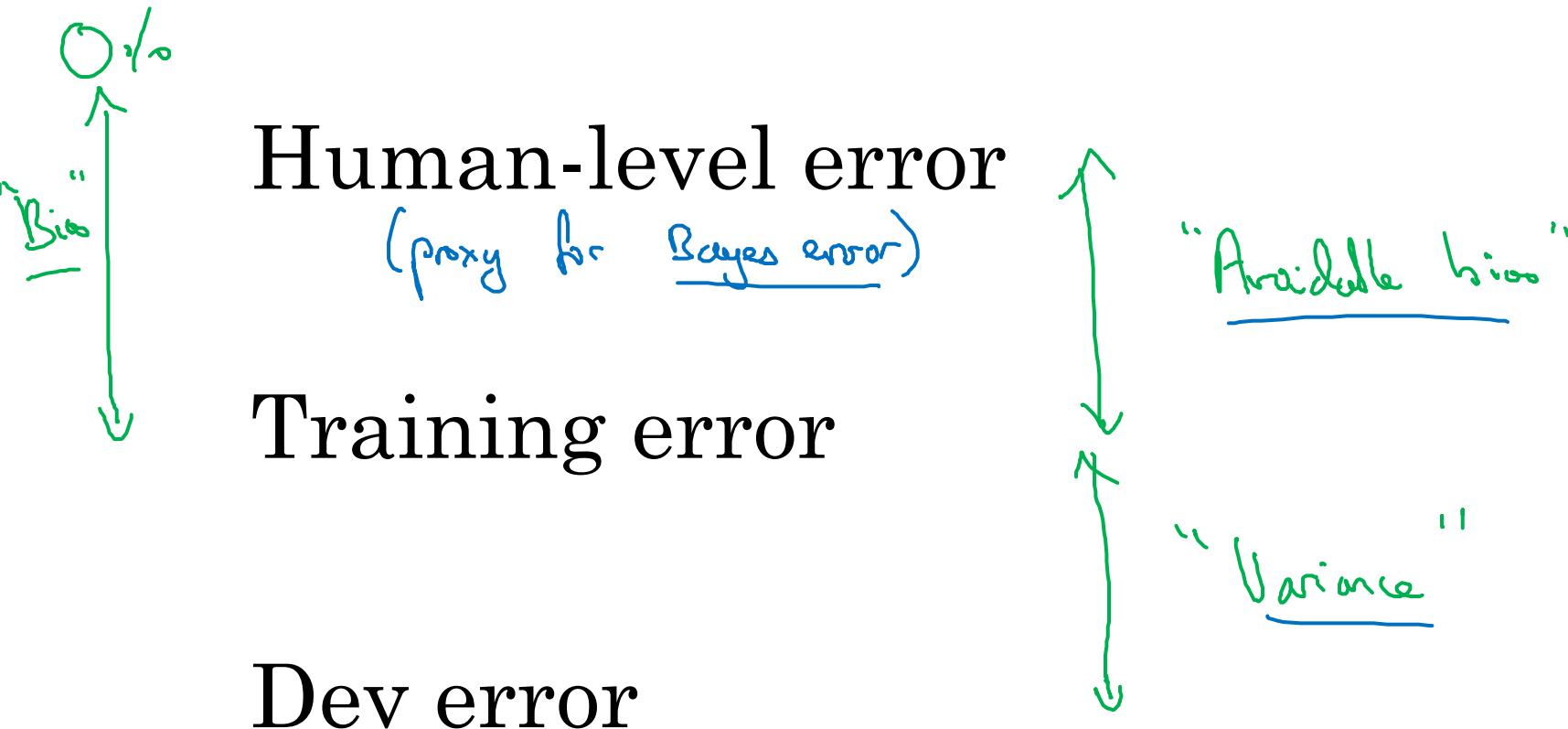


Dev error



$$\begin{aligned} & \rightarrow 0.7\% \\ & \rightarrow \frac{0.5\%}{0.2\%} \quad 1\% \leftarrow \\ & \qquad \qquad \qquad 0.0\% \\ & \rightarrow 0.7\% \quad \leftarrow \\ & \qquad \qquad \qquad 0.1\% \leftarrow \\ & \rightarrow 0.8\% \end{aligned}$$

# Summary of bias/variance with human-level performance



## Understanding human-level performance

Human-level error gives an estimate of Bayes error.

### Example 1: Medical image classification

This is an example of a medical image classification in which the input is a radiology image and the output is a diagnosis classification decision.

	Classification error (%)
Typical human	3.0
Typical doctor	1.0
Experienced doctor	0.7
Team of experienced doctors	0.5

The definition of human-level error depends on the purpose of the analysis, in this case, by definition the Bayes error is lower or equal to 0.5%.

### Example 2: Error analysis

	Classification error (%)		
	Scenario A	Scenario B	Scenario C
Human (proxy for Bayes error)	1	1	0.5
	0.7	0.7	
	0.5	0.5	
Training error	5	1	0.7
Development error	6	5	0.8

#### Scenario A

In this case, the choice of human-level performance doesn't have an impact. The avoidable bias is between 4%-4.5% and the variance is 1%. Therefore, the focus should be on bias reduction technique.

#### Scenario B

In this case, the choice of human-level performance doesn't have an impact. The avoidable bias is between 0%-0.5% and the variance is 4%. Therefore, the focus should be on variance reduction technique.

#### Scenario C

In this case, the estimate for Bayes error has to be 0.5% since you can't go lower than the human-level performance otherwise the training set is overfitting. Also, the avoidable bias is 0.2% and the variance is 0.1%. Therefore, the focus should be on bias reduction technique.

#### Summary of bias/variance with human-level performance

- Human - level error – proxy for Bayes error
- If the difference between human-level error and the training error is bigger than the difference between the training error and the development error. The focus should be on bias reduction technique
- If the difference between training error and the development error is bigger than the difference between the human-level error and the training error. The focus should be on variance reduction technique



deeplearning.ai

Comparing to human-level performance

---

Surpassing human-level performance

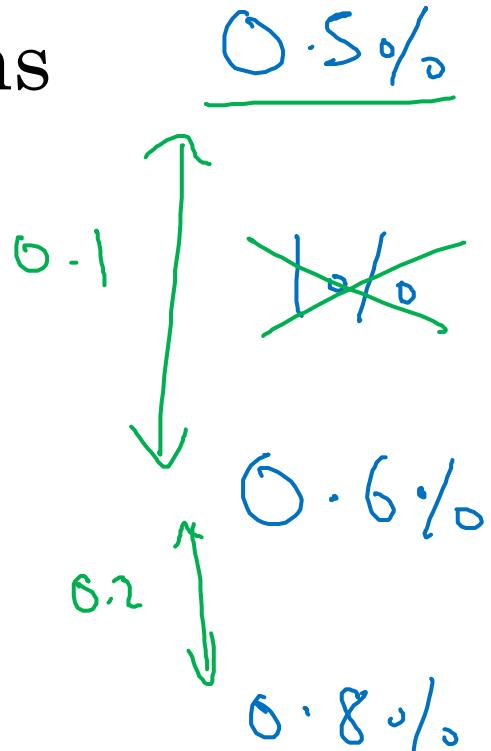
# Surpassing human-level performance

Team of humans

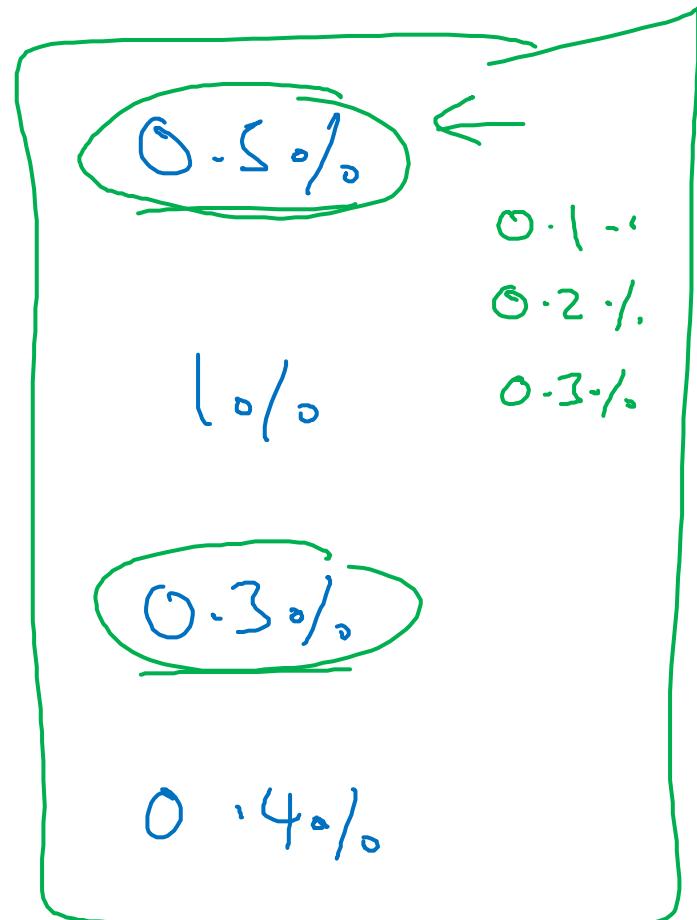
One human

Training error

Dev error

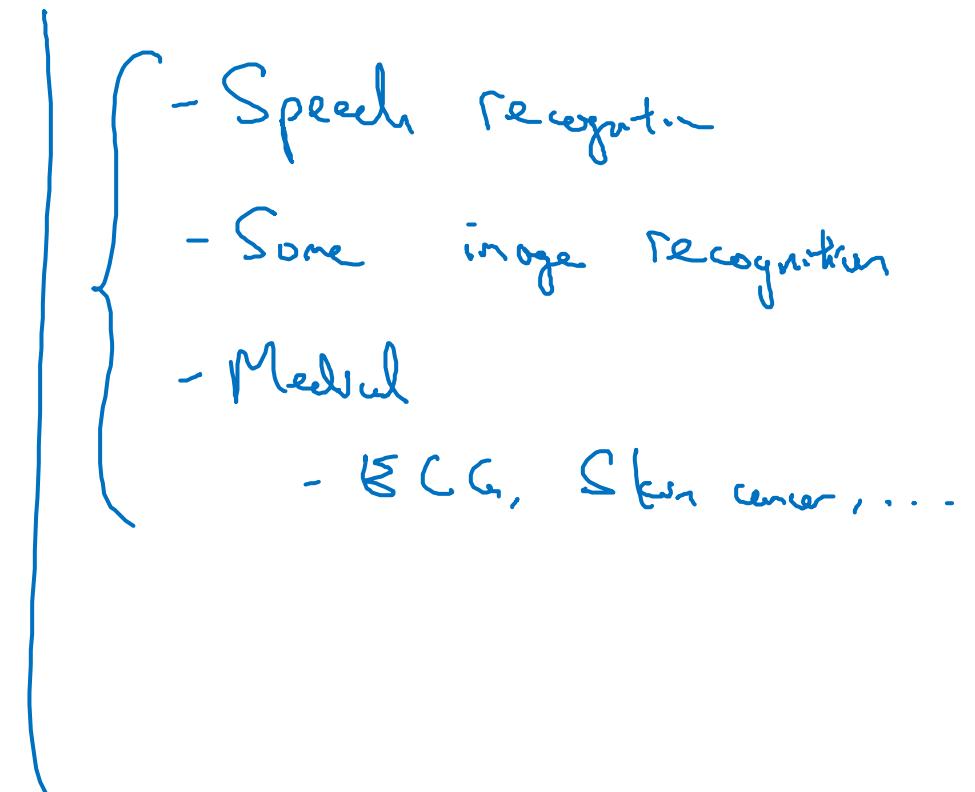


What is avoidable bias?



# Problems where ML significantly surpasses human-level performance

- - Online advertising
- - Product recommendations
- - Logistics (predicting transit time)
- - Loan approvals



Structural data

Not natural perception

Lots of data

## Surpassing human-level performance

Example1: Classification task

	Classification error (%)	
	Scenario A	Scenario B
Team of humans	0.5	0.5
One human	1.0	1
Training error	0.6	0.3
Development error	0.8	0.4

Scenario A

In this case, the Bayes error is 0.5%, therefore the available bias is 0.1% et the variance is 0.2%.

Scenario B

In this case, there is not enough information to know if bias reduction or variance reduction has to be done on the algorithm. It doesn't mean that the model cannot be improved, it means that the conventional ways to know if bias reduction or variance reduction are not working in this case.

There are many problems where machine learning significantly surpasses human-level performance, especially with structured data:

- Online advertising
- Product recommendations
- Logistics (predicting transit time)
- Loan approvals



deeplearning.ai

Comparing to human-level performance

---

Improving your model performance

# The two fundamental assumptions of supervised learning

1. You can fit the training set pretty well.



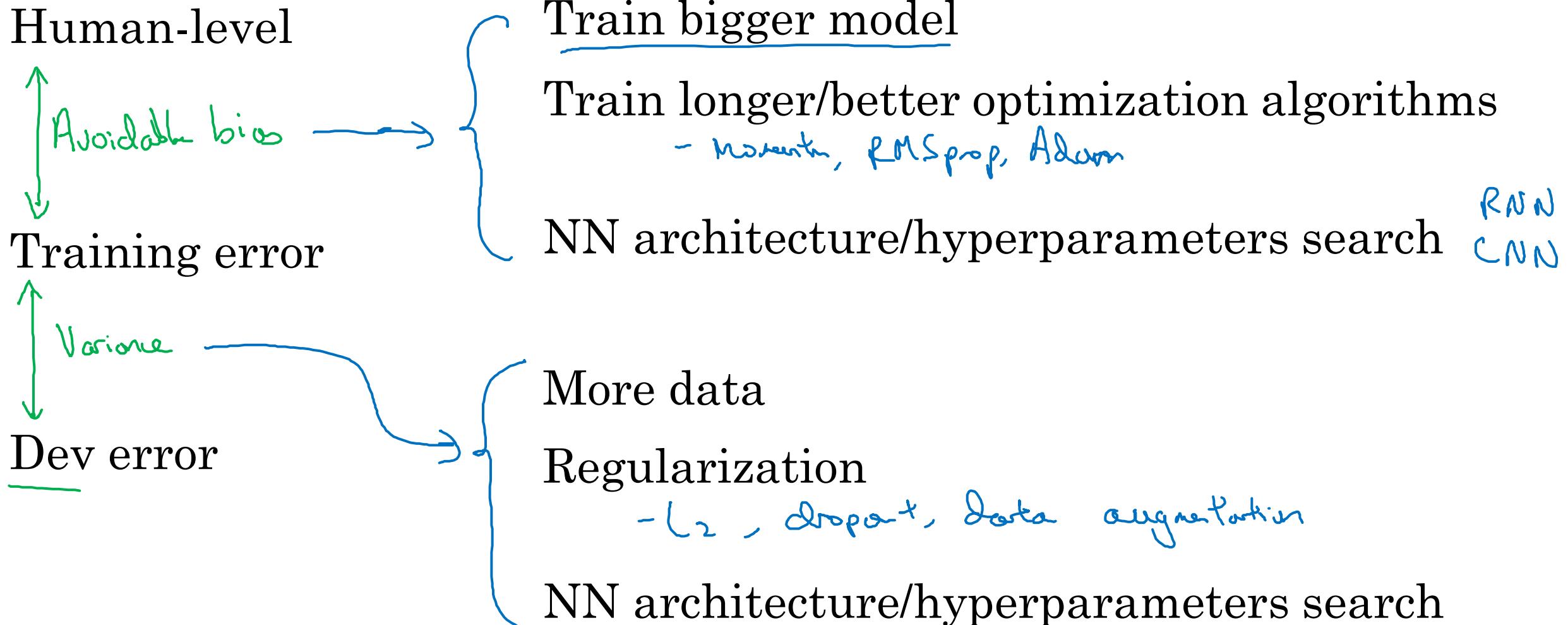
$\sim$  Avoidable bias

2. The training set performance generalizes pretty well to the dev/test set.



$\sim$  Variance

# Reducing (avoidable) bias and variance



# Improving your model performance

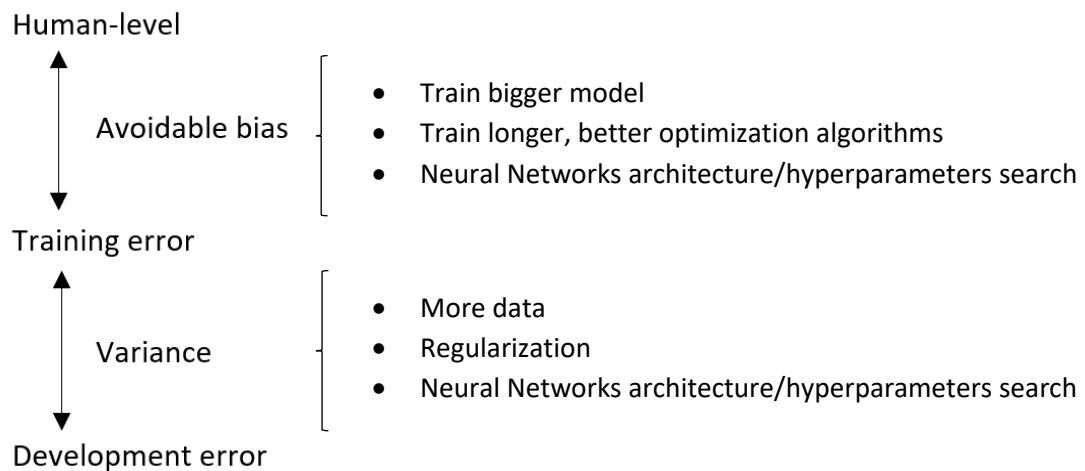
## The two fundamental assumptions of supervised learning

There are 2 fundamental assumptions of supervised learning. The first one is to have a low avoidable bias which means that the training set fits well. The second one is to have a low or acceptable variance which means that the training set performance generalizes well to the development set and test set.

If the difference between human-level error and the training error is bigger than the difference between the training error and the development error, the focus should be on bias reduction technique which are training a bigger model, training longer or change the neural networks architecture or try various hyperparameters search.

If the difference between training error and the development error is bigger than the difference between the human-level error and the training error, the focus should be on variance reduction technique which are bigger data set, regularization or change the neural networks architecture or try various hyperparameters search.

## Summary





deeplearning.ai

# Error Analysis

---

Carrying out error  
analysis

# Look at dev examples to evaluate ideas



90% accuracy  
→ 10% error

Should you try to make your cat classifier do better on dogs? ←

Error analysis: → 5-10 min

- Get ~100 mislabeled dev set examples.
- Count up how many are dogs.

"ceiling"

→ 5%  
10%  
↓  
5/100  
9.5%

→ 50%  
50/100  
100%  
↓  
5%

# Evaluate multiple ideas in parallel

Ideas for cat detection:

- Fix pictures of dogs being recognized as cats ←
- Fix great cats (lions, panthers, etc..) being misrecognized ←
- Improve performance on blurry images ← ↴

Image	Dog	Great Cats	Blurry	Instagram	Comments
1	✓			✓	Pitbull
2			✓	✓	
3		✓	✓		Rainy day at zoo
:	⋮	⋮	⋮	⋮	
% of total	<u>8%</u>	<u>43%~</u>	<u>61%~</u>	<u>12%</u>	



deeplearning.ai

# Error Analysis

---

Cleaning up  
Incorrectly labeled  
data

# Incorrectly labeled examples

x



y

1

0

1

1

0

*Training set.*

DL algorithms are quite robust to random errors in the training set.

*Systematic errors*

# Error analysis



Image	Dog	Great Cat	Blurry	Incorrectly labeled	Comments
...					
98				✓	Labeler missed cat in background
99		✓			
100				✓	Drawing of a cat; Not a real cat.
% of total	<u>8%</u>	<u>43%</u>	<u>61%</u>	<u>6%</u>	

Overall dev set error ..... 100%

Errors due incorrect labels ..... 0.6% ←

Errors due to other causes ..... 9.4% ←

2%

0.6%

1.4%

2.1%

1.9%

Goal of dev set is to help you select between two classifiers A & B.

# Correcting incorrect dev/test set examples

- Apply same process to your dev and test sets to make sure they continue to come from the same distribution
- Consider examining examples your algorithm got right as well as ones it got wrong.  
*(81%)*      *(20%)*
- Train and dev/test data may now come from slightly different distributions.



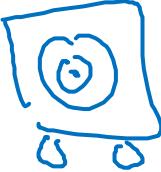
deeplearning.ai

## Error Analysis

---

Build your first system  
quickly, then iterate

# Speech recognition example



- • Noisy background
    - • Café noise
    - • Car noise
  - • Accent
  - • Far from
  - • Young
  - • Stutter
  - • ...
- Guideline:**  
Build your first system quickly, then iterate
- • Set up dev/test set and metric
  - Build initial system quickly
  - Use Bias/Variance analysis & Error analysis to prioritize next steps.



deeplearning.ai

Mismatched training  
and dev/test data

---

Training and testing  
on different  
distributions

# Cat app example

Data from webpages



care about this

Data from mobile app

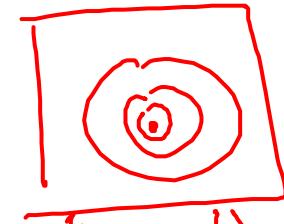
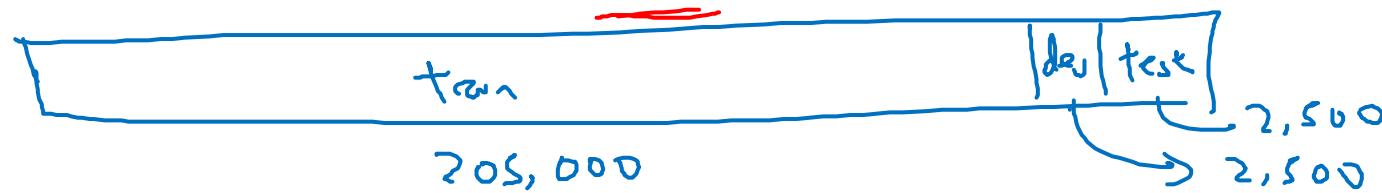


$\rightarrow \approx 200,000$

$210,000$   
shuffle

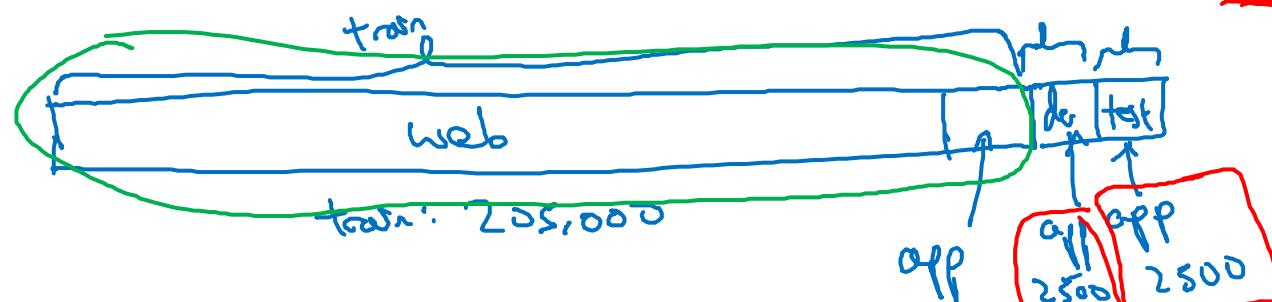
$\rightarrow \approx 10,000$

X Option 1:



$\frac{200K}{210K}$

Option 2:



2381 - web  
119 - mobile app

# Speech recognition example

Speech activated rearview mirror



## Training

Purchased data  $\downarrow \downarrow$   
 $x, y$

Smart speaker control

Voice keyboard

...

500,000 utterances

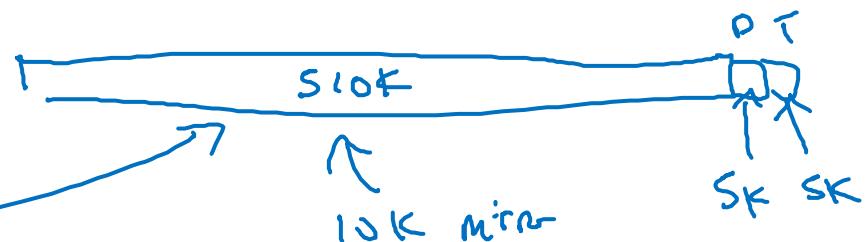
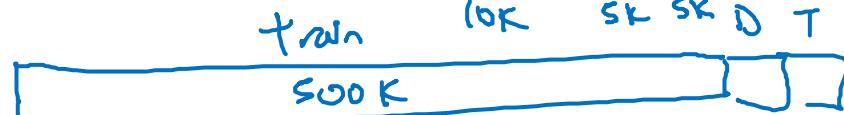
## Dev/test

Speech activated  
rearview mirror

$\rightarrow 20,000$

10K SK SK

train  
500 K





deeplearning.ai

Mismatched training  
and dev/test data

---

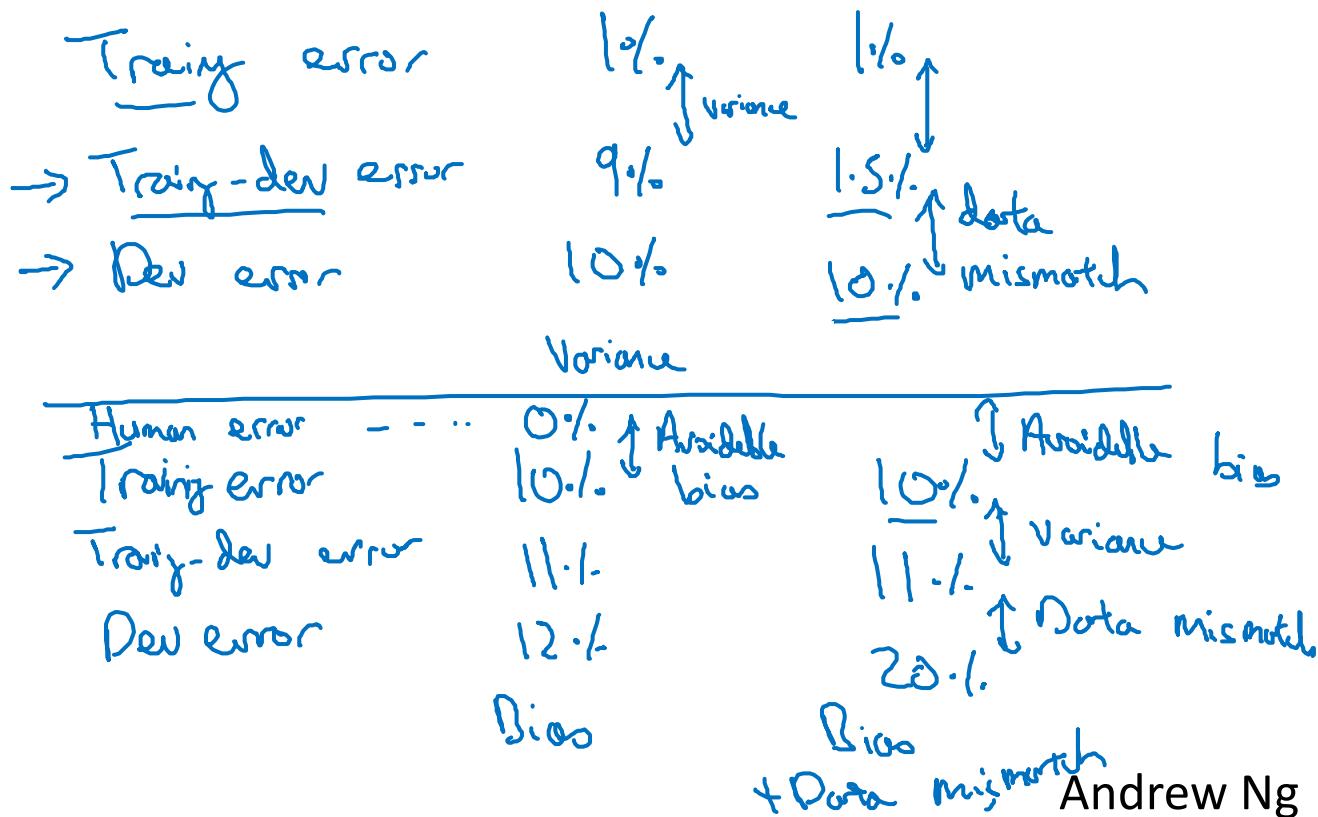
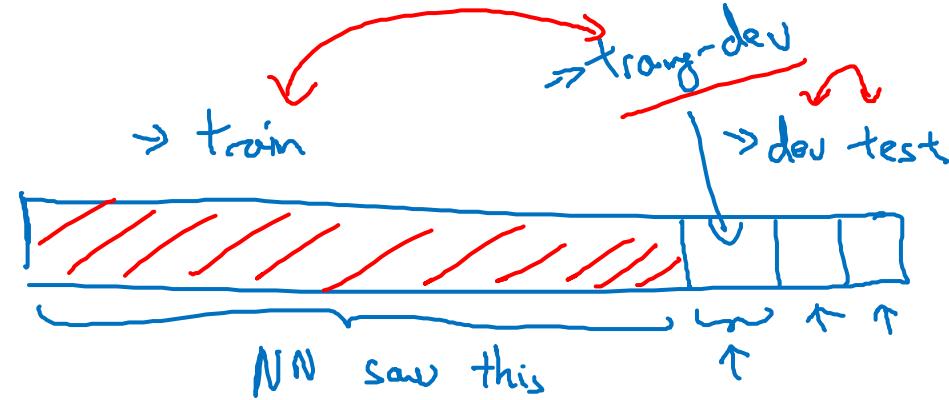
Bias and Variance with  
mismatched data  
distributions

# Cat classifier example

Assume humans get  $\approx 0\%$  error.

Training error ..... 1%  $\downarrow$  9%  
Dev error ..... 10%  $\downarrow$

Training-dev set: Same distribution as training set, but not used for training



# Bias/variance on mismatched training and dev/test sets

Human level

Training set error

Training - dev set error

→ Dev error

→ Test error

4% ↑ avoidable bias

7% ↑ variance

10% ↓ data mismatch

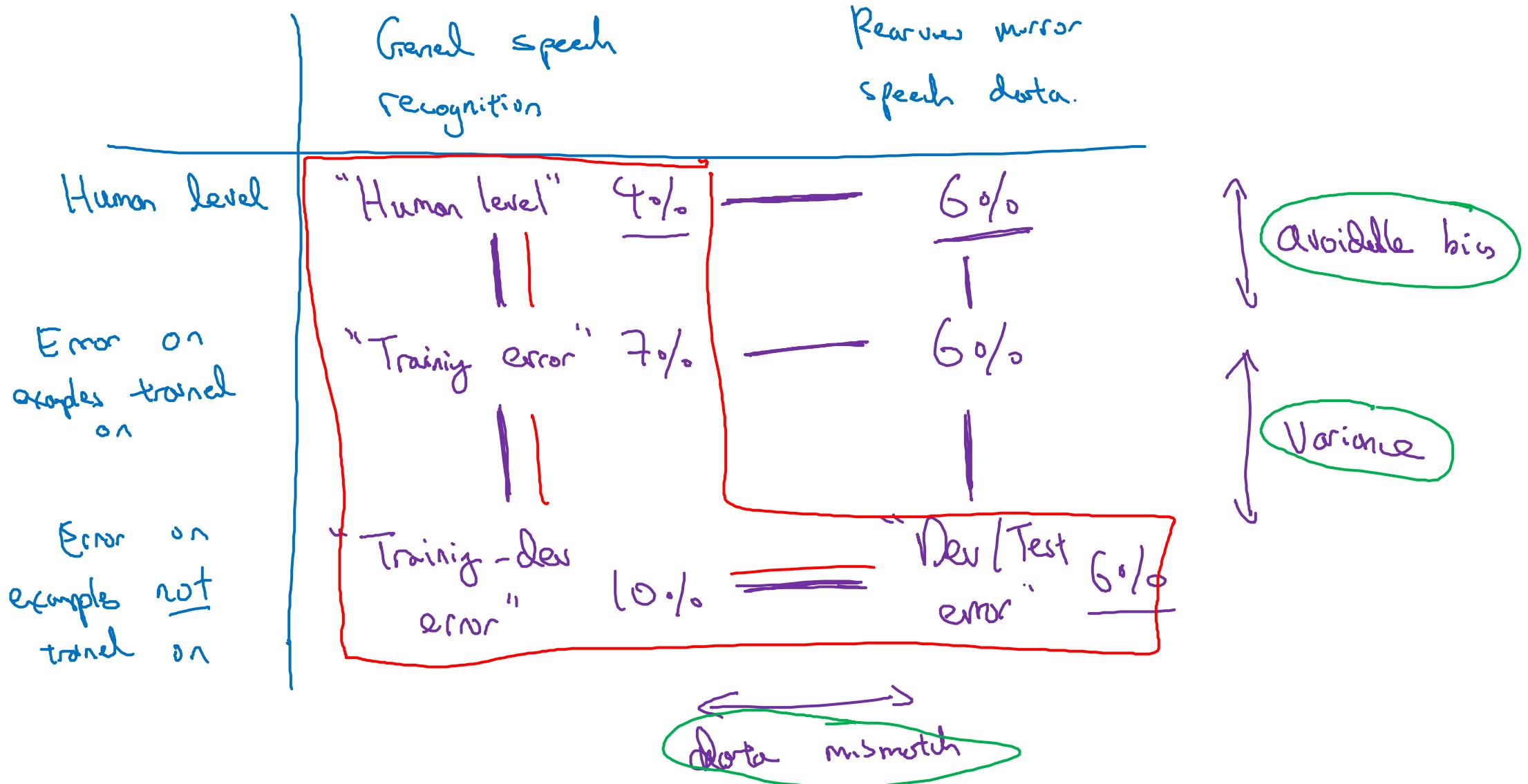
12% ↓ degree of overfitting  
to dev set.

4%

7% }  
10% }

6% }  
6% }

# More general formulation





deeplearning.ai

Mismatched training  
and dev/test data

---

Addressing data  
mismatch

# Addressing data mismatch

- • Carry out manual error analysis to try to understand difference between training and dev/test sets

E.g. noisy - car noise

street numbers

- • Make training data more similar; or collect more data similar to dev/test sets

E.g. Simulate noisy in-car data

# Artificial data synthesis



+

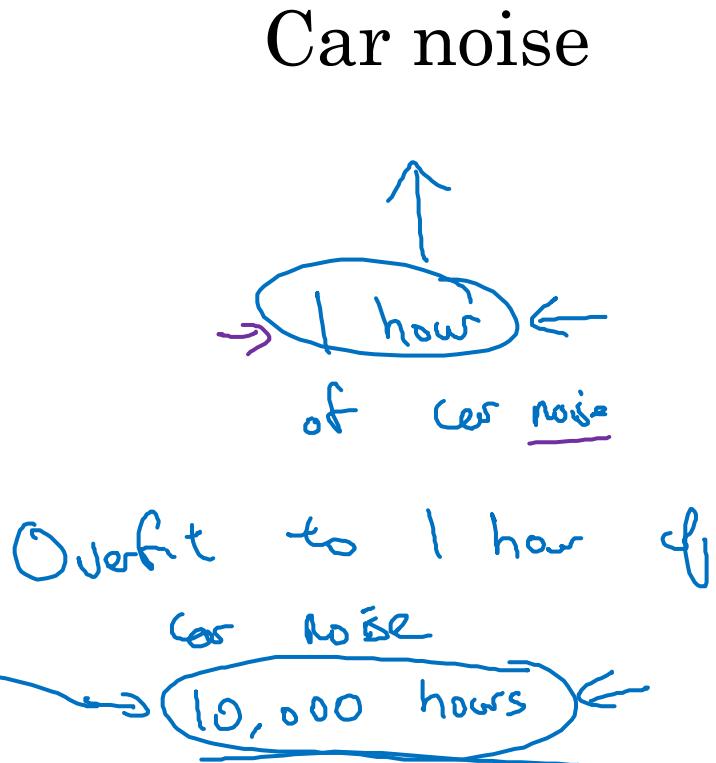


=

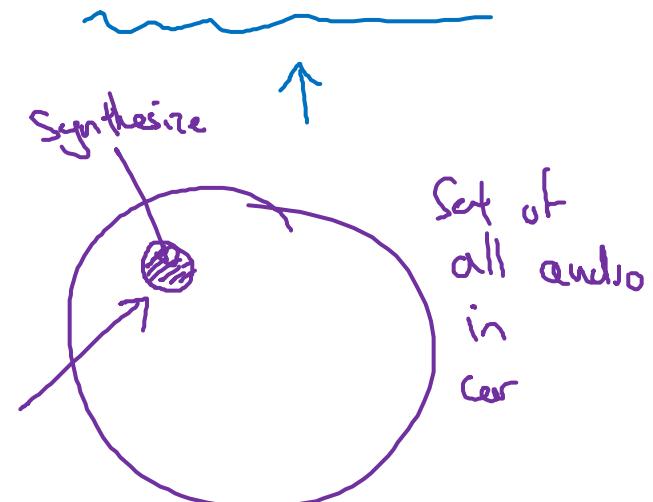


“The quick brown  fox jumps over the lazy dog.”

10,000 hours



Synthesized in-car audio

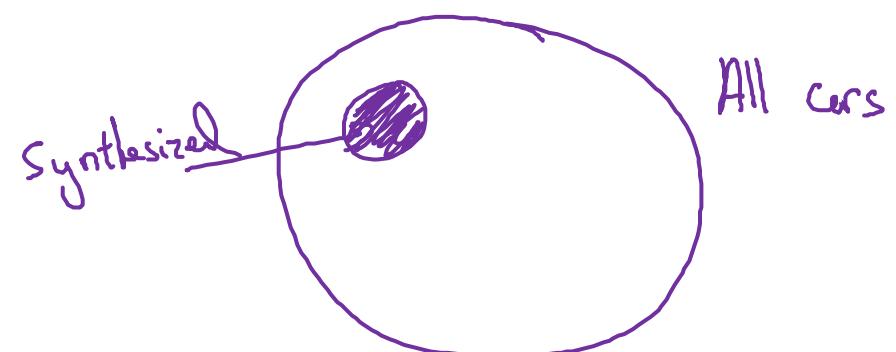


# Artificial data synthesis

Car recognition:



$N \approx 20$  cars





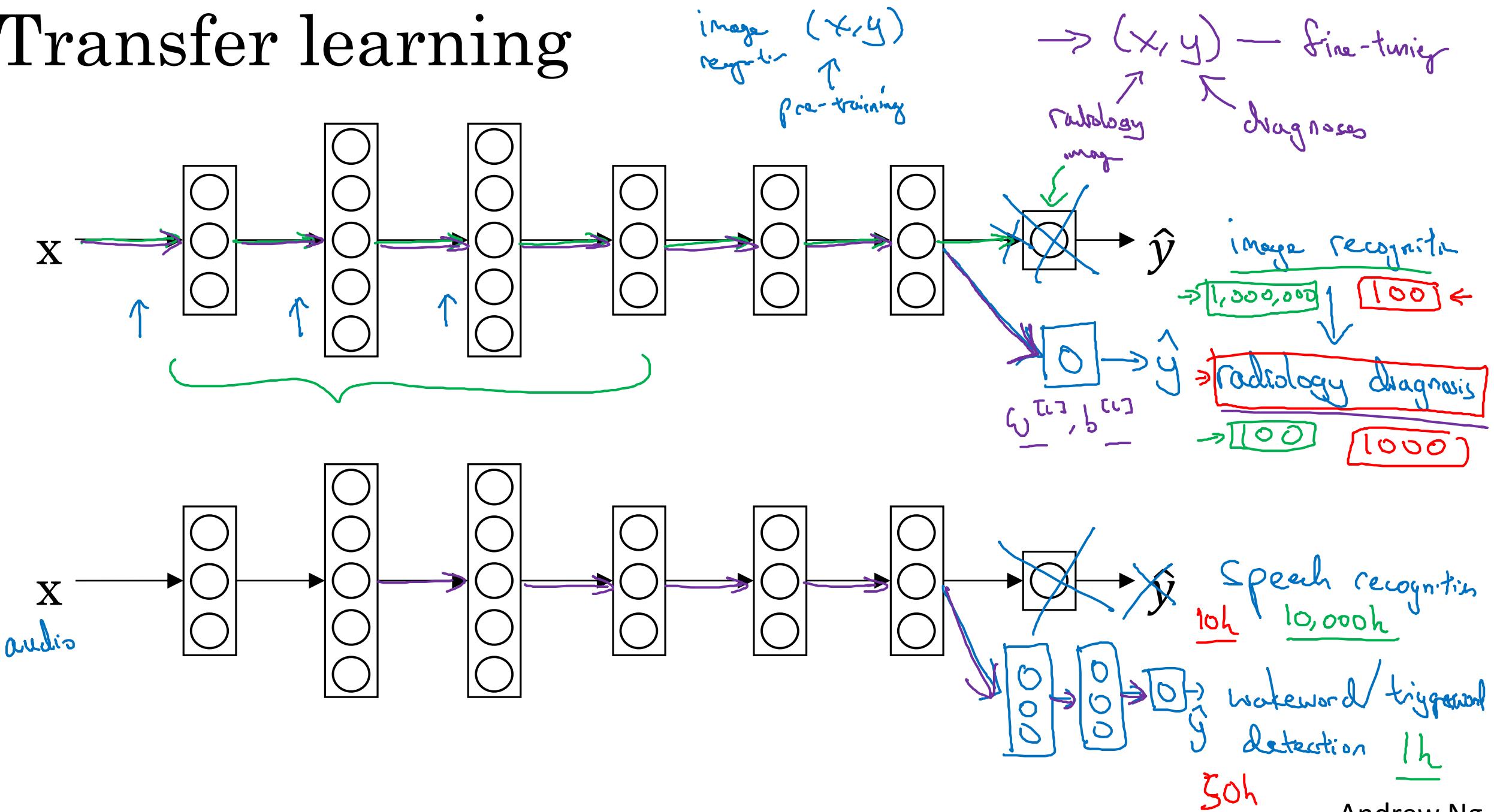
deeplearning.ai

Learning from  
multiple tasks

---

Transfer learning

# Transfer learning



# When transfer learning makes sense

Transfer from A  $\rightarrow$  B

- Task A and B have the same input  $x$ .
- You have a lot more data for Task A than Task B.  

- Low level features from A could be helpful for learning B.



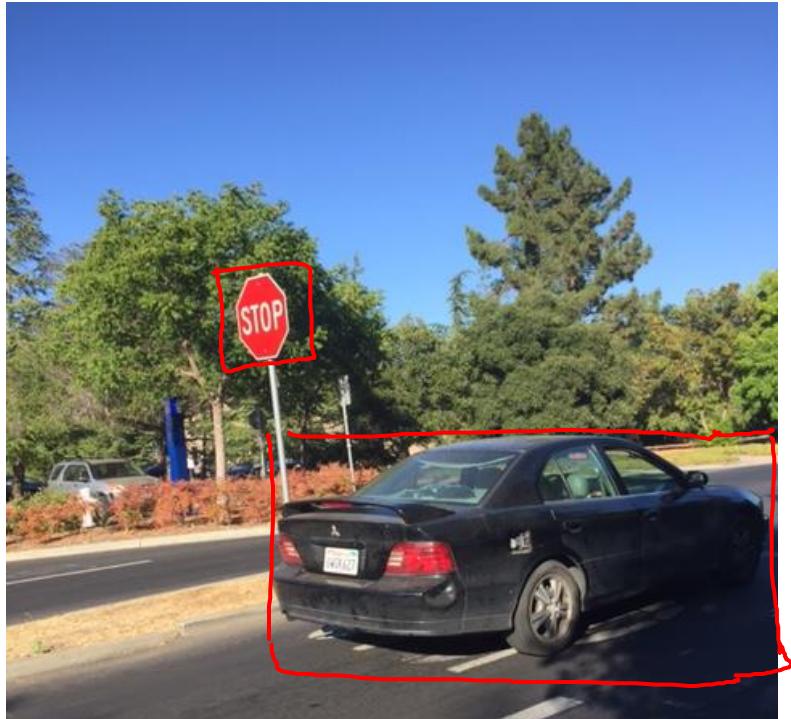
deeplearning.ai

Learning from  
multiple tasks

---

Multi-task  
learning

# Simplified autonomous driving example



$x^{(i)}$

Pedestrians

Cars

Stop signs

Traffic lights

⋮

$y^{(i)}$

0

1

1

0

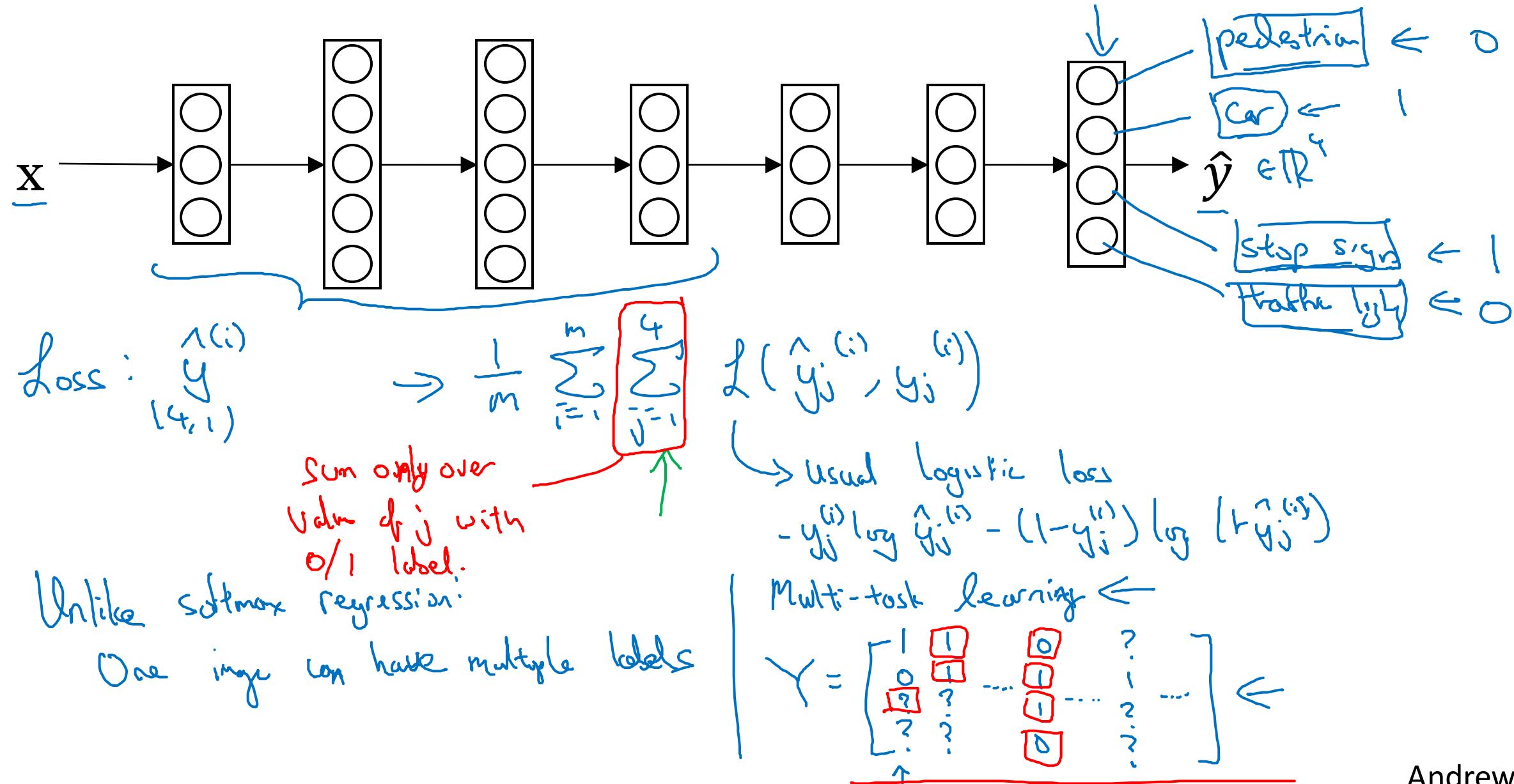
⋮

(4, 1)

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & y^{(3)}, \dots, y^{(m)} \end{bmatrix}$$

(4, m)

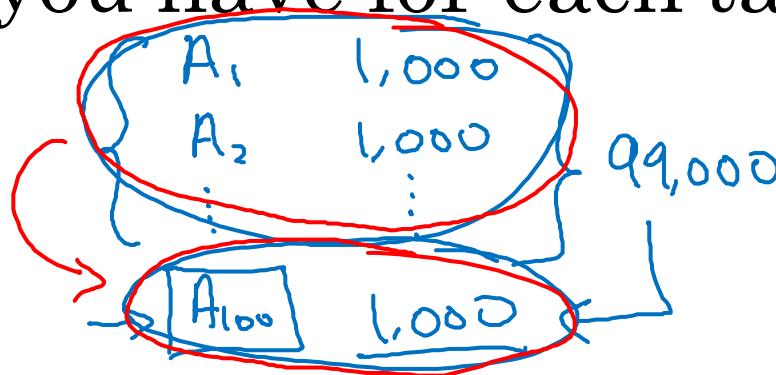
# Neural network architecture



# When multi-task learning makes sense

- Training on a set of tasks that could benefit from having shared lower-level features.
- Usually: Amount of data you have for each task is quite similar.

$$\begin{array}{ll} A & \underline{1,000,000} \\ \downarrow & \downarrow \\ B & \underline{1,000} \end{array}$$



- Can train a big enough neural network to do well on all the tasks.



deeplearning.ai

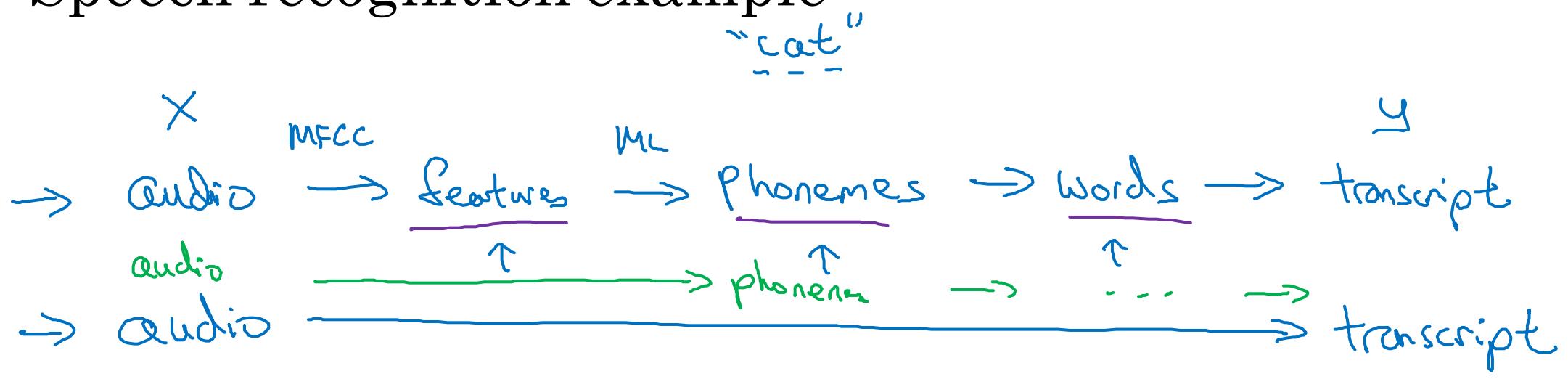
End-to-end deep  
learning

---

What is  
end-to-end  
deep learning

# What is end-to-end learning?

## Speech recognition example



3,000h



10,000h



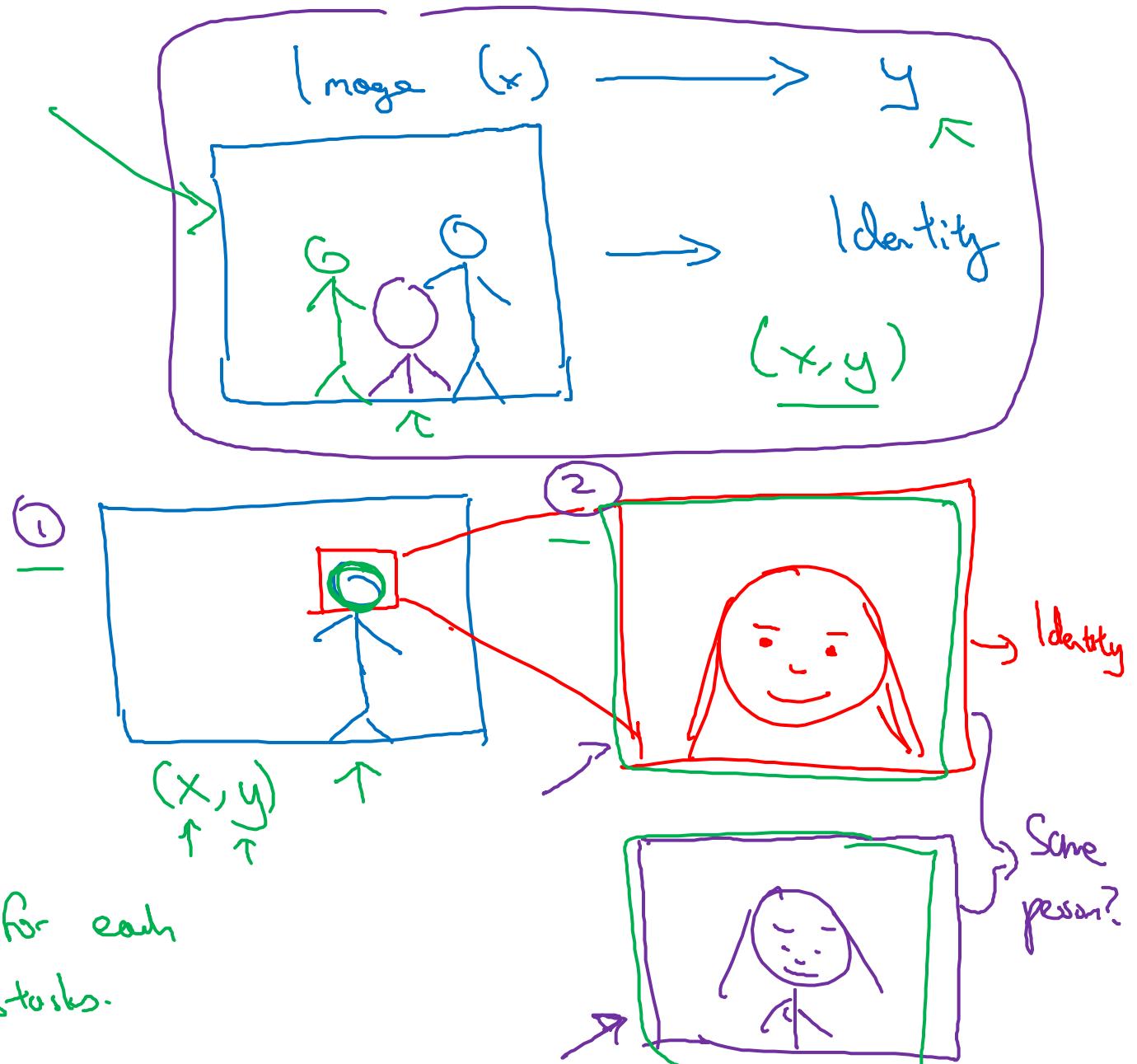
100,000h

# Face recognition



[Image courtesy of Baidu]

Have data for each  
of 2 subtasks.

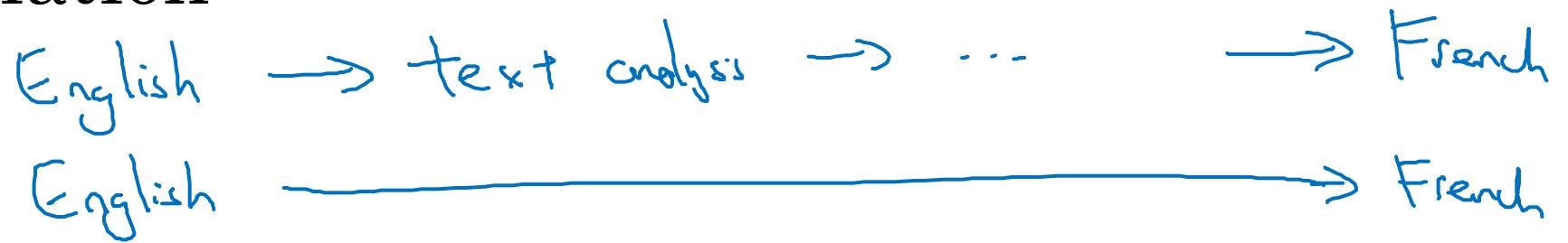


# More examples

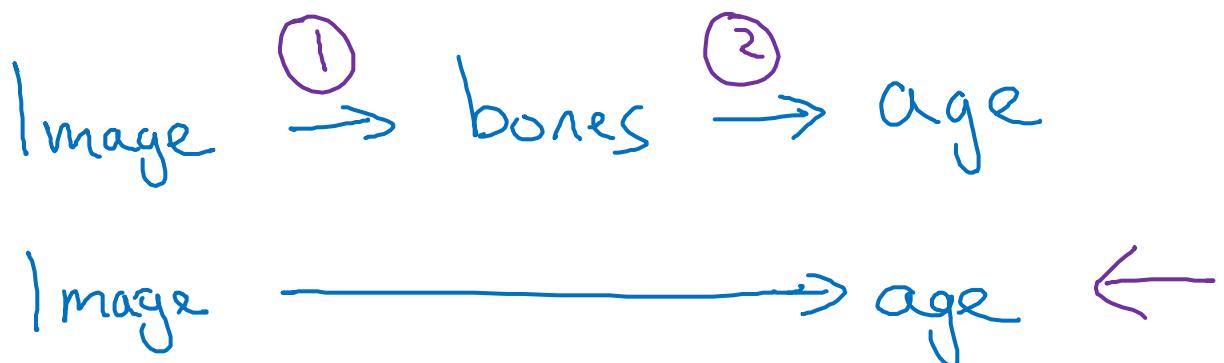
## Machine translation

$(x, y)$

English → French



Estimating child's age:





deeplearning.ai

End-to-end deep  
learning

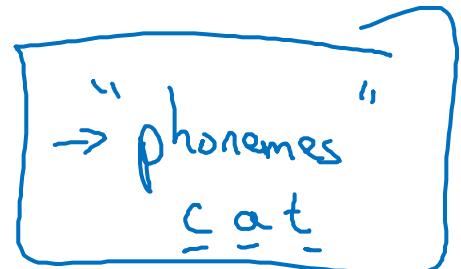
---

Whether to use  
end-to-end learning

# Pros and cons of end-to-end deep learning

Pros:

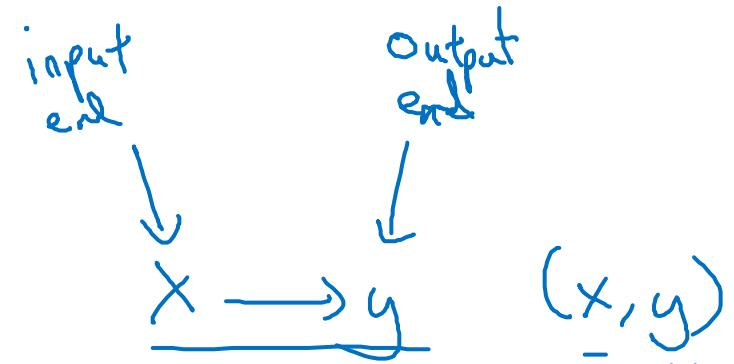
- Let the data speak  $x \rightarrow y$
- Less hand-designing of components needed



Cons:

- May need large amount of data
- Excludes potentially useful hand-designed components

$$x - - - - \rightarrow y$$

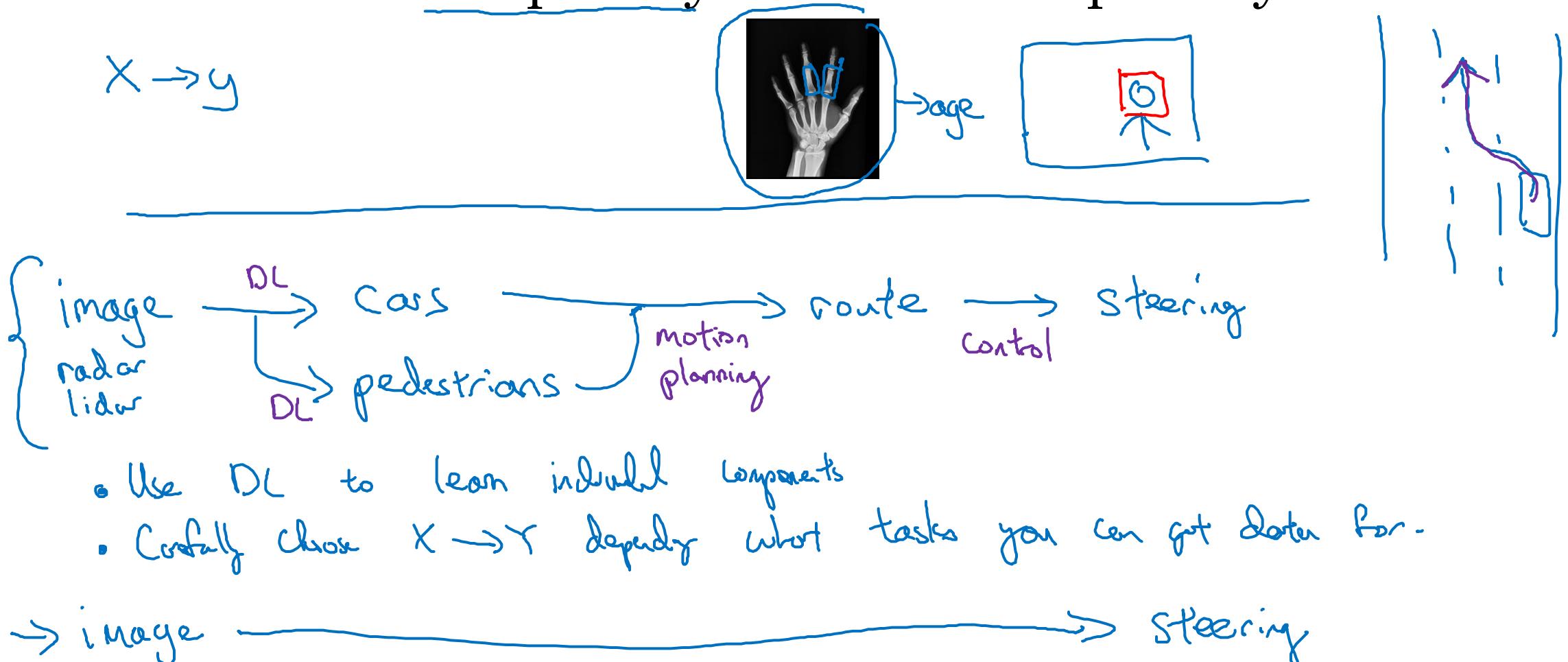


Data  
-----

Hand-design  
-----

# Applying end-to-end deep learning

Key question: Do you have sufficient data to learn a function of the complexity needed to map  $x$  to  $y$ ?





deeplearning.ai

# Convolutional Neural Networks

---

## Computer vision

# Computer Vision Problems

Image Classification



Cat? (0/1)

Neural Style Transfer



Object detection



Andrew Ng

# Deep Learning on large images



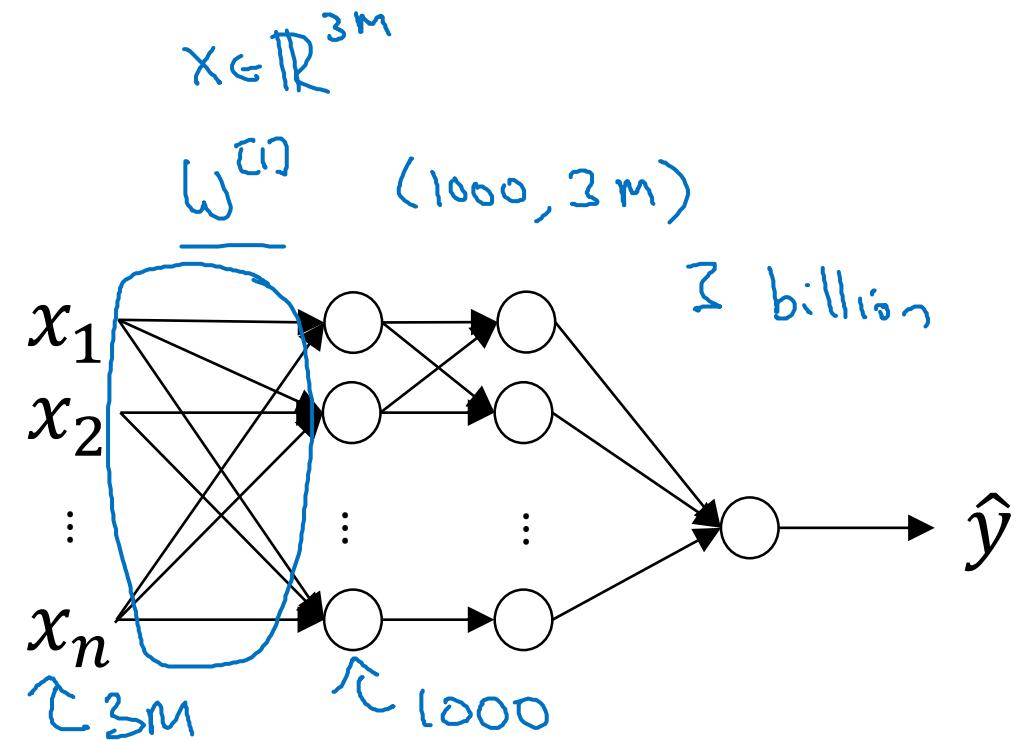
$64 \times 64 \times 3$

→ Cat? (0/1)

12288



$1000 \times 1000 \times 3$   
= 3 million





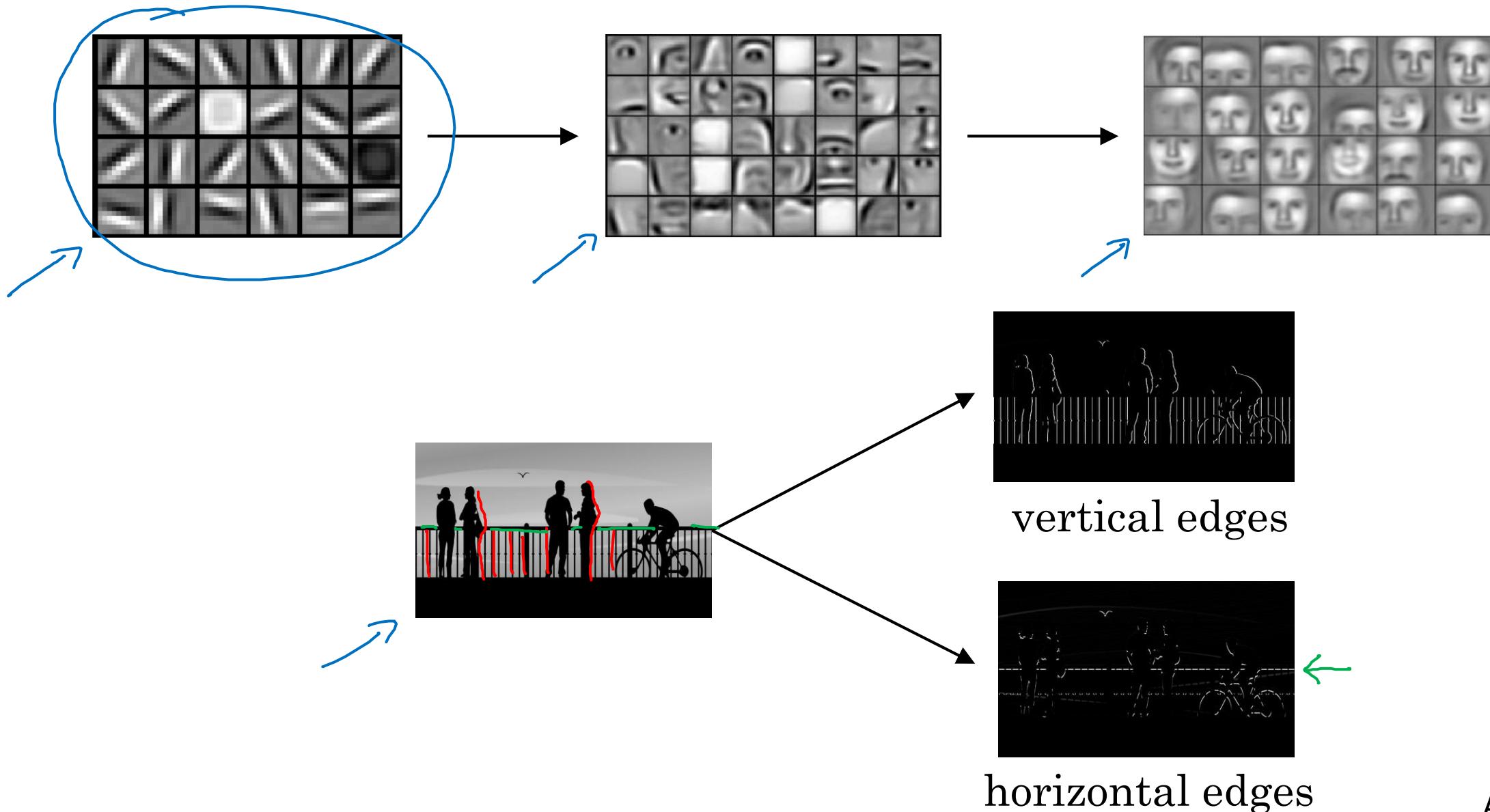
deeplearning.ai

# Convolutional Neural Networks

---

Edge detection  
example

# Computer Vision Problem

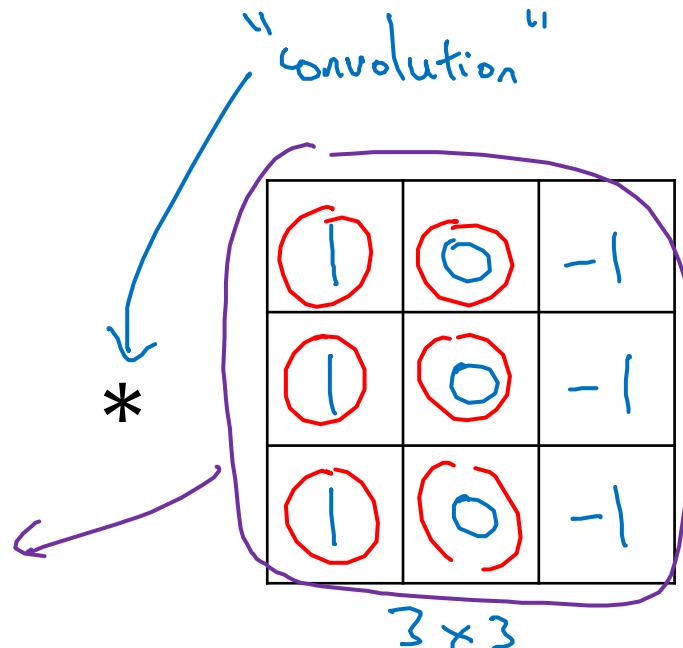


# Vertical edge detection

$$3 \times 1 + 1 \times 1 + 2 \times 1 + 0 \times 0 + 5 \times 0 + 7 \times 0 + 1 \times 1 + 8 \times -1 + 2 \times -1 = -5$$

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9

$6 \times 6$



=

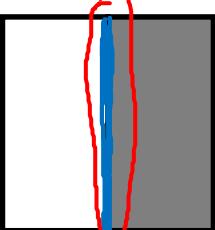
-5	-4	0	8
-10	-2	2	3
0	-2	-4	-7
-3	-2	-3	-16

$4 \times 4$

# Vertical edge detection

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

$6 \times 6$



$\uparrow \uparrow \uparrow$

\*

1	0	-1
1	0	-1
1	0	-1

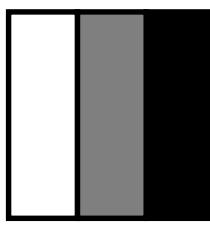
$3 \times 3$

=

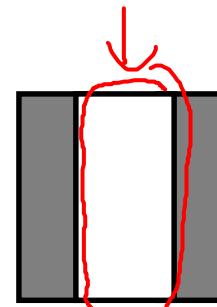
0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

$\uparrow 4 \times 4$

\*



$\uparrow \uparrow \uparrow$



Andrew Ng



deeplearning.ai

# Convolutional Neural Networks

---

More edge  
detection

# Vertical edge detection examples

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0



0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10



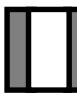
\*

1	0	-1
1	0	-1
1	0	-1



=

0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0



\*

1	0	-1
1	0	-1
1	0	-1

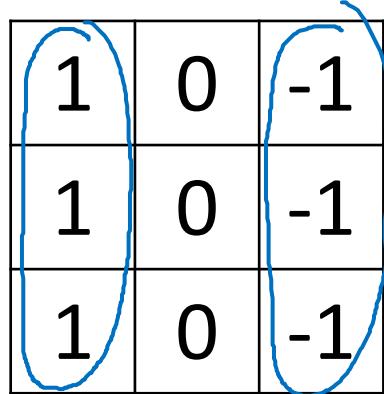


=

0	-30	-30	0
0	-30	-30	0
0	-30	-30	0
0	-30	-30	0



# Vertical and Horizontal Edge Detection

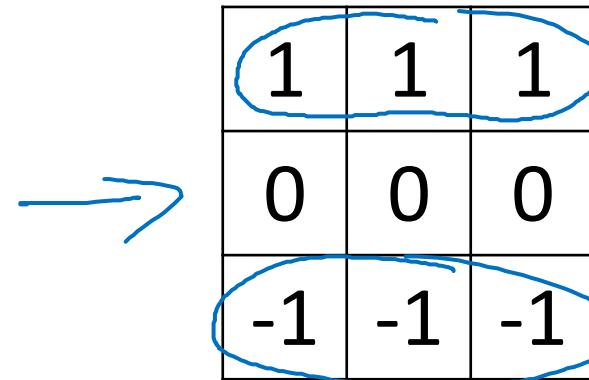


A 3x3 matrix with values:

1	0	-1
1	0	-1
1	0	-1

Handwritten annotations show blue circles around the '1's in the first and third columns of the first and last rows.

Vertical

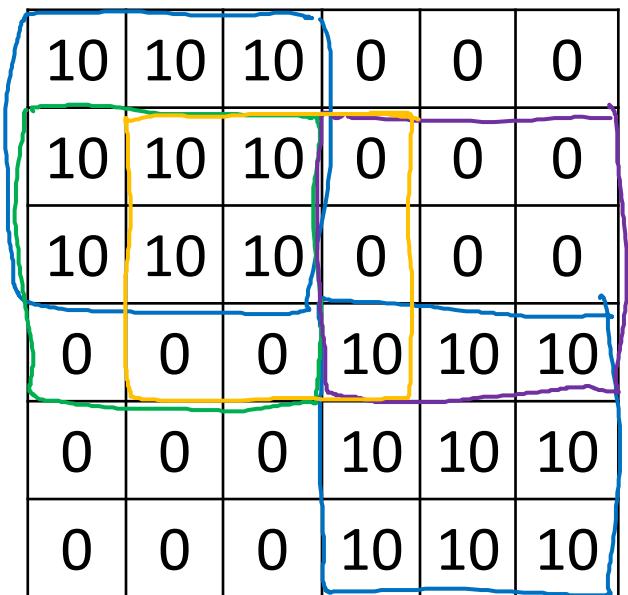


A 3x3 matrix with values:

1	1	1
0	0	0
-1	-1	-1

Handwritten annotations show blue circles around the '1's in the first and third columns of the first and last rows.

Horizontal

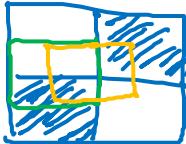


A 6x6 matrix with values:

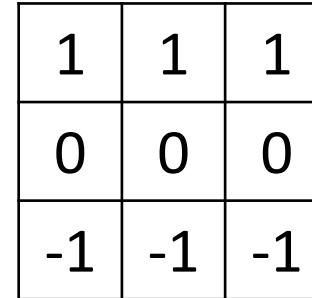
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
0	0	0	10	10	10
0	0	0	10	10	10
0	0	0	10	10	10

Handwritten annotations include green, orange, and purple boxes highlighting vertical edges in the first three columns of the first three rows.

$6 \times 6$



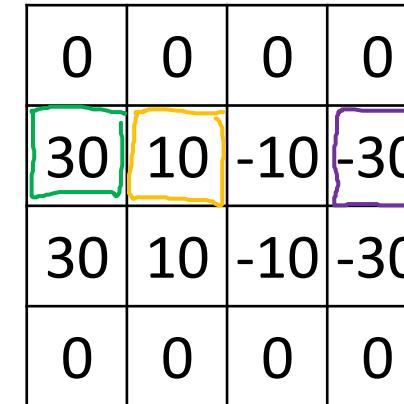
\*



A 3x3 matrix with values:

1	1	1
0	0	0
-1	-1	-1

=



A 6x6 matrix with values:

0	0	0	0
30	10	-10	-30
30	10	-10	-30
0	0	0	0

Handwritten annotations include green, orange, and purple boxes highlighting horizontal edges in the second row.

# Learning to detect edges

1	0	-1
1	0	-1
1	0	-1

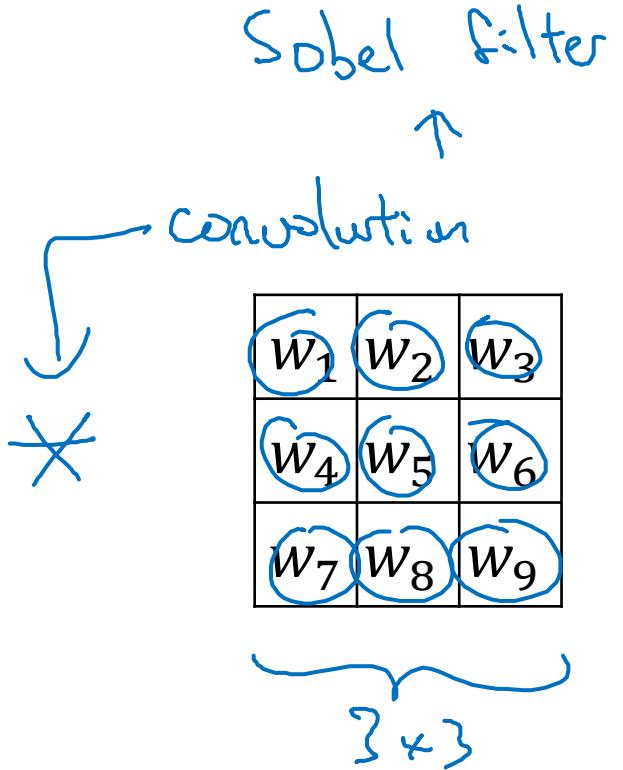
→

1	0	-1
2	0	-2
1	0	-1

3	0	-3
10	0	-10
3	0	-3

↑

3	0	1	2	7	4
1	5	8	9	3	1
2	7	2	5	1	3
0	1	3	1	7	8
4	2	1	6	2	8
2	4	5	2	3	9



=

$45^\circ$

$70^\circ$

$73^\circ$


Scharr filter

↑



deeplearning.ai

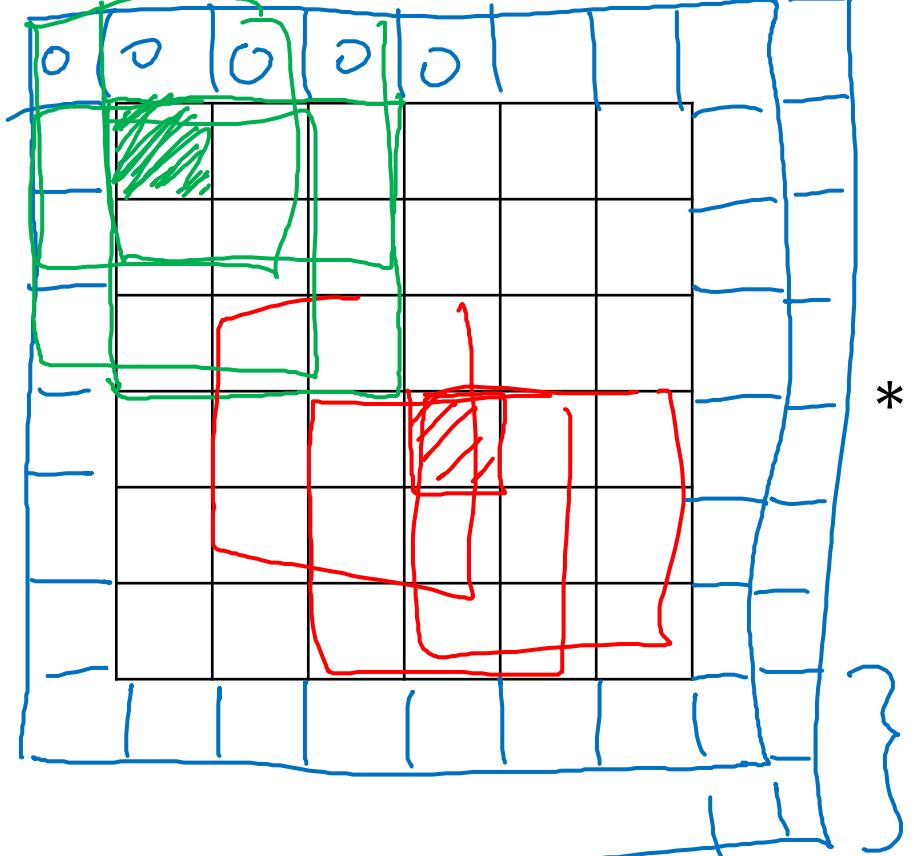
# Convolutional Neural Networks

---

## Padding

# Padding

- shrinky output
- throw away info from edge

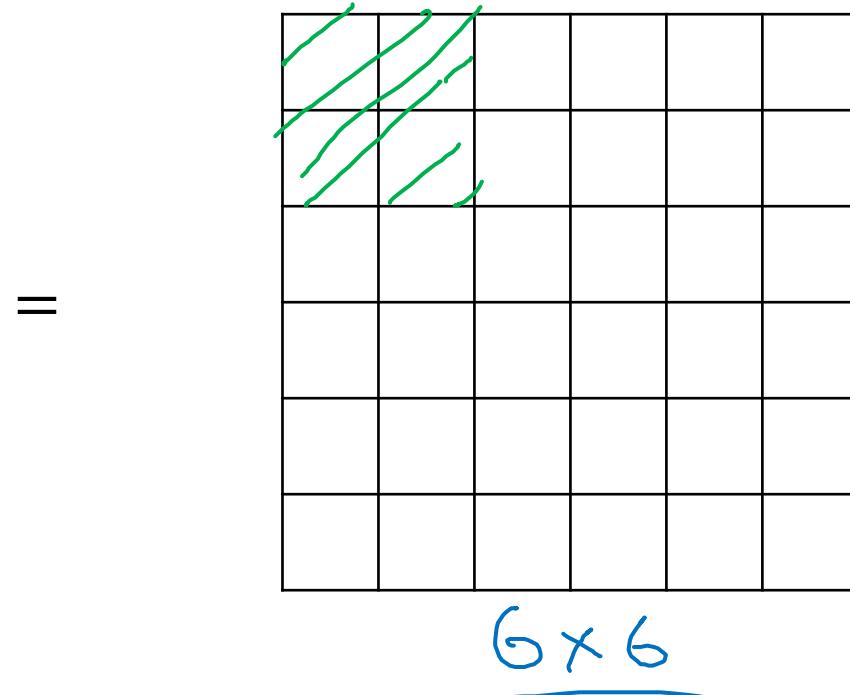


$$* \quad \begin{array}{|c|c|c|}\hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array}$$

$3 \times 3$   
 $f \times f$

$$n-f+1 \times n-f+1$$
$$6-3+1=4$$

$$P = \text{padding} = 1$$



$$\frac{n+2p-f+1 \times n+2p-f+1}{6+2-3+1 \times \underline{\quad}} = 6 \times 6$$

# Valid and Same convolutions

→ n → padding

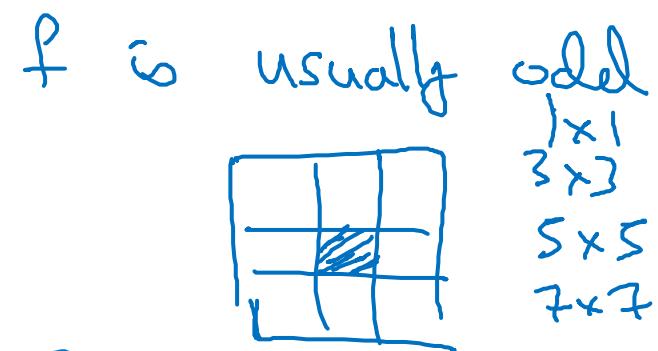
“Valid”:  $n \times n \quad * \quad f \times f \quad \rightarrow \frac{n-f+1}{f} \times \frac{n-f+1}{f}$

$6 \times 6 \quad * \quad 3 \times 3 \quad \rightarrow \quad 4 \times 4$

“Same”: Pad so that output size is the same as the input size.

$$n + 2p - f + 1 = n \Rightarrow p = \frac{f-1}{2}$$

$3 \times 3 \quad p = \frac{3-1}{2} = 1 \quad | \quad \begin{matrix} S \times S \\ f = 3 \end{matrix} \quad p=2$





deeplearning.ai

# Convolutional Neural Networks

---

## Strided convolutions

# Strided convolution

A 7x7 input matrix with values ranging from -1 to 9. A 3x3 kernel is applied to the top-left 3x3 submatrix. Blue arrows indicate the receptive field of the output unit at position (1,1).

2	3	3	4	7	3	4	4	6	3	2	4	9	4
6	1	6	0	9	1	8	0	7	1	4	0	3	2
3	-3	4	4	8	-3	3	4	8	-3	9	4	7	4
7	1	8	0	3	1	6	0	6	1	3	0	4	2
4	-3	2	4	1	-3	8	4	3	-3	4	4	6	4
3	1	2	0	4	1	1	0	9	1	8	0	3	2
0	-1	1	0	3	-1	9	0	2	-1	1	0	4	3

7x7

$n \times n$  \*  $f \times f$   
 padding p      stride s  
 $s=2$

\*

3	4	4
1	0	2
-1	0	3

$3 \times 3$

stride = 2

=

91	100	83
69	91	127
44	72	74

$3 \times 3$

$\lfloor \frac{z}{2} \rfloor = \text{floor}(z)$

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$

$$\left\lfloor \frac{7+0-3}{2} + 1 \right\rfloor = \frac{4}{2} + 1 = 3$$

# Summary of convolutions

$n \times n$  image       $f \times f$  filter

padding  $p$       stride  $s$

Output size:

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \quad \times \quad \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor$$


# Technical note on cross-correlation vs. convolution

Convolution in math textbook:

2	3	7	5	4	6	2
6	6	9	4	8	7	4
3	4	8	3	3	8	9
7	8	3	6	6	6	3
4	2	1	8	3	3	4
3	2	4	1	9	8	

$$\begin{matrix} & * & \begin{matrix} 3 & 4 & 5 \\ 1 & 0 & 2 \\ -1 & 9 & 7 \end{matrix} \\ \begin{matrix} 7 & 2 & 5 \\ 9 & 0 & 4 \\ -1 & 1 & 3 \end{matrix} & \end{matrix}$$

$$= \begin{matrix} & \begin{matrix} 1 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix} \end{matrix}$$

$$(A * B) * C = A * (B * C)$$



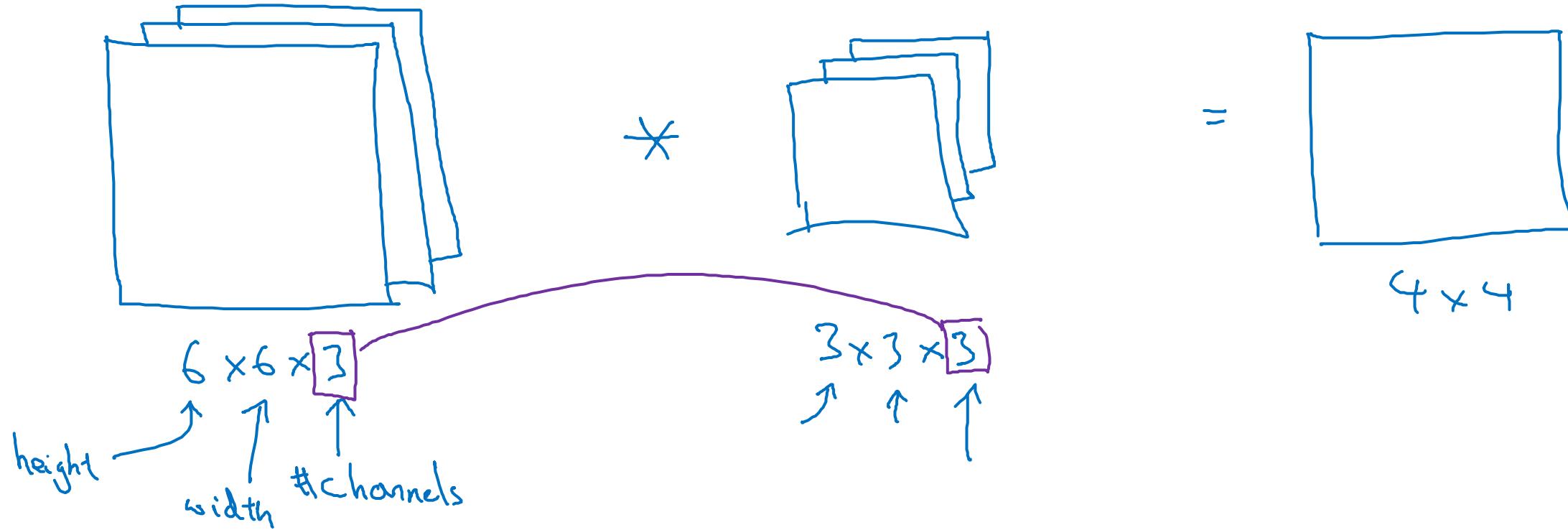
deeplearning.ai

# Convolutional Neural Networks

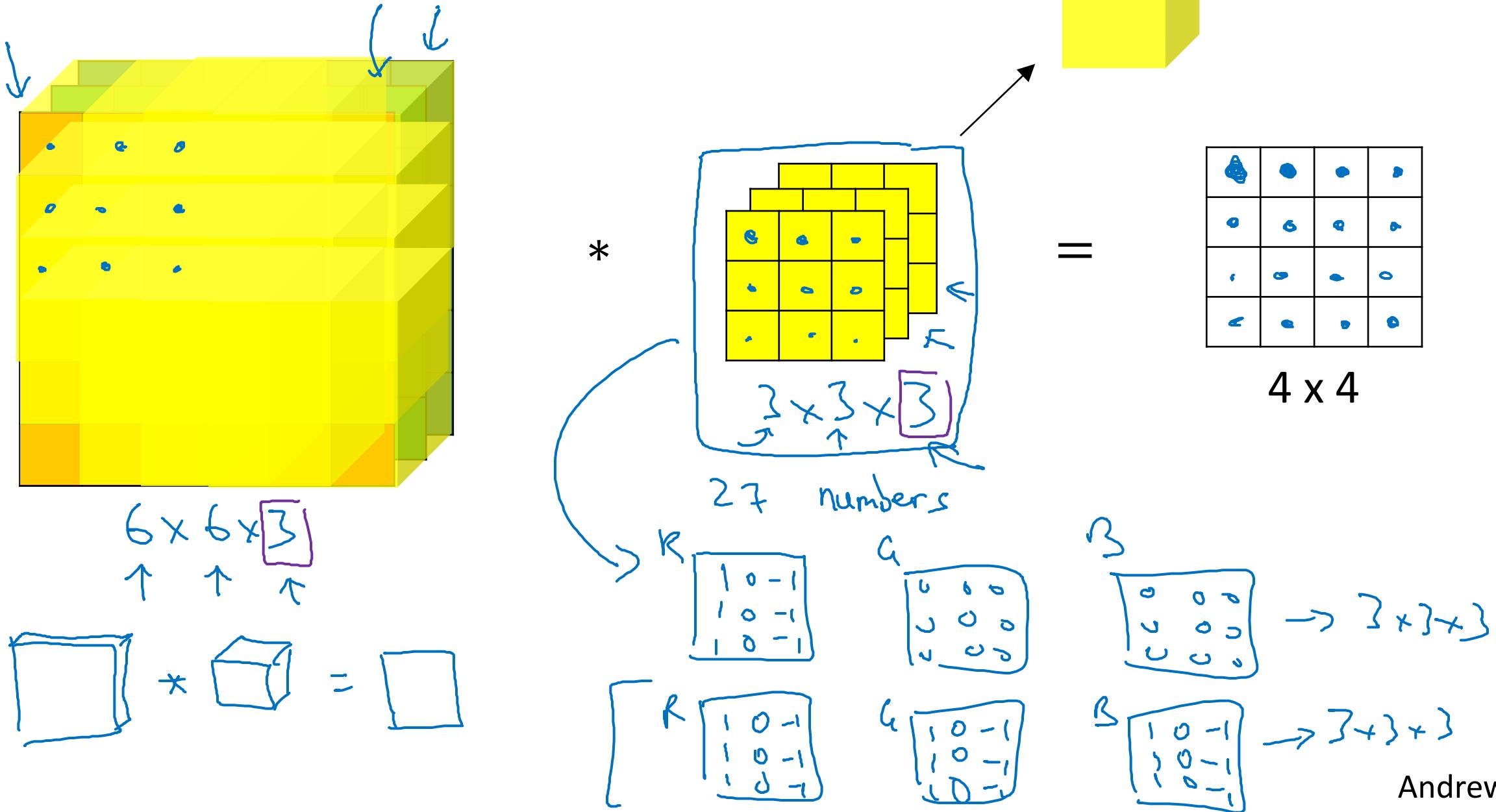
---

## Convolutions over volumes

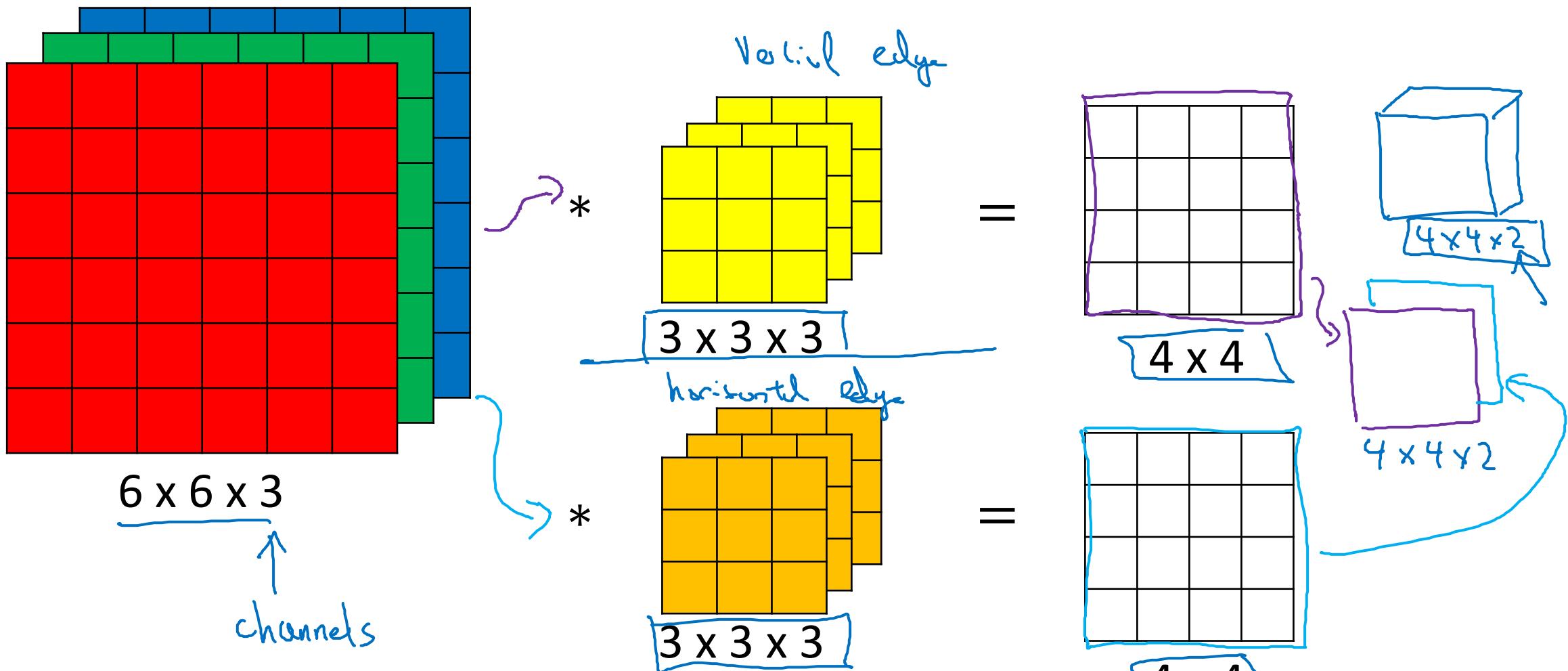
# Convolutions on RGB images



# Convolutions on RGB image



# Multiple filters



Summary:  $n \times n \times n_c$   $\times f \times f \times n_c$   $\rightarrow \frac{n-f+1}{4} \times \frac{n-f+1}{4} \times \frac{n_c}{2} \# \text{filters}$

$6 \times 6 \times 3$   $3 \times 3 \times 3$



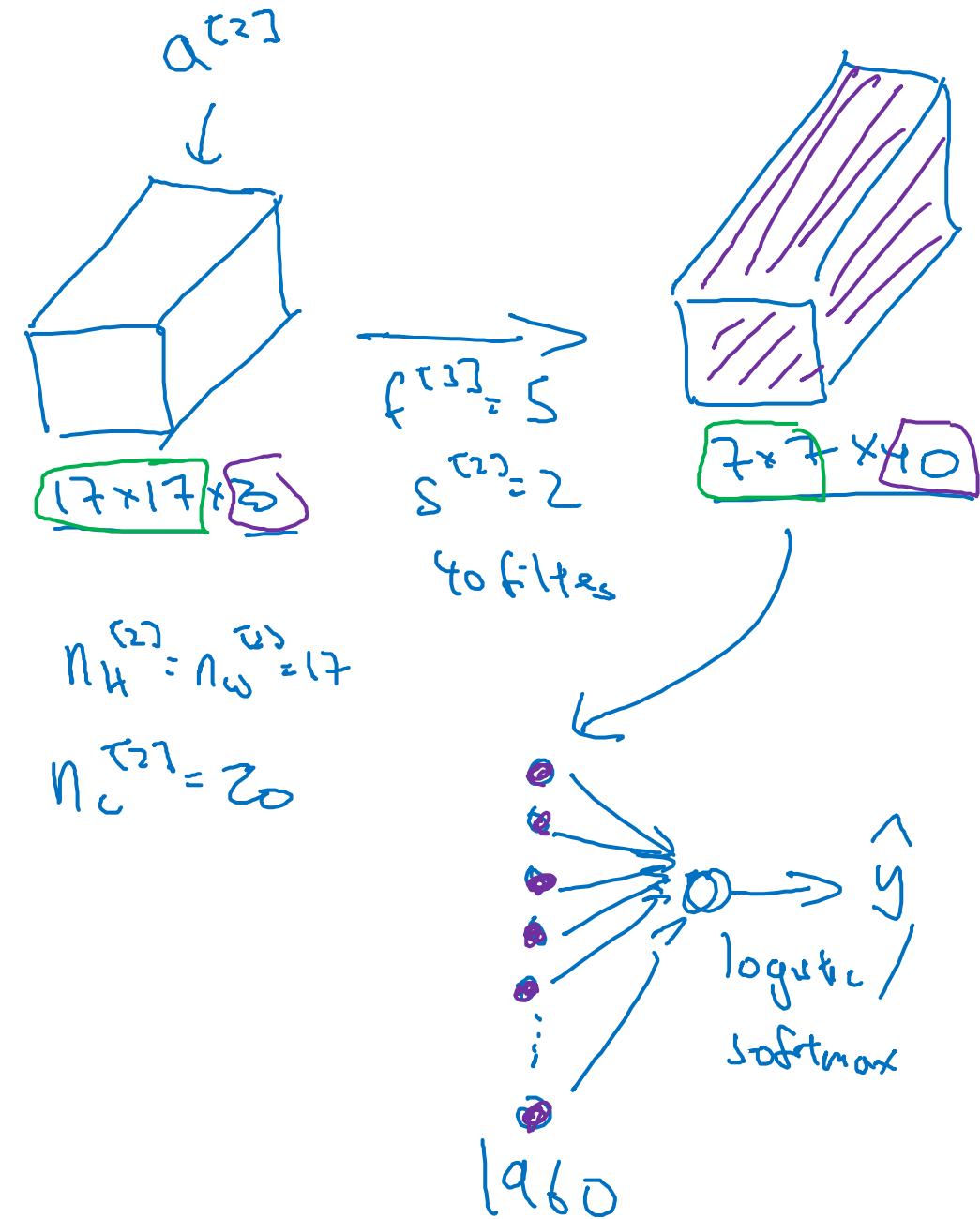
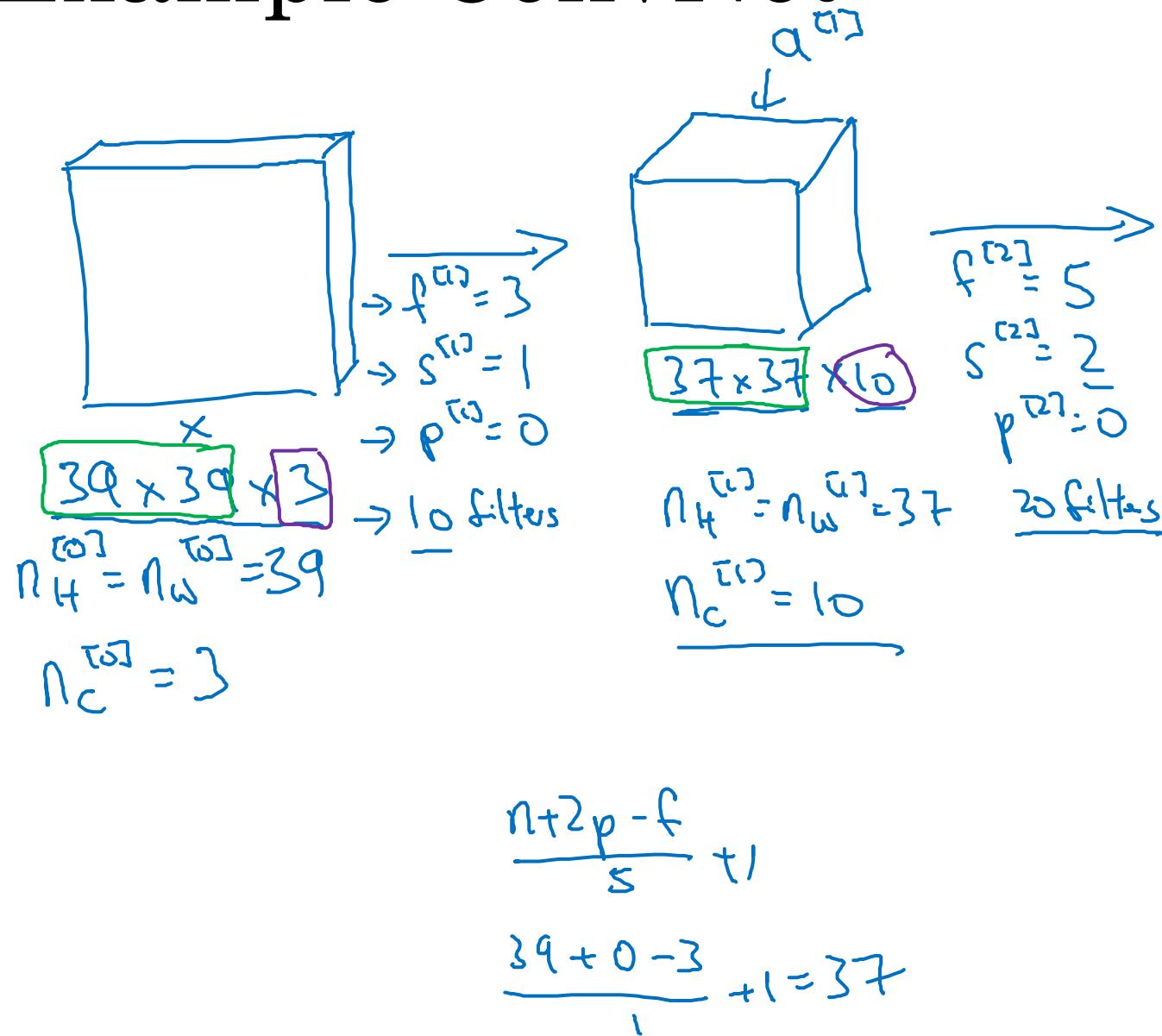
deeplearning.ai

# Convolutional Neural Networks

---

## A simple convolution network example

# Example ConvNet



# Types of layer in a convolutional network:

- Convolution (conv) ←
- Pooling (pool) ←
- Fully connected (Fc) ←



deeplearning.ai

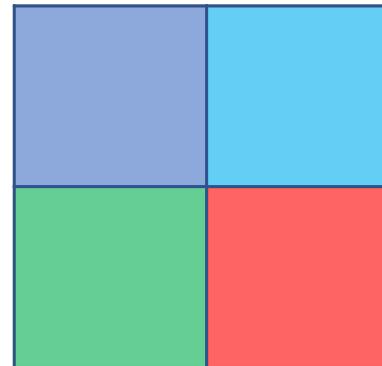
# Convolutional Neural Networks

---

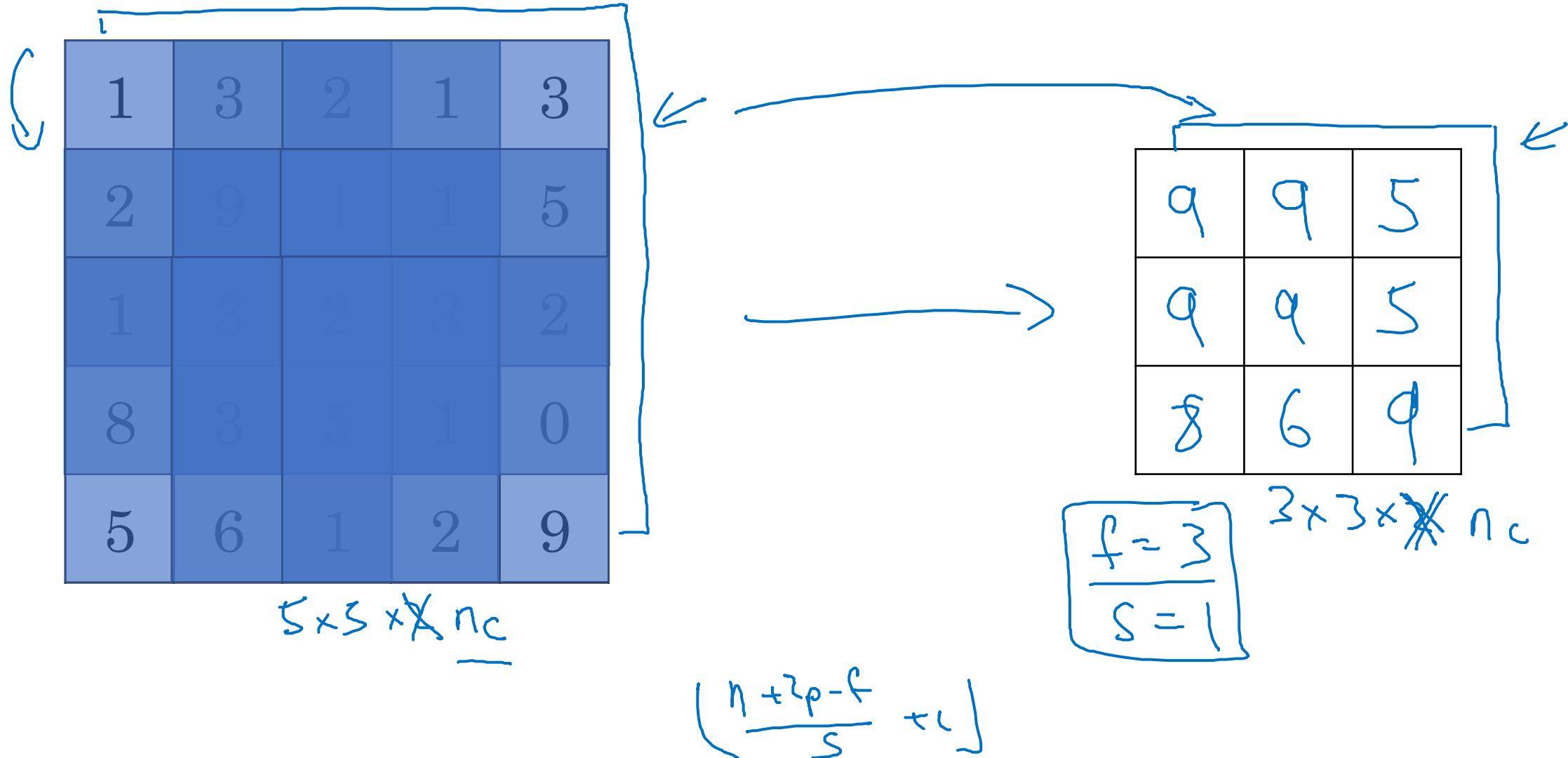
## Pooling layers

# Pooling layer: Max pooling

1	3	2	1
2	9	1	1
1	3	2	3
5	6	1	2



# Pooling layer: Max pooling



# Pooling layer: Average pooling

1	3	2	1
2	9	1	1
1	4	2	3
5	6	1	2



3.75	1.25
4	2

$$f=2$$

$$s=2$$

$$\underbrace{7 \times 7 \times 1000}_{\rightarrow} \rightarrow 1 \times 1 \times 1000$$

# Summary of pooling

Hyperparameters:

$f$  : filter size

$$f=2, s=2$$

$s$  : stride

$$f=3, s=2$$

Max or average pooling

$\rightarrow p$ : padding.

No parameters to learn!

$$n_H \times n_w \times n_c$$



$$\left\lfloor \frac{n_H-f+1}{s} \right\rfloor \times \left\lfloor \frac{n_w-f}{s} + 1 \right\rfloor$$

$$\times n_c$$



deeplearning.ai

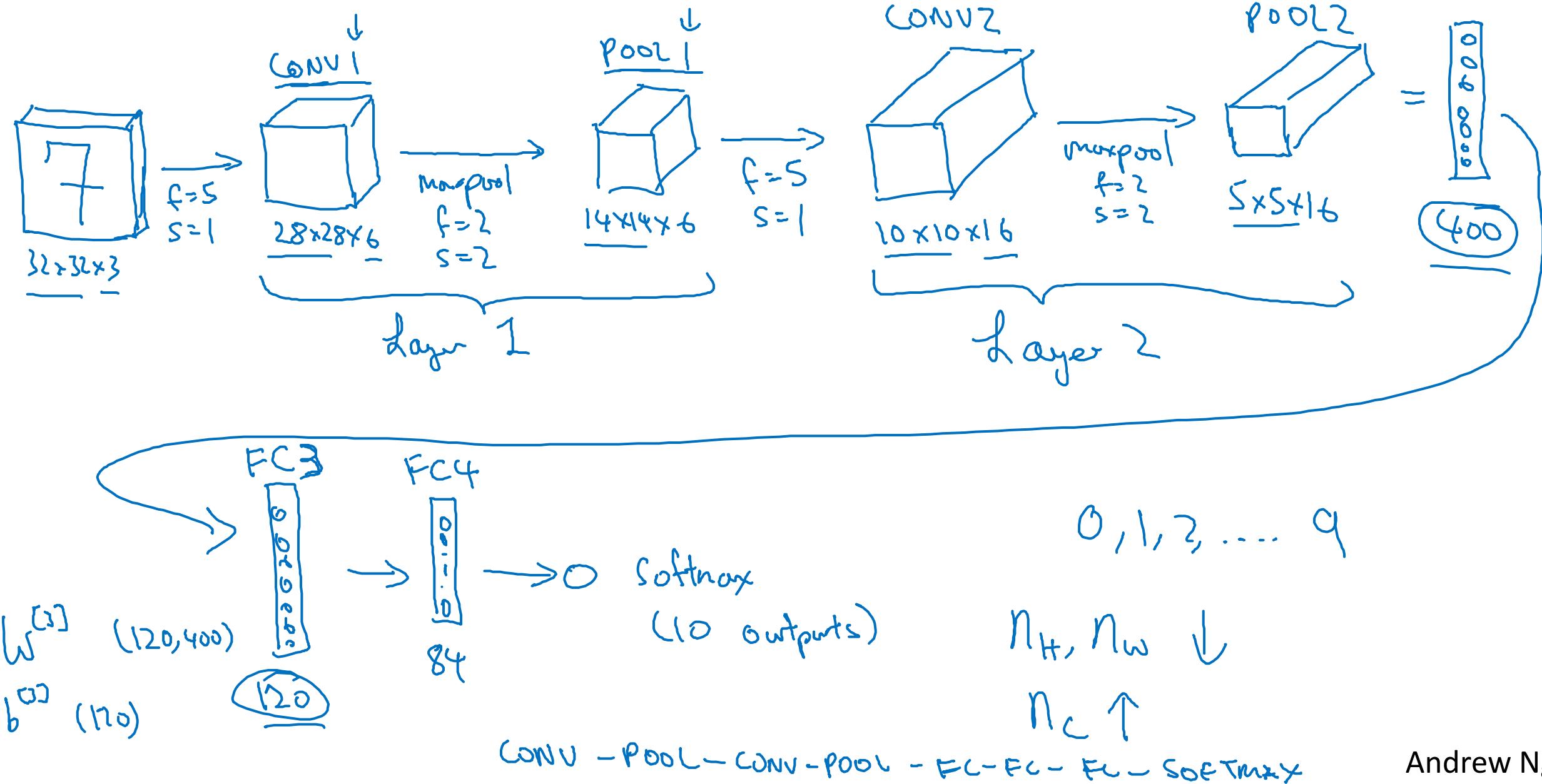
# Convolutional Neural Networks

---

## Convolutional neural network example

# Neural network example

(LeNet-5)



# Neural network example

	Activation shape	Activation Size	# parameters
Input:	(32,32,3)	— 3,072 $a^{[32]}$	0
		—	←
		—	←
		—	←
		—	←
		—	←
		—	—
		—	—
		—	—
		—	—



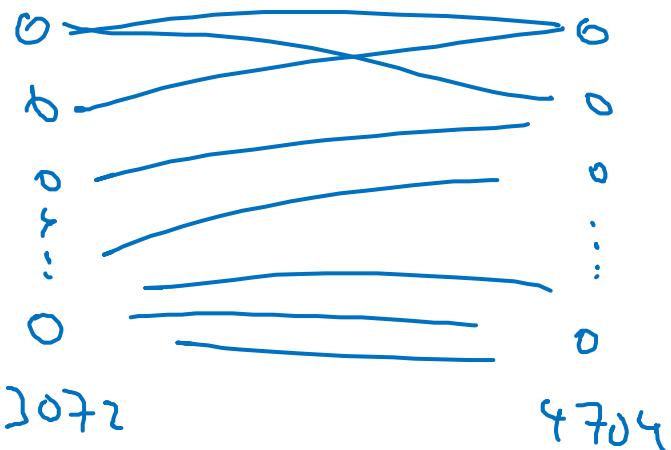
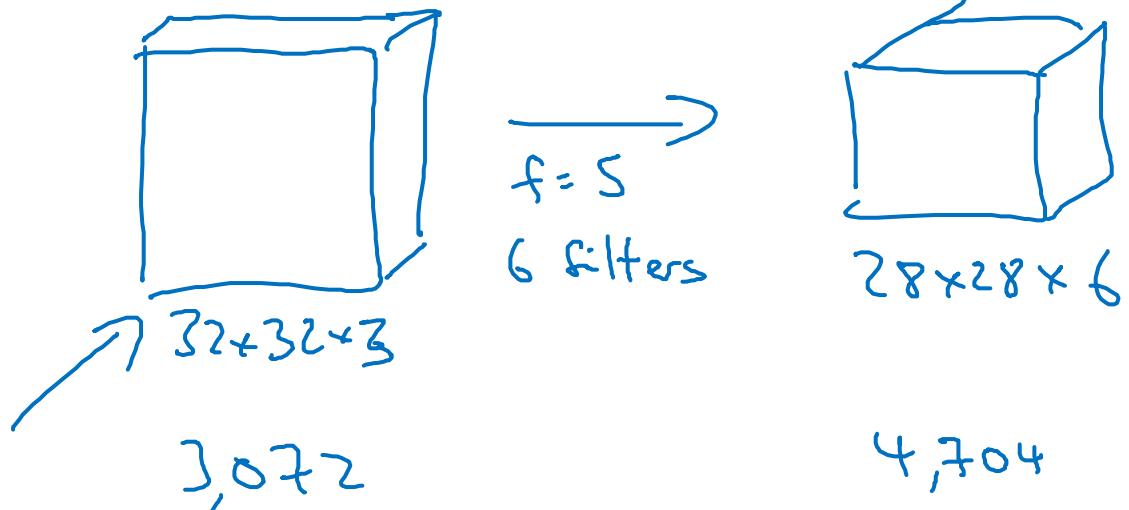
deeplearning.ai

# Convolutional Neural Networks

---

## Why convolutions?

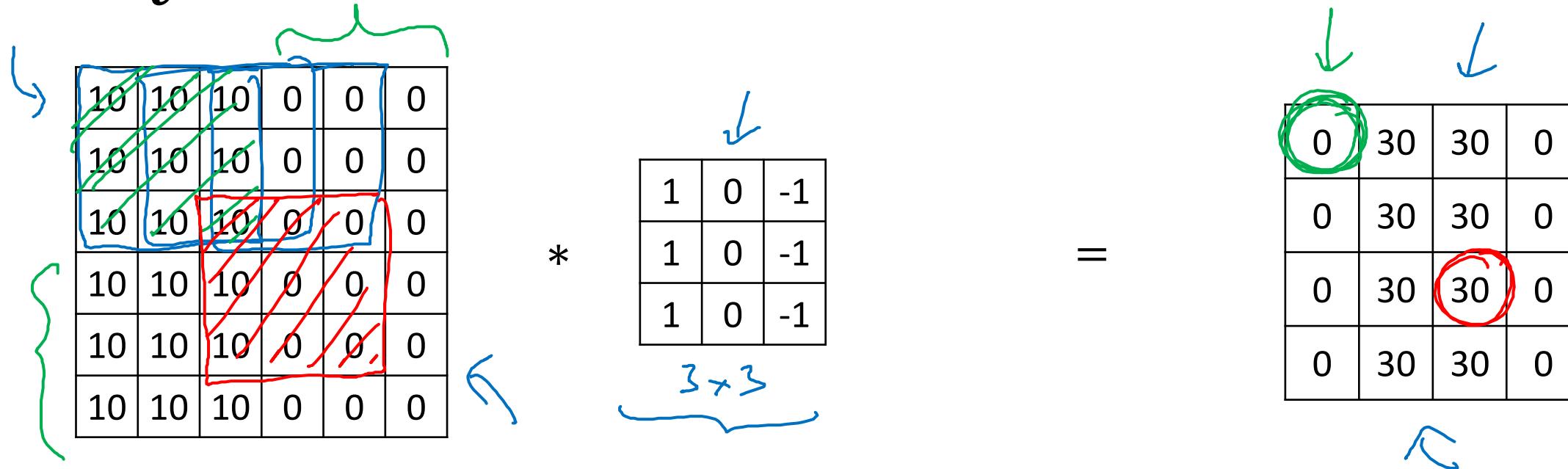
# Why convolutions



$$\begin{aligned} & 5 \times 5 - 25 \\ & 26 \\ & 6 \times 26 = 156 \text{ Parameters} \end{aligned}$$

Below the equation, the text  $3,072 \times 4,704 \approx 14M$  is written, indicating the total number of parameters in the layer.

# Why convolutions

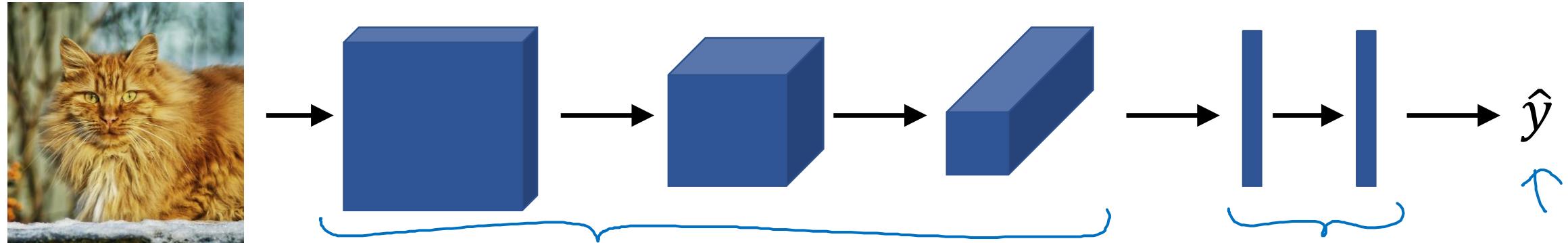


**Parameter sharing:** A feature detector (such as a vertical edge detector) that's useful in one part of the image is probably useful in another part of the image.

→ **Sparsity of connections:** In each layer, each output value depends only on a small number of inputs.

# Putting it together

Training set  $(x^{(1)}, y^{(1)}) \dots (x^{(m)}, y^{(m)})$ .



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

Use gradient descent to optimize parameters to reduce  $J$



deeplearning.ai

# Case Studies

---

Why look at  
case studies?

# Outline

Classic networks:

- LeNet-5 ←
- AlexNet ←
- VGG ←

ResNet (152)

Inception



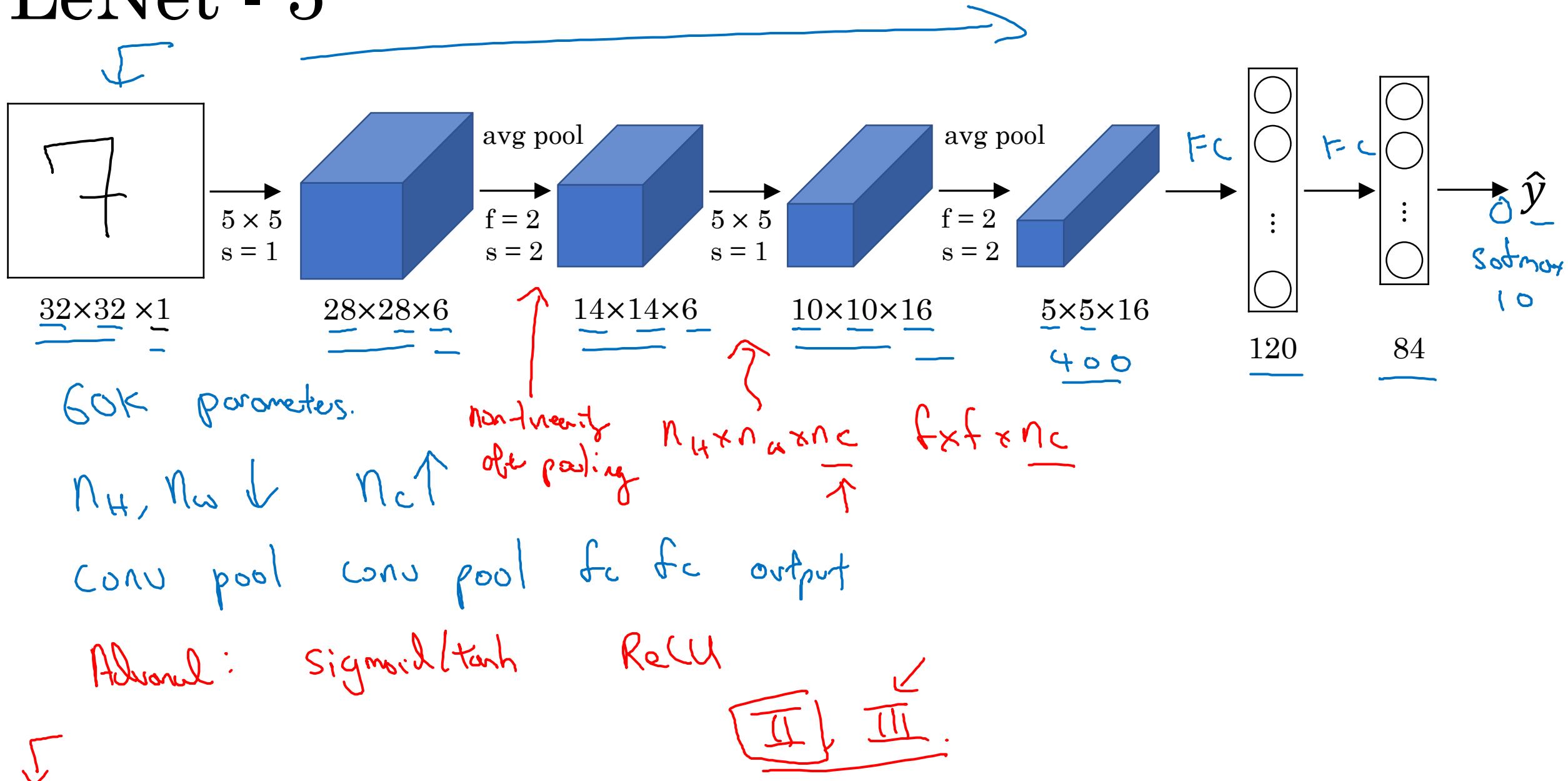
deeplearning.ai

# Case Studies

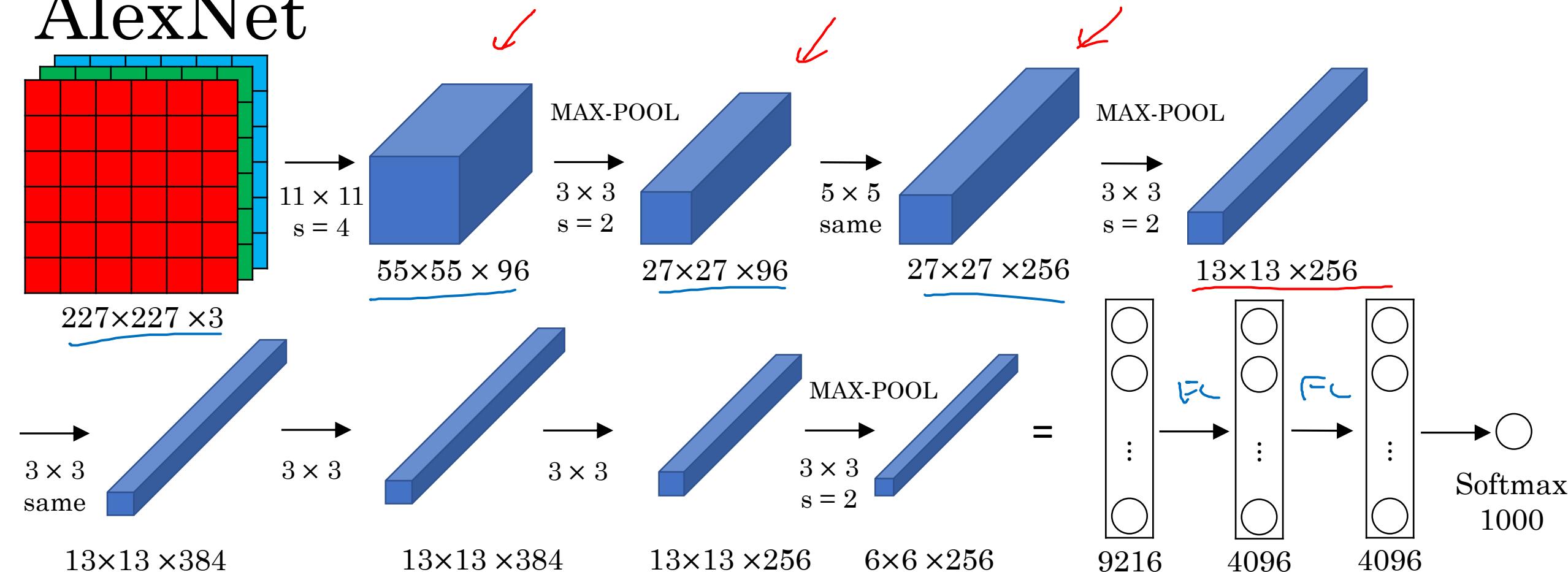
---

## Classic networks

# LeNet - 5

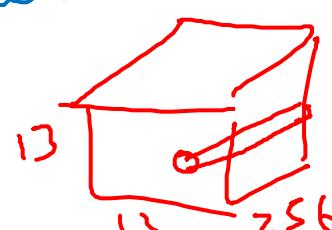


# AlexNet



- Similar to LeNet, but much bigger.
- ReLU

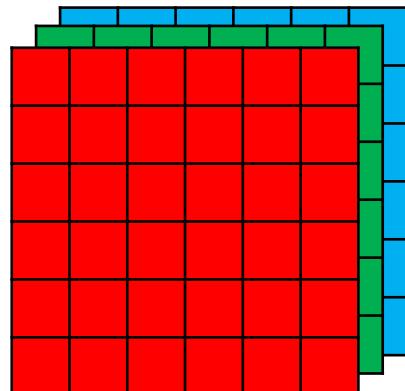
- Multiple GPUs.  
- Local Response Normalization (LRN)



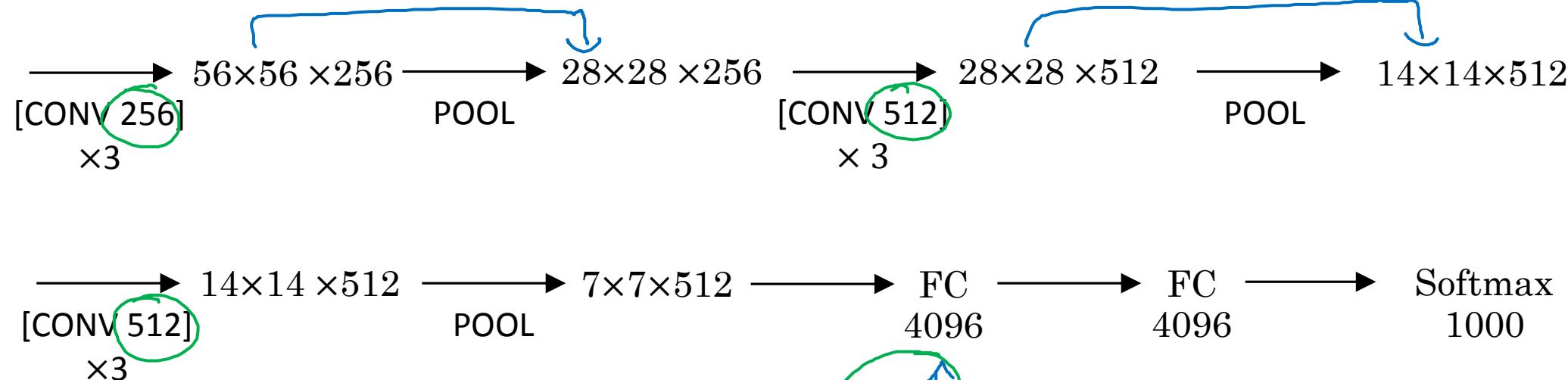
~60M Parameters

# VGG - 16

CONV =  $3 \times 3$  filter,  $s = 1$ , same



$224 \times 224$



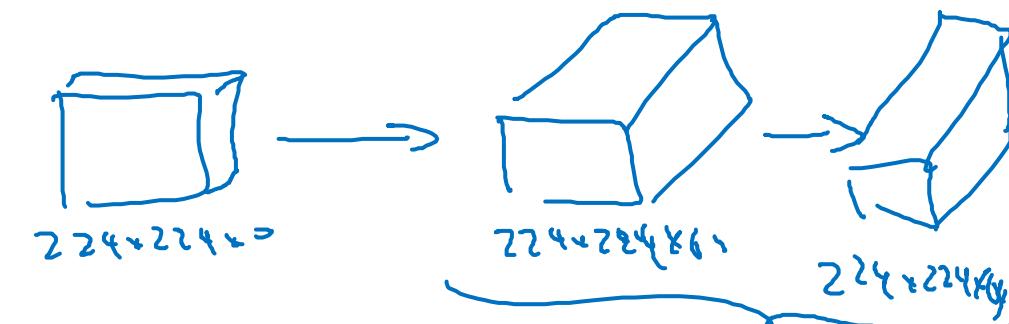
$n_H, n_W \downarrow$

$n_C \uparrow$

$\sim 38M$

# VGG-19

MAX-POOL =  $2 \times 2$ ,  $s = 2$



$56 \times 56 \times 128$

POOL

$224 \times 224 \times 64$



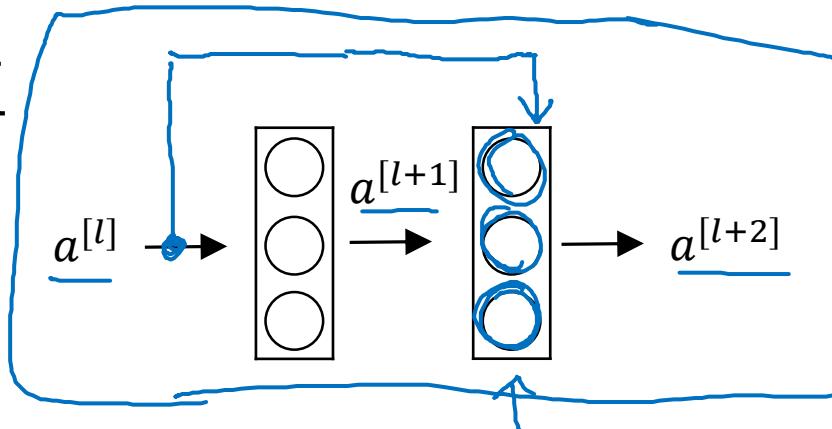
deeplearning.ai

## Case Studies

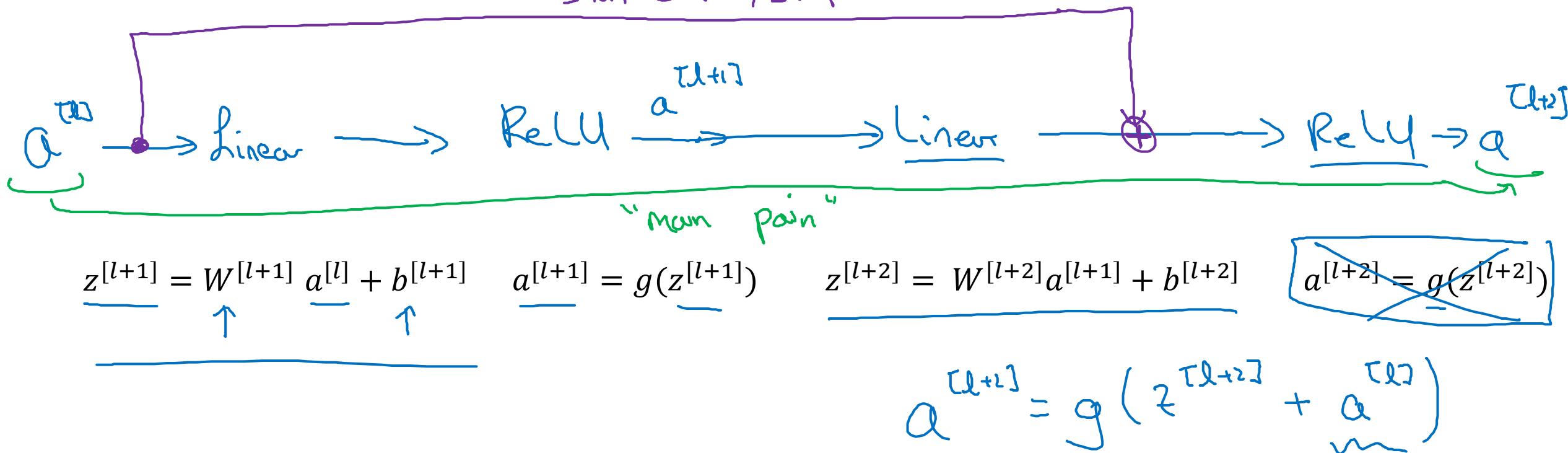
---

# Residual Networks (ResNets)

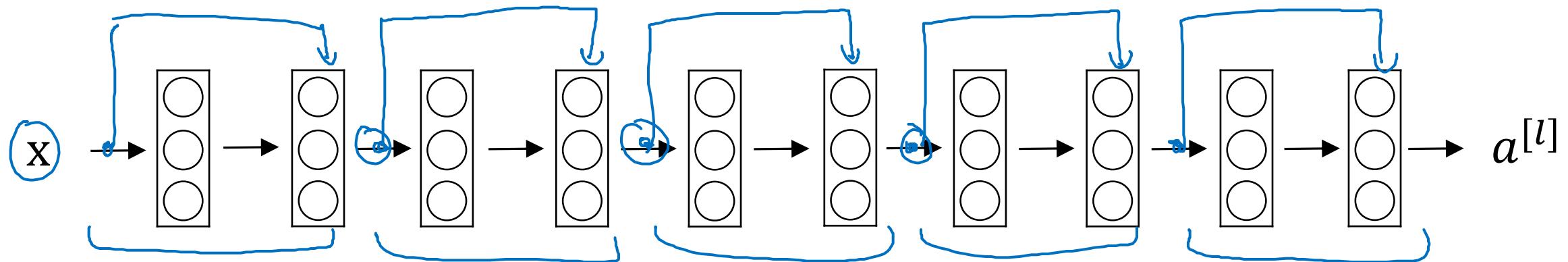
# Residual block



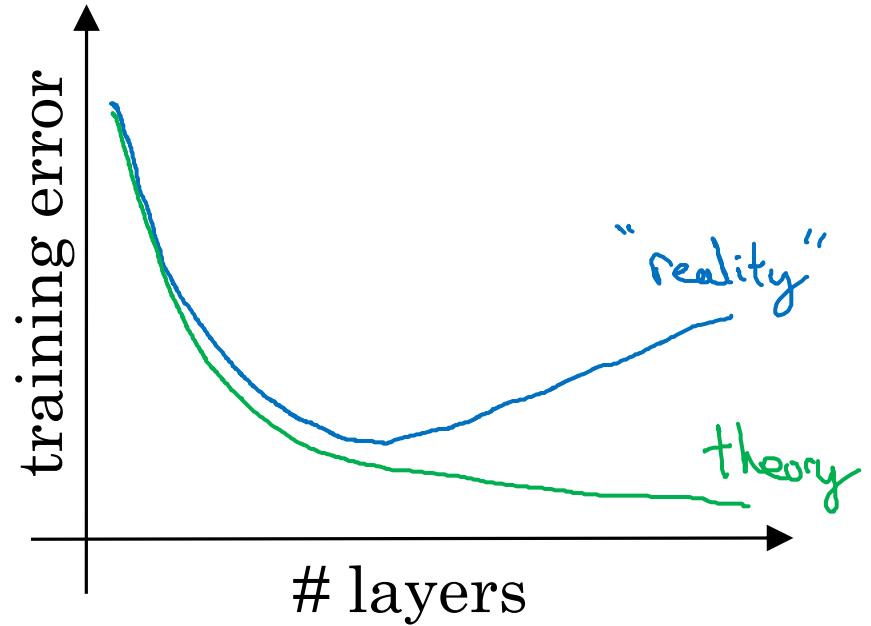
"Short cut" /skip connection



# Residual Network

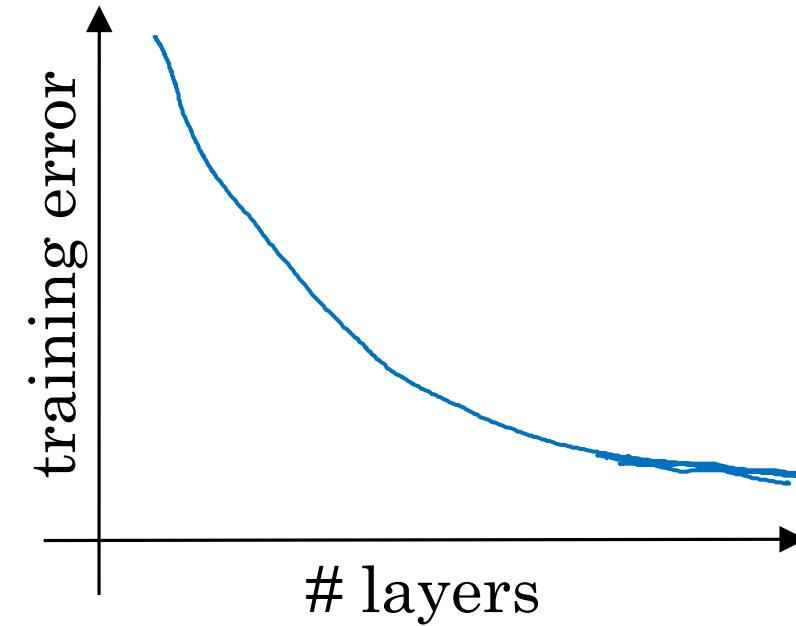


Plain



"Plain network"

ResNet





deeplearning.ai

## Case Studies

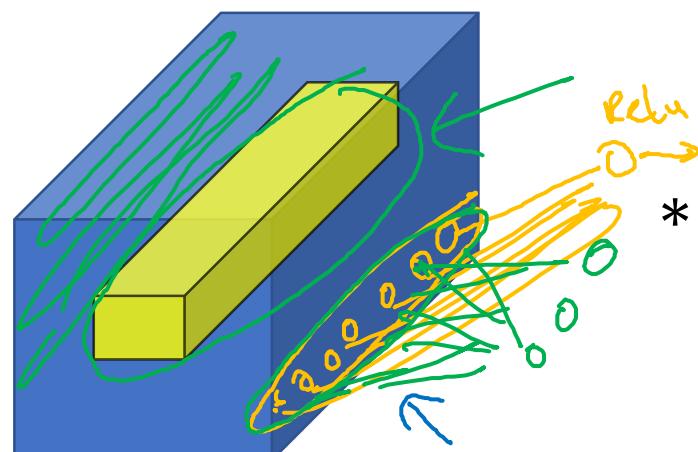
---

# Network in Network and $1 \times 1$ convolutions

# Why does a $1 \times 1$ convolution do?

1	2	3	6	5	8
3	5	5	1	3	4
2	1	3	4	9	3
4	7	8	5	7	9
1	5	3	7	4	8
5	4	9	8	3	5

$6 \times 6$



$6 \times 6 \times 32$

\*

2



=

32 → # filters.

$n_c^{[l+1]}$

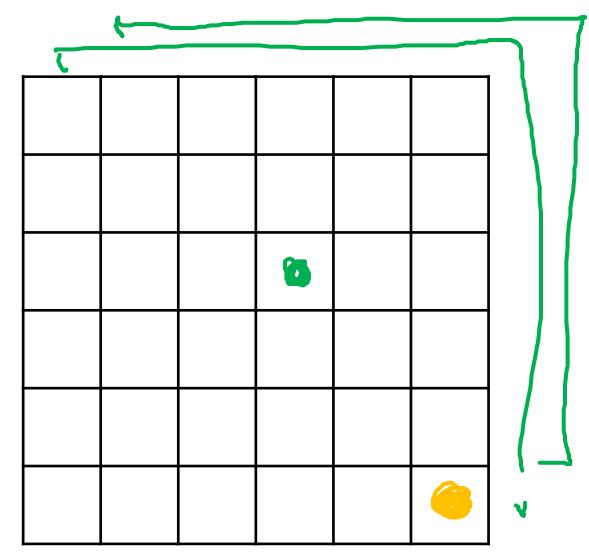
=

ReLU

Network  $\hookrightarrow$  Network

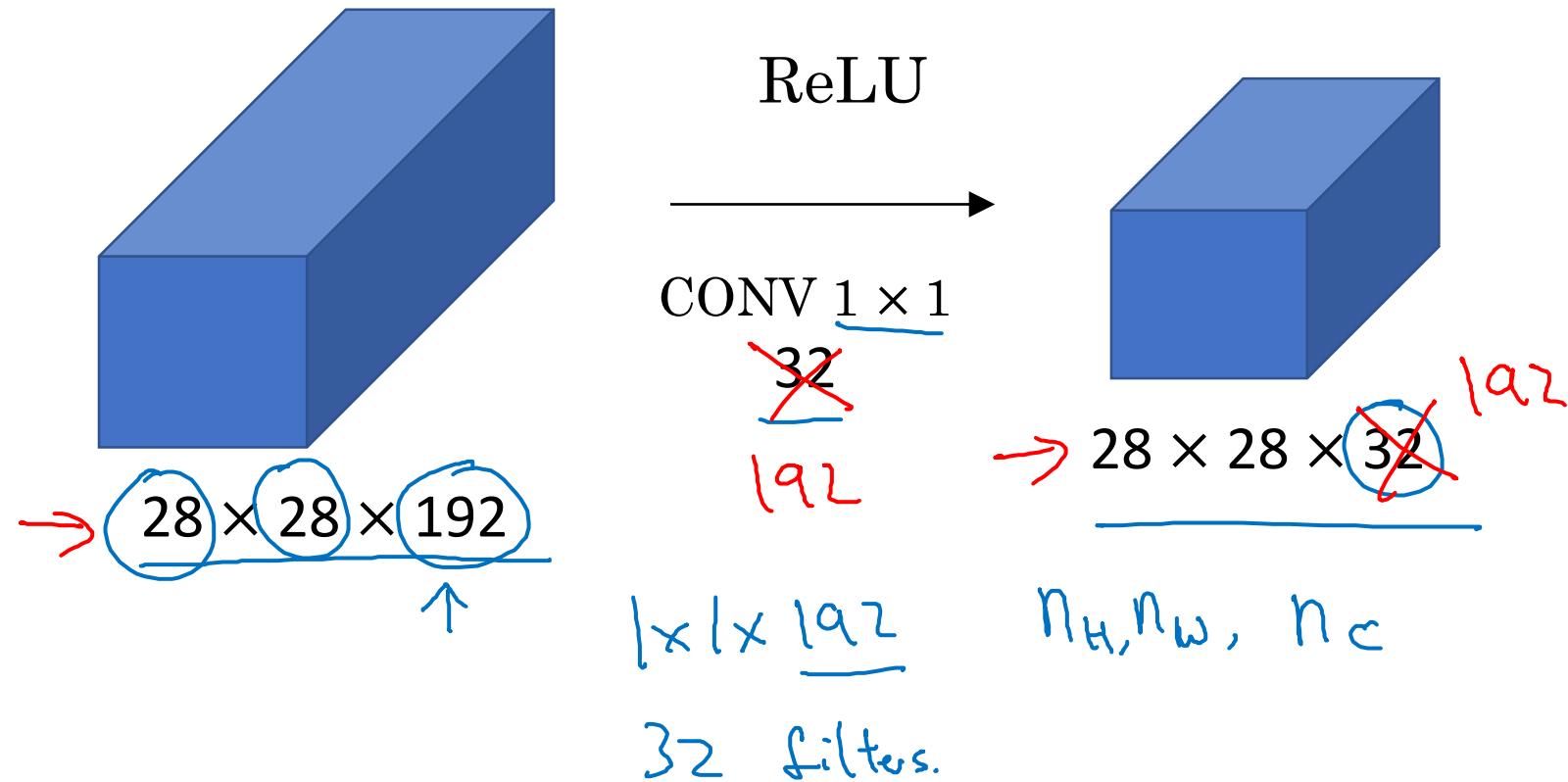
$1 \times 1 \times 32$

2	4	6	...



$6 \times 6 \times \# \text{ filters}$

# Using $1 \times 1$ convolutions





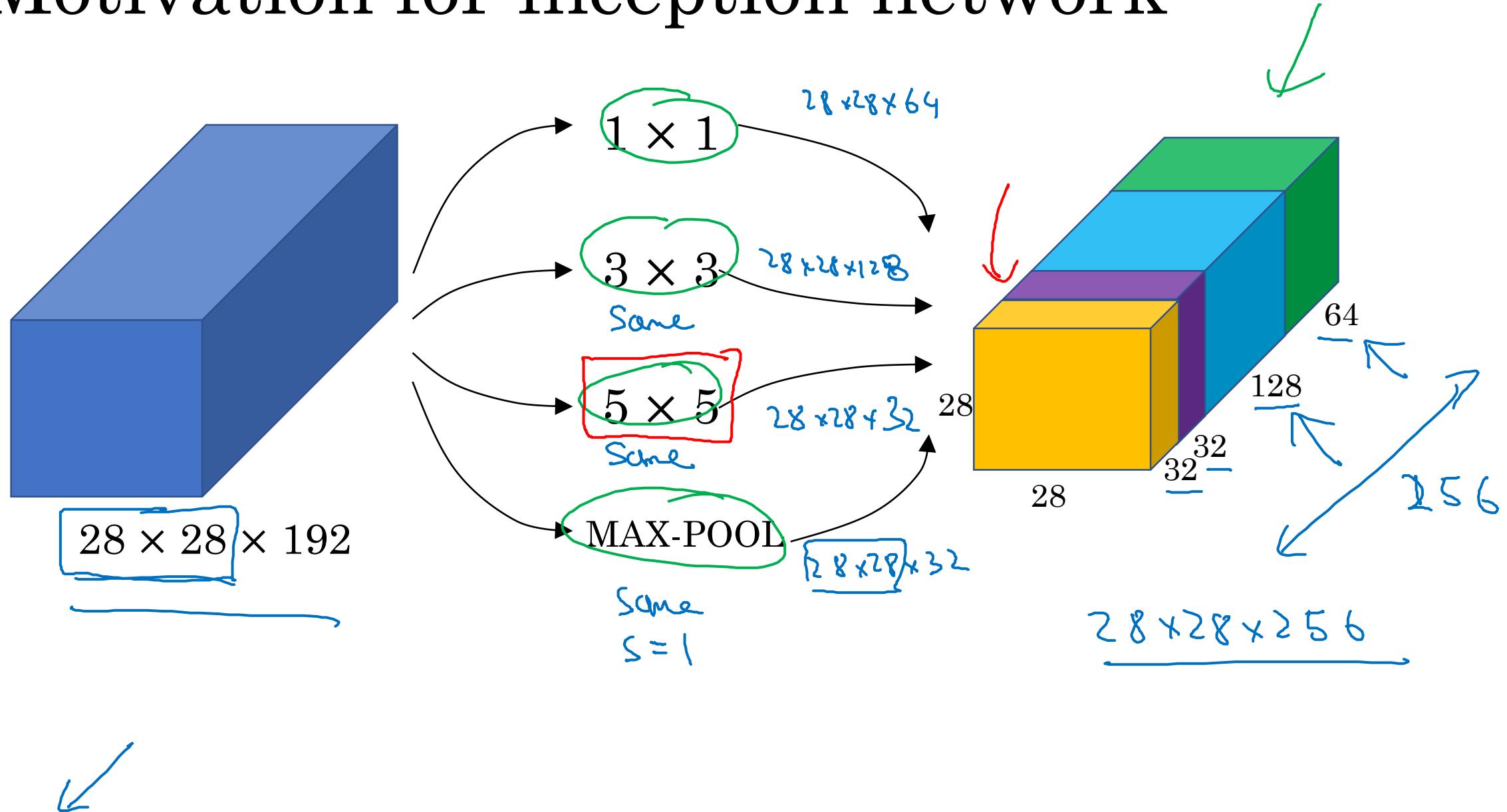
deeplearning.ai

# Case Studies

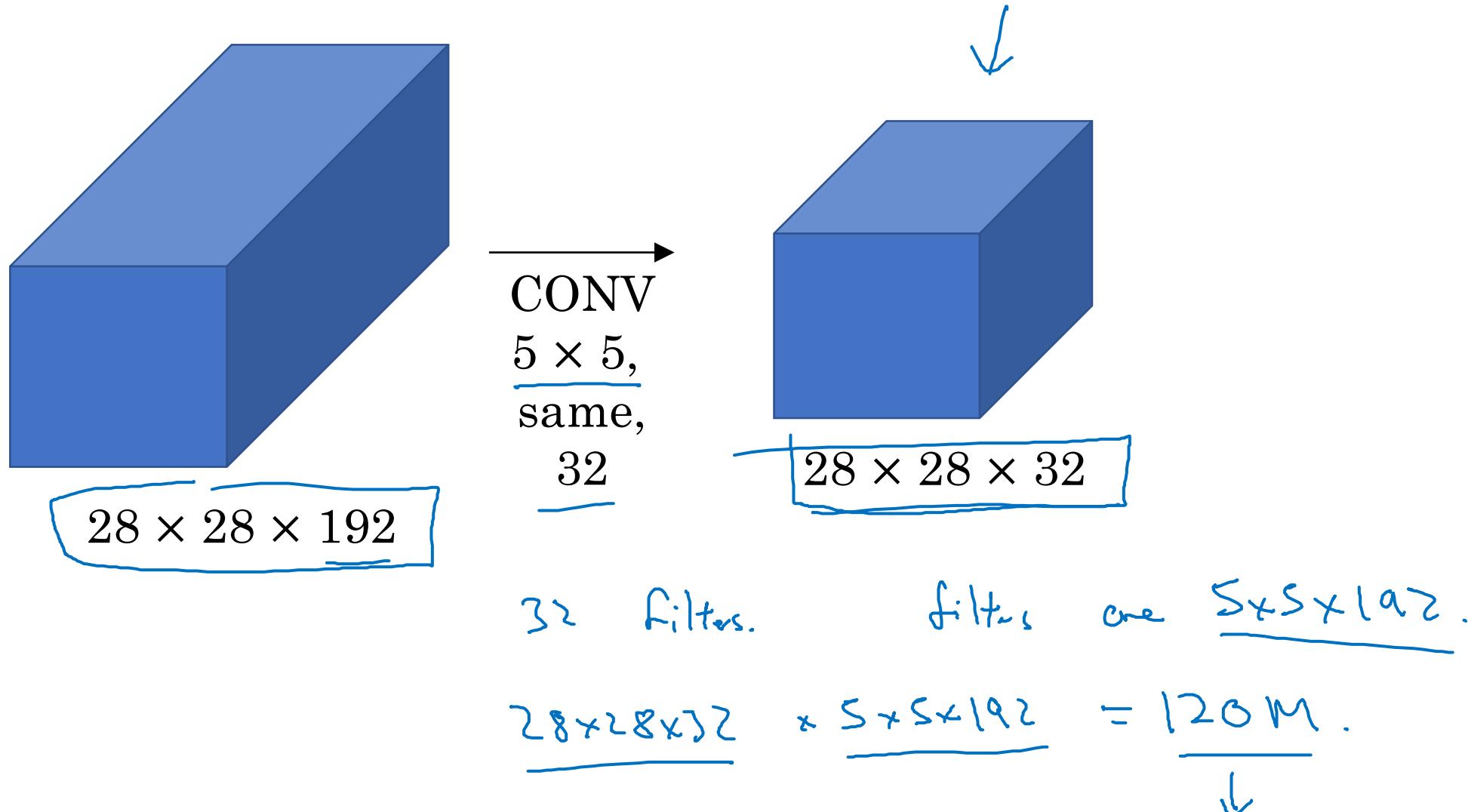
---

## Inception network motivation

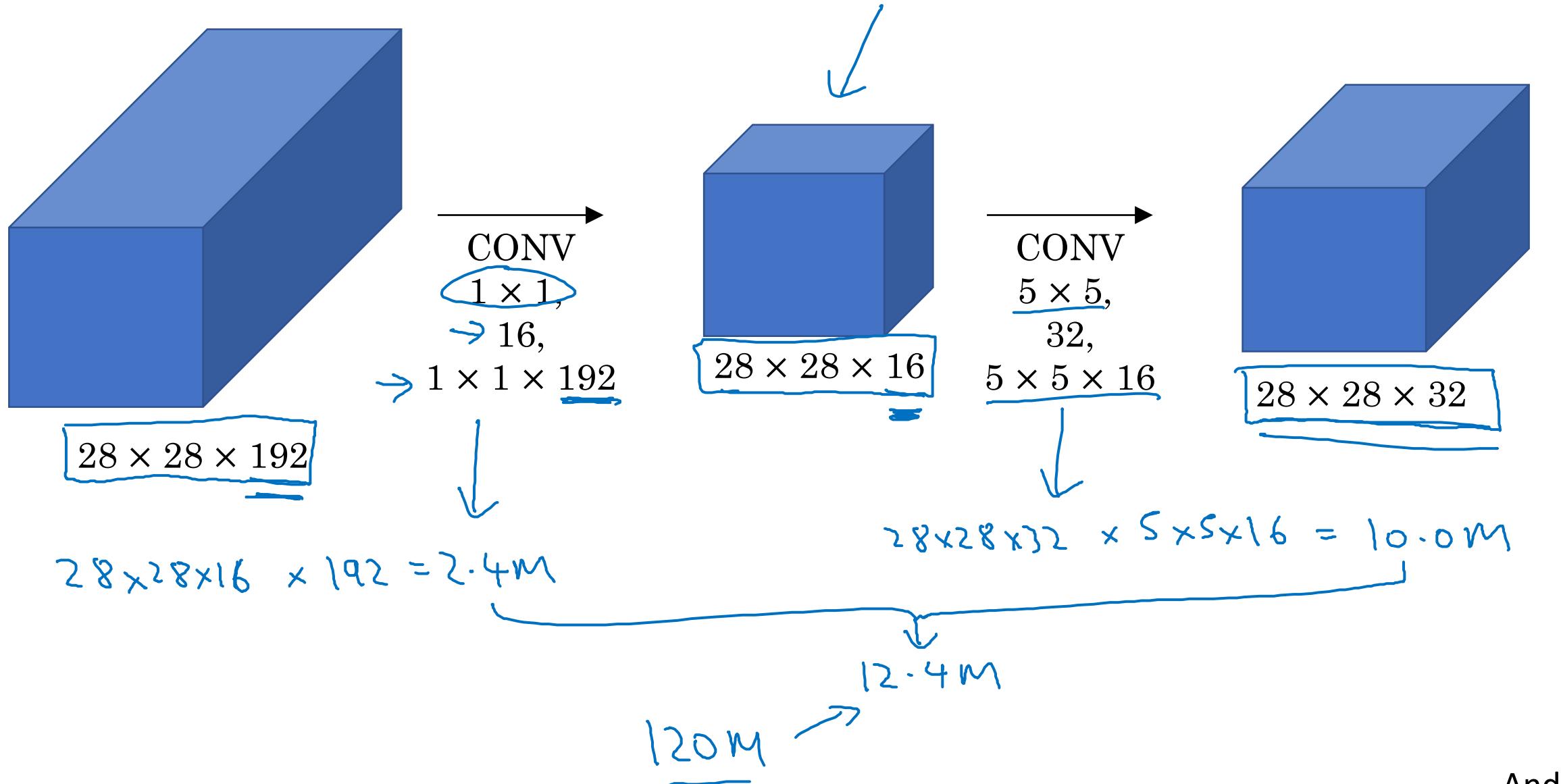
# Motivation for inception network



# The problem of computational cost



# Using $1 \times 1$ convolution





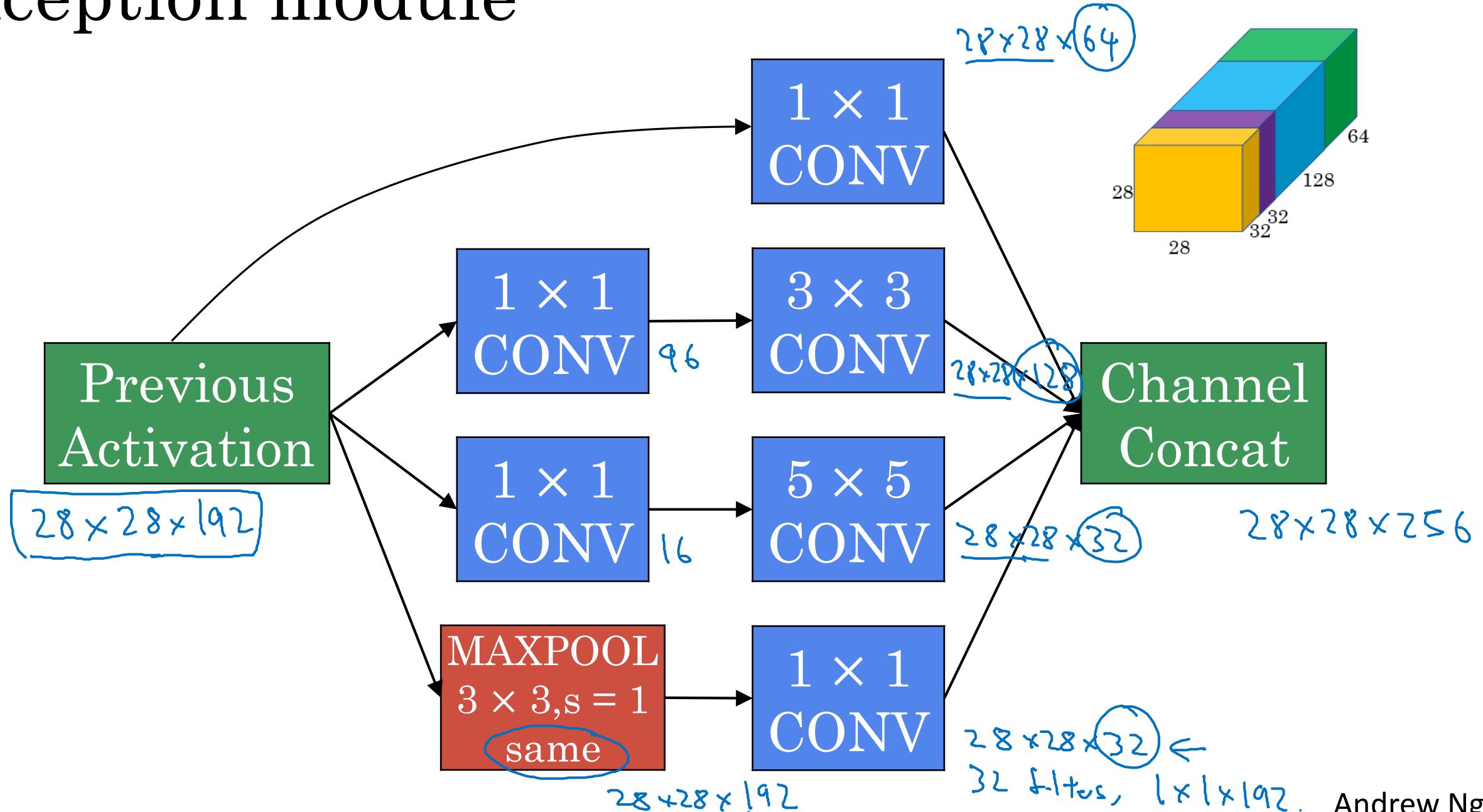
deeplearning.ai

# Case Studies

---

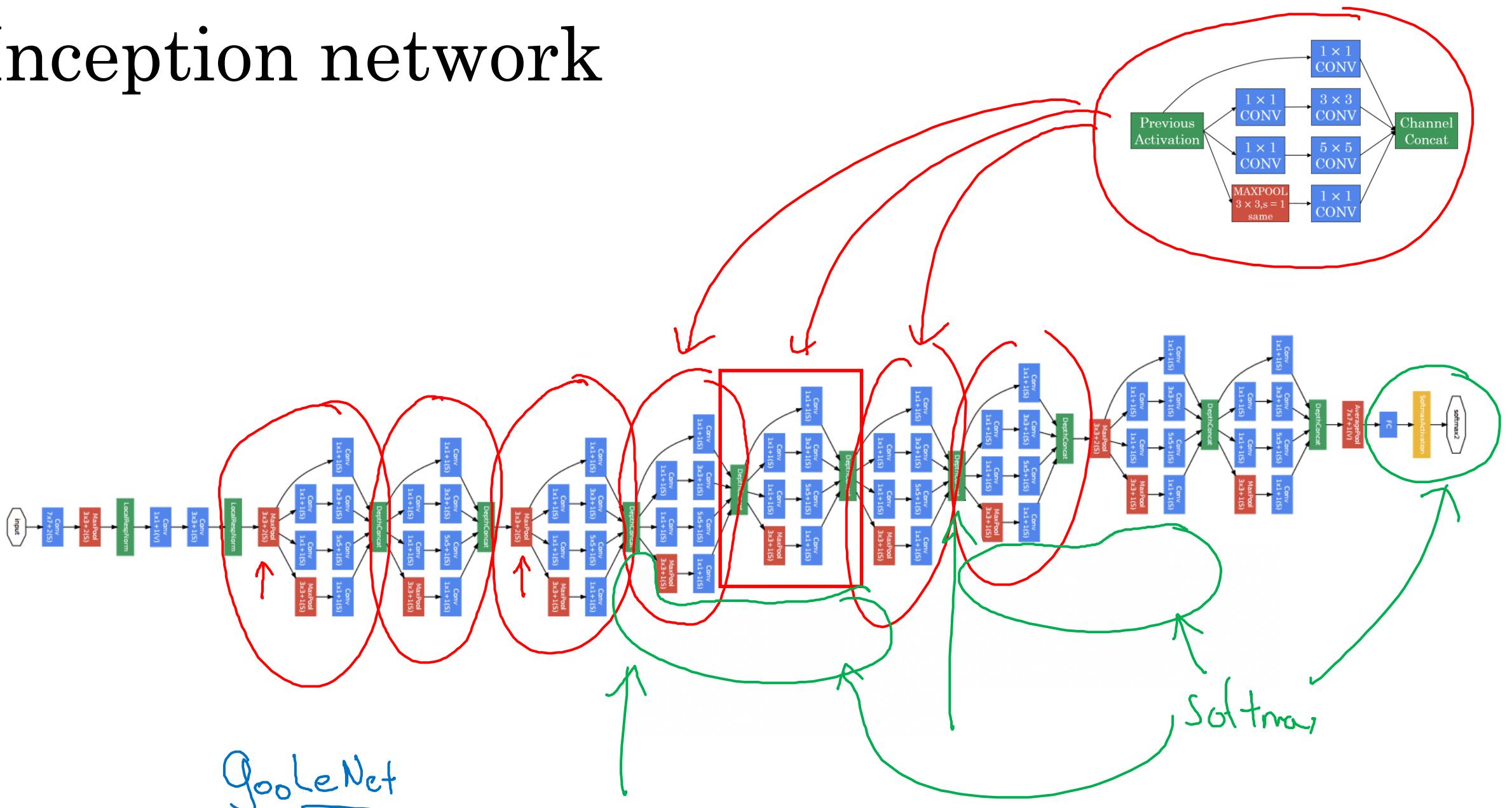
## Inception network

# Inception module



Andrew Ng

# Inception network







deeplearning.ai

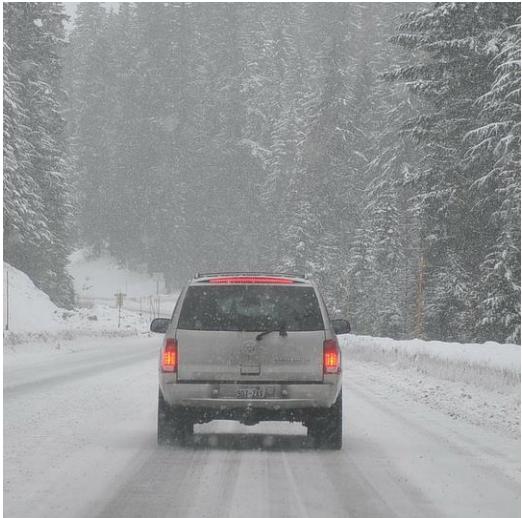
# Object Detection

---

## Object localization

# What are localization and detection?

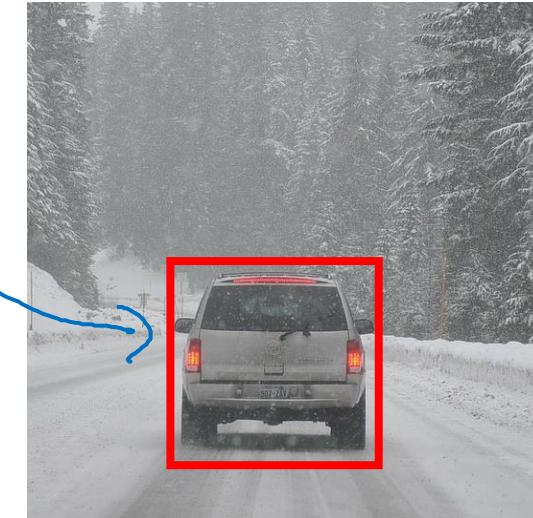
Image classification



"Car"  
|  
object

A blue bracket groups the word "Car" above the image with the word "object" below it, connected by a vertical line.

Classification with  
localization



"Car"

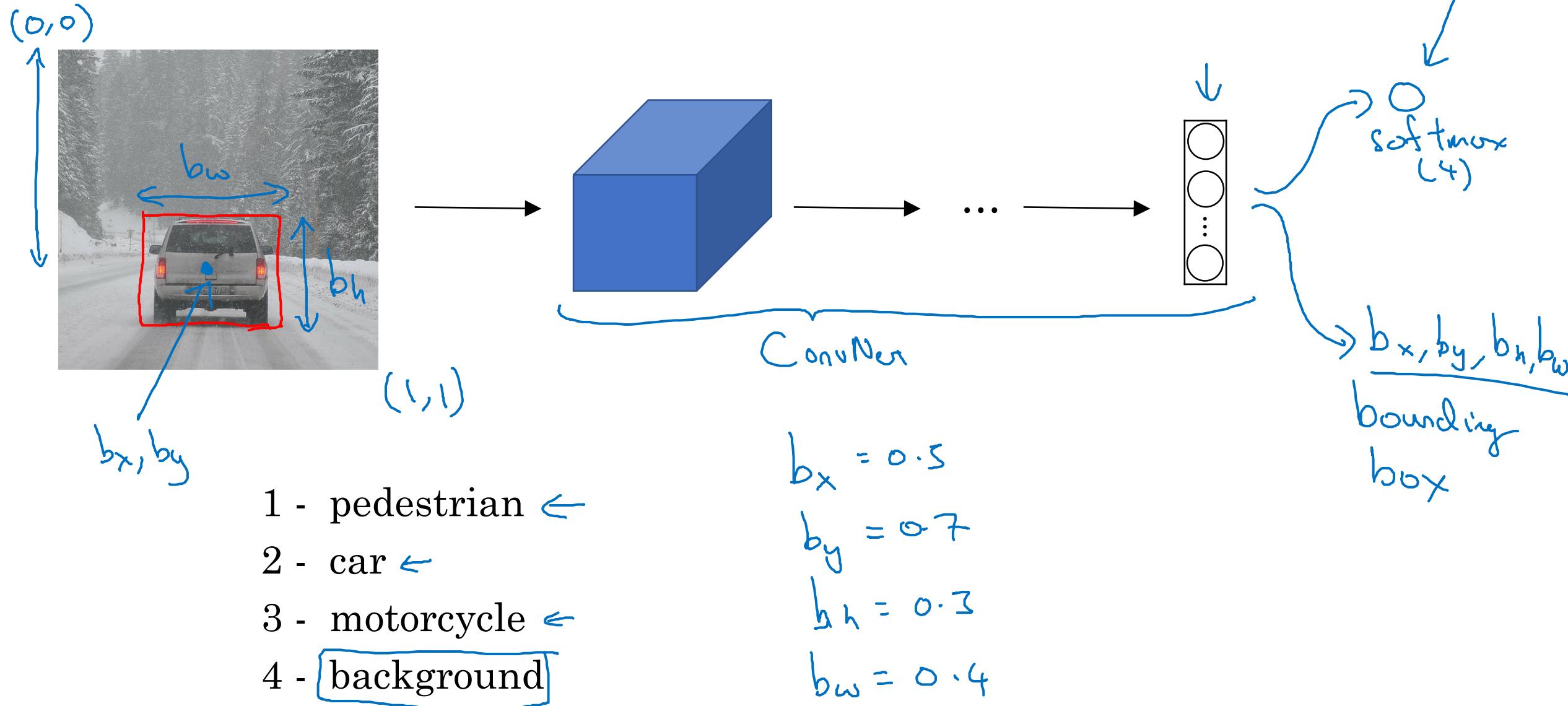
Detection



multiple  
objects

A blue bracket groups the words "multiple objects" below the image.

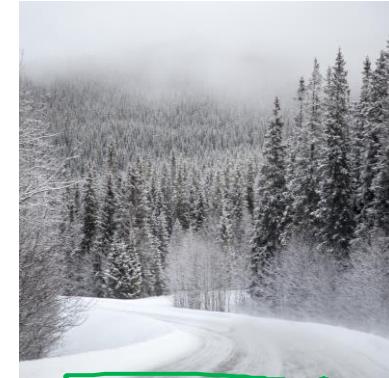
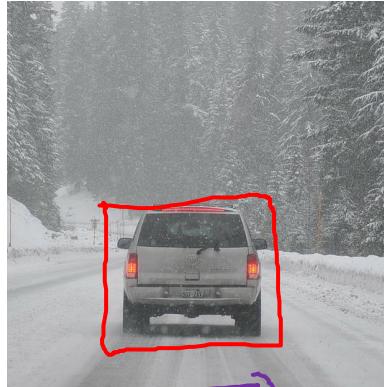
# Classification with localization



# Defining the target label $y$

- 1 - pedestrian
- 2 - car ←
- 3 - motorcycle
- 4 - background ←

Need to output  $b_x, b_y, b_h, b_w$ , class label (1-4)

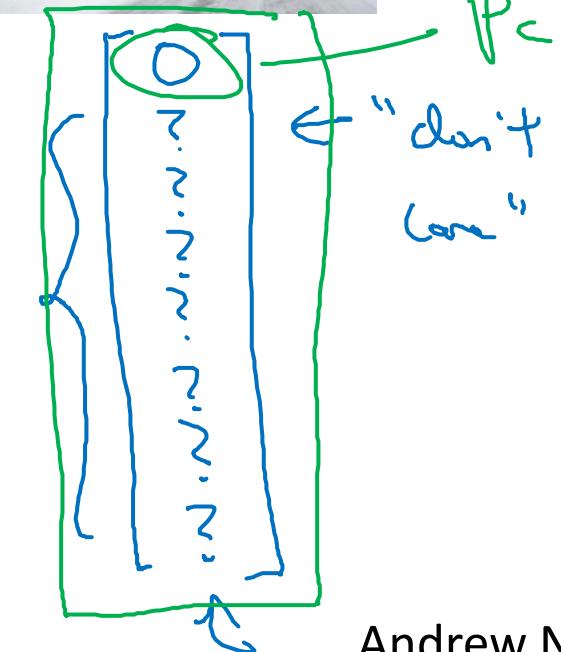
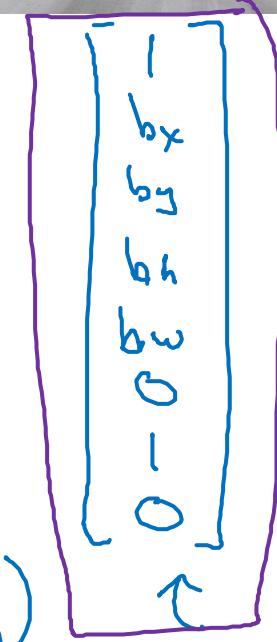


$x =$

$$L(\hat{y}, y) = \begin{cases} (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 \\ + \dots + (\hat{y}_8 - y_8)^2 & \text{if } \underline{y_1 = 1} \\ (\hat{y}_1 - y_1)^2 & \text{if } \underline{y_1 = 0} \end{cases}$$

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} \quad \rightarrow \quad \left\{ \begin{array}{l} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{array} \right\} \text{ is there any object?}$$

$(x, y)$



Andrew Ng



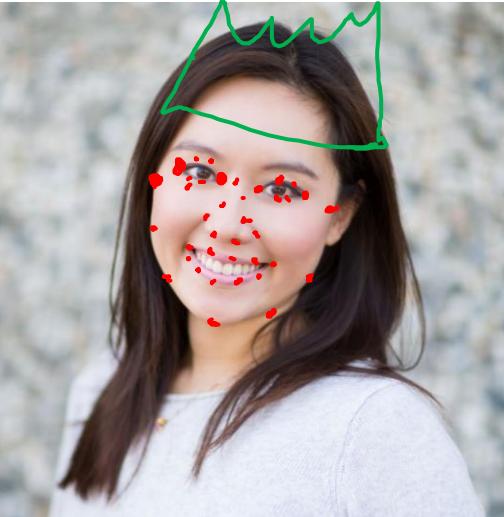
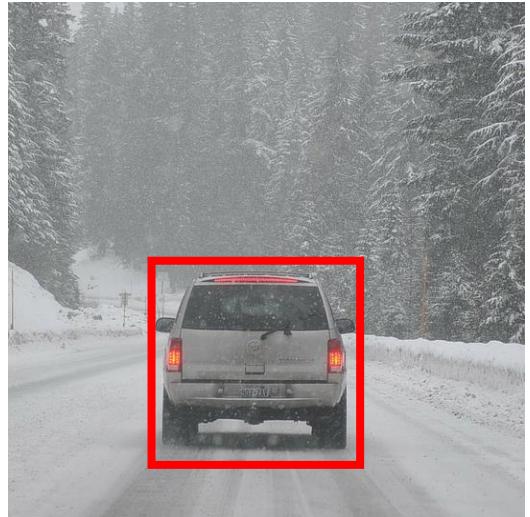
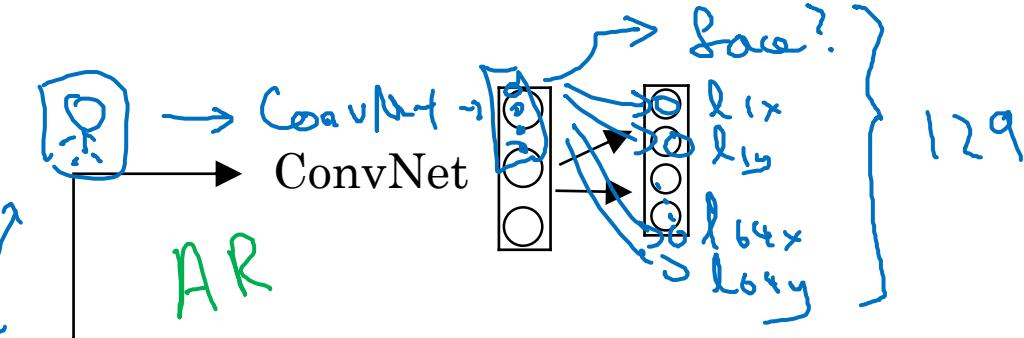
deeplearning.ai

# Object Detection

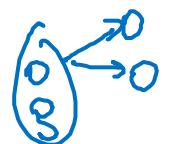
---

## Landmark detection

# Landmark detection



$b_x, b_y, b_h, b_w$



$l_{1x}, l_{1y},$   
 $l_{2x}, l_{2y},$   
 $l_{3x}, l_{3y},$   
 $l_{4x}, l_{4y},$   
:  
 $l_{64x}, l_{64y}$

$x, y$

$l_{1x}, l_{1y},$   
:  
 $l_{32x}, l_{32y}$



deeplearning.ai

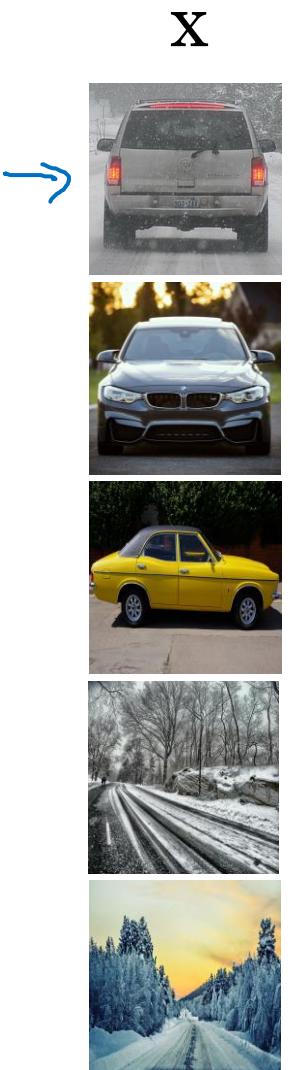
# Object Detection

---

## Object detection

# Car detection example

Training set:



y

1

1

1

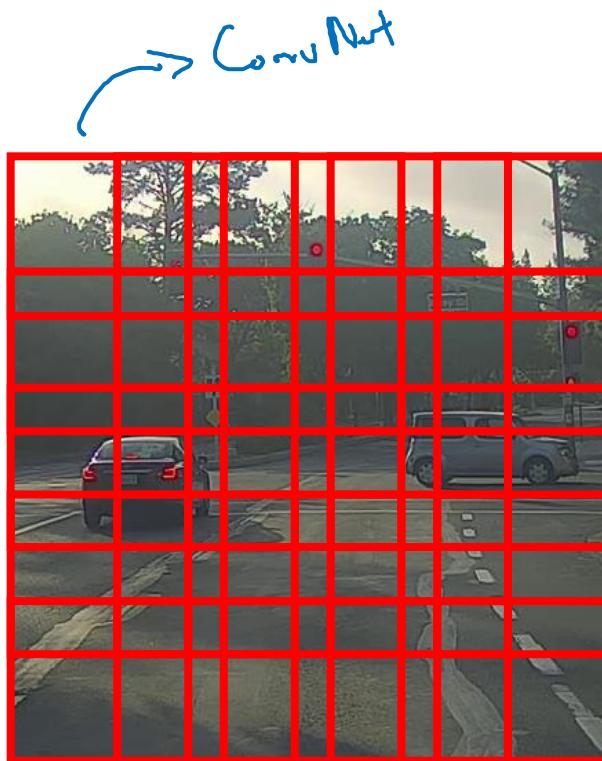
0

0



→ ConvNet → y

# Sliding windows detection



Computation cost



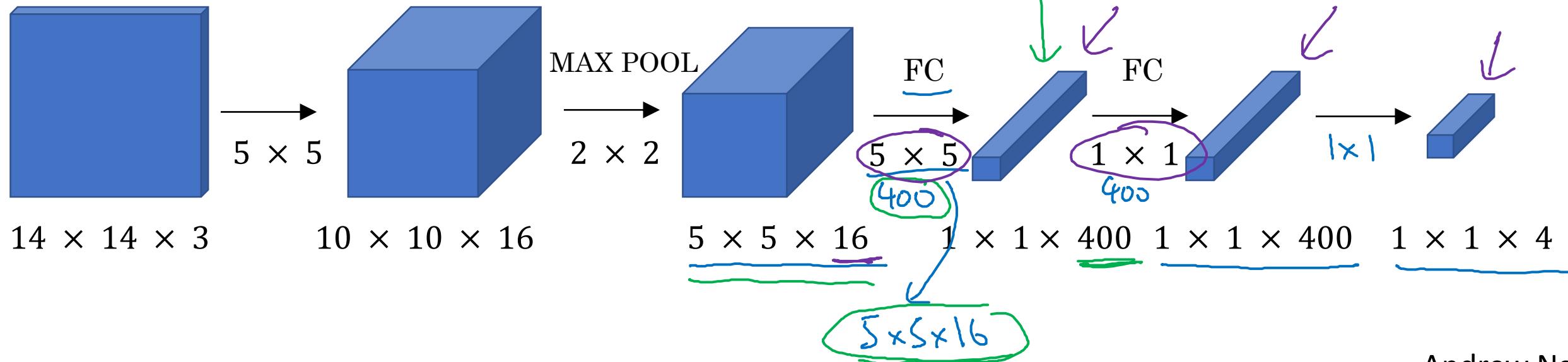
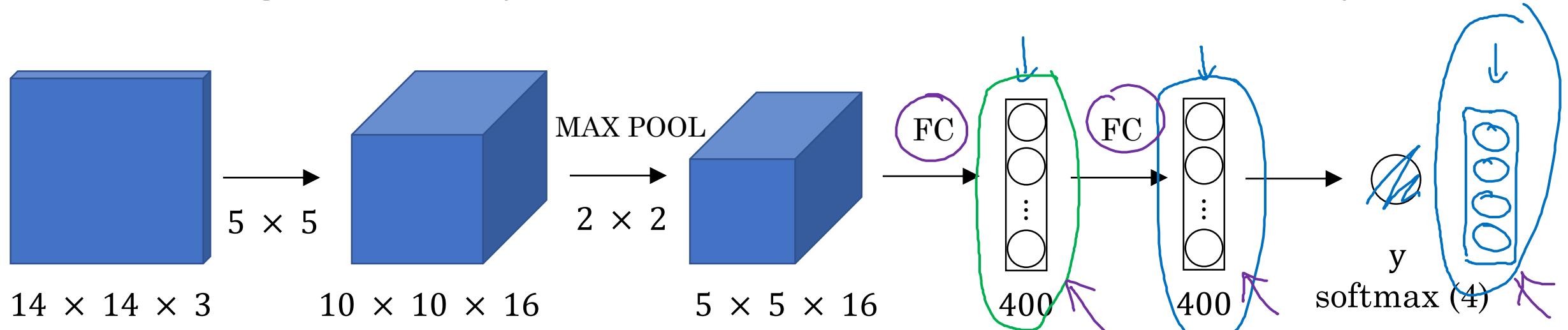
deeplearning.ai

# Object Detection

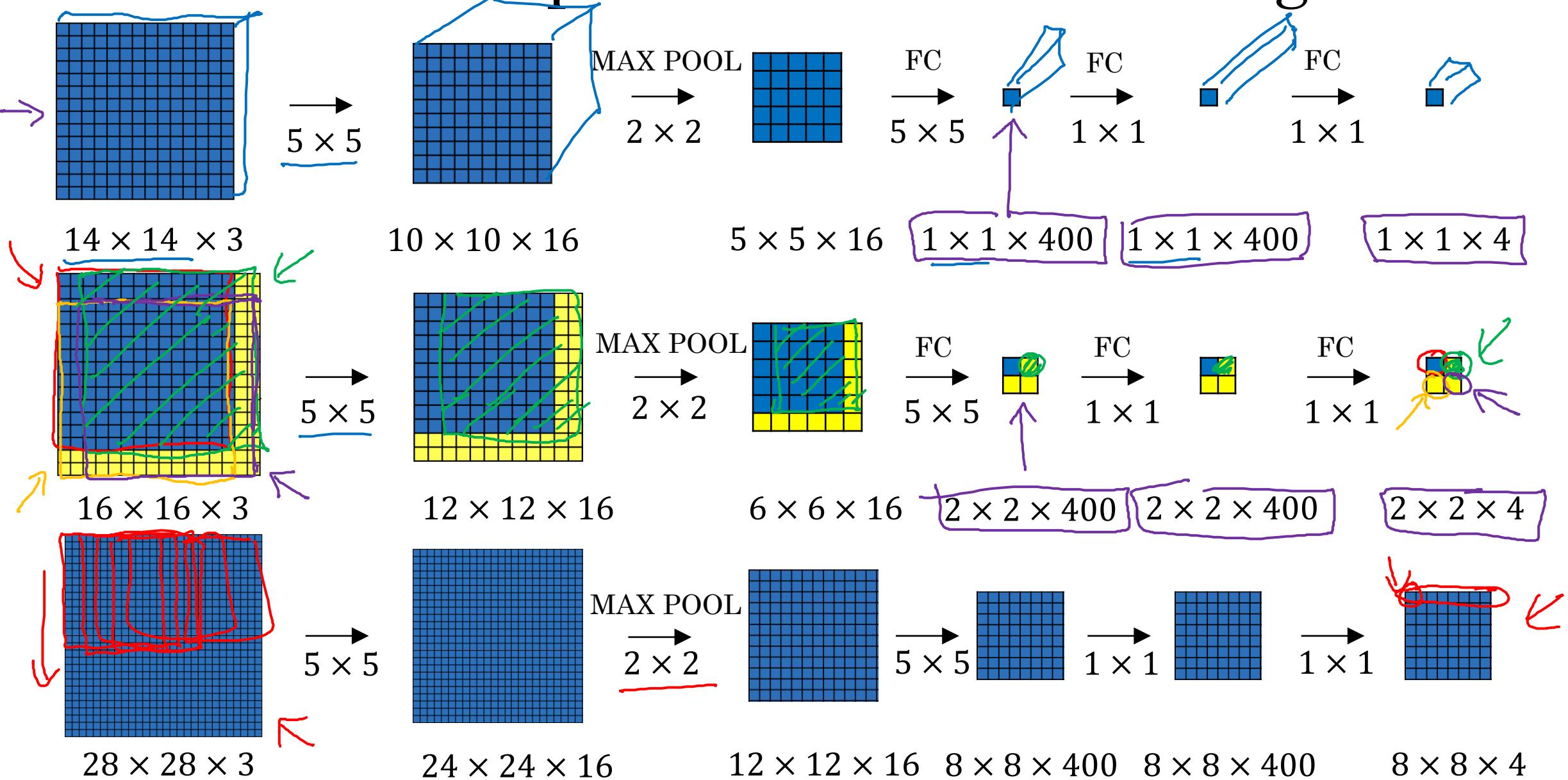
---

## Convolutional implementation of sliding windows

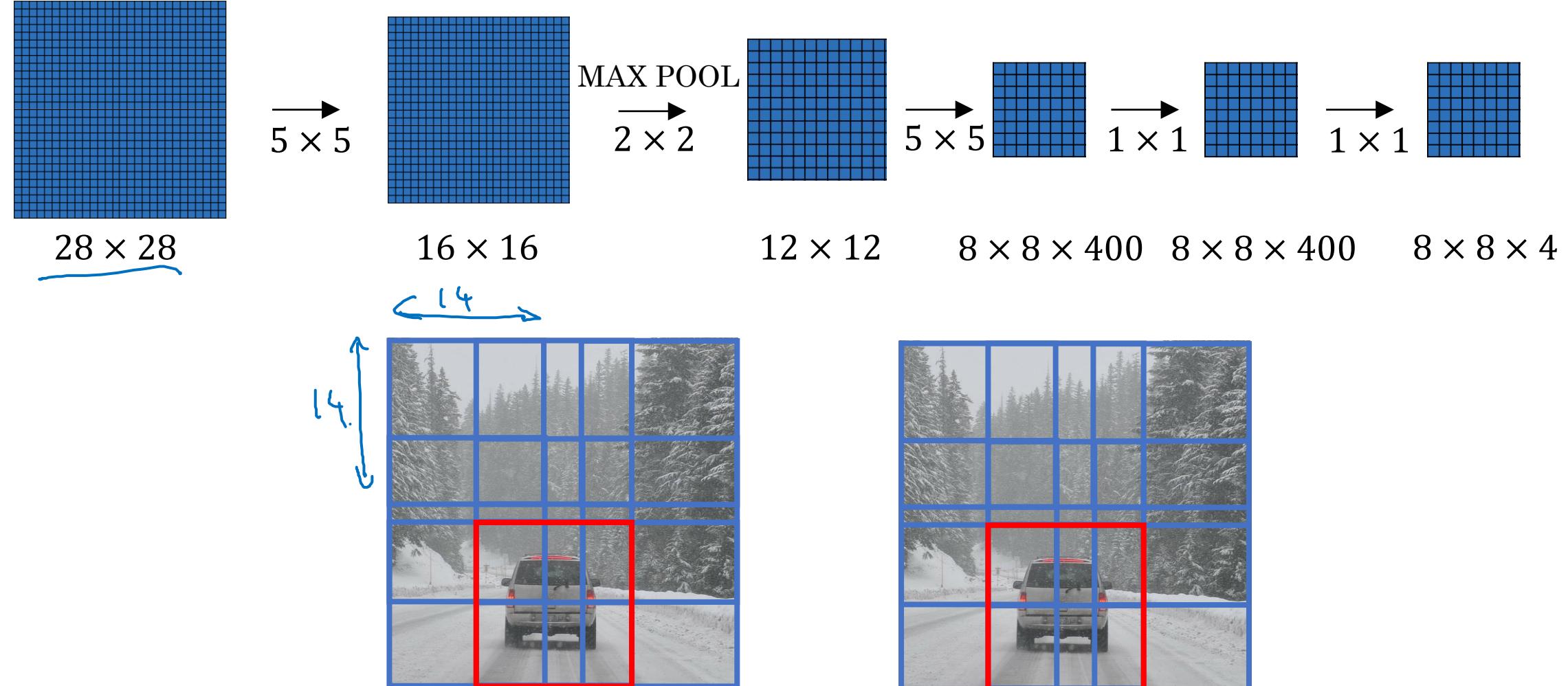
# Turning FC layer into convolutional layers



# Convolution implementation of sliding windows



# Convolution implementation of sliding windows





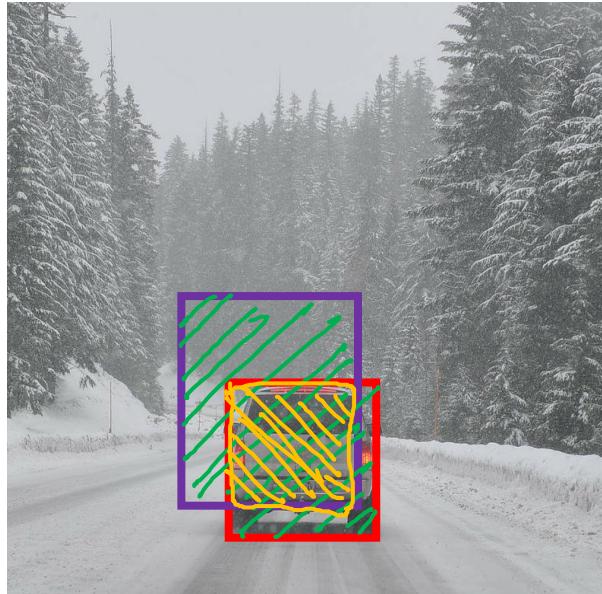
deeplearning.ai

# Object Detection

---

Intersection  
over union

# Evaluating object localization



Intersection over Union (IoU)

$$= \frac{\text{Size of intersection}}{\text{Size of union}}$$
A diagram illustrating the calculation of IoU. It shows two overlapping rectangles: a yellow one at the top right and a green one below it. The overlapping area is shaded with diagonal lines, representing the intersection. The non-overlapping parts of both rectangles are also shaded with diagonal lines, representing the union.

“Correct” if  $\text{IoU} \geq 0.5$

0.6

More generally, IoU is a measure of the overlap between two bounding boxes.



deeplearning.ai

# Object Detection

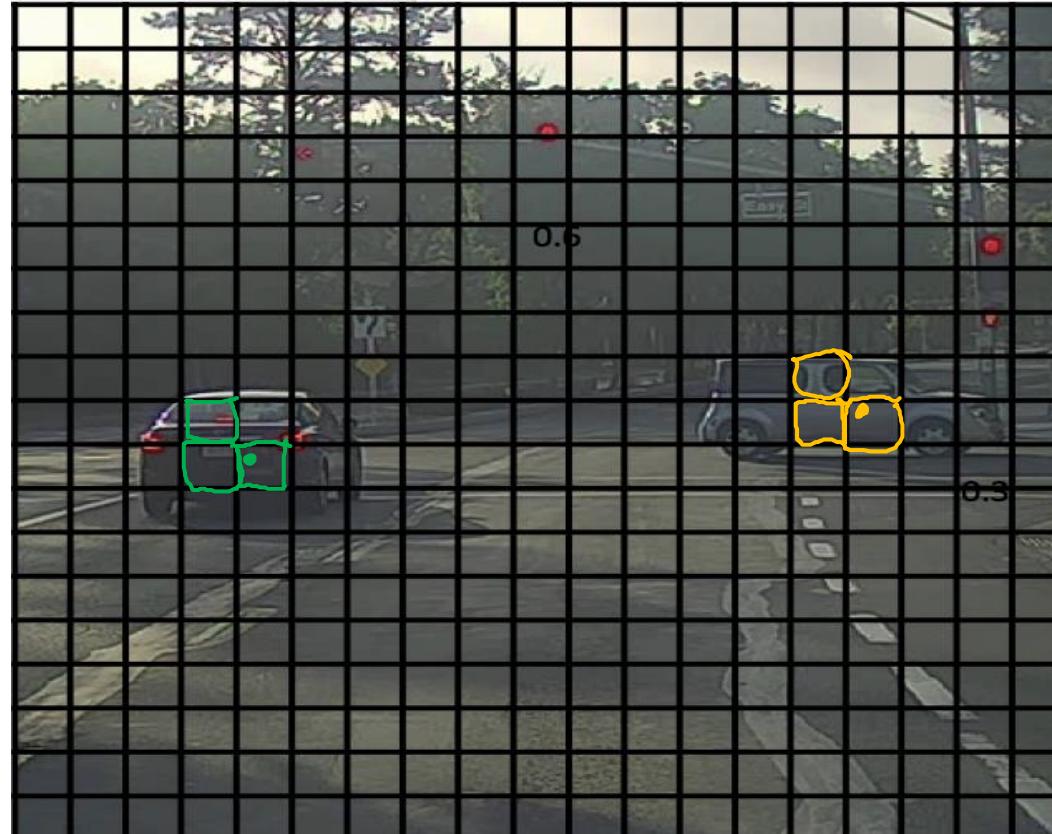
---

Non-max  
suppression

# Non-max suppression example

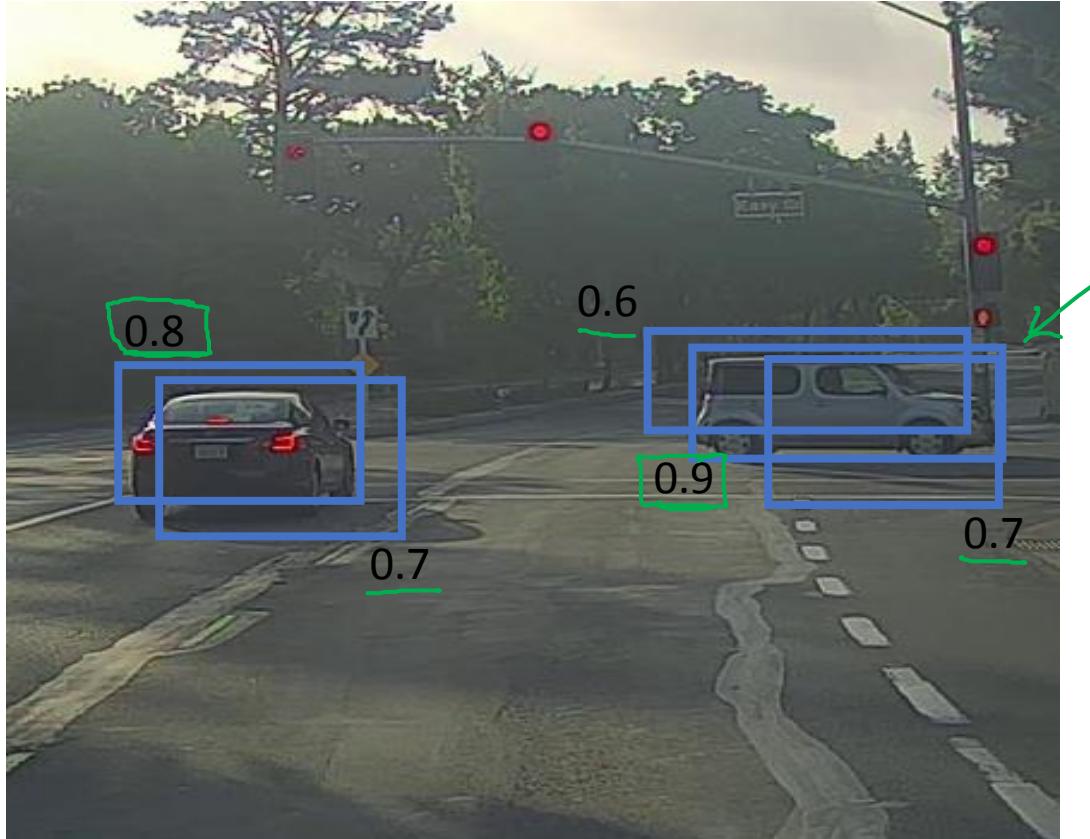


# Non-max suppression example

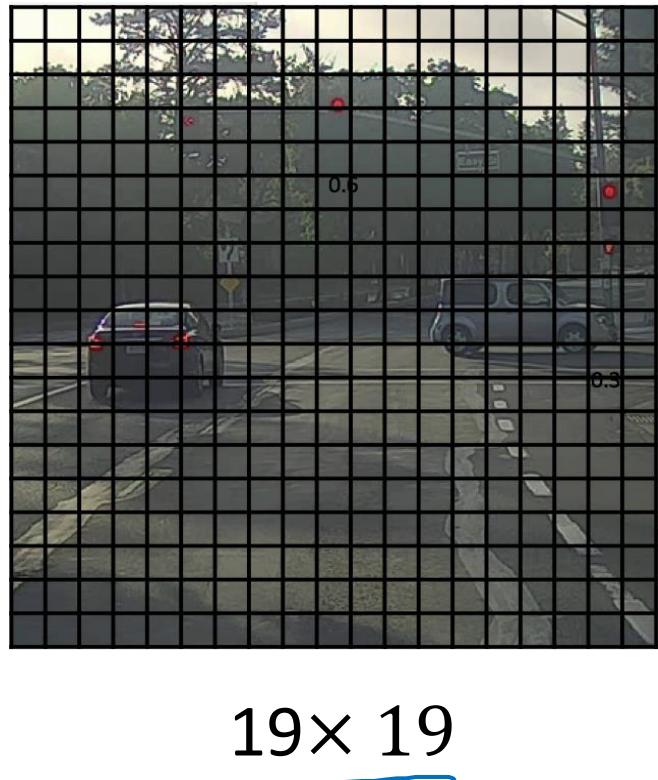


19x19

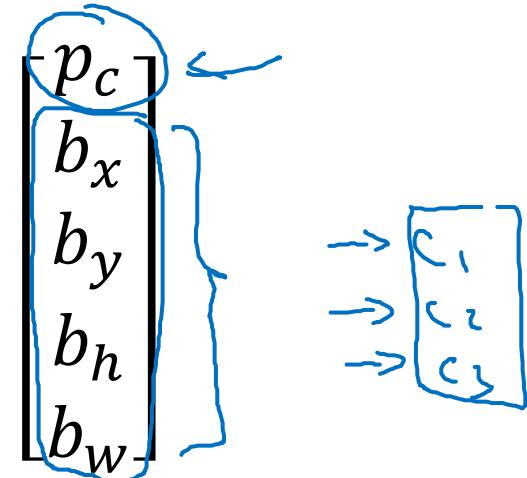
# Non-max suppression example



# Non-max suppression algorithm



Each output prediction is:



Discard all boxes with  $p_c \leq 0.6$

→ While there are any remaining boxes:

- Pick the box with the largest  $p_c$ . Output that as a prediction.
- Discard any remaining box with  $\text{IoU} \geq 0.5$  with the box output in the previous step



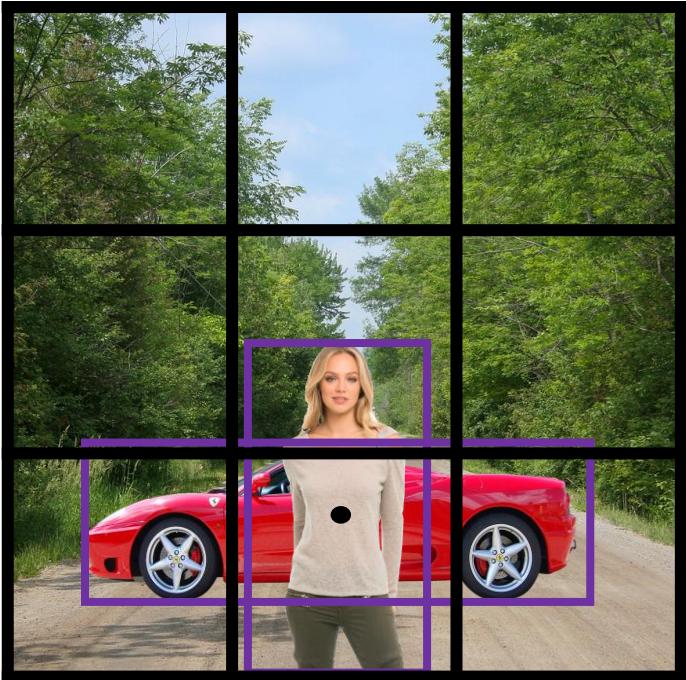
deeplearning.ai

# Object Detection

---

## Anchor boxes

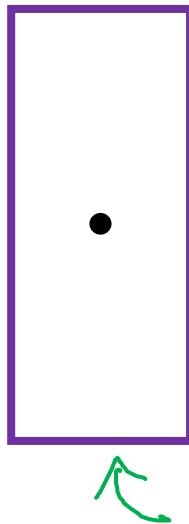
# Overlapping objects:



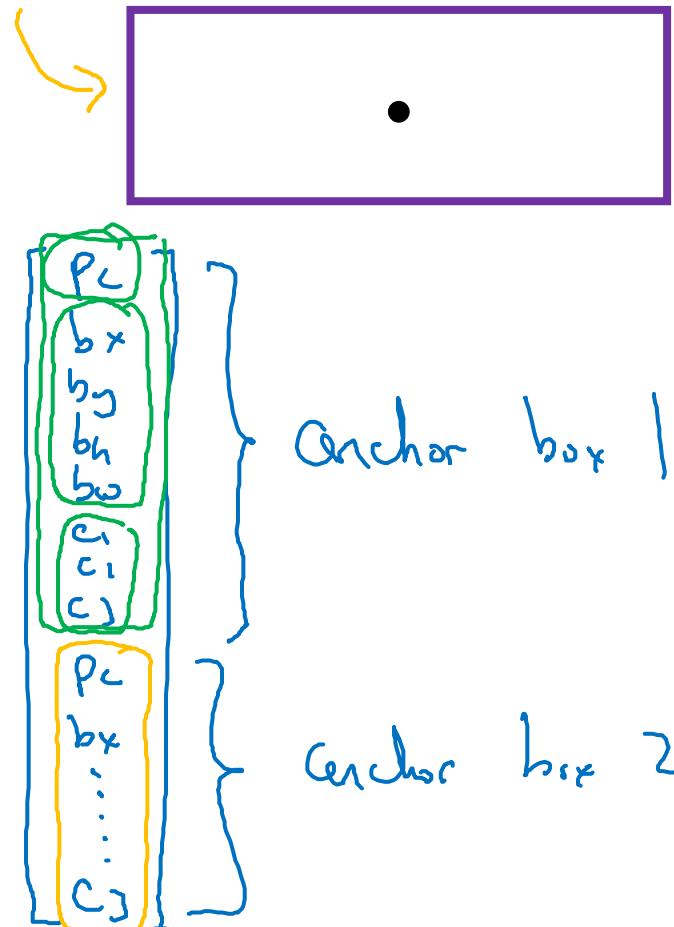
$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

A green bracket above the first four columns indicates they are grouped together. A blue bracket below the last four columns indicates they are grouped together.

Anchor box 1:



Anchor box 2:



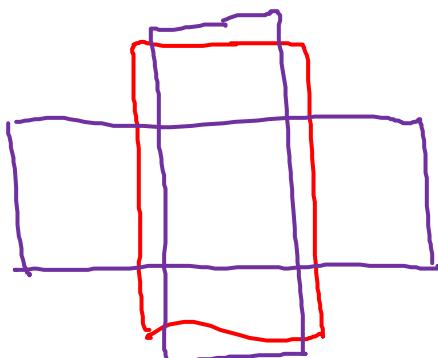
# Anchor box algorithm

Previously:

Each object in training image is assigned to grid cell that contains that object's midpoint.

Output y:

$3 \times 3 \times 8$



With two anchor boxes:

Each object in training image is assigned to grid cell that contains object's midpoint and anchor box for the grid cell with highest IoU.

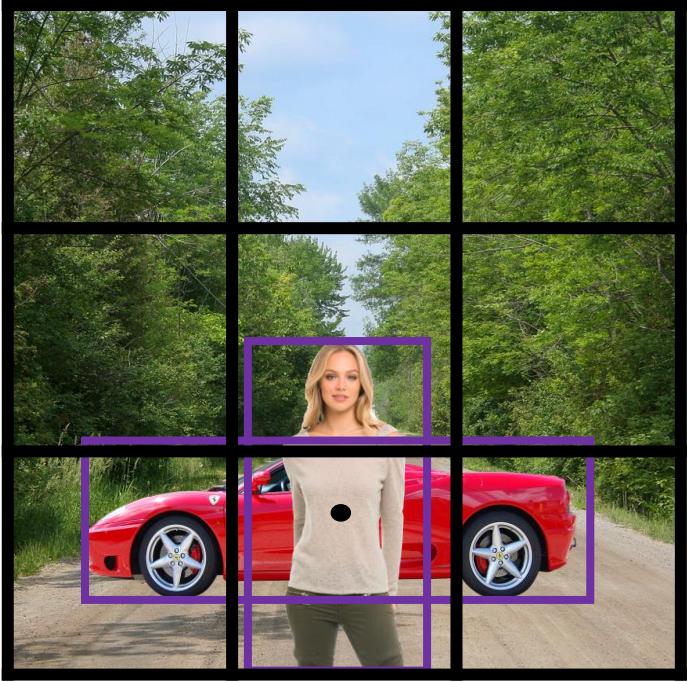
(grid cell, anchor box)

Output y:

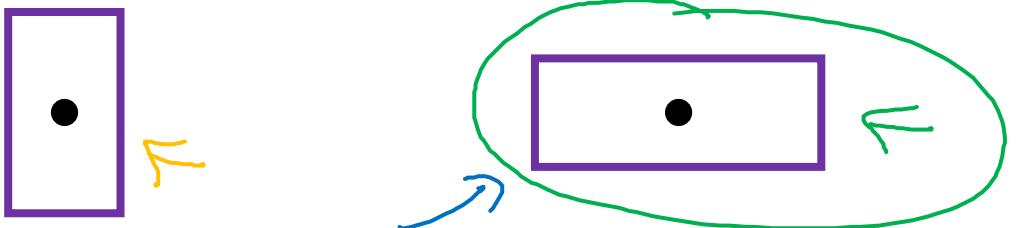
$3 \times 3 \times 16$

$3 \times 3 \times 2 \times 8$

# Anchor box example

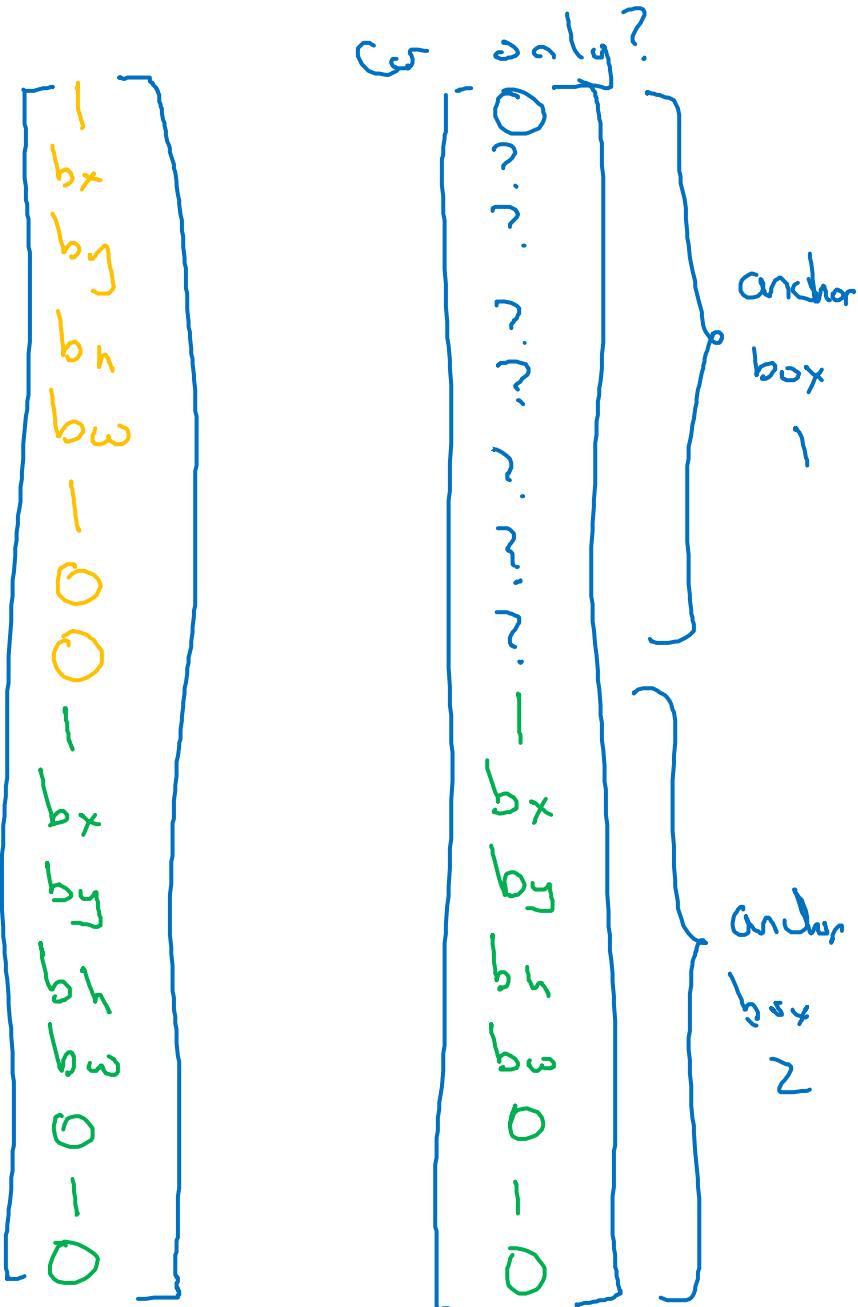


Anchor box 1:    Anchor box 2:



$$y =$$

$$\begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$





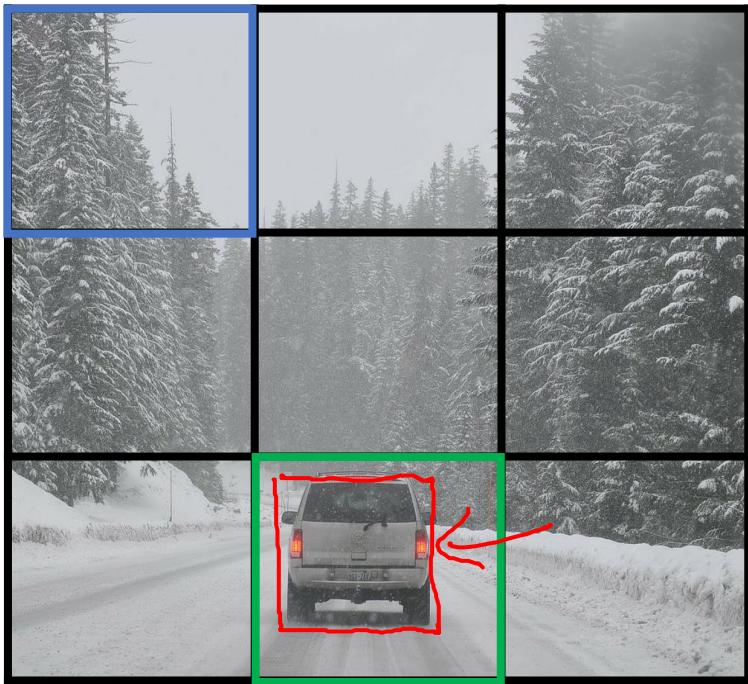
deeplearning.ai

# Object Detection

---

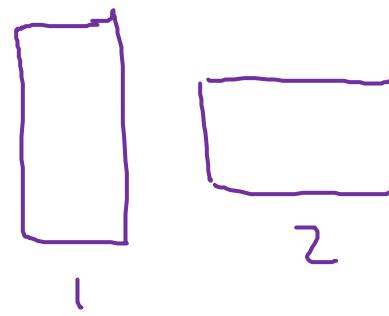
Putting it together:  
YOLO algorithm

# Training



- 1 - pedestrian
- 2 - car ←
- 3 - motorcycle

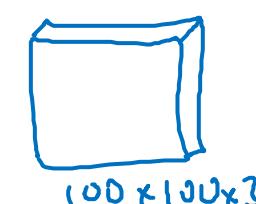
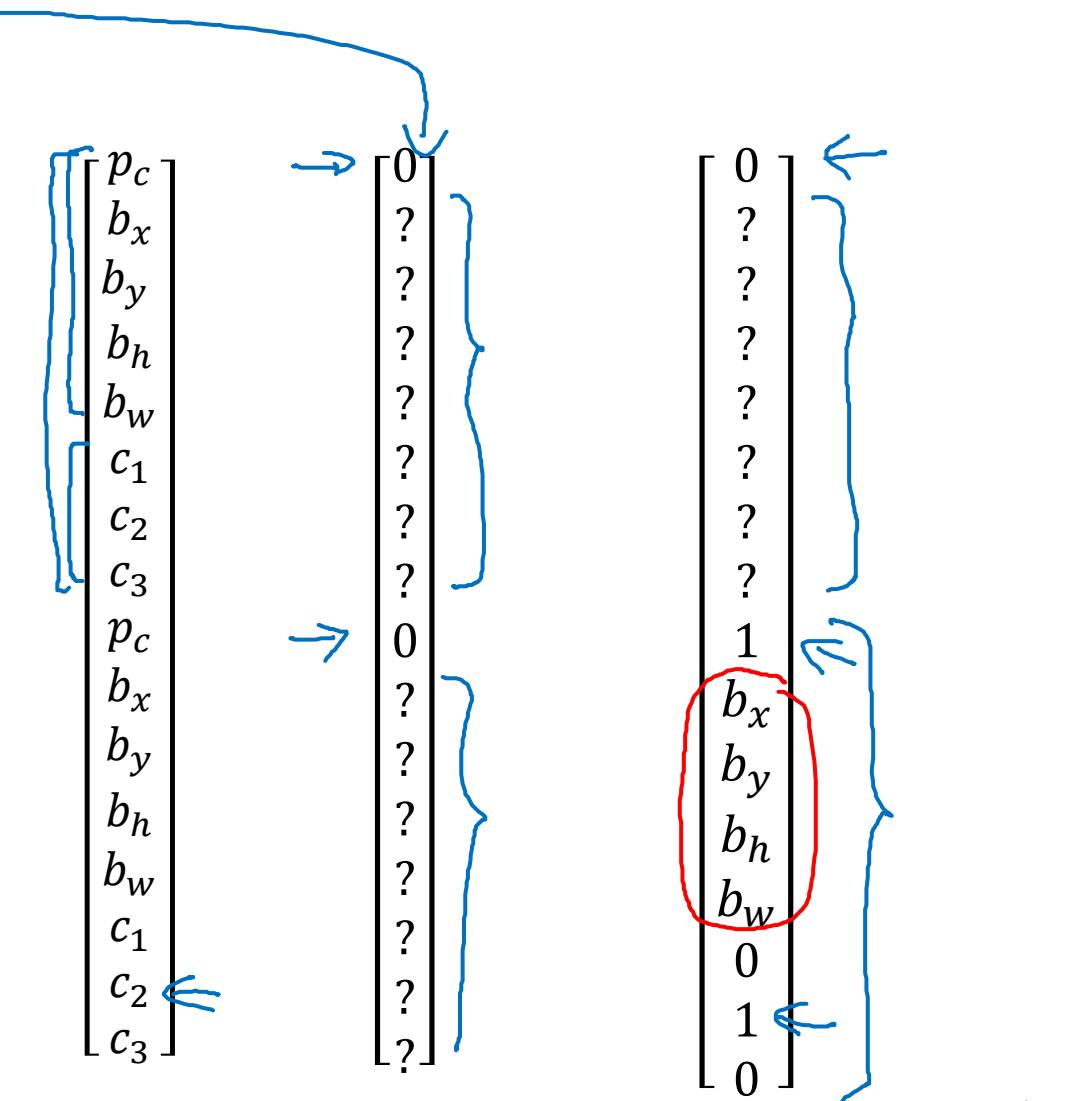
$y =$



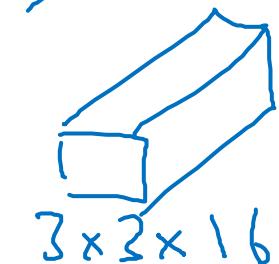
$y$  is  $3 \times 3 \times 2 \times 8$

$3 \times 3 \times 16$  (circled)  
 $19 \times 19 \times 16$   
 $19 \times 19 \times 40$

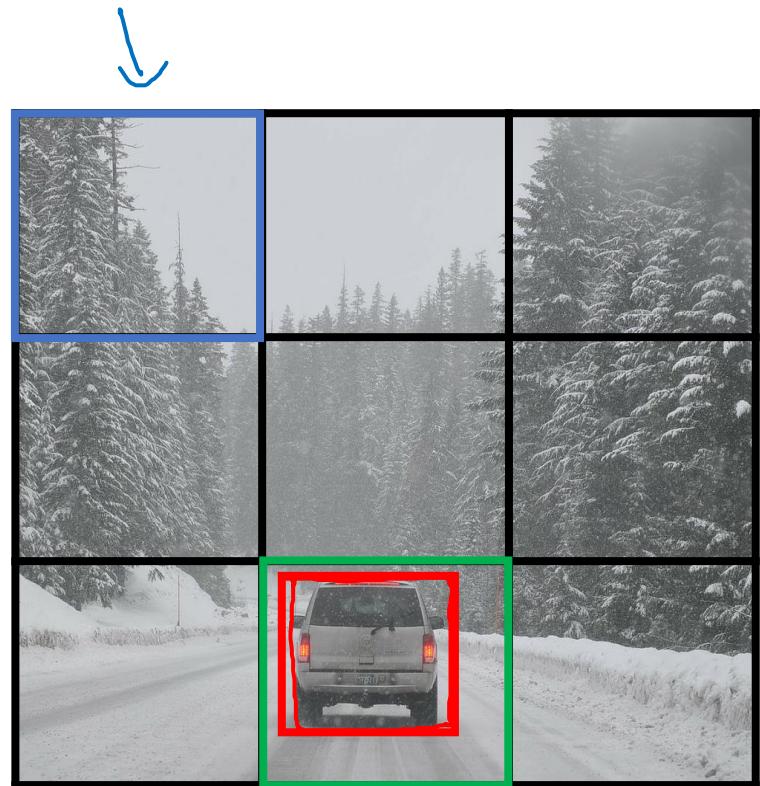
$\uparrow$  #anchors     $\uparrow$   $5 + \# \text{classes}$



$\rightarrow$  ConvNet



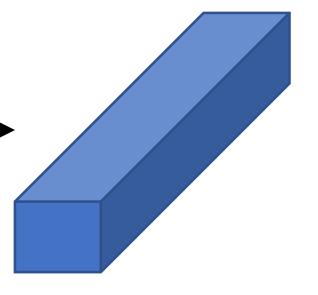
# Making predictions



→

...

→



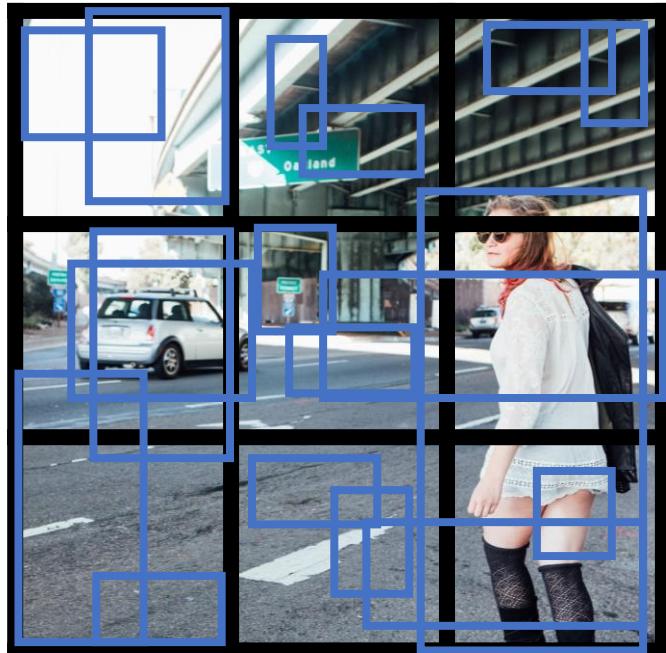
$y =$

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

Annotations on the right side of the equation:

- Blue arrows point to the first three elements ( $p_c, b_x, b_y$ ) of the first column.
- Red arrows point to the next four elements ( $b_h, b_w, c_1, c_2$ ) of the first column.
- Green arrows point to the remaining elements ( $c_3, p_c, b_x, b_y, b_h, b_w, c_1, c_2, c_3$ ) of the second column.
- A blue arrow points up to the bottom element of the second column.

# Outputting the non-max suppressed outputs



- For each grid call, get 2 predicted bounding boxes.
- Get rid of low probability predictions.
- For each class (pedestrian, car, motorcycle) use non-max suppression to generate final predictions.



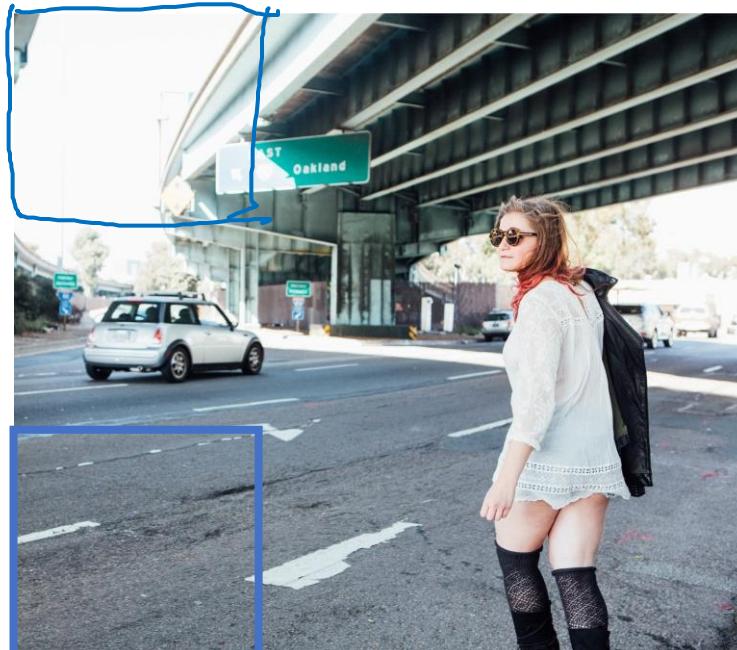
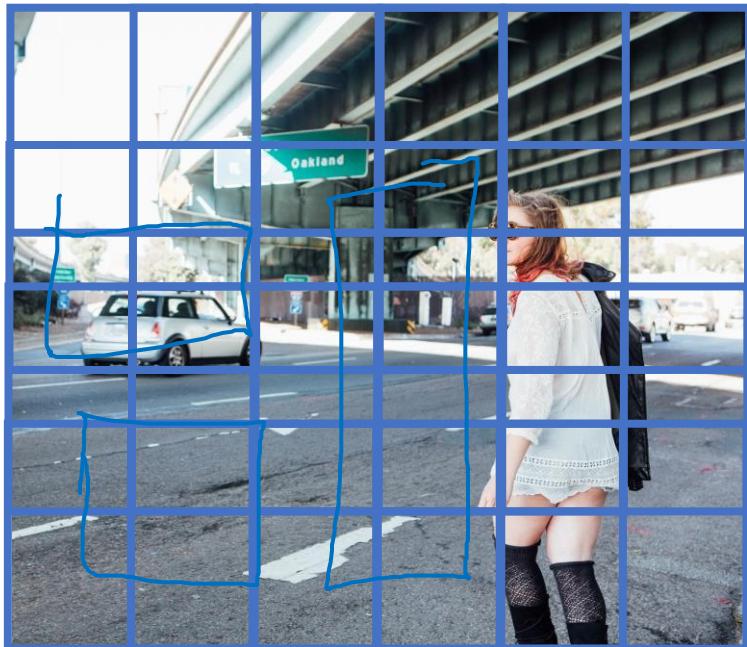
deeplearning.ai

# Object Detection

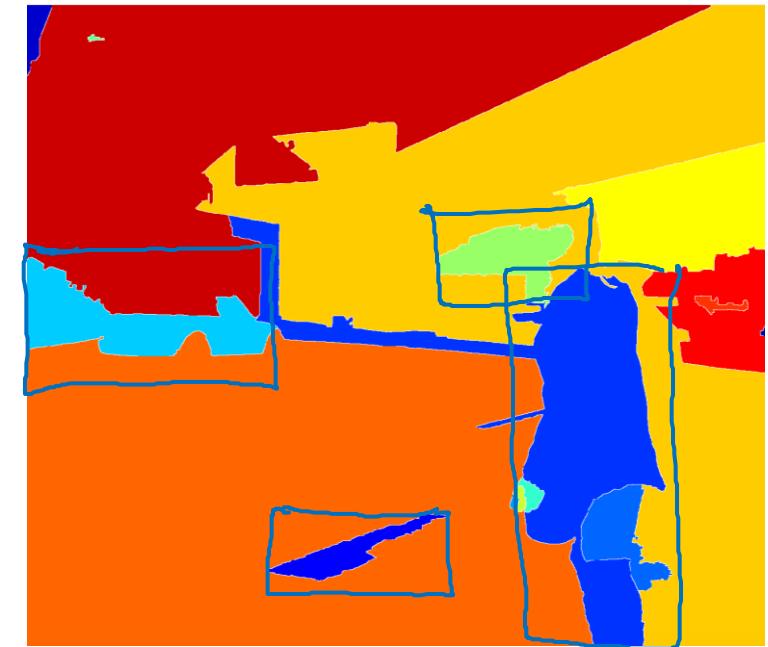
---

Region proposals  
(Optional)

# Region proposal: R-CNN



~



Segmentation algorithm

~ 2,000

# Faster algorithms

- R-CNN: Propose regions. Classify proposed regions one at a time. Output label + bounding box. ←
- Fast R-CNN: Propose regions. Use convolution implementation of sliding windows to classify all the proposed regions. ←
- Faster R-CNN: Use convolutional network to propose regions.

[Girshik et. al, 2013. Rich feature hierarchies for accurate object detection and semantic segmentation]

[Girshik, 2015. Fast R-CNN]

[Ren et. al, 2016. Faster R-CNN: Towards real-time object detection with region proposal networks]

Andrew Ng



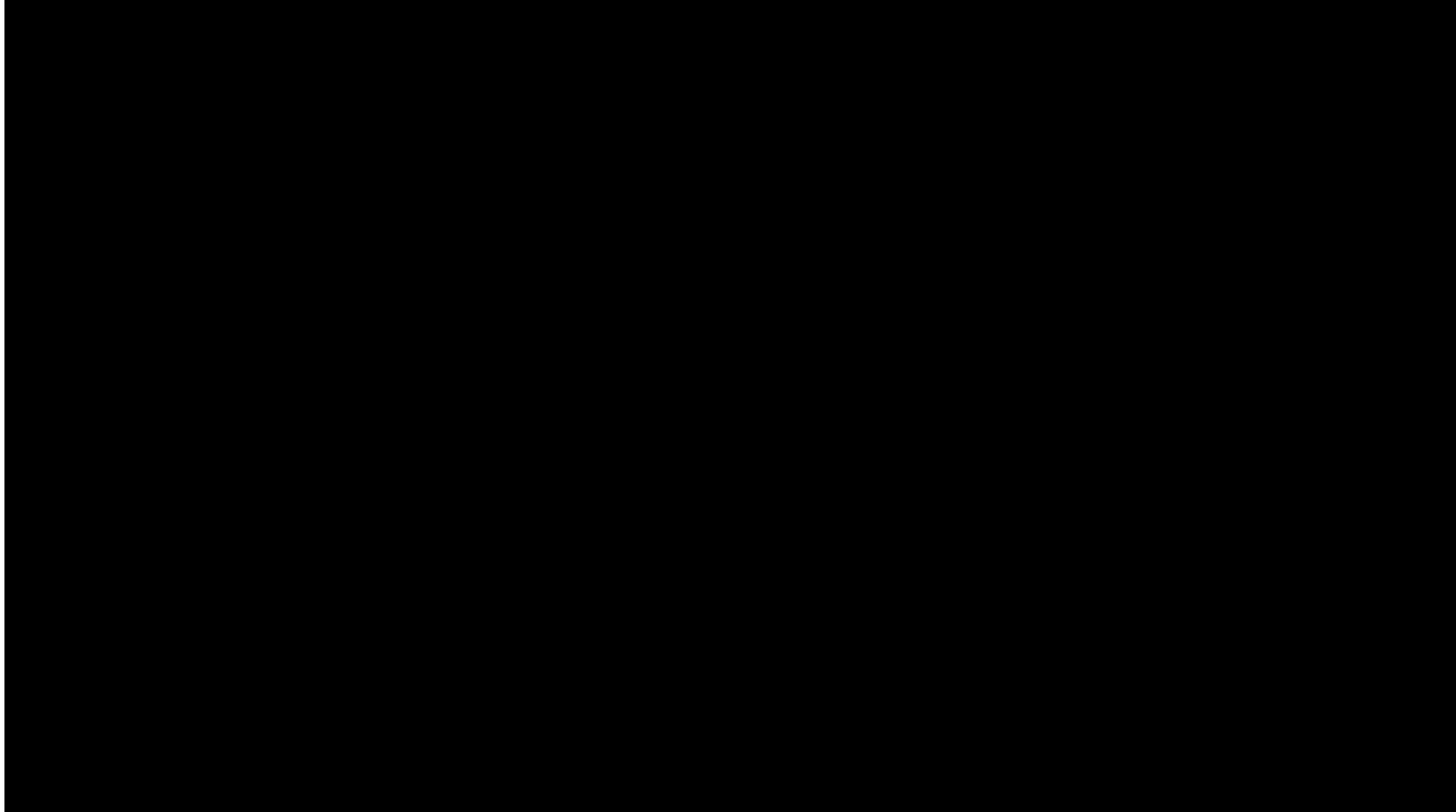
deeplearning.ai

# Face recognition

---

What is face  
recognition?

# Face recognition



# Face verification vs. face recognition

## → Verification

- Input image, name/ID
- Output whether the input image is that of the claimed person

1:1

99%

99.9

## → Recognition

- Has a database of K persons
- Get an input image
- Output ID if the image is any of the K persons (or “not recognized”)

1:K

K=100 ←



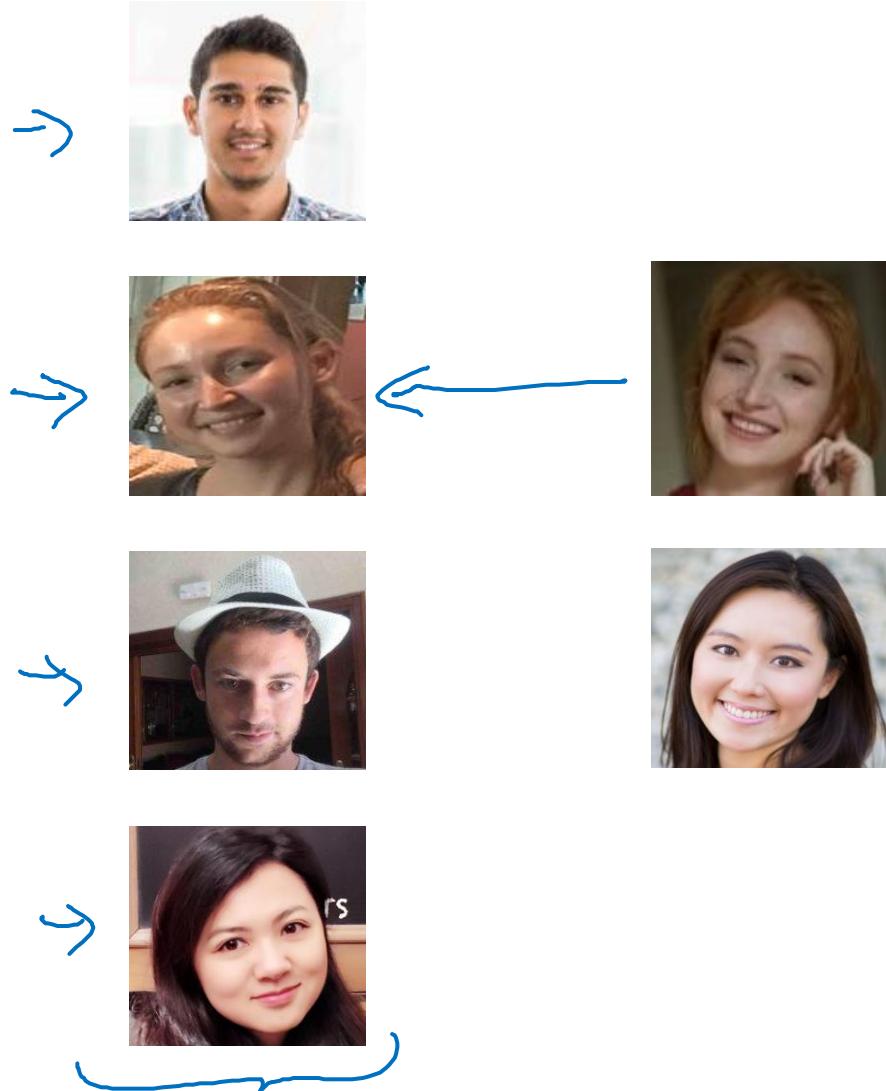
deeplearning.ai

# Face recognition

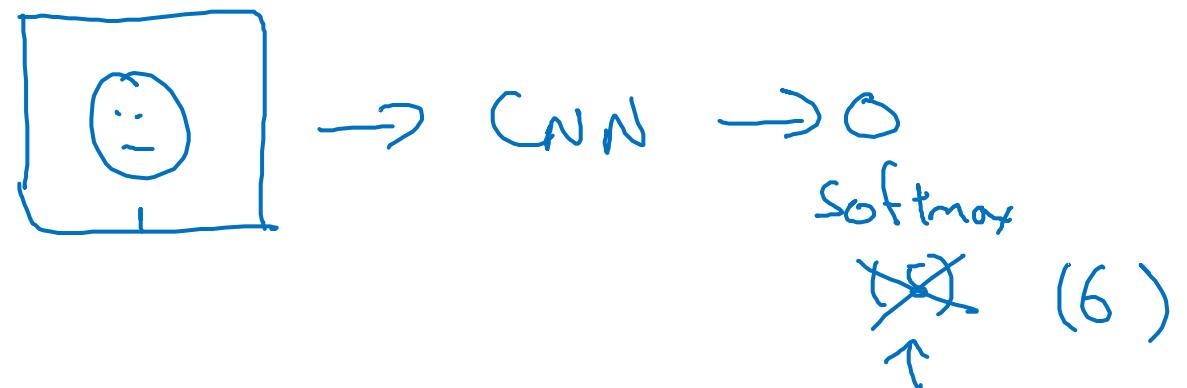
---

# One-shot learning

# One-shot learning



Learning from one example to recognize the person again



# Learning a “similarity” function

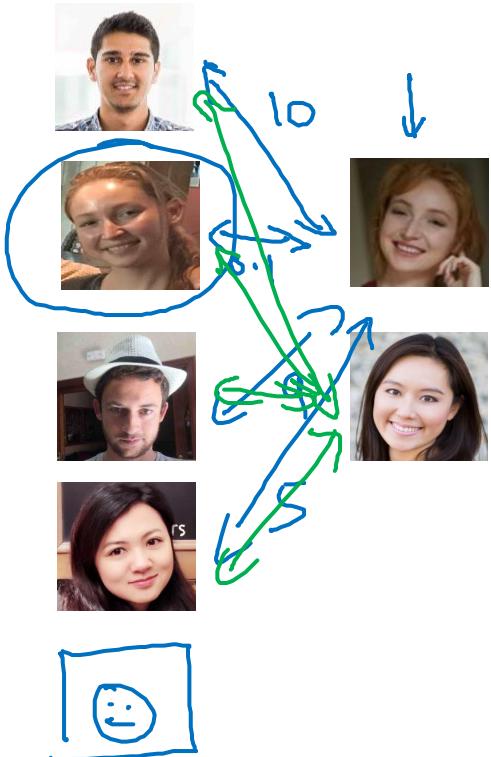
→  $d(\underline{\text{img1}}, \underline{\text{img2}})$  = degree of difference between images

If  $d(\text{img1}, \text{img2}) \leq \tau$

$> \tau$

“some”  
“different”

} Verification.



$d(\text{img1}, \text{img2})$



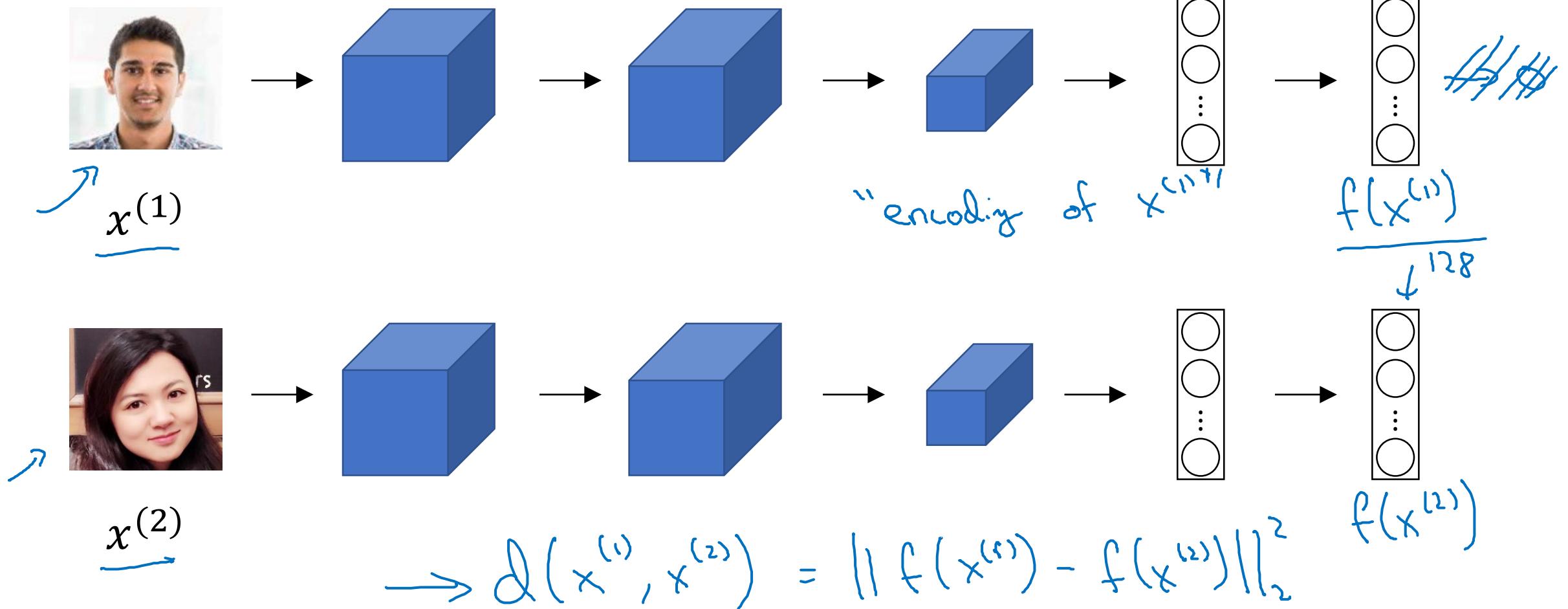
deeplearning.ai

# Face recognition

---

## Siamese network

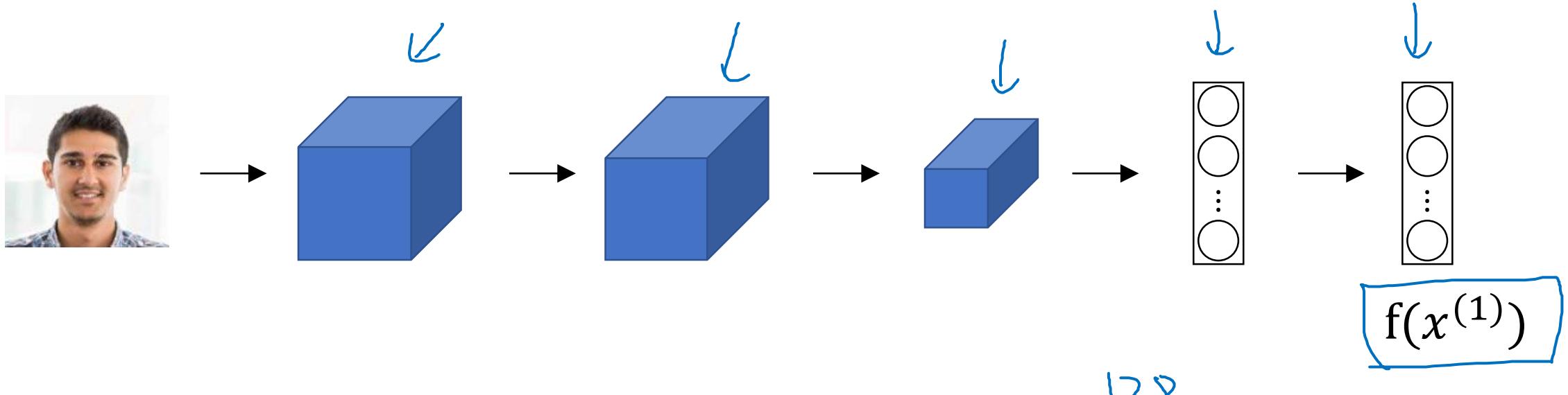
# Siamese network



[Taigman et. al., 2014. DeepFace closing the gap to human level performance]

Andrew Ng

# Goal of learning



Parameters of NN define an encoding  $f(x^{(i)})$  128

Learn parameters so that:

If  $x^{(i)}, x^{(j)}$  are the same person,  $\|f(x^{(i)}) - f(x^{(j)})\|^2$  is small.

If  $x^{(i)}, x^{(j)}$  are different persons,  $\|f(x^{(i)}) - f(x^{(j)})\|^2$  is large.



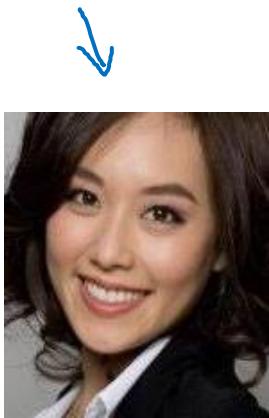
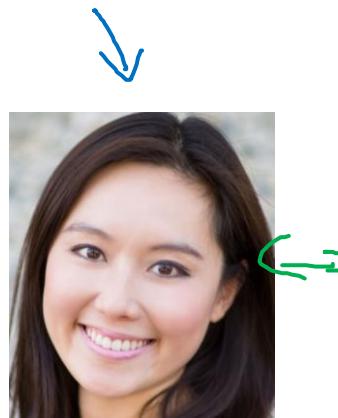
deeplearning.ai

# Face recognition

---

## Triplet loss

# Learning Objective



Anchor A      Positive P       $\rightarrow 0.2$

Negative N       $d(A,N) = \frac{N}{0.5} \rightarrow 0.7$

Want:  $\frac{\|f(A) - f(P)\|^2}{d(A,P)} + \lambda \leq \frac{\|f(A) - f(N)\|^2}{d(A,N)}$

$$\frac{\|f(A) - f(P)\|^2}{\textcircled{O}} - \frac{\|f(A) - f(N)\|^2}{\textcircled{O}} + \lambda \leq \textcircled{O} \quad 4/4 \quad f(\text{img}) = \vec{0}$$

Margin

# Loss function

Given 3 images

$A, P, N$ :

$$\underline{L(A, P, N)} = \max \left( \left[ \|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \lambda \right], 0 \right)$$

$$J = \sum_{i=1}^m L(A^{(i)}, P^{(i)}, N^{(i)})$$

$A, P$   
 $T$

Training set:  $\underbrace{10k}_{\infty}$  pictures of  $\frac{1k}{\infty}$  persons

# Choosing the triplets A,P,N



During training, if A,P,N are chosen randomly,  
 $d(A, P) + \alpha \leq d(A, N)$  is easily satisfied.

$$\underbrace{\|f(A) - f(P)\|^2}_{\text{Distance between } A \text{ and } P} + \alpha \leq \underbrace{\|f(A) - f(N)\|^2}_{\text{Distance between } A \text{ and } N}$$

Choose triplets that're “hard” to train on.

$$\underbrace{d(A, P)}_{\downarrow} + \alpha \approx \underbrace{d(A, N)}_{\uparrow}$$

Face Net  
Deep Face



# Training set using triplet loss

Anchor



Positive



Negative



:

:

:



J

$$d(x^{(i)}, x^{(j)})$$



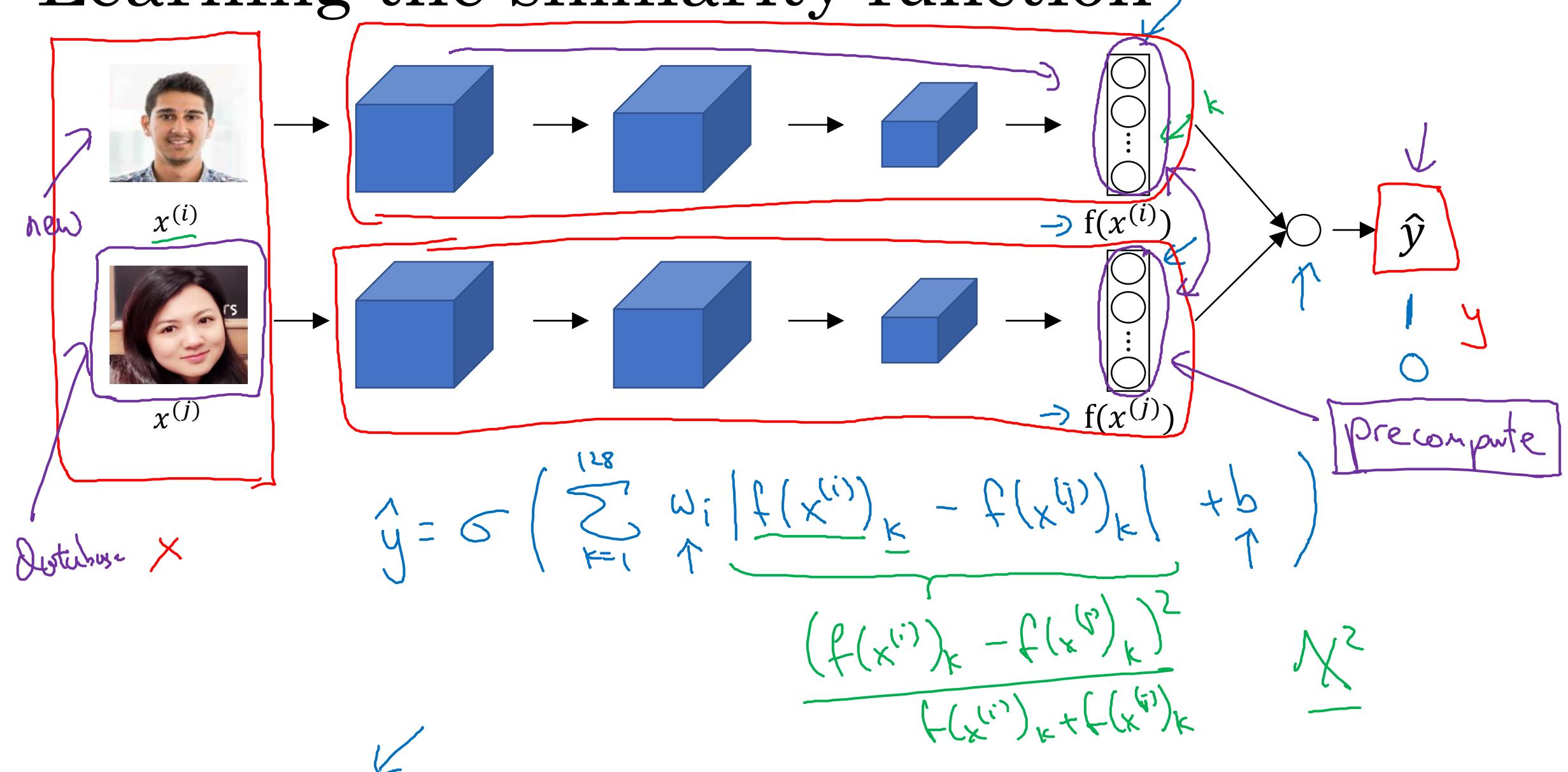
deeplearning.ai

# Face recognition

---

# Face verification and binary classification

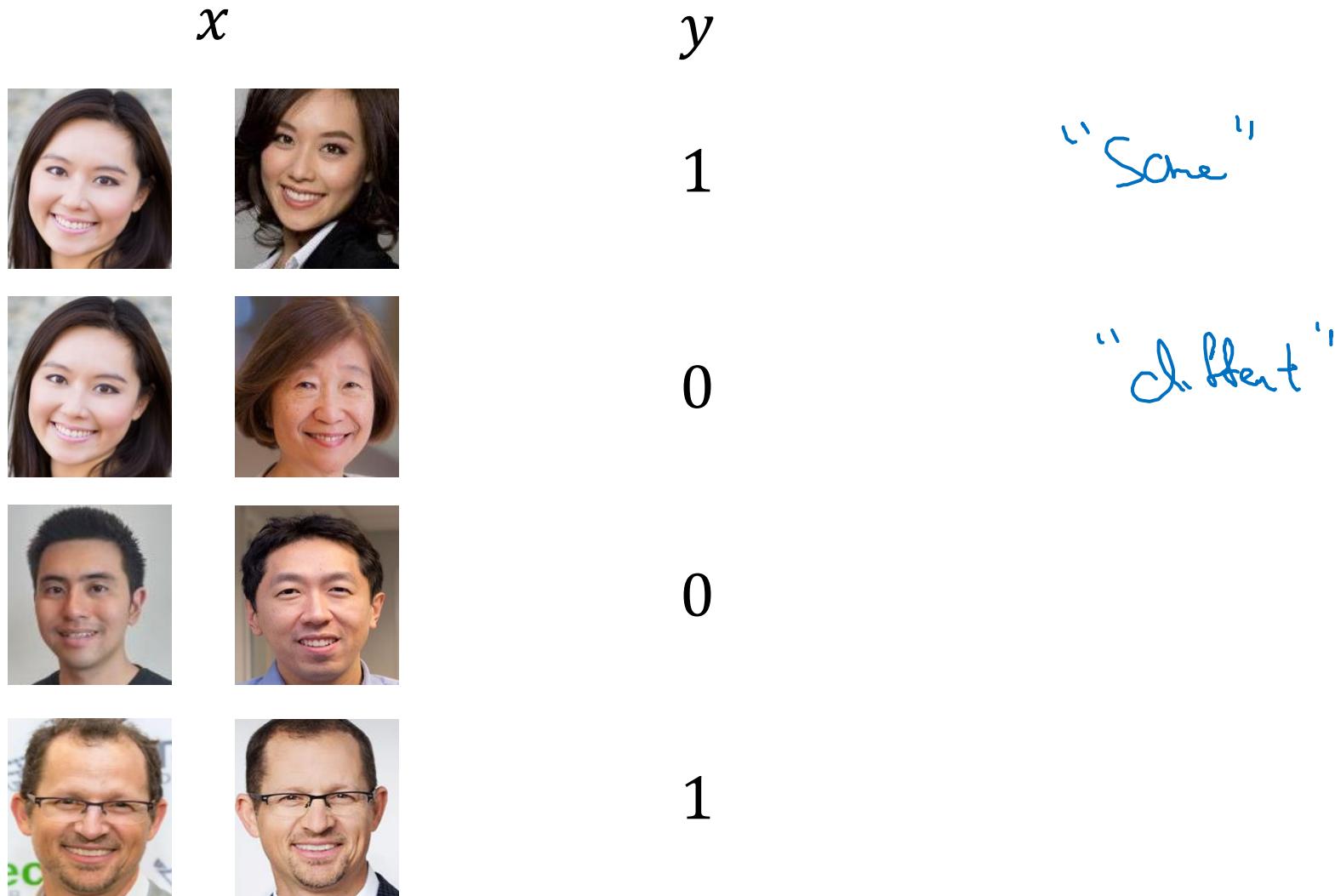
# Learning the similarity function



[Taigman et. al., 2014. DeepFace closing the gap to human level performance]

Andrew Ng

# Face verification supervised learning





deeplearning.ai

# Neural Style Transfer

---

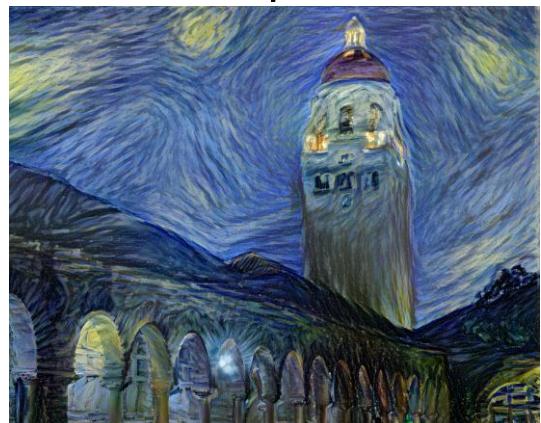
What is neural style  
transfer?

# Neural style transfer



Content ( $c$ )

Style ( $s$ )



Generated image ( $G$ )



Content ( $c$ )

Style ( $s$ )



Generated image ( $G$ )



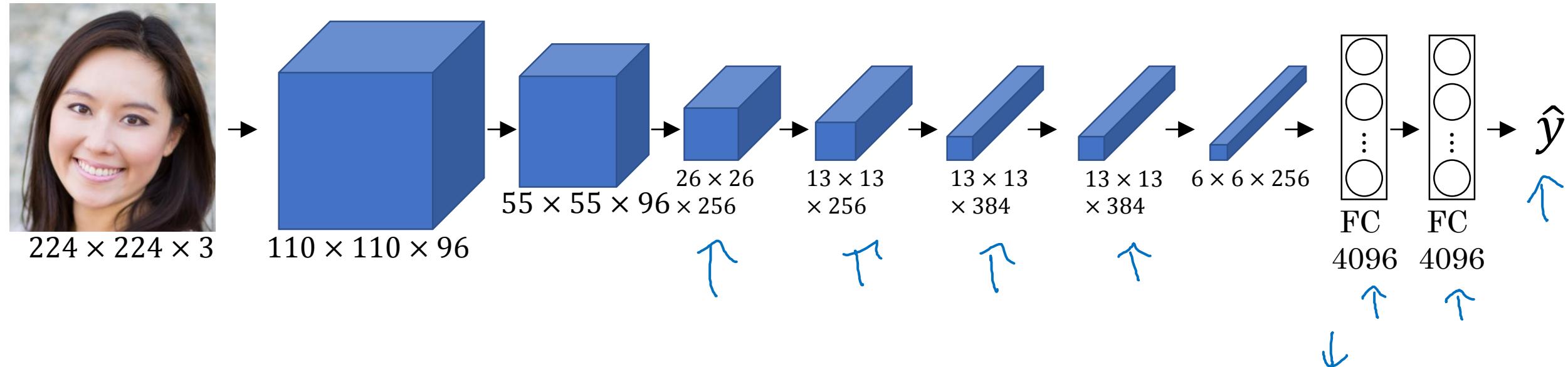
deeplearning.ai

# Neural Style Transfer

---

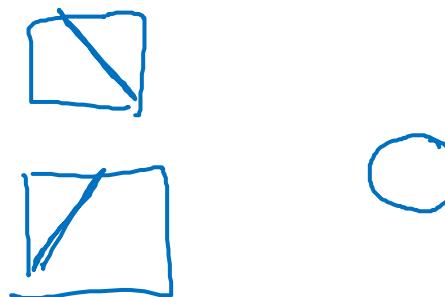
What are deep  
ConvNets learning?

# Visualizing what a deep network is learning

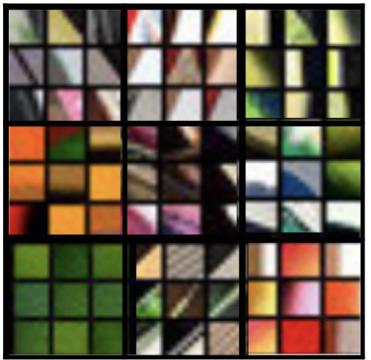


Pick a unit in layer 1. Find the nine image patches that maximize the unit's activation.

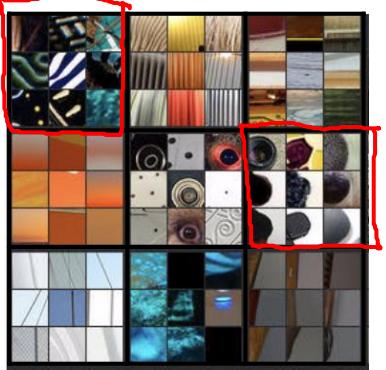
Repeat for other units.



# Visualizing deep layers



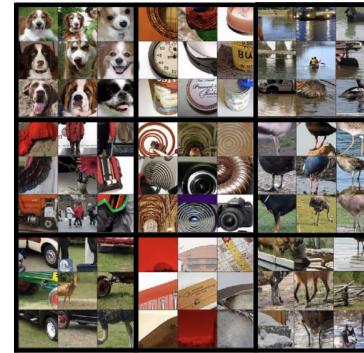
Layer 1



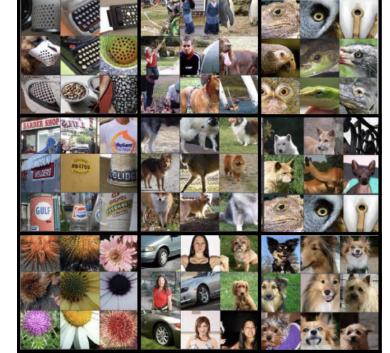
Layer 2



Layer 3

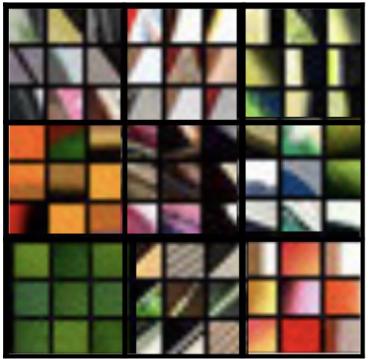


Layer 4



Layer 5

# Visualizing deep layers: Layer 1



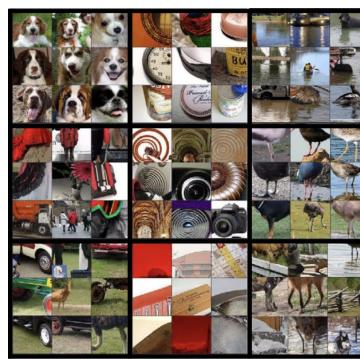
Layer 1



Layer 2



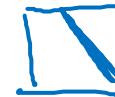
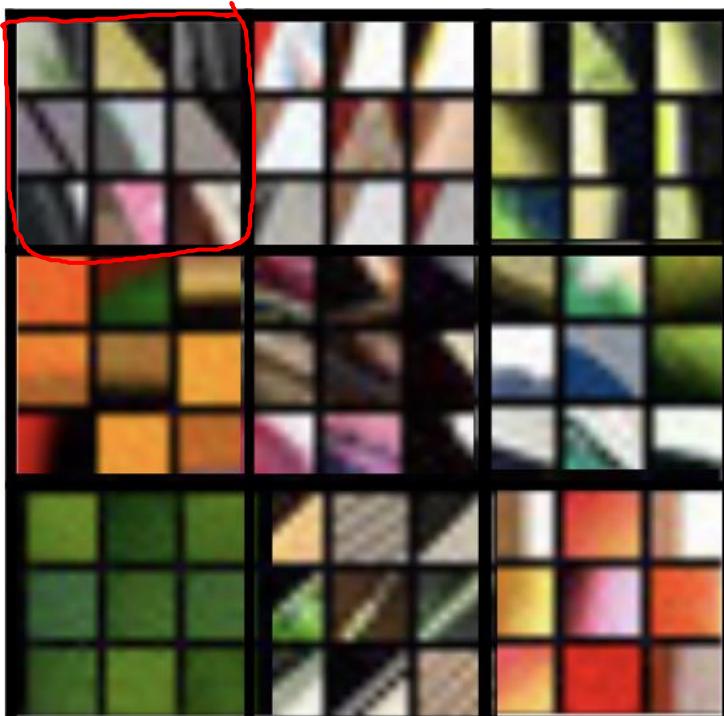
Layer 3



Layer 4



Layer 5



# Visualizing deep layers: Layer 2



Layer 1



Layer 2



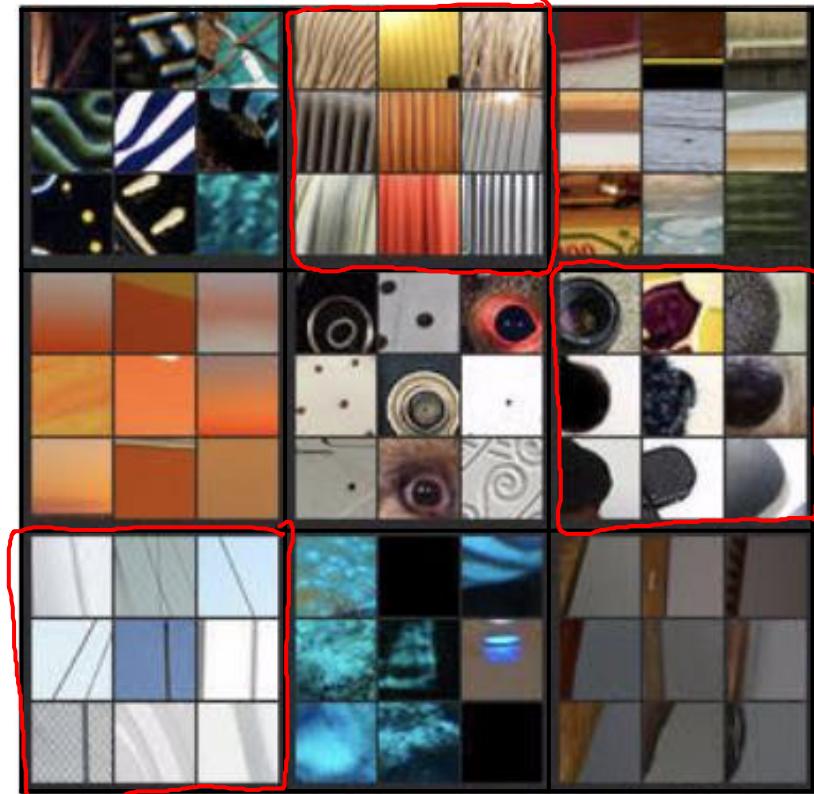
Layer 3



Layer 4



Layer 5



# Visualizing deep layers: Layer 3



Layer 1



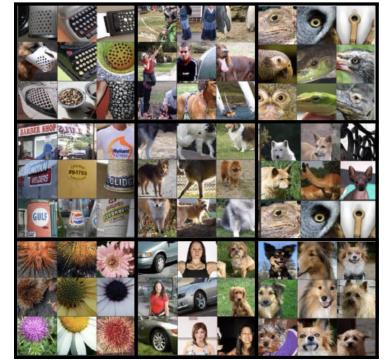
Layer 2



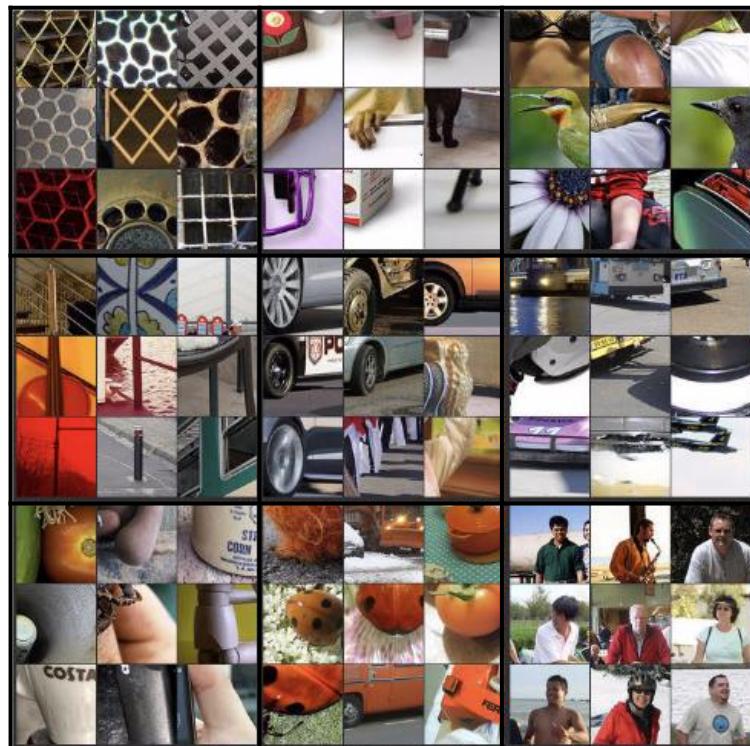
Layer 3



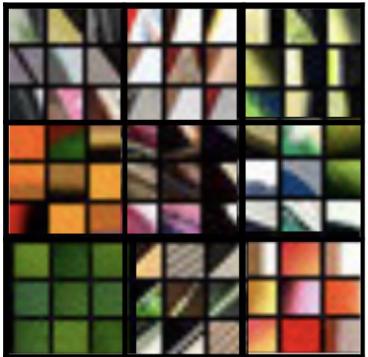
Layer 4



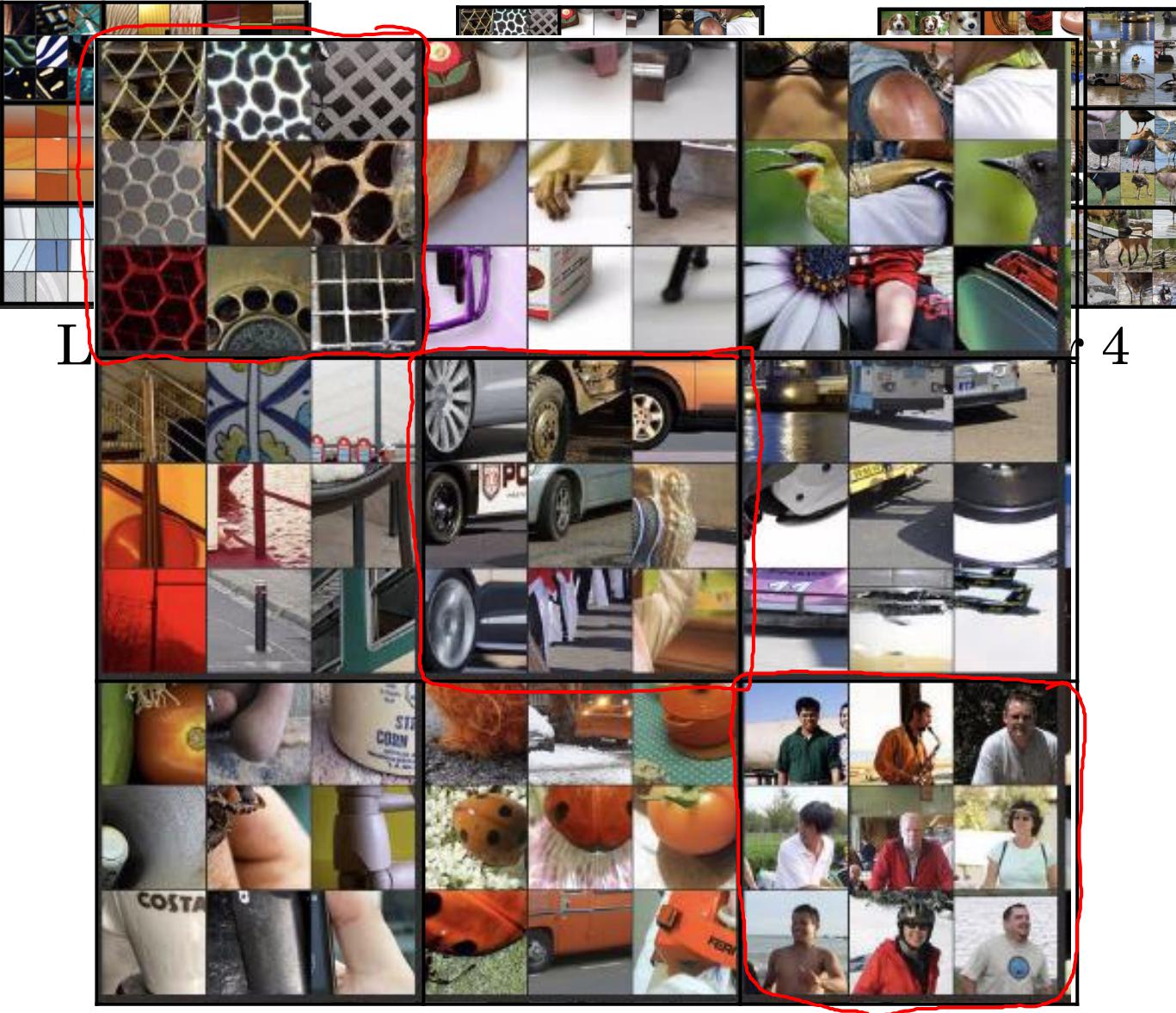
Layer 5



# Visualizing deep layers: Layer 3



Layer 1

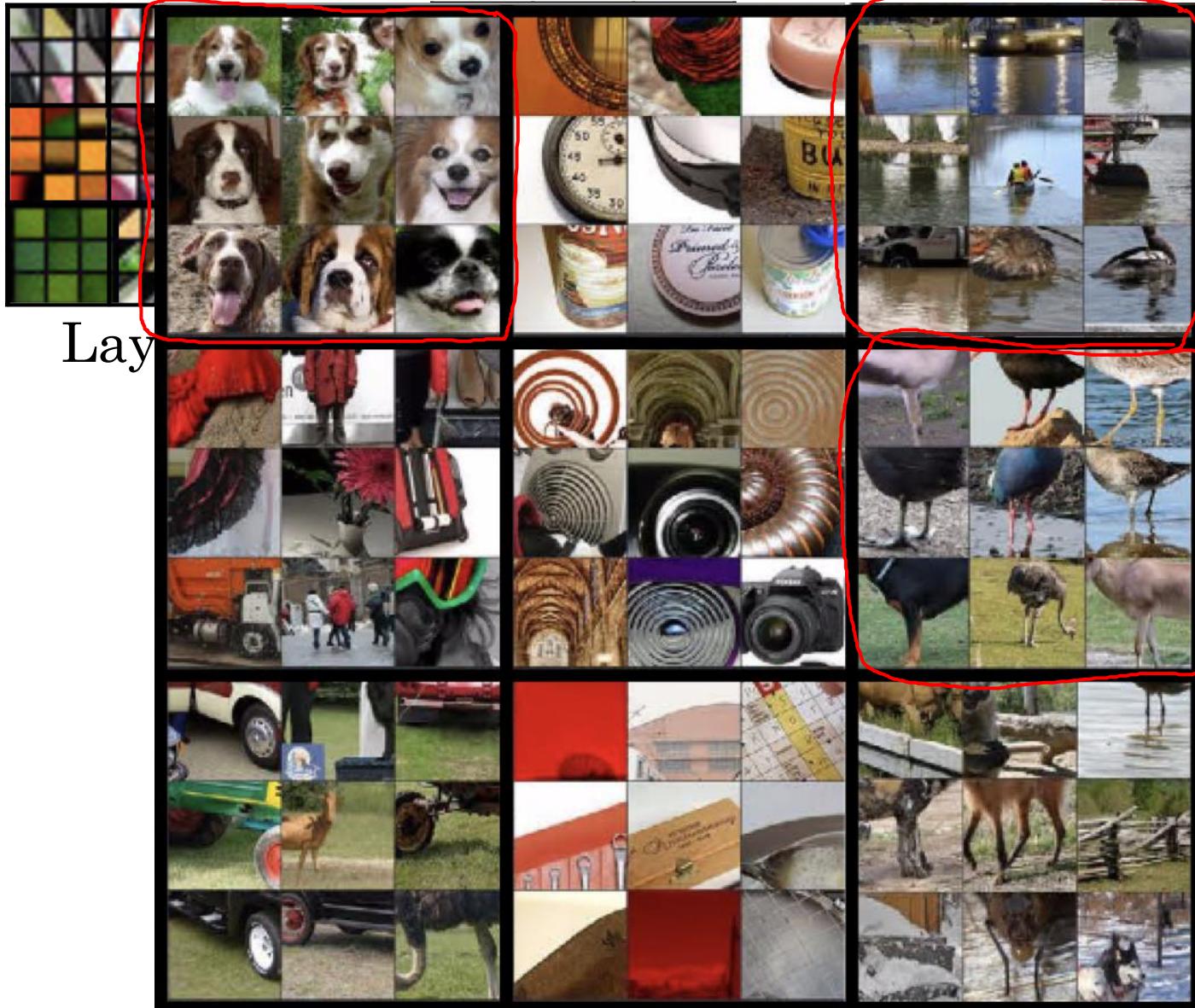


Layer 3



Layer 5

# Visualizing deep layers: Layer 4



Layer 4

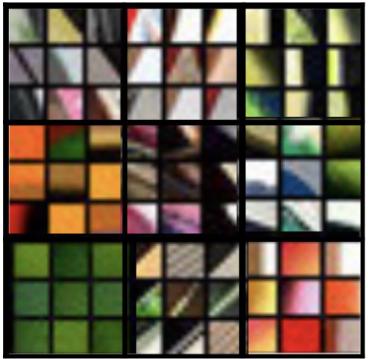


Layer 4



Layer 5

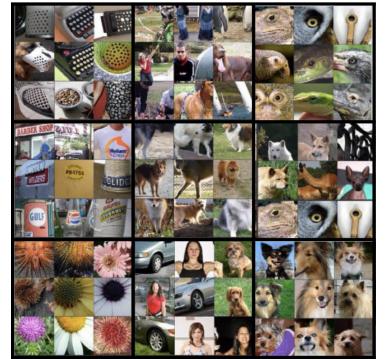
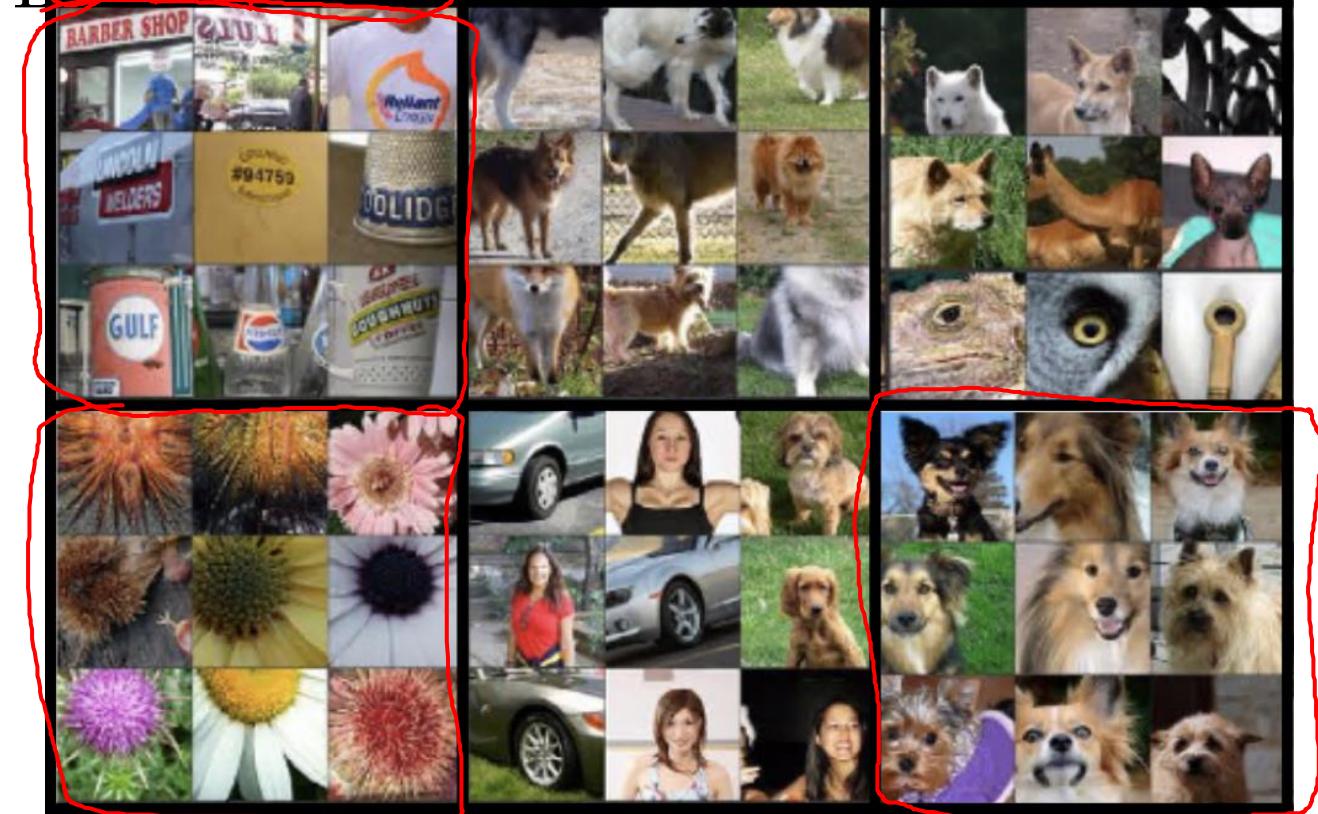
# Visualizing deep layers: Layer 5



Layer 1



I



Layer 5



deeplearning.ai

# Neural Style Transfer

---

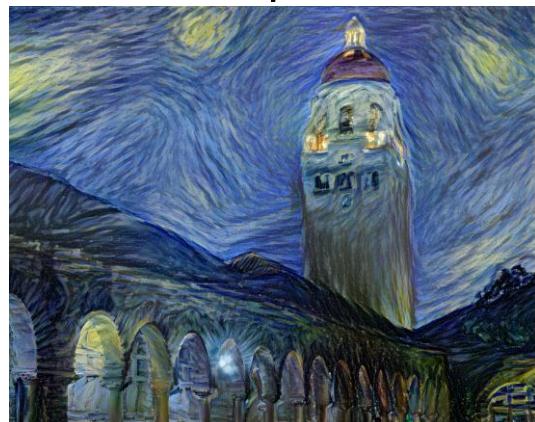
## Cost function

# Neural style transfer cost function



Content C

Style S



Generated image G

$$J(G) = \alpha J_{\text{Content}}(C, G) + \beta J_{\text{Style}}(S, G)$$

# Find the generated image $G$

1. Initiate  $G$  randomly

$\underline{G: 100 \times 100 \times 3}$

$\uparrow$   
 $\text{RGB}$



2. Use gradient descent to minimize  $\underline{J(G)}$

$$G_t := G - \frac{\partial}{\partial G} J(G)$$





deeplearning.ai

# Neural Style Transfer

---

## Content cost function

# Content cost function

$$\underline{J}(G) = \alpha \underline{J}_{content}(C, G) + \beta J_{style}(S, G)$$

- Say you use hidden layer  $\underline{l}$  to compute content cost.
- Use pre-trained ConvNet. (E.g., VGG network)
- Let  $\underline{a}^{[l](C)}$  and  $\underline{a}^{[l](G)}$  be the activation of layer  $\underline{l}$  on the images
- If  $\underline{a}^{[l](C)}$  and  $\underline{a}^{[l](G)}$  are similar, both images have similar content

$$J_{content}(C, G) = \frac{1}{2} \left\| \underline{a}^{[l](C)} - \underline{a}^{[l](G)} \right\|^2$$



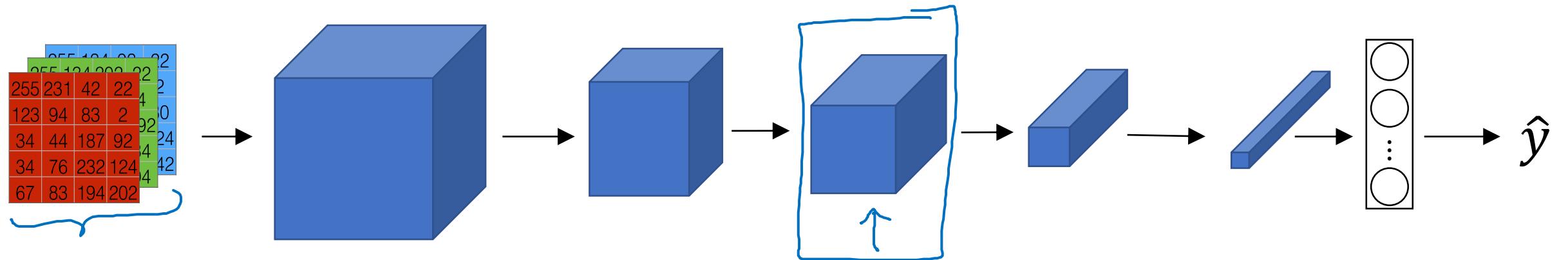
deeplearning.ai

# Neural Style Transfer

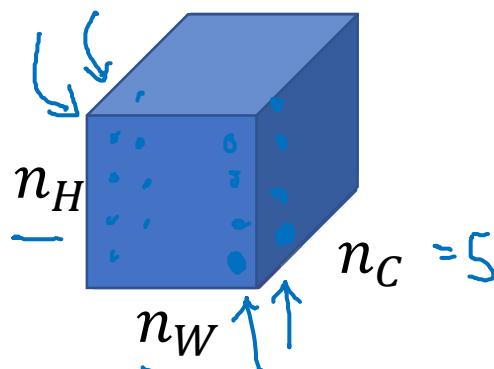
---

## Style cost function

# Meaning of the “style” of an image

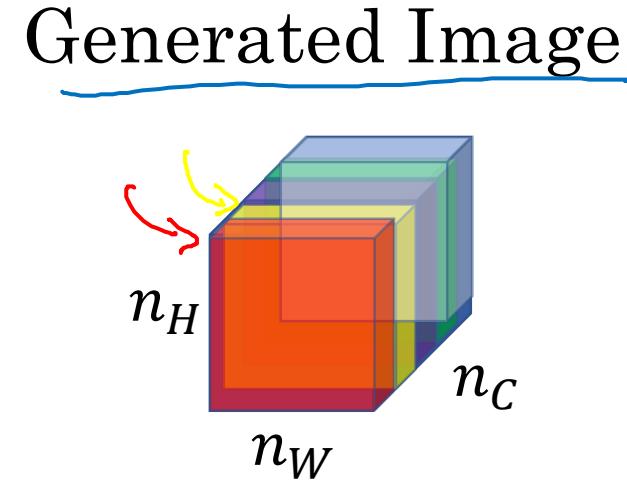
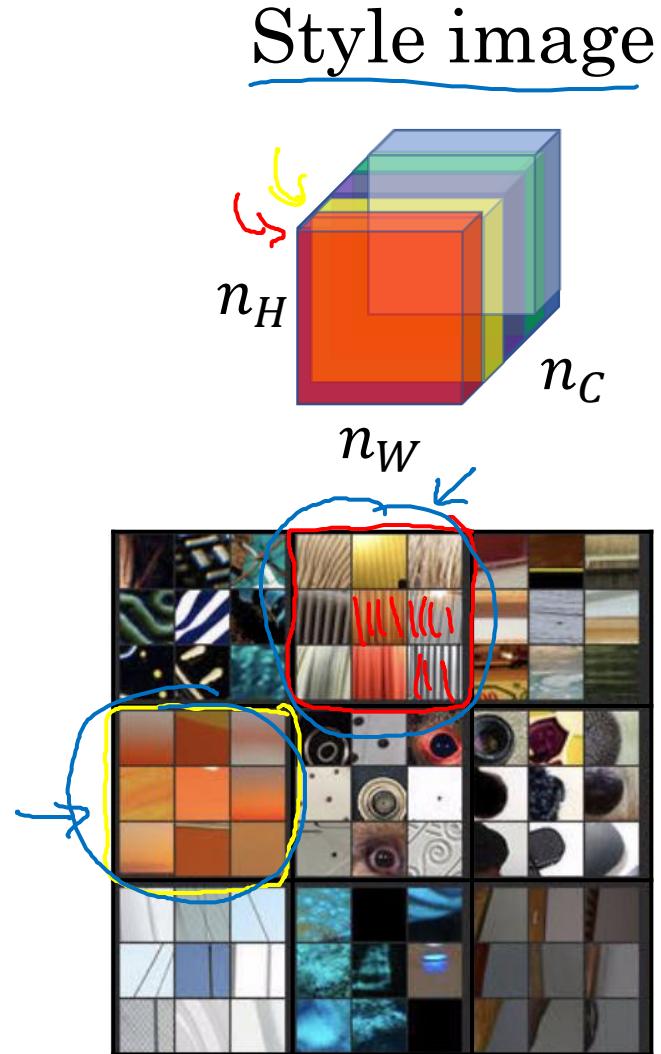


Say you are using layer  $l$ 's activation to measure “style.”  
Define style as correlation between activations across channels.



How correlated are the activations  
across different channels?

# Intuition about style of an image



# Style matrix

Let  $a_{i,j,k}^{[l]}$  = activation at  $(i, j, k)$ .  $G^{[l]}$  is  $n_c^{[l]} \times n_c^{[l]}$

$$\rightarrow G_{kk'}^{[l](s)} = \sum_{i=1}^{n_h^{[l]}} \sum_{j=1}^{n_w^{[l]}} a_{ijk}^{[l](s)} a_{ijk'}^{[l](s)}$$

$$\rightarrow G_{kk'}^{[l](G)} = \sum_{i=1}^{n_h^{[l]}} \sum_{j=1}^{n_w^{[l]}} a_{ijk}^{[l](G)} a_{ijk'}^{[l](G)}$$

H W C  
↓ ↓ ↓

$n_c$

$$G_{kk'}^{[l]} \quad \forall k, k' \in \{1, \dots, n_c\}$$

"Gram matrix"

$$\begin{aligned} J_{style}^{[l]}(S, G) &= \frac{1}{(\dots)} \| G_{kk'}^{[l](s)} - G_{kk'}^{[l](G)} \|_F^2 \\ &= \frac{1}{(2n_h^{[l]} n_w^{[l]} n_c^{[l]})^2} \sum_k \sum_{k'} (G_{kk'}^{[l](s)} - G_{kk'}^{[l](G)})^2 \end{aligned}$$

# Style cost function

$$\left\| G^{[l](s)} - G^{[l](G)} \right\|_F^2$$

$$J_{style}^{[l]}(S, G) = \frac{1}{\left(2n_H^{[l]} n_W^{[l]} n_C^{[l]}\right)^2} \sum_k \sum_{k'} (G_{kk'}^{[l](s)} - G_{kk'}^{[l](G)})$$

$$J_{style}(S, G) = \sum_l \lambda^{[l]} J_{style}^{[l]}(S, G)$$

$$\underbrace{J(G)}_G = \alpha J_{content}(S, G) + \beta J_{style}(S, G)$$



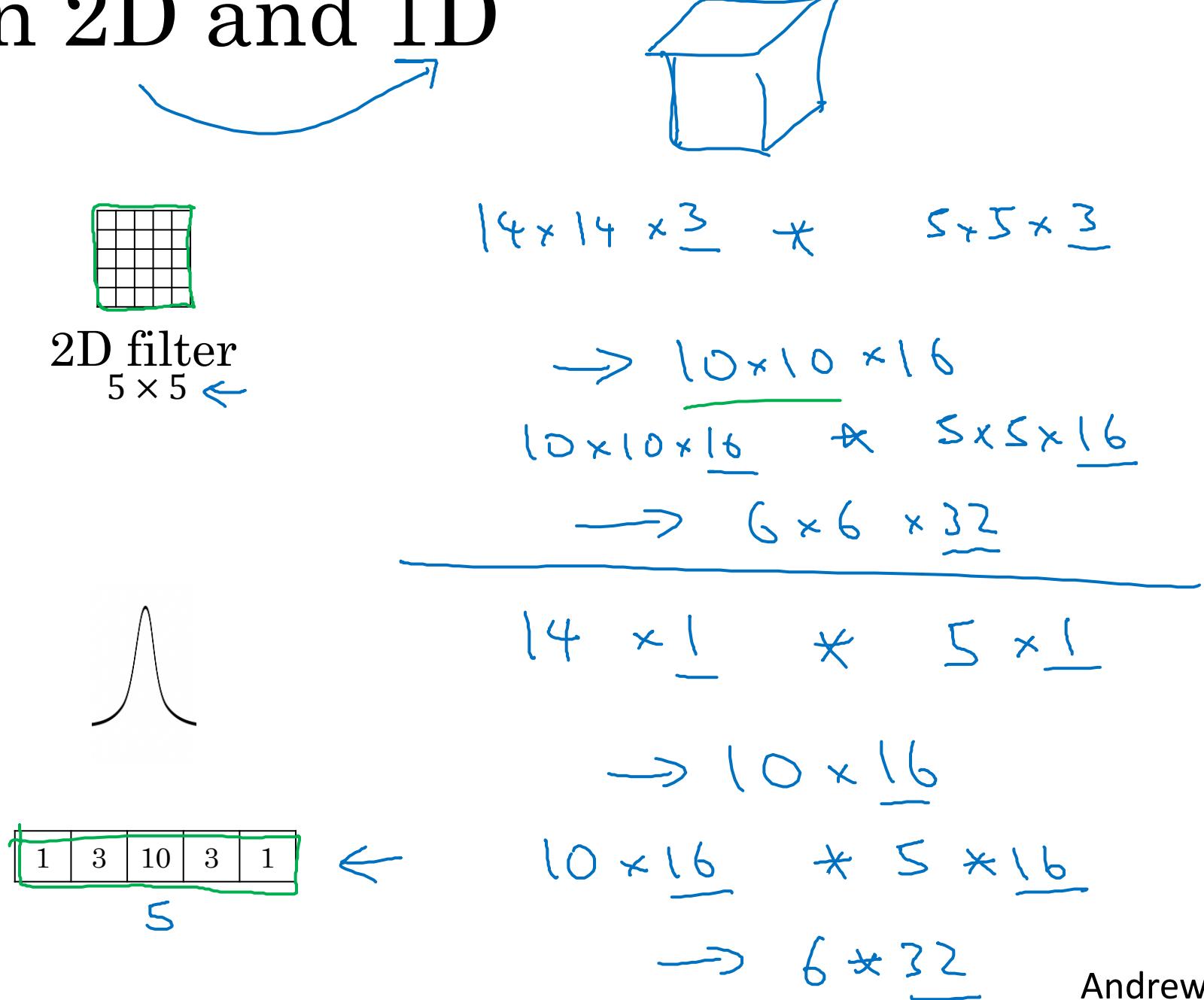
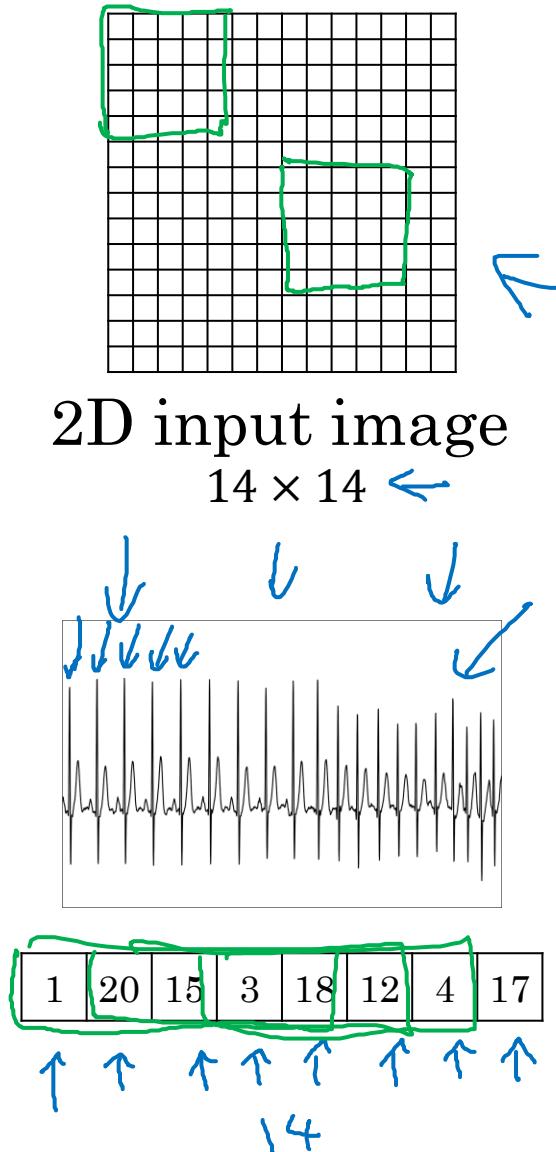
deeplearning.ai

# Convolutional Networks in 1D or 3D

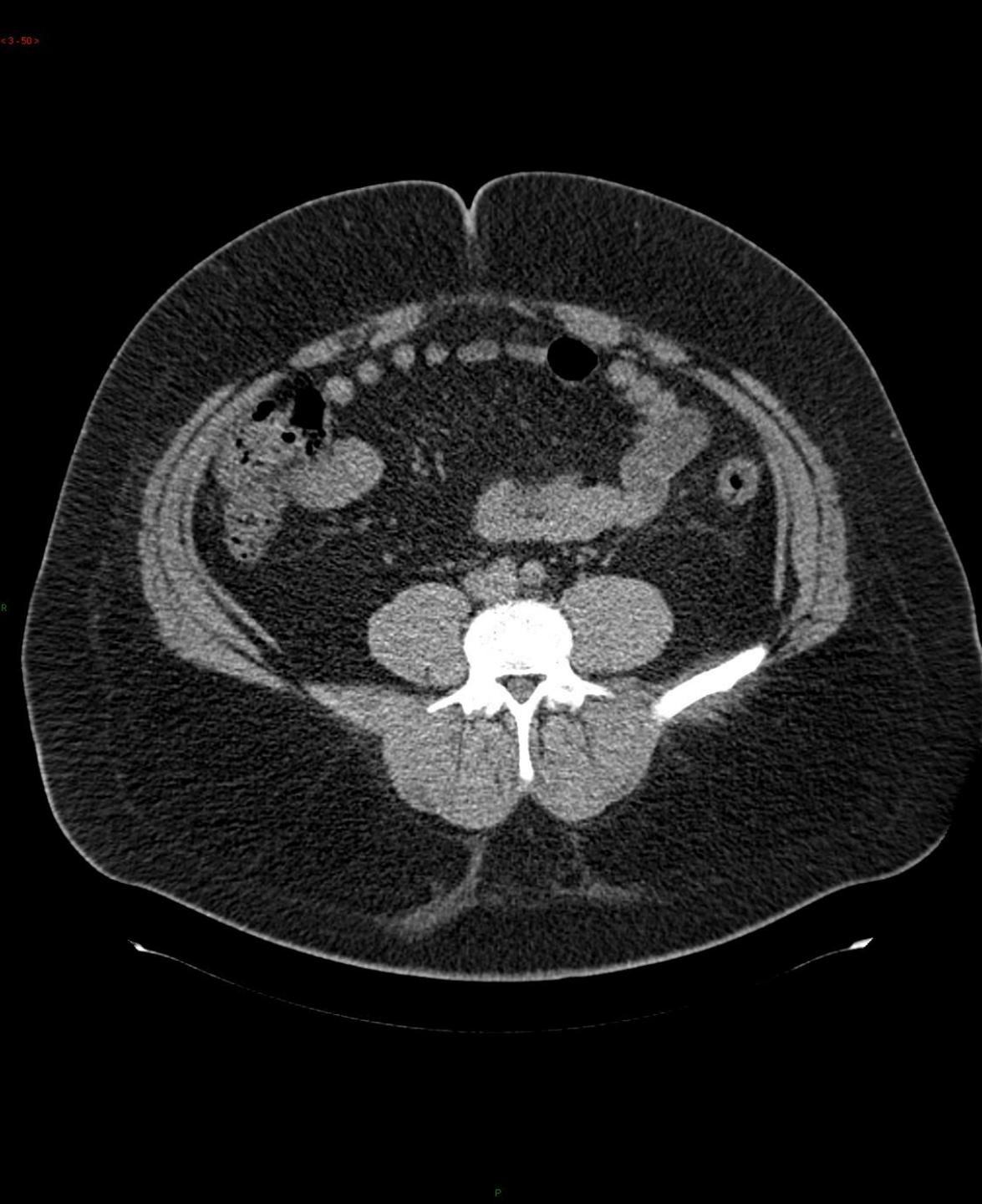
---

1D and 3D  
generalizations of  
models

# Convolutions in 2D and 1D



# 3D data



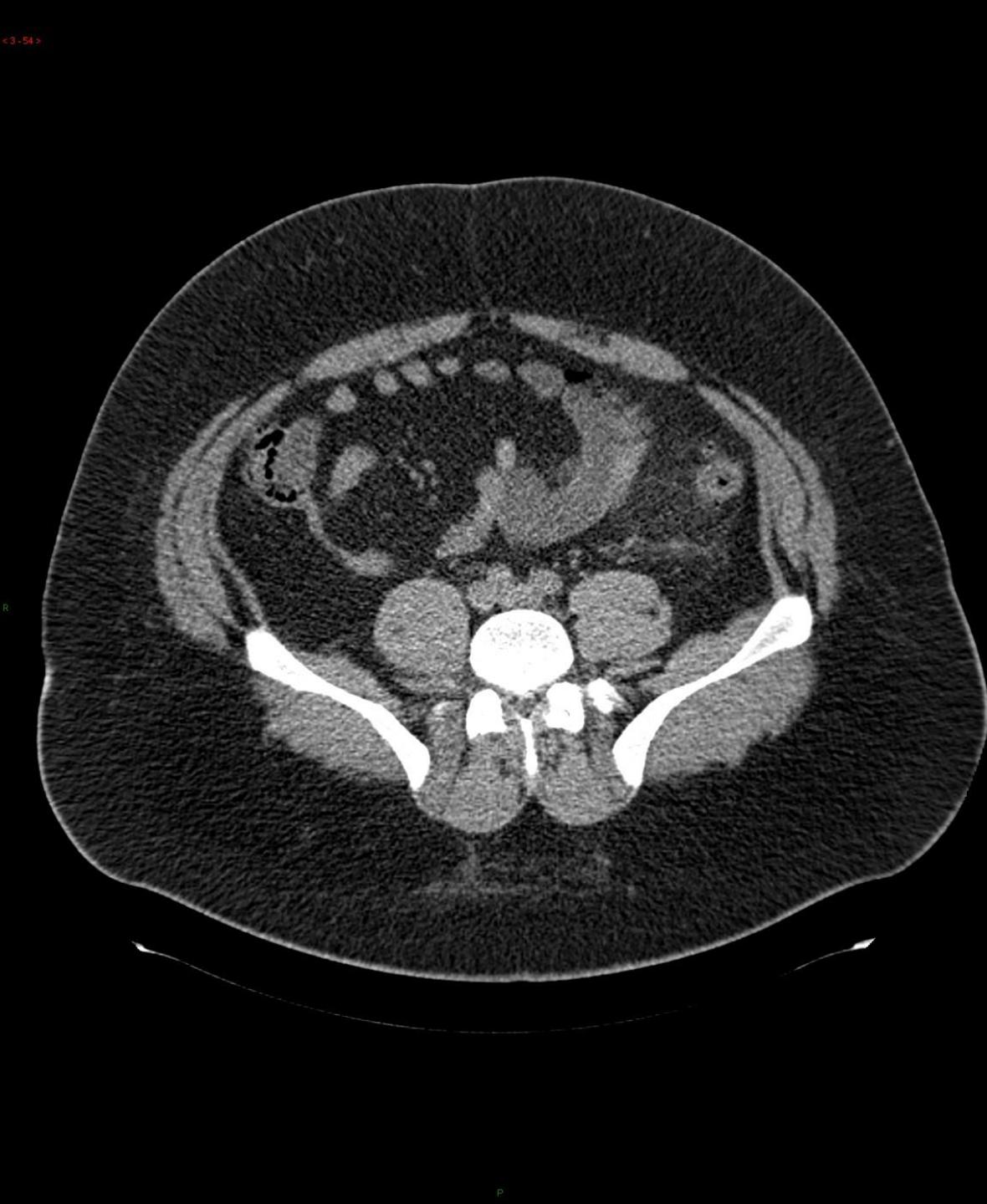
Andrew Ng

# 3D data



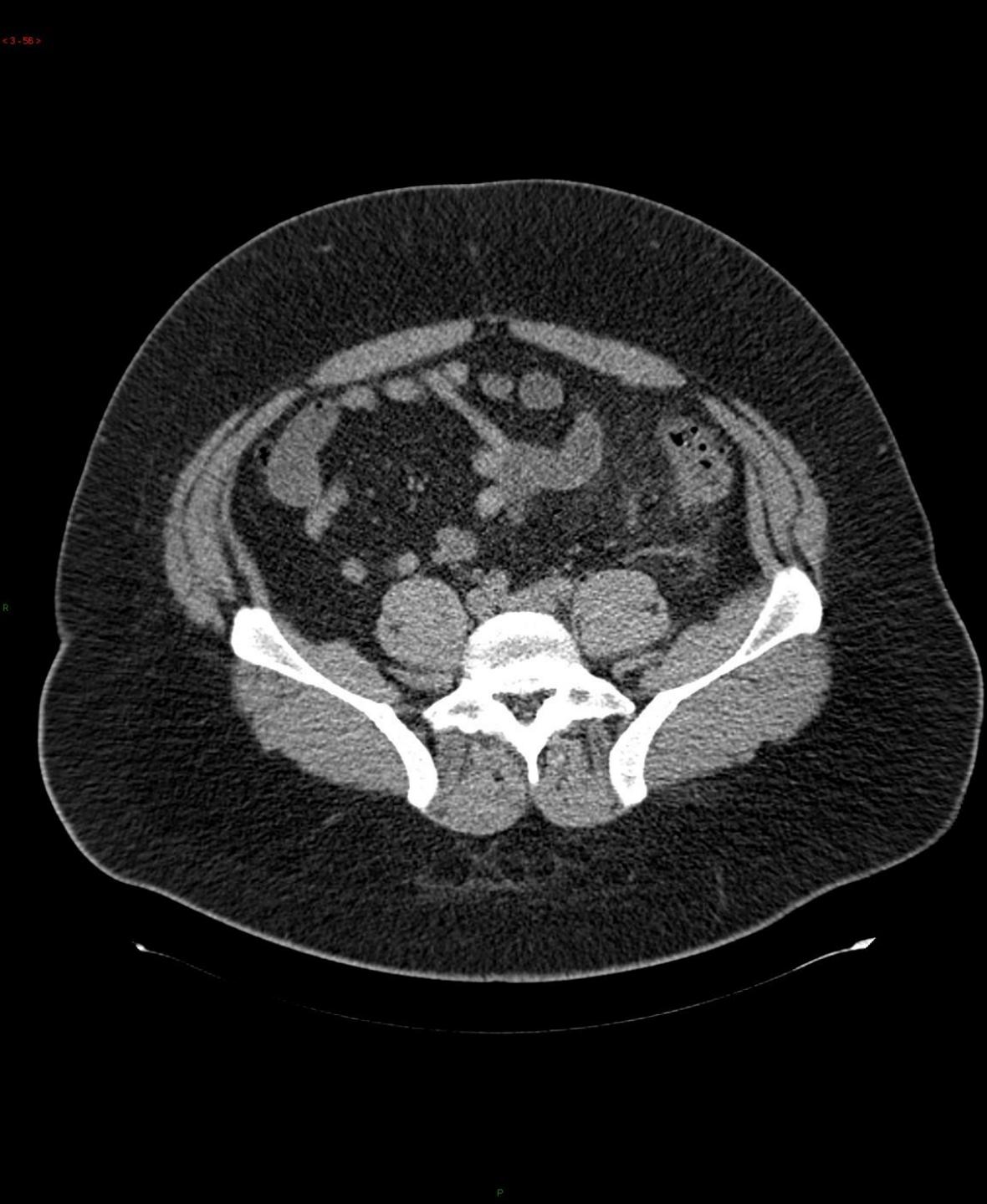
Andrew Ng

# 3D data

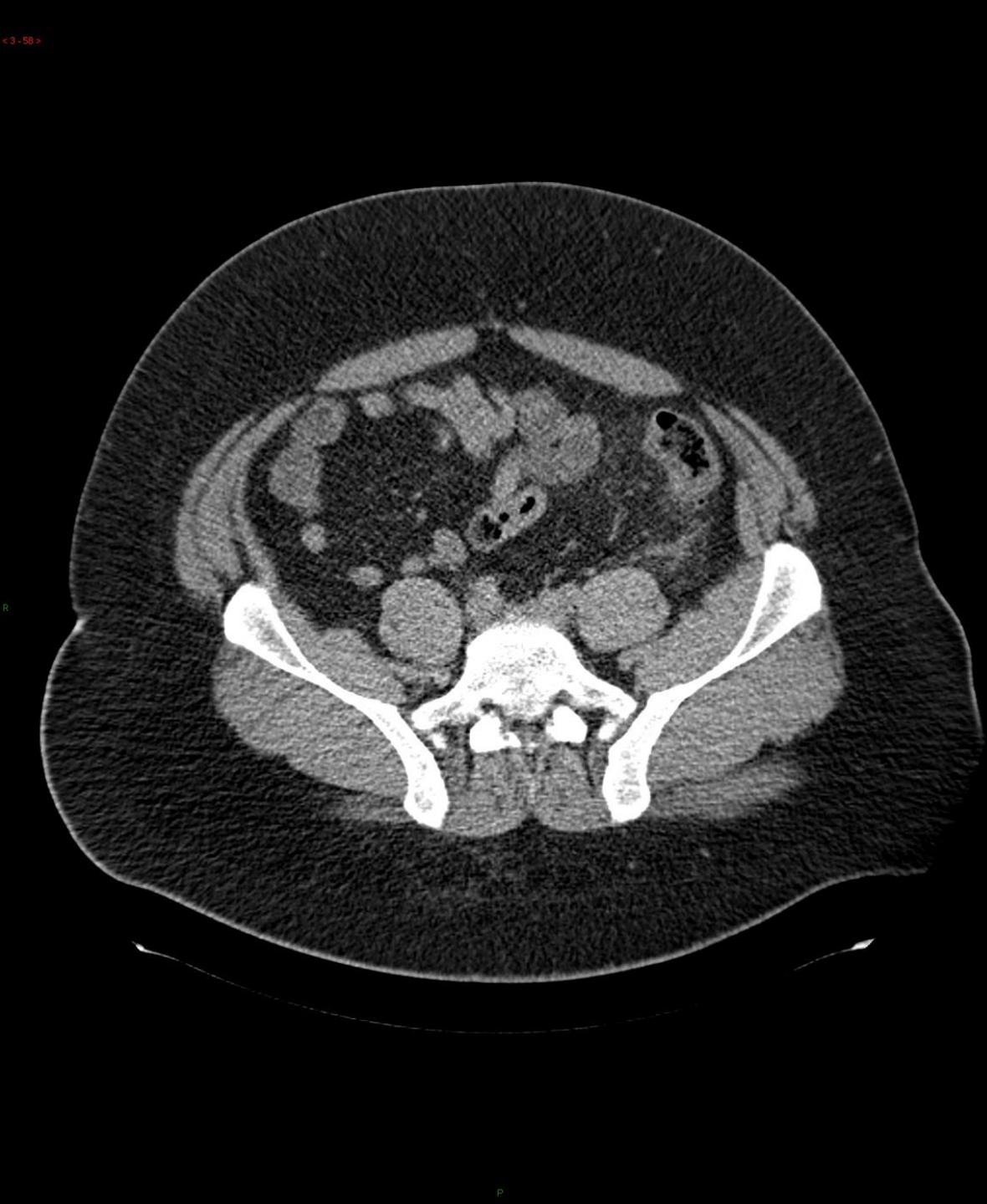


Andrew Ng

# 3D data

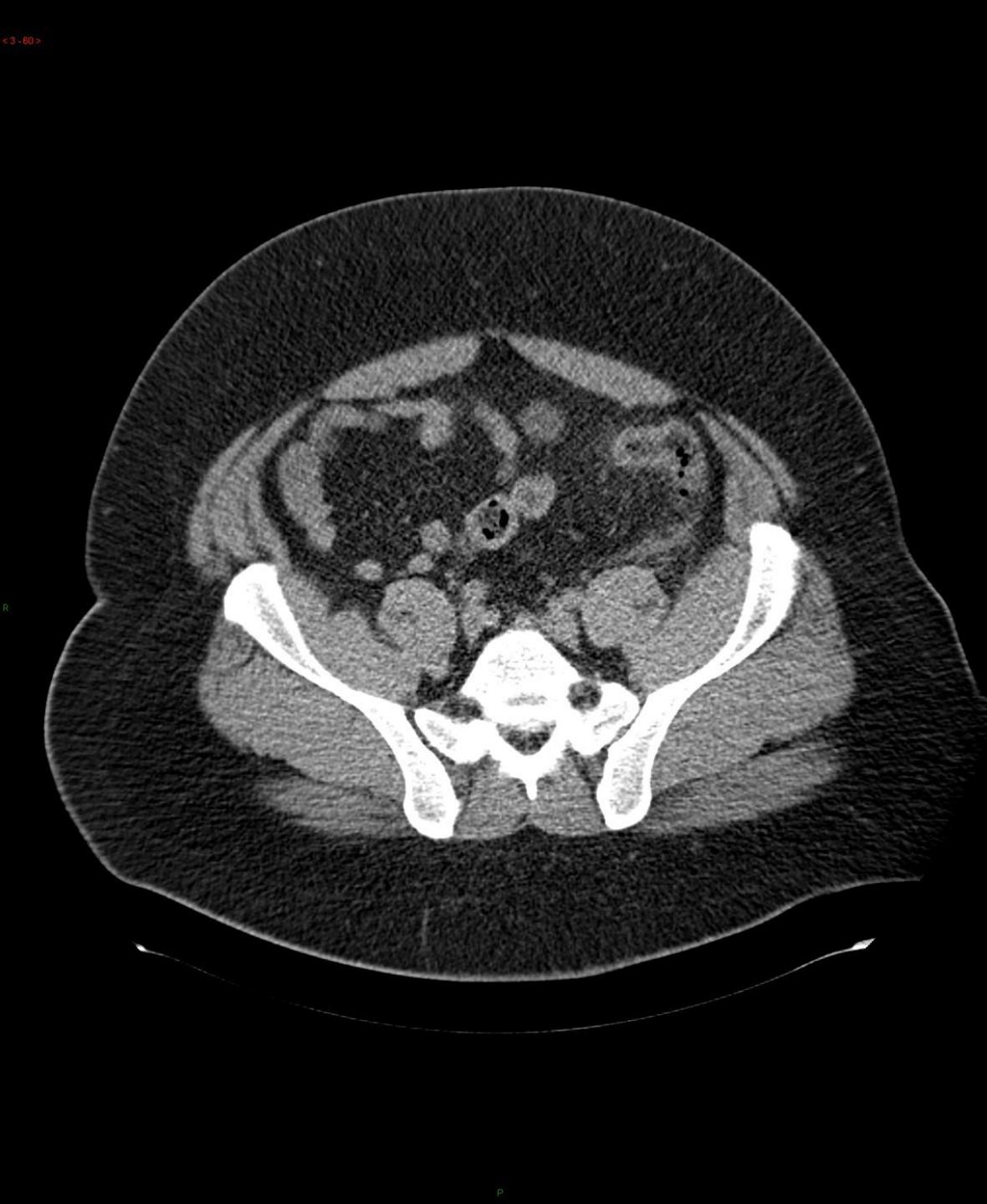


# 3D data



Andrew Ng

# 3D data



Andrew Ng

# 3D data



# 3D data



Andrew Ng

# 3D data

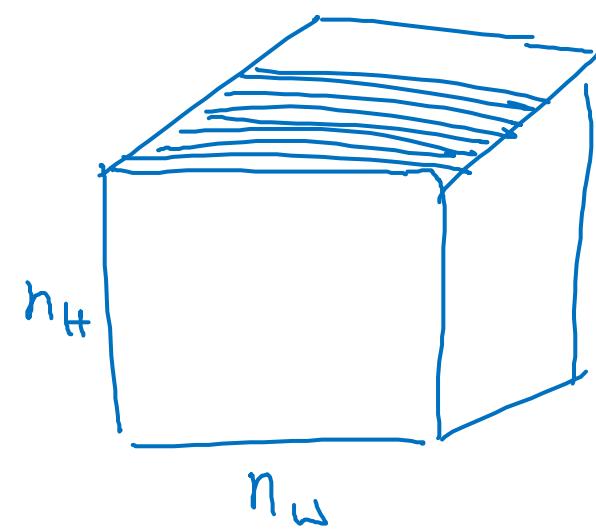


Andrew Ng

# 3D data

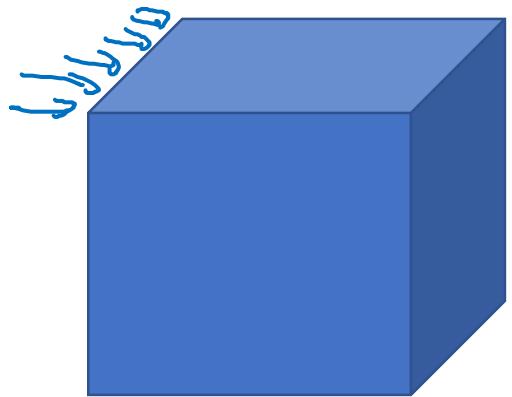


# 3D data

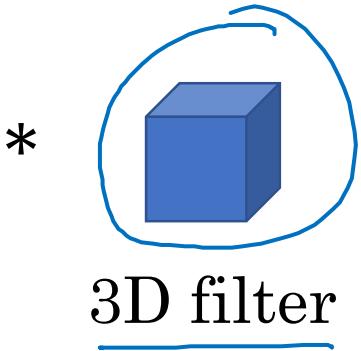


Andrew Ng

# 3D convolution



3D volume



$$\begin{array}{c} \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\ \underbrace{4 \times 14 \times 14}_{\text{Input}} \times 1 \\ * \quad \underbrace{5 \times 5 \times 5}_{\text{Filter}} \times 1 \quad 16 \text{ filters.} \\ \rightarrow 10 \times 10 \times 10 \times 16 \\ * \quad \underbrace{5 \times 5 \times 5}_{\text{Stride}} \times 16 \\ \rightarrow 6 \times 6 \times 6 \times 32 \end{array}$$

Diagram illustrating the computation of 3D convolution. The input volume is  $4 \times 14 \times 14$  (with a stride of 1). It is convolved with a  $5 \times 5 \times 5$  filter to produce 16 feature maps of size  $10 \times 10 \times 10$ . These are then convolved with another  $5 \times 5 \times 5$  filter (with a stride of 1) to produce 32 feature maps of size  $6 \times 6 \times 6$ .



deeplearning.ai

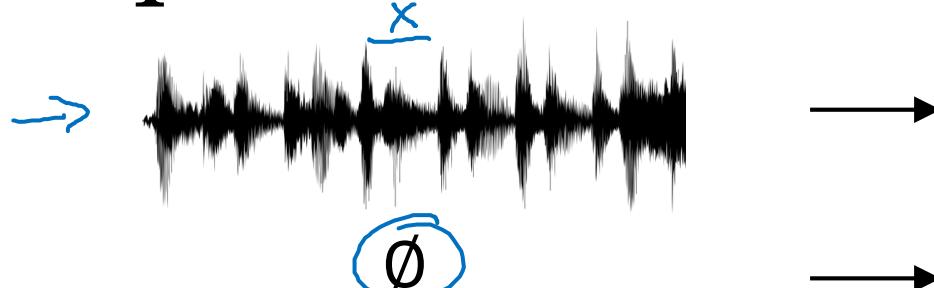
# Recurrent Neural Networks

---

Why sequence  
models?

# Examples of sequence data

Speech recognition



$y$   
“The quick brown fox jumped  
over the lazy dog.”

Music generation



Sentiment classification

“There is nothing to like  
in this movie.”



AGCCCCTGTGAGGAAC TAG

DNA sequence analysis → AGCCCCTGTGAGGAAC TAG



Machine translation

Voulez-vous chanter avec  
moi?



Do you want to sing with  
me?

Video activity recognition



Running

Name entity recognition

Yesterday, Harry Potter  
met Hermione Granger.



Yesterday, Harry Potter  
met Hermione Granger.

Andrew Ng



deeplearning.ai

# Recurrent Neural Networks

---

## Notation

# Motivating example

NLP

x: Harry Potter and Hermione Granger invented a new spell.

$\rightarrow \underline{x}^{<1>} x^{<2>} x^{<3>} \dots x^{<t>} \dots x^{<q>}$

$$T_x = q$$

$\rightarrow y:$

| | 0 | | 0 0 0 0  
 $y^{<1>} y^{<2>} y^{<3>} \dots y^{<q>}$

$$T_y = q$$

$x^{(i)<t>}$

$y^{(i)<t>}$

$$T_x^{(i)} = q$$

15

$$T_y^{(i)}$$

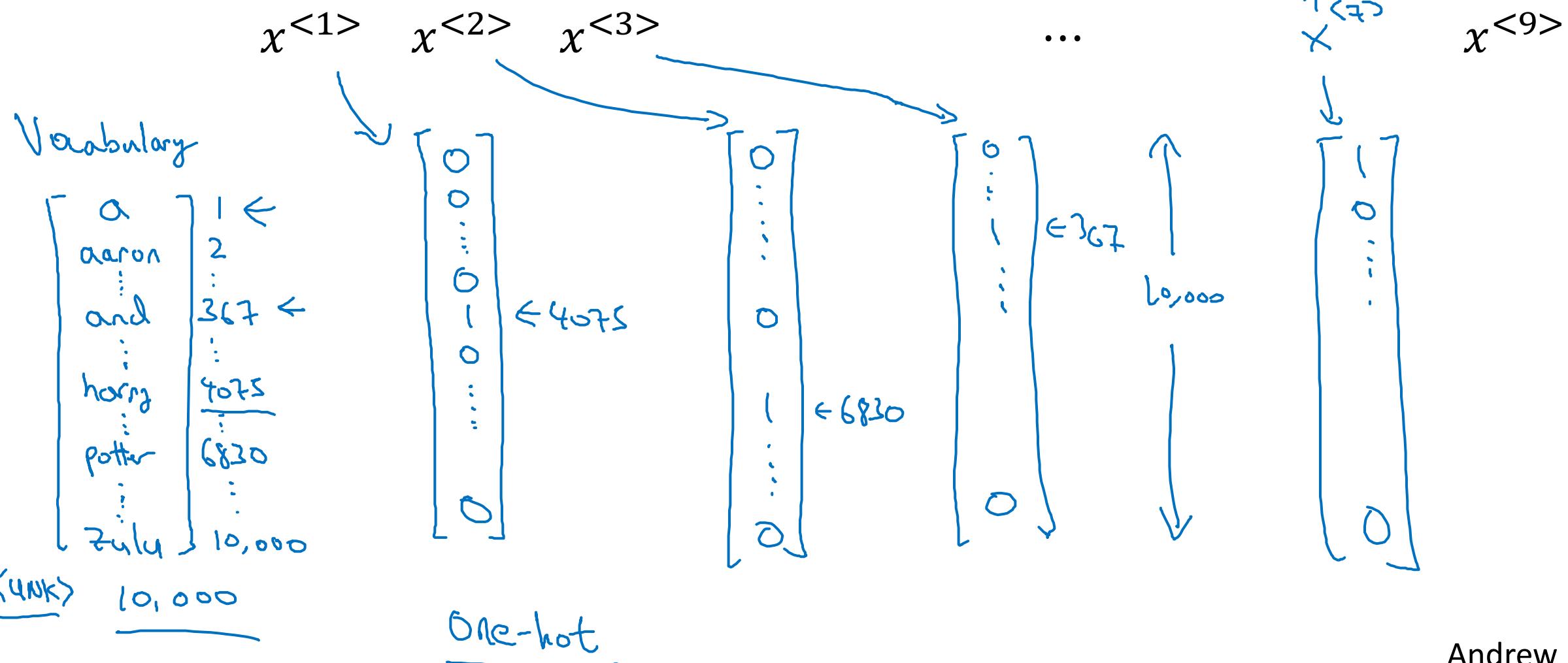
# Representing words

$$x^{<\leftrightarrow>} \quad x \rightarrow y$$

$(x, y)$

x:

Harry Potter and Hermione Granger invented a new spell.



# Representing words

x: Harry Potter and Hermione Granger invented a new spell.

$x^{<1>} \quad x^{<2>} \quad x^{<3>} \quad \dots \quad x^{<9>}$

And = 367  
Invented = 4700  
A = 1  
New = 5976  
Spell = 8376  
Harry = 4075  
Potter = 6830  
Hermione = 4200  
Gran... = 4000



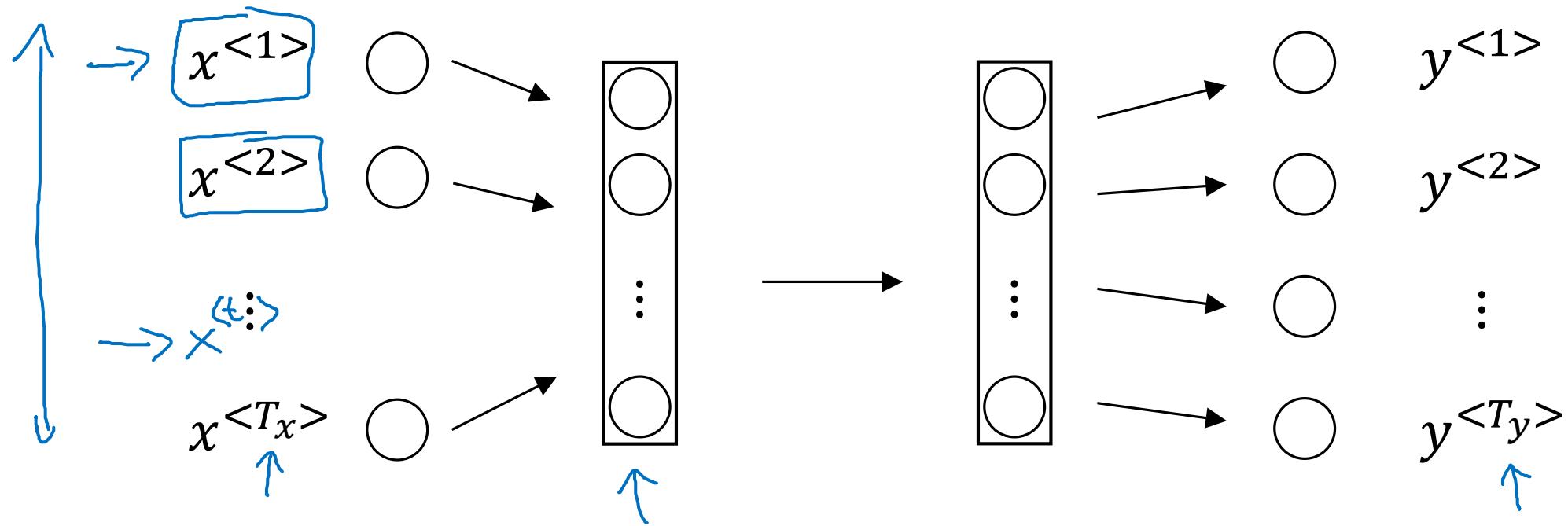
deeplearning.ai

# Recurrent Neural Networks

---

## Recurrent Neural Network Model

# Why not a standard network?

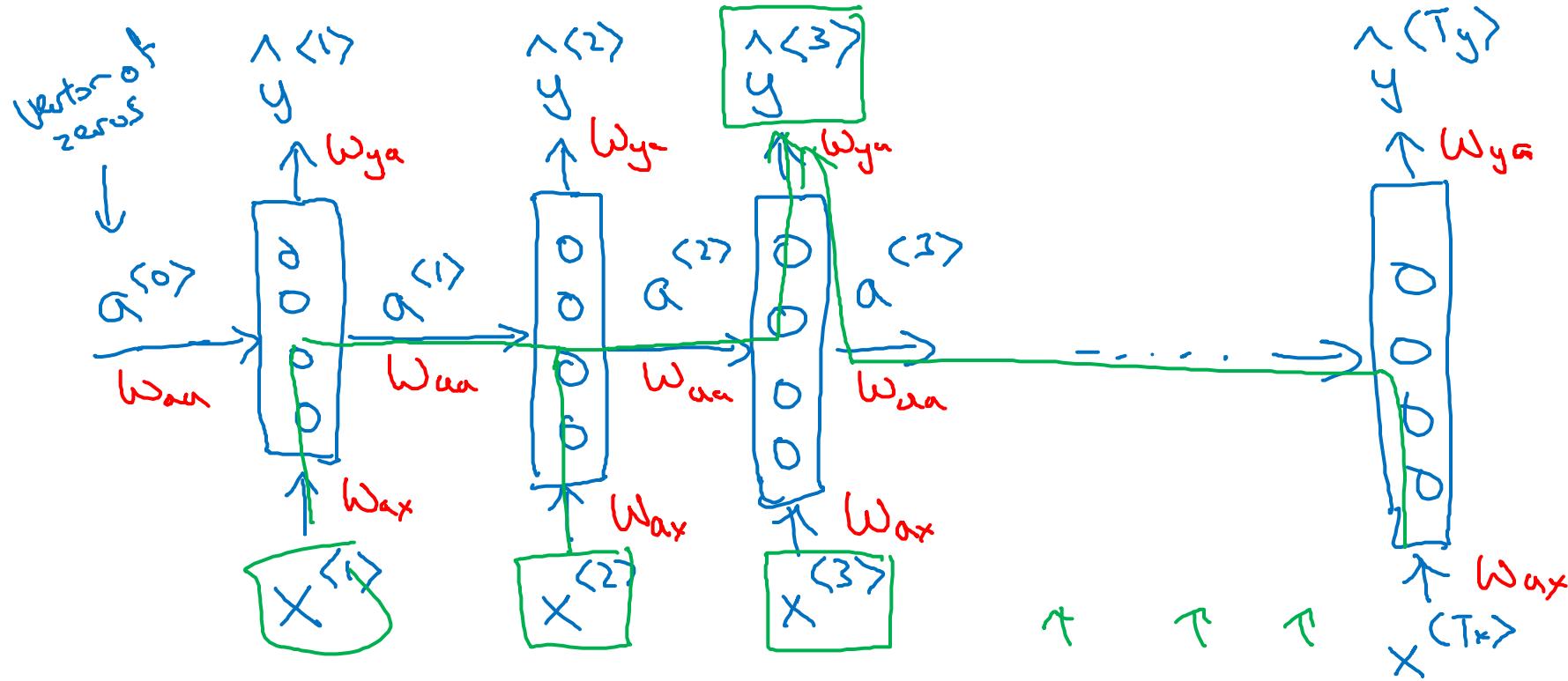


## Problems:

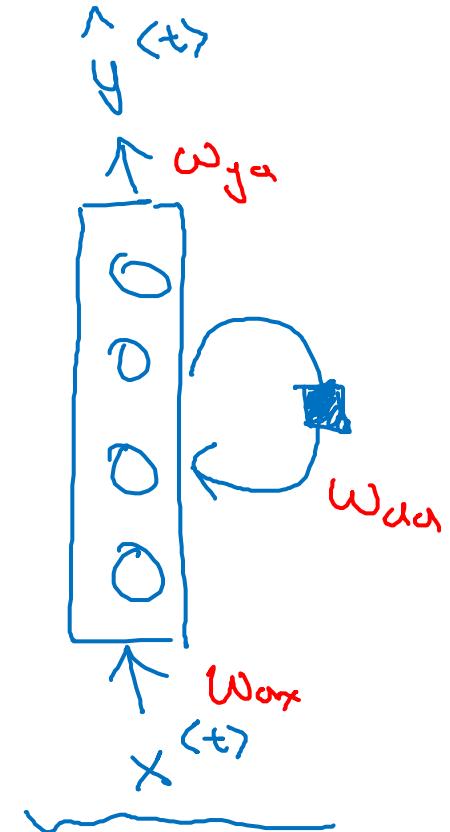
- ↳ - Inputs, outputs can be different lengths in different examples.
- ↳ - Doesn't share features learned across different positions of text.

# Recurrent Neural Networks

$$\overline{T}_x = \overline{T}_y$$



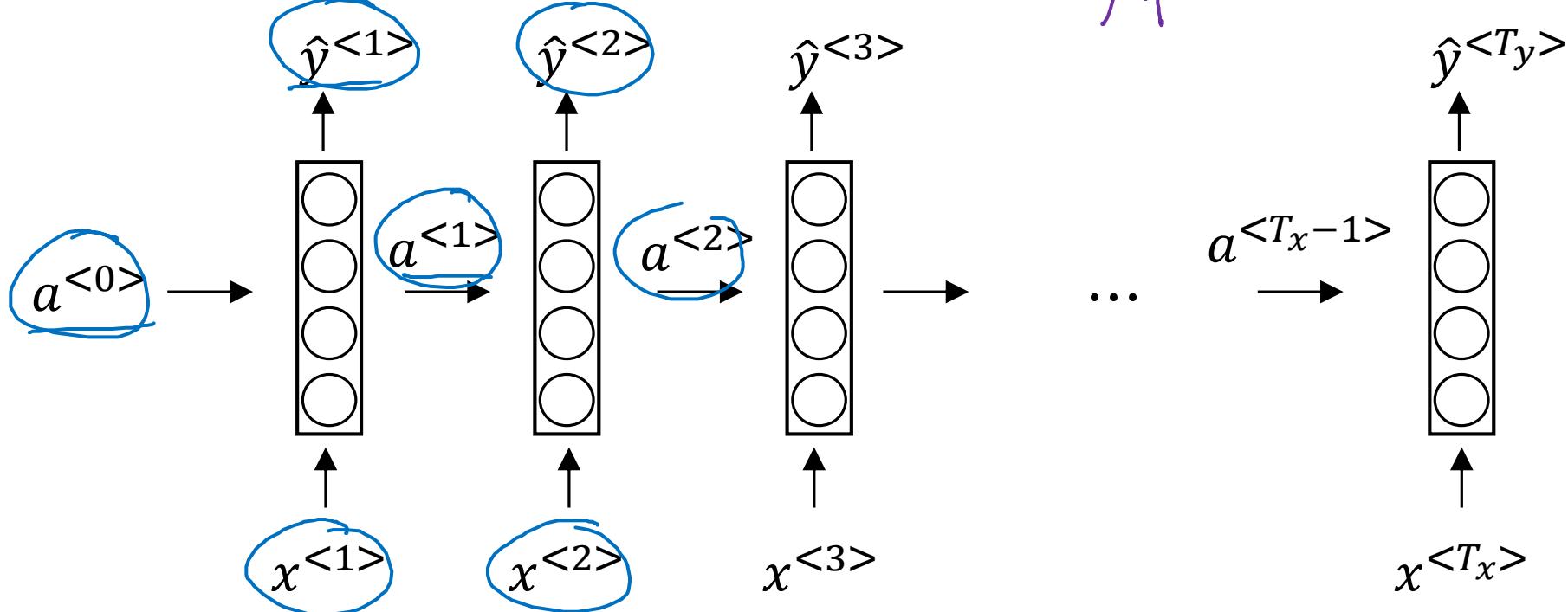
Bidirectional RNN (BRNN)



He said, "Teddy Roosevelt was a great President."

He said, "Teddy bears are on sale!"

# Forward Propagation



$$a^{<0>} = \vec{0}.$$

$$\underline{a^{<t>} = g_1(W_a a^{<t-1>} + W_x x^{<t>} + b_a)} \leftarrow \underline{\tanh \text{ or } \text{ReLU}}$$

$$\underline{\hat{y}^{<t>} = g_2(W_y a^{<t>} + b_y)} \leftarrow \text{Sigmoid}$$

$$a^{<t>} = g(W_a a^{<t-1>} + W_x x^{<t>} + b_a)$$

$$\hat{y}^{<t>} = g(W_y a^{<t>} + b_y)$$

# Simplified RNN notation

$$a^{(t)} = g(W_{aa}a^{(t-1)} + W_{ax}x^{(t)} + b_a)$$

Dimensions:  
 $W_{aa}$ :  $(100, 100)$   
 $W_{ax}$ :  $(100, 10,000)$

$$\hat{y}^{(t)} = g(W_{ya}a^{(t)} + b_y)$$

$$y^{(t)} = g(W_y a^{(t)} + b_y)$$

$\uparrow$        $\uparrow$        $\uparrow$

$$a^{(t)} = g(W_a [a^{(t-1)}, x^{(t)}] + b_a)$$

$$\begin{bmatrix} W_{aa} & W_{ax} \end{bmatrix} = W_a$$

$W_{aa}$ :  $(100, 100)$   
 $W_{ax}$ :  $(100, 10,000)$

$$[a^{(t-1)}, x^{(t)}] = \begin{bmatrix} a^{(t-1)} \\ x^{(t)} \end{bmatrix}$$

$a^{(t-1)}$ :  $100$   
 $x^{(t)}$ :  $10,000$

$$[W_{aa}; W_{ax}] \begin{bmatrix} a^{(t-1)} \\ x^{(t)} \end{bmatrix} = W_{aa}a^{(t-1)} + W_{ax}x^{(t)}$$



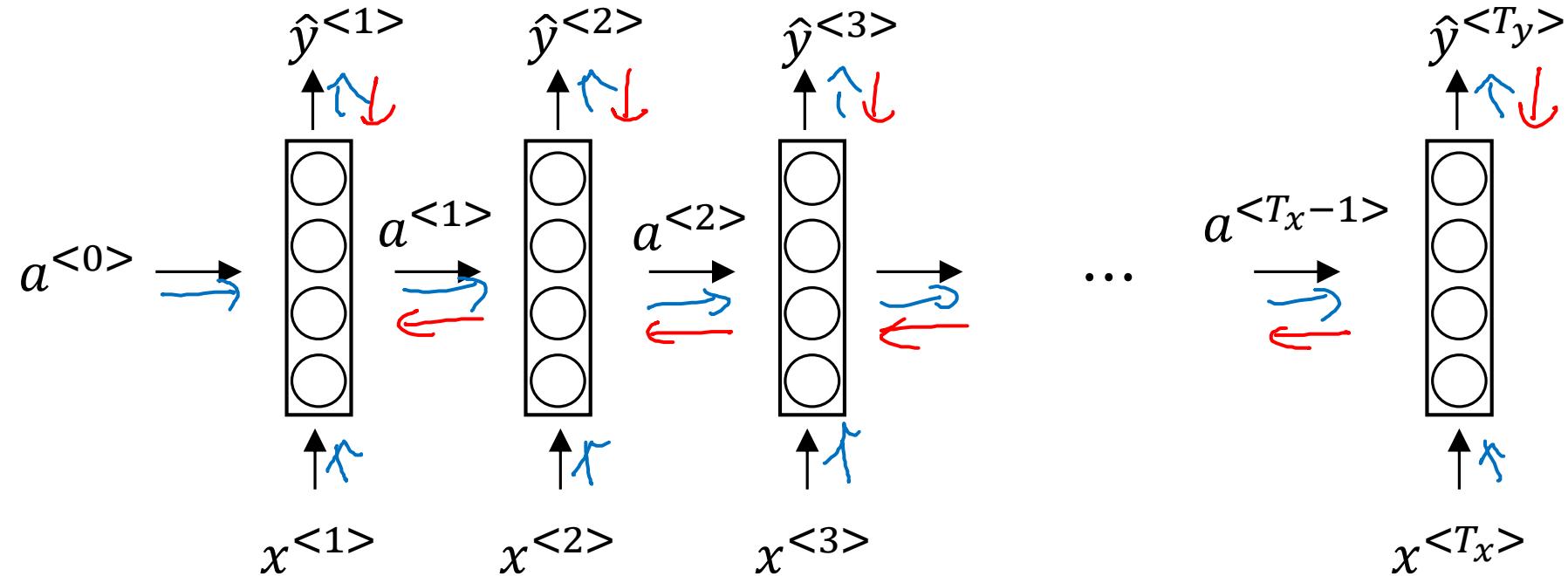
deeplearning.ai

# Recurrent Neural Networks

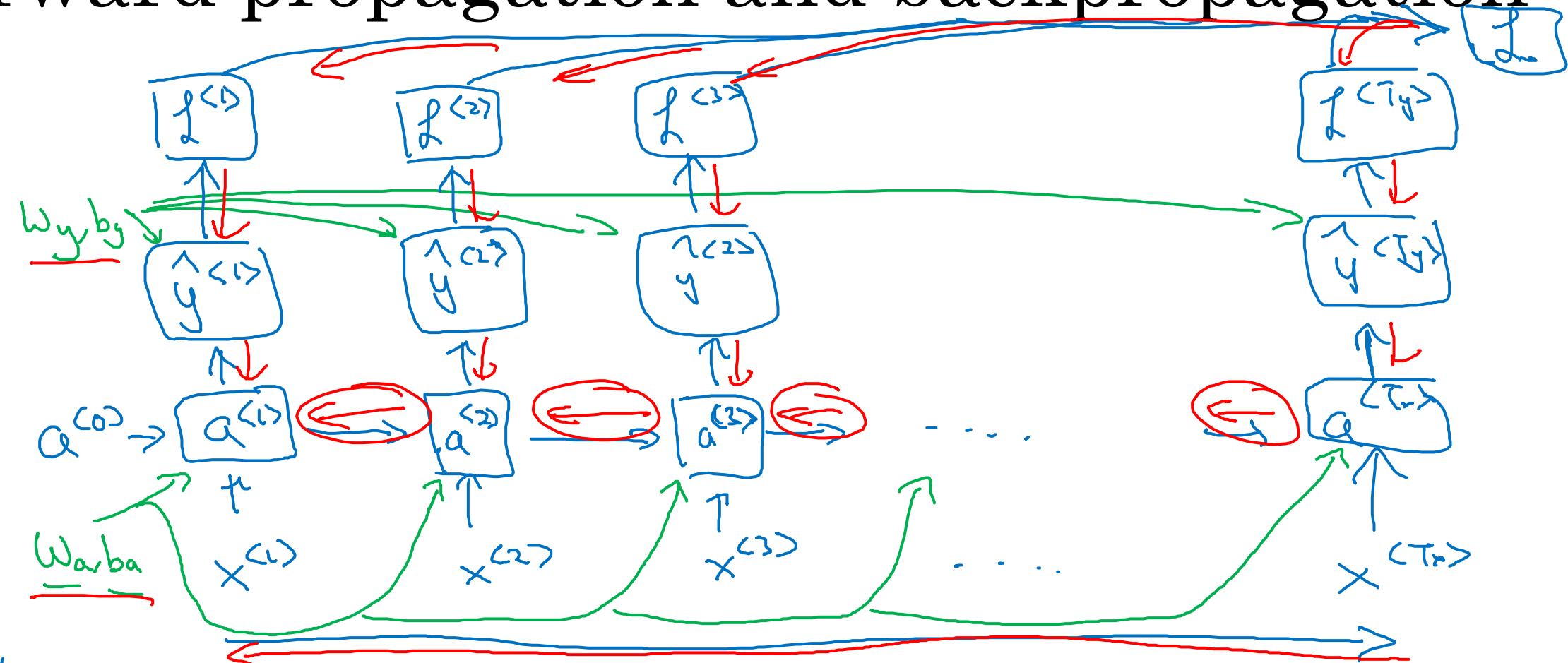
---

## Backpropagation through time

# Forward propagation and backpropagation



# Forward propagation and backpropagation



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

$$L(\hat{y}, y) = \sum_{t=1}^{T_y} L^{(t)}(\hat{y}^{(t)}, y^{(t)})$$

Backpropagation through time



deeplearning.ai

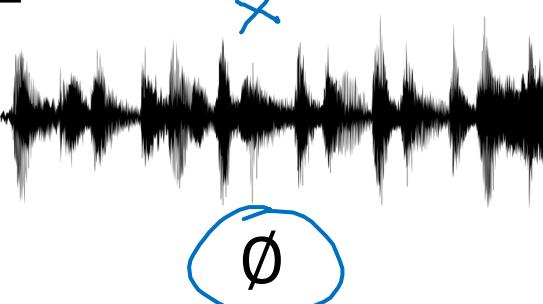
# Recurrent Neural Networks

---

## Different types of RNNs

# Examples of sequence data

Speech recognition



$T_x$        $T_y$   
 $y$

“The quick brown fox jumped  
over the lazy dog.”

Music generation



Sentiment classification

“There is nothing to like  
in this movie.”



DNA sequence analysis

AGCCCCTGTGAGGAAC TAG



AGCCCCTGTGAGGAAC TAG

Machine translation

Voulez-vous chanter avec  
moi?



Do you want to sing with  
me?

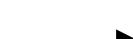
Video activity recognition



Running

Name entity recognition

Yesterday, Harry Potter  
met Hermione Granger.

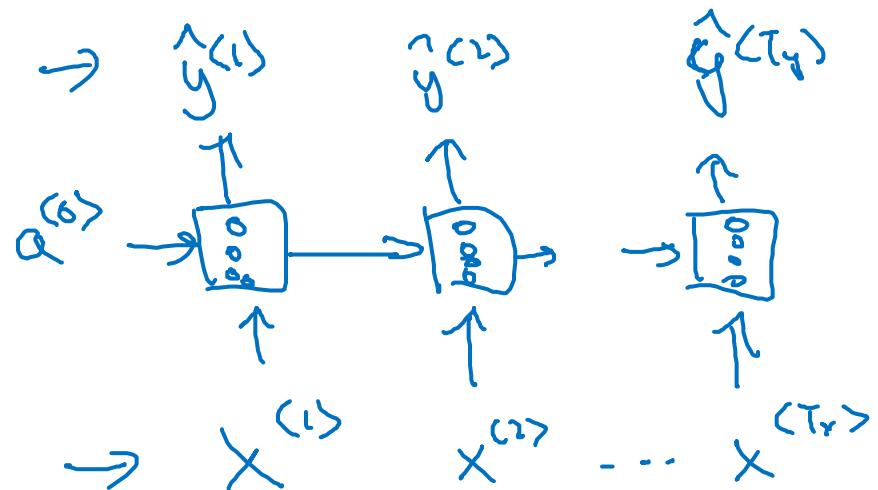


Yesterday, Harry Potter  
met Hermione Granger.

Andrew Ng

# Examples of RNN architectures

$$T_x = T_y$$

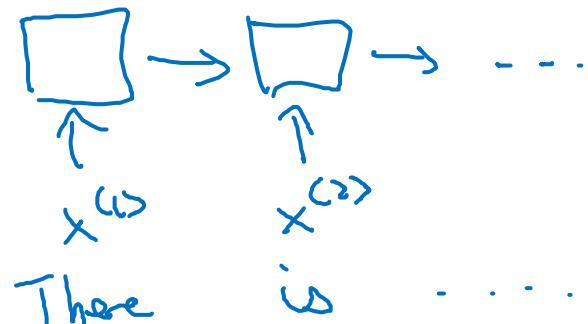


Many-to-many

Sentiment classification

$x = \text{text}$

$y = 0/1 \quad 1 \dots 5$



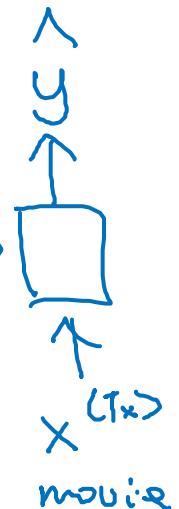
These

$x^{(1)}$

$x^{(2)}$

$\dots$

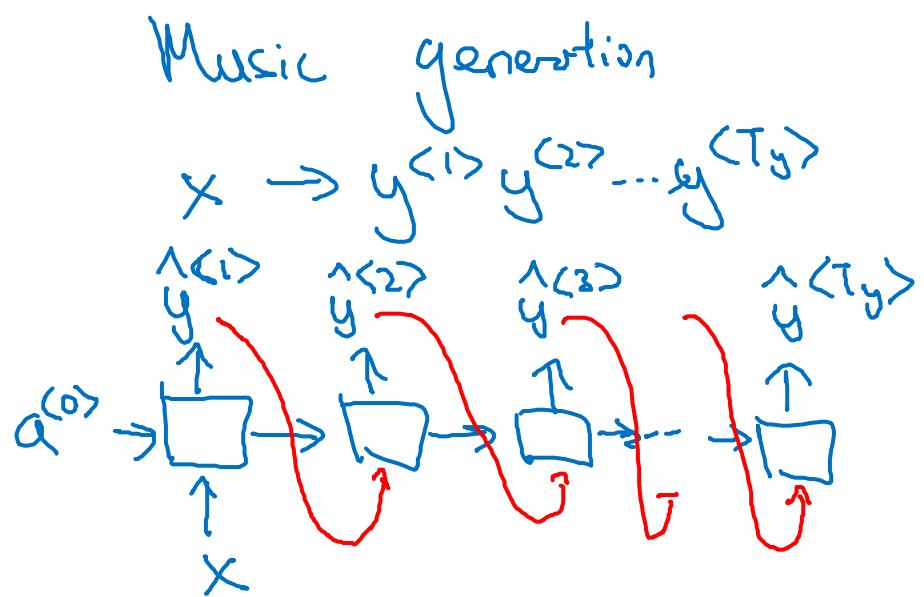
Many - to - one



One-to-one

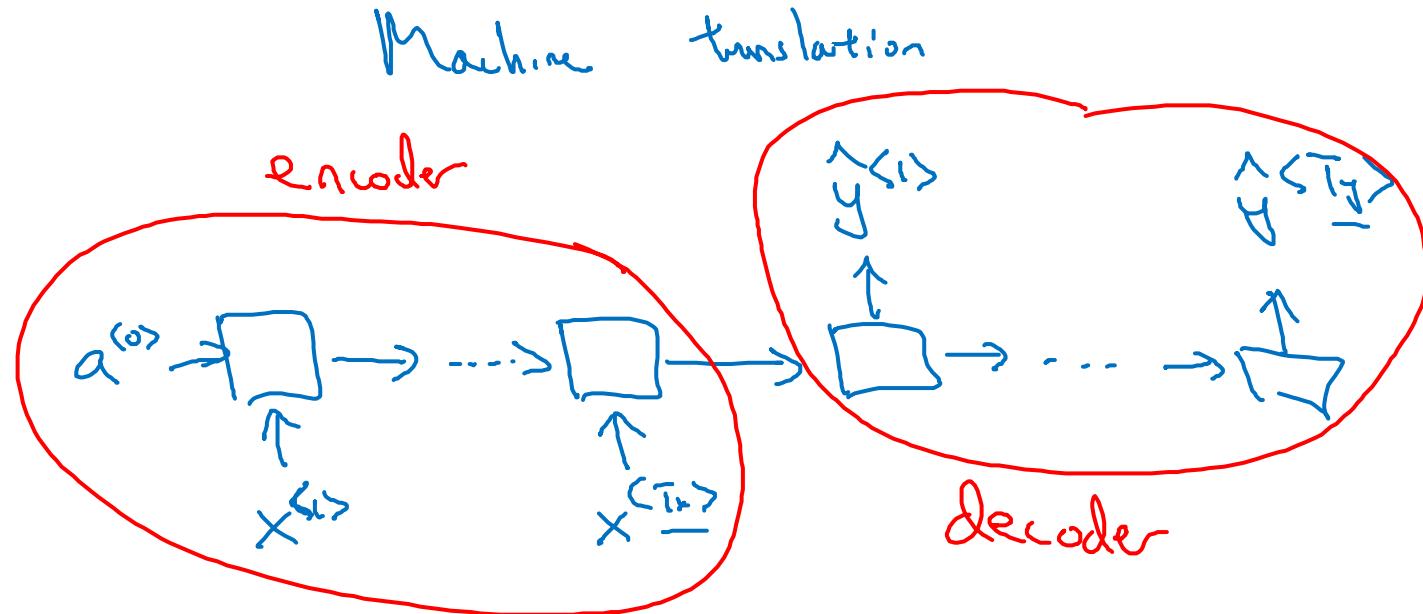


# Examples of RNN architectures



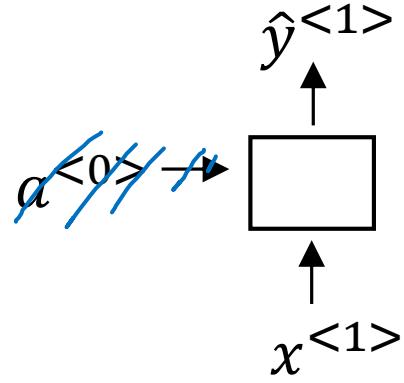
One-to-many

$$x = \phi$$

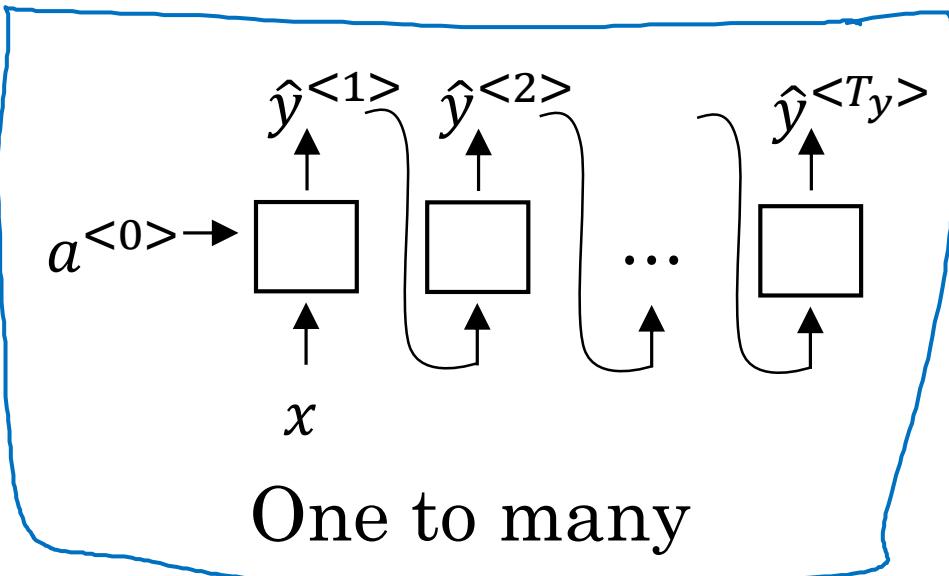


Many - to - many

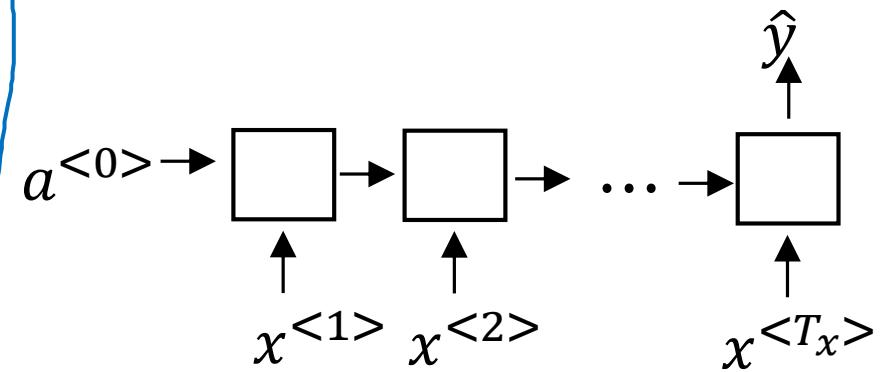
# Summary of RNN types



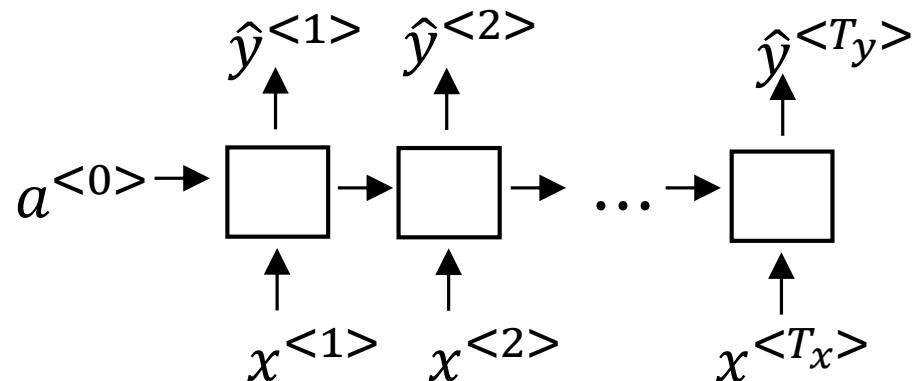
One to one



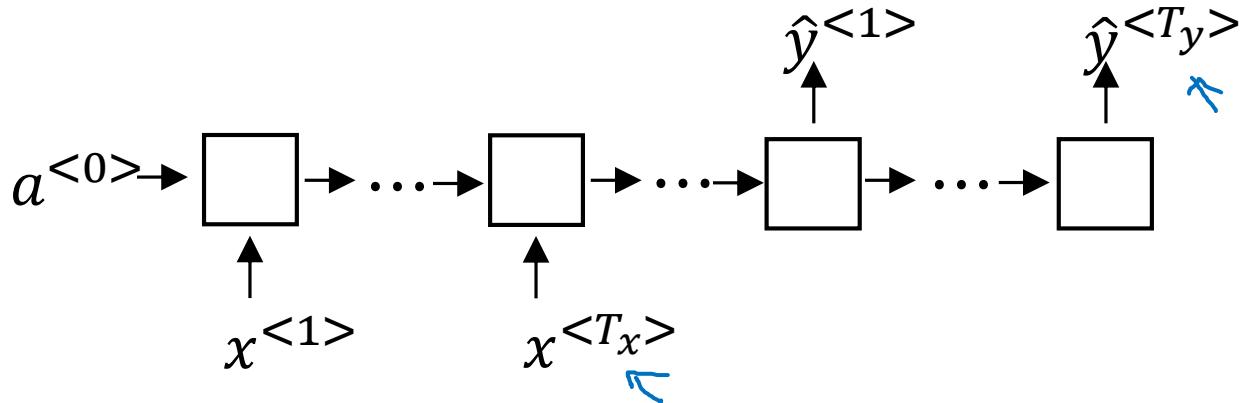
One to many



Many to one



Many to many  
 $T_x = T_y$



Many to many  
 $T_x < T_y$



deeplearning.ai

# Recurrent Neural Networks

---

## Language model and sequence generation

# What is language modelling?

Speech recognition

The apple and pair salad.

→ The apple and pear salad.

$$P(\text{The apple and pair salad}) = 3.2 \times 10^{-3}$$

$$P(\text{The apple and pear salad}) = 5.7 \times 10^{-10}$$

$$P(\text{Sentence}) = ?$$

$$P(y^{(1)}, y^{(2)}, \dots, y^{(T)})$$

# Language modelling with an RNN

Training set: large corpus of english text.

Tokenize

Cats average 15 hours of sleep a day.  $\downarrow \langle \text{EOS} \rangle$

$y^{<1>}$        $y^{<2>}$        $y^{(3)}$

$x^{<t>} = y^{<t-1>}$

...

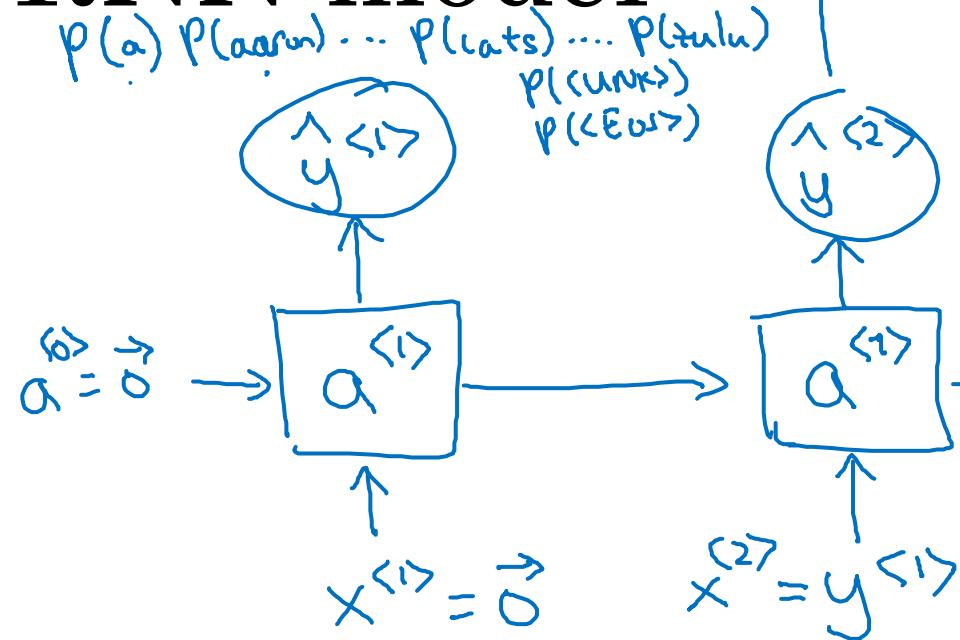
$y^{(8)}$        $y^{(9)}$

The Egyptian ~~Mau~~ is a bread of cat.  $\langle \text{EOS} \rangle$

$\langle \text{UNK} \rangle$

10,000

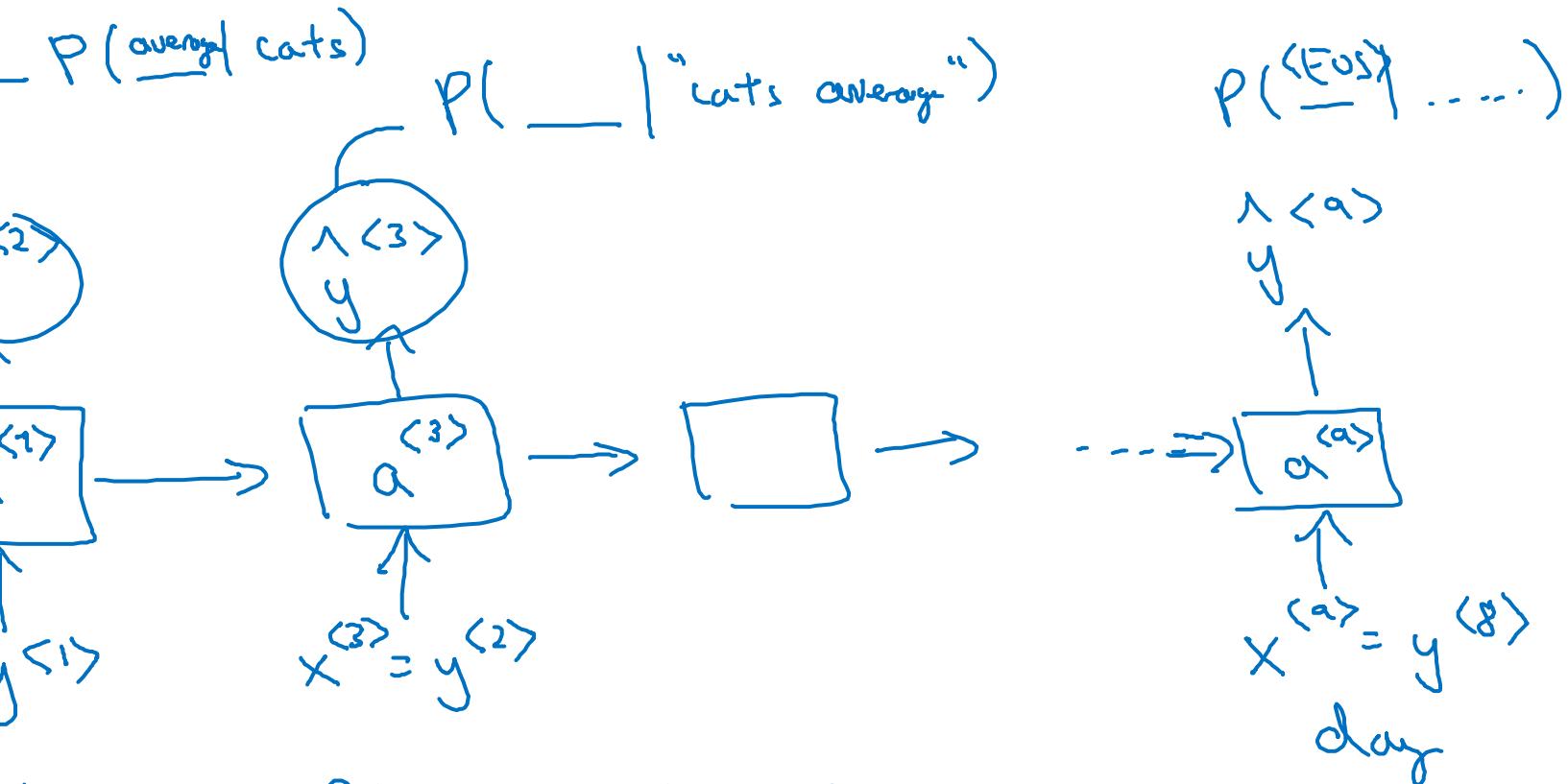
# RNN model



$\rightarrow$  Cats average 15 hours of sleep a day.  $<\text{EOS}>$

$$\mathcal{L}(\hat{y}^{(t)}, y^{(t)}) = - \sum_i y_i^{(t)} \log \hat{y}_i^{(t)}$$

$$\mathcal{L} = \sum_t \mathcal{L}^{(t)}(\hat{y}^{(t)}, y^{(t)})$$



$$P(y^{(1)}, y^{(2)}, y^{(3)}) \leftarrow$$

$$= \frac{p(y^{(1)}) p(y^{(2)} | y^{(1)})}{p(y^{(3)} | y^{(1)}, y^{(2)})}$$



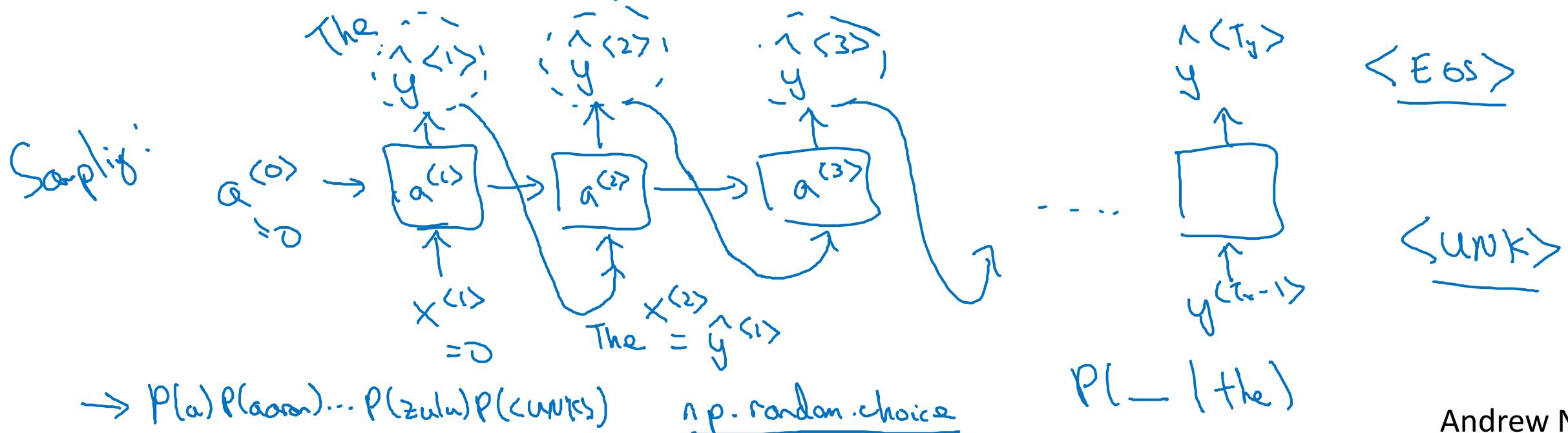
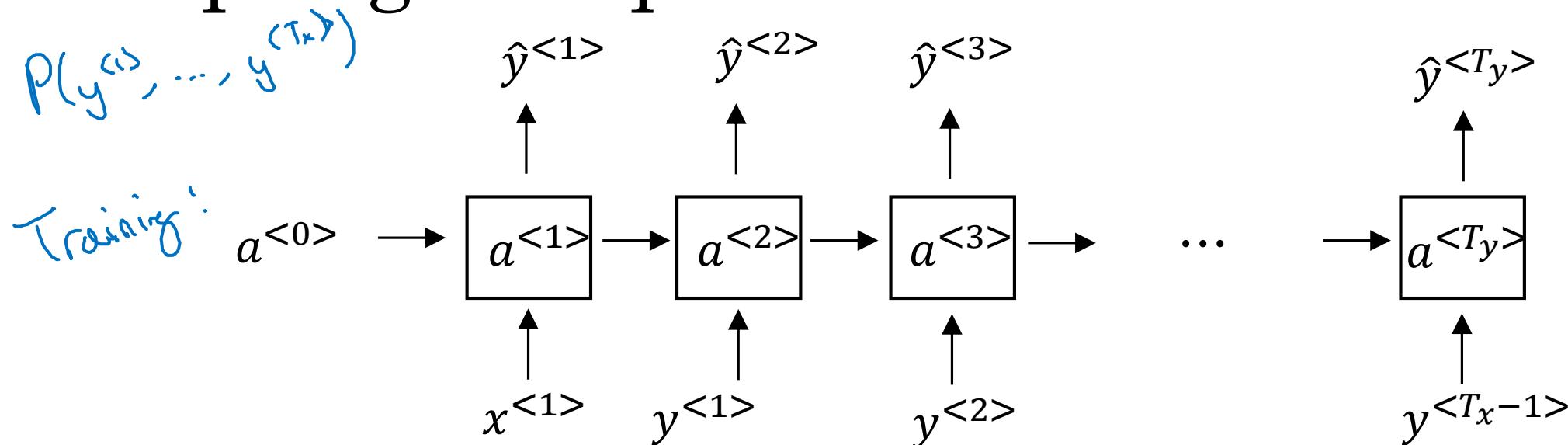
deeplearning.ai

# Recurrent Neural Networks

---

Sampling novel  
sequences

# Sampling a sequence from a trained RNN



# Character-level language model

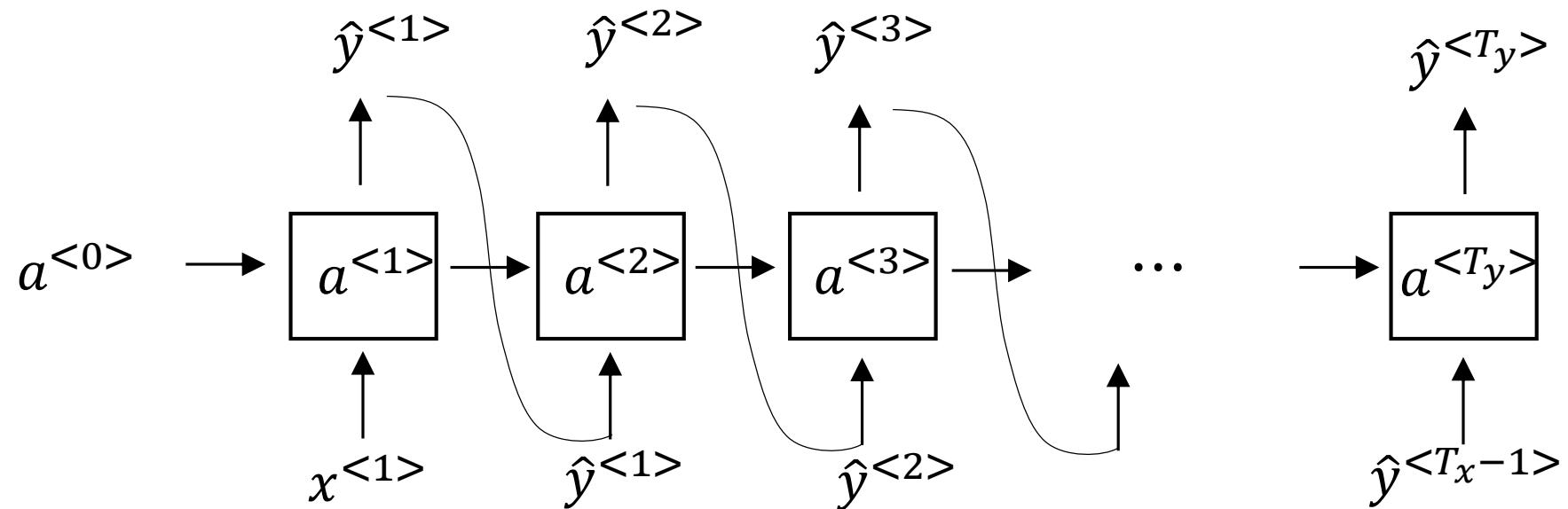
→ Vocabulary = [a, aaron, ..., zulu, <UNK>] ↪

$\rightarrow \text{Vocabulary} = [a, b, c, \dots, z, \cup, \circ, \rightarrow, ;, 0, \dots, 9, A, \dots, Z]$

$$y^{(0)} - y^{(1)} = \frac{y^{(2)}}{y^{(3)}}$$

Cat average  
↑ ↑ ↑ ↑ ...

May



# Sequence generation

## News

President enrique peña nieto, announced  
sench's sulk former coming football langston  
paring.

“I was not at all surprised,” said hich langston.

“Concussion epidemic”, to be examined. ←

The gray football the told some and this has on  
the uefa icon, should money as.

## Shakespeare

The mortal moon hath her eclipse in love.  
And subject of this thou art another this fold.

When lesser be my love to me see sabl's.  
For whose are ruse of mine eyes heaves.



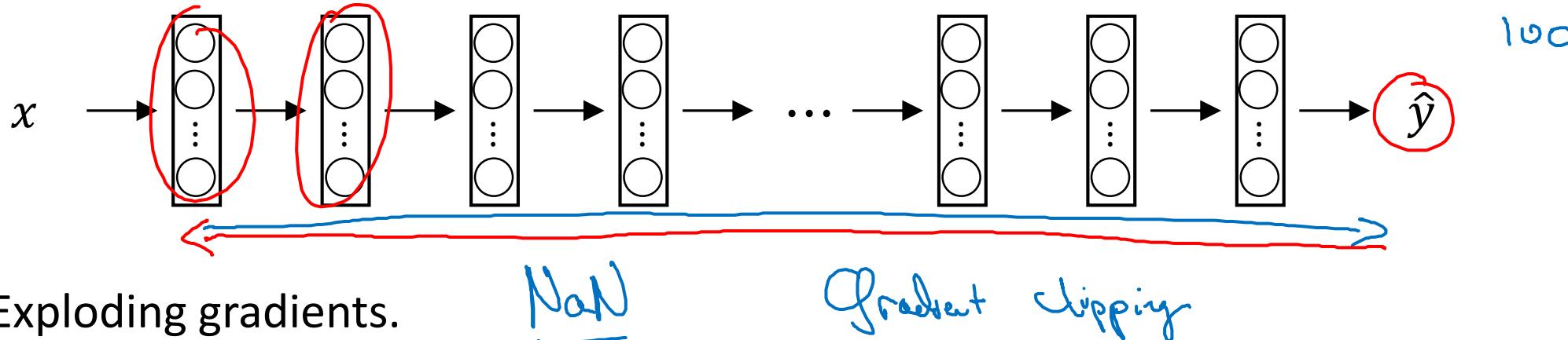
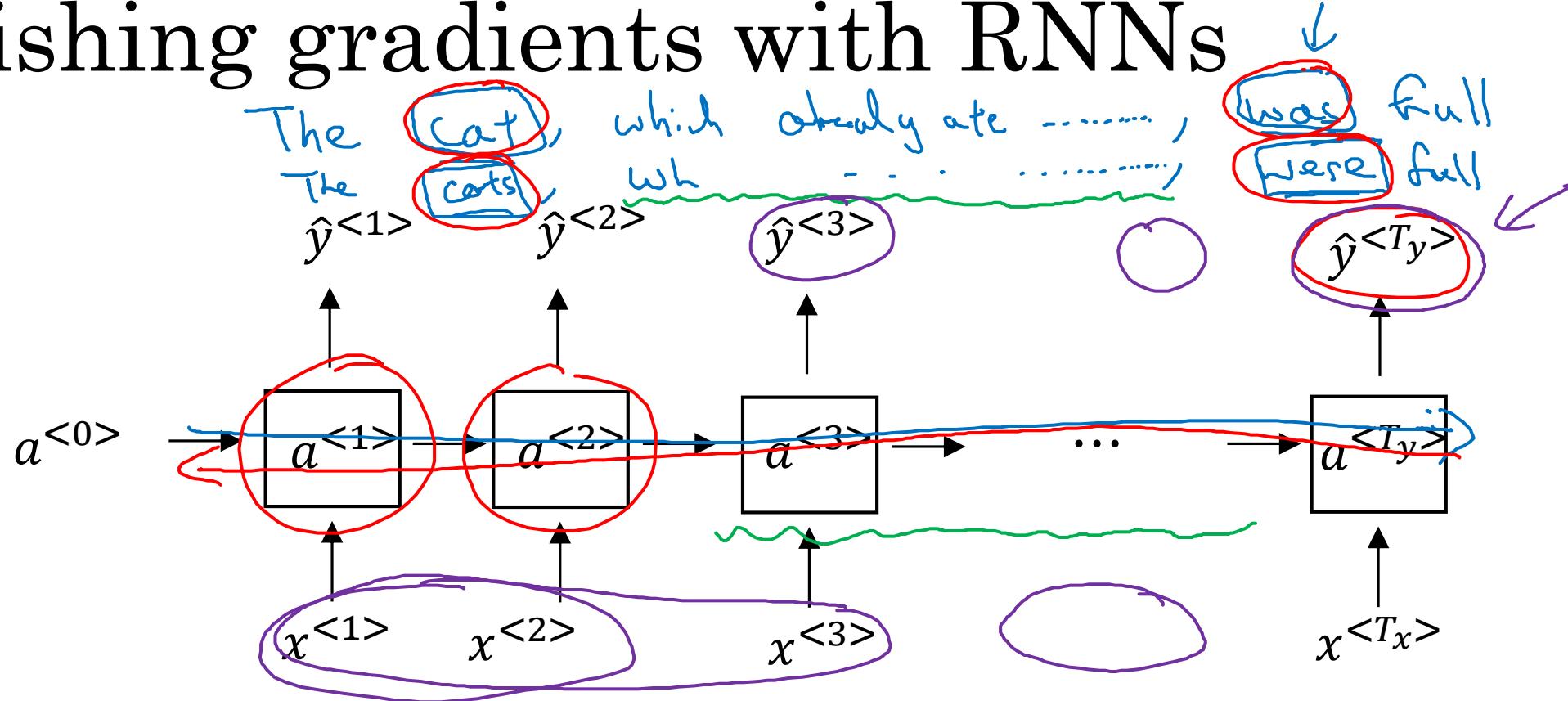
deeplearning.ai

# Recurrent Neural Networks

---

## Vanishing gradients with RNNs

# Vanishing gradients with RNNs





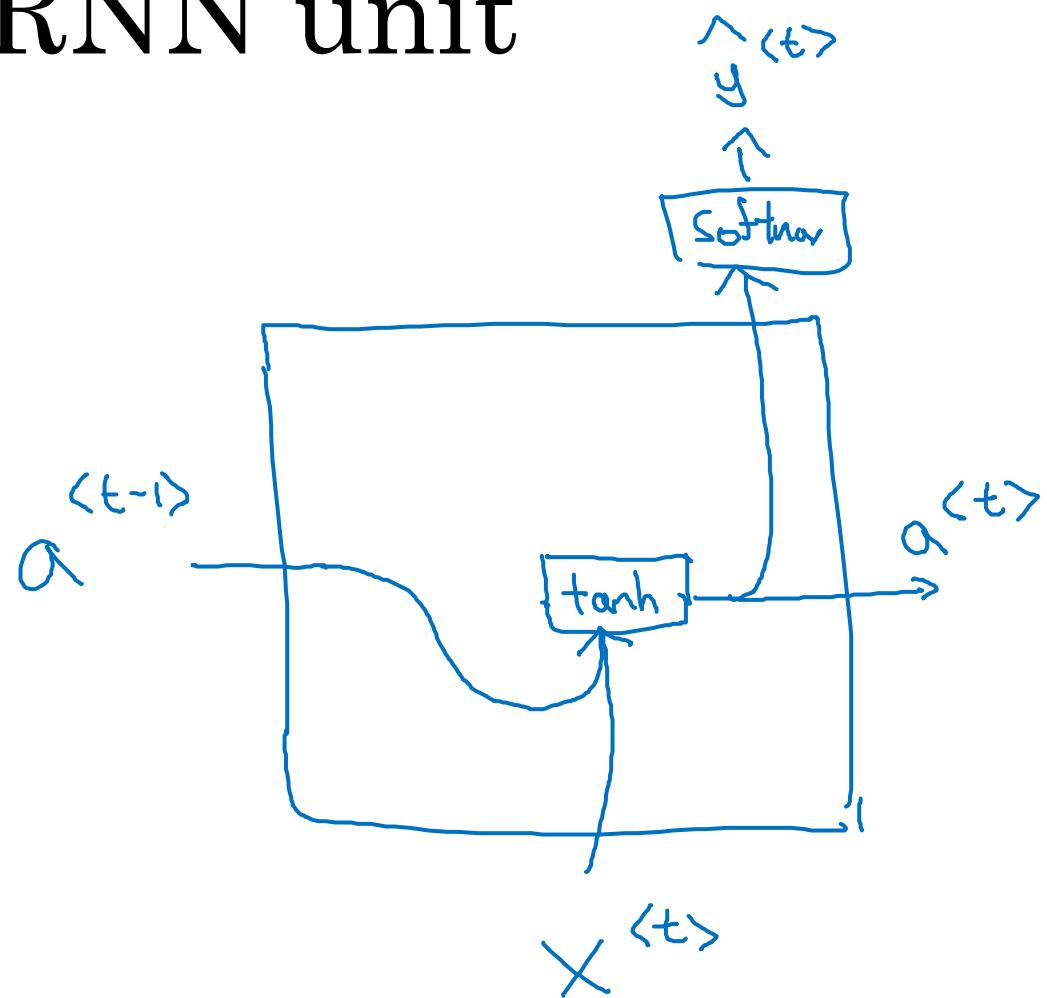
deeplearning.ai

# Recurrent Neural Networks

---

## Gated Recurrent Unit (GRU)

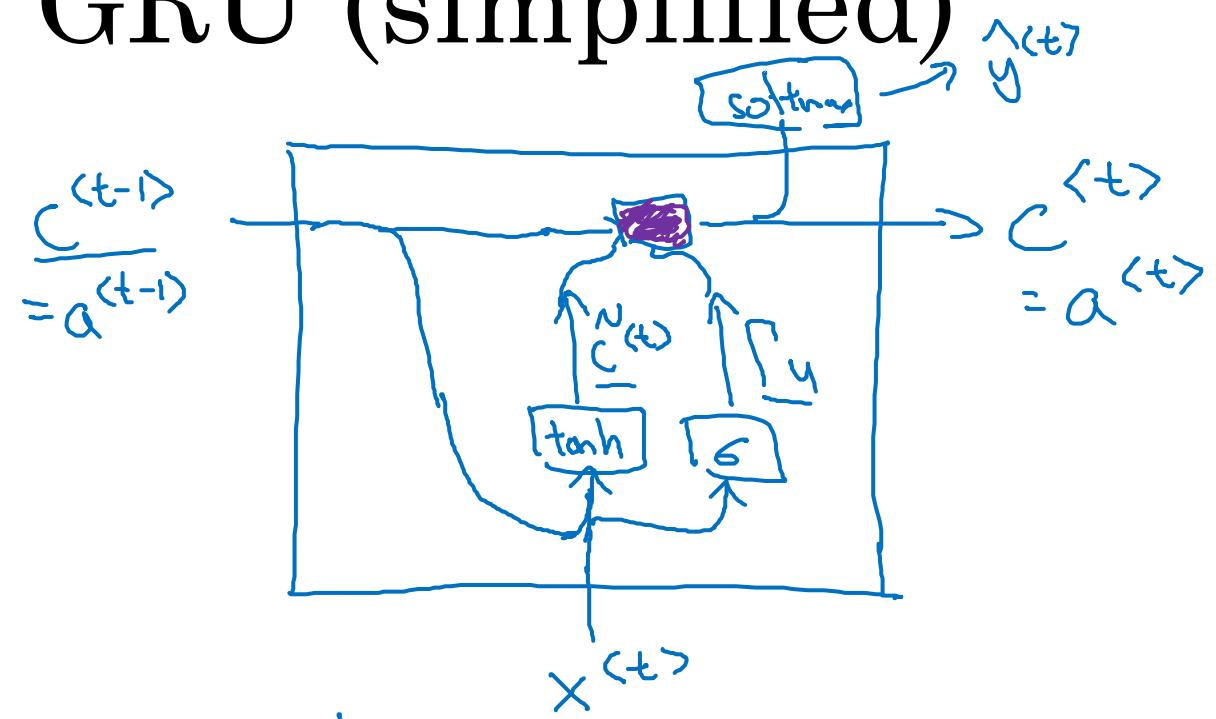
# RNN unit



$$\underline{a^{(t)}} = g(W_a[\underline{a^{(t-1)}, x^{(t)}}] + b_a)$$

A handwritten equation for the hidden state  $\underline{a}^{(t)}$ . Above the equation, the word "tanh" is written with a downward arrow pointing to the  $g$  function. The equation itself is  $\underline{a}^{(t)} = g(W_a[\underline{a}^{(t-1)}, x^{(t)}] + b_a)$ . The  $\underline{\quad}$  symbol is used to denote vectors.

# GRU (simplified)



$\Gamma_u = 1$

$\Gamma_u = 0$   $\Gamma_u = 0$   $\Gamma_u = 0$  ...

The cat, which already ate ..., was full.

$C = \text{memory cell}$

$C^{(t)} = \alpha^{(t)}$

$N_c^{(t)} = \tanh(W_c [c^{(t-1)}, x^{(t)}] + b_c)$

$\Gamma_u^{(t)} = \sigma(W_u [c^{(t-1)}, x^{(t)}] + b_u)$

"update"

$C^{(t)} = \Gamma_u * N_c^{(t)} + (1 - \Gamma_u) * C^{(t-1)}$

element-wise Gate

$\Gamma_u = 0.000001$

[Cho et al., 2014. On the properties of neural machine translation: Encoder-decoder approaches]

[Chung et al., 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling]

Andrew Ng

# Full GRU

$$\tilde{h} \quad \tilde{c}^{<t>} = \tanh(W_c[\tilde{c}_r^{<t-1>}, x^{<t>}] + b_c)$$

$$u \quad \Gamma_u = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$$

$$r \quad \Gamma_r = \sigma(W_r[c^{<t-1>}, x^{<t>}] + b_r)$$

LSTM

$$h \quad c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1 - \Gamma_u) * c^{<t-1>}$$

The cat, which ate already, was full.



deeplearning.ai

# Recurrent Neural Networks

---

LSTM (long short  
term memory) unit

# GRU and LSTM

## GRU

$$\tilde{c}^{<t>} = \tanh(W_c[\Gamma_r * \underline{\tilde{c}}^{<t-1>}, x^{<t>}] + b_c)$$

$$\underline{\Gamma_u} = \sigma(W_u[c^{<t-1>}, x^{<t>}] + b_u)$$

$$\underline{\Gamma_r} = \sigma(W_r[c^{<t-1>}, x^{<t>}] + b_r)$$

$$c^{<t>} = \underline{\Gamma_u} * \underline{\tilde{c}}^{<t>} + (1 - \underline{\Gamma_u}) * c^{<t-1>}$$

$$a^{<t>} = c^{<t>}$$



## LSTM

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$

$$\underline{\Gamma_u} = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$

$$\underline{\Gamma_f} = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$

$$\underline{\Gamma_o} = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$

$$c^{<t>} = \underline{\Gamma_u} * \underline{\tilde{c}}^{<t>} + \underline{\Gamma_f} * \underline{c}^{<t-1>}$$

$$a^{<t>} = \underline{\Gamma_o} * c^{<t>}$$

# LSTM in pictures

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$

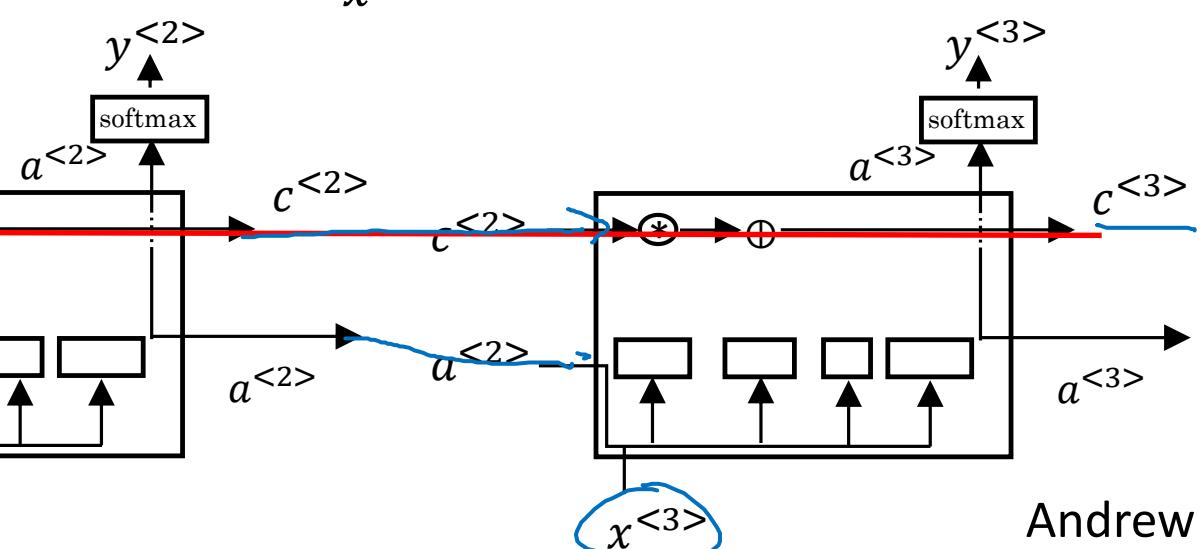
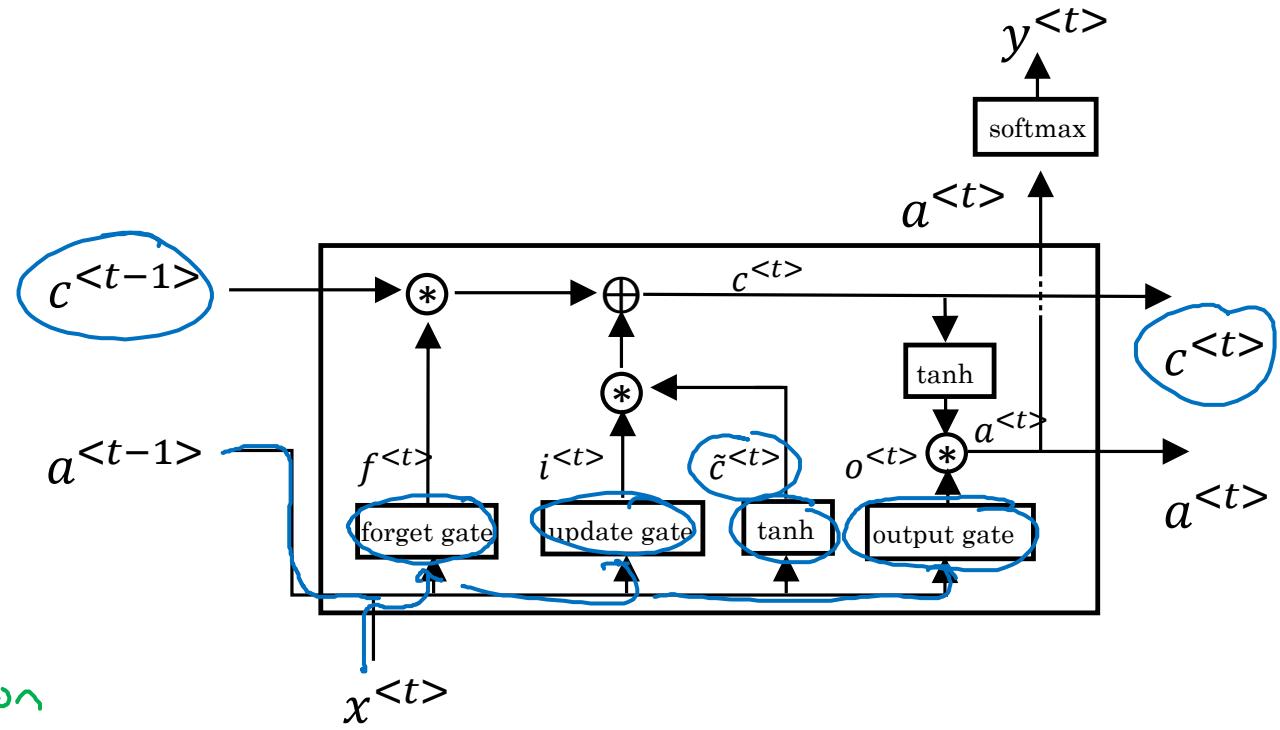
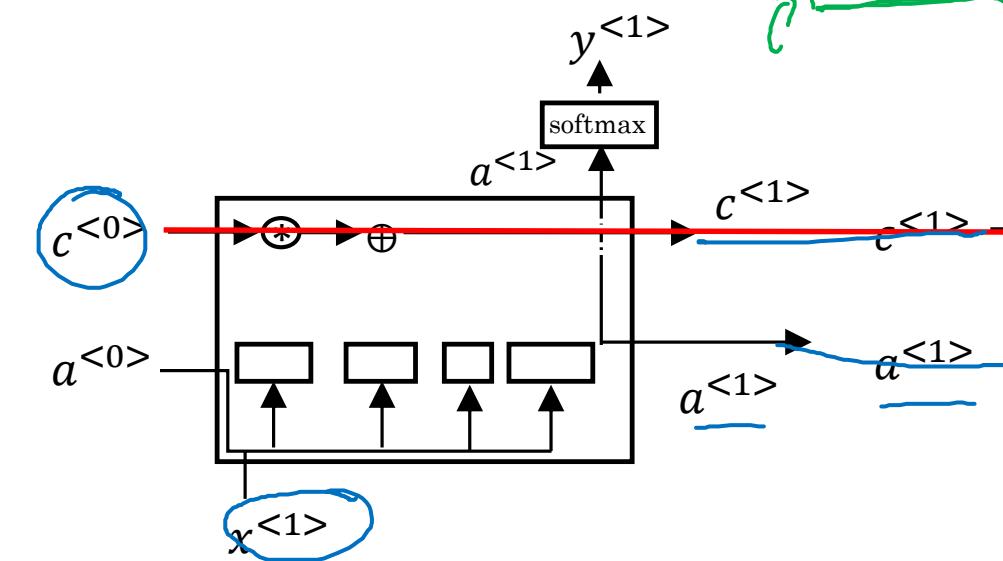
$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$

$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$

$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$

$$a^{<t>} = \Gamma_o * c^{<t>}$$

peephole connection



Andrew Ng



deeplearning.ai

# Recurrent Neural Networks

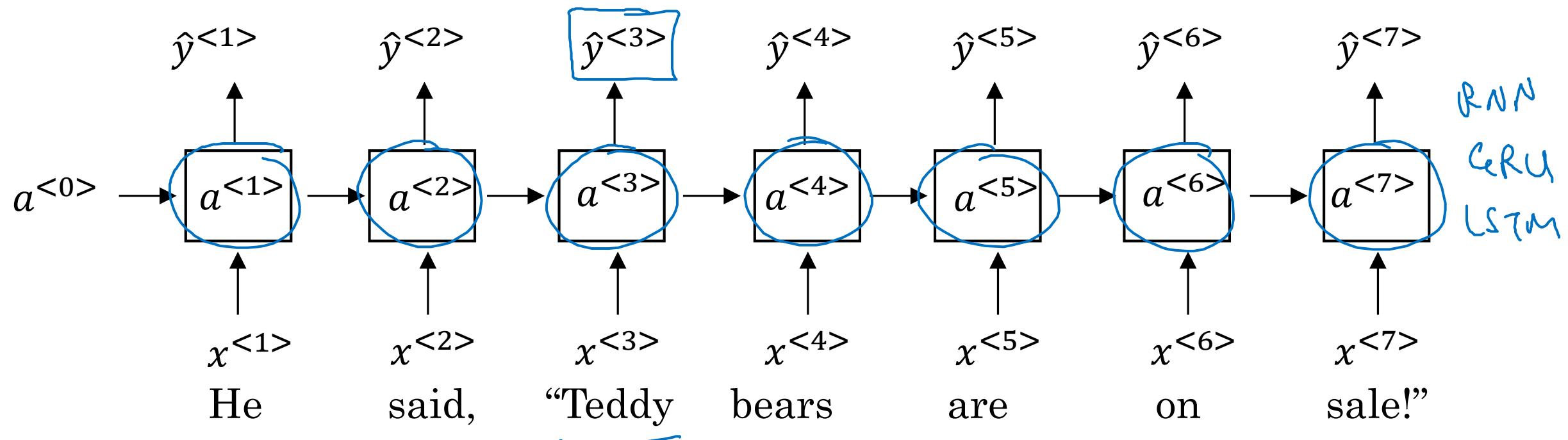
---

## Bidirectional RNN

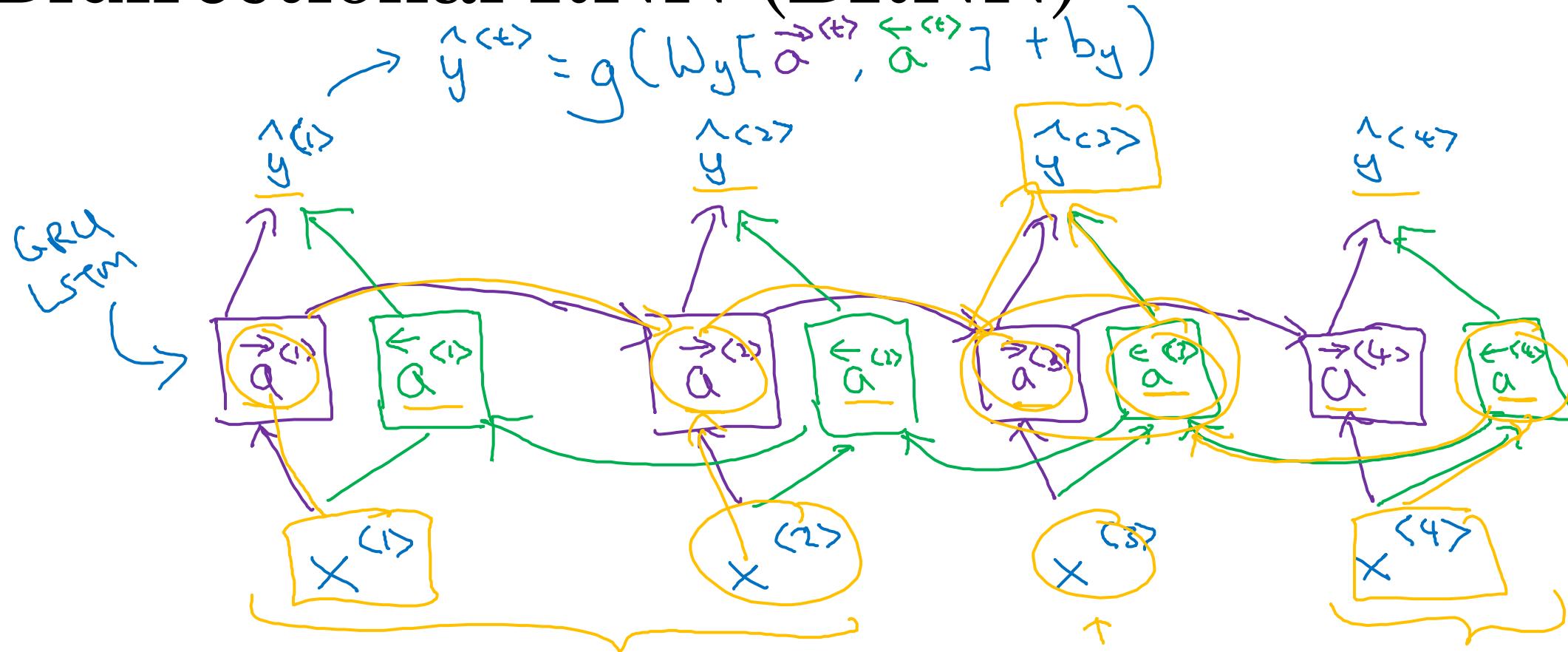
# Getting information from the future

He said, “Teddy bears are on sale!”

He said, “Teddy Roosevelt was a great President!”



# Bidirectional RNN (BRNN)



Acyclic graph

BRNN w/ LSTM

He saw,

"Teddy Roosevelt ..."



deeplearning.ai

# Sequence to sequence models

---

## Basic models

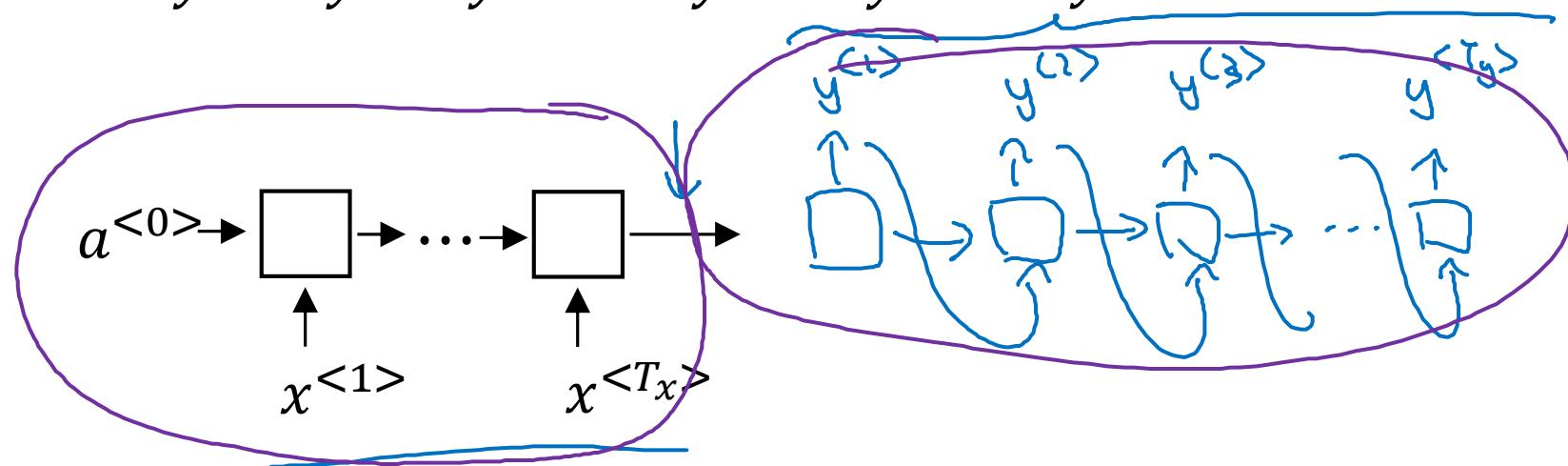
# Sequence to sequence model

$x^{<1>} \quad x^{<2>} \quad x^{<3>} \quad x^{<4>} \quad x^{<5>}$

Jane visite l'Afrique en septembre

→ Jane is visiting Africa in September.

$y^{<1>} \quad y^{<2>} \quad y^{<3>} \quad y^{<4>} \quad y^{<5>} \quad y^{<6>}$

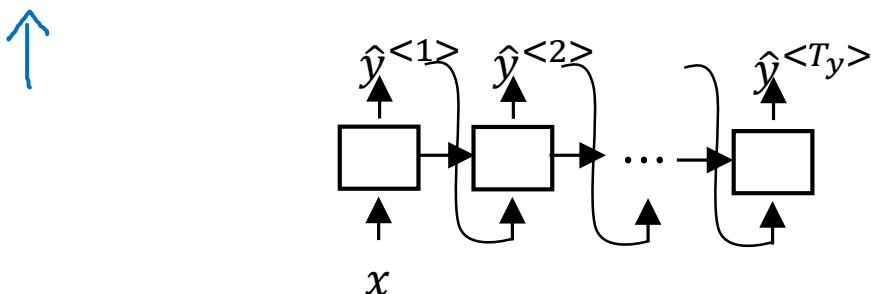
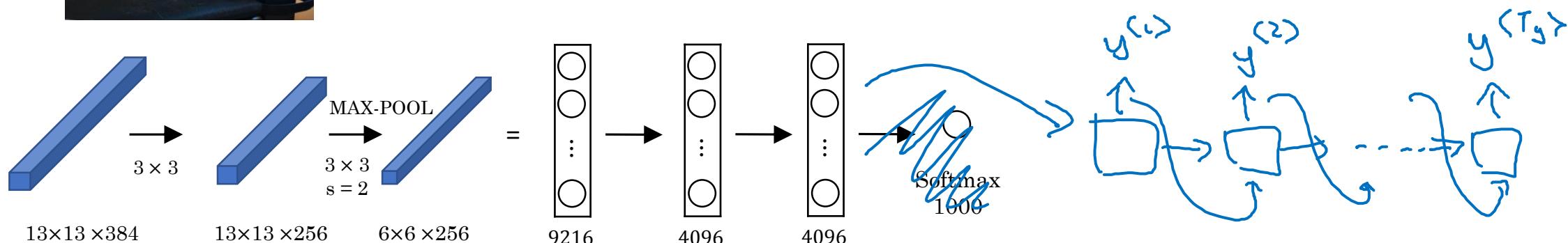
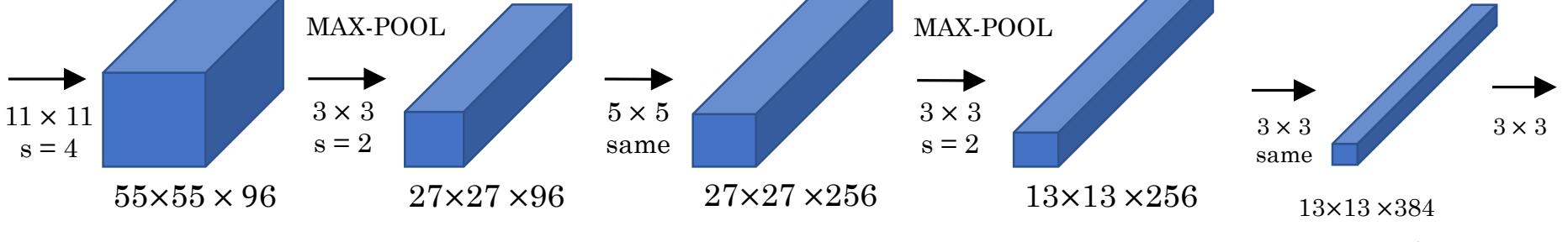
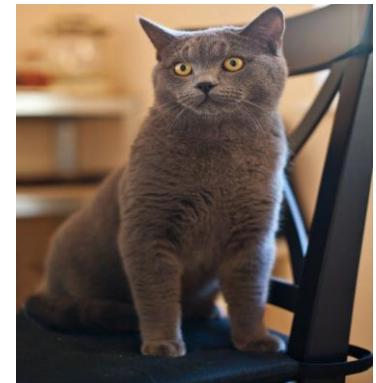


[Sutskever et al., 2014. Sequence to sequence learning with neural networks] ↪

[Cho et al., 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation] ↪

Andrew Ng

# Image captioning



[Mao et. al., 2014. Deep captioning with multimodal recurrent neural networks]

[Vinyals et. al., 2014. Show and tell: Neural image caption generator]

[Karpathy and Li, 2015. Deep visual-semantic alignments for generating image descriptions]

Andrew Ng



deeplearning.ai

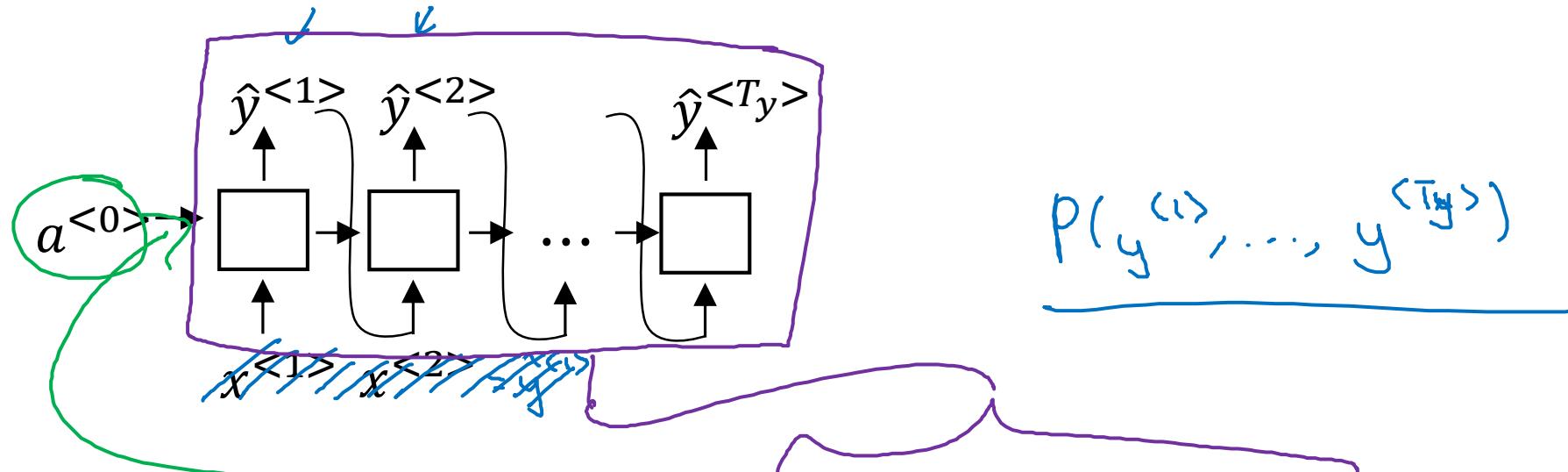
# Sequence to sequence models

---

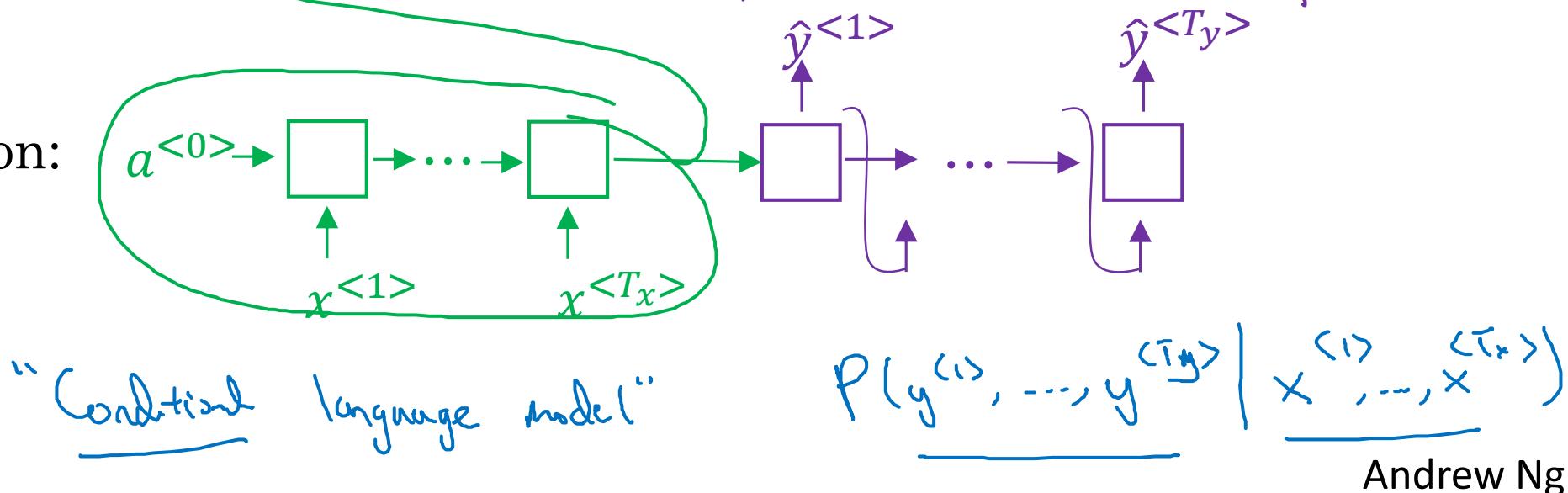
Picking the most  
likely sentence

# Machine translation as building a conditional language model

Language model:



Machine translation:



Andrew Ng

# Finding the most likely translation

Jane visite l'Afrique en septembre.

$$P(y^{<1>}, \dots, y^{<T_y>} | x)$$

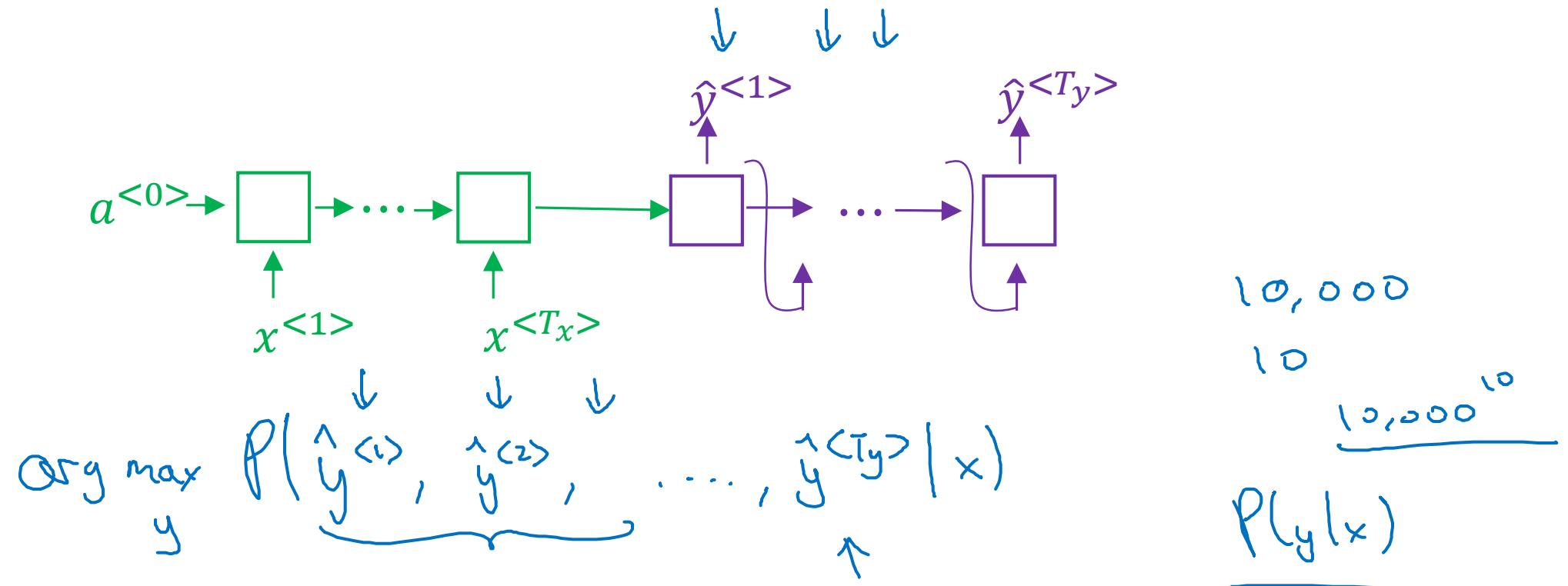
French  
↓  
English

- Jane is visiting Africa in September.
- Jane is going to be visiting Africa in September.
- In September, Jane will visit Africa.
- Her African friend welcomed Jane in September.

$$\arg \max_{y^{<1>}, \dots, y^{<T_y>}} P(y^{<1>}, \dots, y^{<T_y>} | x)$$

# Why not a greedy search?

$$P(\hat{y}^{(1)} | x)$$



→ Jane is visiting Africa in September.

→ Jane is going to be visiting Africa in September.

$$P(\text{Jane is going } | x) > P(\text{Jane is visiting } | x)$$



deeplearning.ai

# Sequence to sequence models

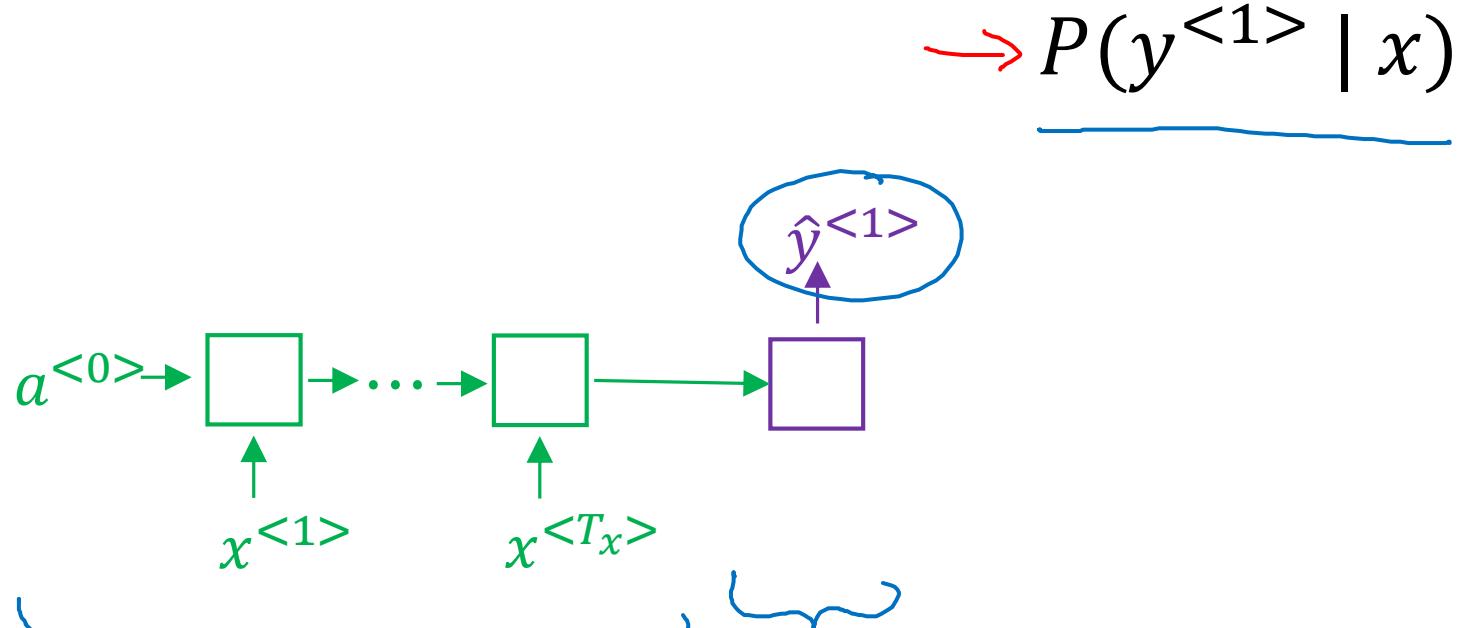
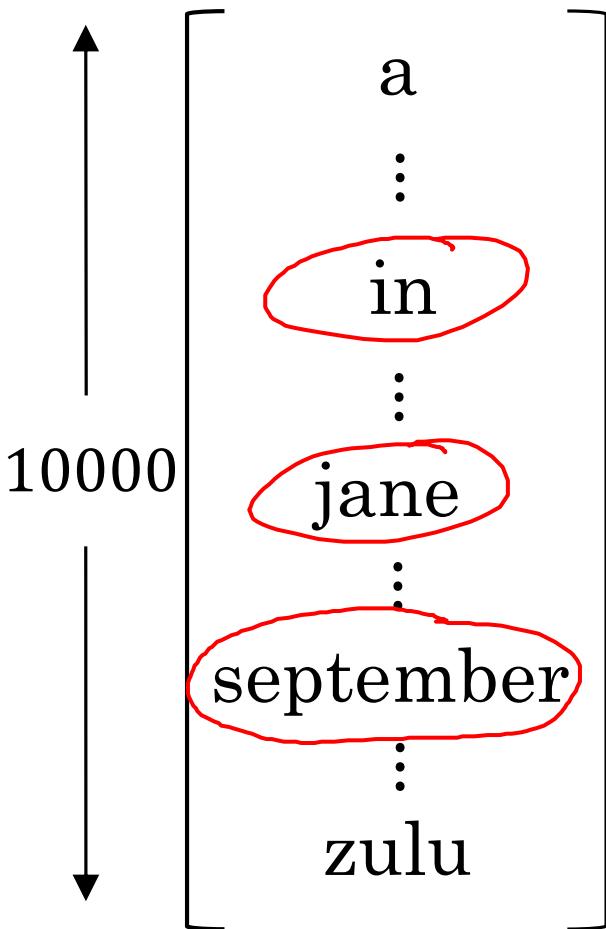
---

## Beam search

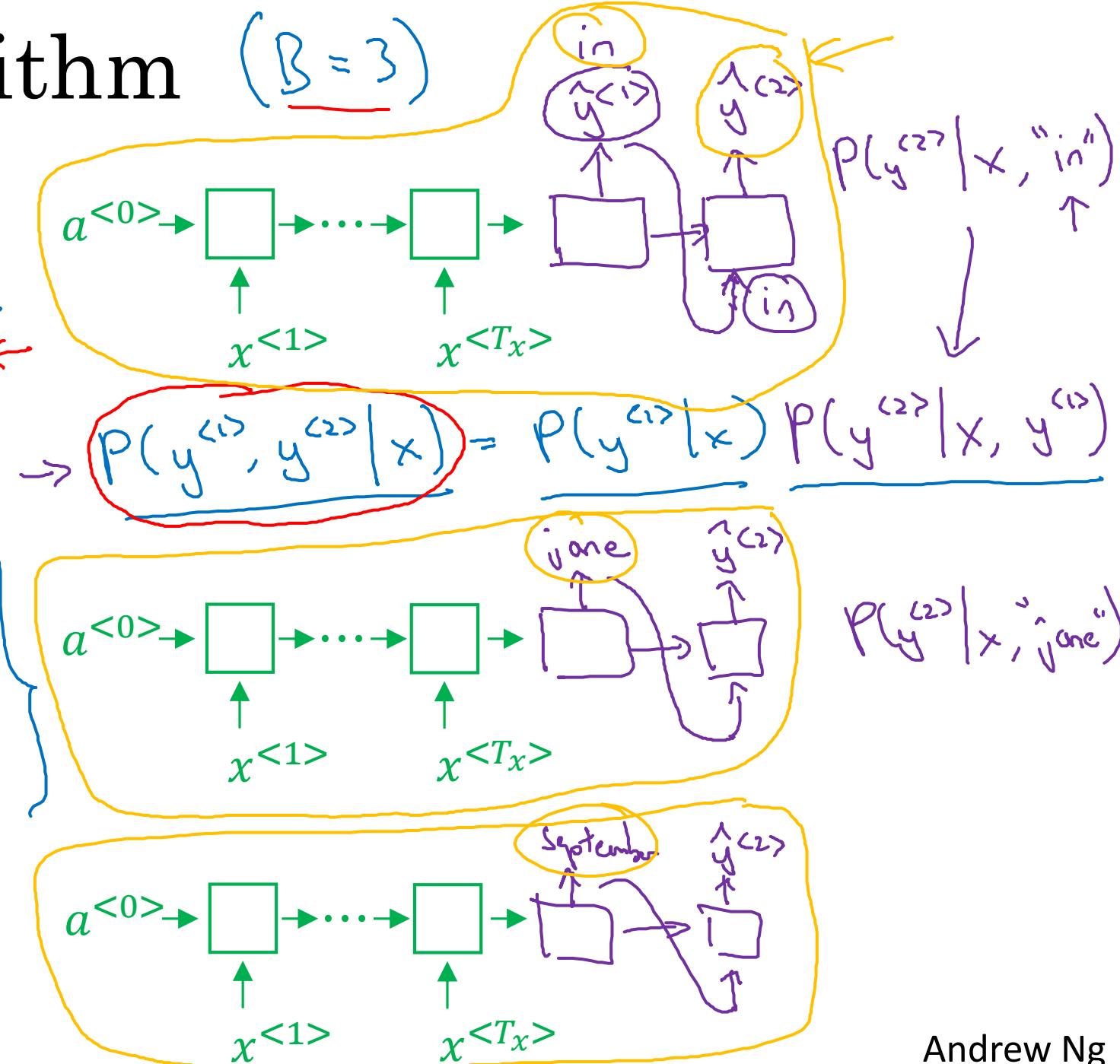
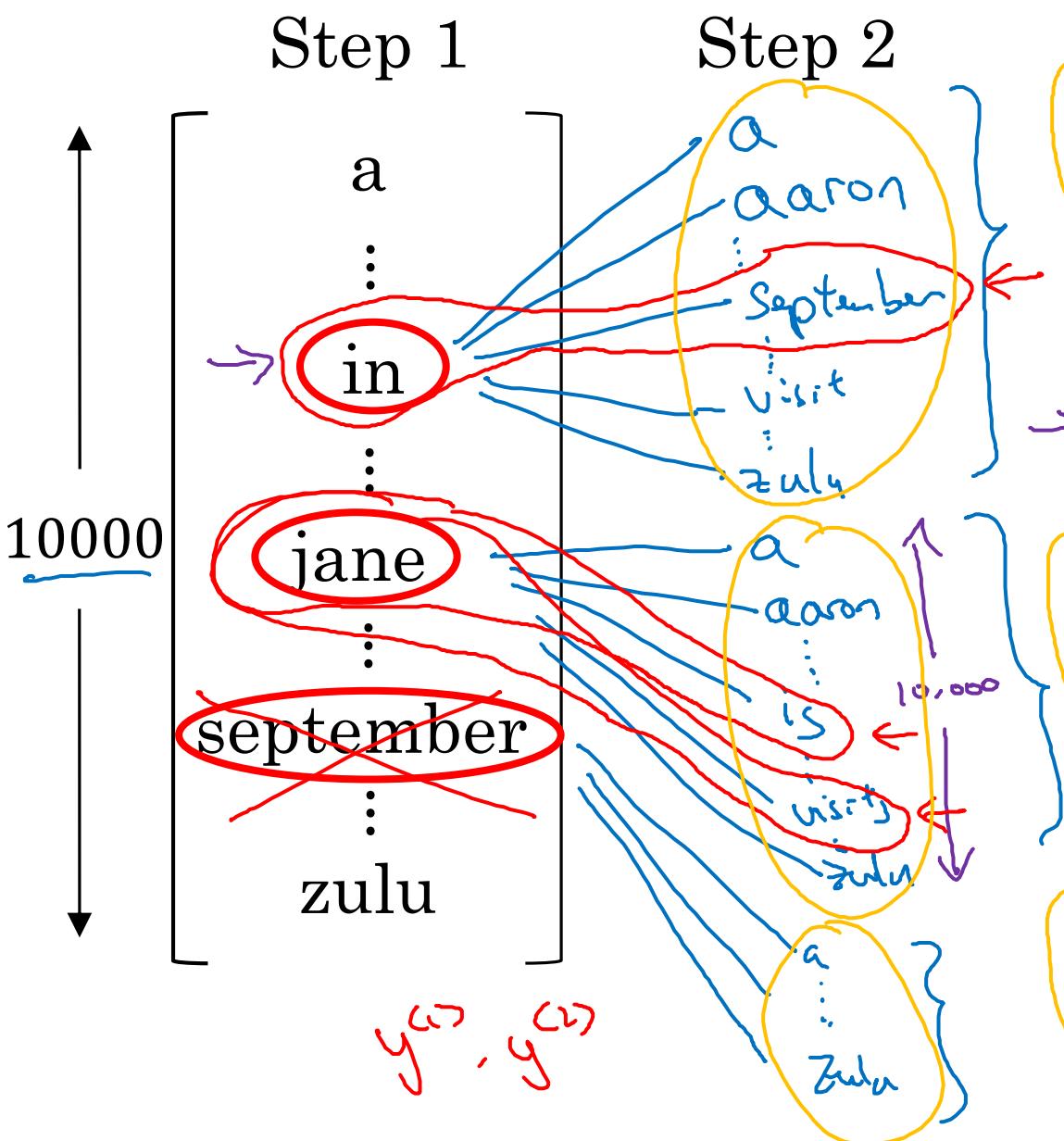
# Beam search algorithm

B = 3 (beam width)

Step 1



# Beam search algorithm



# Beam search ( $B = 3$ )

in september

$a^{<0>} \rightarrow \square \rightarrow \dots \rightarrow \square \rightarrow \hat{y}^{<3>}$

jane is

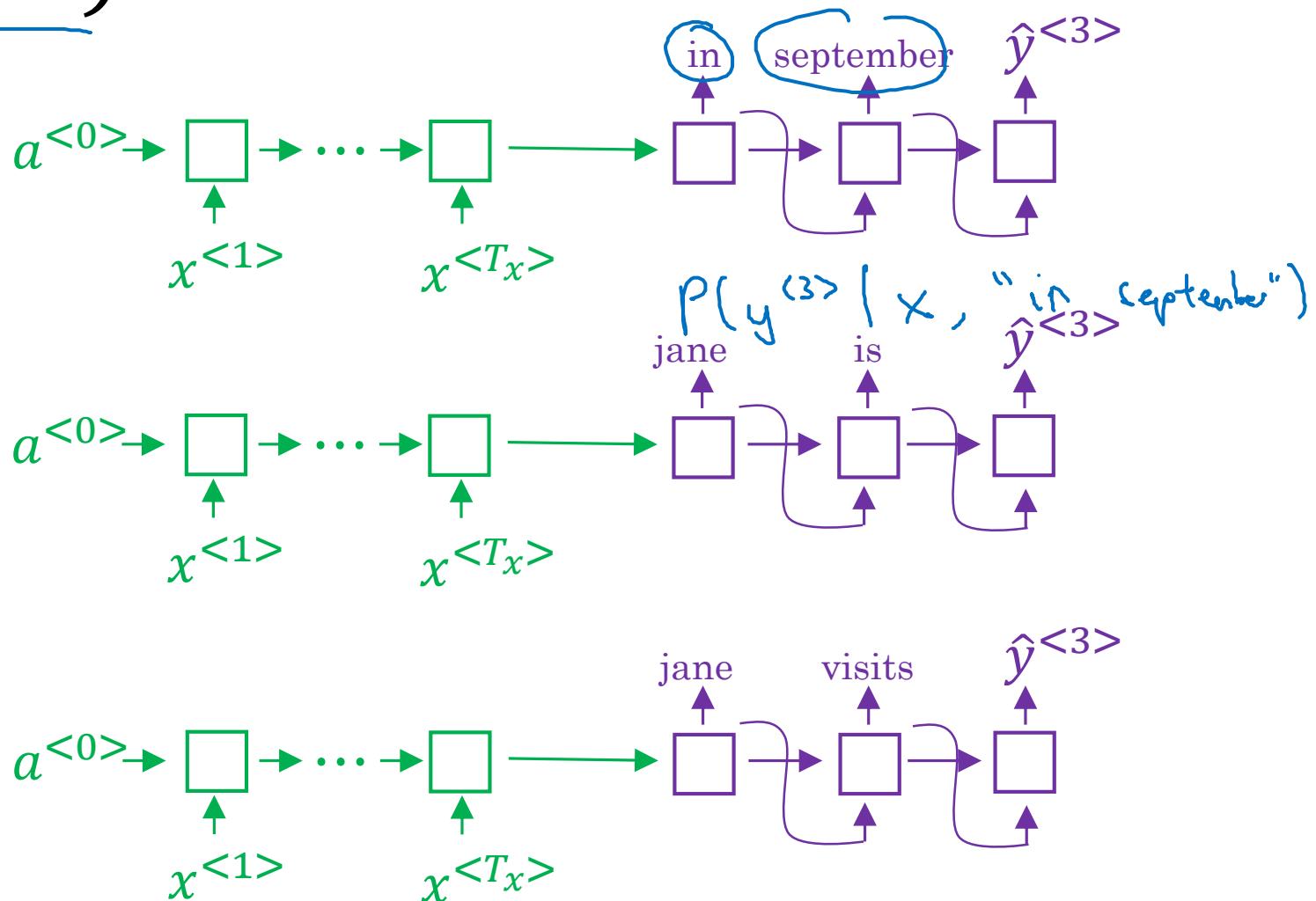
$a^{<0>} \rightarrow \square \rightarrow \dots \rightarrow \square \rightarrow \hat{y}^{<3>}$

jane visits

$a^{<0>} \rightarrow \square \rightarrow \dots \rightarrow \square \rightarrow \hat{y}^{<3>}$

$$P(y^{<1>}, y^{<2>} | x)$$

$B=1 \rightsquigarrow$  greedy search



jane visits africa in september. <EOS>



deeplearning.ai

# Sequence to sequence models

---

## Refinements to beam search

# Length normalization

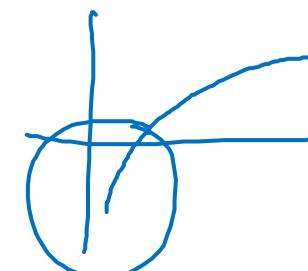
$$\arg \max_y \prod_{t=1}^{T_y} P(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>})$$

$P(y^{<1>} \dots y^{<T_y>} | x) = P(y^{<1>} | x) P(y^{<2>} | x, y^{<1>}) \dots P(y^{<T_y>} | x, y^{<1>} \dots, y^{<T_y-1>})$

$$\arg \max_y \sum_{t=1}^{T_y} \log P(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>})$$

$$T_y = 1, 2, 3, \dots, 30.$$

$$\frac{1}{T_y^\alpha} \sum_{t=1}^{T_y} \log P(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>})$$



$$\log P(y|x) \leftarrow$$

$$P(y|x) \leftarrow$$

$$\alpha = 0.7$$

$$\begin{cases} d = 1 \\ d = 0 \end{cases}$$

# Beam search discussion

Beam width B?

$1 \rightarrow 3 \rightarrow 10, \quad 100, \quad 1000 \rightarrow 3000$

large B: better result, slower  
small B: worse result, faster

Unlike exact search algorithms like BFS (Breadth First Search) or DFS (Depth First Search), Beam Search runs faster but is not guaranteed to find exact maximum for  $\arg \max_y P(y|x)$ .

y



deeplearning.ai

# Sequence to sequence models

---

## Error analysis on beam search

# Example

Jane visite l'Afrique en septembre.

→ RNN

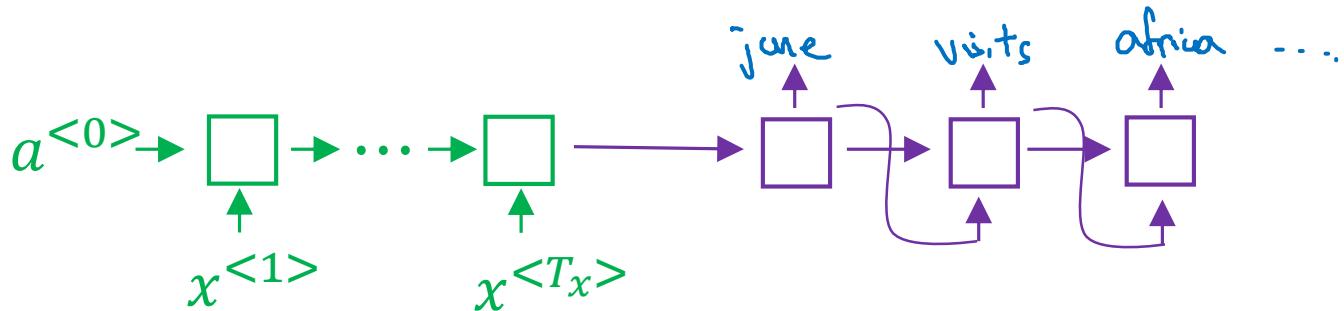
→ Beam Search

B↑

Human: Jane visits Africa in September. ( $y^*$ )

Algorithm: Jane visited Africa last September. ( $\hat{y}$ ) ←

$$\text{RNN} \text{ computes } P(y^*|x) \geq P(\hat{y}|x)$$



# Error analysis on beam search

$$P(y^*|x)$$

Human: Jane visits Africa in September. ( $y^*$ )

$$P(\hat{y}|x)$$

Algorithm: Jane visited Africa last September. ( $\hat{y}$ )

Case 1:  $P(y^*|x) > P(\hat{y}|x) \leftarrow$

$$\arg \max_y P(y|x)$$

Beam search chose  $\hat{y}$ . But  $y^*$  attains higher  $P(y|x)$ .

Conclusion: Beam search is at fault.

Case 2:  $P(y^*|x) \leq P(\hat{y}|x) \leftarrow$

$y^*$  is a better translation than  $\hat{y}$ . But RNN predicted  $P(y^*|x) < P(\hat{y}|x)$ .

Conclusion: RNN model is at fault.

# Error analysis process

Human	Algorithm	$P(y^* x)$	$P(\hat{y} x)$	At fault?
Jane visits Africa in September.	Jane visited Africa last September.	<u><math>2 \times 10^{-10}</math></u>	<u><math>1 \times 10^{-10}</math></u>	B
...	...	—	—	R
...	...	—	—	R
			—	R
				:

Figures out what fraction of errors are “due to” beam search vs. RNN model



deeplearning.ai

# Sequence to sequence models

---

Bleu score  
(optional)

# Evaluating machine translation

French: Le chat est sur le tapis.

Reference 1: The cat is on the mat. 

Reference 2: There is a cat on the mat. 

MT output: the the the the the the.

Precision:

Modified precision:

Bleu  
bilingual evaluation understudy

# Bleu score on bigrams

Example: Reference 1: The cat is on the mat. ←

Reference 2: There is a cat on the mat. ←

MT output: The cat the cat on the mat. ←

	Count	Count <sub>clip</sub>	
the cat	2 ←	1 ←	
cat the	1 ←	0	
cat on	1 ←	1 ←	
on the	1 ←	1 ←	
the mat	1 ←	1 ←	

$$\frac{4}{6}$$

# Bleu score on unigrams

Example: Reference 1: The cat is on the mat.

$$P_1, P_2, = 1.0$$

Reference 2: There is a cat on the mat.

→ MT output: The cat the cat on the mat. (↑)

$$P_1 = \frac{\sum_{\text{unigrams} \in \hat{y}} \text{count}_{clip}(\text{unigram})}{\sum_{\text{unigrams} \in \hat{y}} \text{count}(\text{unigram})}$$

↑  
Unigram

Count (unigram) Count (unigram)

$\sum_{\text{unigrams} \in \hat{y}}$   $\text{count}_{clip}(\text{unigram})$

$$p_n = \frac{\sum_{n\text{-gram} \in \hat{y}} \text{count}_{clip}(n\text{-gram})}{\sum_{n\text{-grams} \in \hat{y}} \text{count}(n\text{-gram})}$$

↑  
 $n\text{-gram}$

$\sum_{n\text{-grams} \in \hat{y}}$   $\text{count}(n\text{-gram})$

$\sum_{n\text{-gram} \in \hat{y}} \text{count}_{clip}(n\text{-gram})$  Count clip (n-gram)

# Bleu details

$p_n$  = Bleu score on n-grams only

$P_1, P_2, P_3, P_4$

Combined Bleu score:

$$BP \exp\left(\frac{1}{4} \sum_{n=1}^4 p_n\right)$$

$BP$  = brevity penalty

$$BP = \begin{cases} 1 & \text{if } \underline{\text{MT\_output\_length}} > \underline{\text{reference\_output\_length}} \\ \exp(1 - \text{MT\_output\_length}/\text{reference\_output\_length}) & \text{otherwise} \end{cases}$$





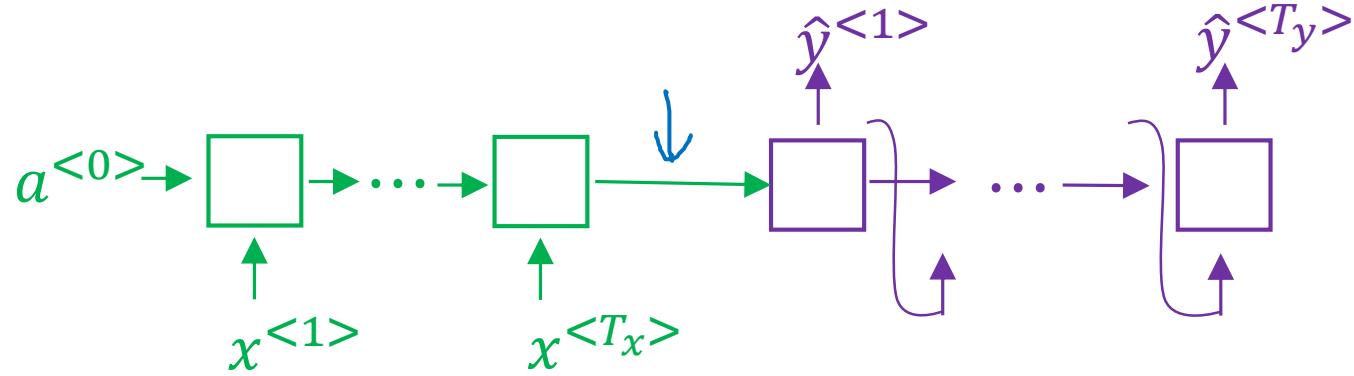
deeplearning.ai

# Sequence to sequence models

---

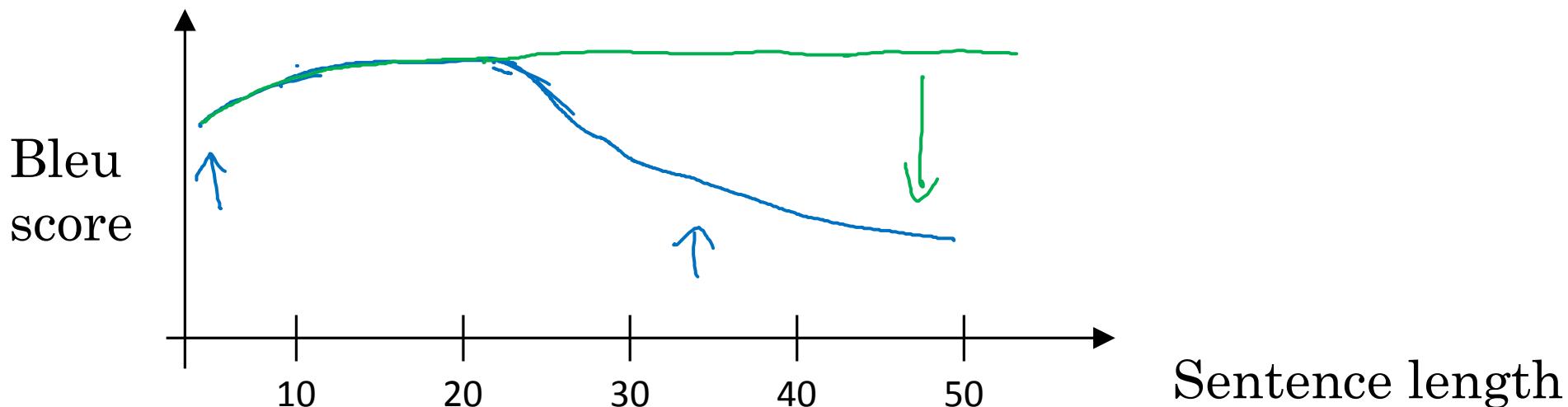
## Attention model intuition

# The problem of long sequences

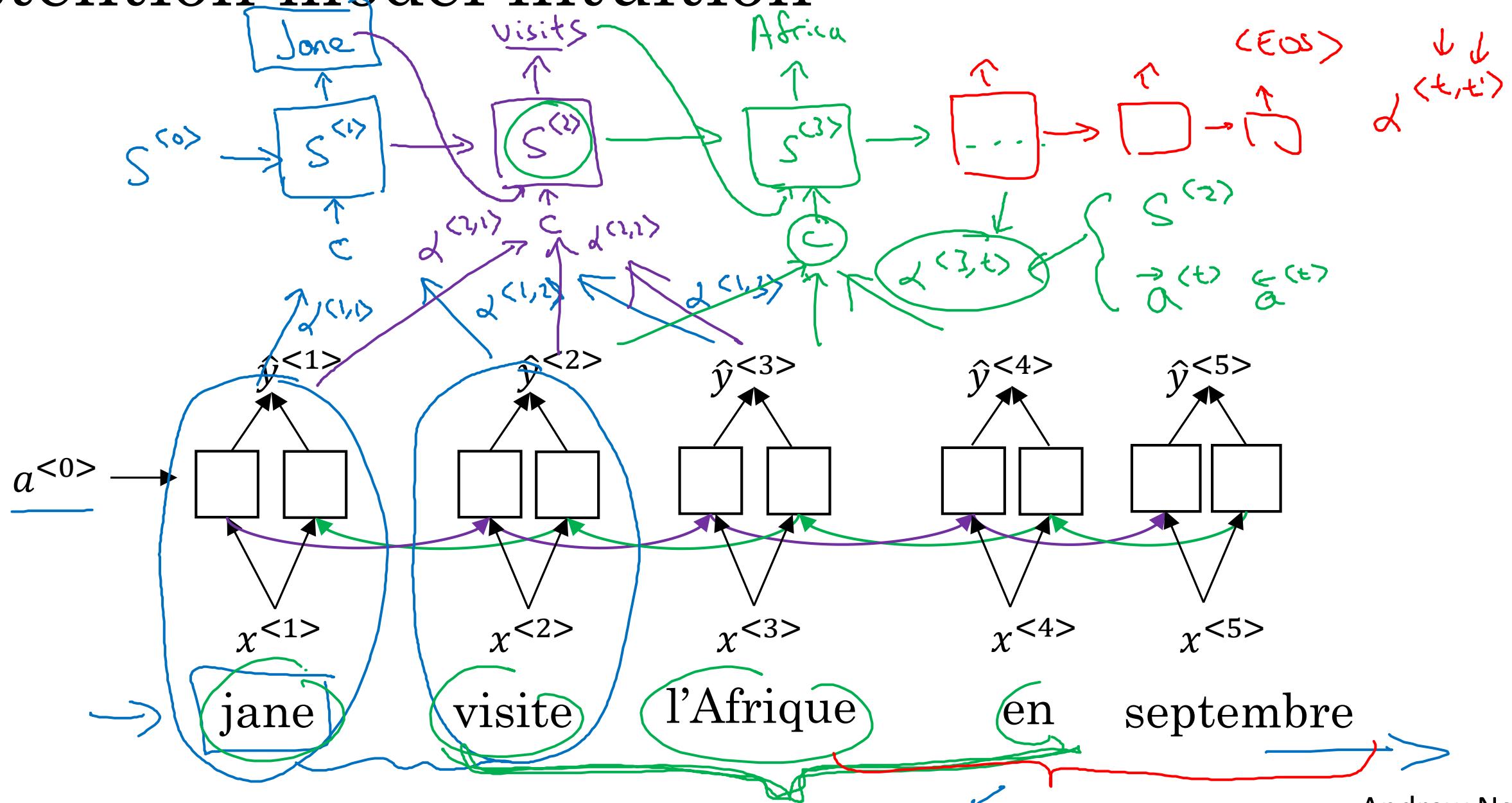


Jane s'est rendue en Afrique en septembre dernier, a apprécié la culture et a rencontré beaucoup de gens merveilleux; elle est revenue en parlant comment son voyage était merveilleux, et elle me tente d'y aller aussi.

Jane went to Africa last September, and enjoyed the culture and met many wonderful people; she came back raving about how wonderful her trip was, and is tempting me to go too.



# Attention model intuition





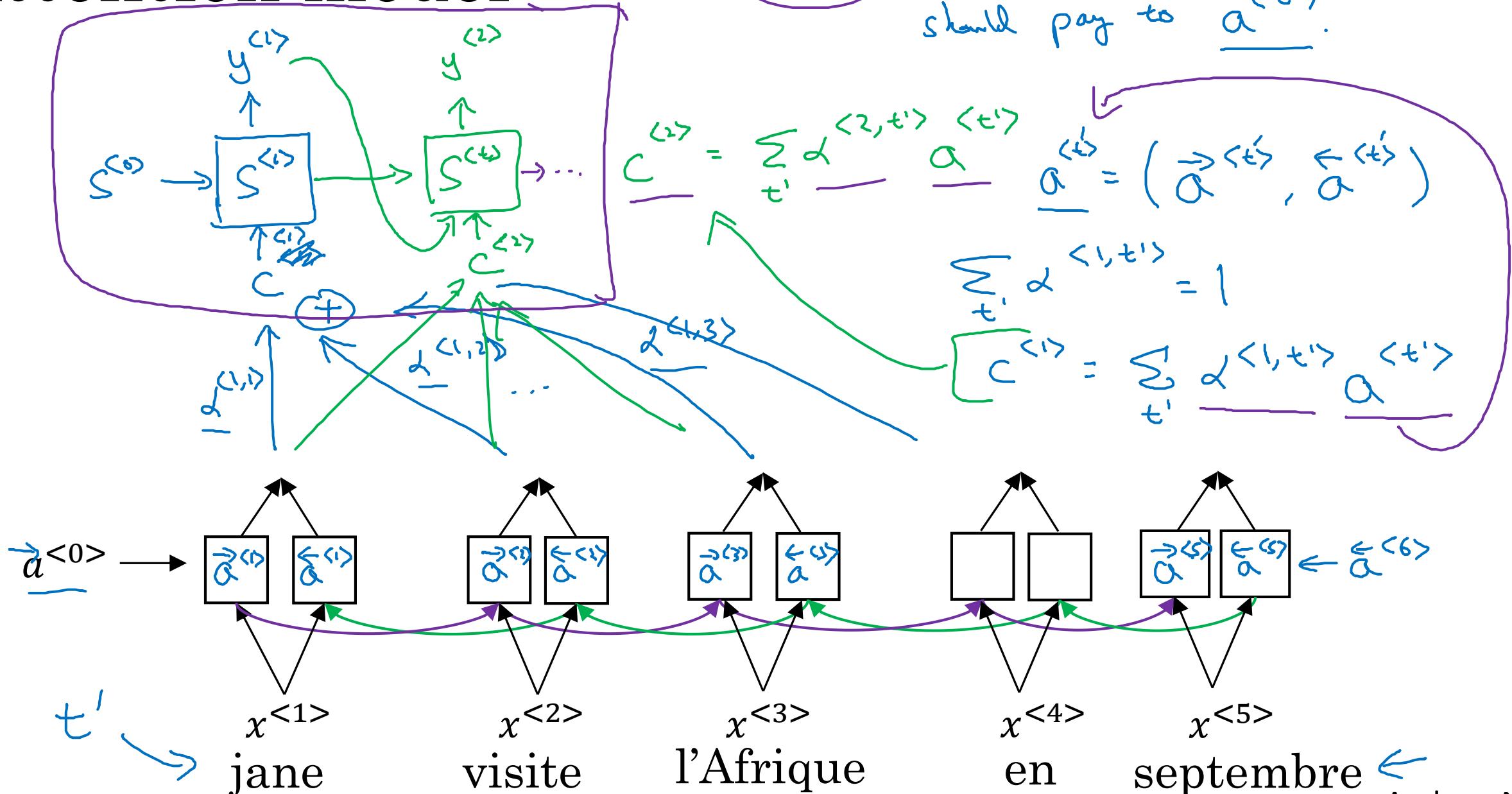
deeplearning.ai

# Sequence to sequence models

---

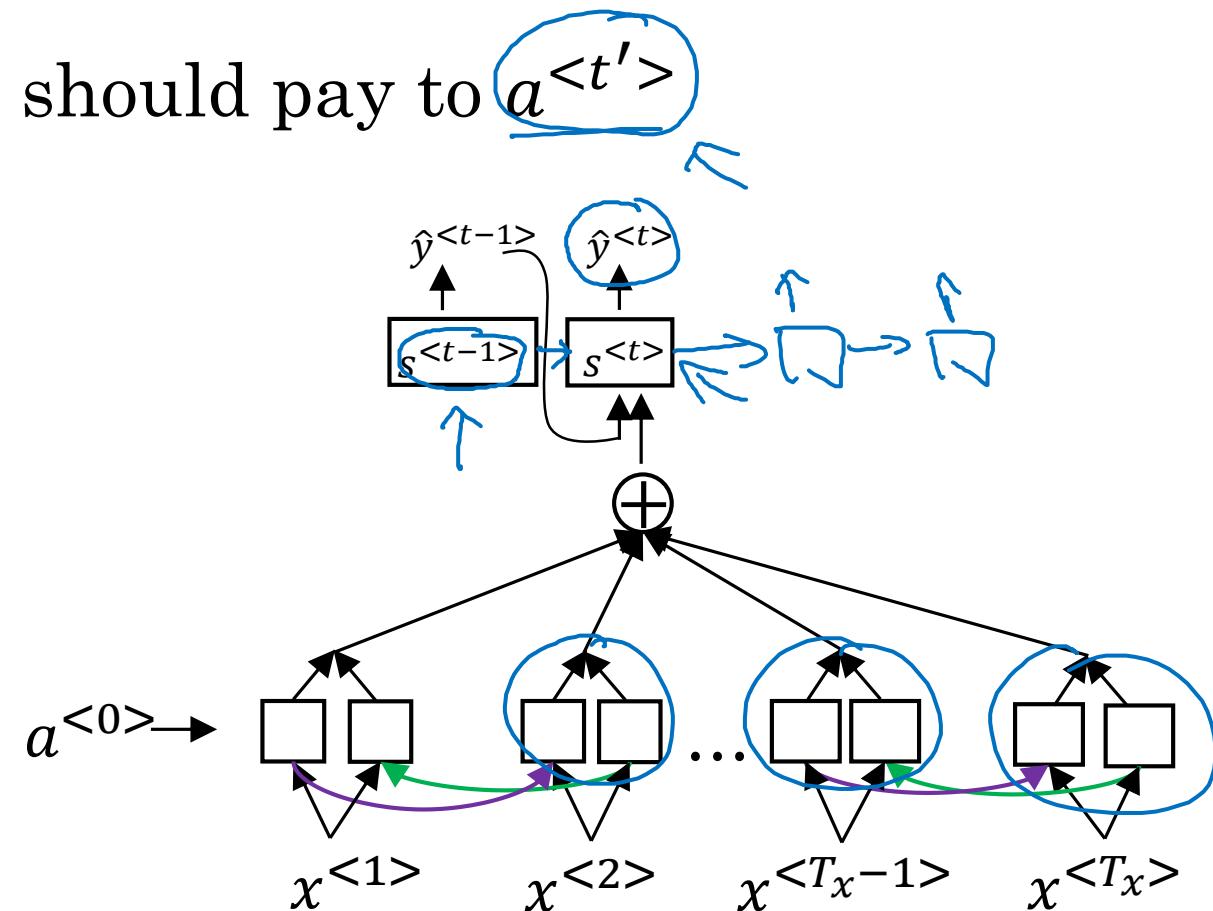
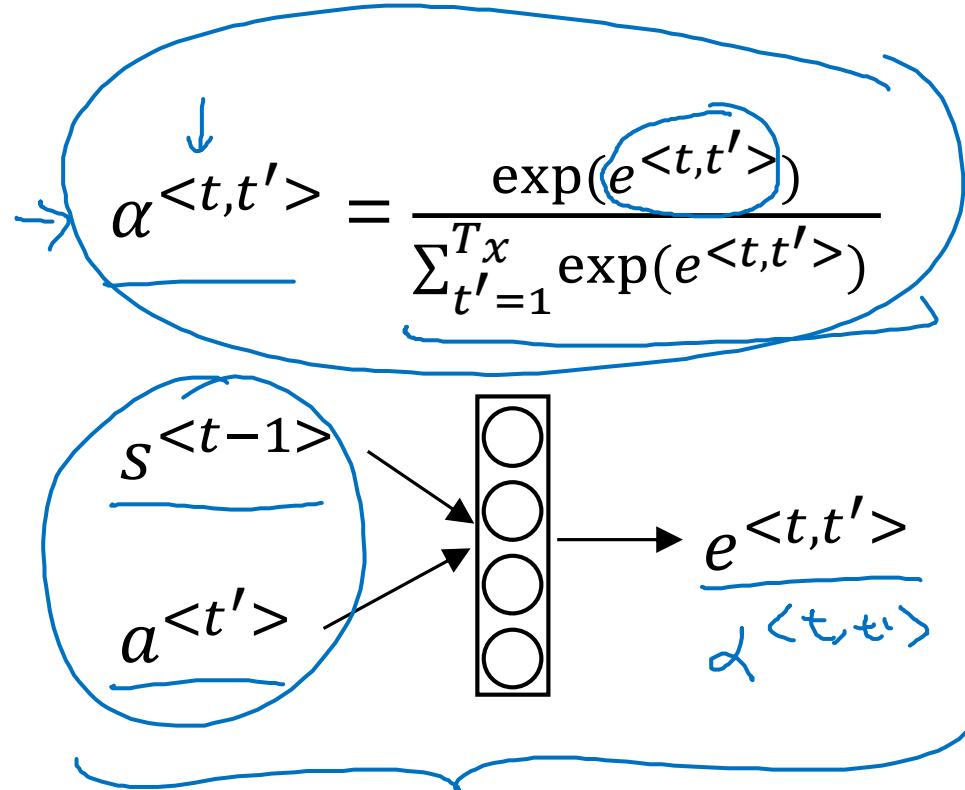
## Attention model

# Attention model



# Computing attention $\underline{\alpha^{'}}$

$\alpha^{'}$  = amount of attention  $y^{'}$  should pay to  $a^{'}$



[Bahdanau et. al., 2014. Neural machine translation by jointly learning to align and translate]

[Xu et. al., 2015. Show, attend and tell: Neural image caption generation with visual attention]

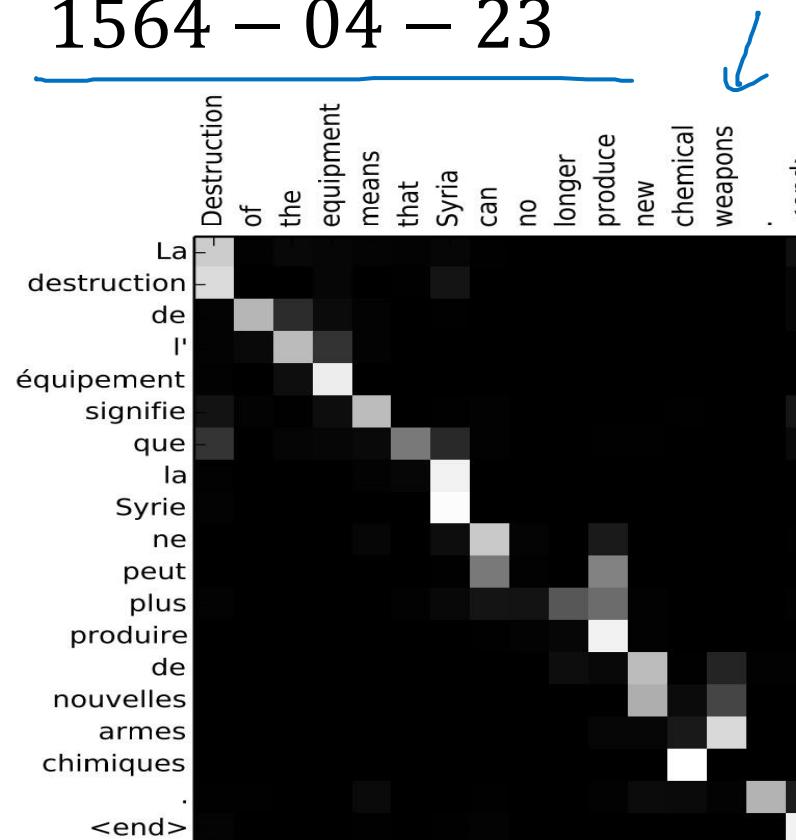
Andrew Ng

# Attention examples

July 20th 1969 → 1969 – 07 – 20

23 April, 1564 → 1564 – 04 – 23

Visualization of  $\alpha^{<t,t'>}$ :





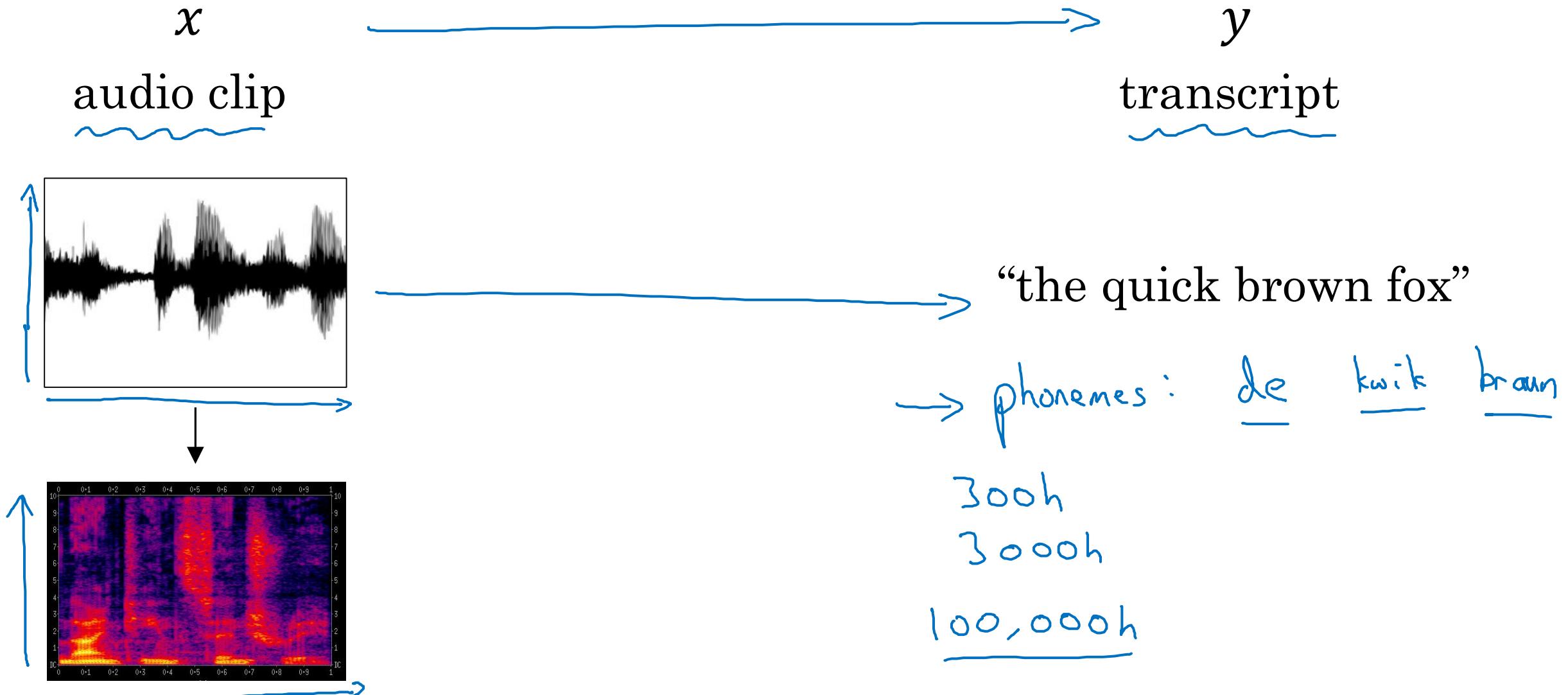
deeplearning.ai

Audio data

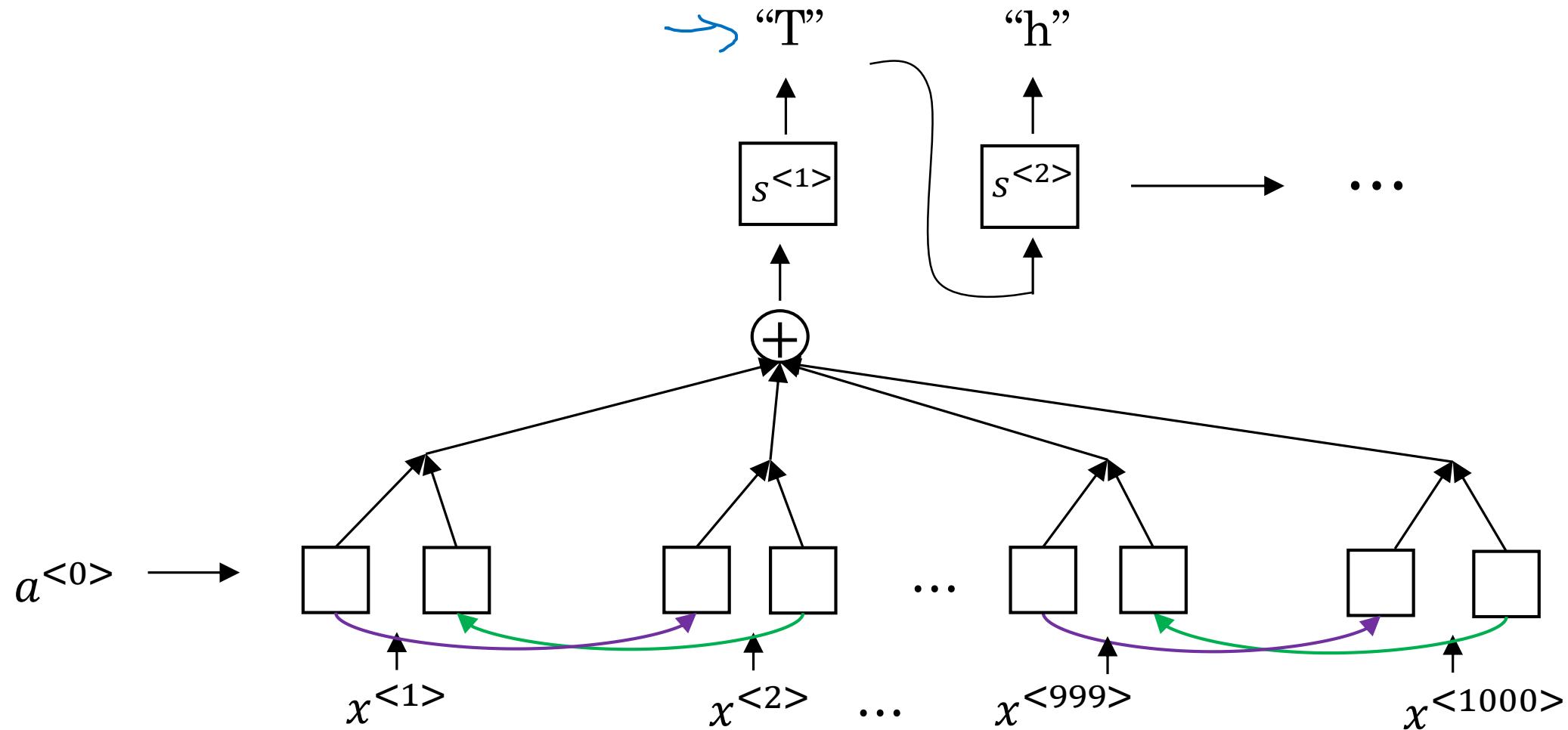
---

Speech recognition

# Speech recognition problem

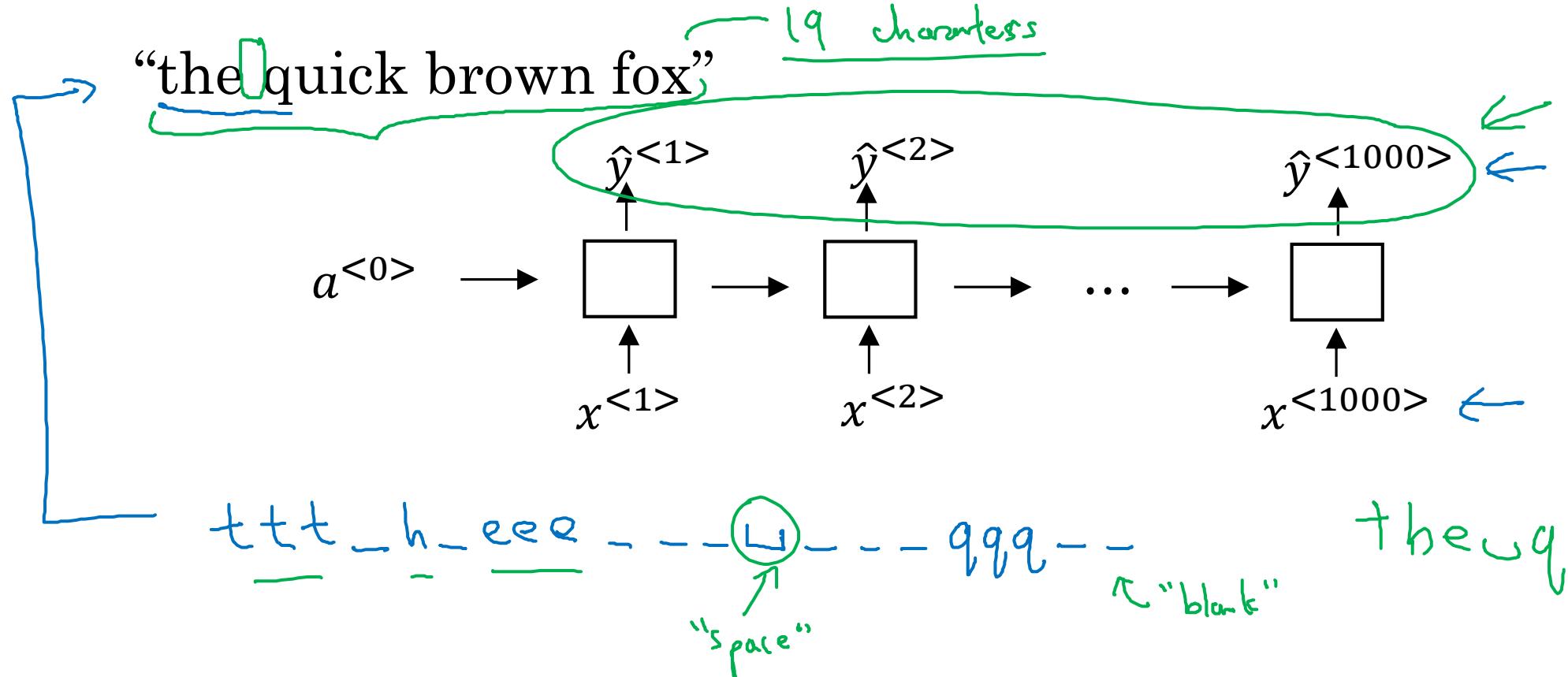


# Attention model for speech recognition



# CTC cost for speech recognition

(Connectionist temporal classification)



Basic rule: collapse repeated characters not separated by "blank"



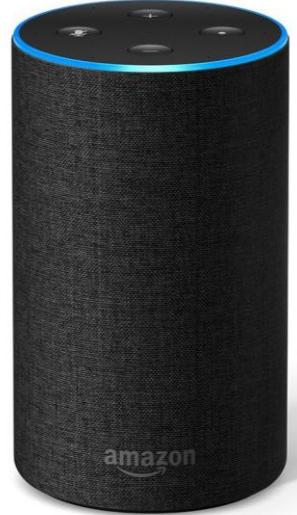
deeplearning.ai

Audio data

---

Trigger word  
detection

# What is trigger word detection?



Amazon Echo  
(Alexa)



Baidu DuerOS  
(xiaodunihao)

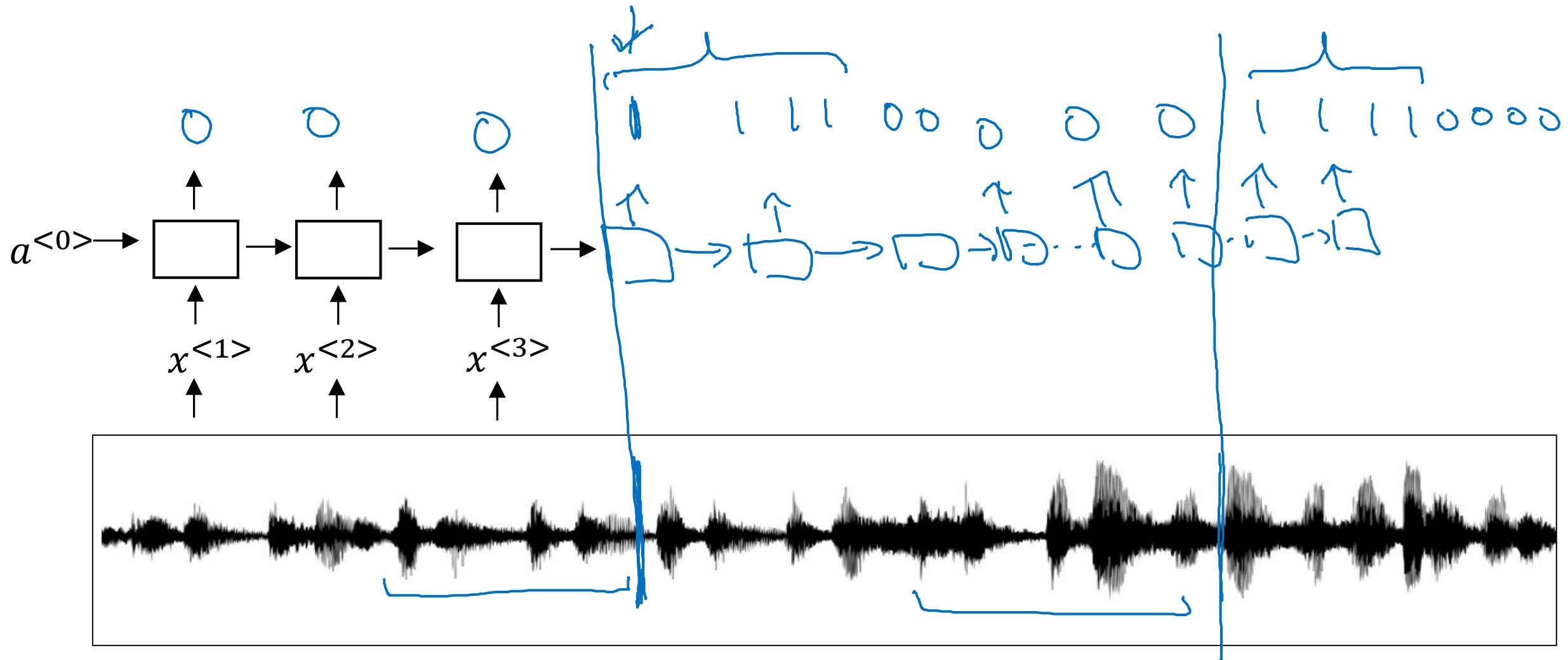


Apple Siri  
(Hey Siri)



Google Home  
(Okay Google)

# Trigger word detection algorithm





deeplearning.ai

# Conclusion

---

## Summary and thank you

# Specialization outline

1. Neural Networks and Deep Learning
2. Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization
3. Structuring Machine Learning Projects
4. Convolutional Neural Networks
5. Sequence Models

# Deep learning is a super power

Please buy this  
from shutterstock  
and replace in  
final video.



Thank you.

- Andrew Ng