

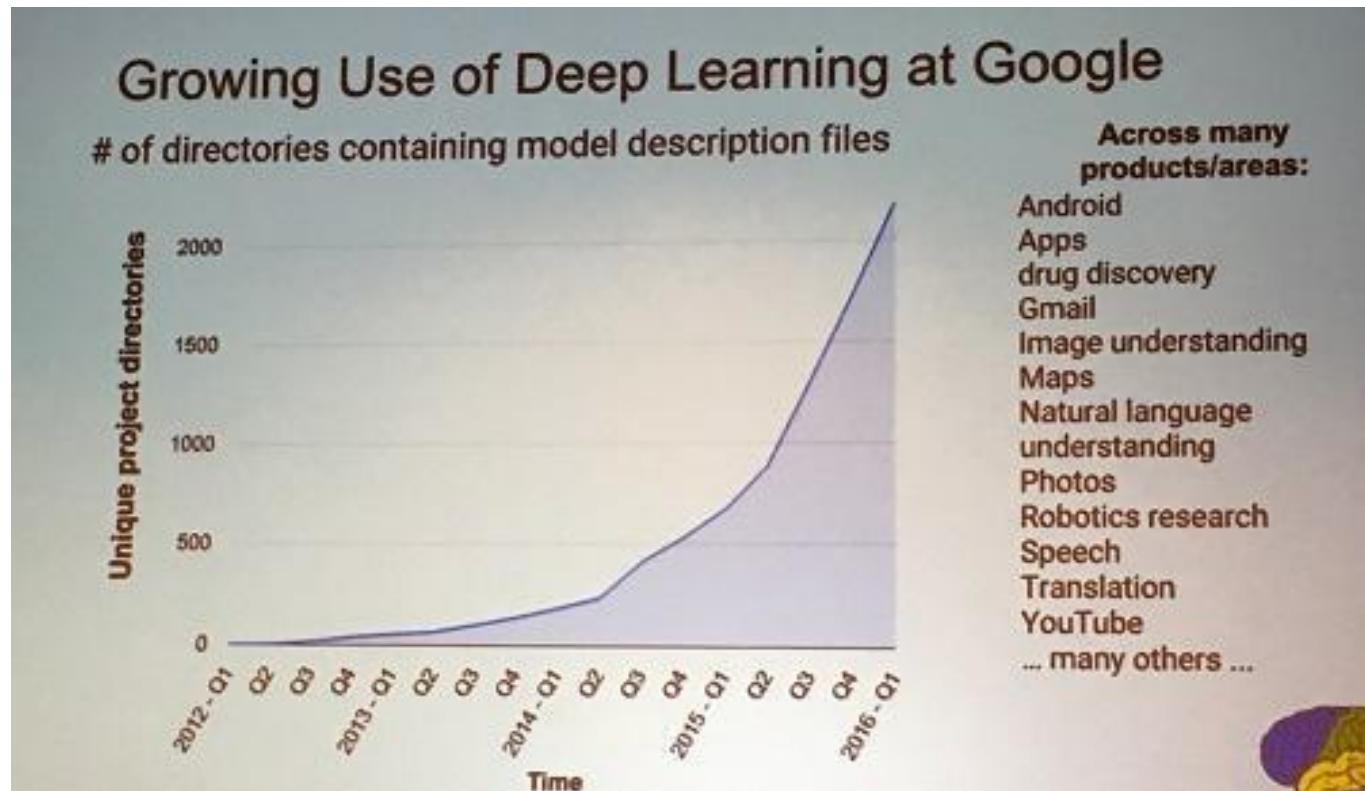
# Deep Learning Tutorial

李宏毅

Hung-yi Lee

# Deep learning attracts lots of attention.

- I believe you have seen lots of exciting results before.



Deep learning trends at Google. Source: SIGMOD/Jeff Dean

This talk focuses on the basic techniques.

# Outline

Lecture I: Introduction of Deep Learning



Lecture II: Tips for Training Deep Neural Network



Lecture III: Variants of Neural Network



Lecture IV: Next Wave

# Lecture I: Introduction of Deep Learning

# Outline of Lecture I

Introduction of Deep Learning

Let's start with general machine learning.

Why Deep?

“Hello World” for Deep Learning

# Machine Learning ≈ Looking for a Function

- Speech Recognition

$$f\left( \begin{array}{c} \text{[A waveform plot]} \\ \text{[A waveform plot]} \end{array} \right) = \text{“How are you”}$$

- Image Recognition

$$f\left( \begin{array}{c} \text{[A kitten image]} \\ \text{[A kitten image]} \end{array} \right) = \text{“Cat”}$$

- Playing Go

$$f\left( \begin{array}{c} \text{[A Go board diagram]} \\ \text{[A Go board diagram]} \end{array} \right) = \text{“5-5”}$$

(next move)

- Dialogue System

$$f\left( \begin{array}{c} \text{“Hi”} \\ \text{(what the user said)} \end{array} \right) = \text{“Hello”}$$

(system response)

# Framework

Image Recognition:

$$f(\text{}) = \text{"cat"}$$



$$f_1(\text{}) = \text{"cat"} \quad f_2(\text{}) = \text{"money"}$$

$$f_1(\text{}) = \text{"dog"} \quad f_2(\text{}) = \text{"snake"}$$

# Framework

A set of function

Model  
 $f_1, f_2 \dots$

Goodness of function  $f$

Training Data

Image Recognition:

$$f(\text{) = "cat"$$

$$f_1(\text{) = "cat"$$
      
$$f_2(\text{) = "money"$$

Better!

$$f_1(\text{) = "dog"$$
      
$$f_2(\text{) = "snake"$$

Supervised Learning

function input:

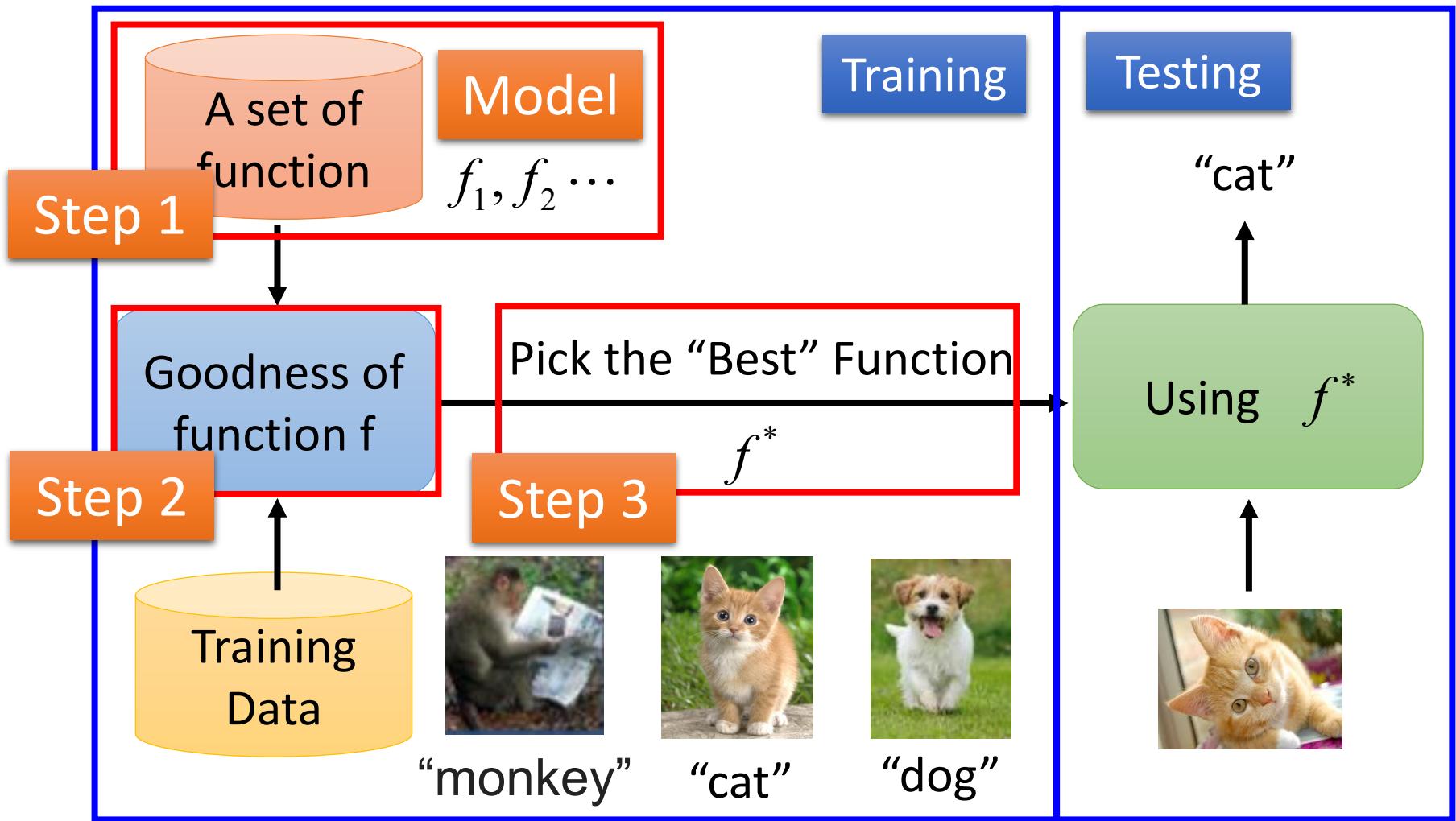


function output: “monkey”    “cat”    “dog”

# Framework

Image Recognition:

$$f(\text{cat image}) = \text{"cat"}$$



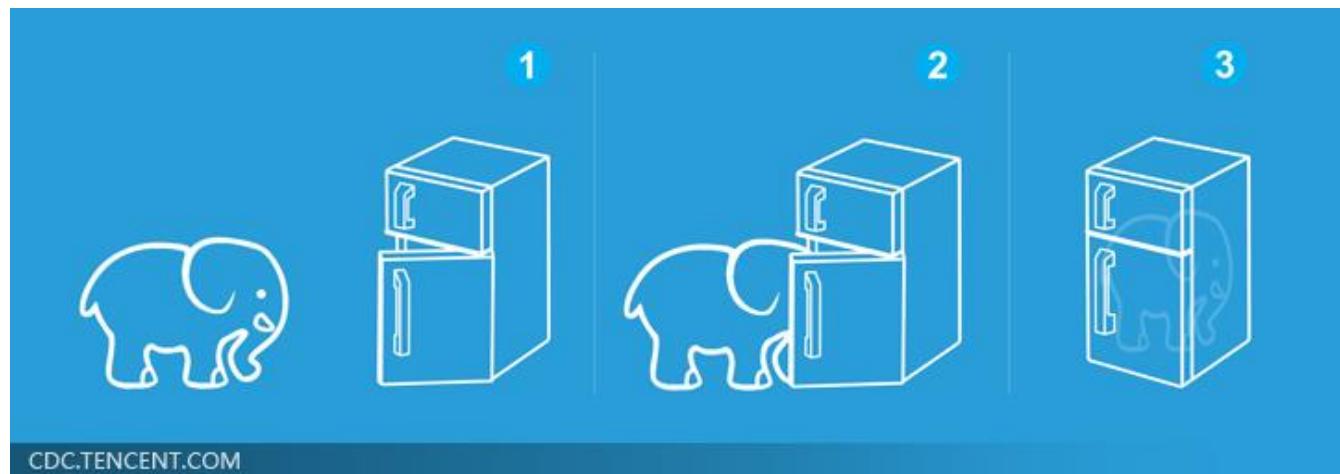
# Three Steps for Deep Learning

Step 1:  
define a set  
of function

Step 2:  
goodness of  
function

Step 3: pick  
the best  
function

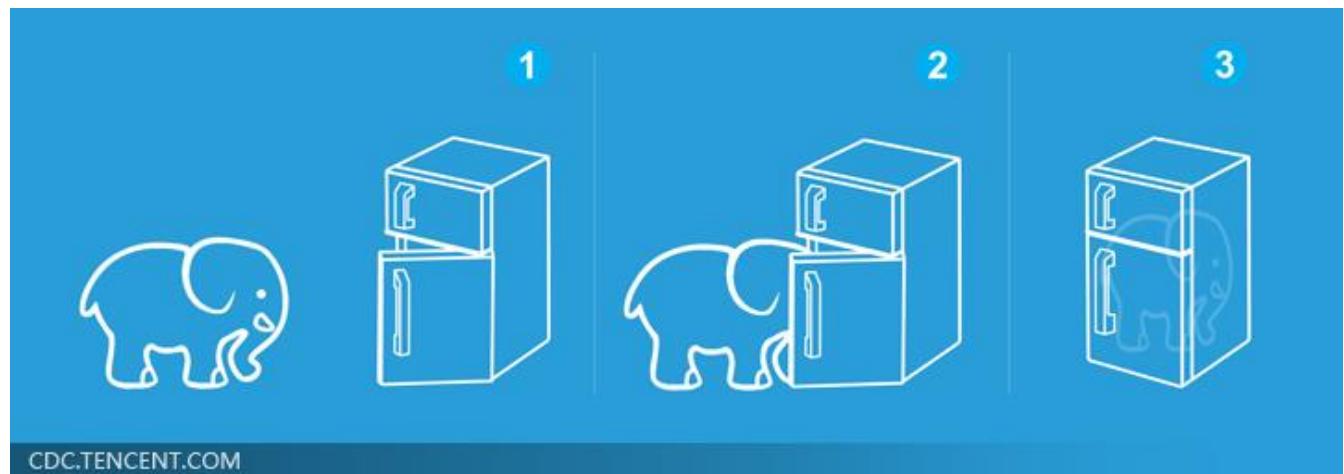
Deep Learning is so simple .....



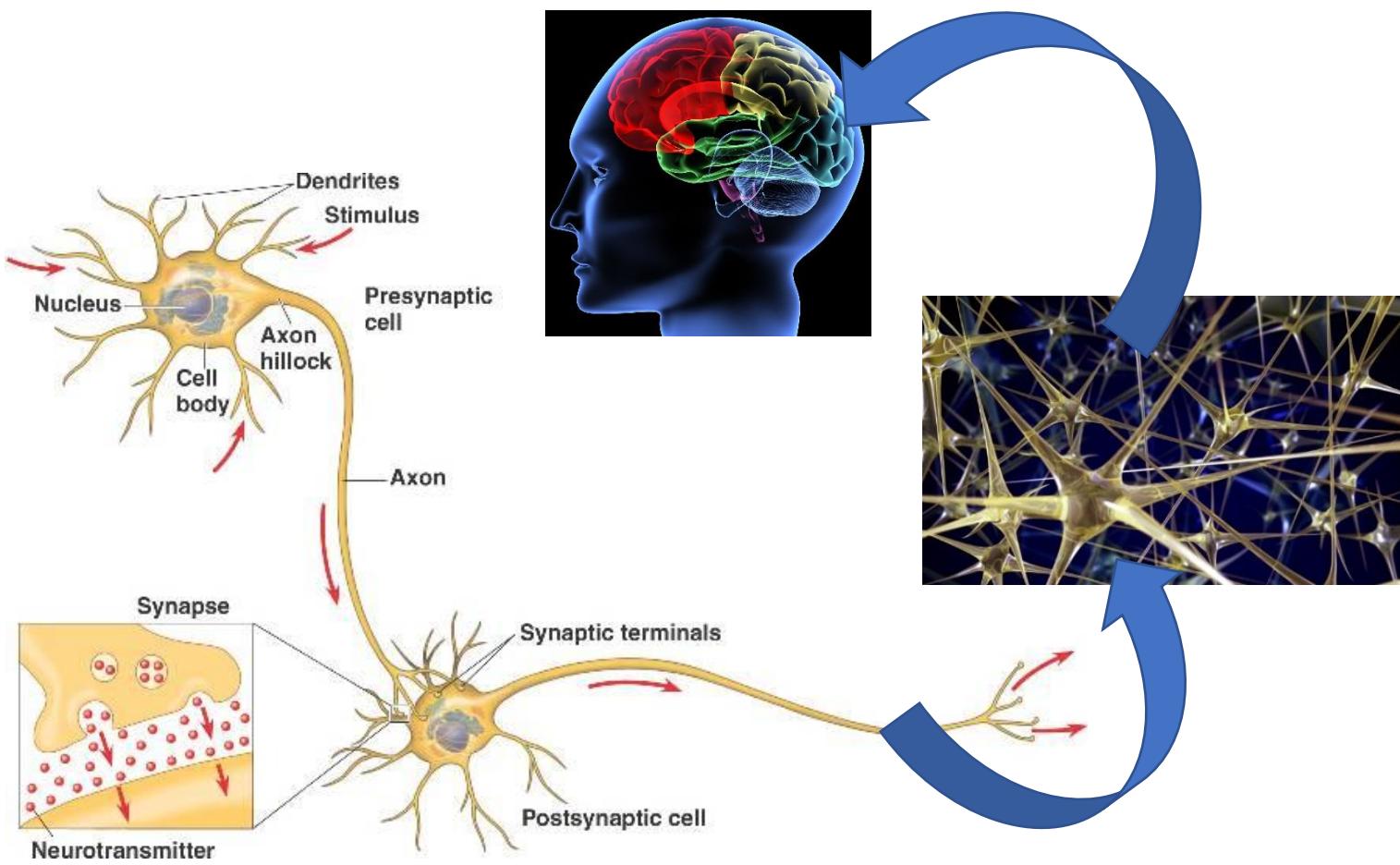
# Three Steps for Deep Learning



Deep Learning is so simple .....



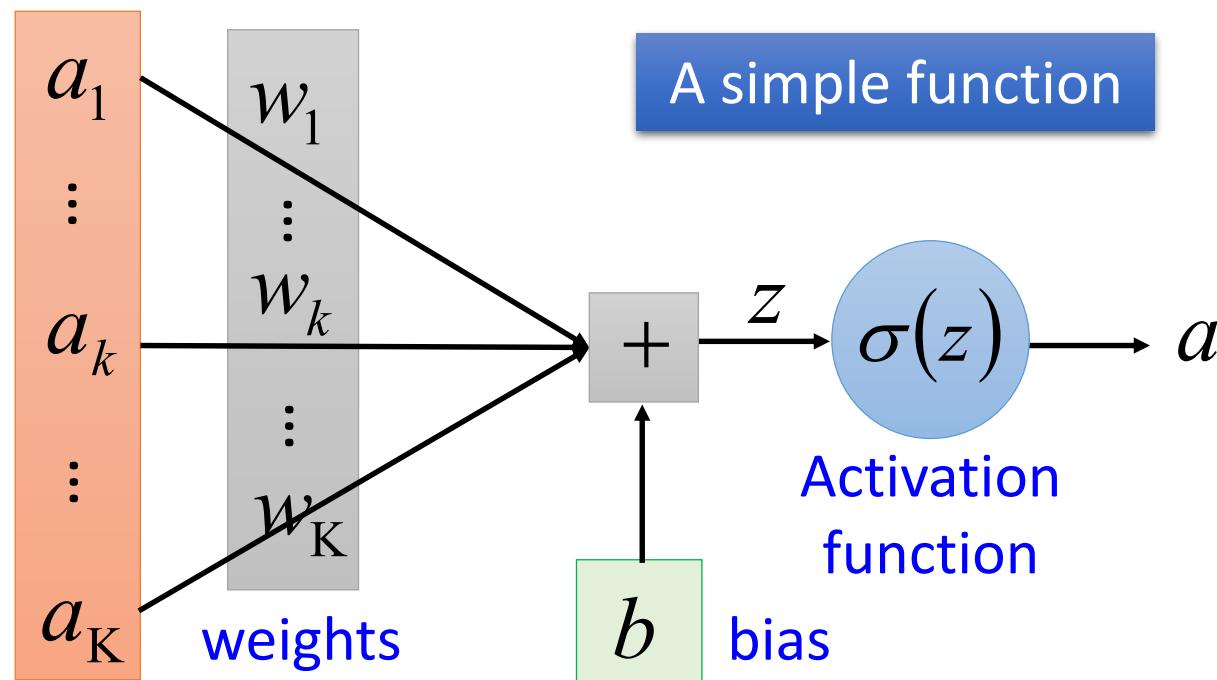
# Human Brains



# Neural Network

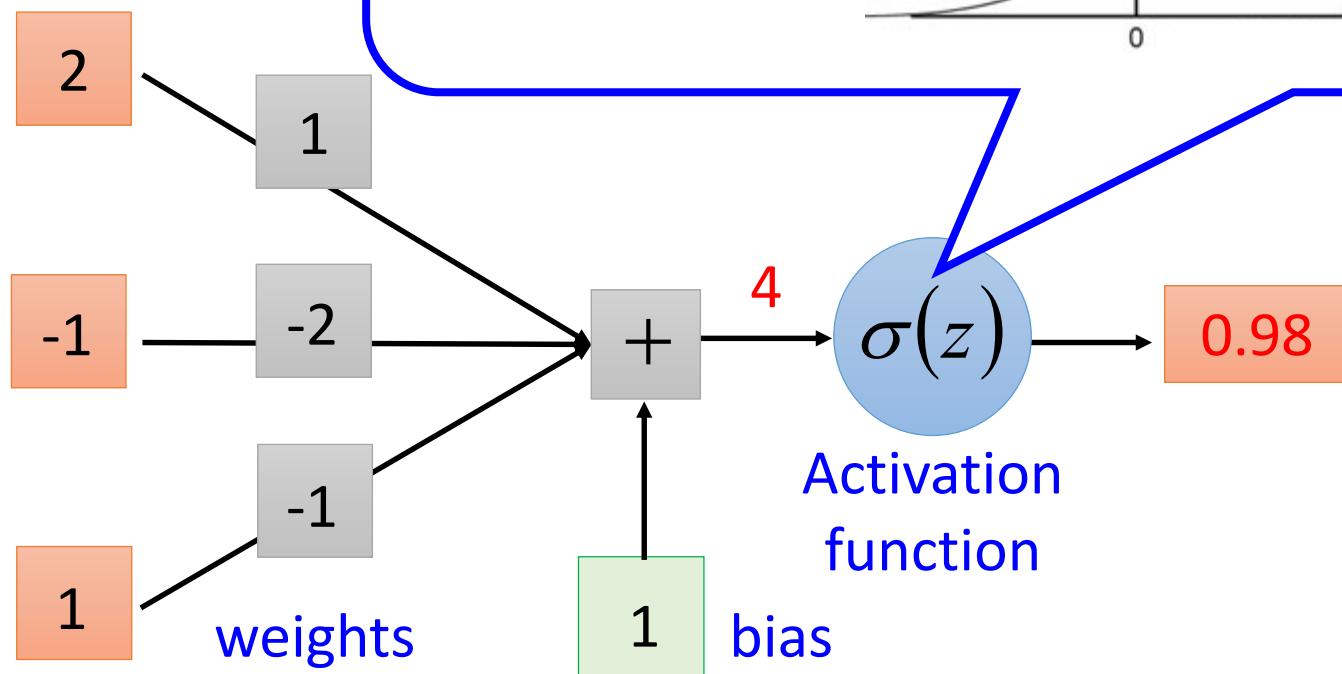
## Neuron

$$z = a_1 w_1 + \dots + a_k w_k + \dots + a_K w_K + b$$



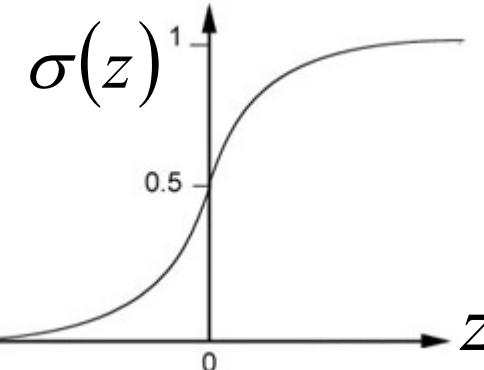
# Neural Network

## Neuron



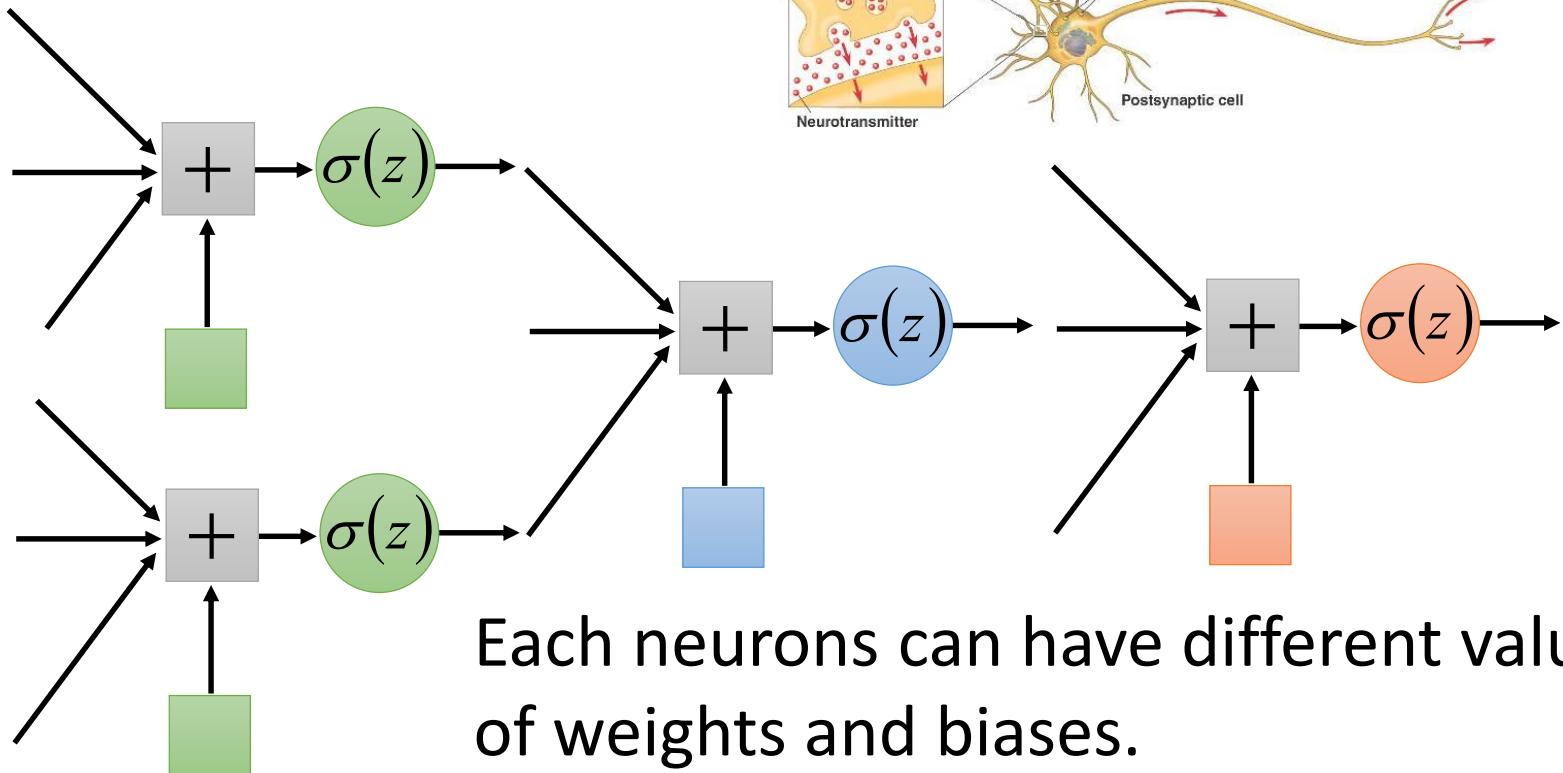
Sigmoid Function

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



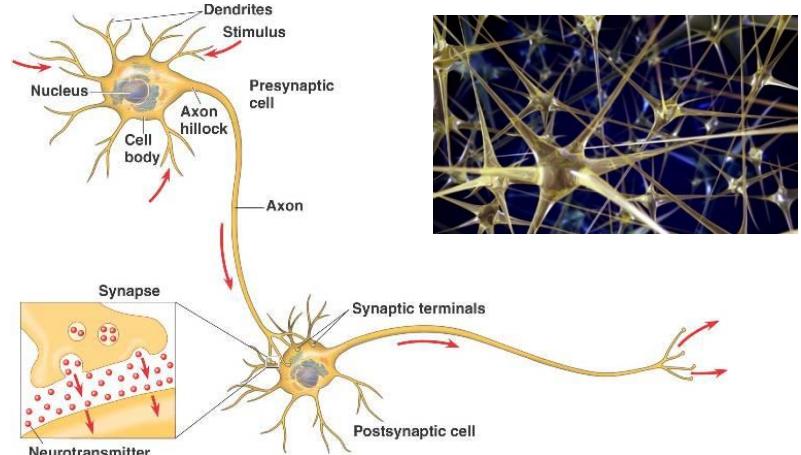
# Neural Network

Different connections leads to different network structure

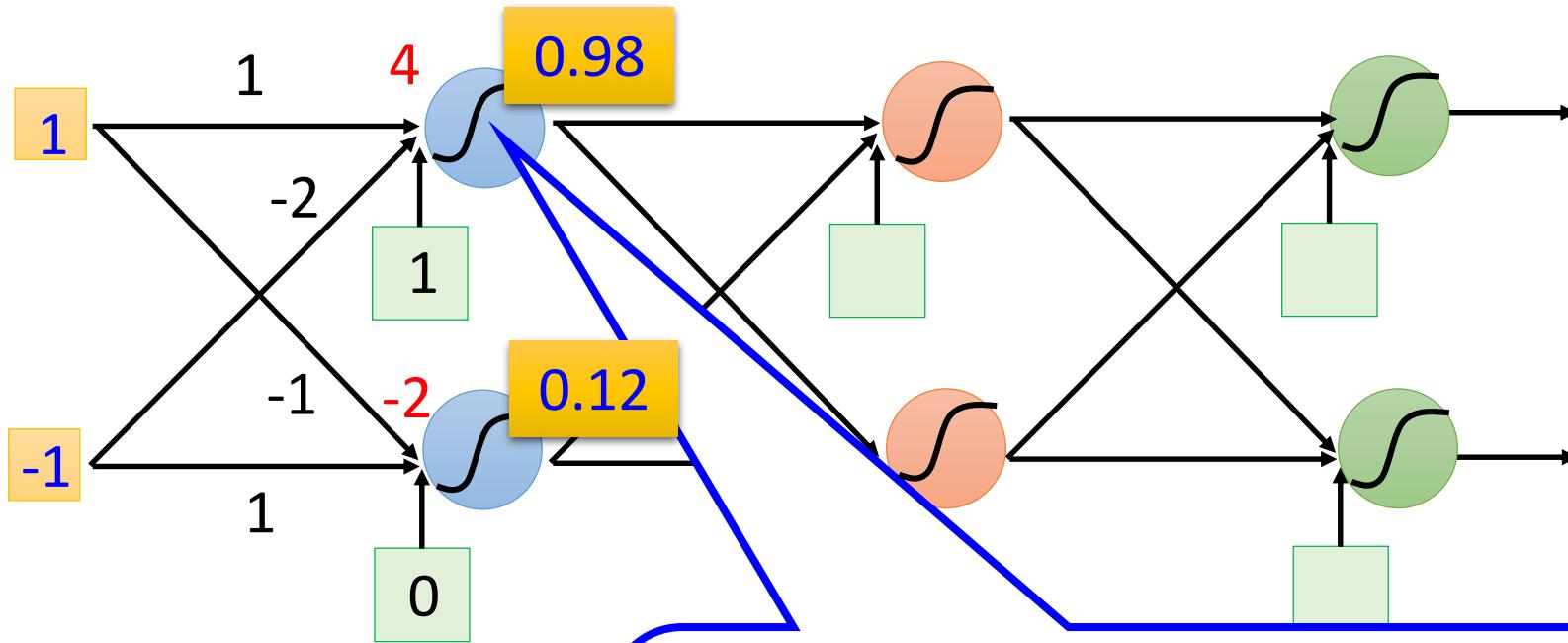


Each neurons can have different values of weights and biases.

Weights and biases are network parameters  $\theta$

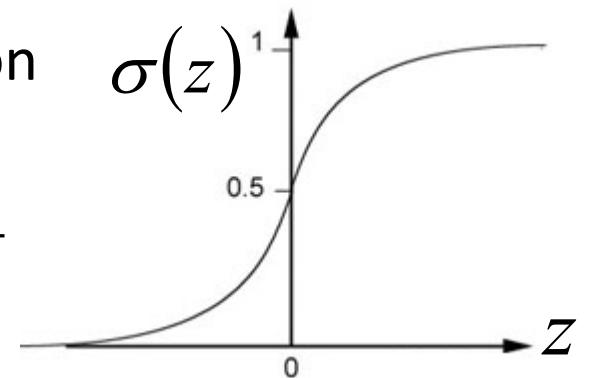


# Fully Connect Feedforward Network

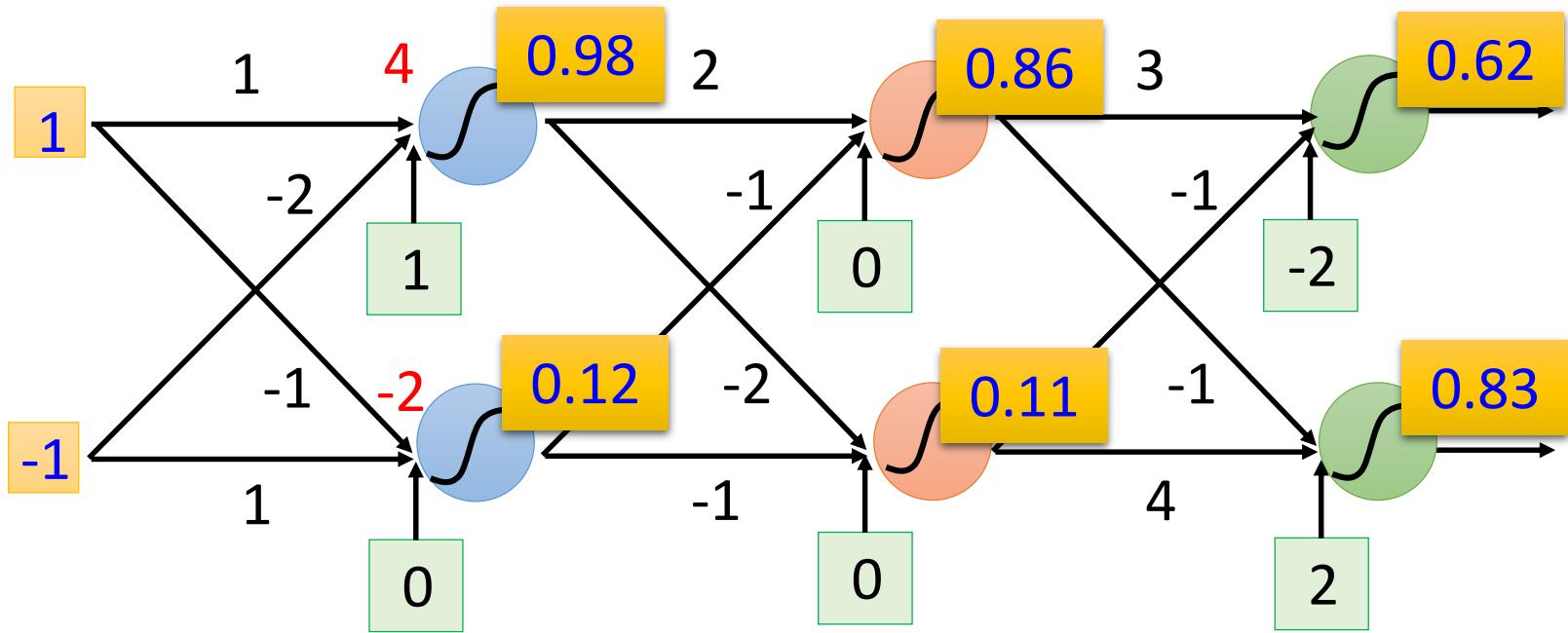


Sigmoid Function

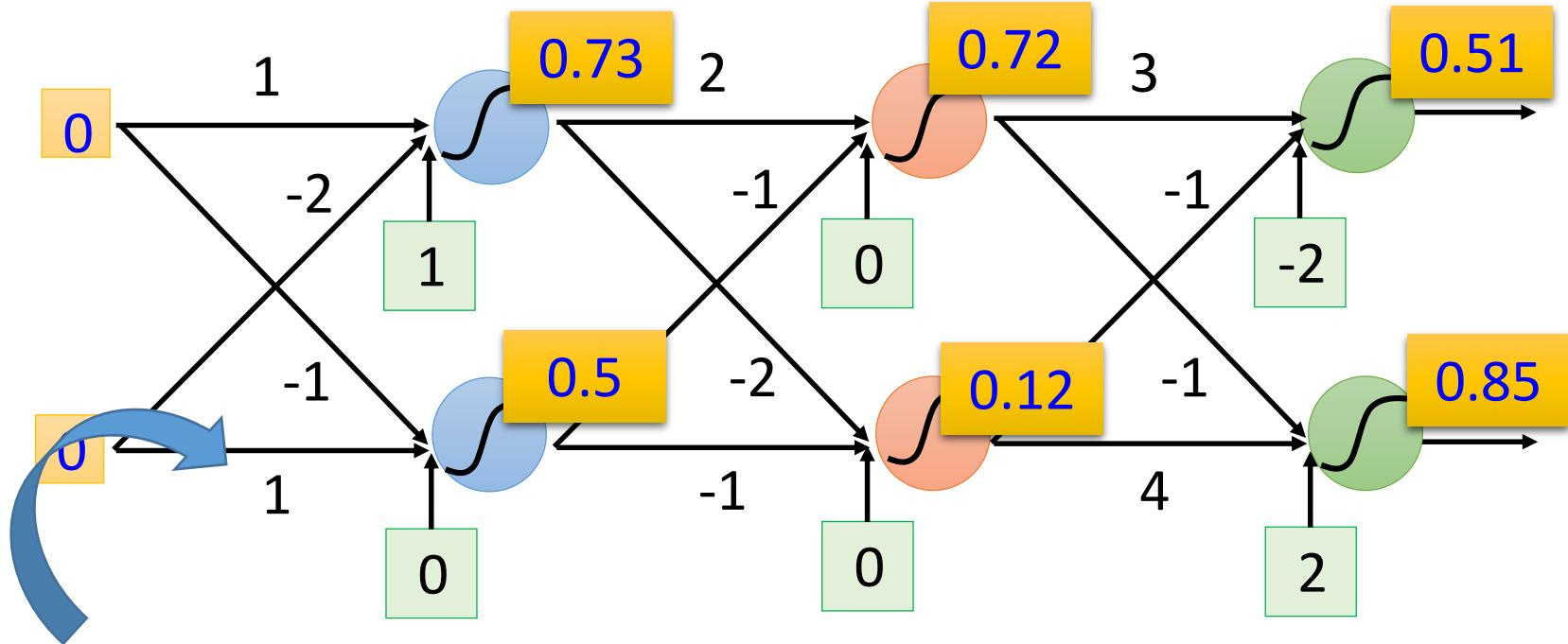
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



# Fully Connect Feedforward Network



# Fully Connect Feedforward Network



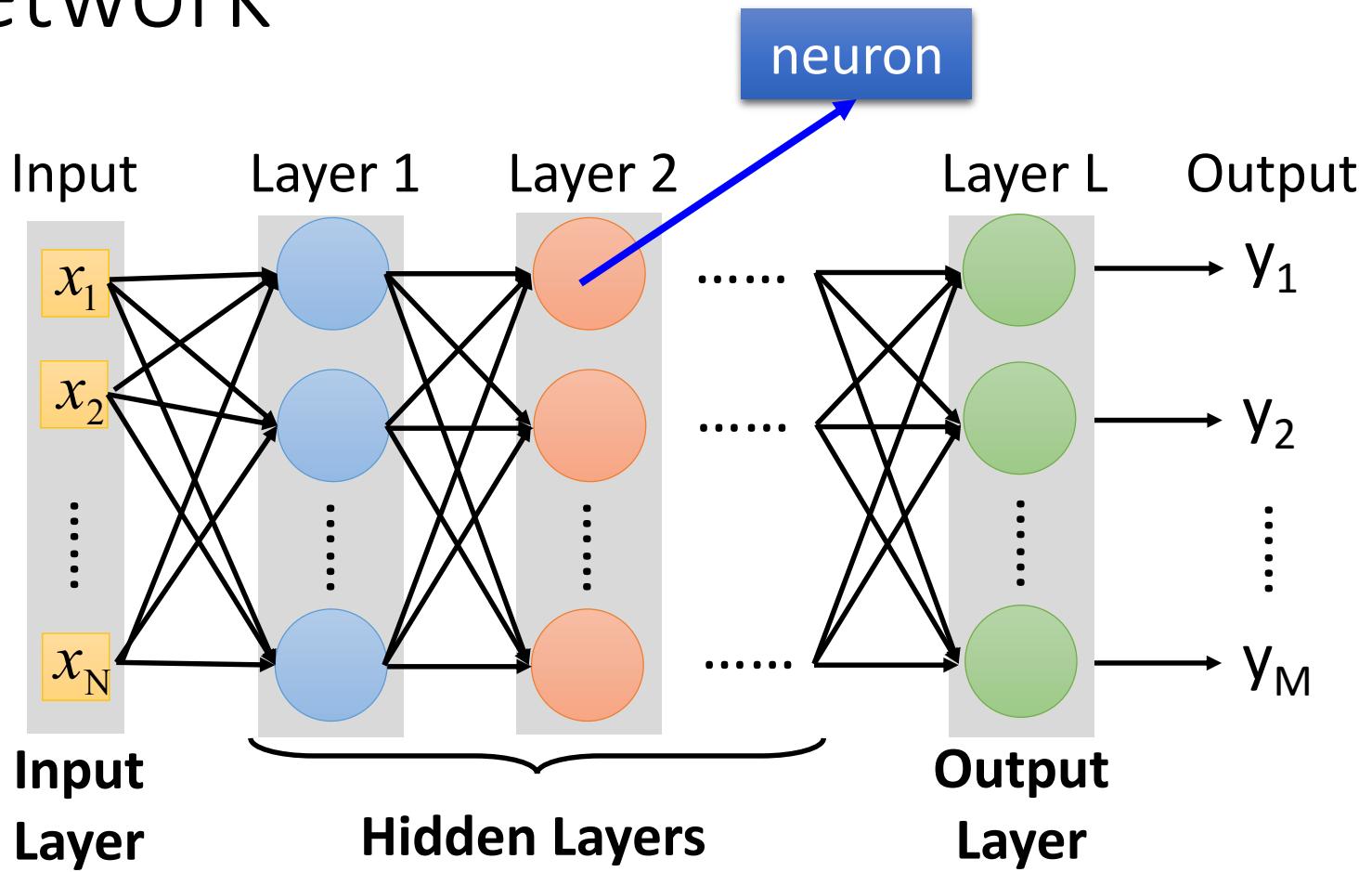
This is a function.  
Input vector, output vector

$$f \left( \begin{bmatrix} 1 \\ -1 \end{bmatrix} \right) = \begin{bmatrix} 0.62 \\ 0.83 \end{bmatrix} \quad f \left( \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} 0.51 \\ 0.85 \end{bmatrix}$$

Given parameters  $\theta$ , define a function

Given network structure, define a function set

# Fully Connect Feedforward Network



Deep means many hidden layers

# Output Layer (Option)

- Softmax layer as the output layer

## Ordinary Layer

$$z_1 \rightarrow \sigma \rightarrow y_1 = \sigma(z_1)$$

$$z_2 \rightarrow \sigma \rightarrow y_2 = \sigma(z_2)$$

$$z_3 \rightarrow \sigma \rightarrow y_3 = \sigma(z_3)$$

In general, the output of network can be any value.

May not be easy to interpret

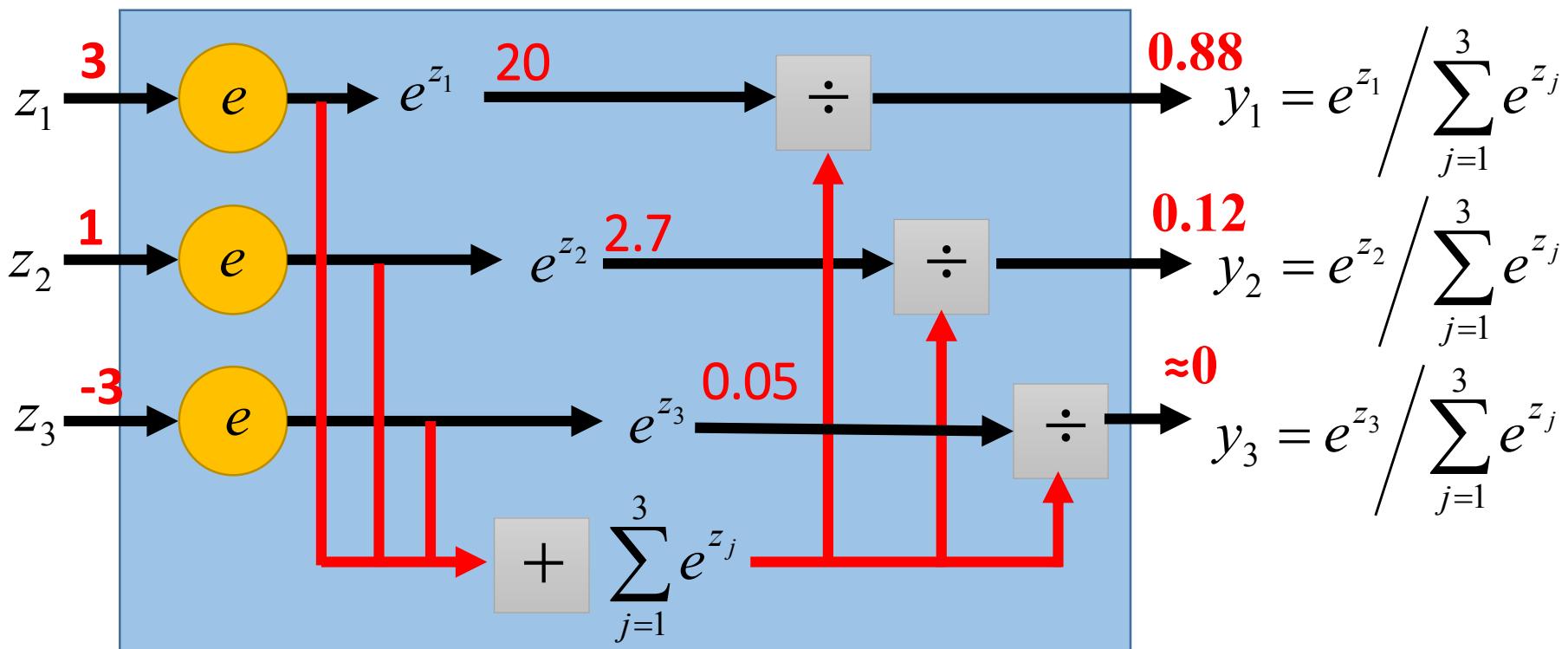
# Output Layer (Option)

- Softmax layer as the output layer

**Probability:**

- $1 > y_i > 0$
- $\sum_i y_i = 1$

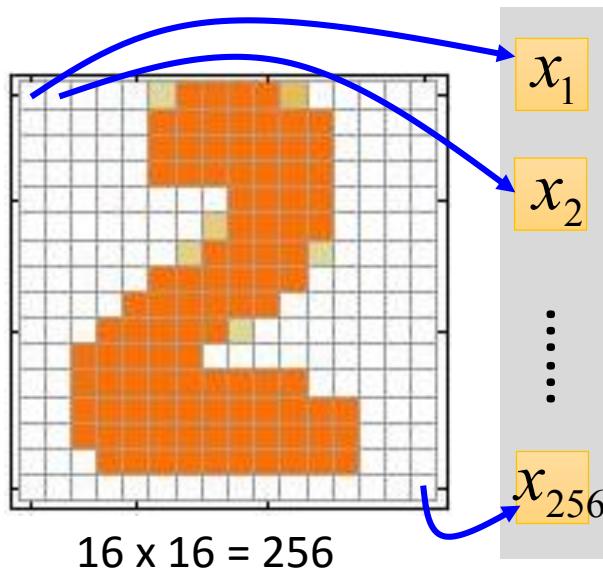
**Softmax Layer**



# Example Application



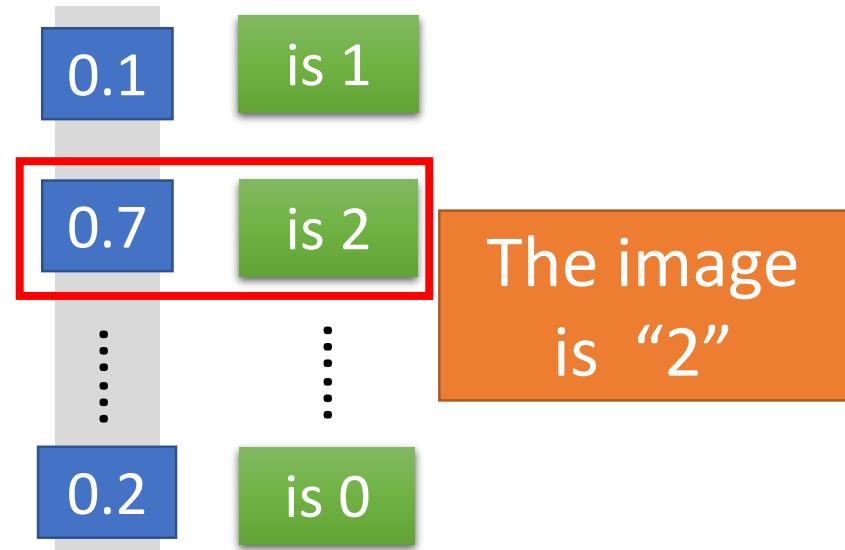
## Input



Ink  $\rightarrow$  1

No ink  $\rightarrow$  0

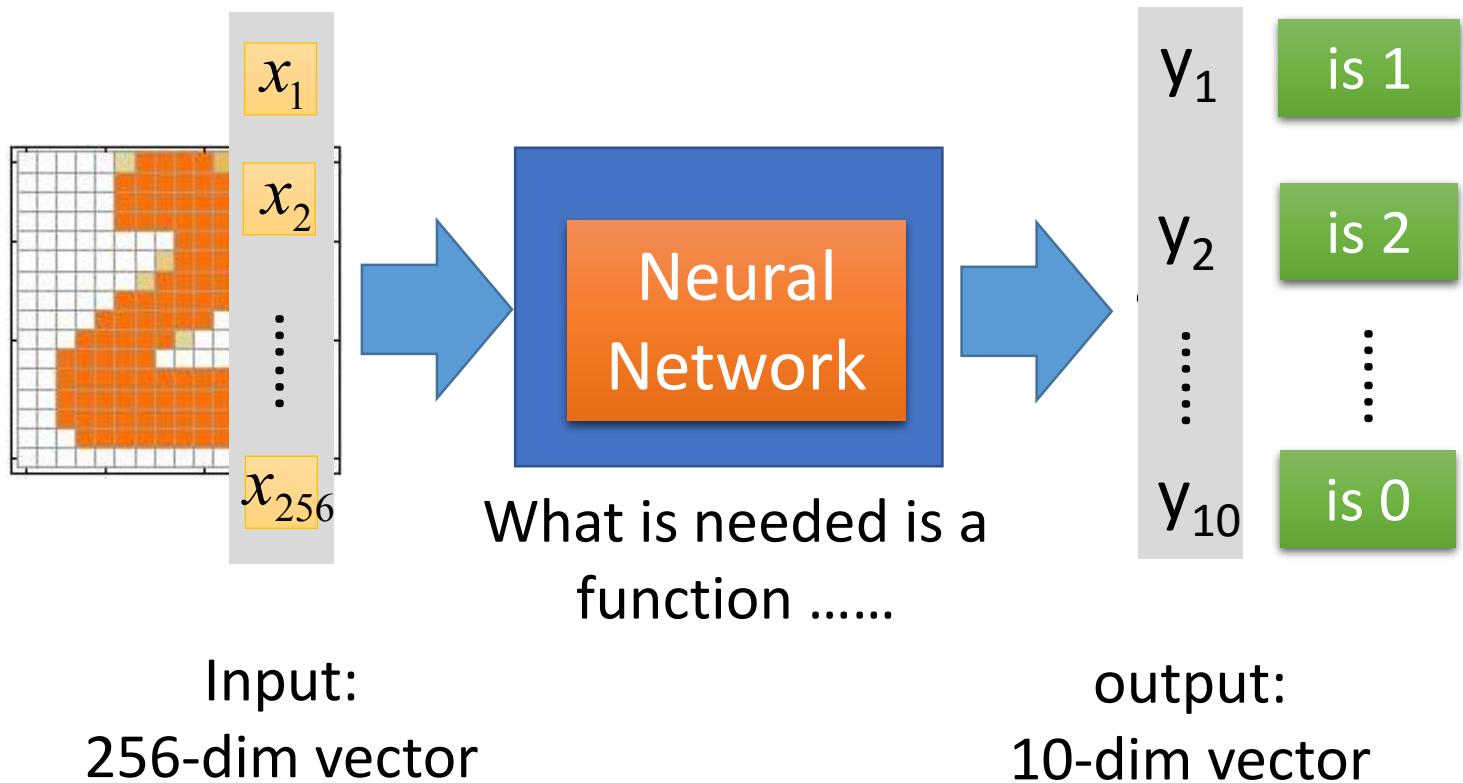
## Output



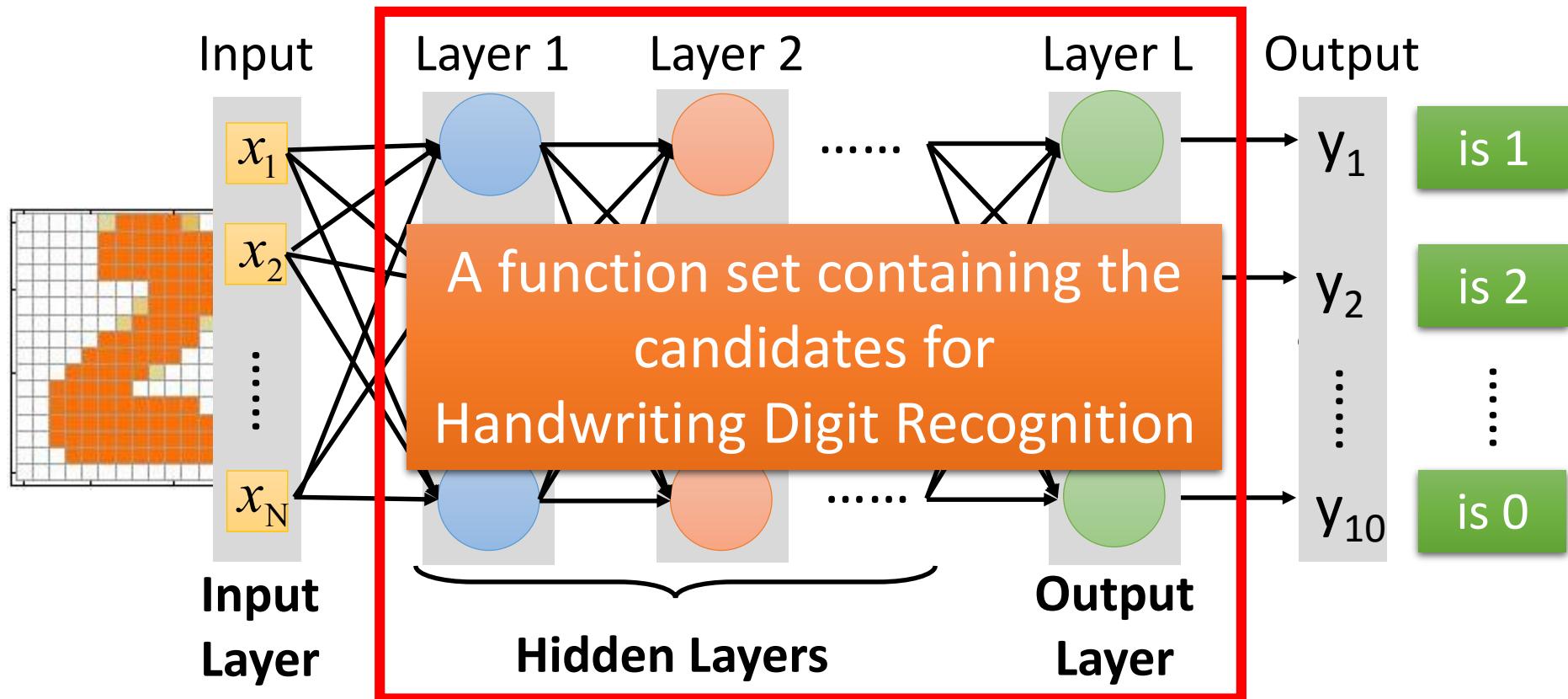
Each dimension represents the confidence of a digit.

# Example Application

- Handwriting Digit Recognition

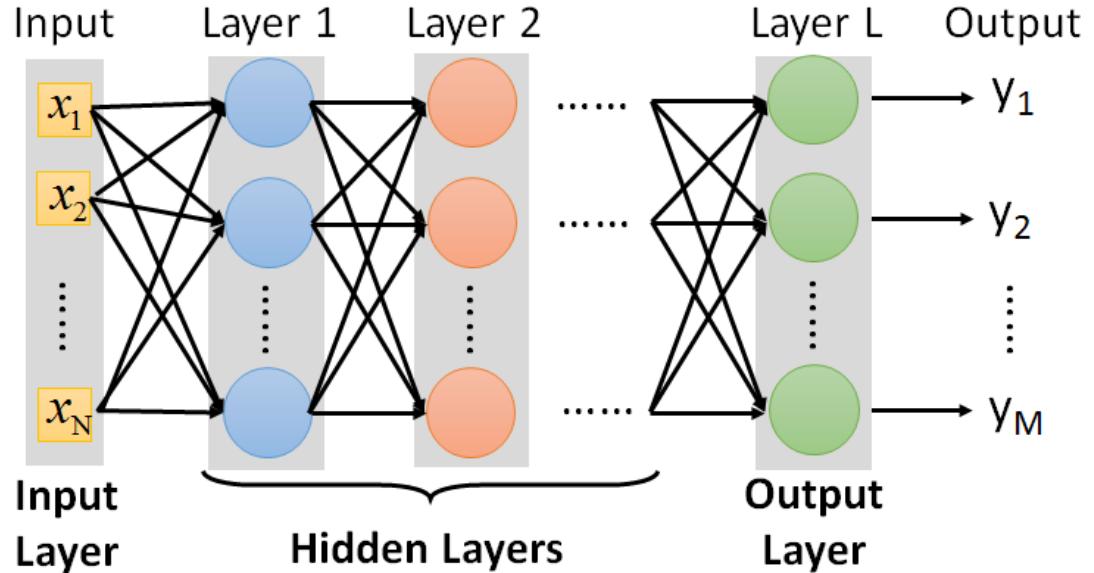


# Example Application



You need to decide the network structure to let a good function in your function set.

# FAQ



- Q: How many layers? How many neurons for each layer?

Trial and Error

+

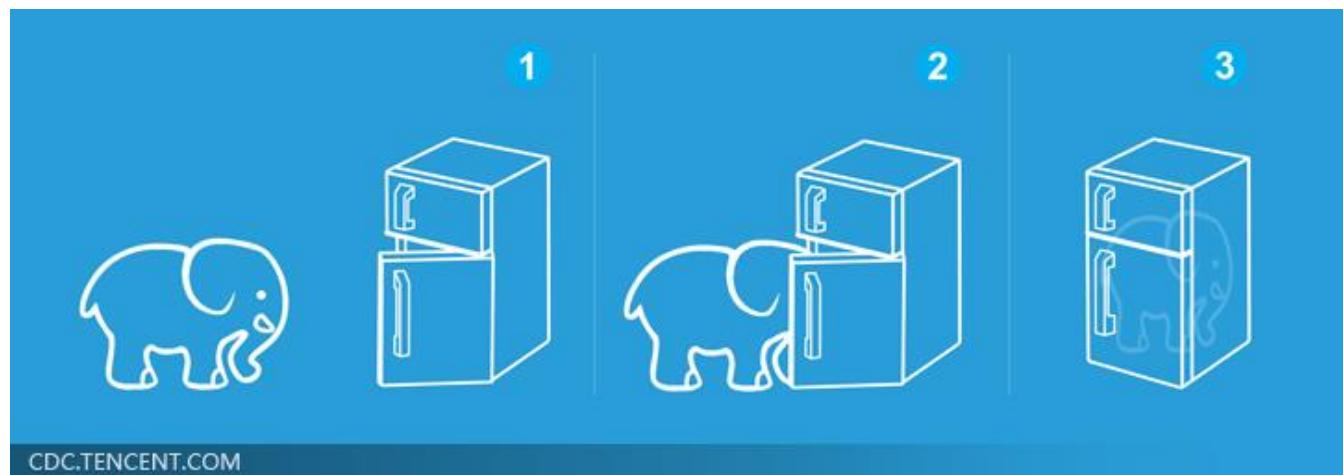
Intuition

- Q: Can the structure be automatically determined?

# Three Steps for Deep Learning



Deep Learning is so simple .....



# Training Data

- Preparing training data: images and their labels



“5”



“0”



“4”



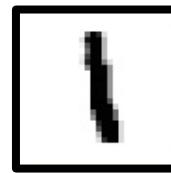
“1”



“9”



“2”



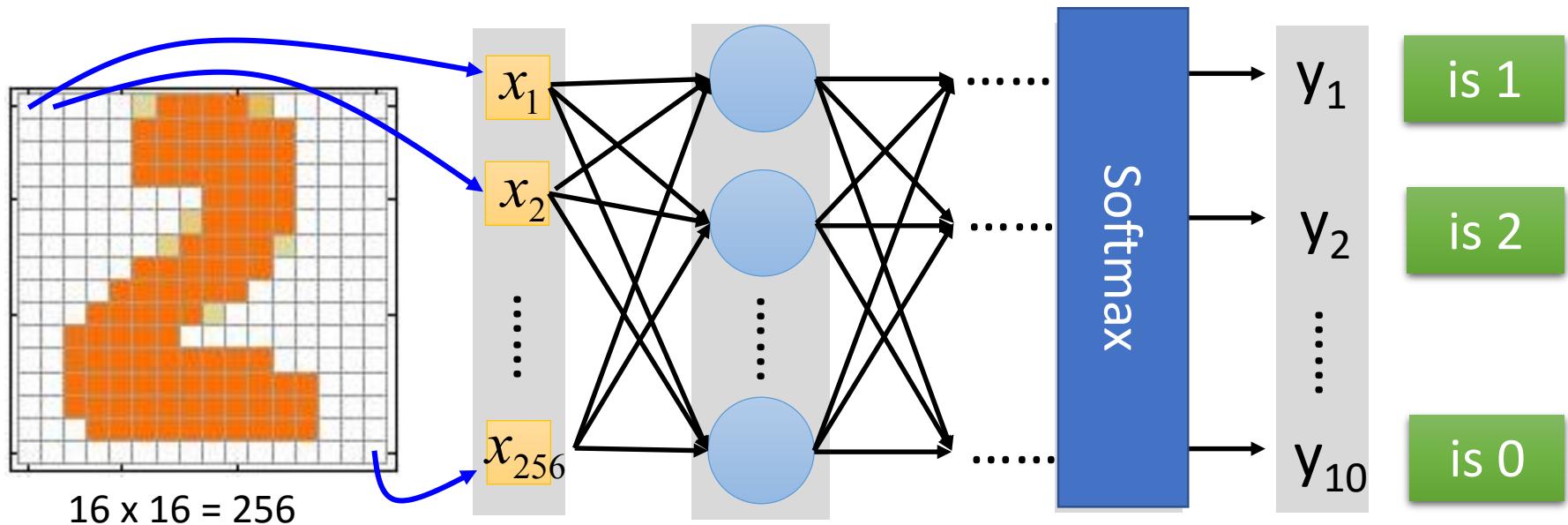
“1”



“3”

The learning target is defined on  
the training data.

# Learning Target



Ink  $\rightarrow 1$

No ink  $\rightarrow 0$

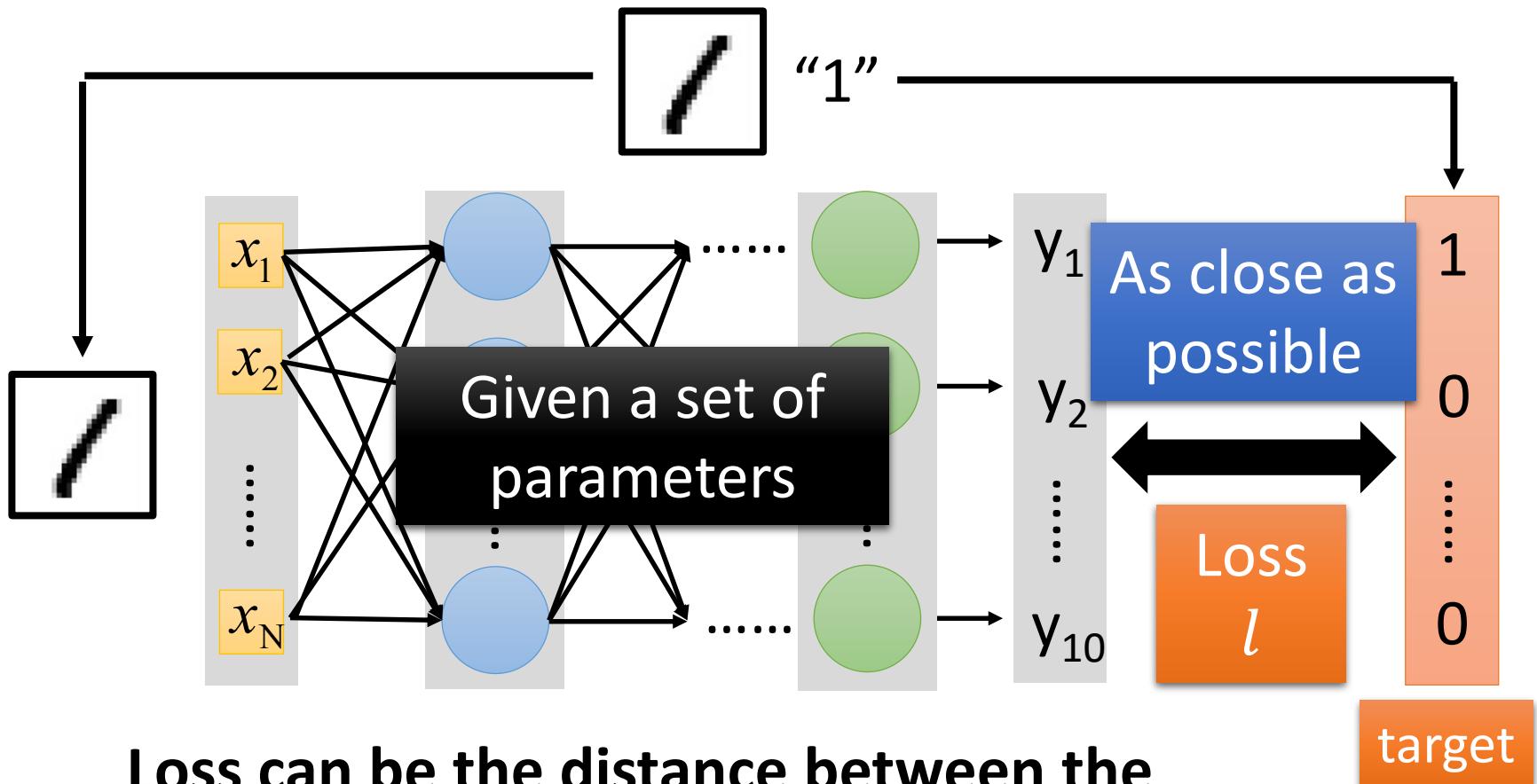
The learning target is .....

Input:  $\rightarrow y_1$  has the maximum value

Input:  $\rightarrow y_2$  has the maximum value

# LOSS

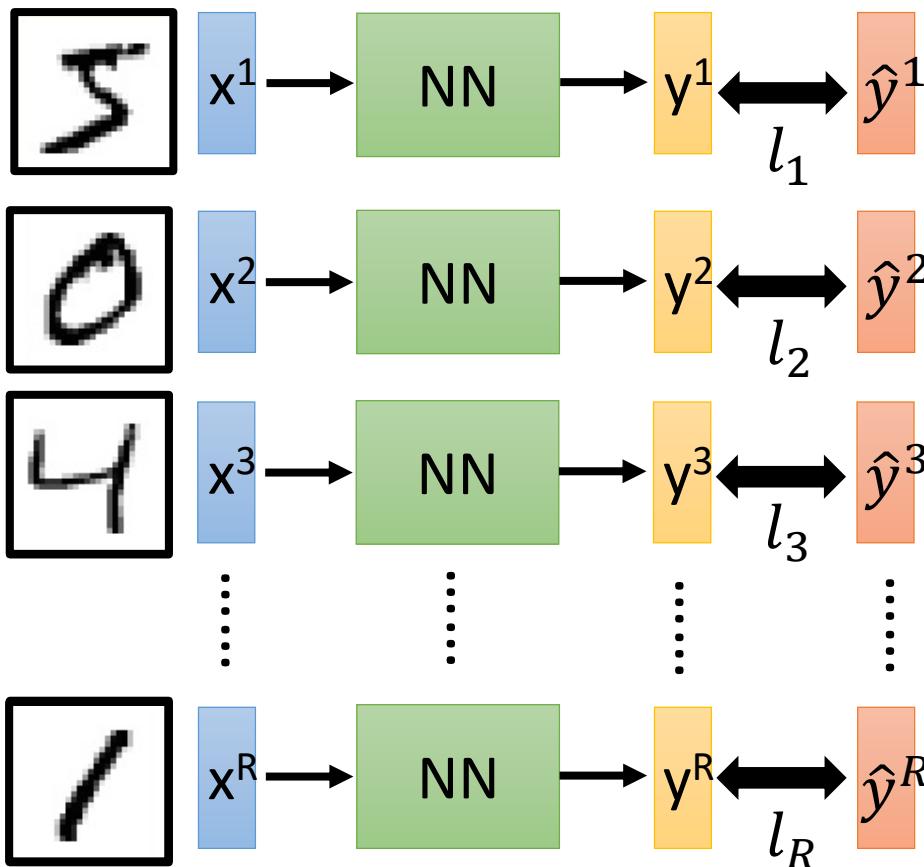
A good function should make the loss of all examples as small as possible.



Loss can be the distance between the network output and target

# Total Loss

For all training data ...



Total Loss:

$$L = \sum_{r=1}^R l_r$$

As small as possible

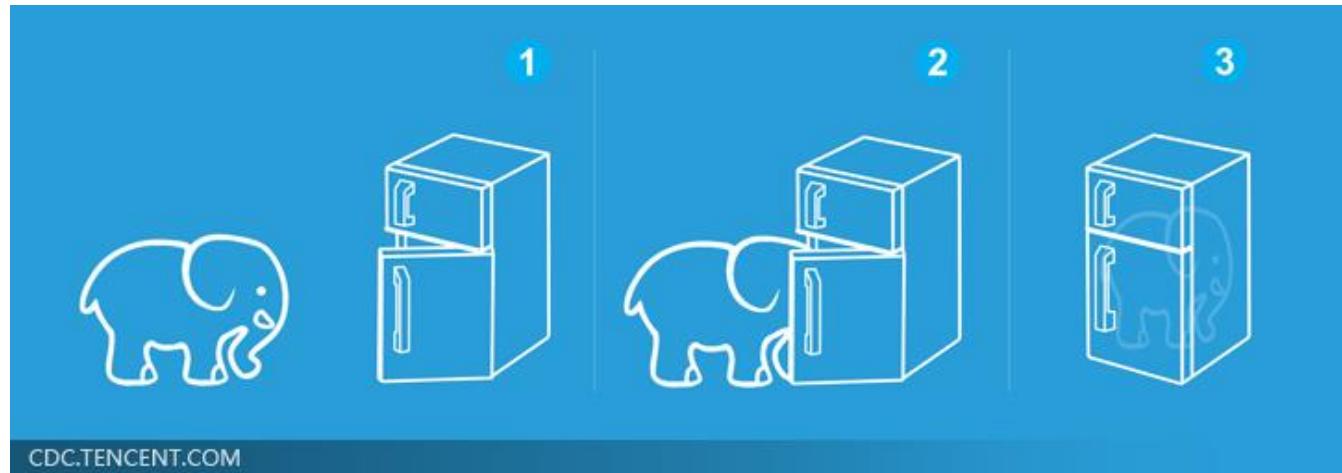
Find a function in  
function set that  
minimizes total loss  $L$

Find the network  
parameters  $\theta^*$  that  
minimize total loss  $L$

# Three Steps for Deep Learning



Deep Learning is so simple .....



# How to pick the best function

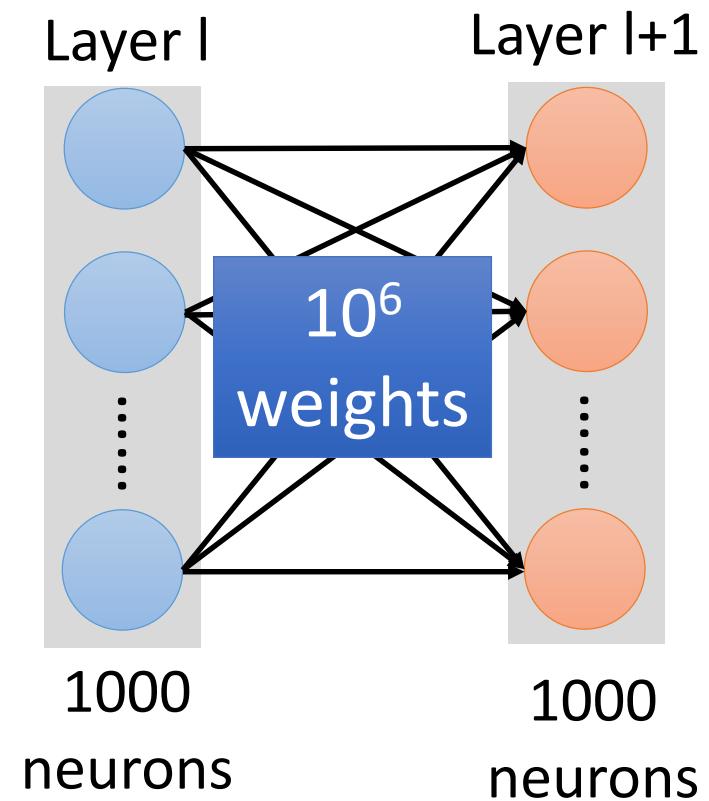
Find network parameters  $\theta^*$  that minimize total loss L

Enumerate all possible values

Network parameters  $\theta =$   
 $\{w_1, w_2, w_3, \dots, b_1, b_2, b_3, \dots\}$

Millions of parameters

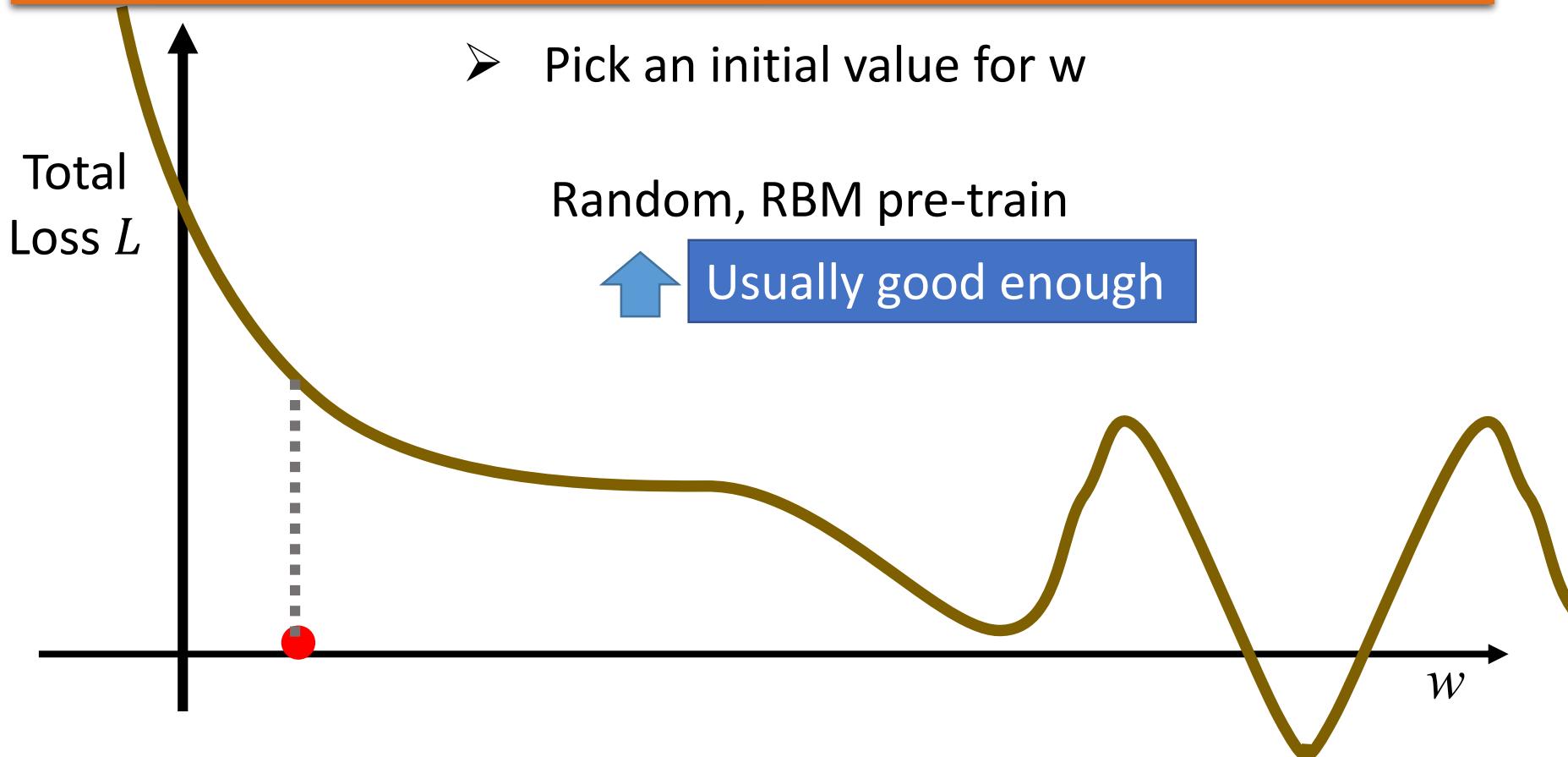
E.g. speech recognition: 8 layers and  
1000 neurons each layer



# Gradient Descent

Network parameters  $\theta = \{w_1, w_2, \dots, b_1, b_2, \dots\}$

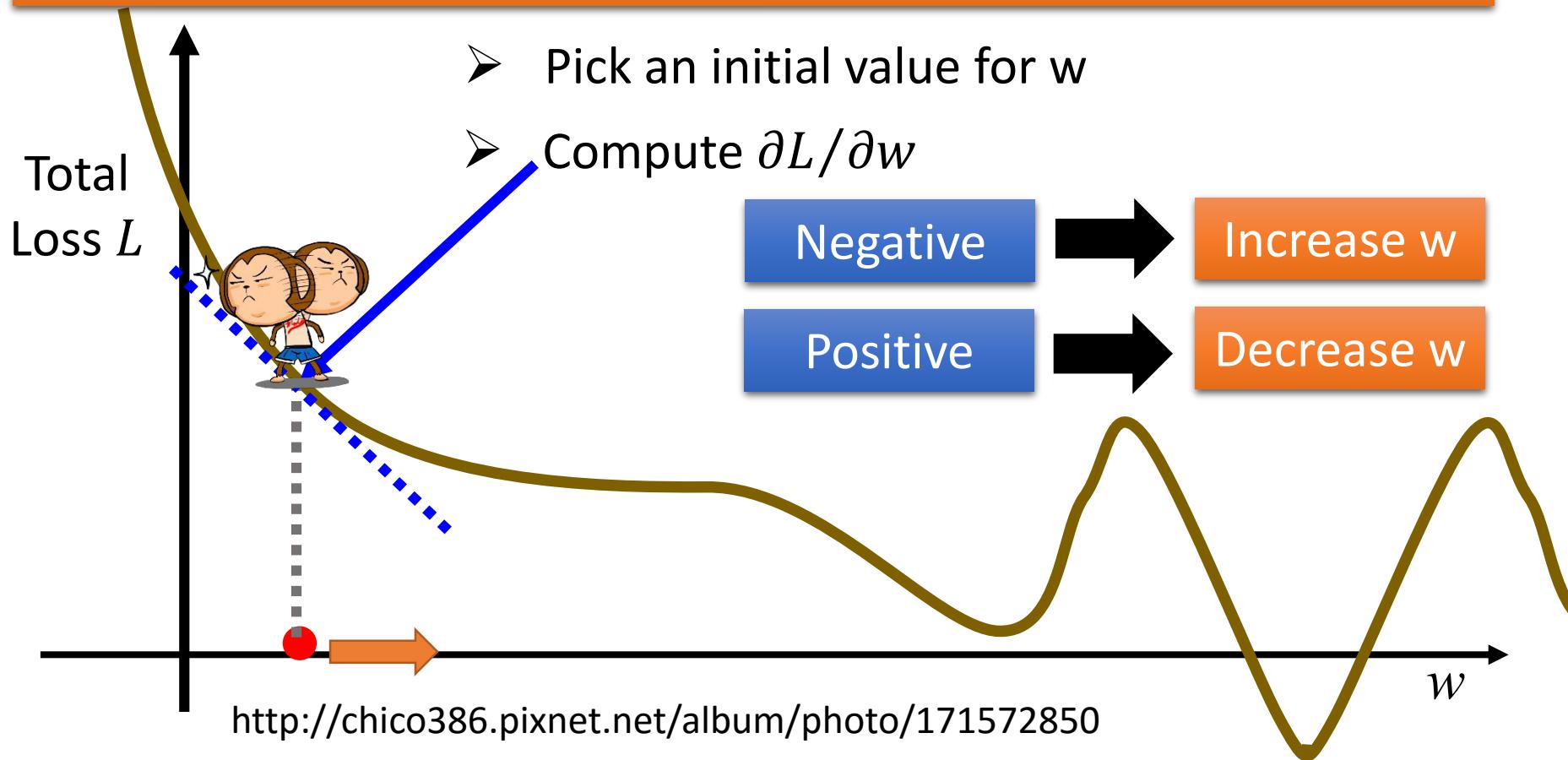
Find network parameters  $\theta^*$  that minimize total loss L



# Gradient Descent

Network parameters  $\theta = \{w_1, w_2, \dots, b_1, b_2, \dots\}$

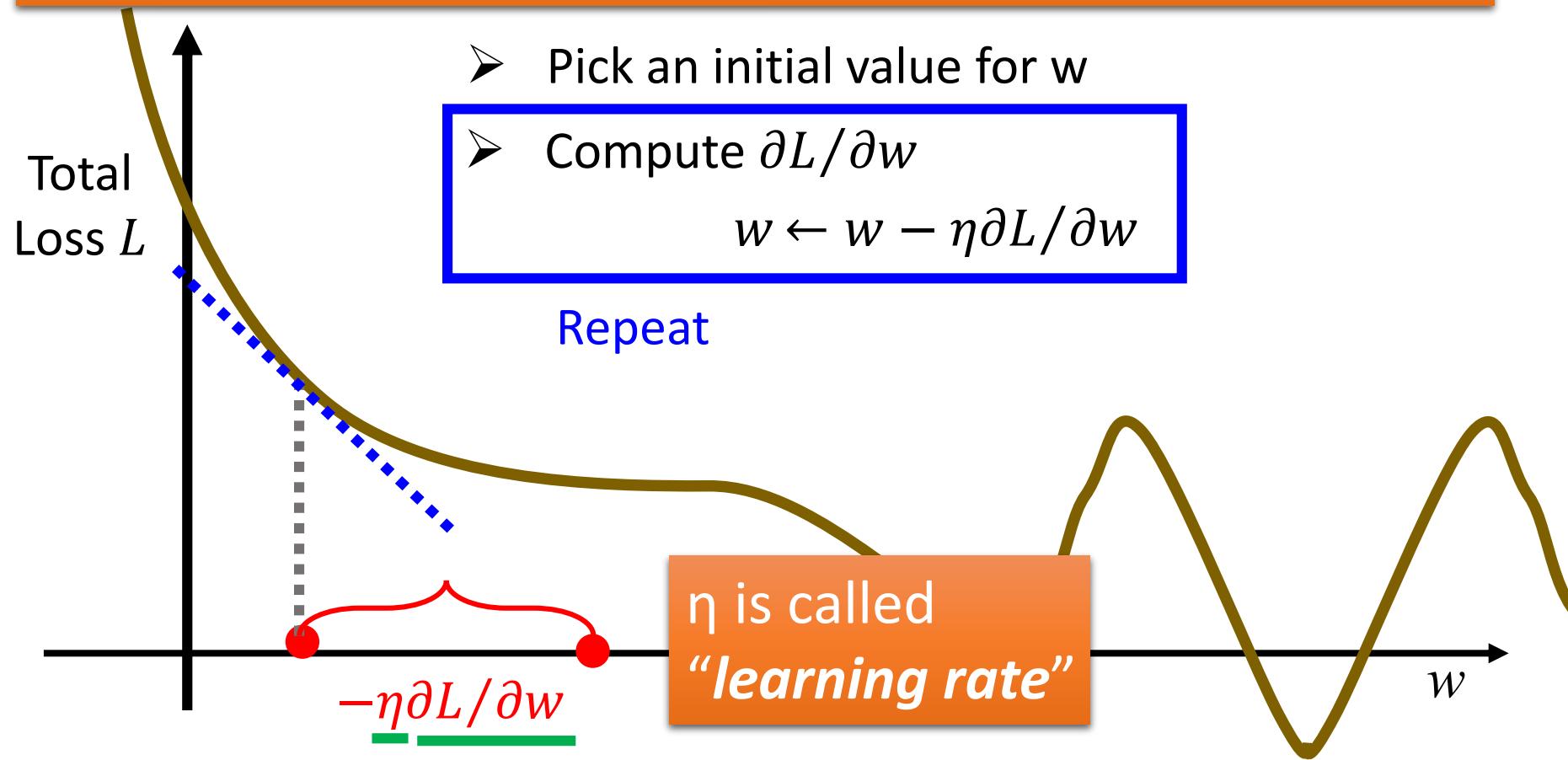
Find network parameters  $\theta^*$  that minimize total loss L



# Gradient Descent

Network parameters  $\theta = \{w_1, w_2, \dots, b_1, b_2, \dots\}$

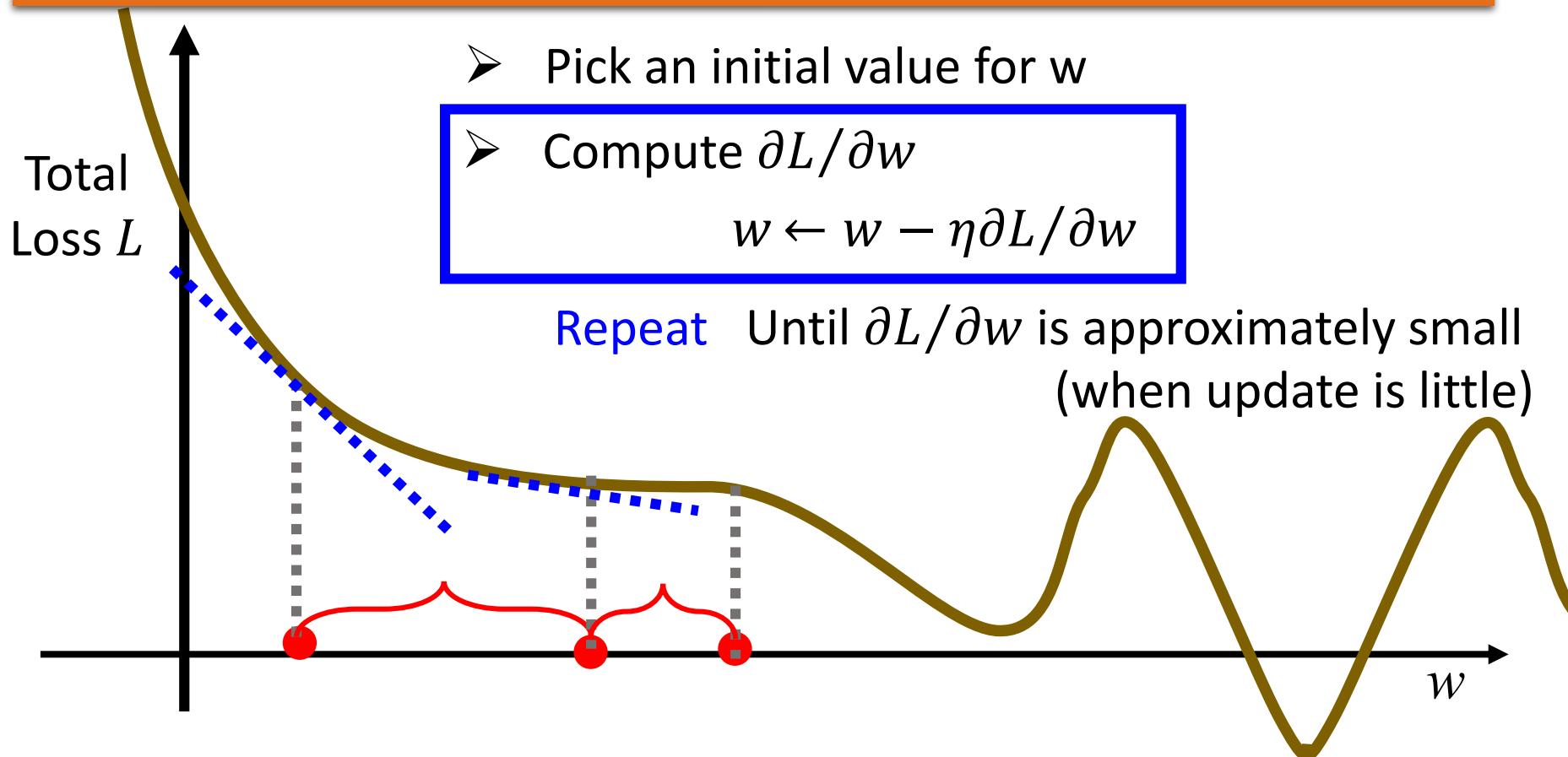
Find network parameters  $\theta^*$  that minimize total loss L



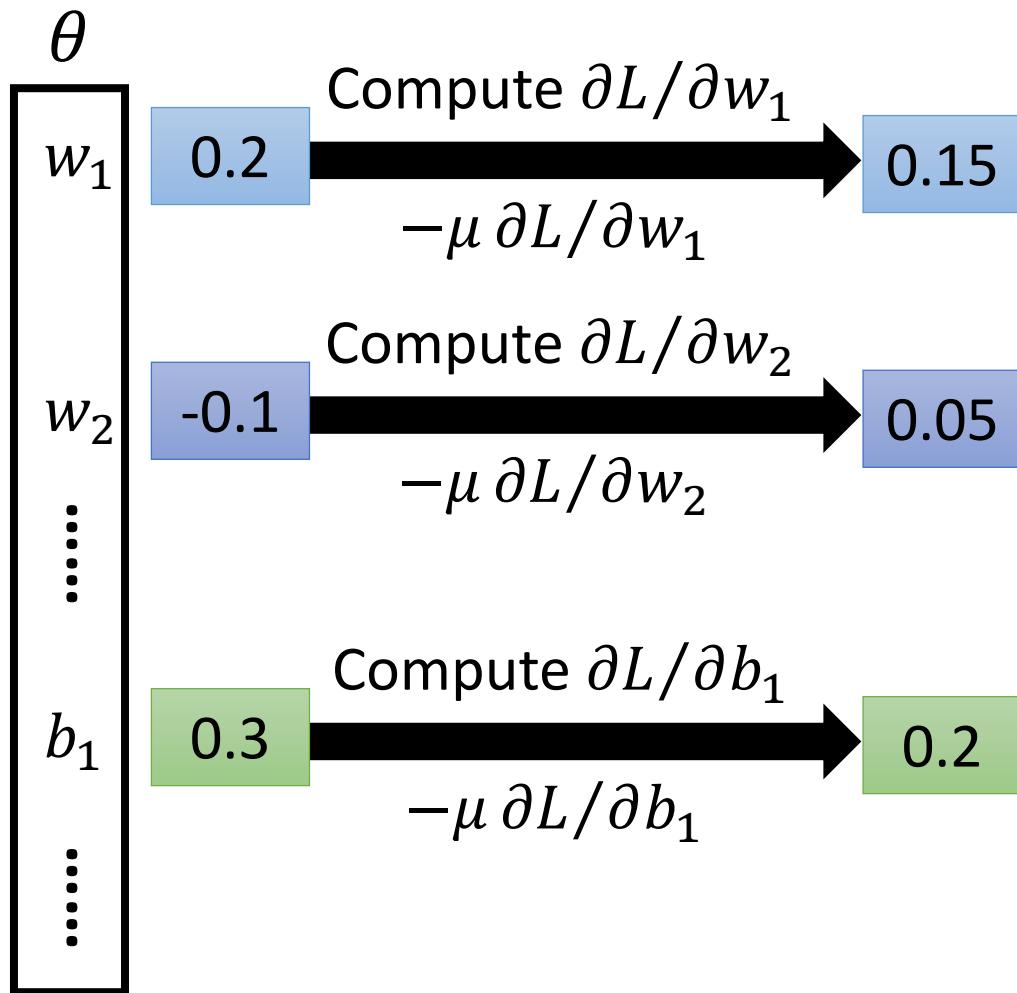
# Gradient Descent

Network parameters  $\theta = \{w_1, w_2, \dots, b_1, b_2, \dots\}$

Find network parameters  $\theta^*$  that minimize total loss L



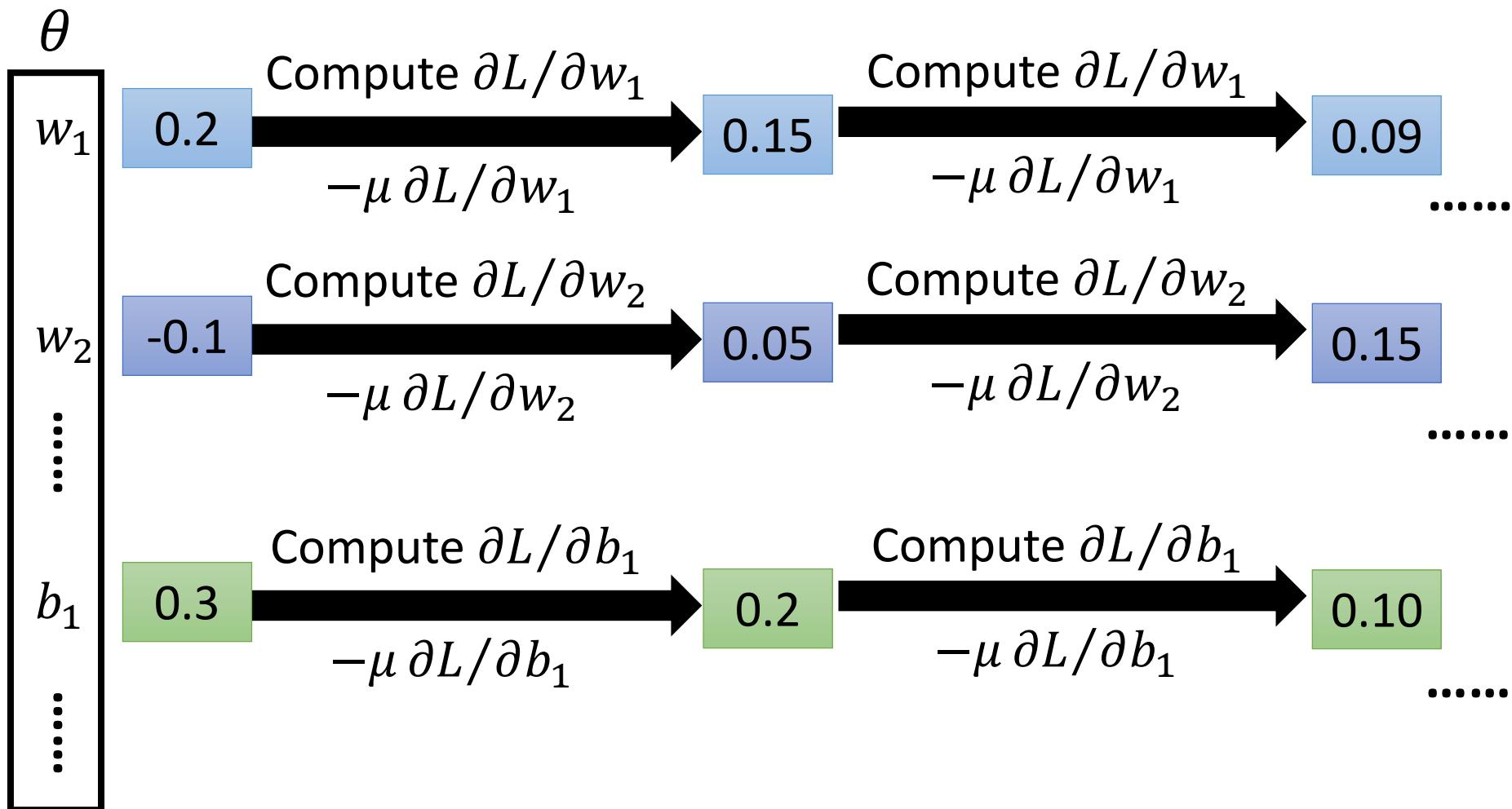
# Gradient Descent



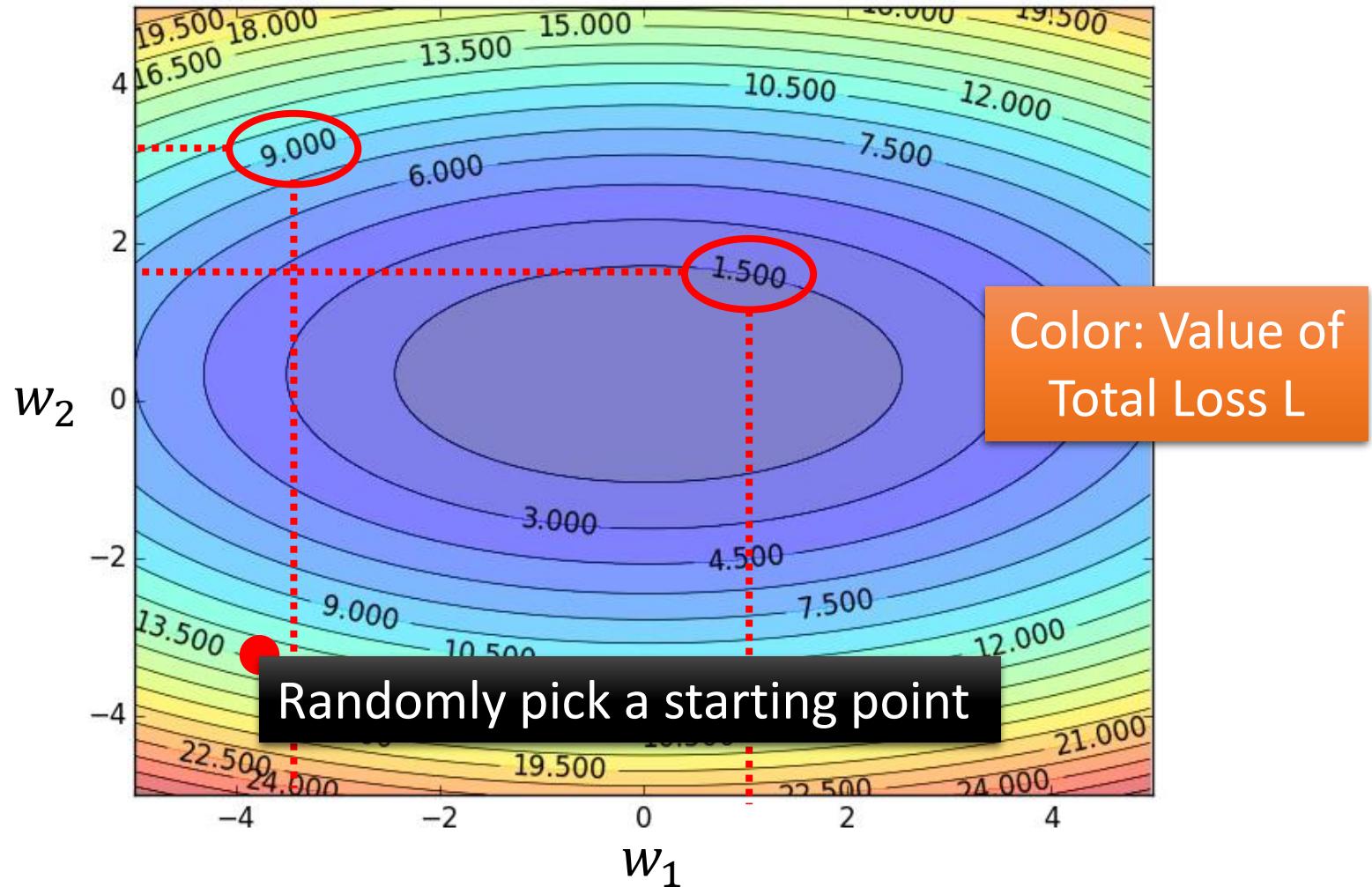
$$\nabla L = \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \vdots \\ \frac{\partial L}{\partial b_1} \\ \vdots \end{bmatrix}$$

gradient

# Gradient Descent

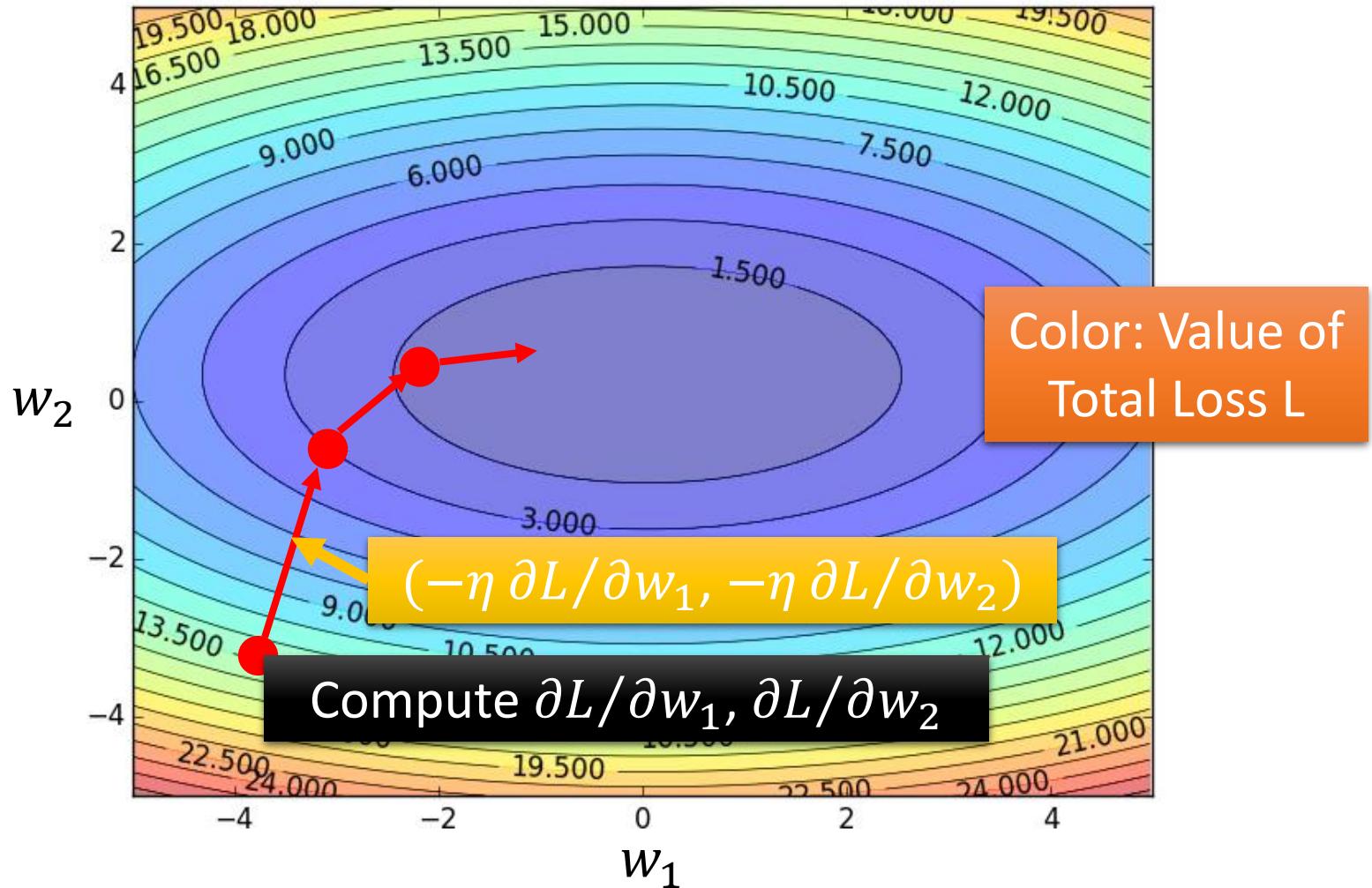


# Gradient Descent



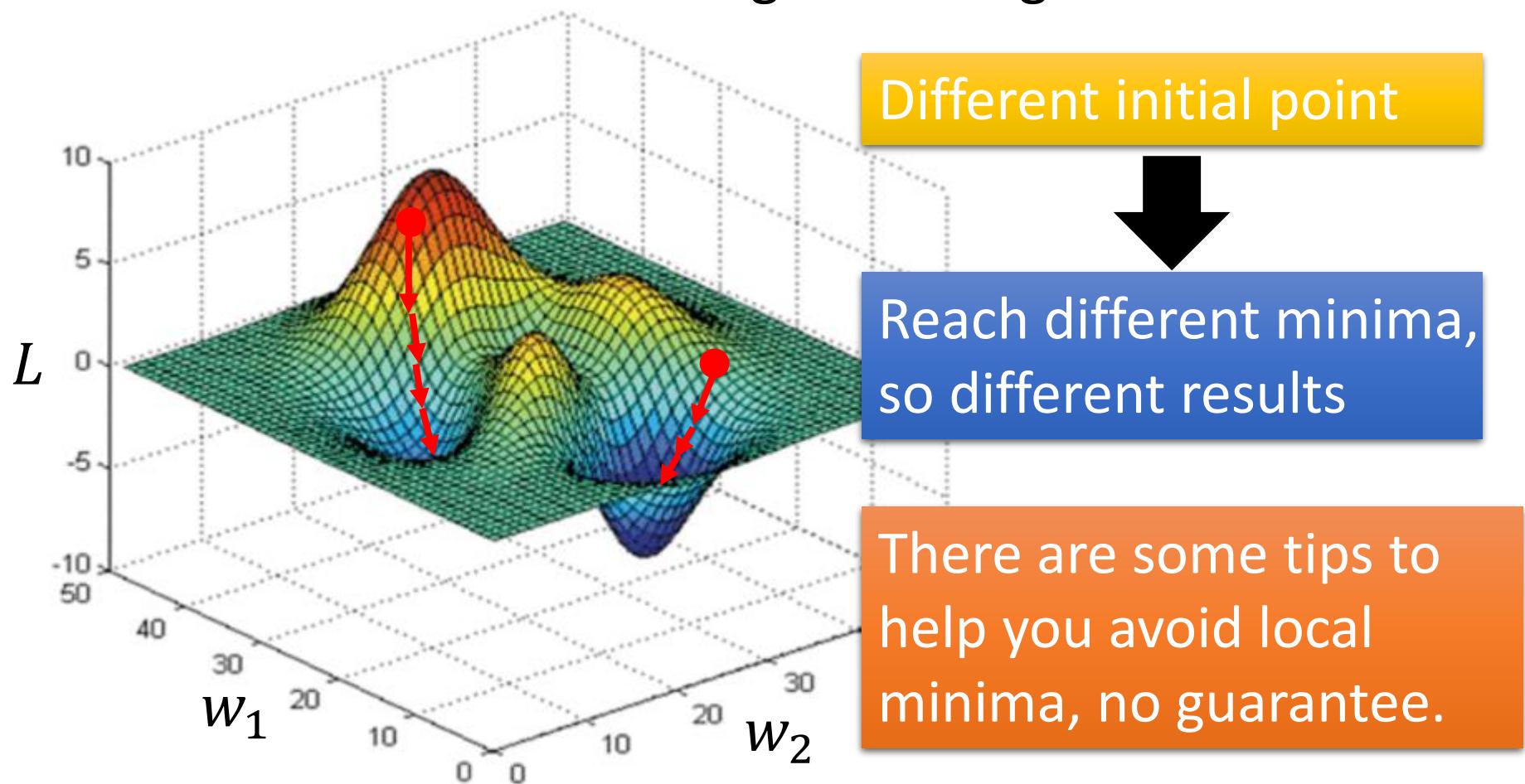
# Gradient Descent

Hopfully, we would reach  
a minima .....



# Gradient Descent - Difficulty

- Gradient descent never guarantee global minima



You are playing Age of Empires ...

You cannot see the whole map.

$$(-\eta \partial L / \partial w_1, -\eta \partial L / \partial w_2)$$

Compute  $\partial L / \partial w_1, \partial L / \partial w_2$

$w_2$

$w_1$

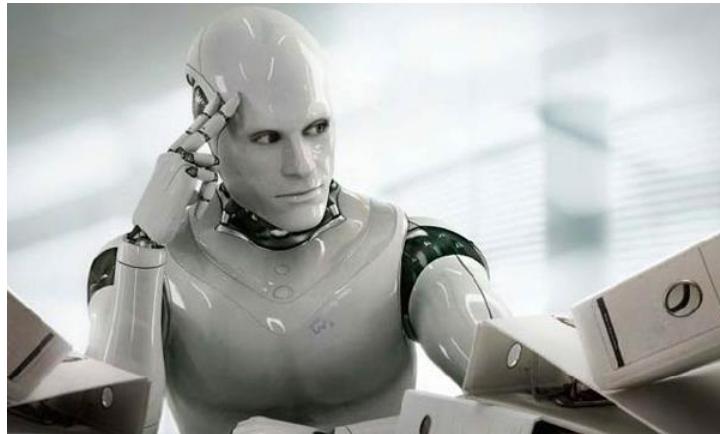


# Gradient Descent

This is the “learning” of machines in deep learning .....

→ Even alpha go using this approach.

People image .....



Actually .....



I hope you are not too disappointed :p

# Backpropagation

- Backpropagation: an efficient way to compute  $\partial L / \partial w$ 
  - Ref:  
[http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS\\_2015\\_2/Lecture/DNN%20backprop.ecm.mp4/index.html](http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/DNN%20backprop.ecm.mp4/index.html)



theano

**libdnn**  
台大周伯威  
同學開發

Caffe

Microsoft  
**CNTK**



**mxnet**

Don't worry about  $\partial L / \partial w$ , the toolkits will handle it.

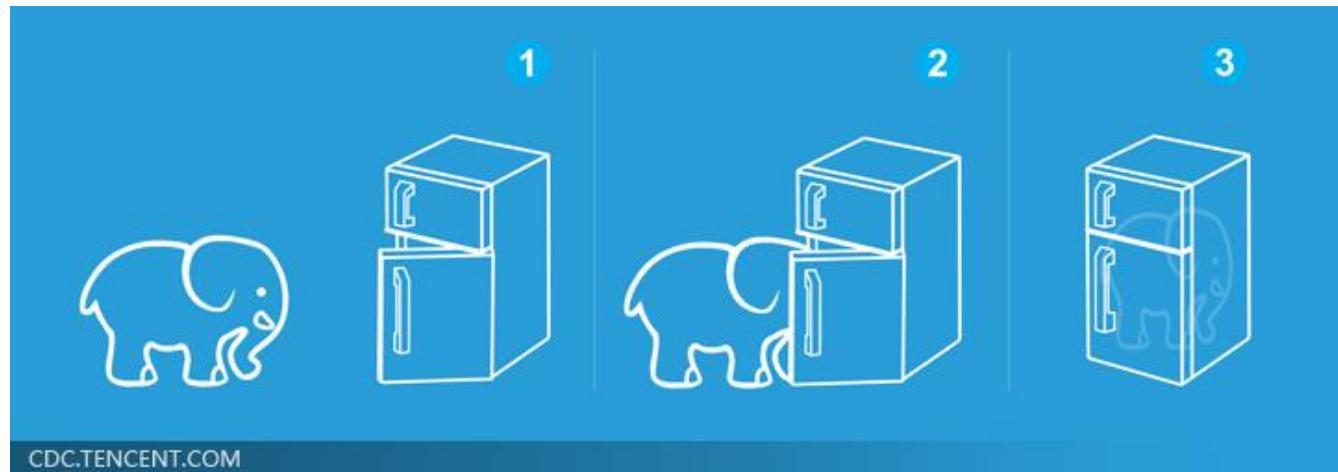
# Concluding Remarks

Step 1:  
define a set  
of function

Step 2:  
goodness of  
function

Step 3: pick  
the best  
function

Deep Learning is so simple .....



# Outline of Lecture I

Introduction of Deep Learning

Why Deep?

“Hello World” for Deep Learning

# Deeper is Better?

Layer X Size	Word Error Rate (%)
1 X 2k	24.2
2 X 2k	20.4
3 X 2k	18.4
4 X 2k	17.8
5 X 2k	17.2
7 X 2k	17.1

Not surprised, more parameters, better performance

Seide, Frank, Gang Li, and Dong Yu. "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks." *Interspeech*. 2011.

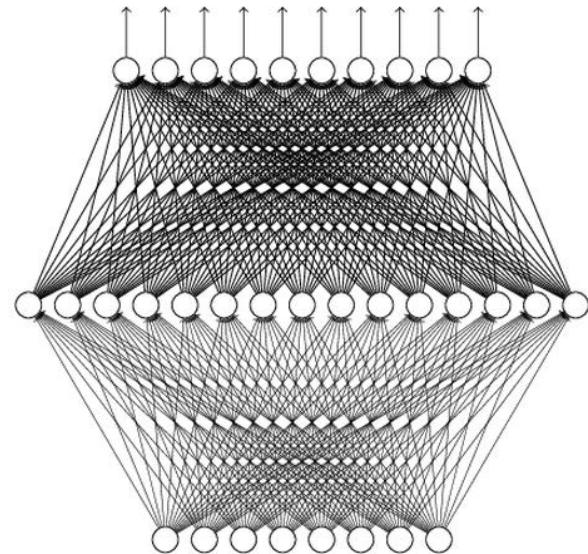
# Universality Theorem

Any continuous function  $f$

$$f : R^N \rightarrow R^M$$

Can be realized by a network  
with one hidden layer

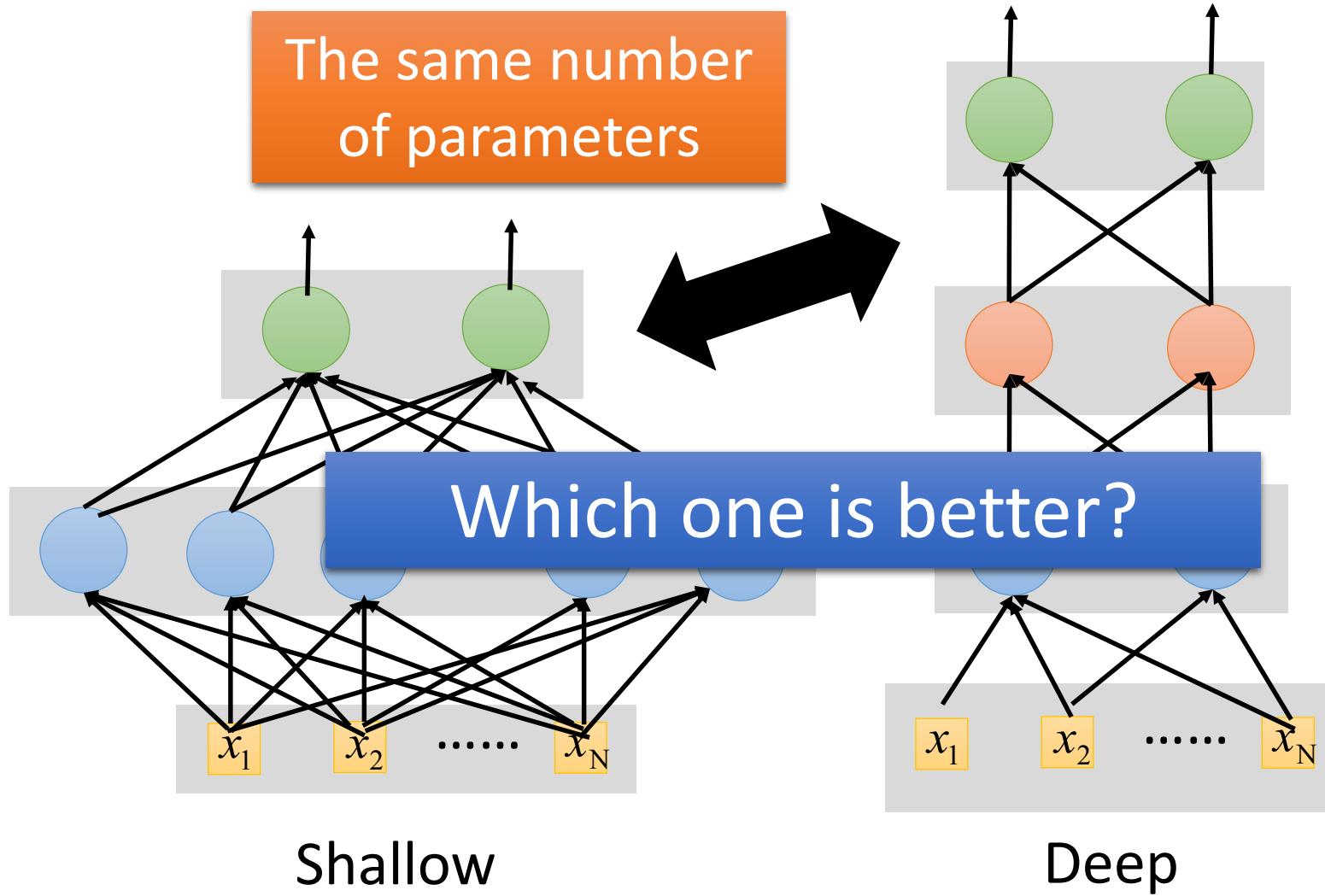
(given **enough** hidden  
neurons)



Reference for the reason:  
<http://neuralnetworksanddeeplearning.com/chap4.html>

Why “Deep” neural network not “Fat” neural network?

# Fat + Short v.s. Thin + Tall



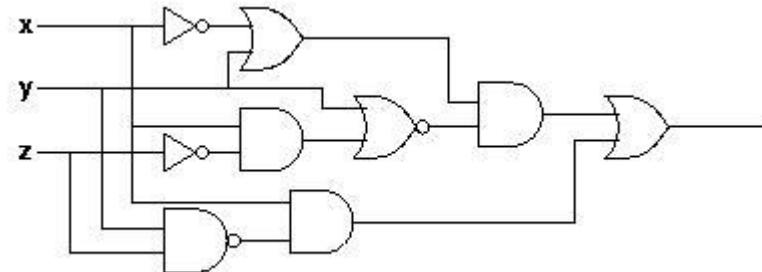
# Fat + Short v.s. Thin + Tall

Layer X Size	Word Error Rate (%)	Layer X Size	Word Error Rate (%)
1 X 2k	24.2		
2 X 2k	20.4		
3 X 2k	18.4		
4 X 2k	17.8		
5 X 2k	17.2	1 X 3772	22.5
7 X 2k	17.1	1 X 4634	22.6
		1 X 16k	22.1

Why?

Seide, Frank, Gang Li, and Dong Yu. "Conversational Speech Transcription Using Context-Dependent Deep Neural Networks." *Interspeech*. 2011.

# Analogy



## Logic circuits

- Logic circuits consists of **gates**
- **A two layers of logic gates** can represent **any Boolean function.**
- Using multiple layers of logic gates to build some functions are much simpler



less gates needed



- Neural network consists of **neurons**
- **A hidden layer network** can represent **any continuous function.**
- Using multiple layers of neurons to represent some functions are much simpler



less parameters

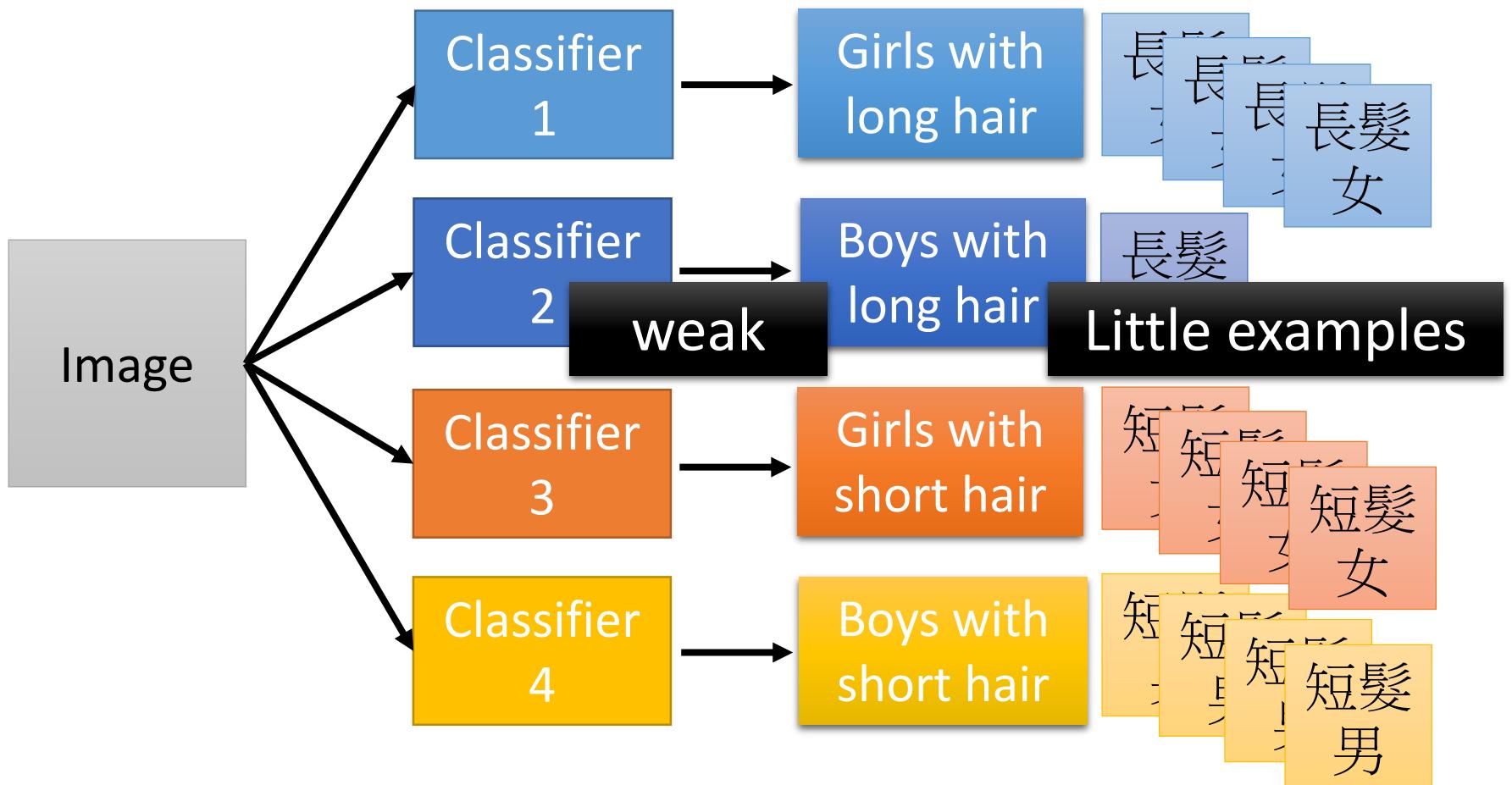


less data?

This page is for EE background.

# Modularization

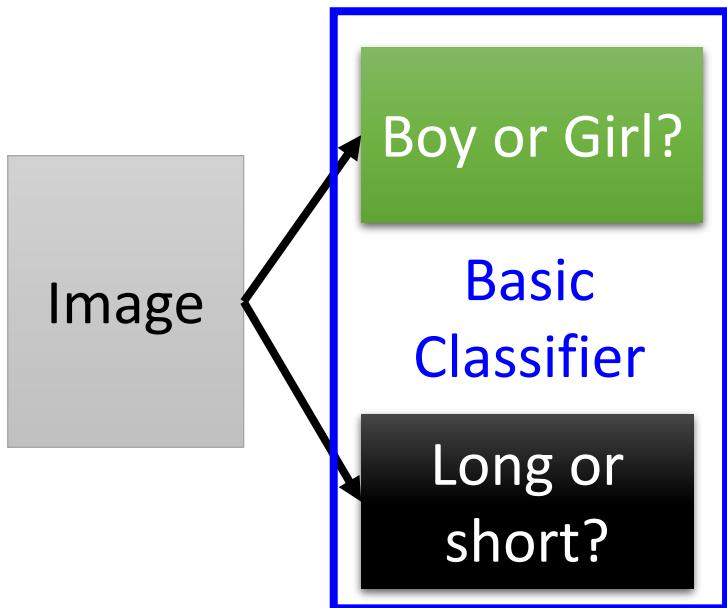
- Deep → Modularization



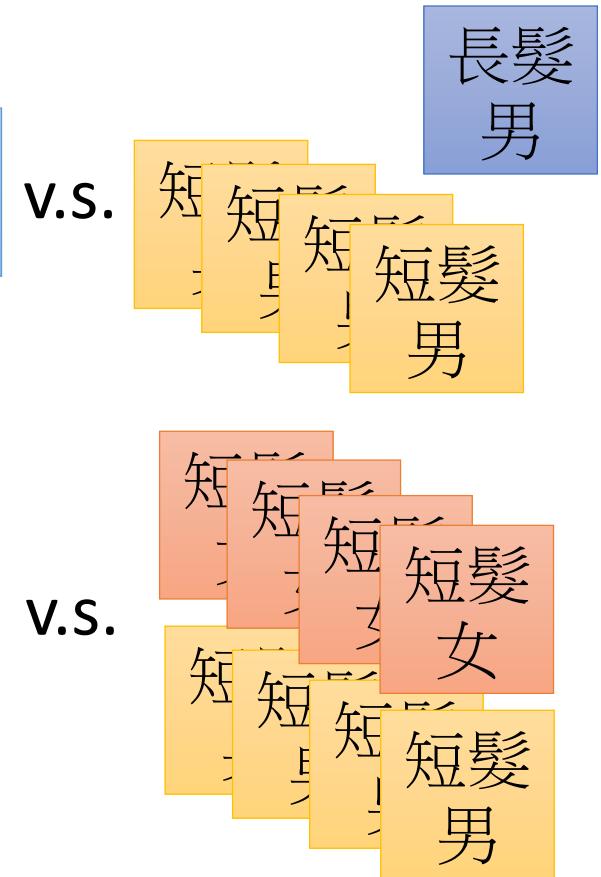
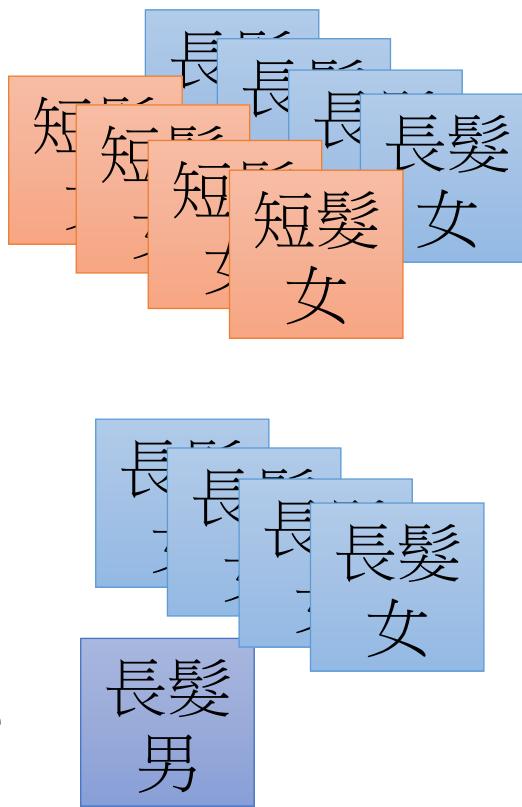
# Modularization

Each basic classifier can have sufficient training examples.

- Deep → Modularization



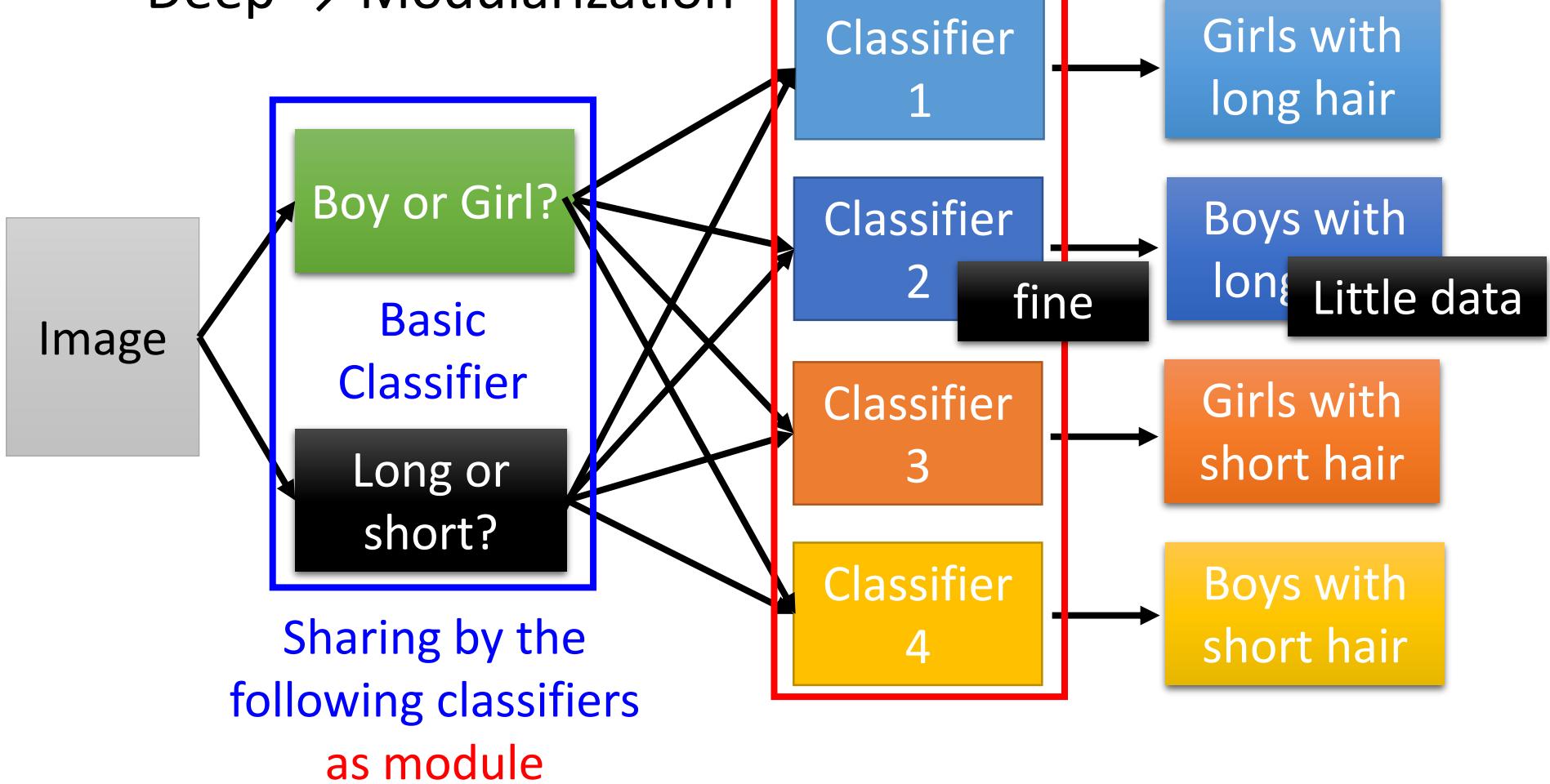
Classifiers for the  
attributes



# Modularization

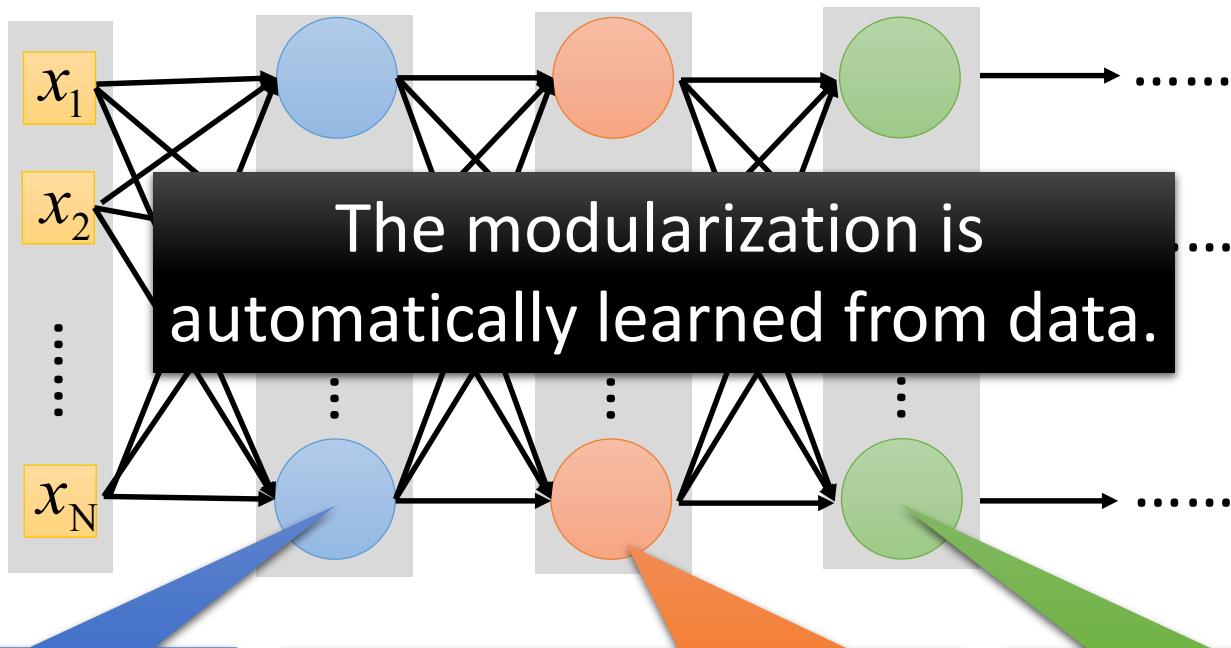
can be trained by little data

- Deep → Modularization



# Modularization

- Deep  $\rightarrow$  Modularization → Less training data?



The most basic classifiers

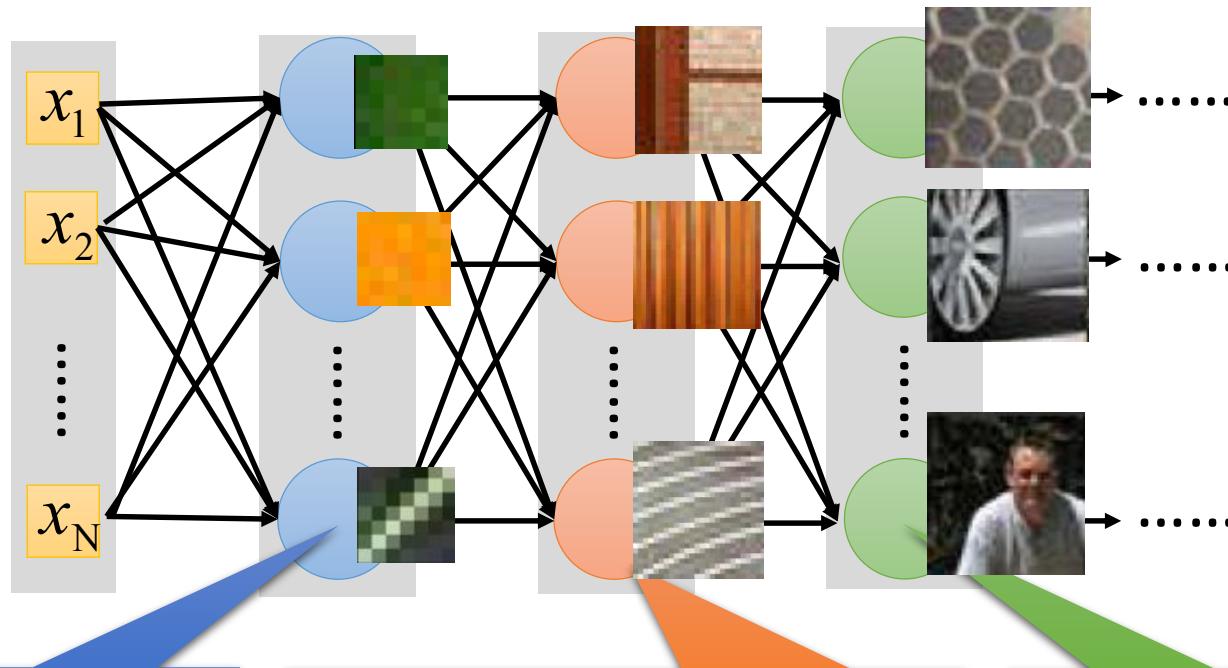
Use 1<sup>st</sup> layer as module to build classifiers

Use 2<sup>nd</sup> layer as module .....

# Modularization

Reference: Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014* (pp. 818-833)

- Deep → Modularization



The most basic  
classifiers

Use 1<sup>st</sup> layer as module  
to build classifiers

Use 2<sup>nd</sup> layer as  
module .....

# Outline of Lecture I

Introduction of Deep Learning

Why Deep?

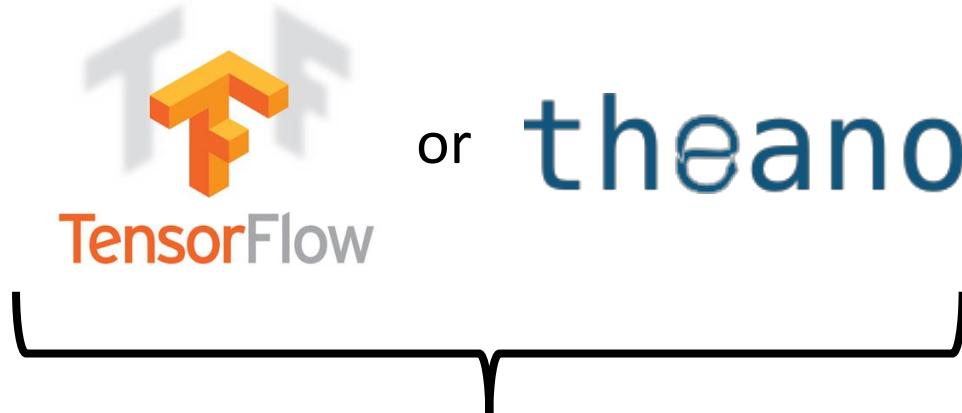
“Hello World” for Deep Learning

# Keras

If you want to learn theano:

[http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS\\_2015\\_2/Lecture/Theano%20DNN.ecm.mp4/index.html](http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/Theano%20DNN.ecm.mp4/index.html)

[http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS\\_2015\\_2/Lecture/RNN%20training%20\(v6\).ecm.mp4/index.html](http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/RNN%20training%20(v6).ecm.mp4/index.html)



Interface of  
TensorFlow or  
Theano



Easy to learn and use  
(still have some flexibility)  
You can modify it if you can write  
TensorFlow or Theano

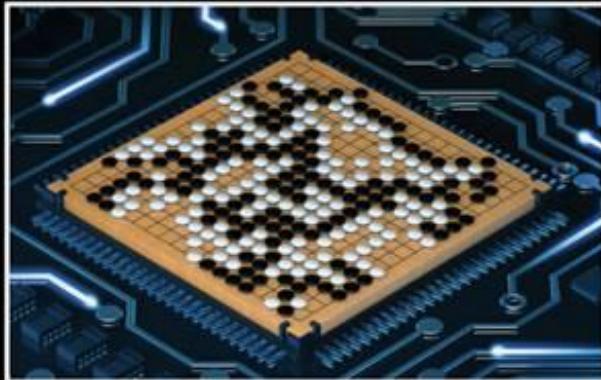
Very flexible  
Need some  
effort to learn

# Keras

- François Chollet is the author of Keras.
  - He currently works for Google as a deep learning engineer and researcher.
- Keras means *horn* in Greek
- Documentation: <http://keras.io/>
- Example:  
<https://github.com/fchollet/keras/tree/master/examples>

# 使用 Keras 心得

## Deep Learning研究生



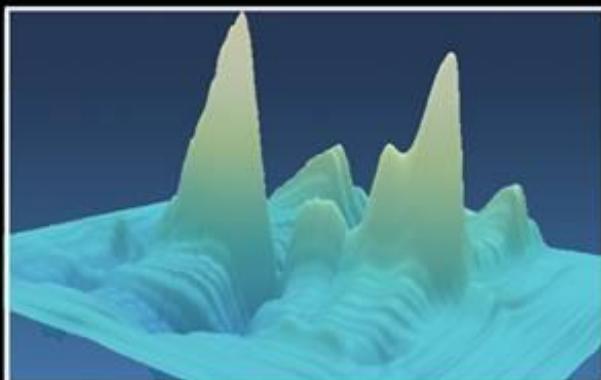
朋友覺得我在



我媽覺得我在



大眾覺得我在



指導教授覺得我在



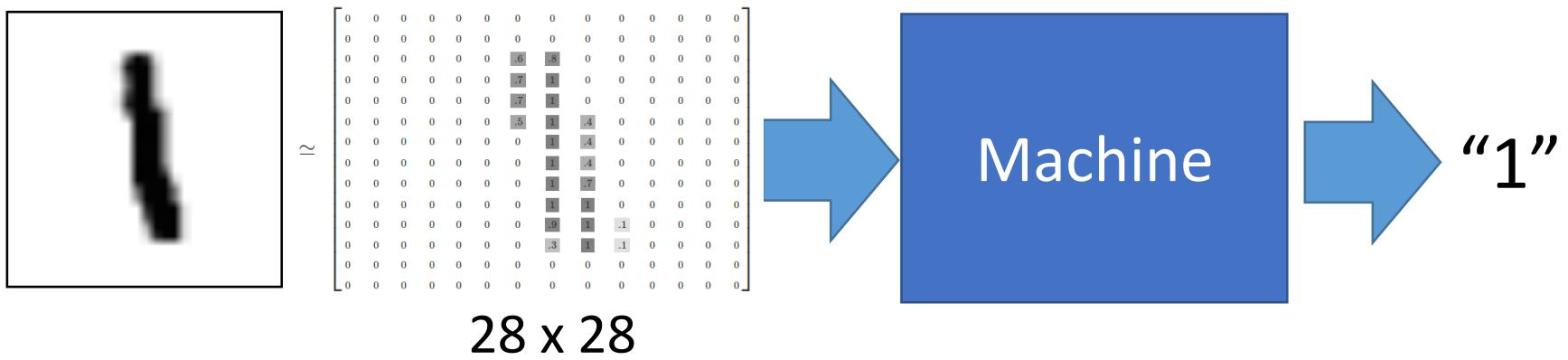
我以為我在



事實上我在

# Example Application

- # • Handwriting Digit Recognition



MNIST Data: <http://yann.lecun.com/exdb/mnist/>  
“Hello world” for deep learning

Keras provides data sets loading function: <http://keras.io/datasets/>

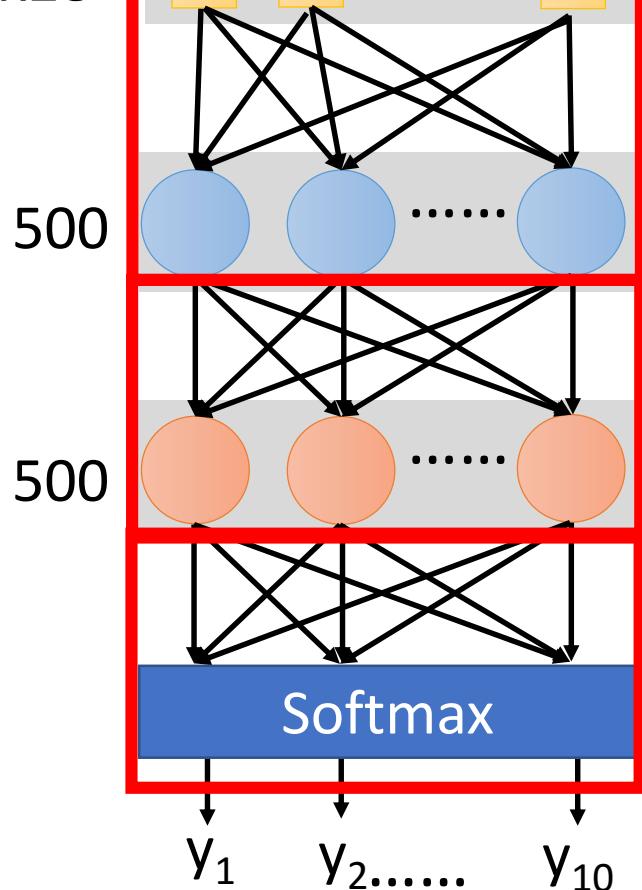
# Keras

Step 1:  
define a set  
of function

Step 2:  
goodness of  
function

Step 3: pick  
the best  
function

28x28



```
model = Sequential()
```

```
model.add( Dense( input_dim=28*28,  
                  output_dim=500 ) )  
model.add( Activation('sigmoid') )
```

```
model.add( Dense( output_dim=500 ) )  
model.add( Activation('sigmoid') )
```

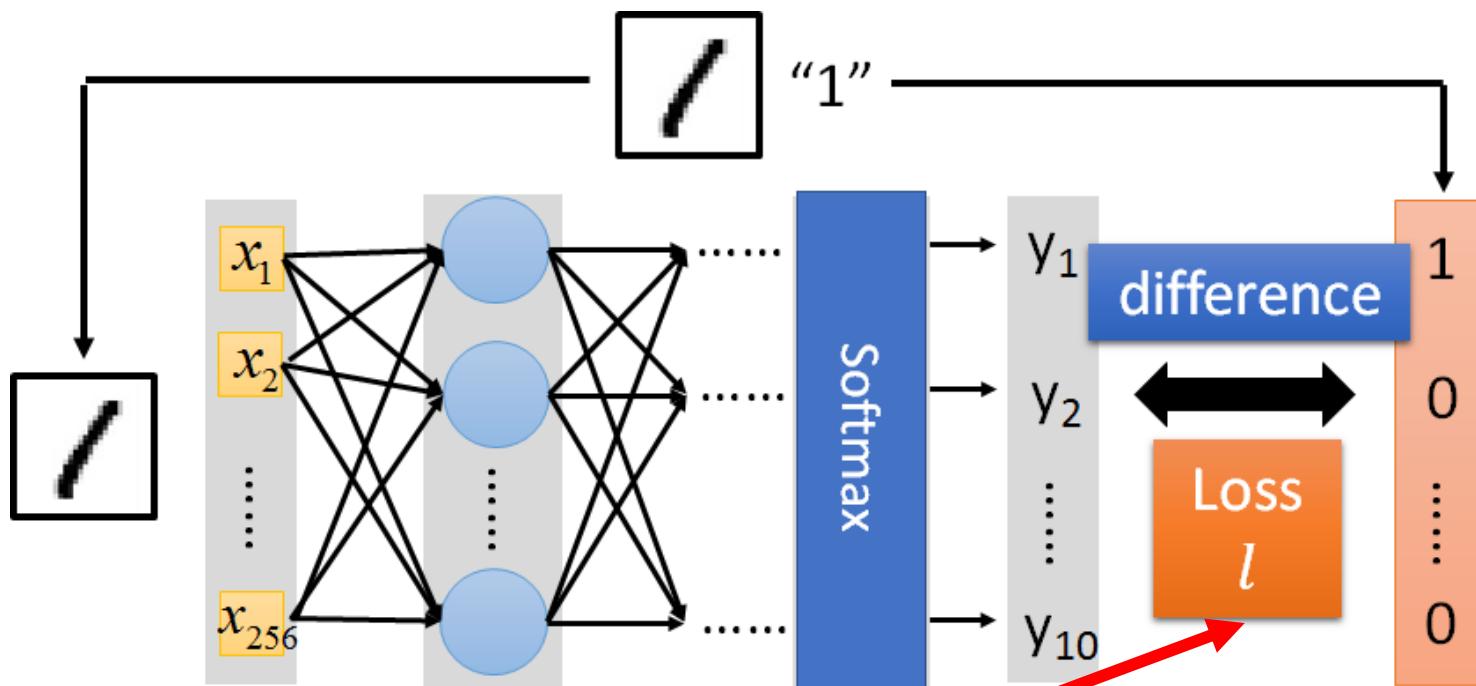
```
model.add( Dense( output_dim=10 ) )  
model.add( Activation('softmax') )
```

# Keras

Step 1:  
define a set  
of function

Step 2:  
goodness of  
function

Step 3: pick  
the best  
function



```
model.compile(loss='mse',  
              optimizer=SGD(lr=0.1),  
              metrics=['accuracy'])
```

# Keras

Step 1:  
define a set  
of function

Step 2:  
goodness of  
function

Step 3: pick  
the best  
function

## Step 3.1: Configuration

```
model.compile(loss='mse',  
               optimizer=SGD(lr=0.1),  
               metrics=['accuracy'])
```

$$w \leftarrow w - \eta \partial L / \partial w$$

0.1

## Step 3.2: Find the optimal network parameters

```
model.fit(x_train, y_train, batch_size=100, nb_epoch=20)
```

Training data  
(Images)

Labels  
(digits)

Next lecture

# Keras

Step 1:  
define a set  
of function

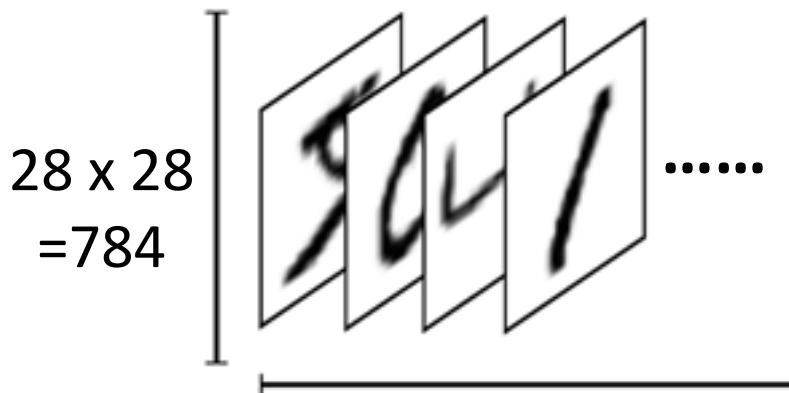
Step 2:  
goodness of  
function

Step 3: pick  
the best  
function

## Step 3.2: Find the optimal network parameters

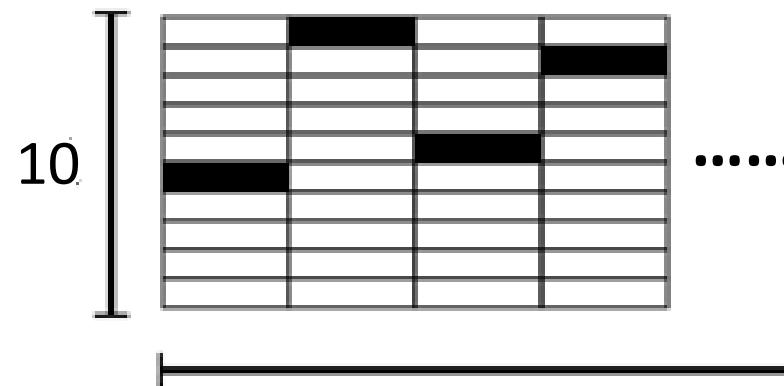
```
model.fit(x_train, y_train, batch_size=100, nb_epoch=20)
```

numpy array



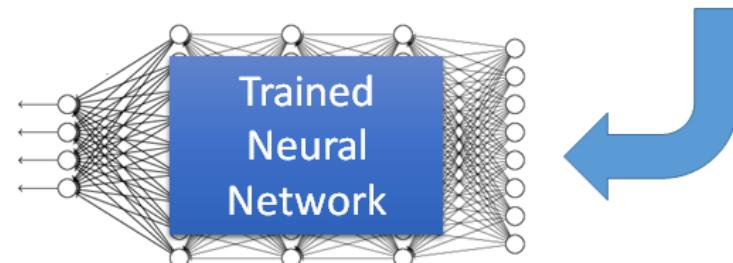
Number of training examples

numpy array



Number of training examples

# Keras



Save and load models

<http://keras.io/getting-started/faq/#how-can-i-save-a-keras-model>

How to use the neural network (testing):

```
score = model.evaluate(x_test, y_test)
case 1: print('Total loss on Testing Set:', score[0])
          print('Accuracy of Testing Set:', score[1])
```

```
case 2: result = model.predict(x_test)
```

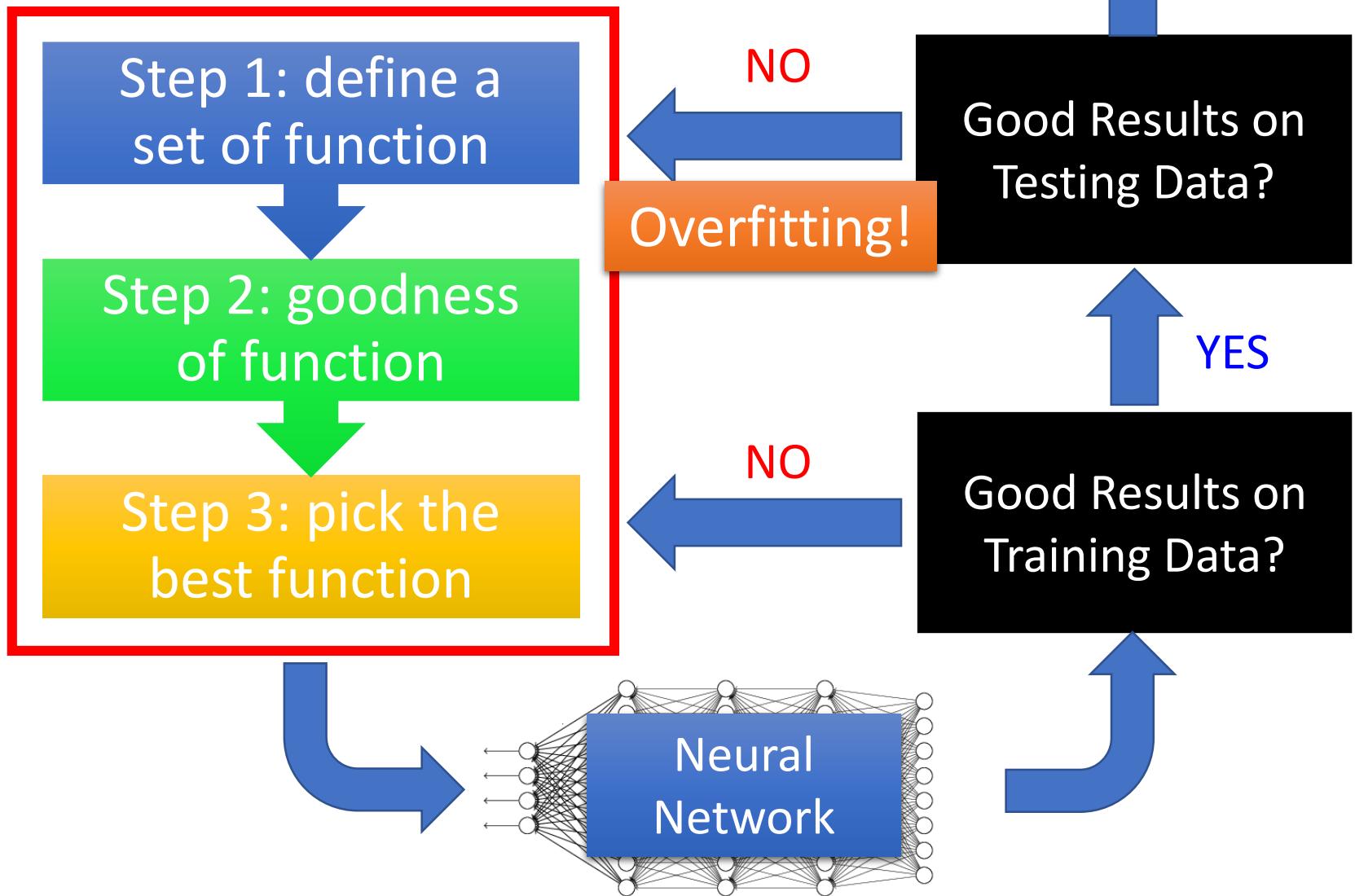
# Keras

- Using GPU to speed training
  - Way 1
    - THEANO\_FLAGS=device=gpu0 python YourCode.py
  - Way 2 (in your code)
    - import os
    - os.environ["THEANO\_FLAGS"] = "device=gpu0"

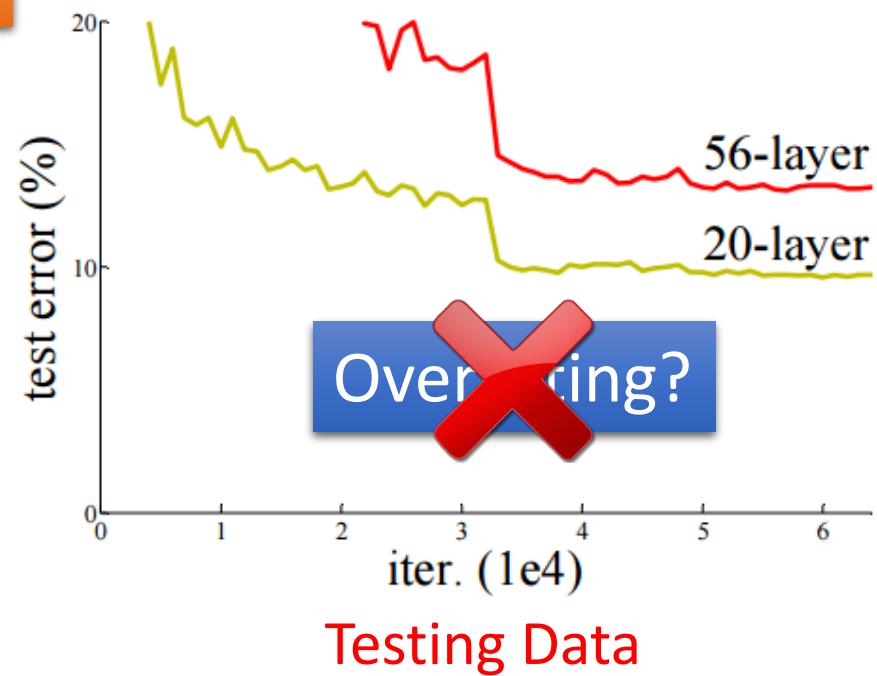
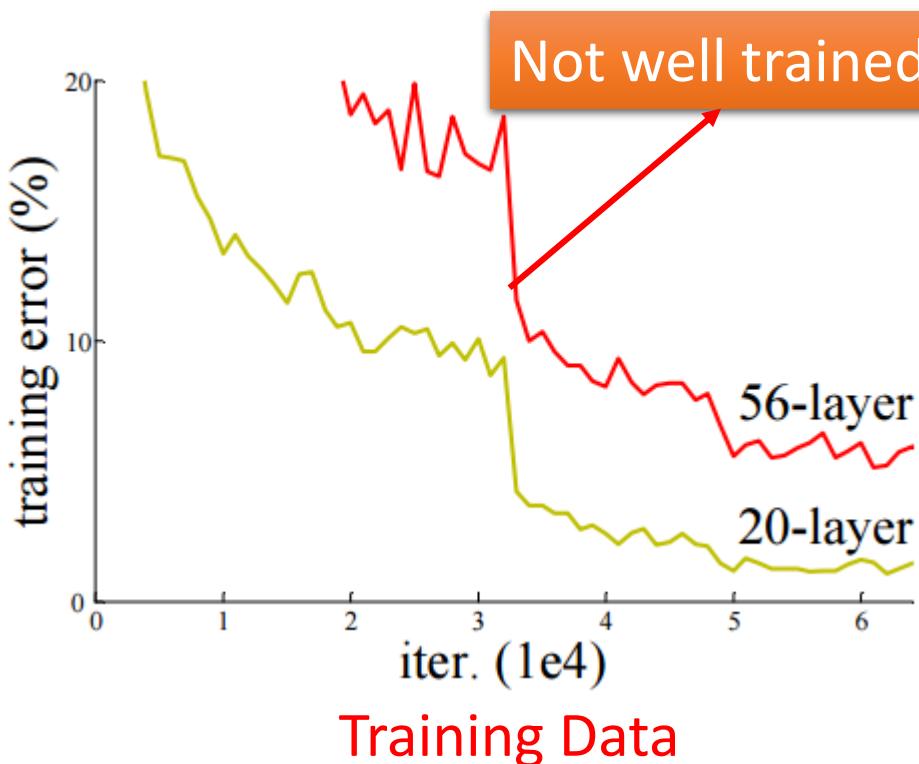
# Live Demo

# Lecture II: Tips for Training DNN

# Recipe of Deep Learning



# Do not always blame Overfitting

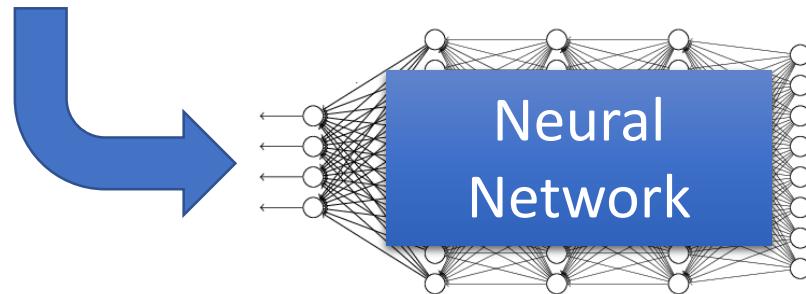


**Deep Residual Learning for Image Recognition**  
<http://arxiv.org/abs/1512.03385>

# Recipe of Deep Learning

Different approaches for different problems.

e.g. dropout for good results on testing data



Good Results on Testing Data?

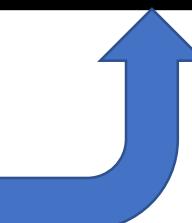
Good Results on Training Data?



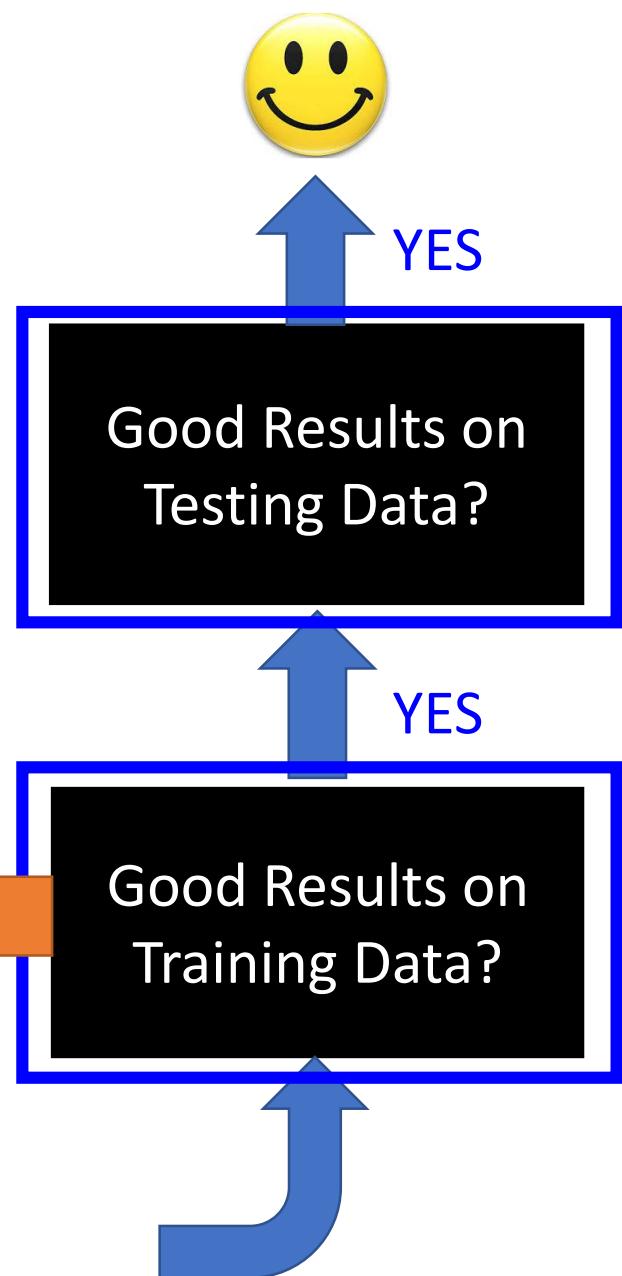
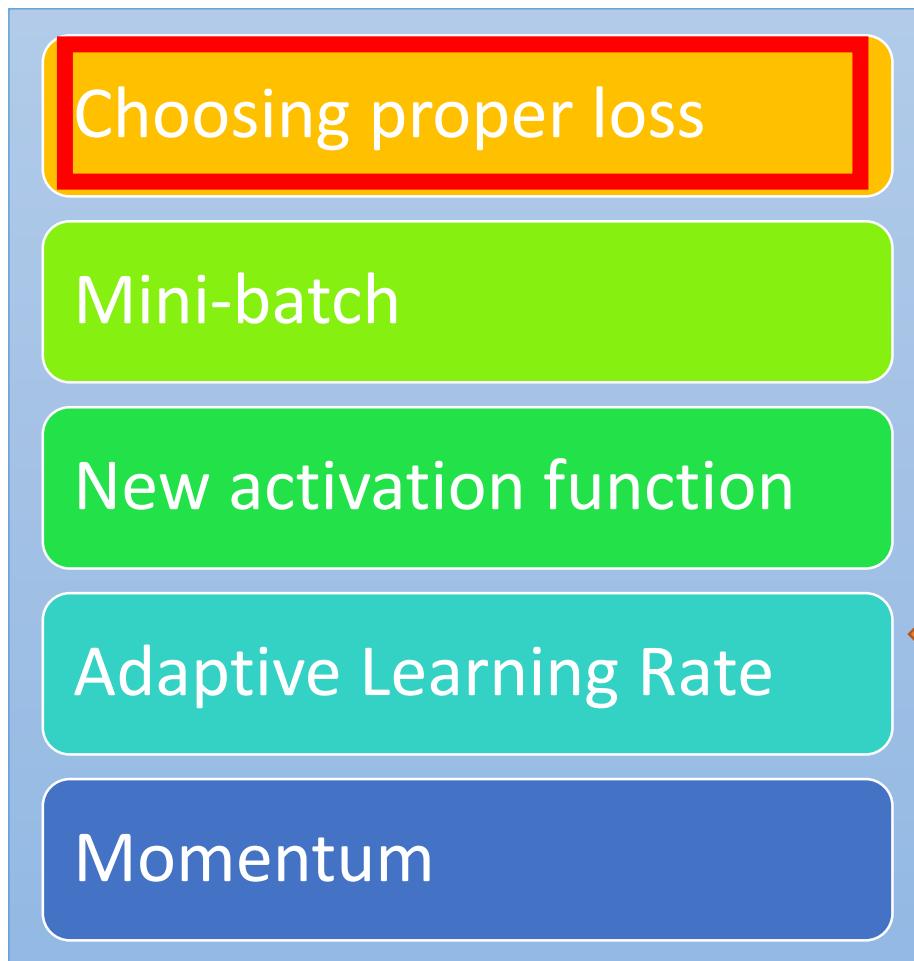
YES



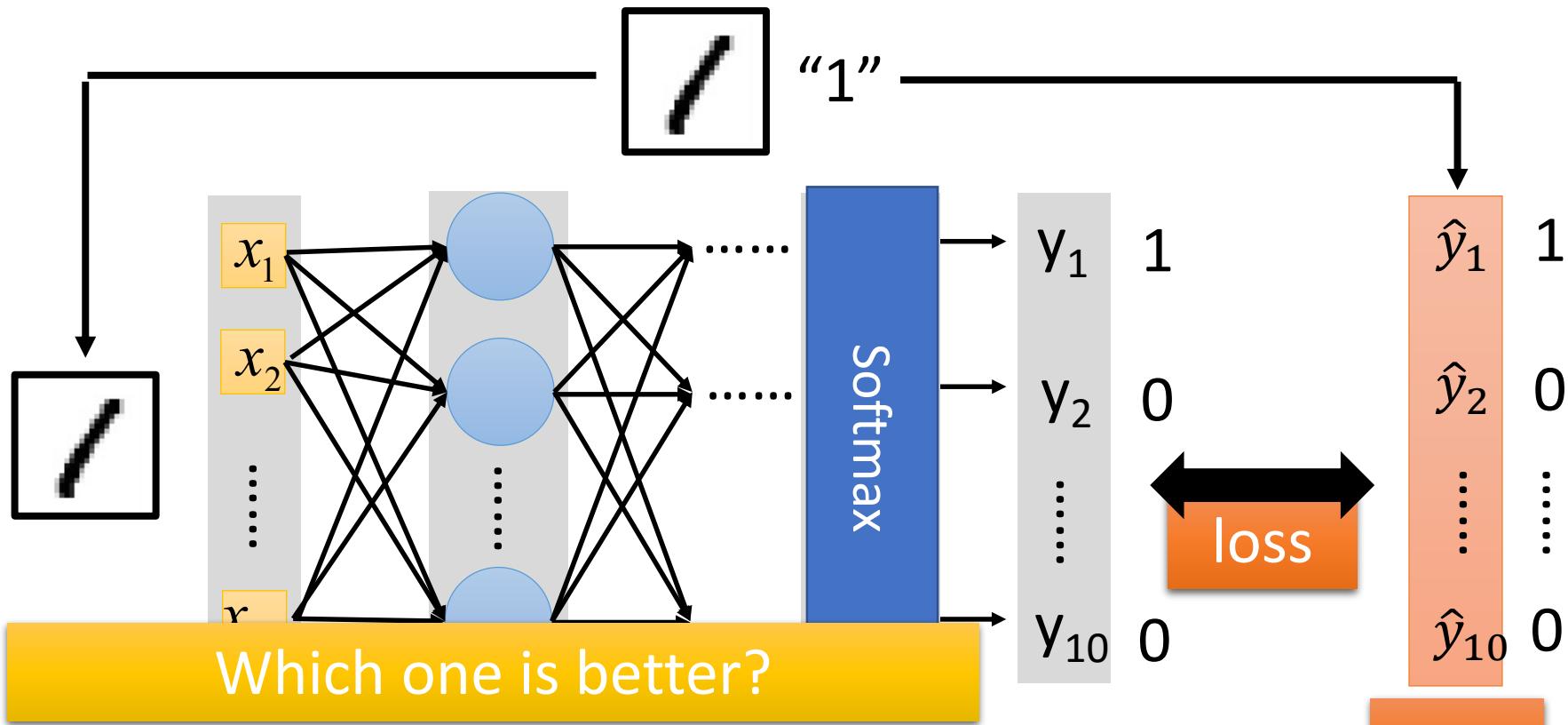
YES



# Recipe of Deep Learning



# Choosing Proper Loss



Square  
Error

$$\sum_{i=1}^{10} (y_i - \hat{y}_i)^2 = 0$$

Cross  
Entropy

$$-\sum_{i=1}^{10} \hat{y}_i \ln y_i = 0$$

target

# Let's try it

## Square Error

```
model.compile(loss='mse',  
              optimizer=SGD(lr=0.1),  
              metrics=['accuracy'])
```

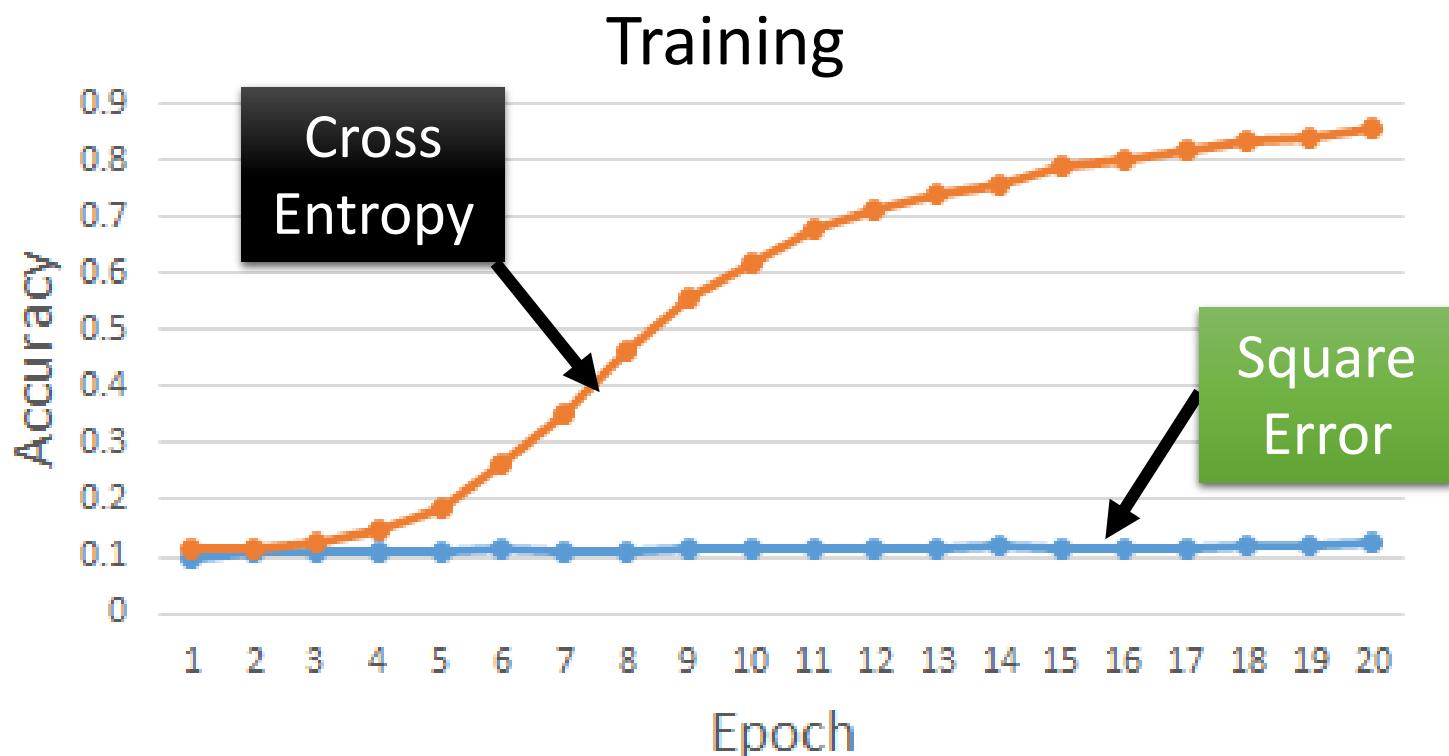
## Cross Entropy

```
model.compile(loss='categorical_crossentropy',  
              optimizer=SGD(lr=0.1),  
              metrics=['accuracy'])
```

Let's try it

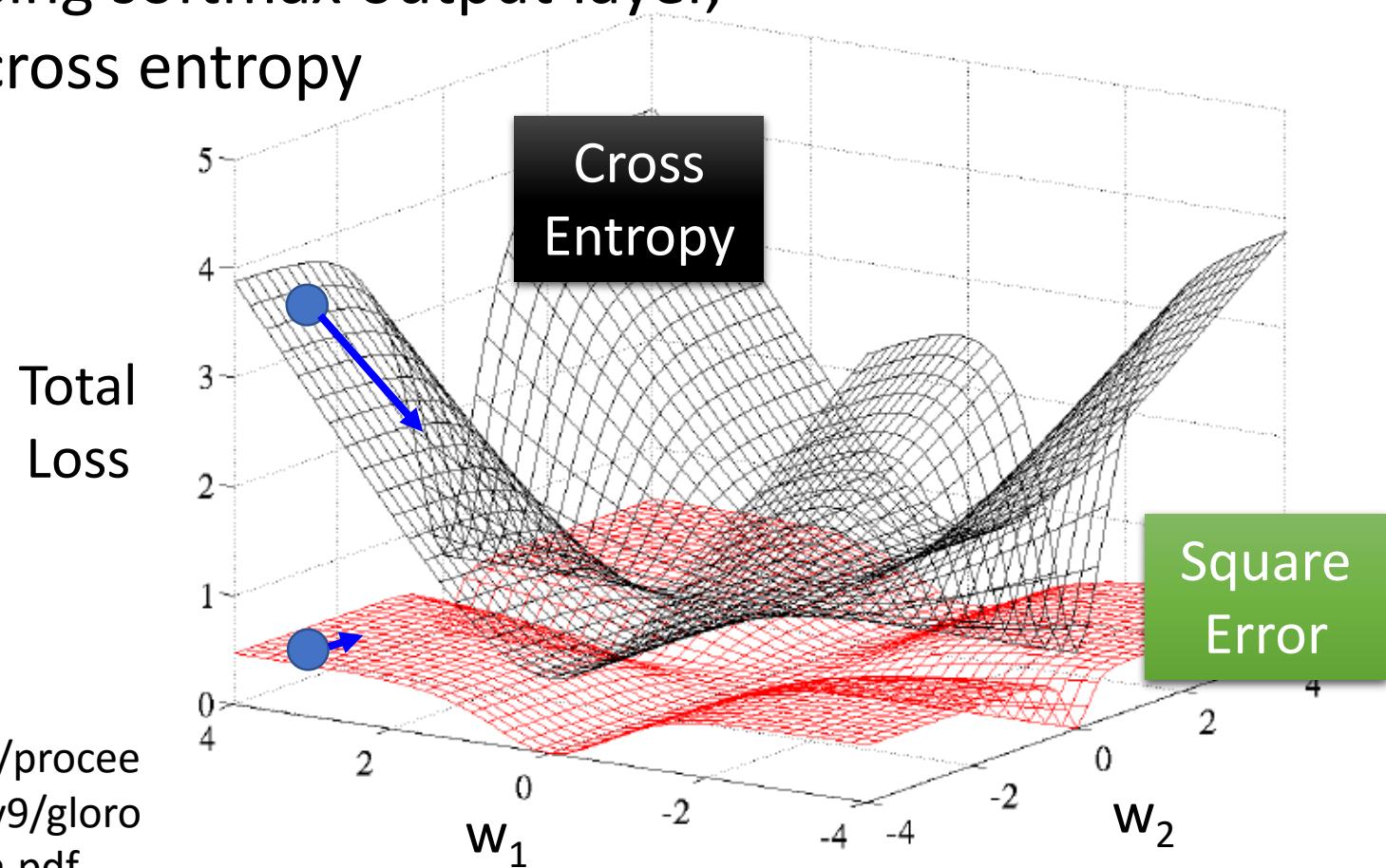
Testing:

	Accuracy
Square Error	0.11
Cross Entropy	0.84

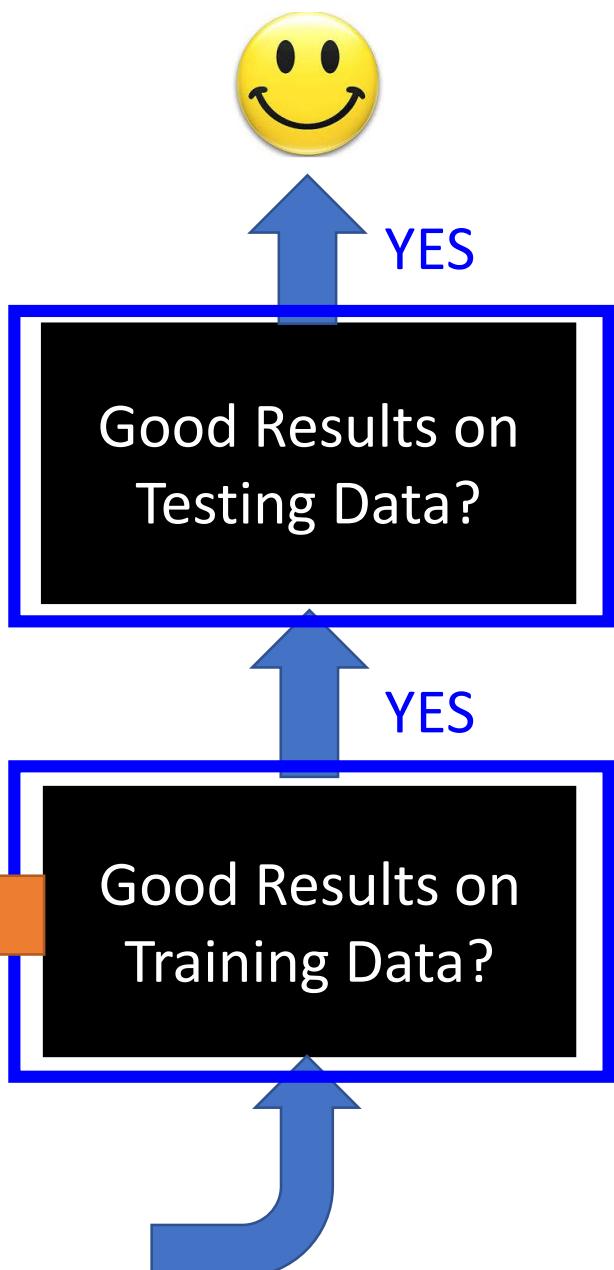


# Choosing Proper Loss

When using softmax output layer,  
choose cross entropy



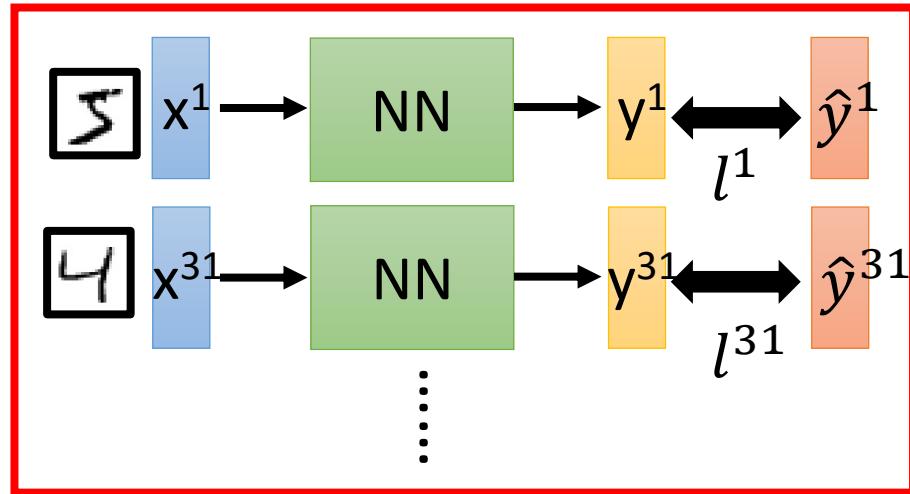
# Recipe of Deep Learning



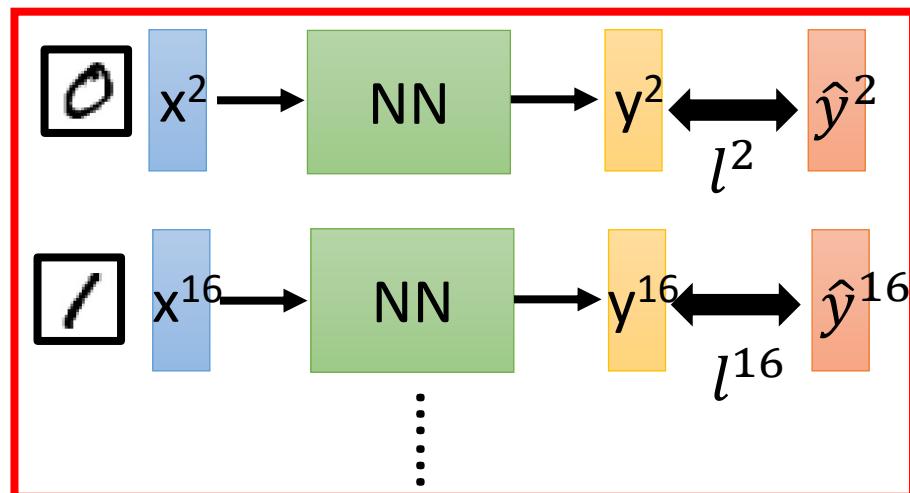
We do not really minimize total loss!

# Mini-batch

Mini-batch



Mini-batch



- Randomly initialize network parameters

- Pick the 1<sup>st</sup> batch  
 $L' = l^1 + l^{31} + \dots$   
Update parameters once
- Pick the 2<sup>nd</sup> batch  
 $L'' = l^2 + l^{16} + \dots$   
Update parameters once  
⋮
- Until all mini-batches have been picked

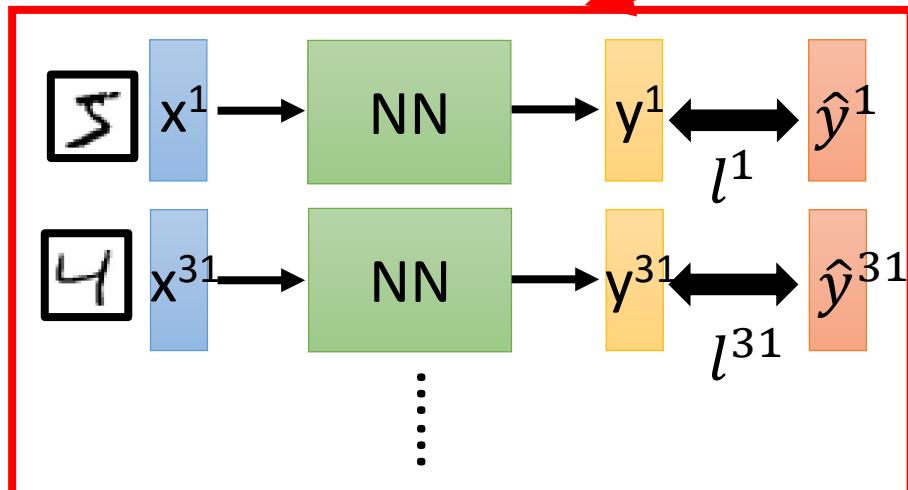
one epoch

Repeat the above process

# Mini-batch

```
model.fit(x_train, y_train, batch_size=100, nb_epoch=20)
```

Mini-batch



100 examples in a mini-batch

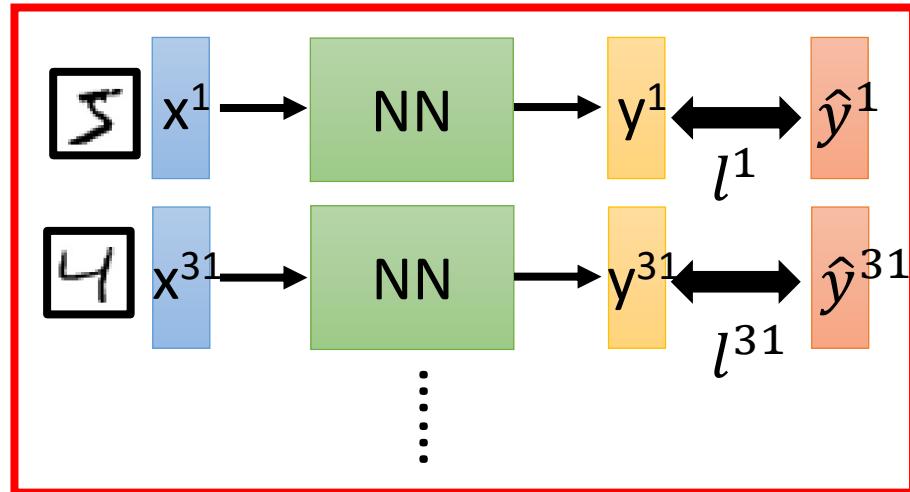
Repeat 20 times

- Pick the 1<sup>st</sup> batch  
 $L' = l^1 + l^{31} + \dots$   
Update parameters once
  - Pick the 2<sup>nd</sup> batch  
 $L'' = l^2 + l^{16} + \dots$   
Update parameters once  
⋮
  - Until all mini-batches have been picked
- one epoch

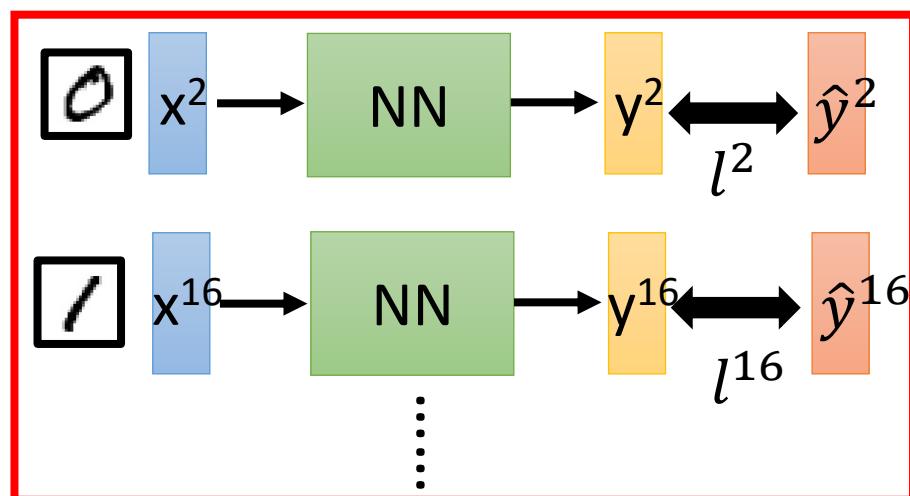
We do not really minimize total loss!

# Mini-batch

Mini-batch



Mini-batch

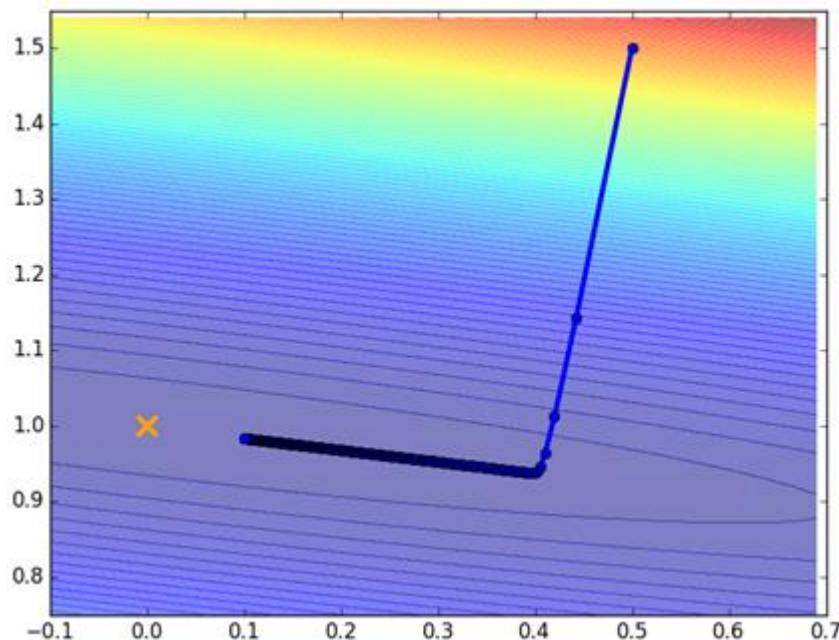


- Randomly initialize network parameters
- Pick the 1<sup>st</sup> batch  
 $L' = l^1 + l^{31} + \dots$   
Update parameters once
- Pick the 2<sup>nd</sup> batch  
 $L'' = l^2 + l^{16} + \dots$   
Update parameters once  
⋮

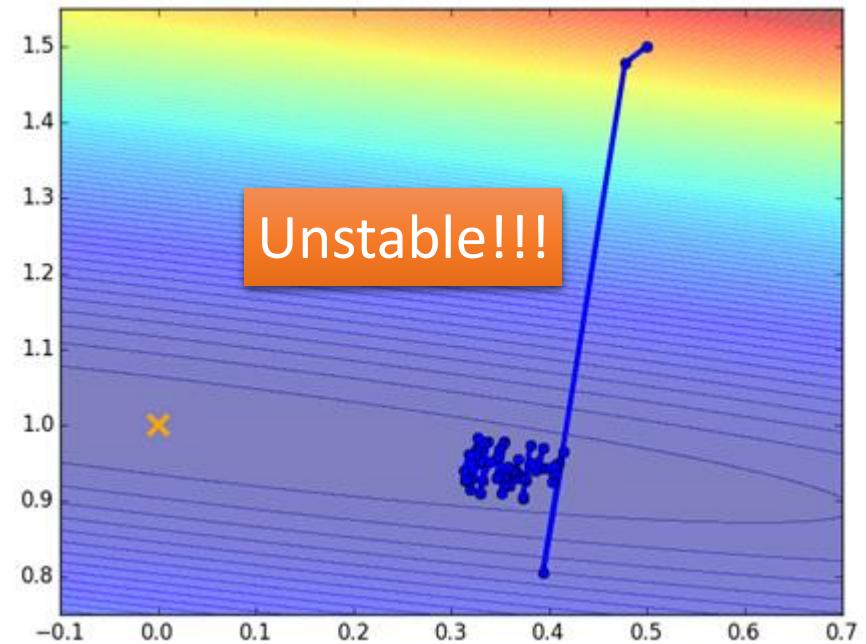
L is different each time  
when we update  
parameters!

# Mini-batch

*Original Gradient Descent*



*With Mini-batch*



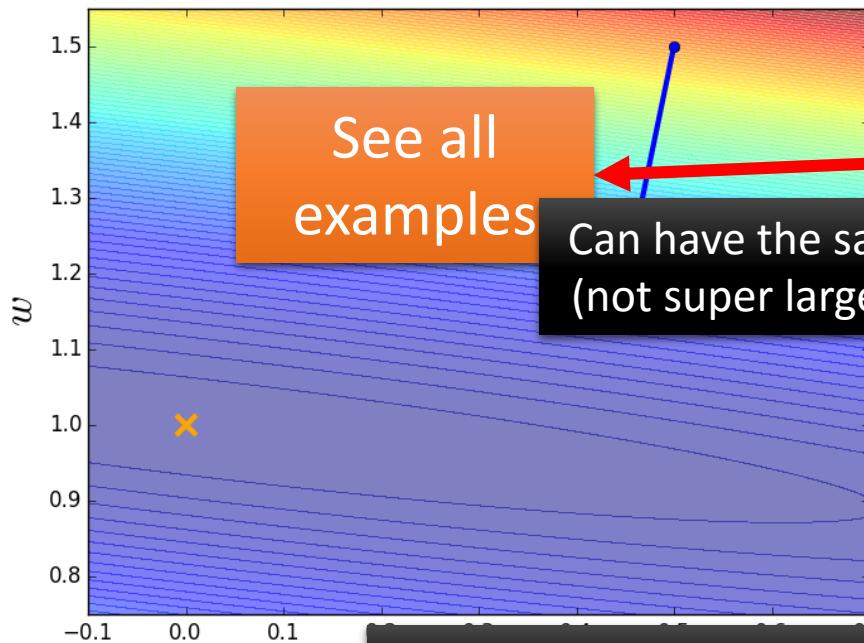
The colors represent the total loss.

# Mini-batch is Faster

Not always true with parallel computing.

## Original Gradient Descent

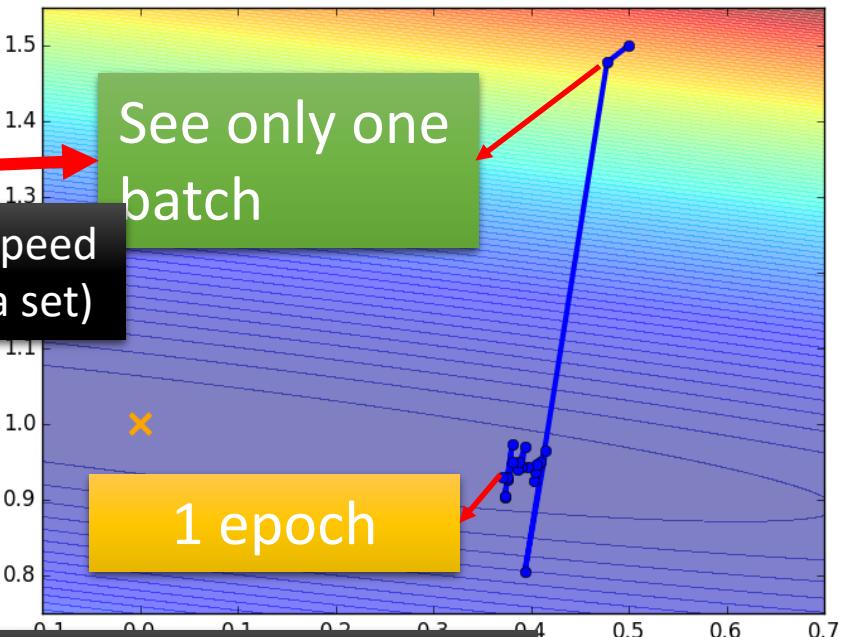
Update after seeing all examples



Mini-batch has better performance!

## With Mini-batch

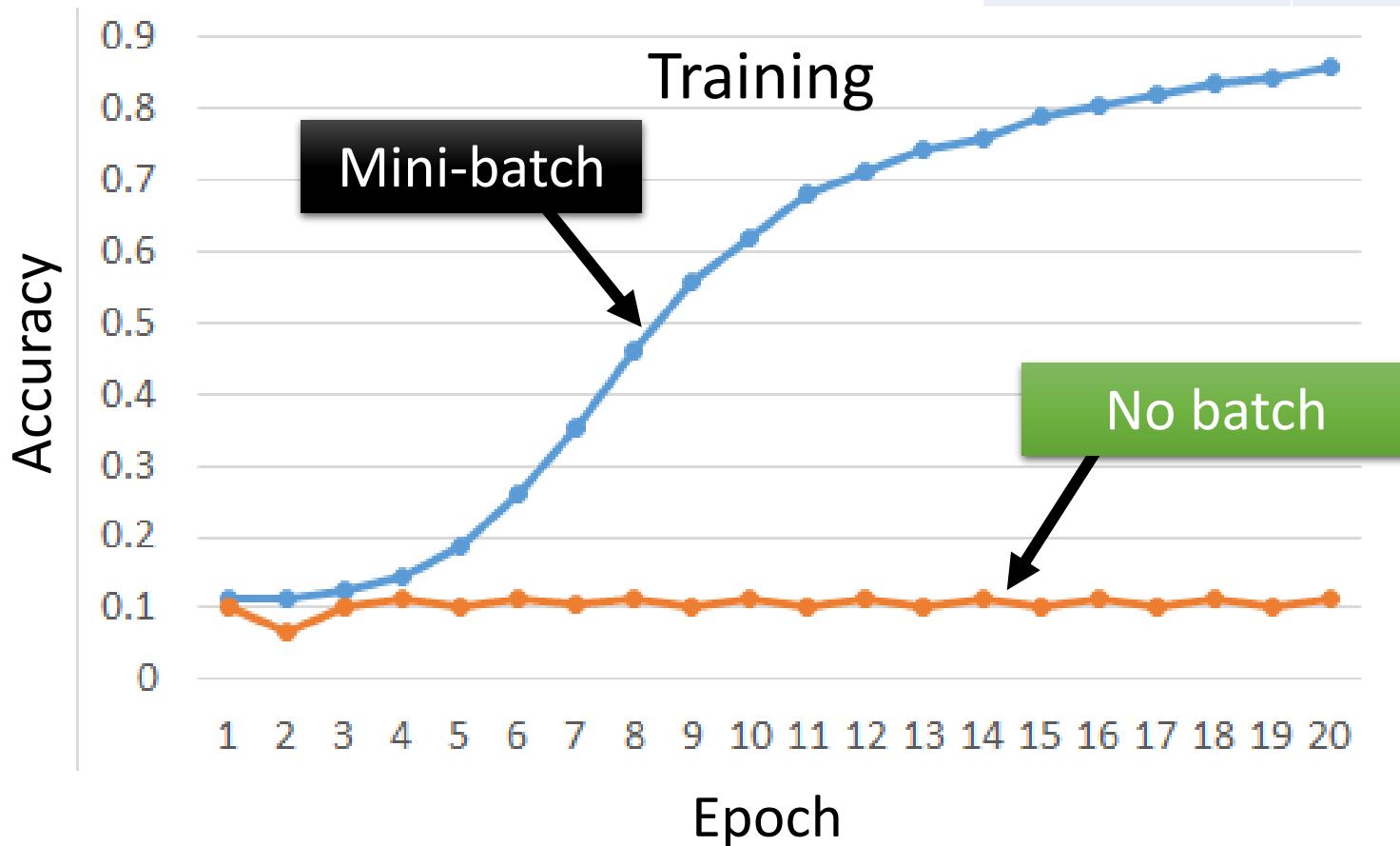
If there are 20 batches, update 20 times in one epoch.



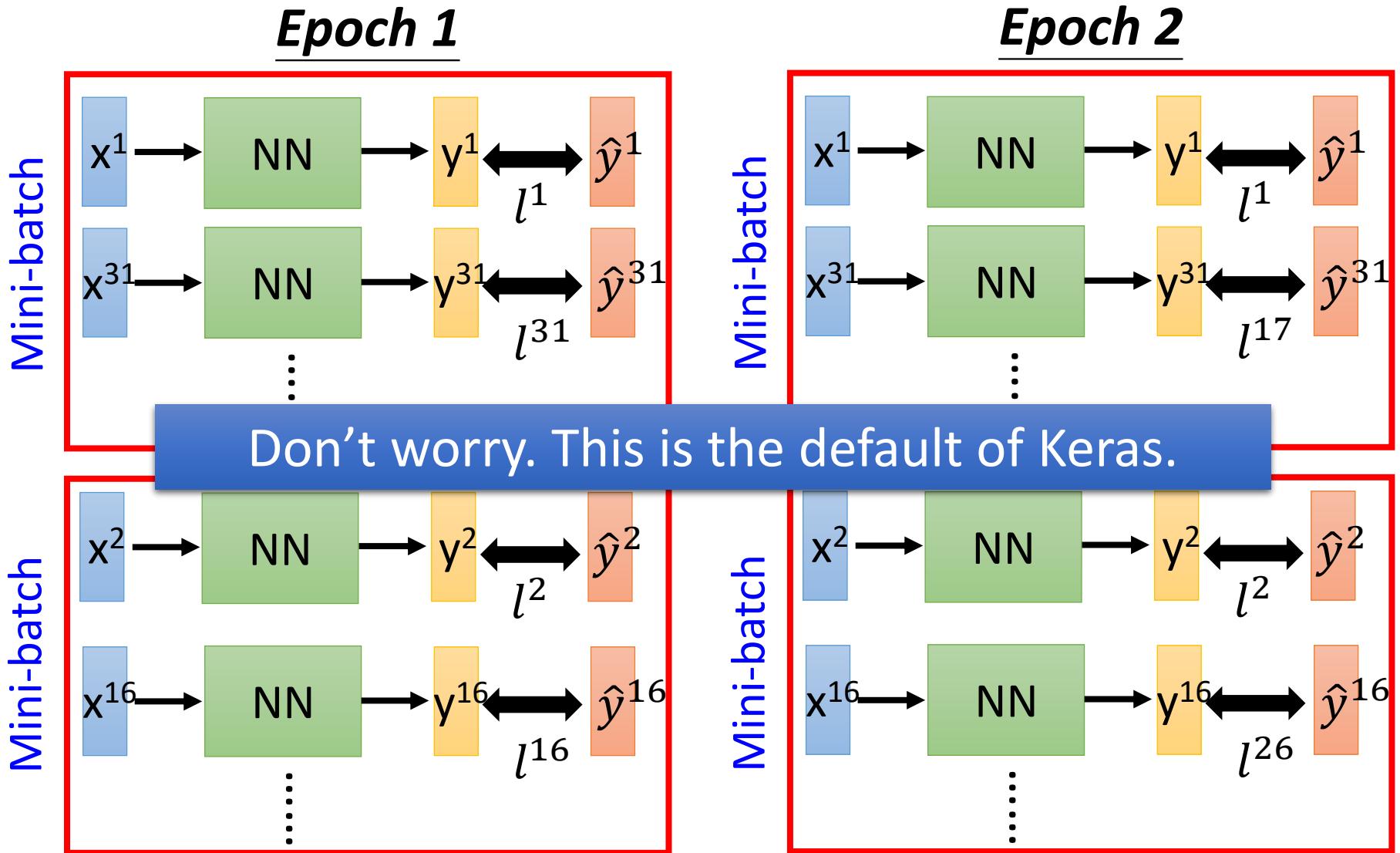
Testing:

# Mini-batch is Better!

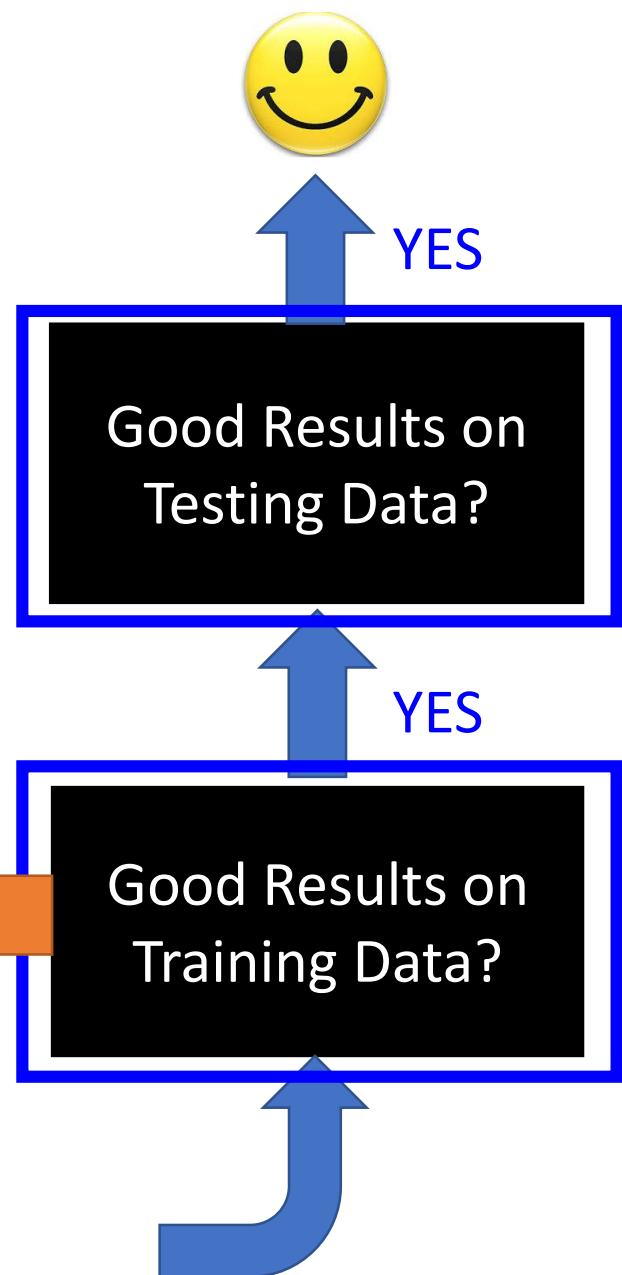
	Accuracy
Mini-batch	0.84
No batch	0.12



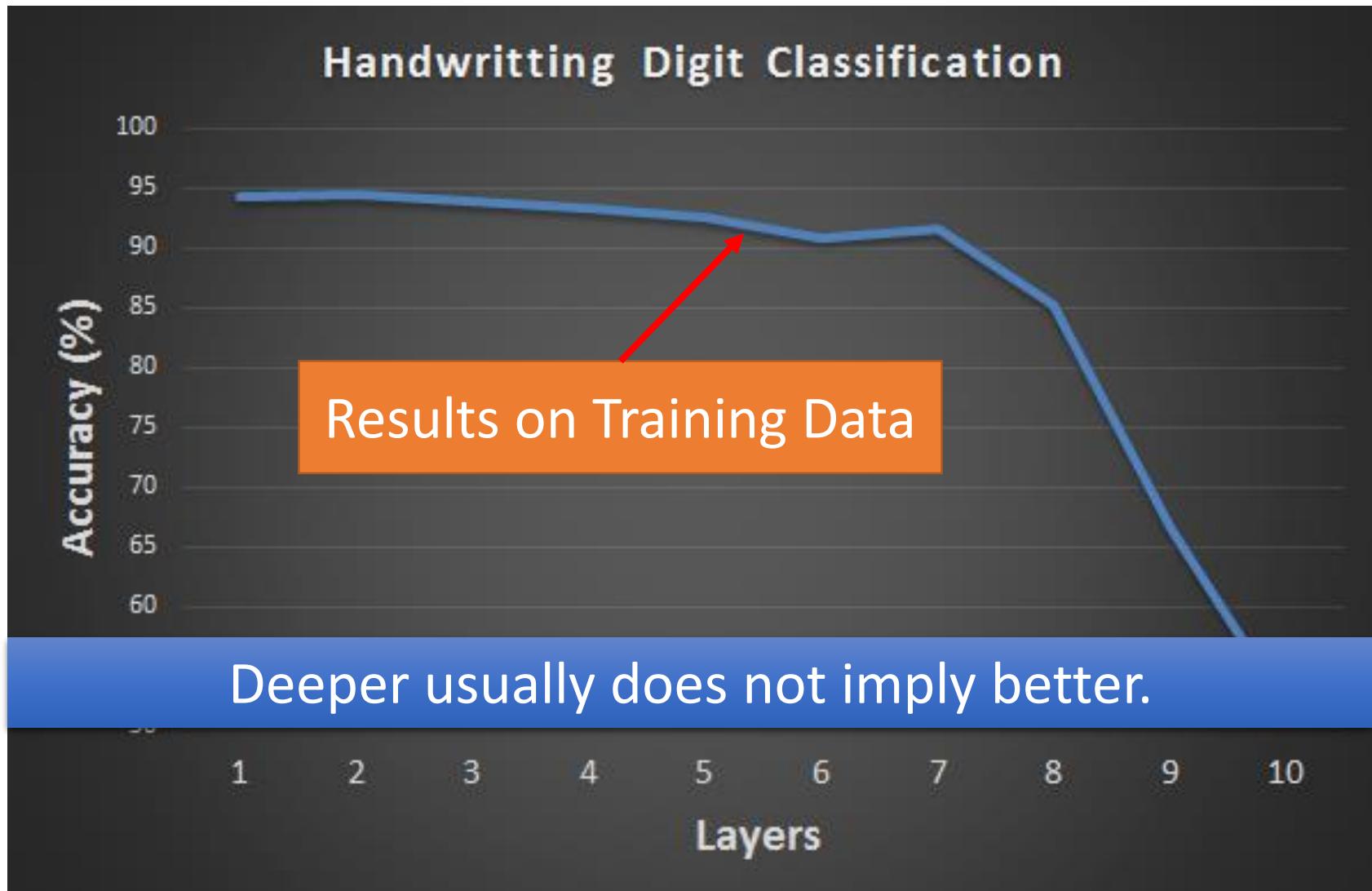
# *Shuffle the training examples for each epoch*



# Recipe of Deep Learning



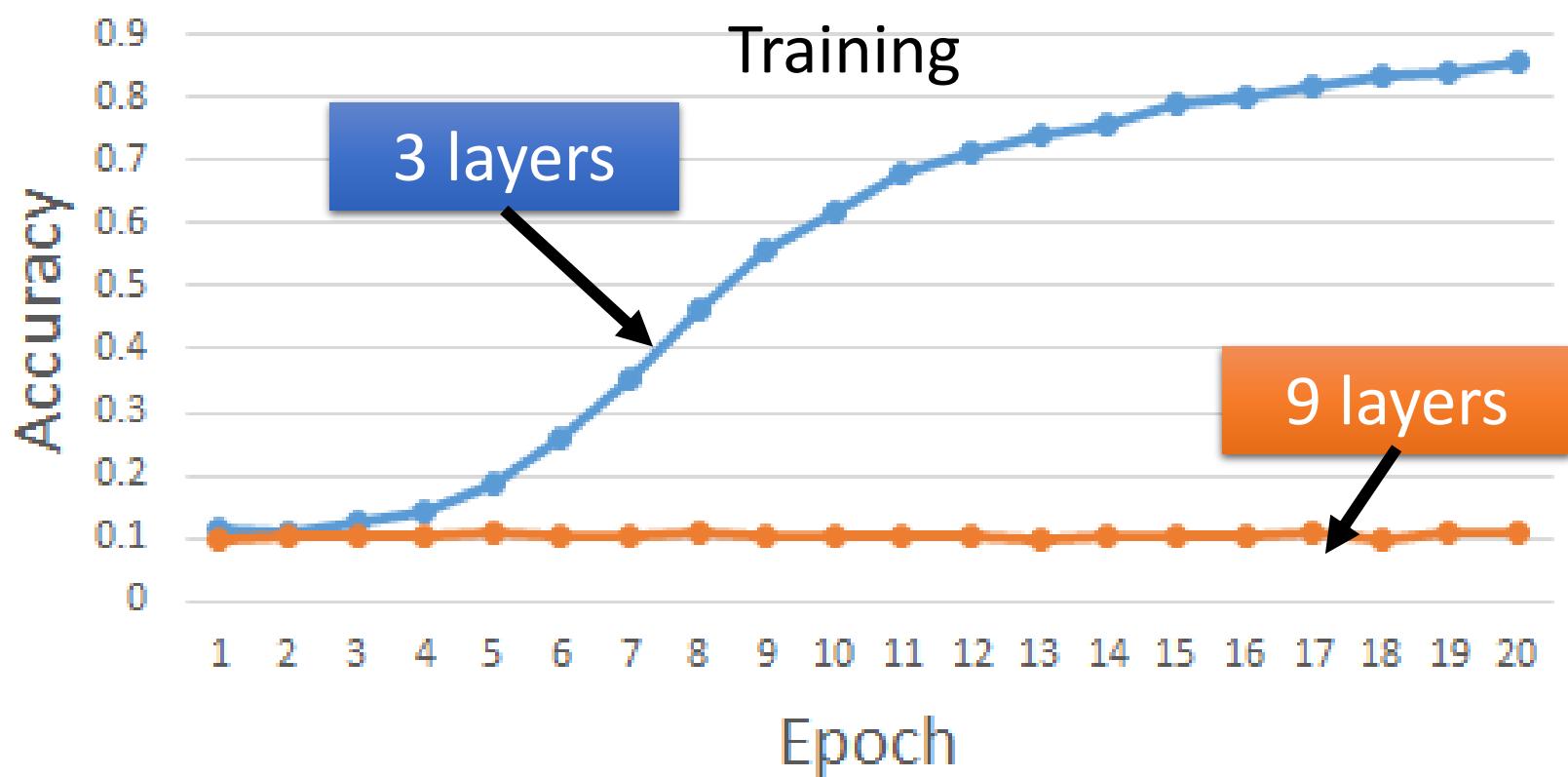
# Hard to get the power of Deep ...



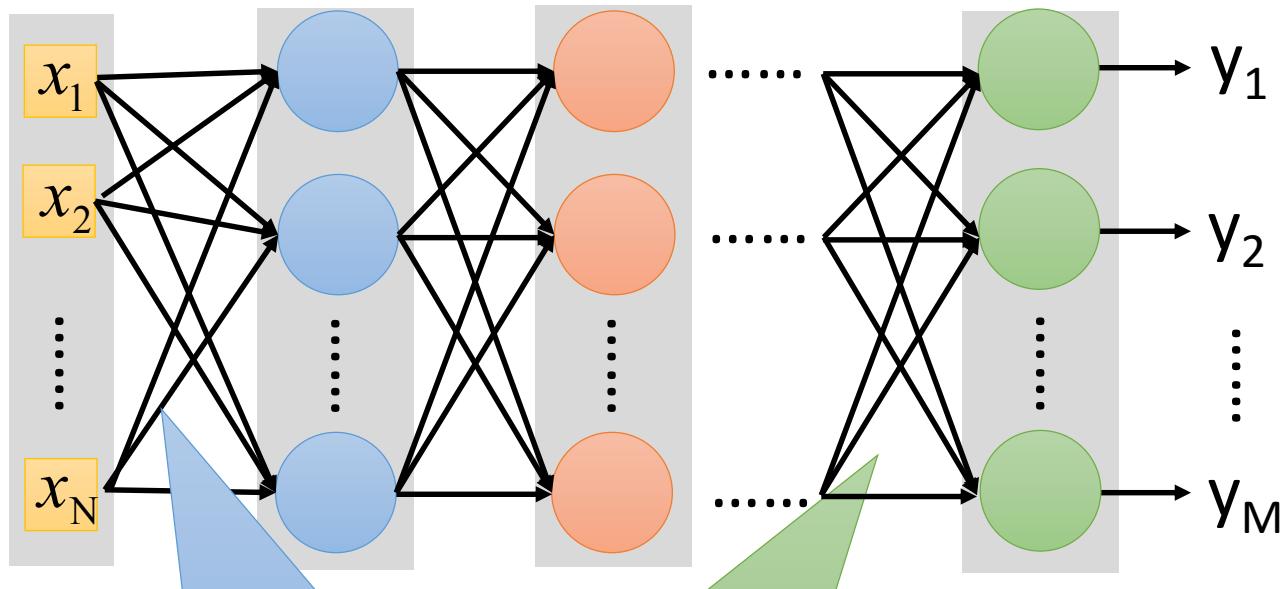
# Let's try it

Testing:

	Accuracy
3 layers	0.84
9 layers	0.11



# Vanishing Gradient Problem



Smaller gradients

Learn very slow

Almost random

Larger gradients

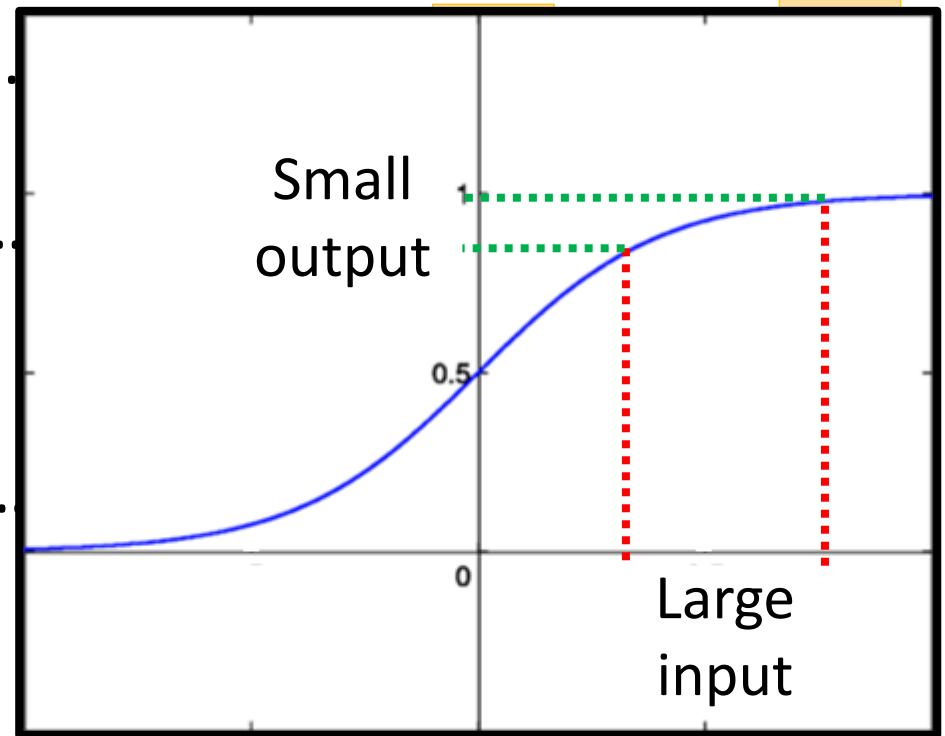
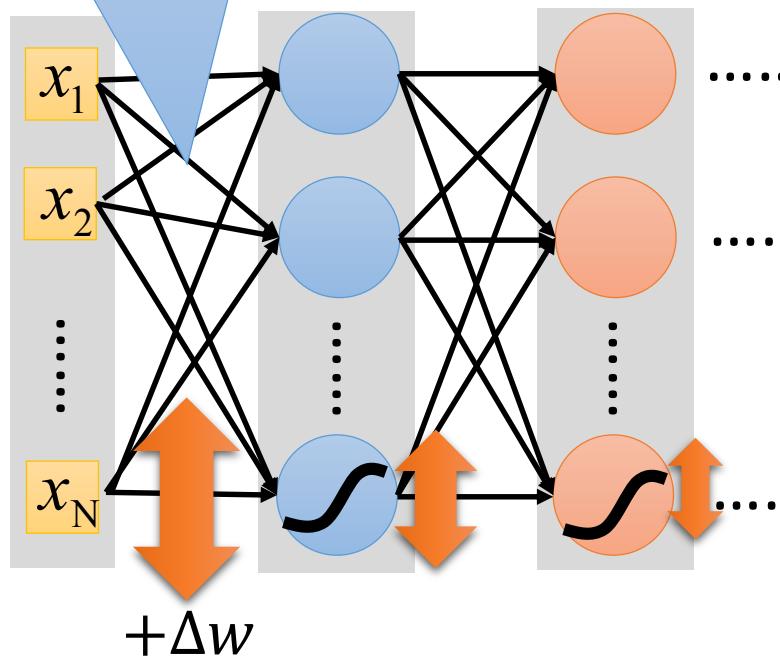
Learn very fast

Already converge

based on random!?

# Vanishing Gradient Problem

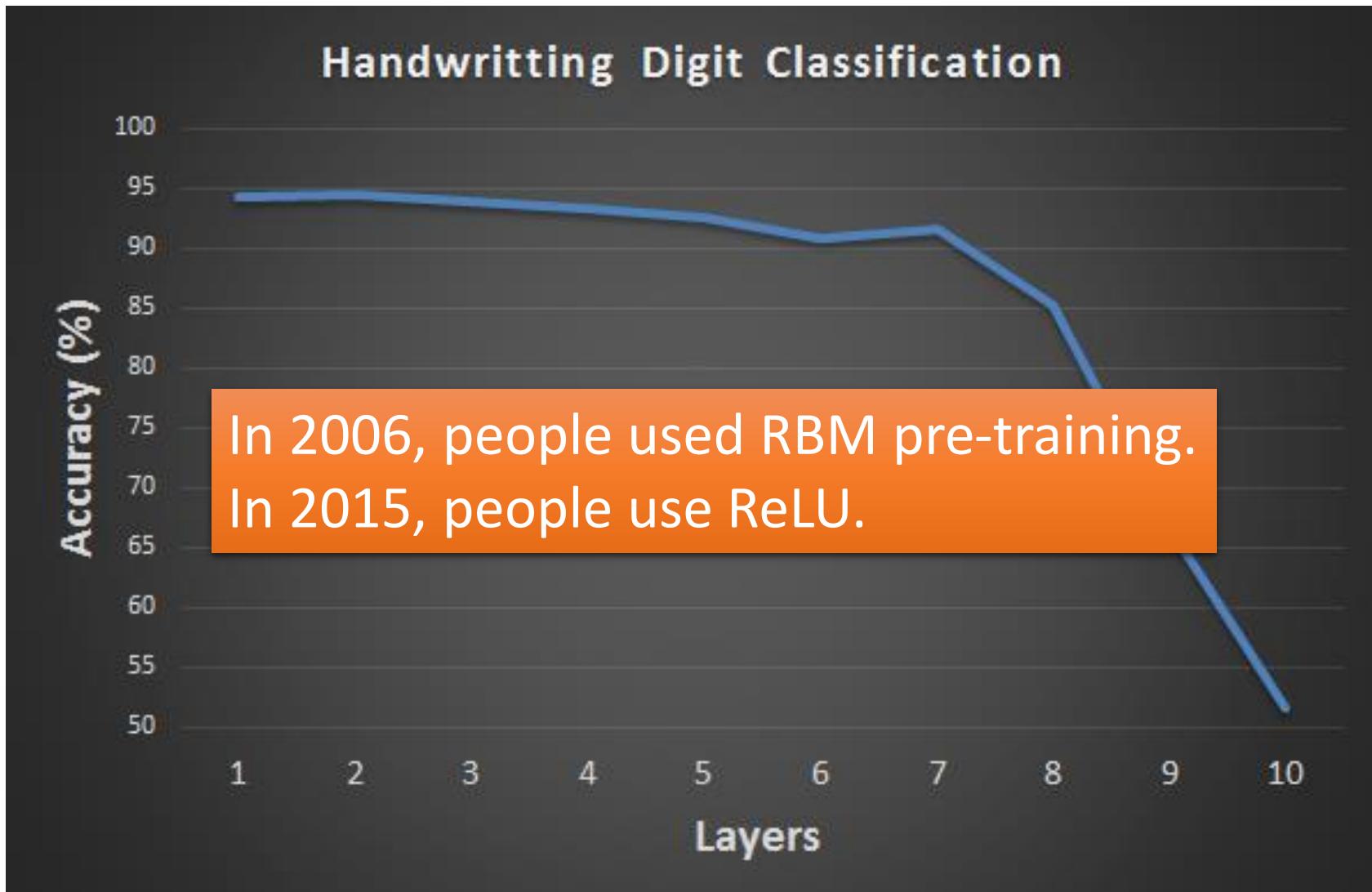
# Smaller gradients



## Intuitive way to compute the derivatives ...

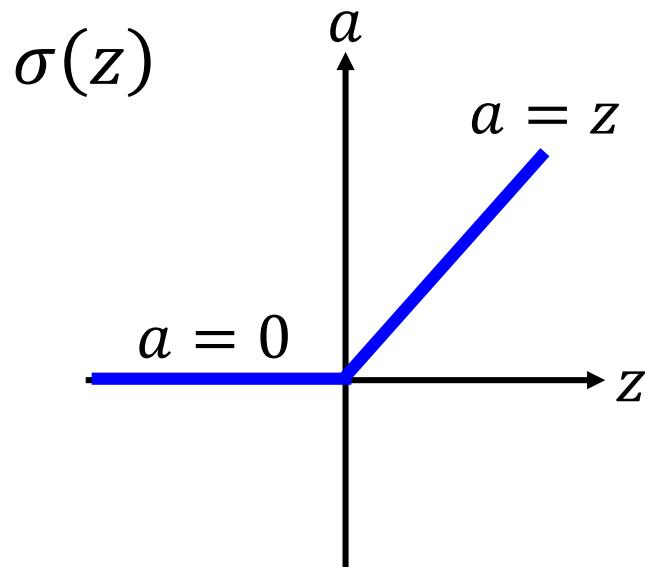
$$\frac{\partial l}{\partial w} = ? \quad \frac{\Delta l}{\Delta w}$$

# Hard to get the power of Deep ...



# ReLU

- Rectified Linear Unit (ReLU)

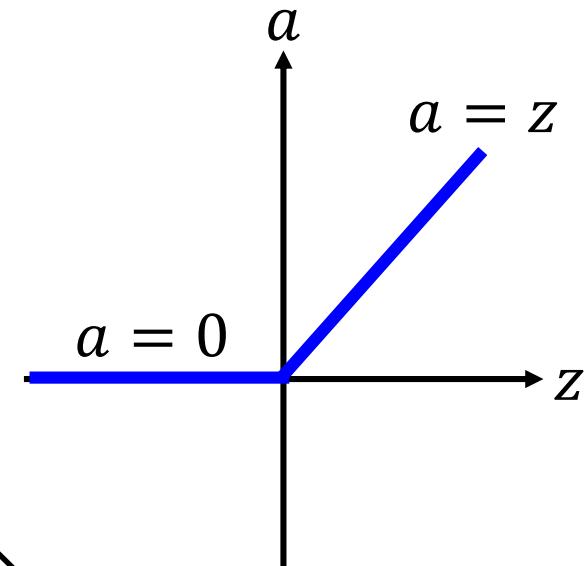
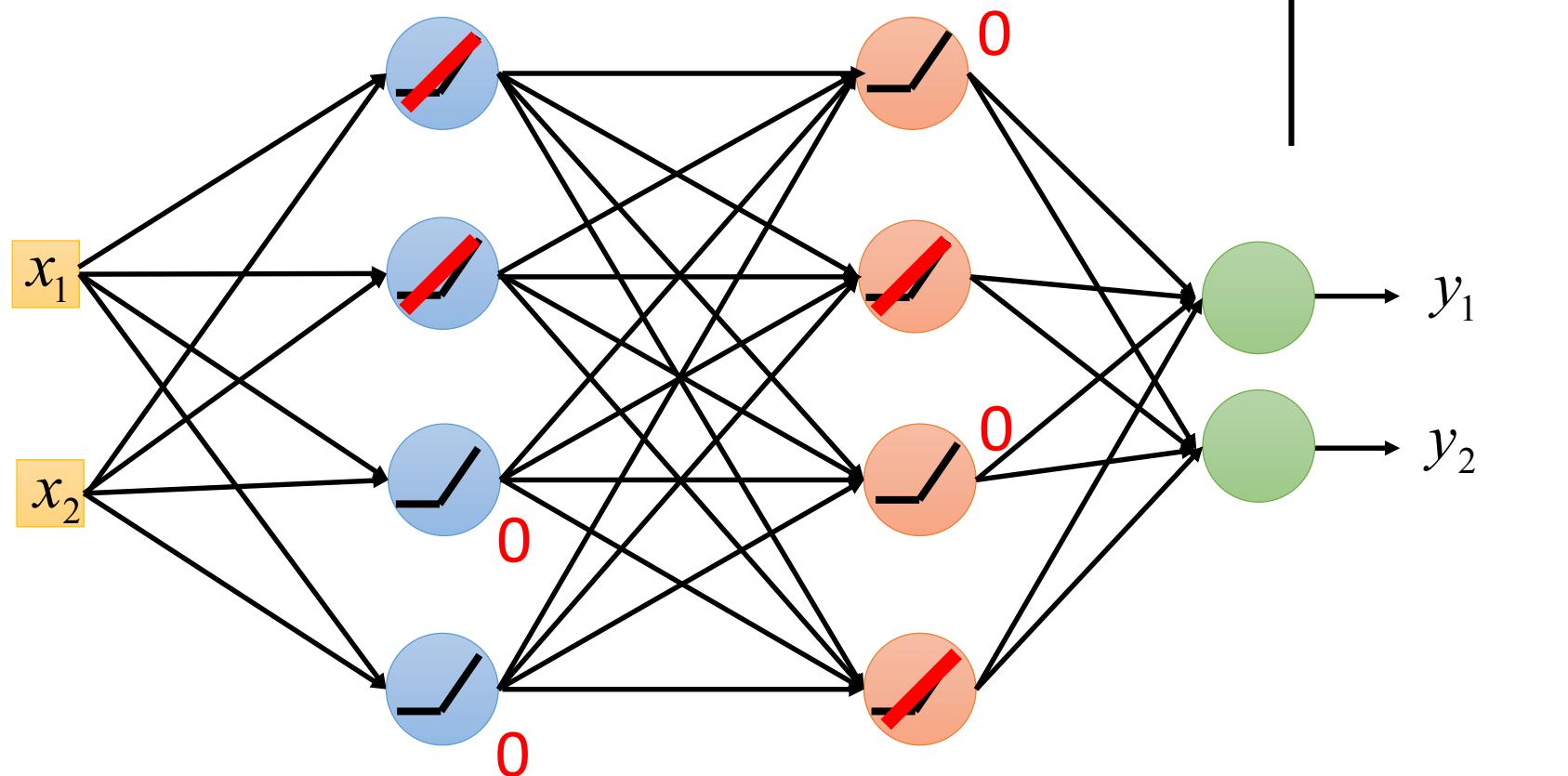


[Xavier Glorot, AISTATS'11]  
[Andrew L. Maas, ICML'13]  
[Kaiming He, arXiv'15]

## Reason:

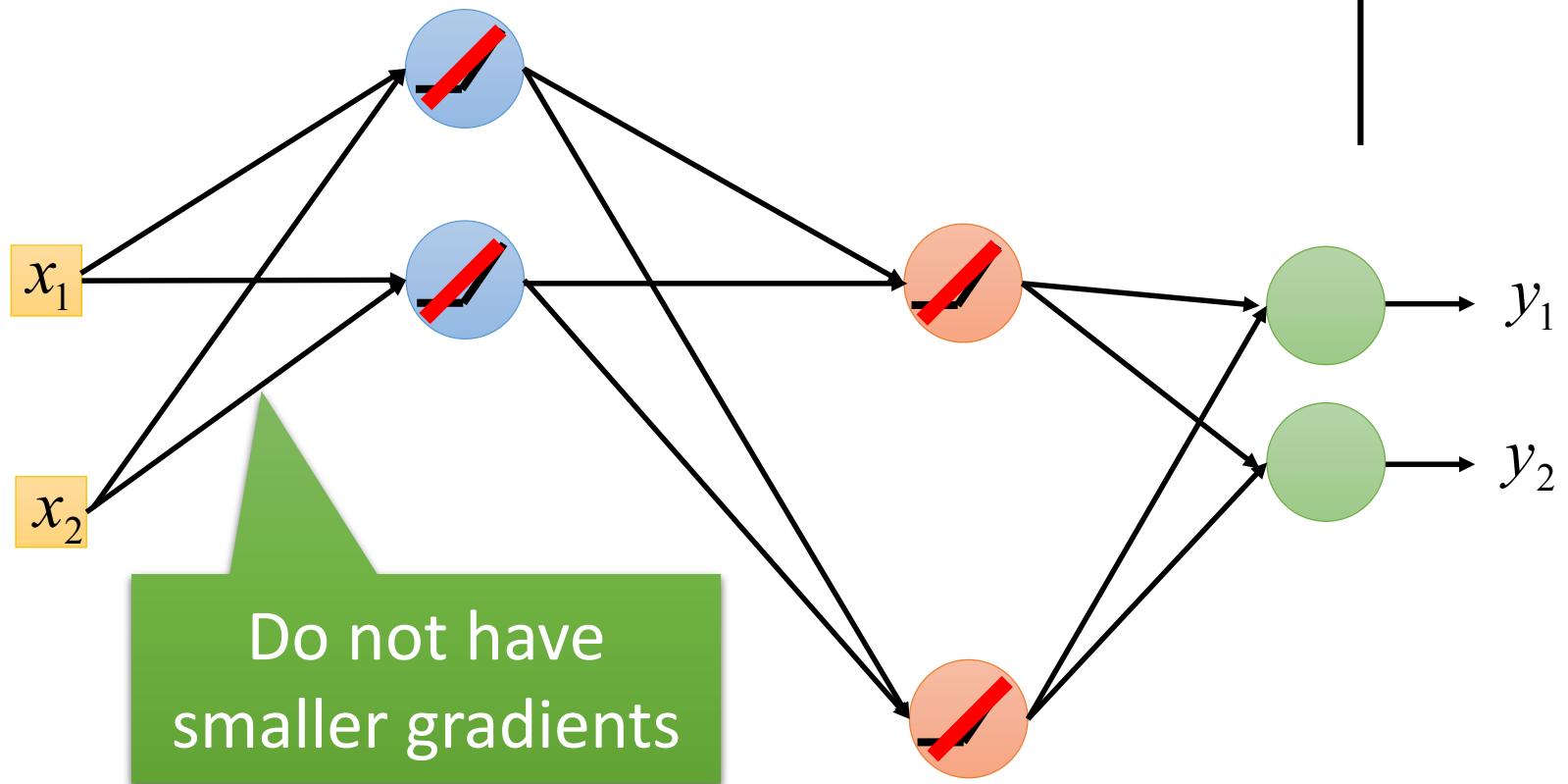
1. Fast to compute
2. Biological reason
3. Infinite sigmoid with different biases
4. Vanishing gradient problem

# ReLU



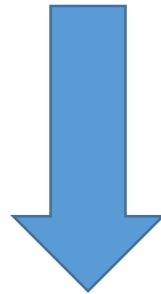
# ReLU

A Thinner linear network



Let's try it

```
model.add( Activation('sigmoid') )
```



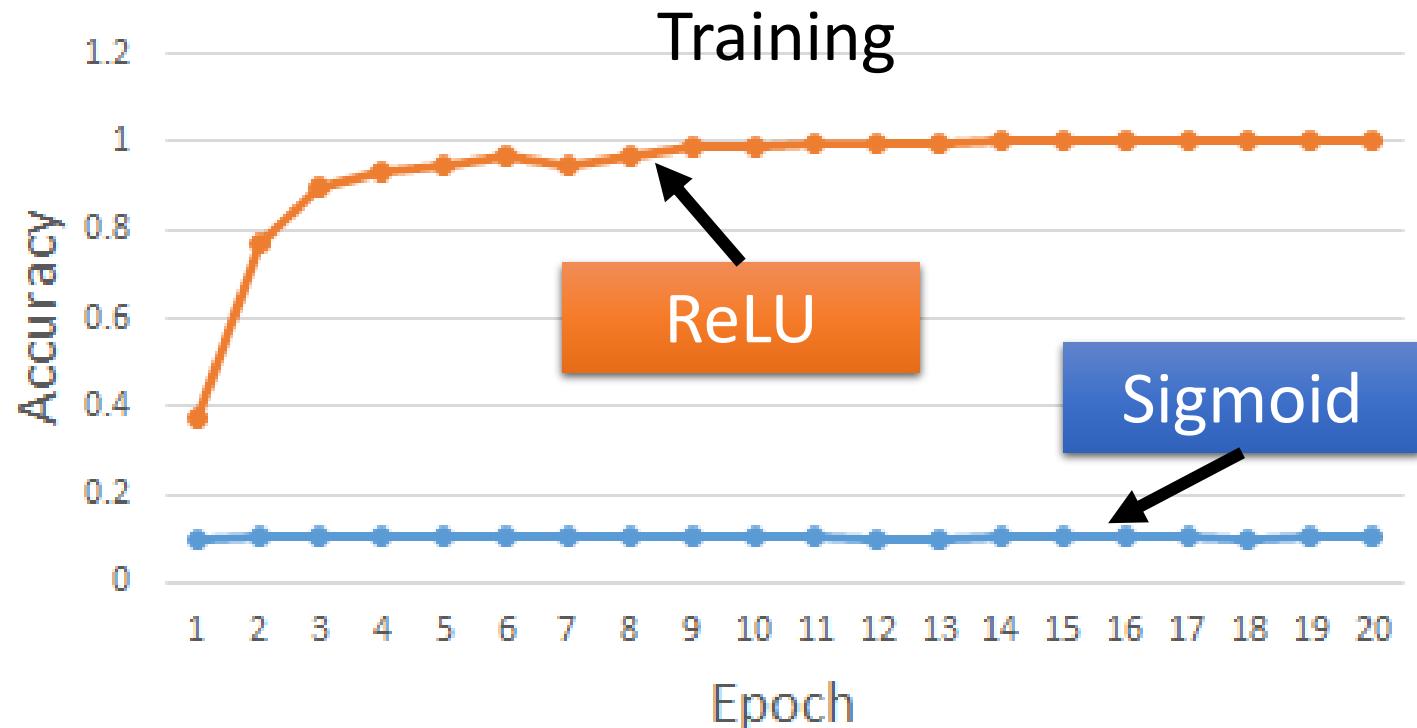
```
model.add( Activation('relu') )
```

# Let's try it

- 9 layers

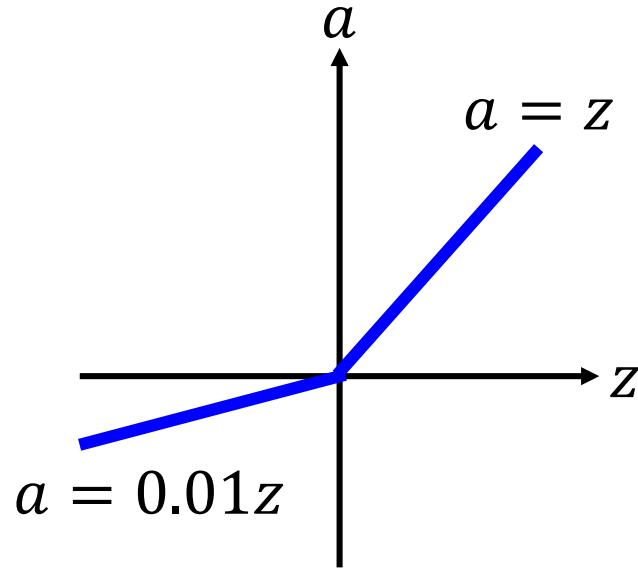
Testing:

9 layers	Accuracy
Sigmoid	0.11
ReLU	0.96

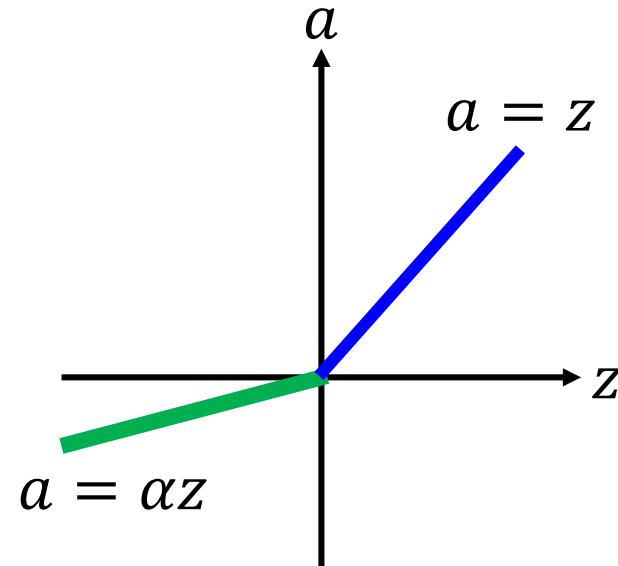


# ReLU - variant

*Leaky ReLU*



*Parametric ReLU*

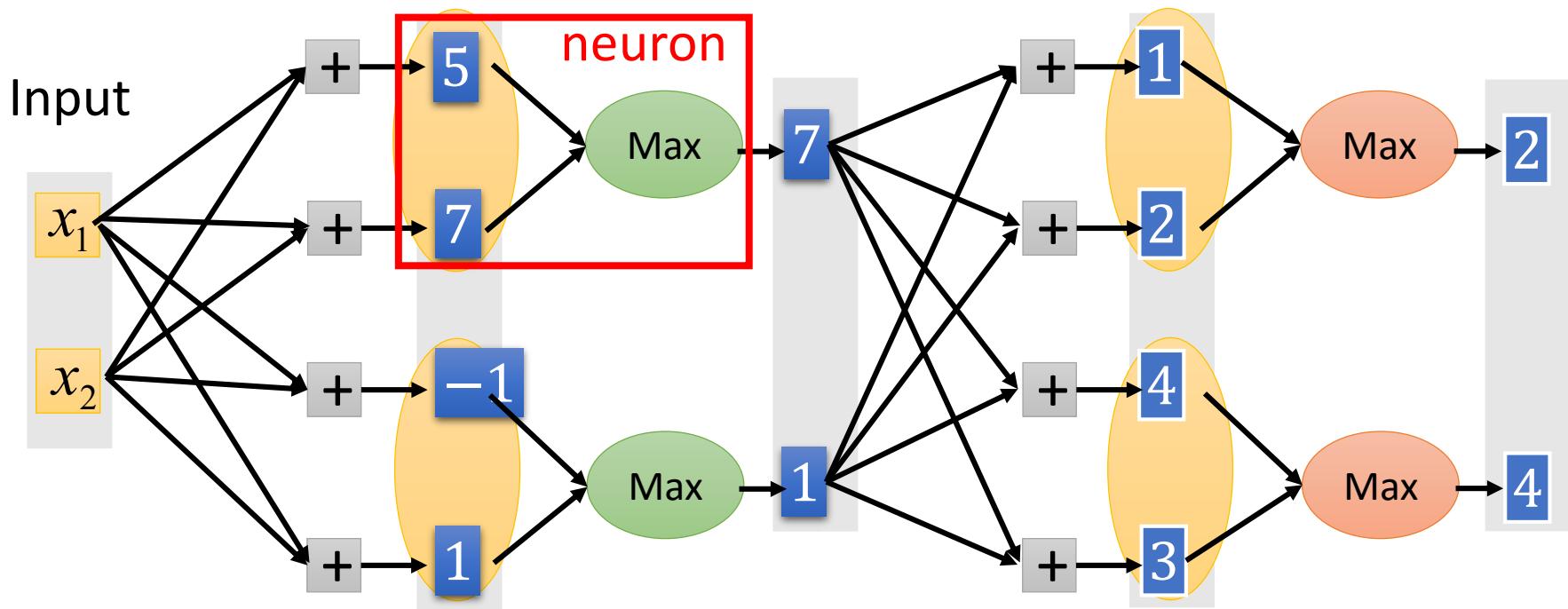


$\alpha$  also learned by  
gradient descent

# Maxout

ReLU is a special cases of Maxout

- Learnable activation function [Ian J. Goodfellow, ICML'13]



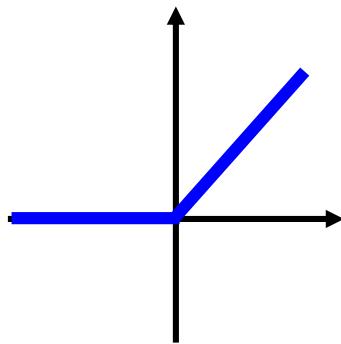
You can have more than 2 elements in a group.

# Maxout

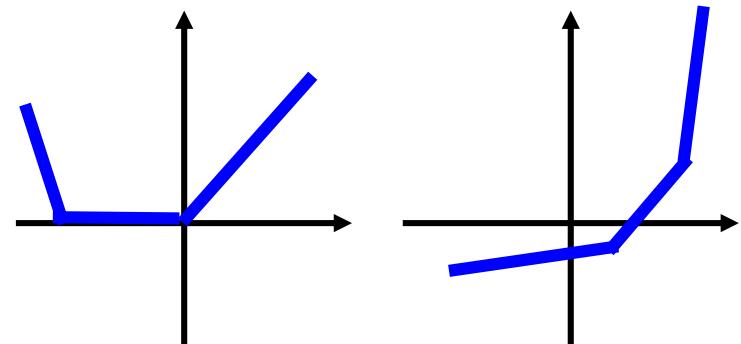
ReLU is a special cases of Maxout

- Learnable activation function [Ian J. Goodfellow, ICML'13]
  - Activation function in maxout network can be any piecewise linear convex function
  - How many pieces depending on how many elements in a group

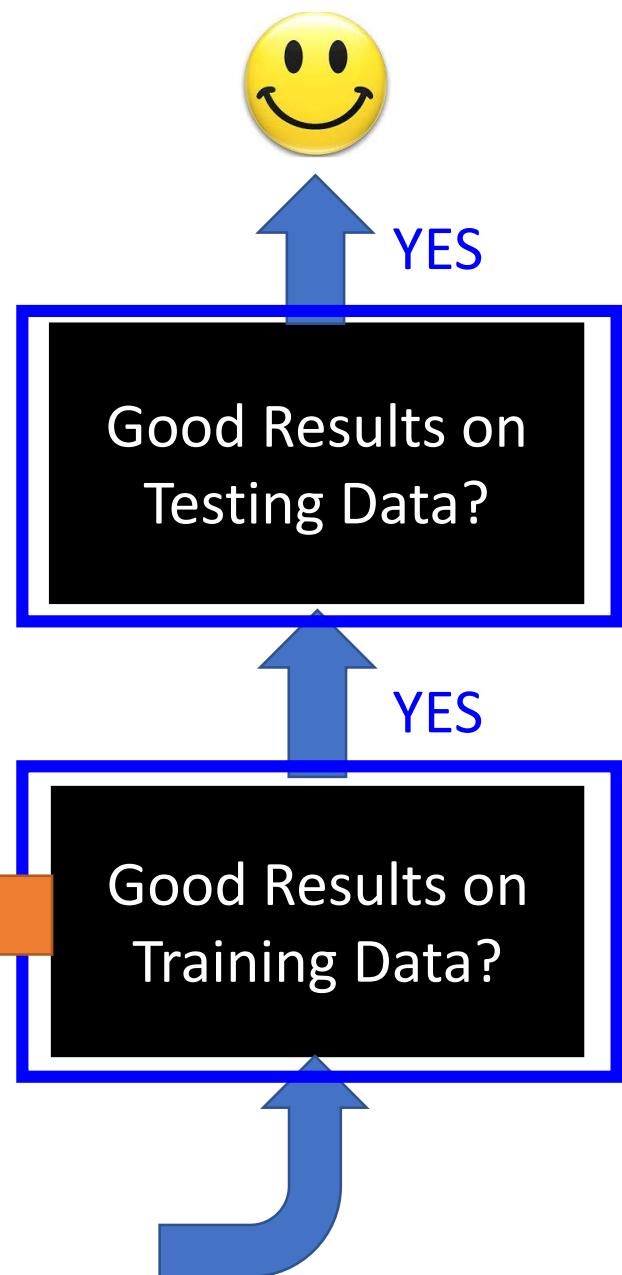
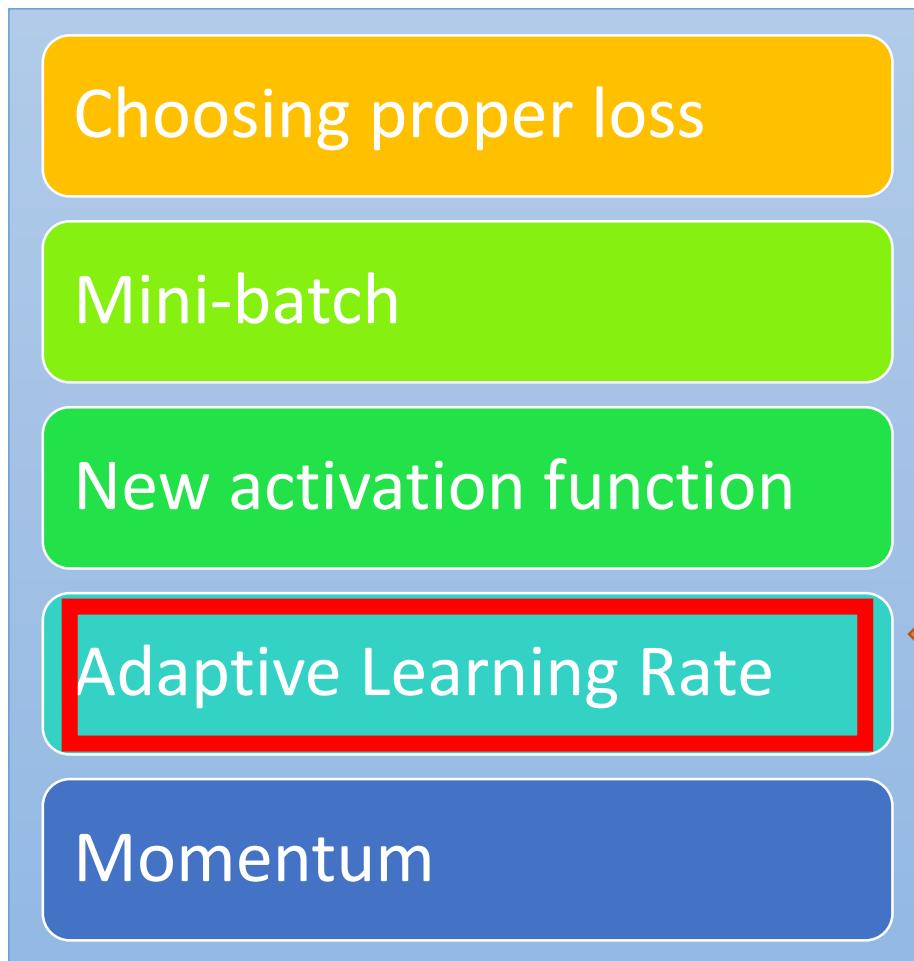
2 elements in a group



3 elements in a group

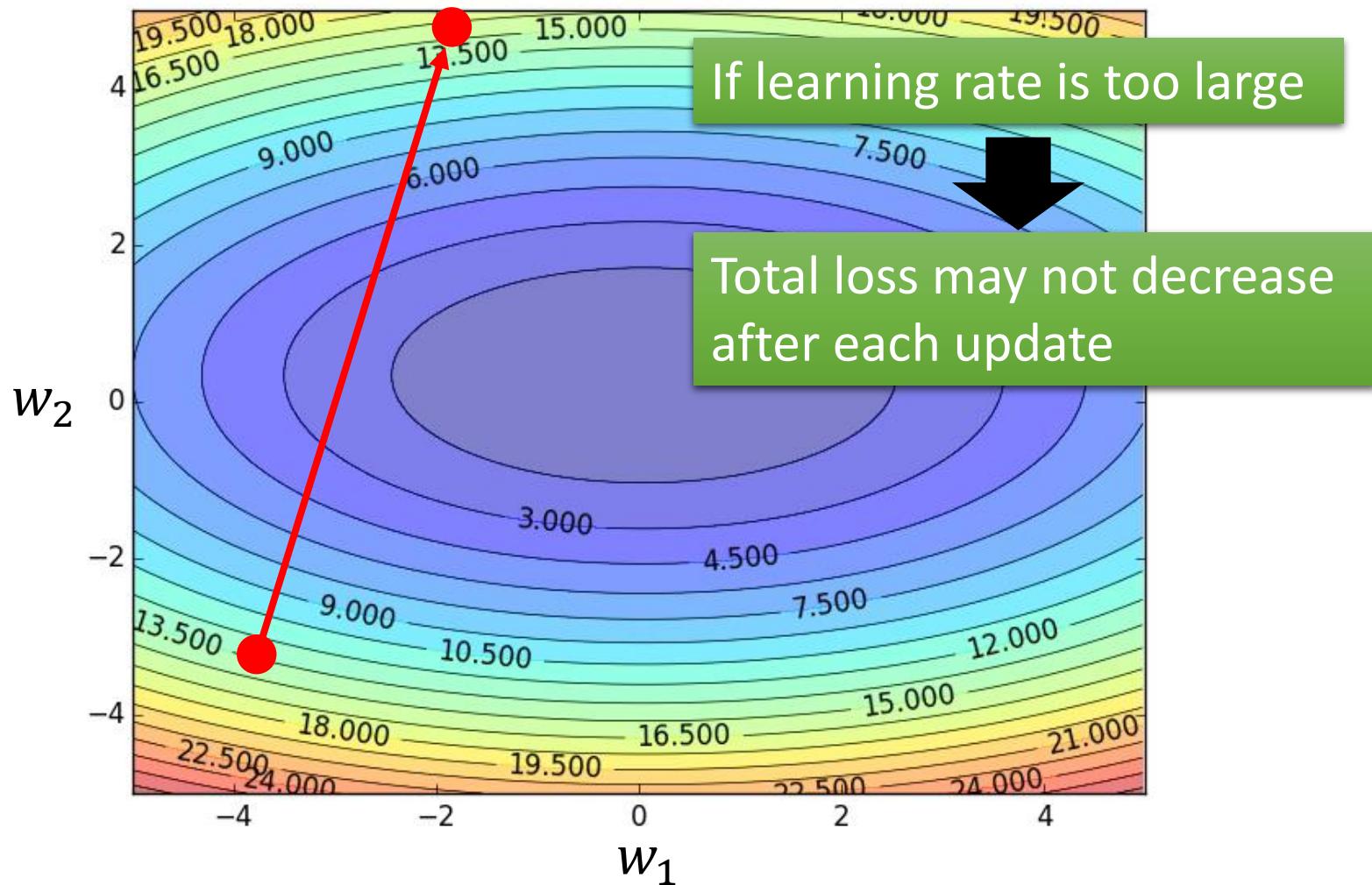


# Recipe of Deep Learning



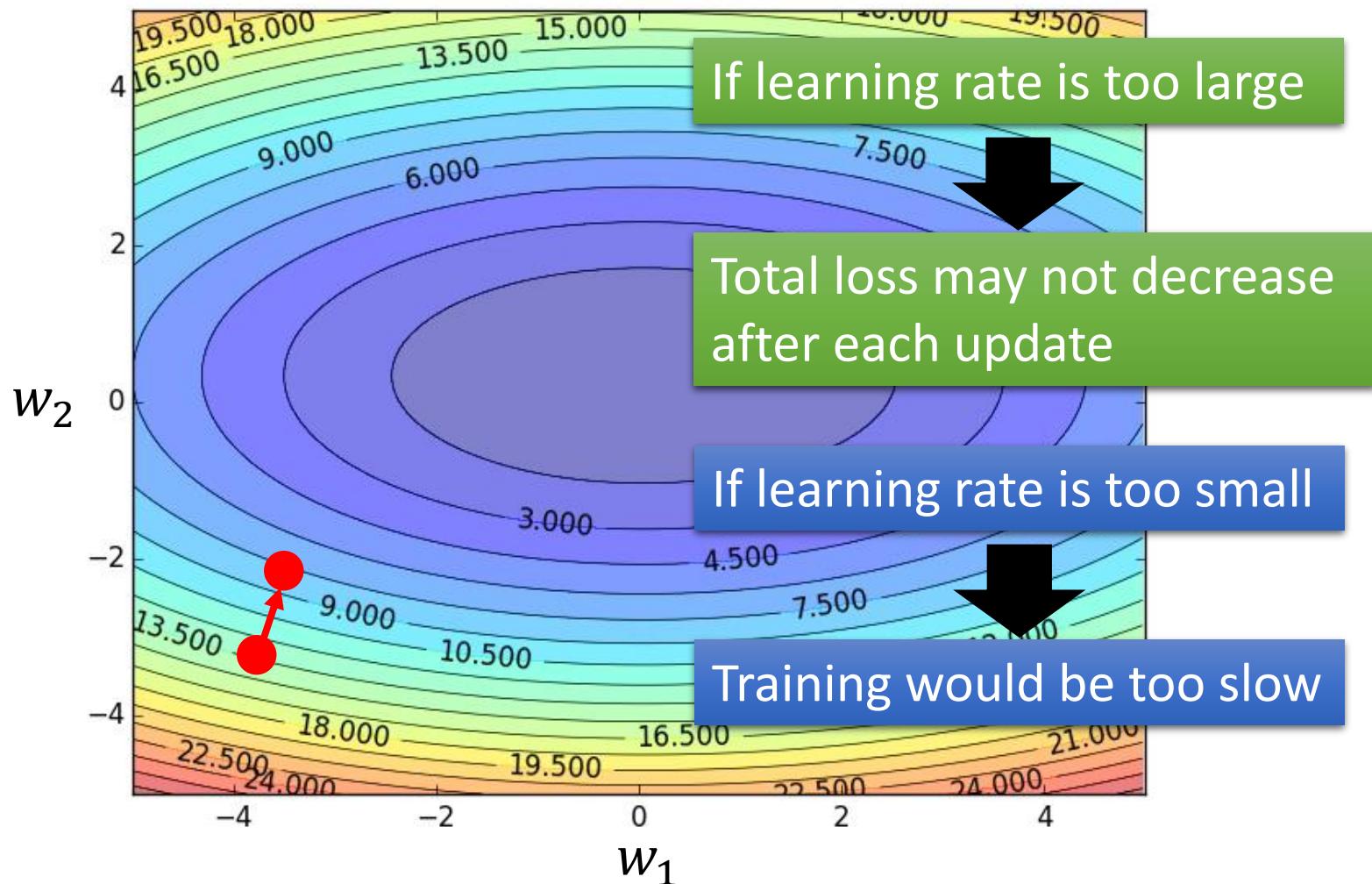
# Learning Rates

Set the learning rate  $\eta$  carefully



# Learning Rates

Set the learning rate  $\eta$  carefully



# Learning Rates

- Popular & Simple Idea: Reduce the learning rate by some factor every few epochs.
  - At the beginning, we are far from the destination, so we use larger learning rate
  - After several epochs, we are close to the destination, so we reduce the learning rate
  - E.g. 1/t decay:  $\eta^t = \eta / \sqrt{t + 1}$
- Learning rate cannot be one-size-fits-all
  - Giving different parameters different learning rates

# Adagrad

Original:  $w \leftarrow w - \eta \partial L / \partial w$

Adagrad:  $w \leftarrow w - \boxed{\eta_w} \partial L / \partial w$

Parameter dependent learning rate

$$\eta_w = \frac{\eta}{\sqrt{\sum_{i=0}^t (g^i)^2}}$$

constant

$g^i$  is  $\partial L / \partial w$  obtained at the i-th update

Summation of the square of the previous derivatives

# Adagrad

$$\eta_w = \frac{\eta}{\sqrt{\sum_{i=0}^t (g^i)^2}}$$

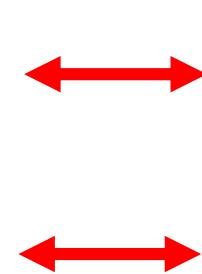
$w_1$	$\mathbf{g}^0$
	0.1

Learning rate:

$$\frac{\eta}{\sqrt{0.1^2}}$$

$$= \frac{\eta}{0.1}$$

$$\frac{\eta}{\sqrt{0.1^2 + 0.2^2}} = \frac{\eta}{\sqrt{0.01 + 0.04}} = \frac{\eta}{\sqrt{0.05}} = \frac{\eta}{\sqrt{0.05}} = \frac{\eta}{0.22}$$



$w_2$	$\mathbf{g}^0$
	20.0

Learning rate:

$$\frac{\eta}{\sqrt{20^2}}$$

$$= \frac{\eta}{20}$$

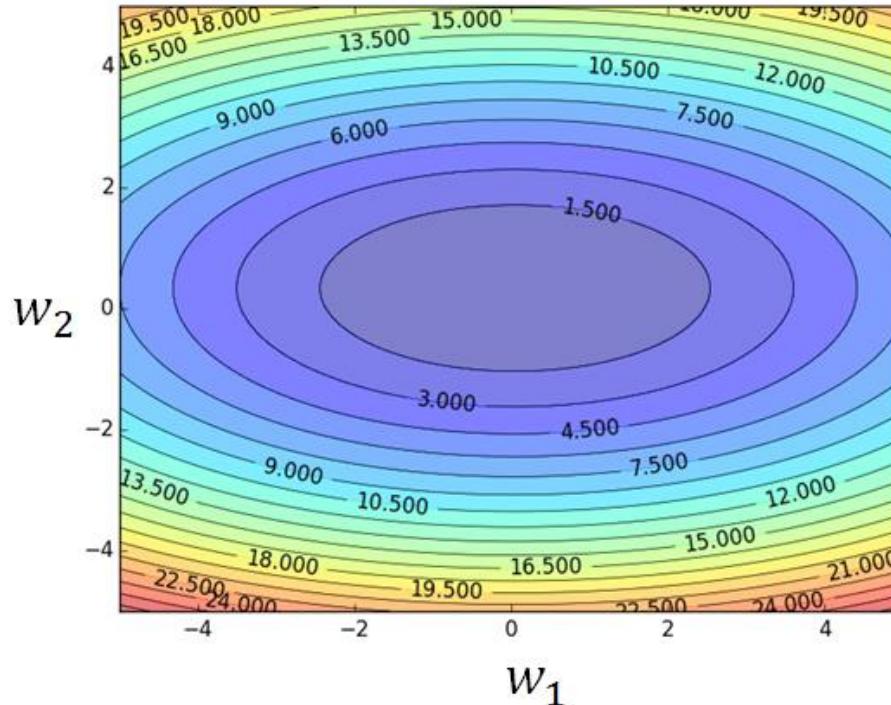
$$\frac{\eta}{\sqrt{20^2 + 10^2}} = \frac{\eta}{\sqrt{400 + 100}} = \frac{\eta}{\sqrt{500}} = \frac{\eta}{\sqrt{500}} = \frac{\eta}{22}$$

- Observation:**
1. Learning rate is smaller and smaller for all parameters
  2. Smaller derivatives, larger learning rate, and vice versa

Why?

Larger derivatives

Smaller Learning Rate



Smaller Derivatives



Larger Learning Rate

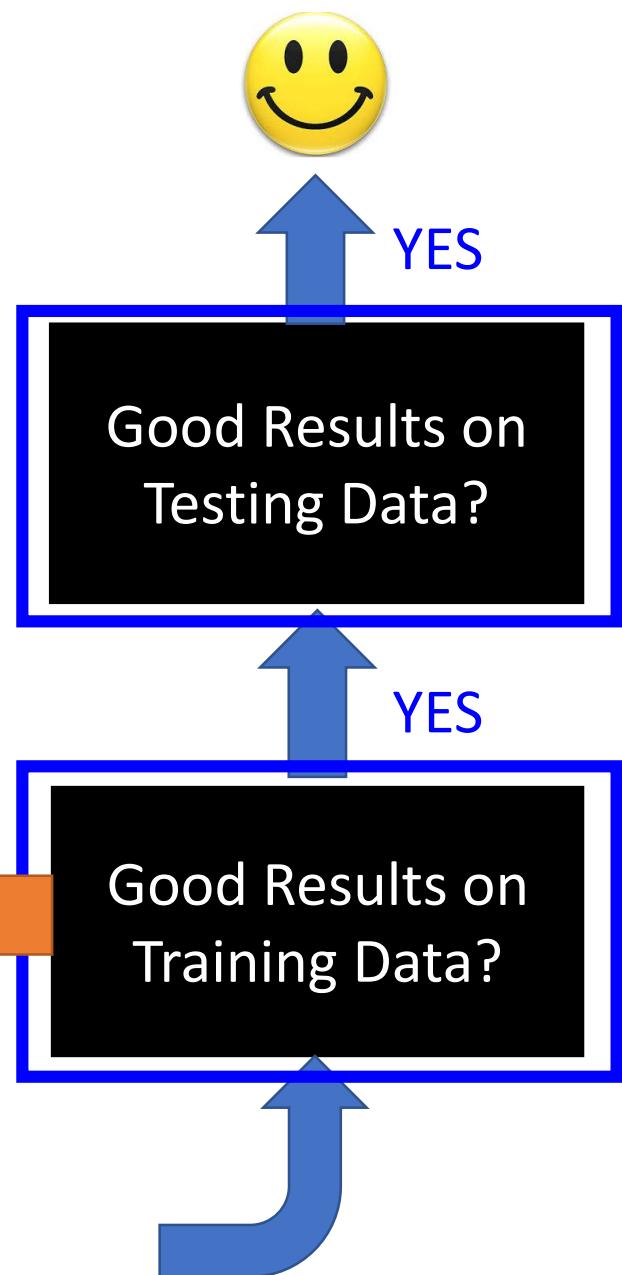
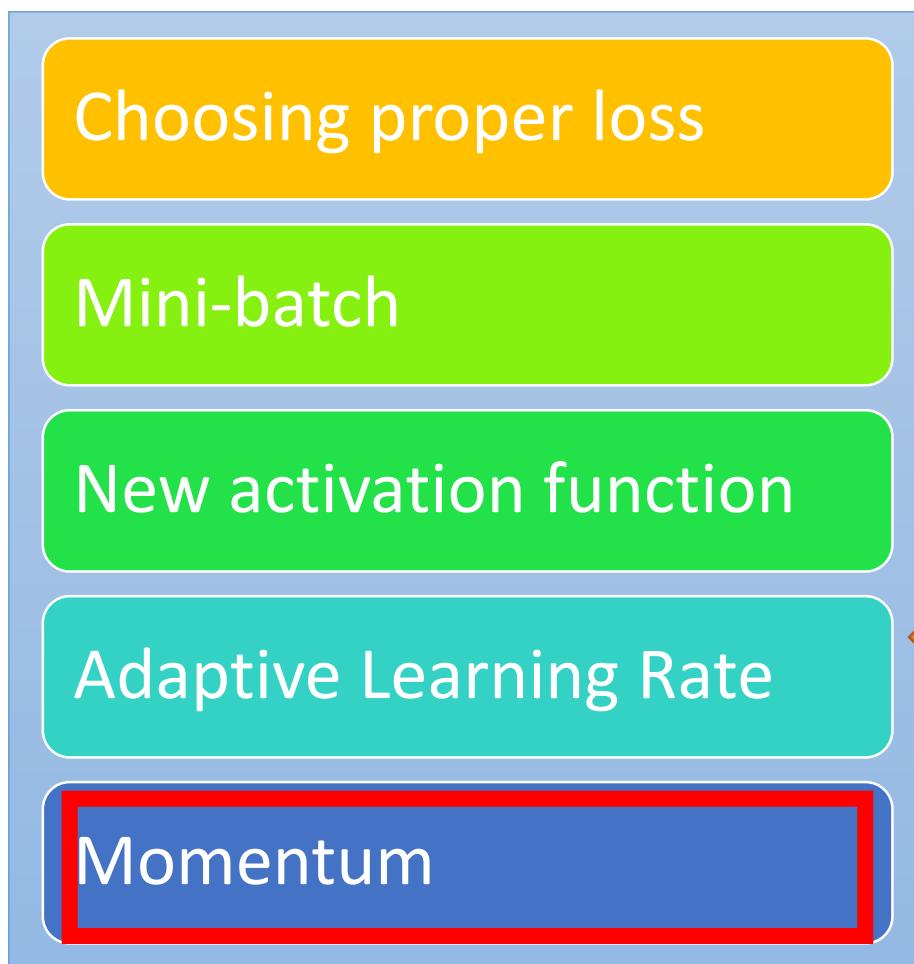
2. Smaller derivatives, larger learning rate, and vice versa

Why?

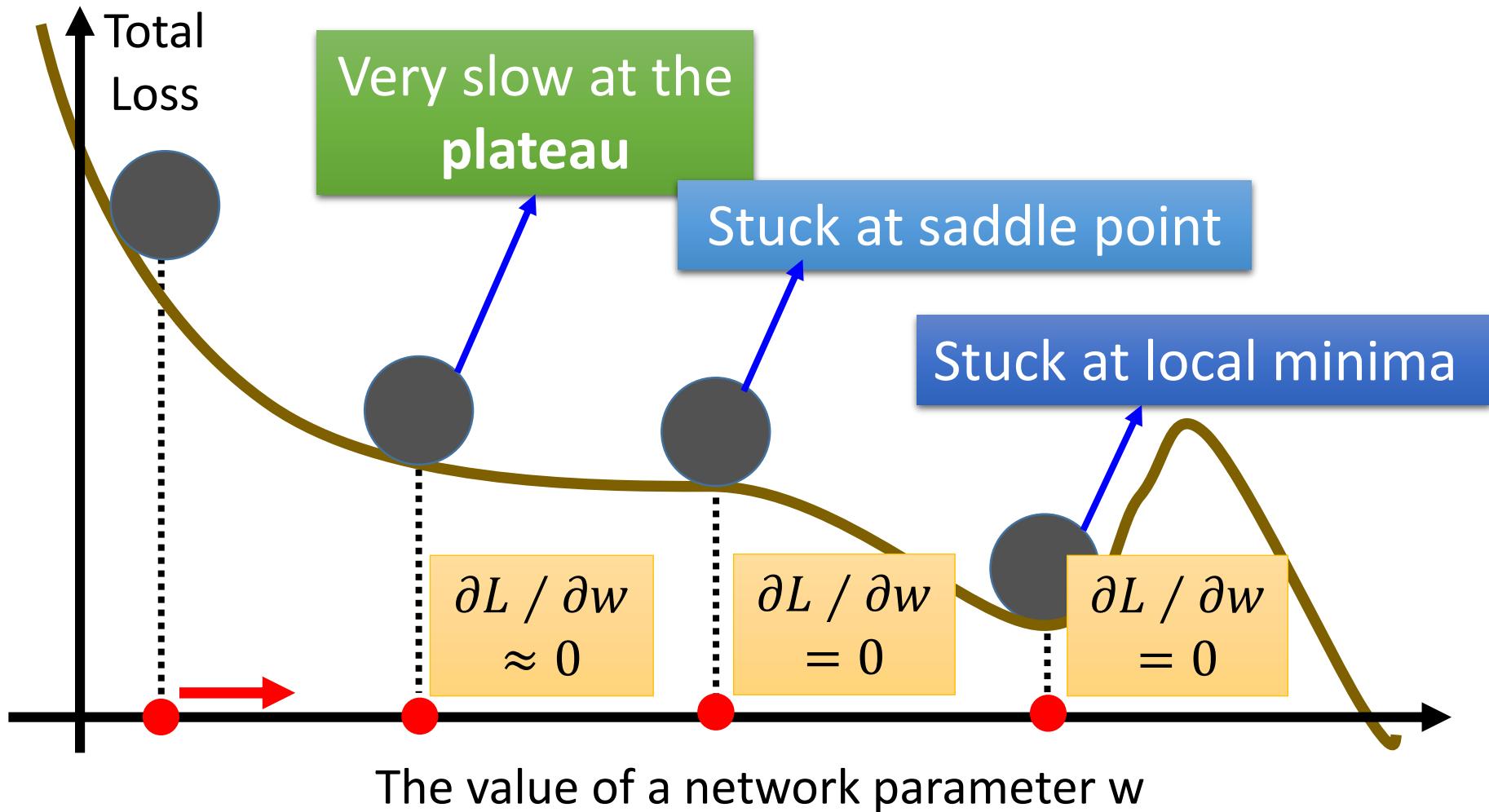
# Not the whole story .....

- Adagrad [John Duchi, JMLR'11]
- RMSprop
  - <https://www.youtube.com/watch?v=O3sxAc4hxZU>
- Adadelta [Matthew D. Zeiler, arXiv'12]
- “No more pesky learning rates” [Tom Schaul, arXiv'12]
- AdaSecant [Caglar Gulcehre, arXiv'14]
- Adam [Diederik P. Kingma, ICLR'15]
- Nadam
  - [http://cs229.stanford.edu/proj2015/054\\_report.pdf](http://cs229.stanford.edu/proj2015/054_report.pdf)

# Recipe of Deep Learning

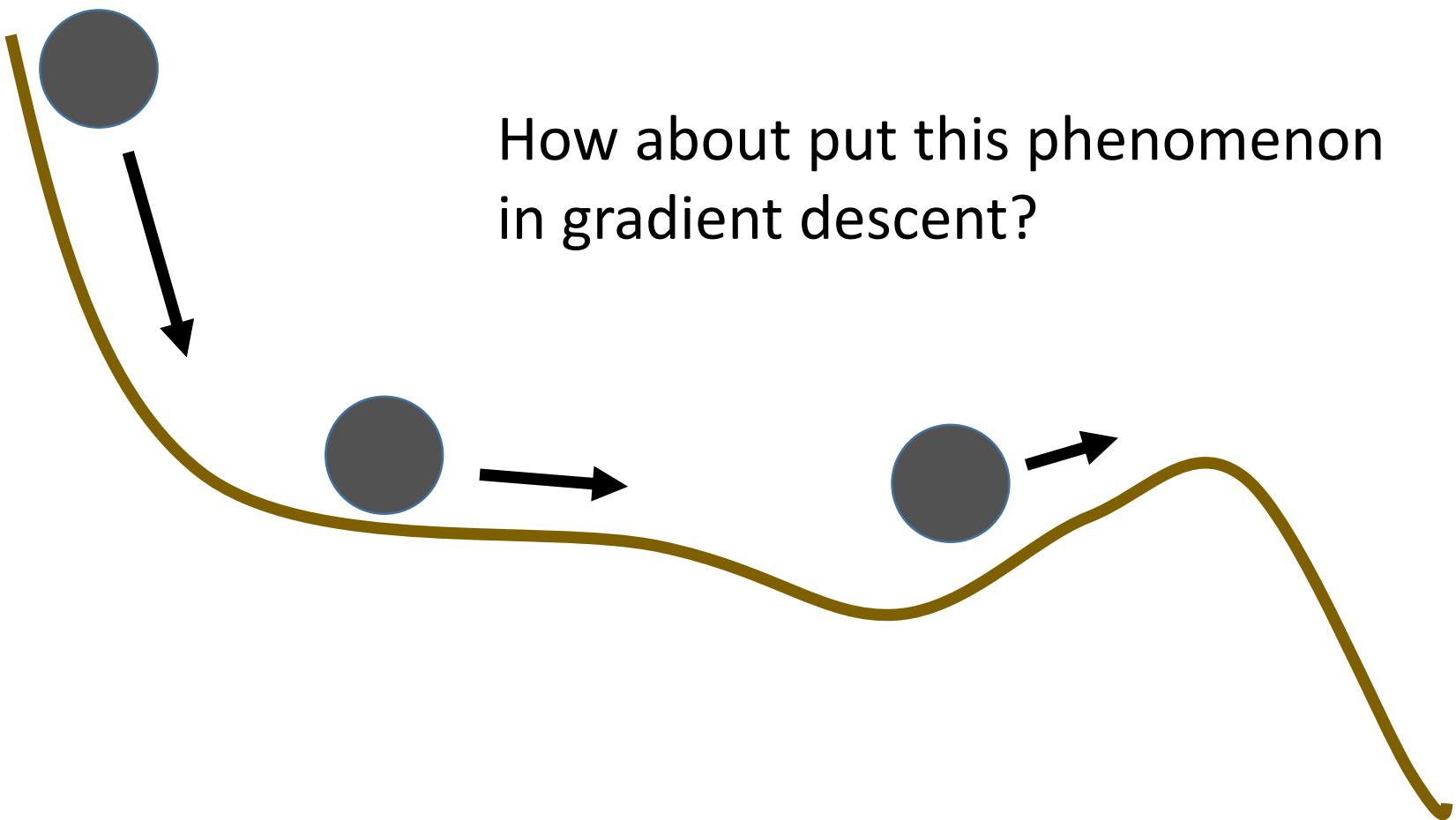


# Hard to find optimal network parameters



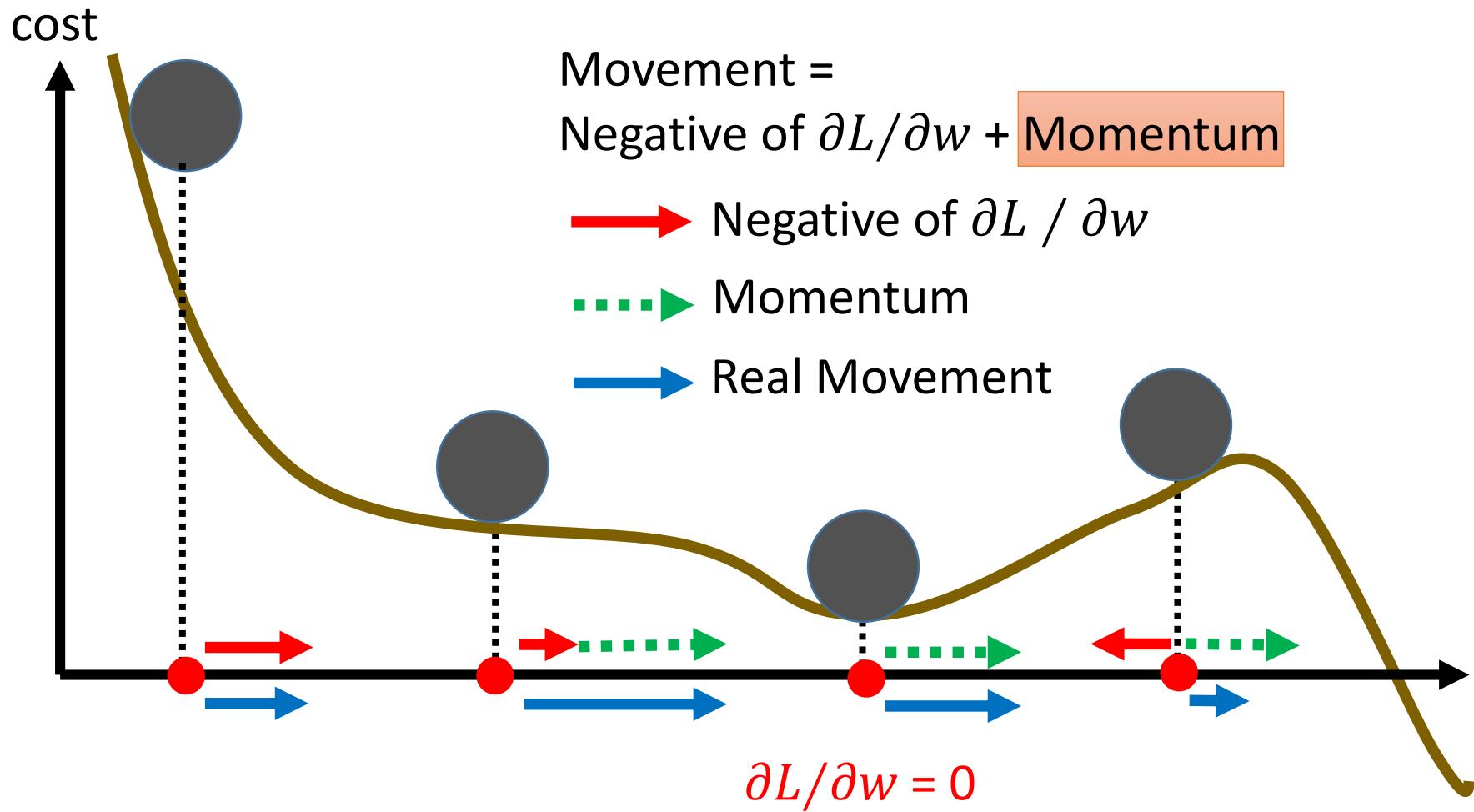
# In physical world .....

- Momentum



# Momentum

Still not guarantee reaching global minima, but give some hope .....



# Adam

## RMSProp (Advanced Adagrad) + Momentum

```
model.compile(loss='categorical_crossentropy',
               optimizer=SGD(lr=0.1),
               metrics=['accuracy'])
```

```
model.compile(loss='categorical_crossentropy',
               optimizer=Adam(),
               metrics=['accuracy'])
```

**Algorithm 1:** *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation.  $g_t^2$  indicates the elementwise square  $g_t \odot g_t$ . Good default settings for the tested machine learning problems are  $\alpha = 0.001$ ,  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$  and  $\epsilon = 10^{-8}$ . All operations on vectors are element-wise. With  $\beta_1^t$  and  $\beta_2^t$  we denote  $\beta_1$  and  $\beta_2$  to the power  $t$ .

---

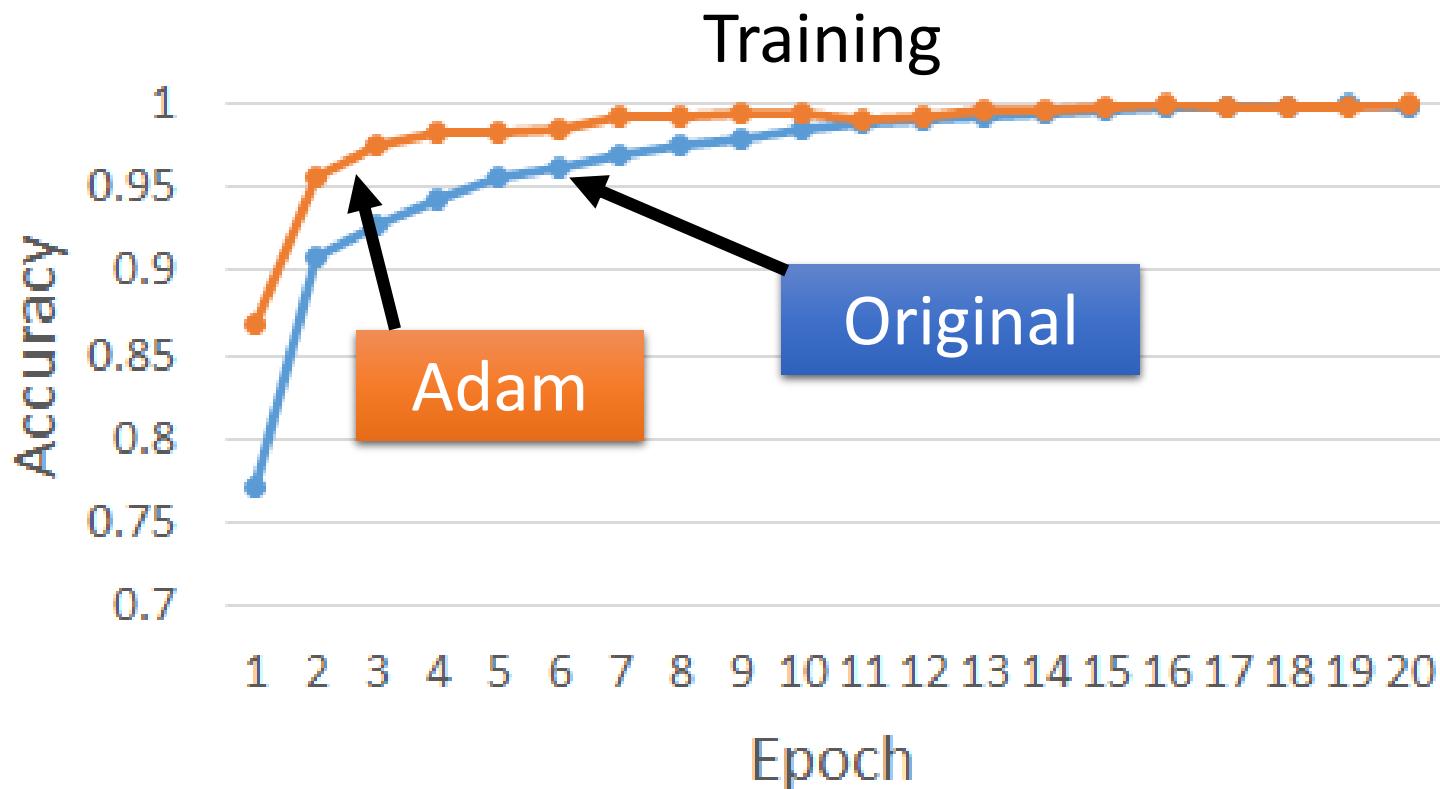
**Require:**  $\alpha$ : Stepsize  
**Require:**  $\beta_1, \beta_2 \in [0, 1]$ : Exponential decay rates for the moment estimates  
**Require:**  $f(\theta)$ : Stochastic objective function with parameters  $\theta$   
**Require:**  $\theta_0$ : Initial parameter vector  
 $m_0 \leftarrow 0$  (Initialize 1<sup>st</sup> moment vector)  
 $v_0 \leftarrow 0$  (Initialize 2<sup>nd</sup> moment vector)  
 $t \leftarrow 0$  (Initialize timestep)  
**while**  $\theta_t$  not converged **do**  
     $t \leftarrow t + 1$   
     $g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$  (Get gradients w.r.t. stochastic objective at timestep  $t$ )  
     $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$  (Update biased first moment estimate)  
     $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$  (Update biased second raw moment estimate)  
     $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$  (Compute bias-corrected first moment estimate)  
     $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$  (Compute bias-corrected second raw moment estimate)  
     $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$  (Update parameters)  
**end while**  
**return**  $\theta_t$  (Resulting parameters)

---

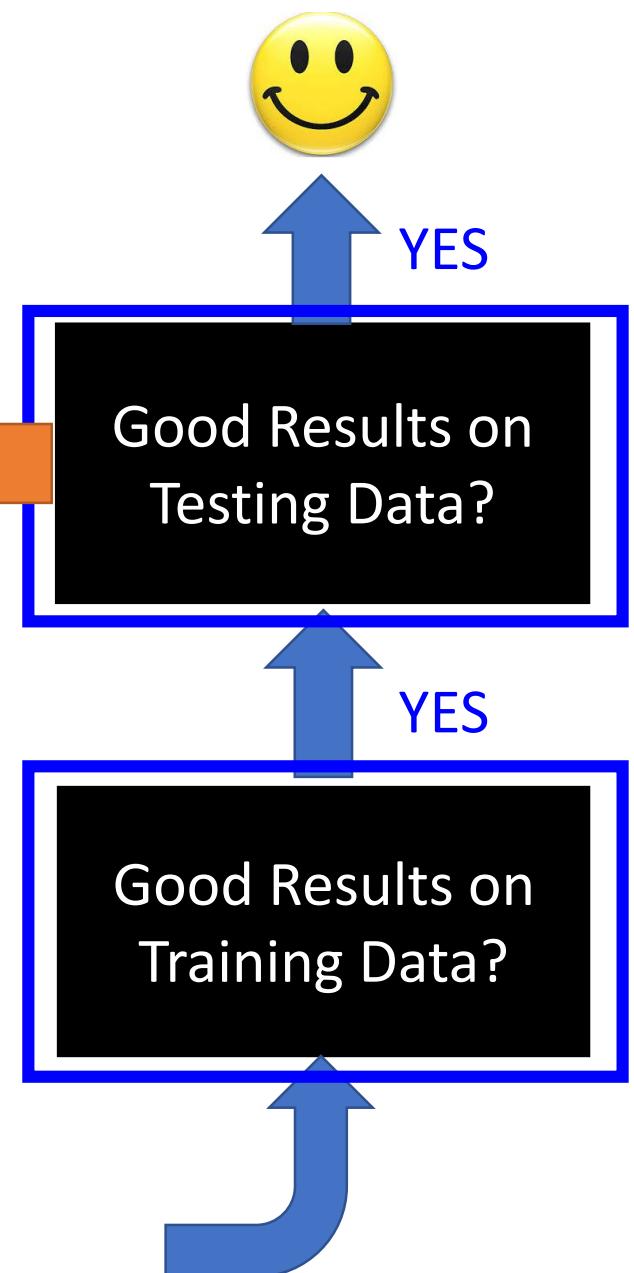
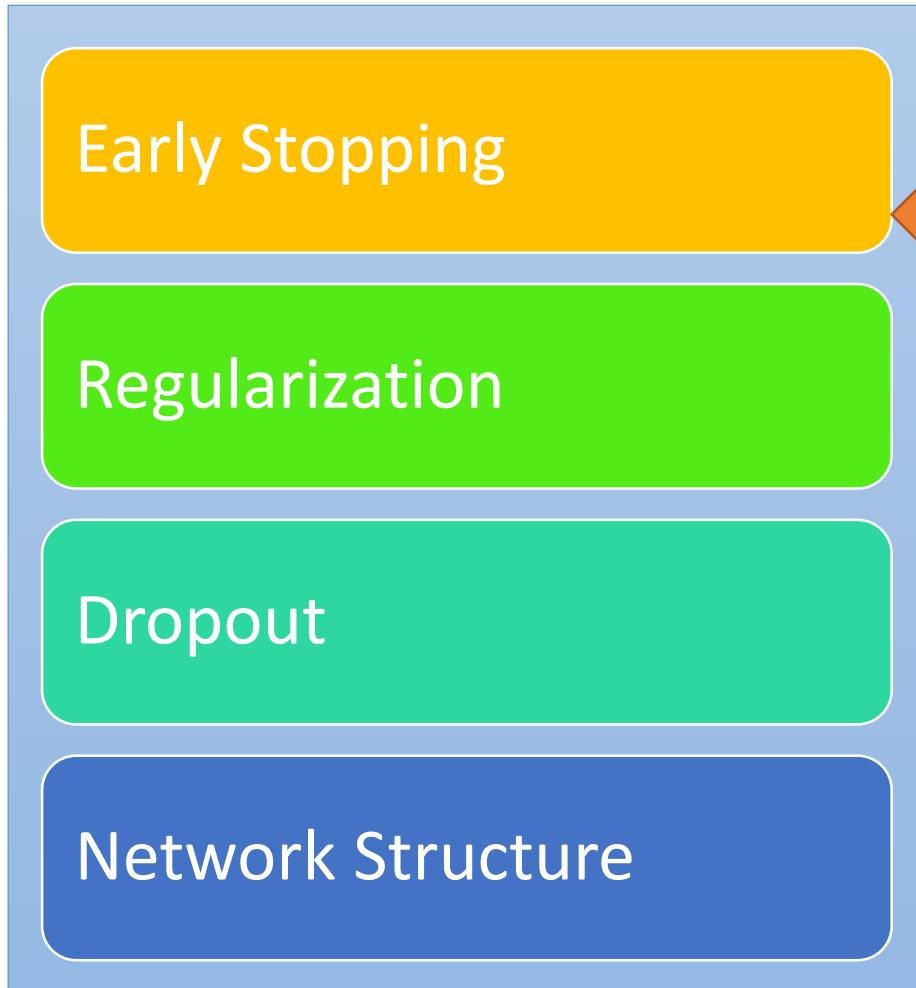
# Let's try it

- ReLU, 3 layer

	Accuracy
Original	0.96
Adam	0.97



# Recipe of Deep Learning



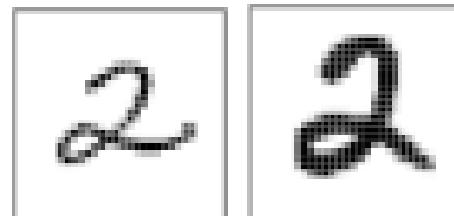
# Why Overfitting?

- Training data and testing data can be different.

Training Data:



Testing Data:



Learning target is defined by the training data.

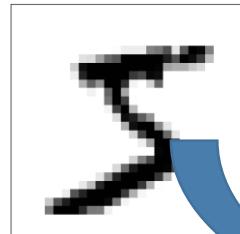
The parameters achieving the learning target do not necessarily have good results on the testing data.

# Panacea for Overfitting

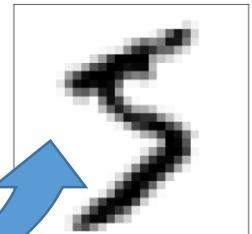
- Have more training data
- *Create* more training data (?)

Handwriting recognition:

Original  
Training Data:



Created  
Training Data:



Shift 15 °

# Why Overfitting?

- For experiments, we added some noises to the testing data

```
-1.36230370e-01, 1.03749340e-01, 1.15432226e-01,  
2.58670464e-01, 1.48774333e+00, 1.92885328e+00,  
1.70038673e+00, 2.46242981e+00, 1.21244572e+00,  
-9.28660713e-01, 3.63209761e-01, -1.81327713e+00,  
-1.97910760e-01, 4.32874592e-01, -5.40565788e-01,  
2.95630655e-01, 2.07984424e+00, -1.84243292e+00,  
-5.11166017e-01, -5.80935128e-01, 1.06273647e+00,  
1.80551097e-02, 2.27983997e-02, -1.67979148e+00,  
8.12423001e-01, -6.25888706e-01, -1.25027082e+00,  
6.15135458e-01, -1.21394611e-01, -1.28089527e+00,  
3.24609806e-01, 6.70569391e-01, 1.49161323e-01,  
8.01573609e-01, 6.43116741e-01, -9.37629233e-02,  
1.74677366e+00, 6.80996008e-01, -7.03717611e-01,  
1.02079749e-01, 1.19505614e+00, -2.77959386e-01,  
-5.21652916e-02, 3.53683601e-01, -4.08310762e-01,  
-1.81042967e+00, -9.03308062e-01, 1.05404509e+00,  
-9.80876877e-01, 3.52078891e-01, 6.65981840e-01,  
1.06550150e+00, -2.28433613e-01, 3.64483904e-01,  
-1.51484666e+00, -7.52612872e-02, -2.97058082e-01,  
-7.27414382e-01, -2.45875340e-01, -1.27948942e-01,  
-3.69310620e-01, -2.62300428e+00, 2.11585073e+00,  
6.85561585e-01, -1.57443985e-01, 1.38128777e+00,  
6.84265587e-02, 3.12536292e-01, 4.54253185e-01,  
-7.88471875e-01, -6.58403343e-02, -1.41847985e+00,  
-1.39753340e-01, -5.55354856e-01, -5.01917779e-01,  
6.93118522e-01, -2.45360497e-01, -1.26943186e+00,  
-2.62323855e-01])  
n [3]: x test[0]
```

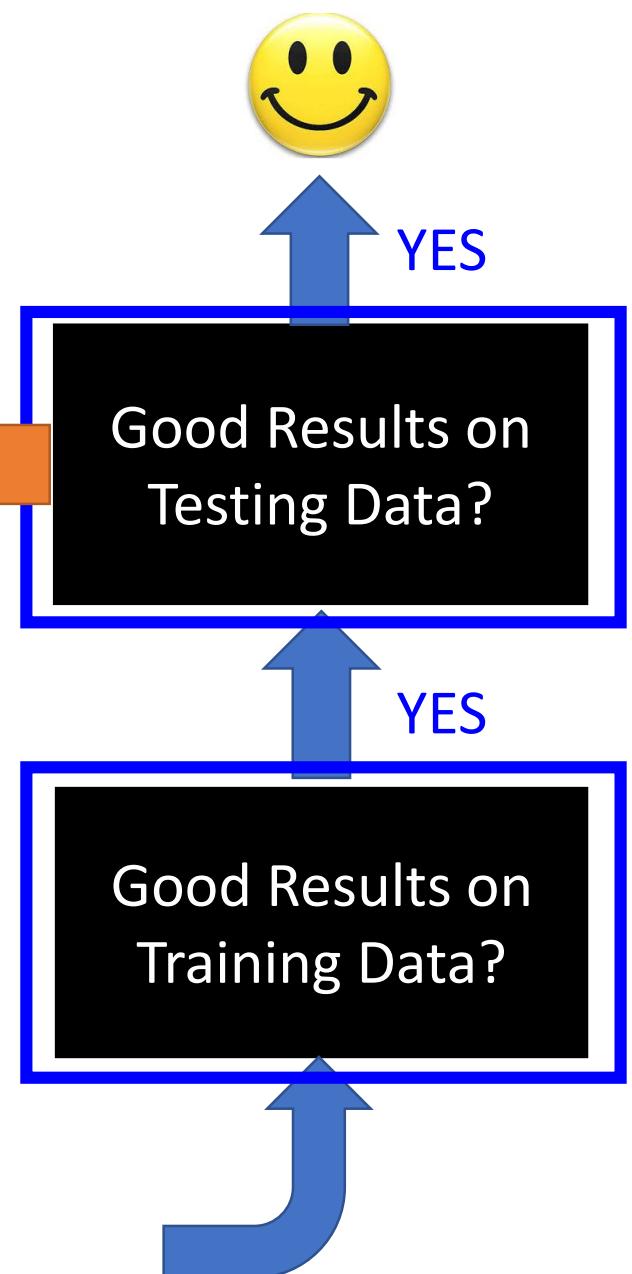
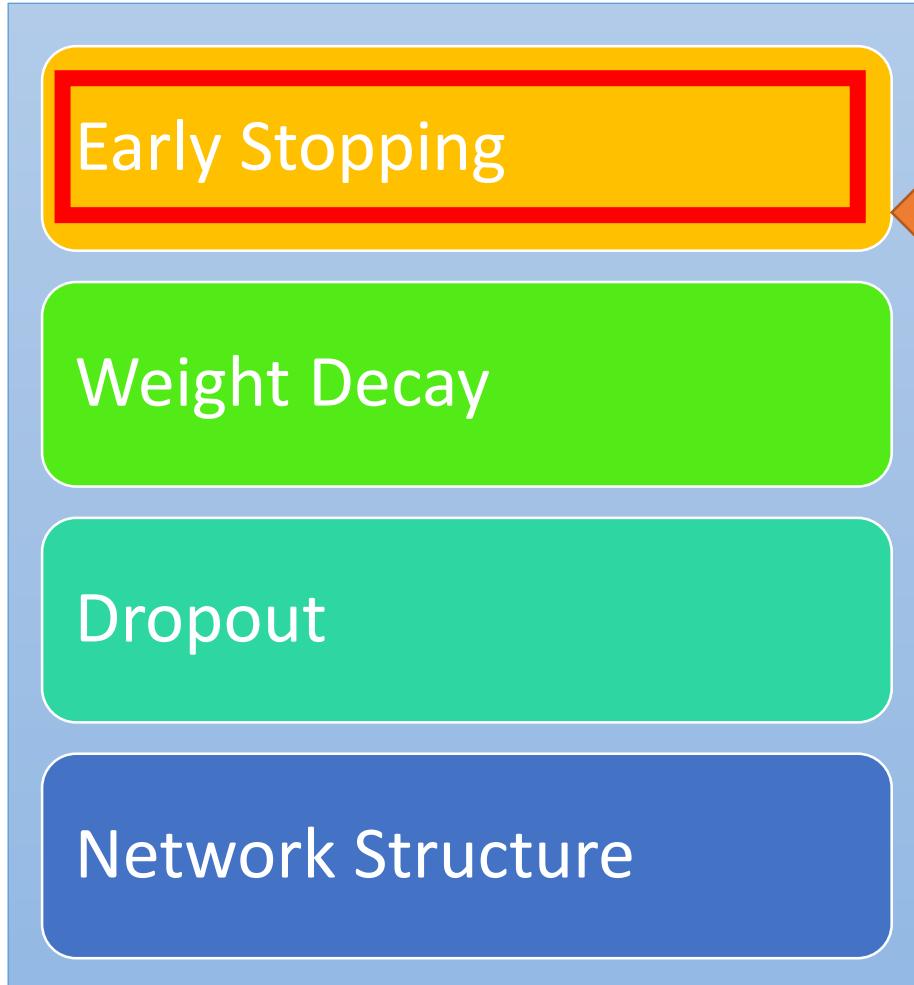
# Why Overfitting?

- For experiments, we added some noises to the testing data

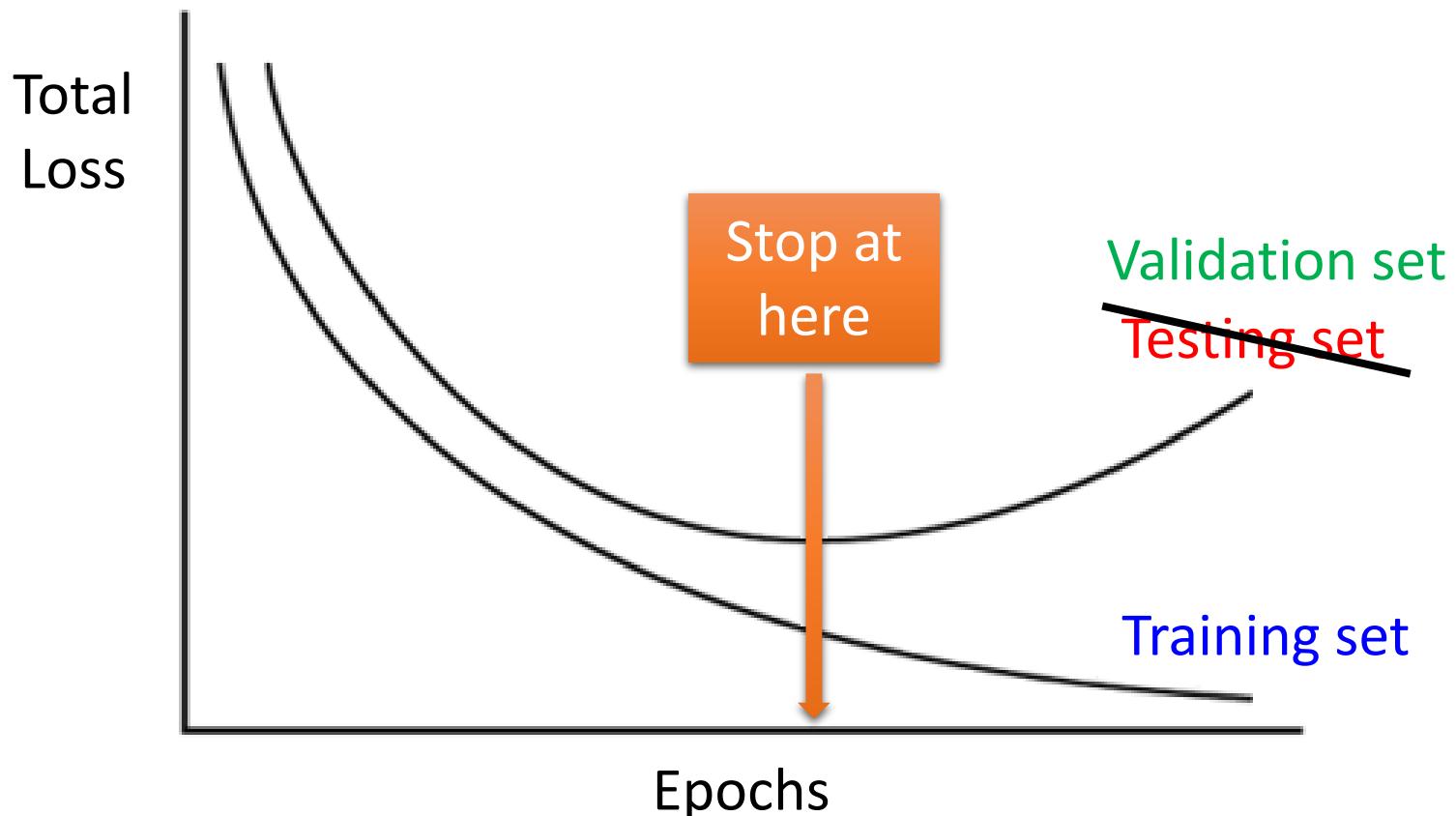
Testing:	Accuracy
Clean	0.97
Noisy	0.50

Training is not influenced.

# Recipe of Deep Learning

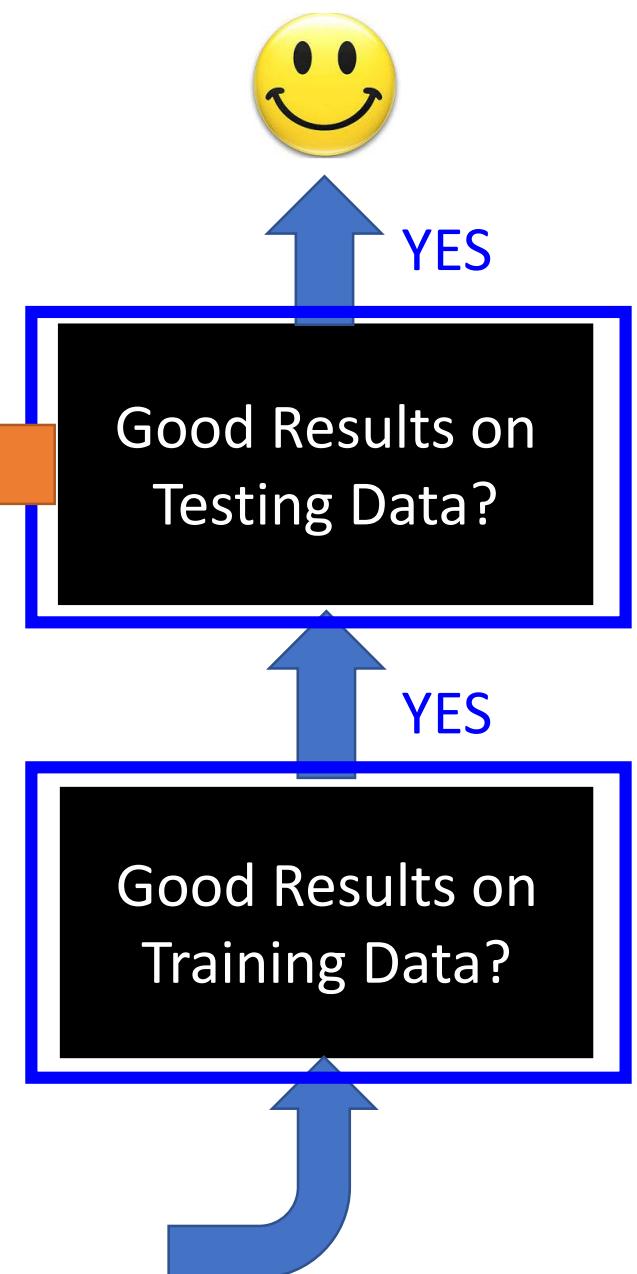
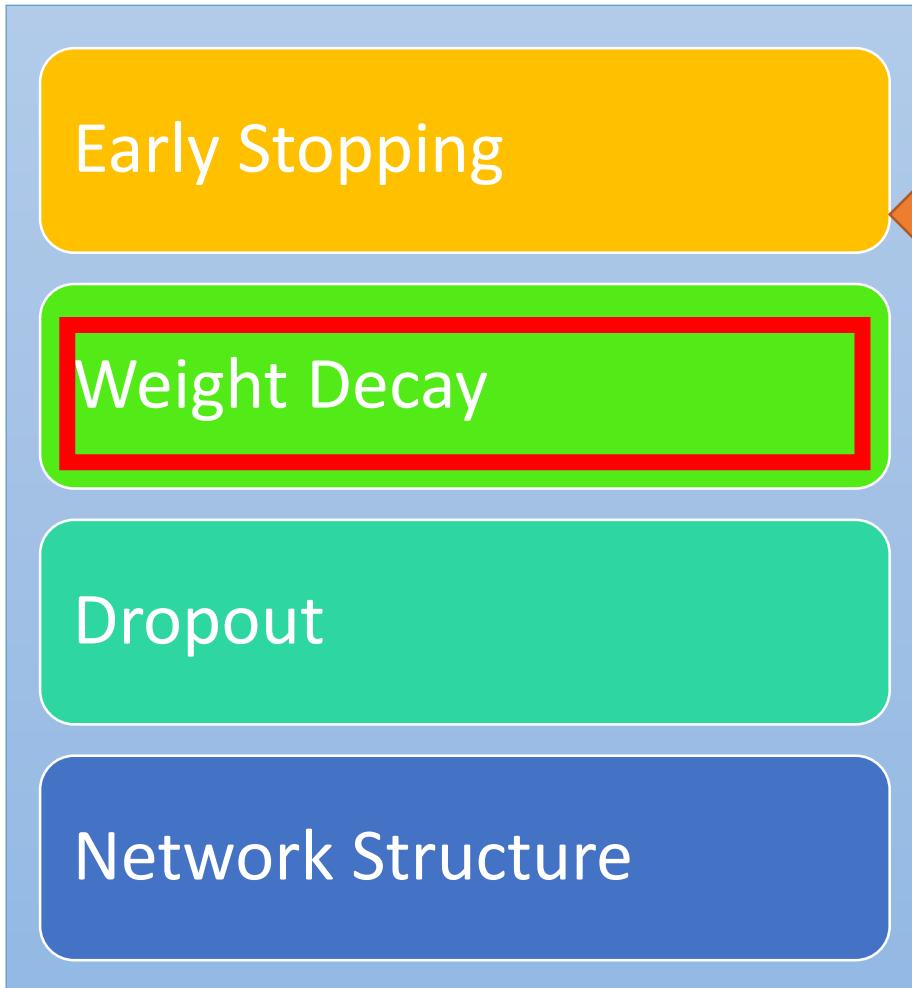


# Early Stopping



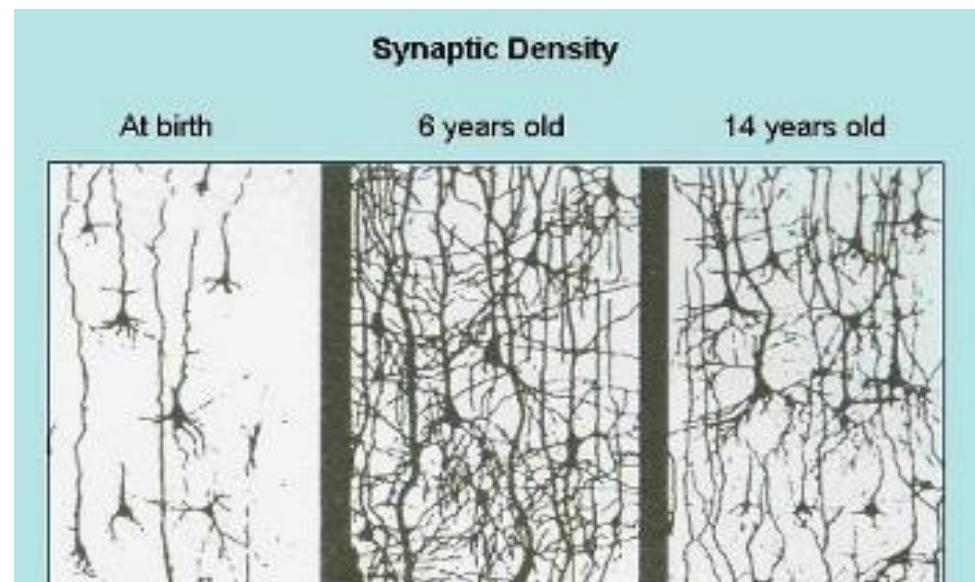
Keras: <http://keras.io/getting-started/faq/#how-can-i-interrupt-training-when-the-validation-loss-isnt-decreasing-anymore>

# Recipe of Deep Learning

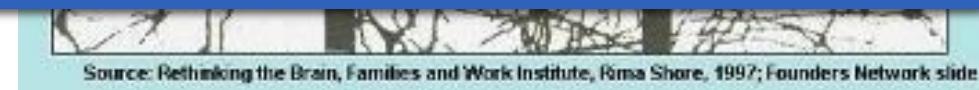


# Weight Decay

- Our brain prunes out the useless link between neurons.

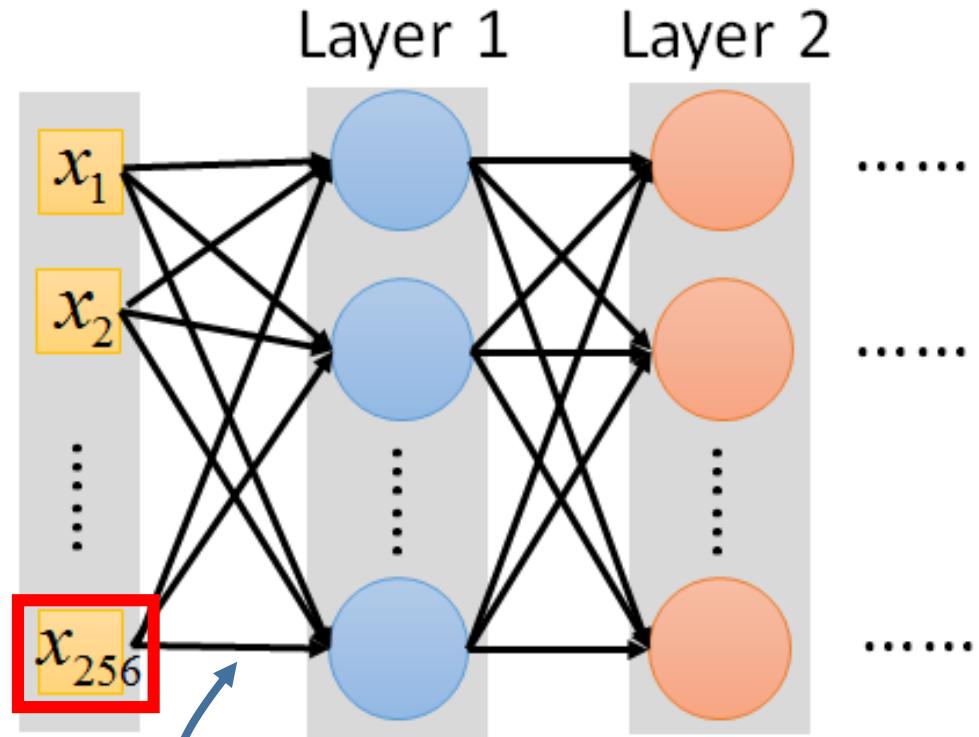
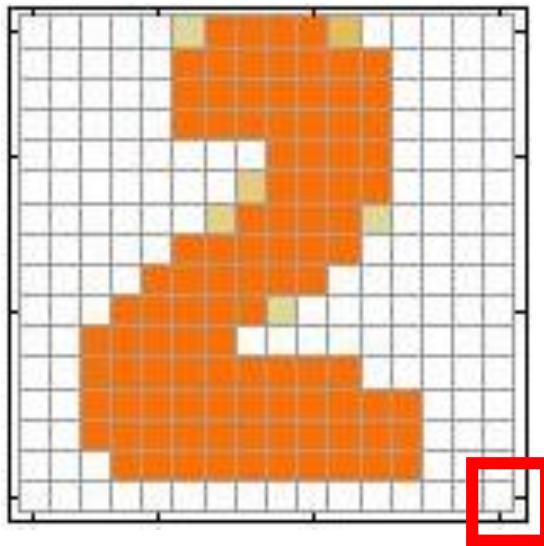


Doing the same thing to machine's brain improves the performance.



Source: Rethinking the Brain, Families and Work Institute, Rita Shore, 1997; Founders Network slide

# Weight Decay



Weight decay is one kind of regularization

Useless

Close to zero (萎缩了)

# Weight Decay

- Implementation

$$\text{Original: } w \leftarrow w - \eta \frac{\partial L}{\partial w}$$

$$\lambda = 0.01$$

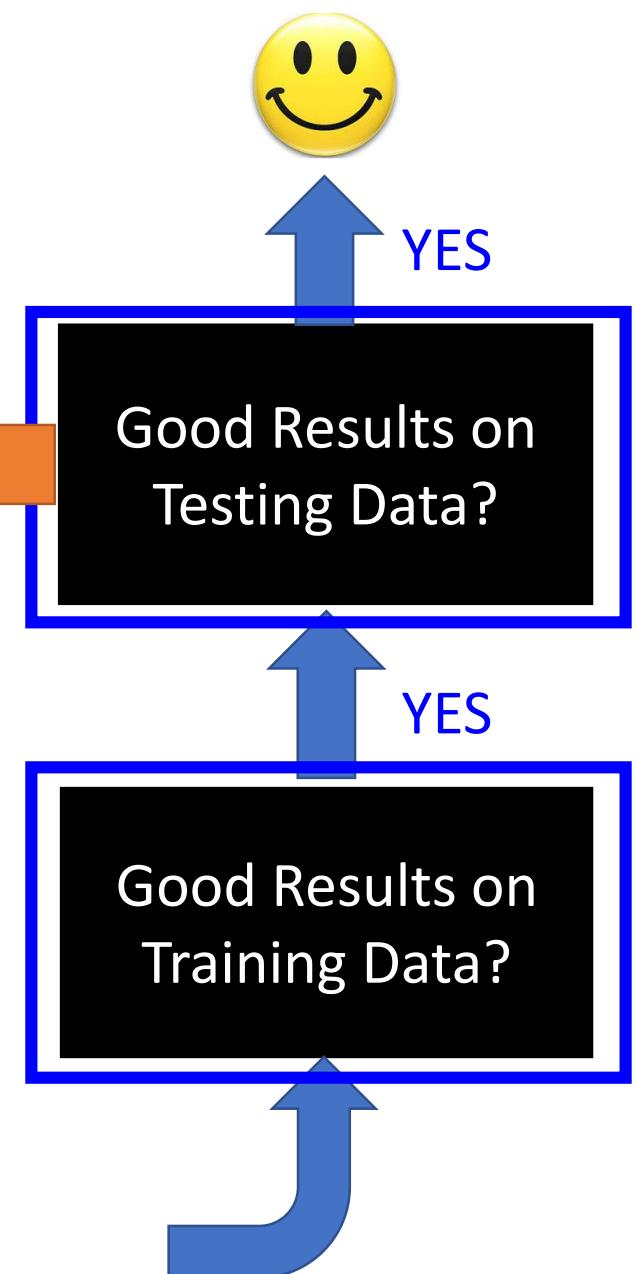
Weight Decay:

$$w \leftarrow \underbrace{0.99}_{\downarrow} w - \eta \frac{\partial L}{\partial w}$$

Smaller and smaller

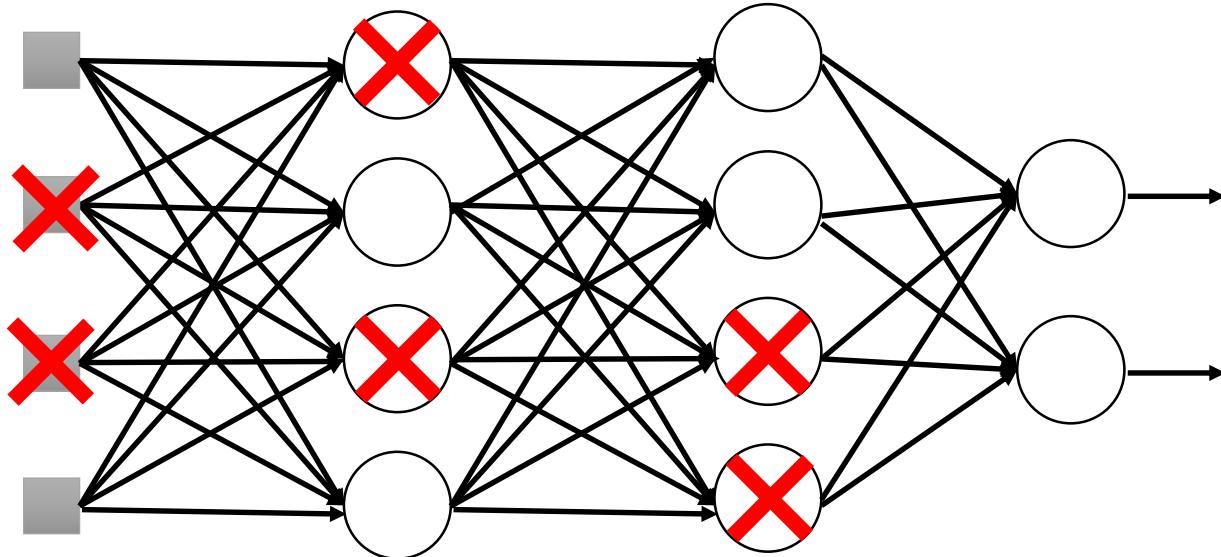
Keras: <http://keras.io/regularizers/>

# Recipe of Deep Learning



# Dropout

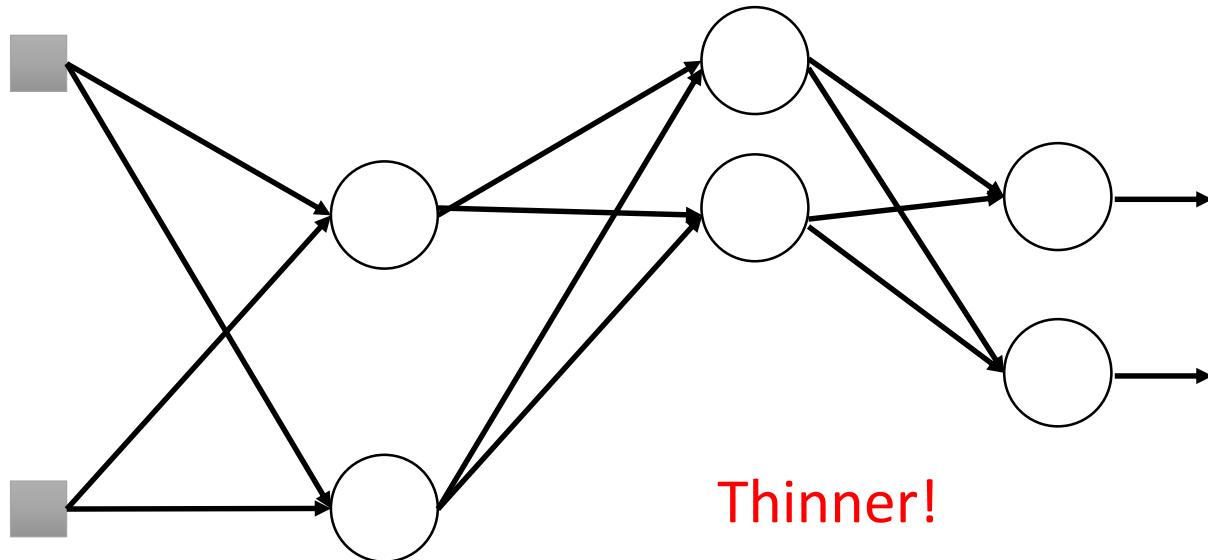
## Training:



- **Each time before updating the parameters**
  - Each neuron has  $p\%$  to dropout

# Dropout

## Training:

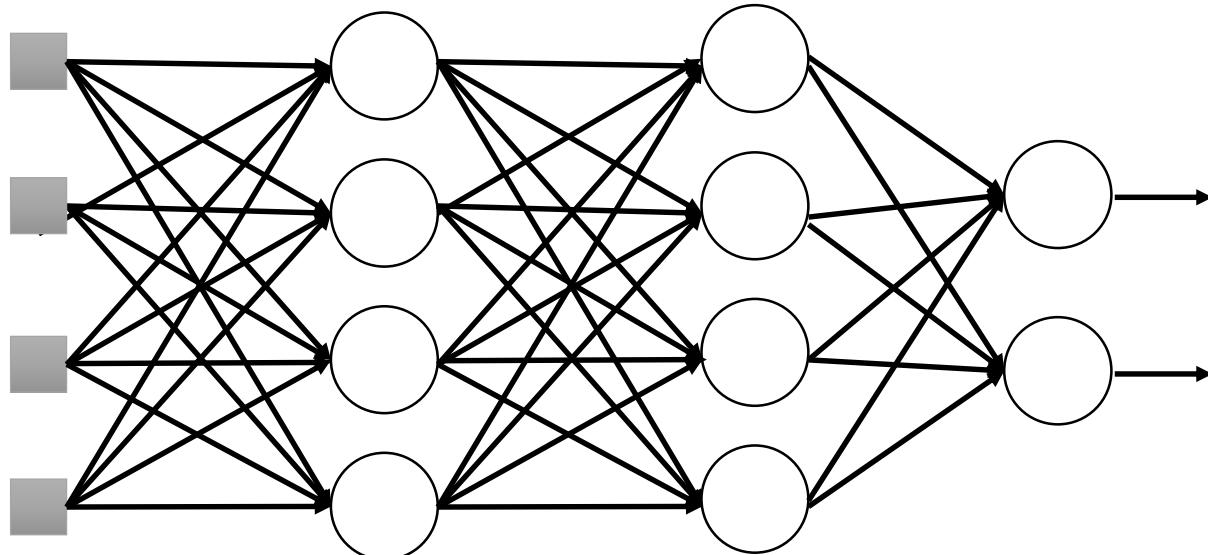


- **Each time before updating the parameters**
  - Each neuron has  $p\%$  to dropout
    - ➡ **The structure of the network is changed.**
  - Using the new network for training

For each mini-batch, we resample the dropout neurons

# Dropout

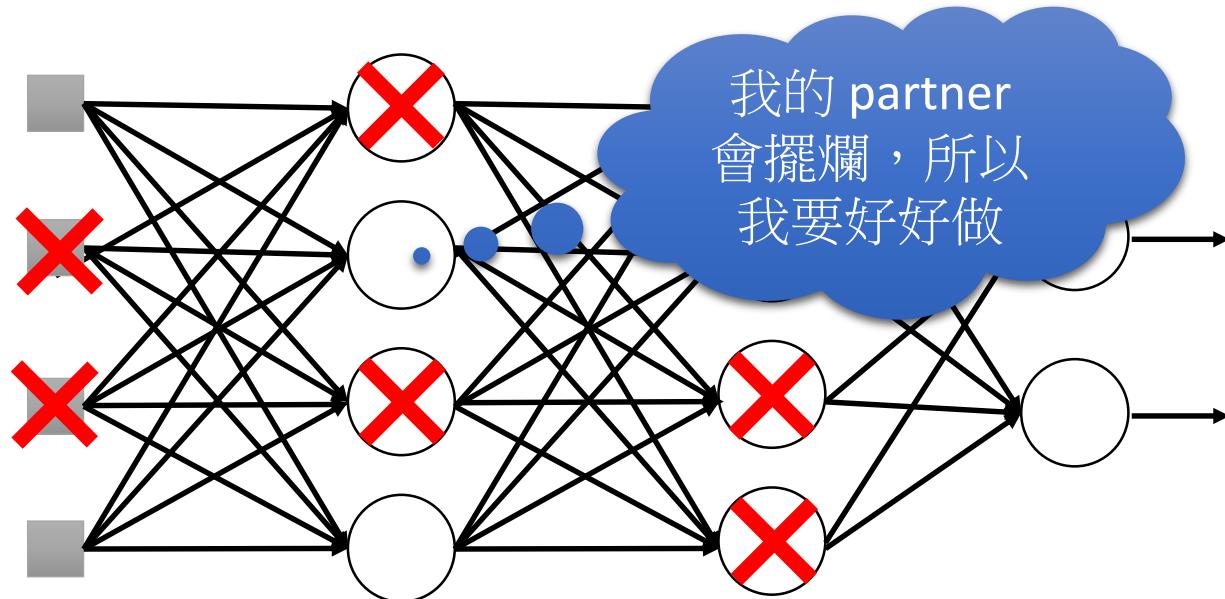
## Testing:



### ➤ No dropout

- If the dropout rate at training is  $p\%$ ,  
all the weights times  $(1-p)\%$
- Assume that the dropout rate is 50%.  
If a weight  $w = 1$  by training, set  $w = 0.5$  for testing.

# Dropout - Intuitive Reason



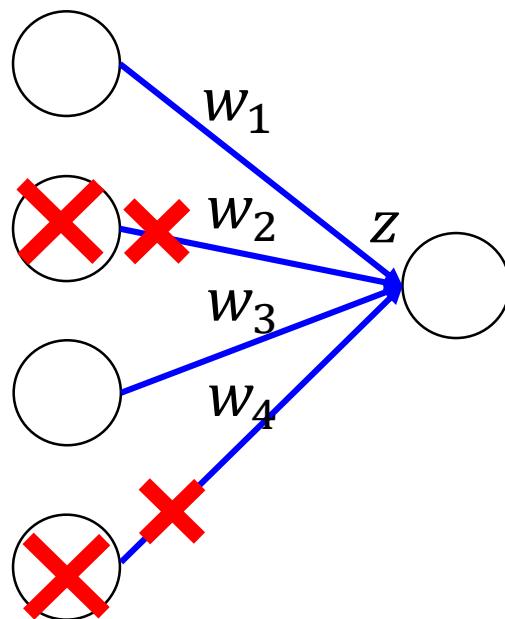
- When teams up, if everyone expect the partner will do the work, nothing will be done finally.
- However, if you know your partner will dropout, you will do better.
- When testing, no one dropout actually, so obtaining good results eventually.

# Dropout - Intuitive Reason

- Why the weights should multiply  $(1-p)\%$  (dropout rate) when testing?

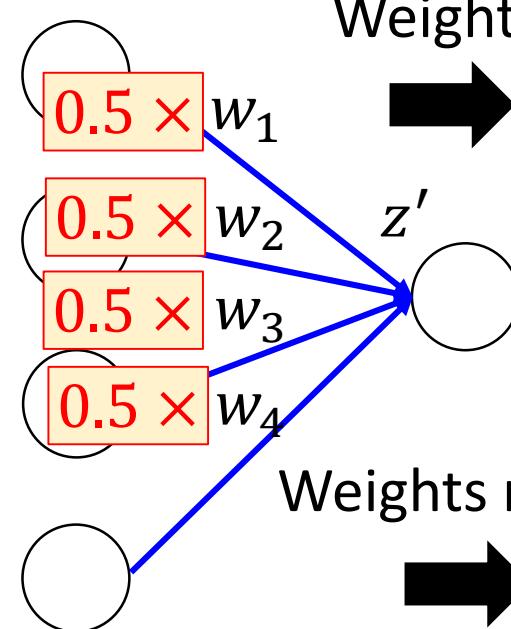
## Training of Dropout

Assume dropout rate is 50%



## Testing of Dropout

No dropout



Weights from training

$$0.5 \times w_1 \rightarrow z' \approx 2z$$

$$0.5 \times w_2 \rightarrow z'$$

$$0.5 \times w_3 \rightarrow z'$$

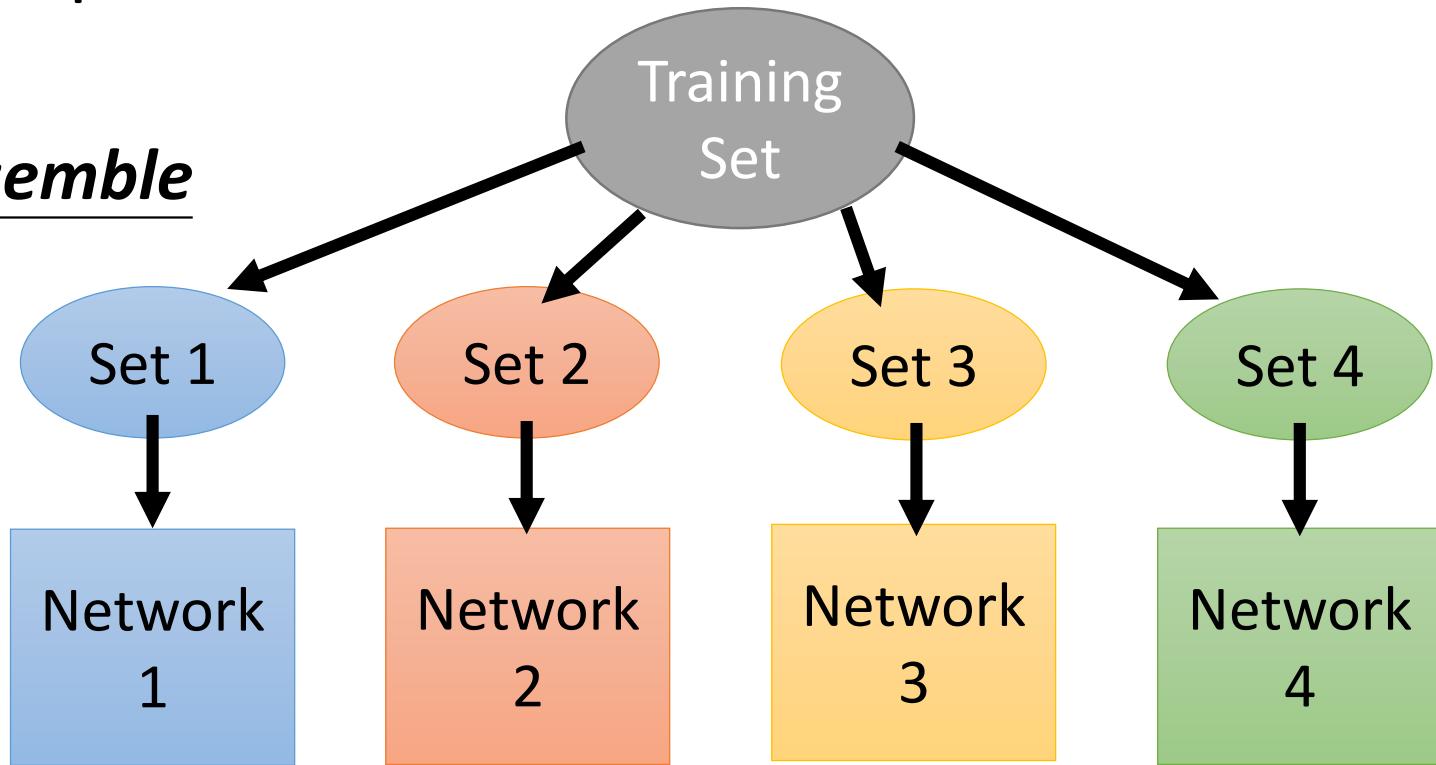
$$0.5 \times w_4 \rightarrow z'$$

Weights multiply  $(1-p)\%$

$$\rightarrow z' \approx z$$

# Dropout is a kind of ensemble.

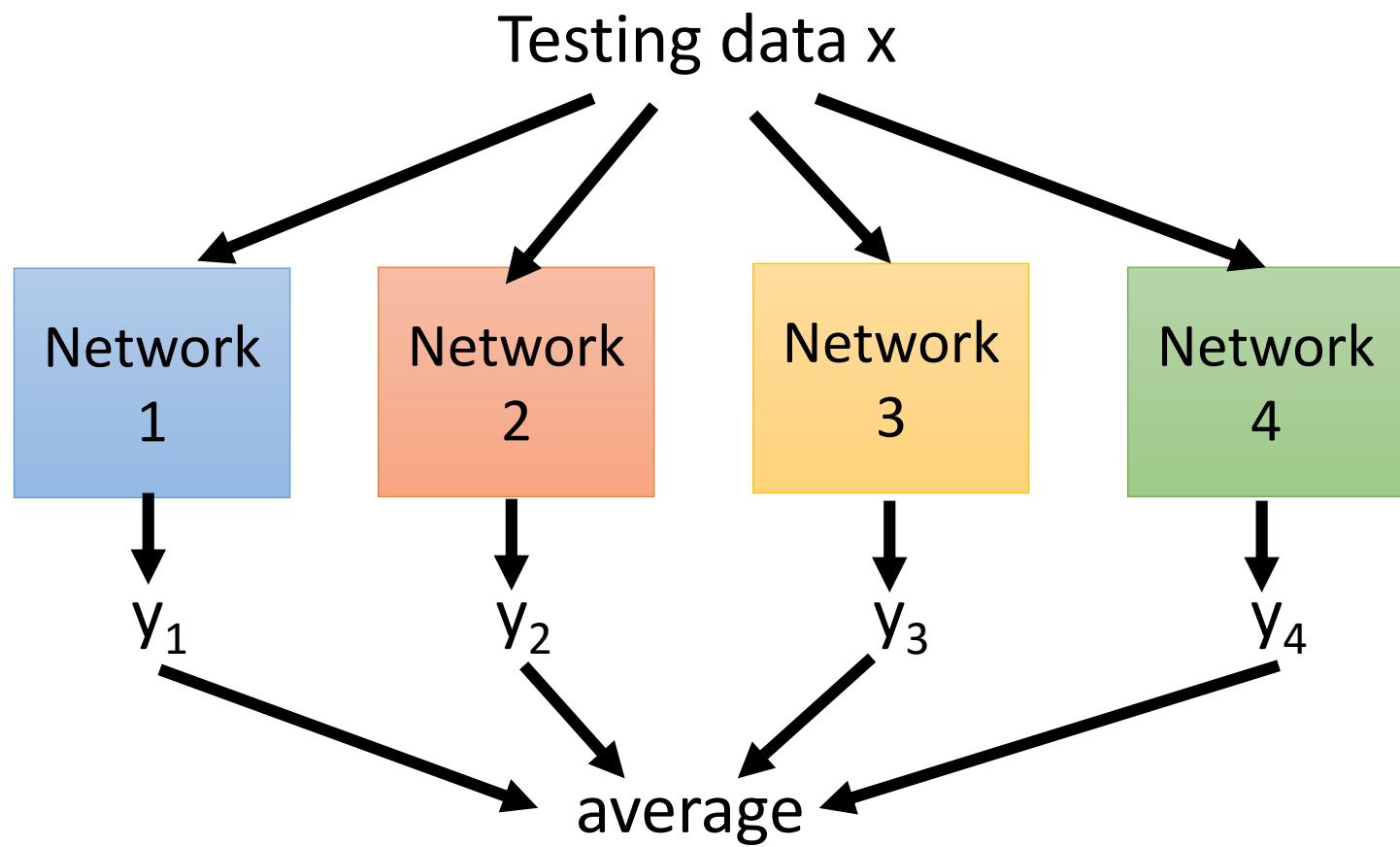
## Ensemble



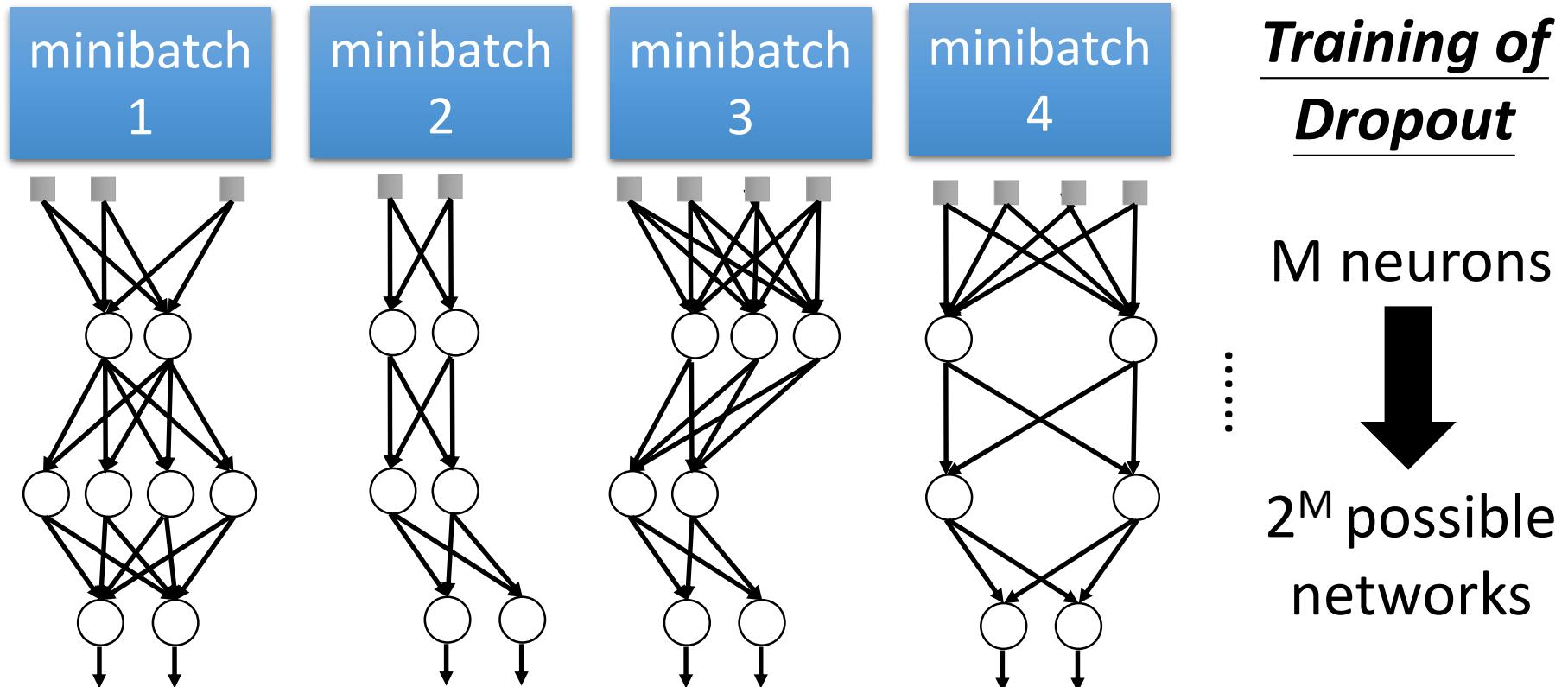
Train a bunch of networks with different structures

# Dropout is a kind of ensemble.

## Ensemble



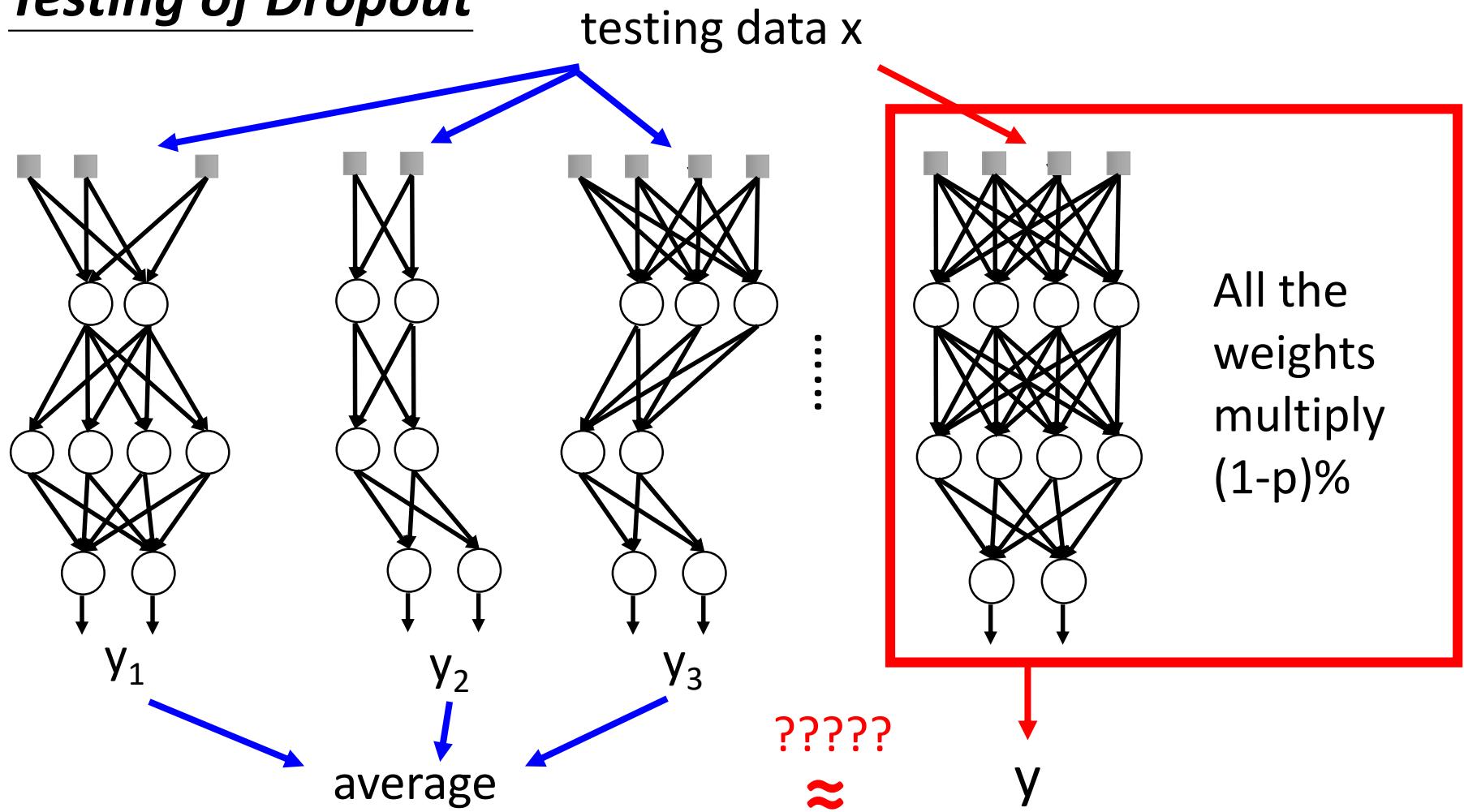
# Dropout is a kind of ensemble.



- Using one mini-batch to train one network
- Some parameters in the network are shared

# Dropout is a kind of ensemble.

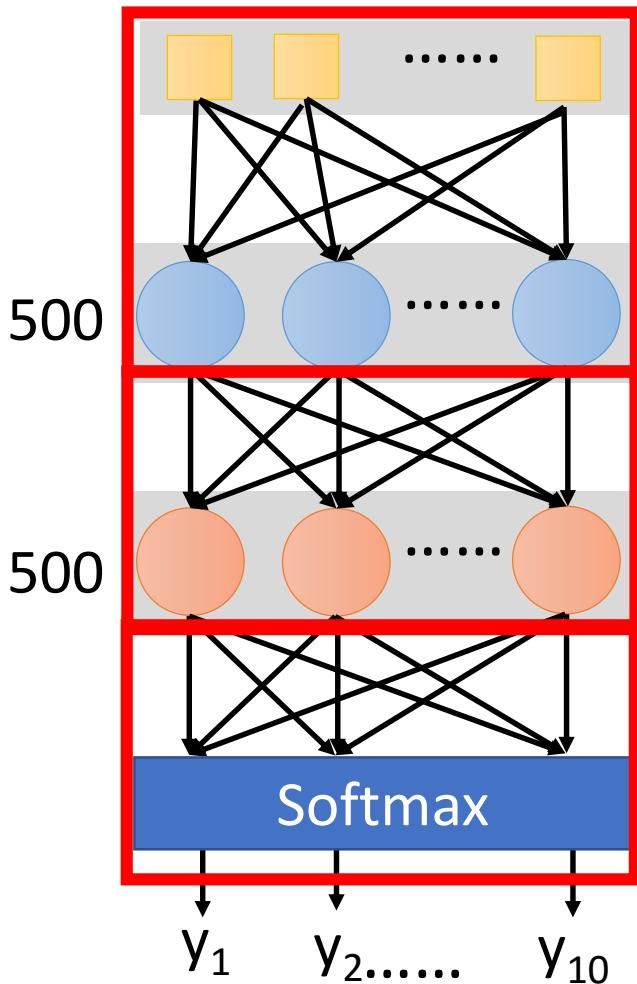
## Testing of Dropout



# More about dropout

- More reference for dropout [Nitish Srivastava, JMLR'14] [Pierre Baldi, NIPS'13][Geoffrey E. Hinton, arXiv'12]
- Dropout works better with Maxout [Ian J. Goodfellow, ICML'13]
- Dropconnect [Li Wan, ICML'13]
  - Dropout delete neurons
  - Dropconnect deletes the connection between neurons
- Annealed dropout [S.J. Rennie, SLT'14]
  - Dropout rate decreases by epochs
- Standout [J. Ba, NISP'13]
  - Each neural has different dropout rate

# Let's try it



```
model = Sequential()
```

```
model.add( Dense( input_dim=28*28,  
                  output_dim=500 ) )  
model.add( Activation('sigmoid') )
```

**model.add( dropout(0.8) )**

```
model.add( Dense( output_dim=500 ) )  
model.add( Activation('sigmoid') )
```

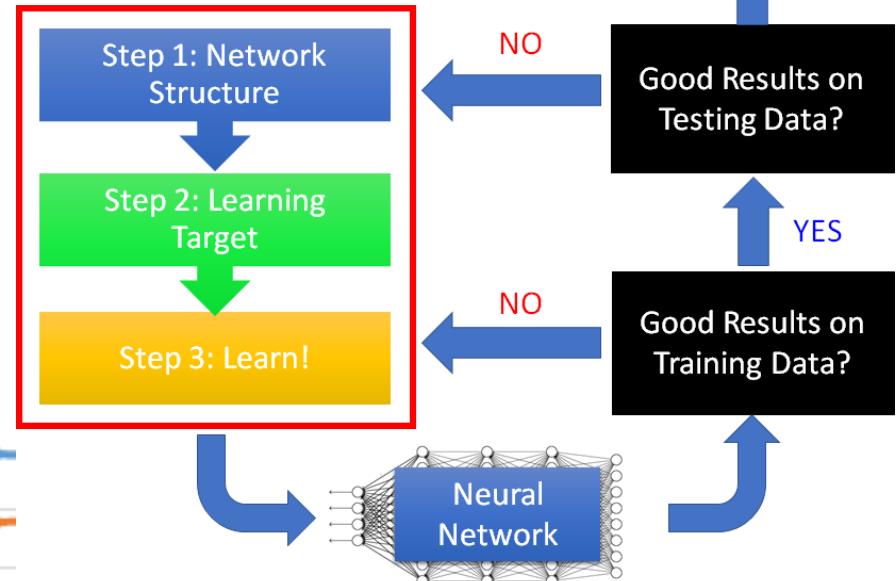
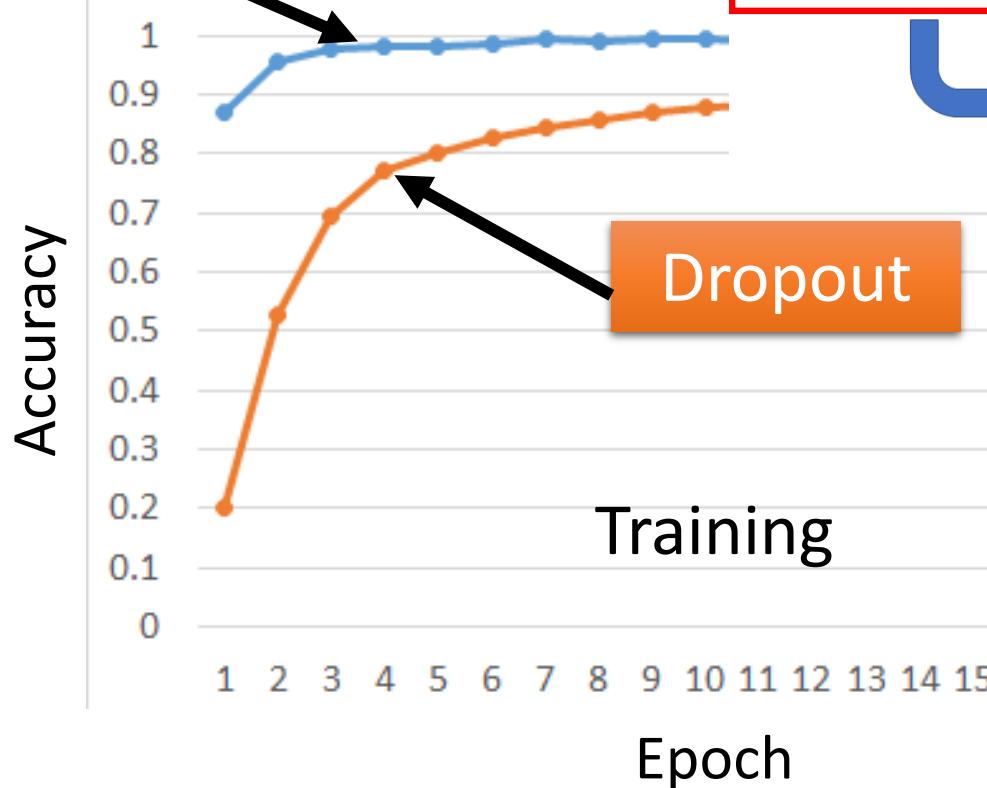
**model.add( dropout(0.8) )**

```
model.add( Dense(output_dim=10) )  
model.add( Activation('softmax') )
```

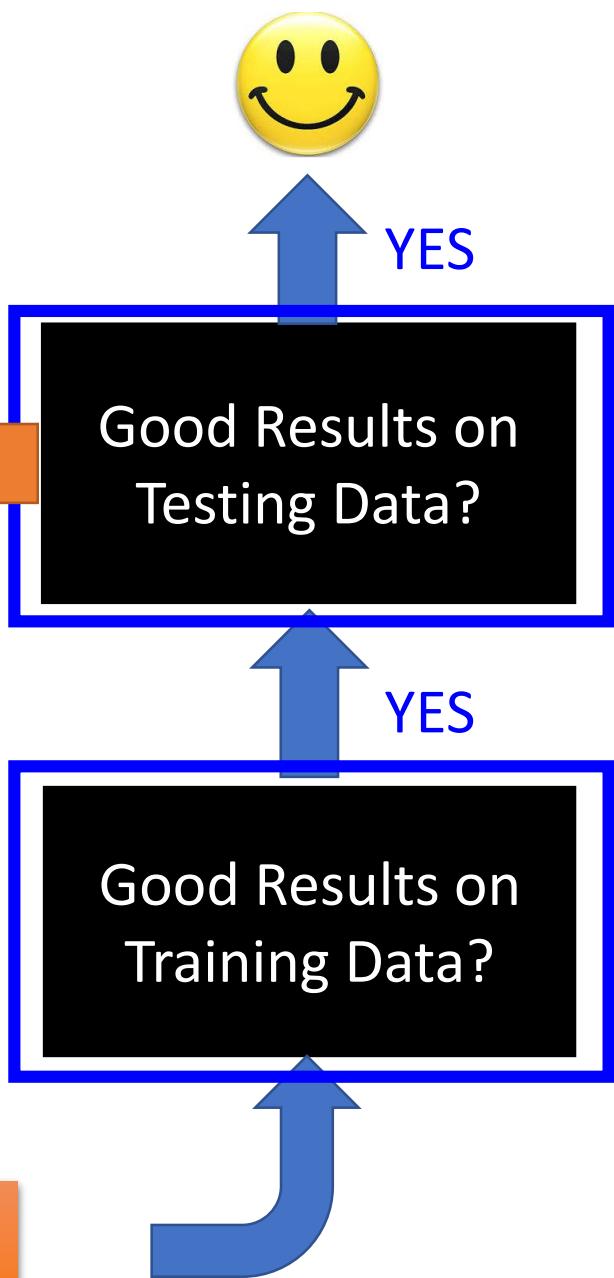
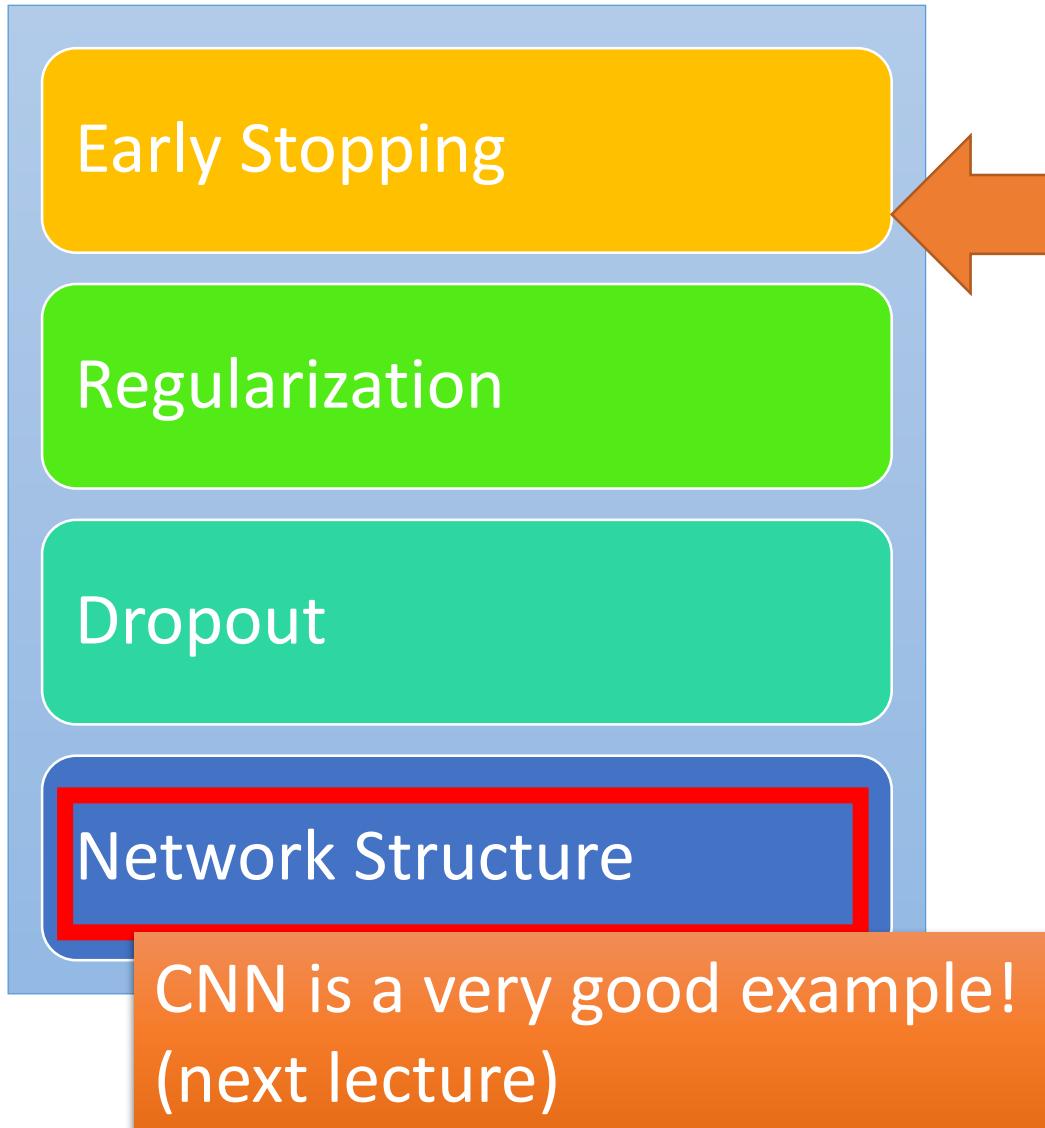


# Let's try it

No Dropout

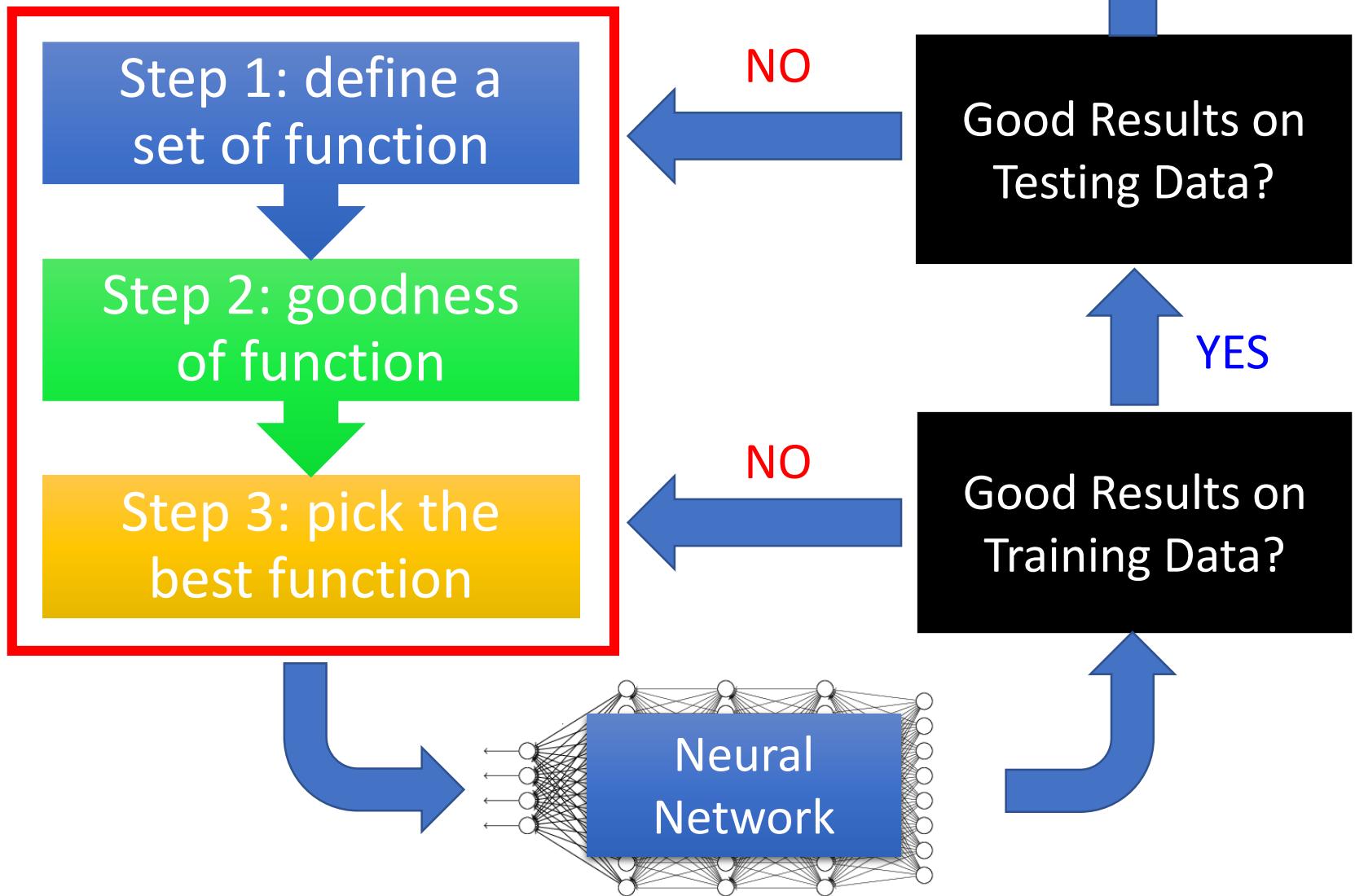


# Recipe of Deep Learning



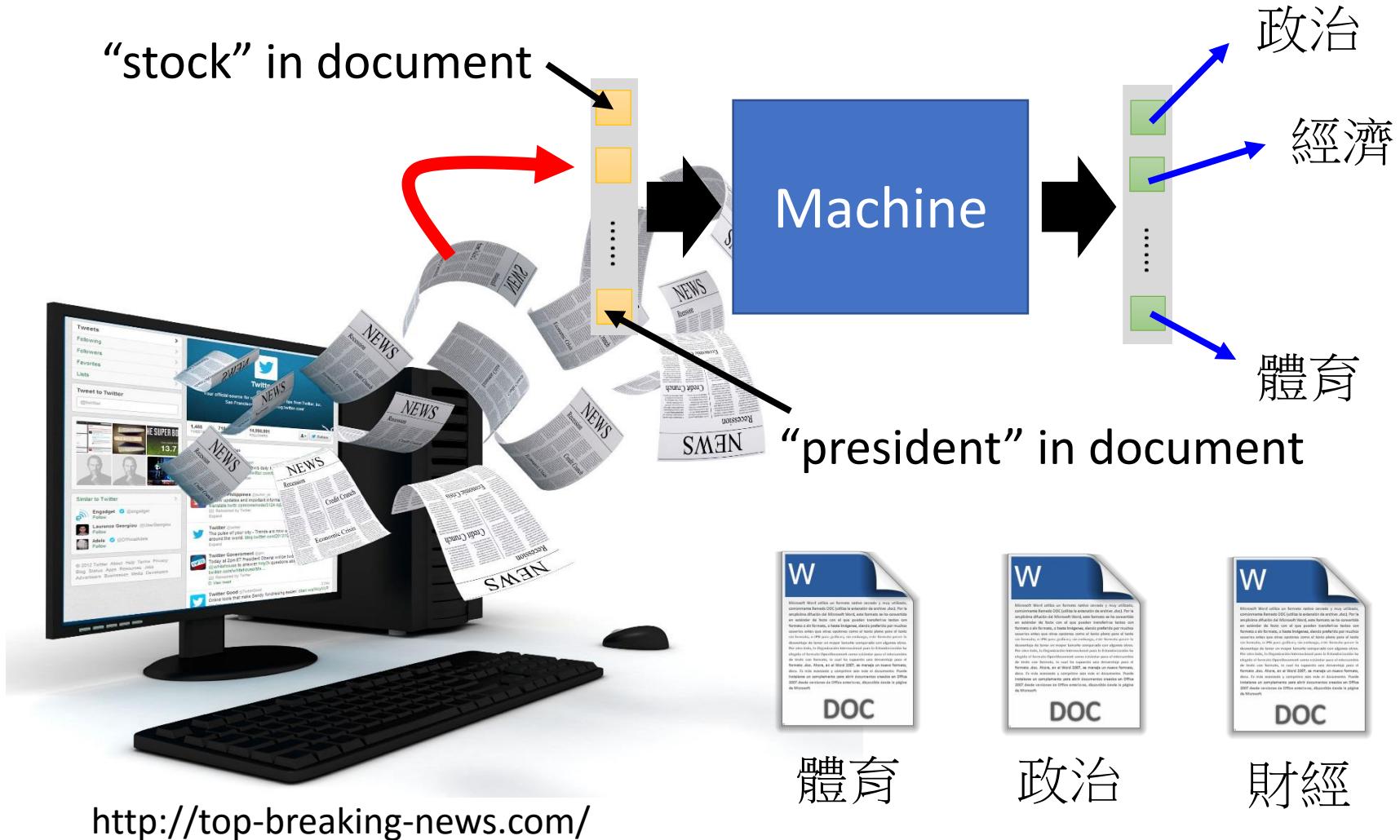
# Concluding Remarks of Lecture II

# Recipe of Deep Learning



Let's try another task

# Document Classification



# Data

```
In [8]: x_train.shape  
Out[8]: (8982, 1000)
```

```
In [9]: y_train.shape  
Out[9]: (8982, 46)
```

```
In [12]: x_train[0]
```

Out[12]:

```
array([[ 0.,  1.,  1.,  0.,  1.,  1.,  1.,  1.,  1.], [ 0.,  0.,  1.,  1.,  1.,  0.,  1.,  0.,  0.], [ 1.,  0.,  0.,  1.,  1.,  0.,  1.,  0.,  0.], [ 1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.], [ 1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.], [ 0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.], [ 0.,  0.,  0.,  0.,  0.,  1.,  1.,  0.,  0.], [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.], [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.], [ 0.,  0.,  0.,  0.,  0.,  1.,  0.,  1.,  0.], [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.], [ 0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.], [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.], [ 0.,  0.,  1.,  0.,  1.,  0.,  0.,  0.,  0.], [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.], [ 0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.], [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.], [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.], [ 0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.], [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.], [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.]])
```

```
In [10]: x_test.shape
```

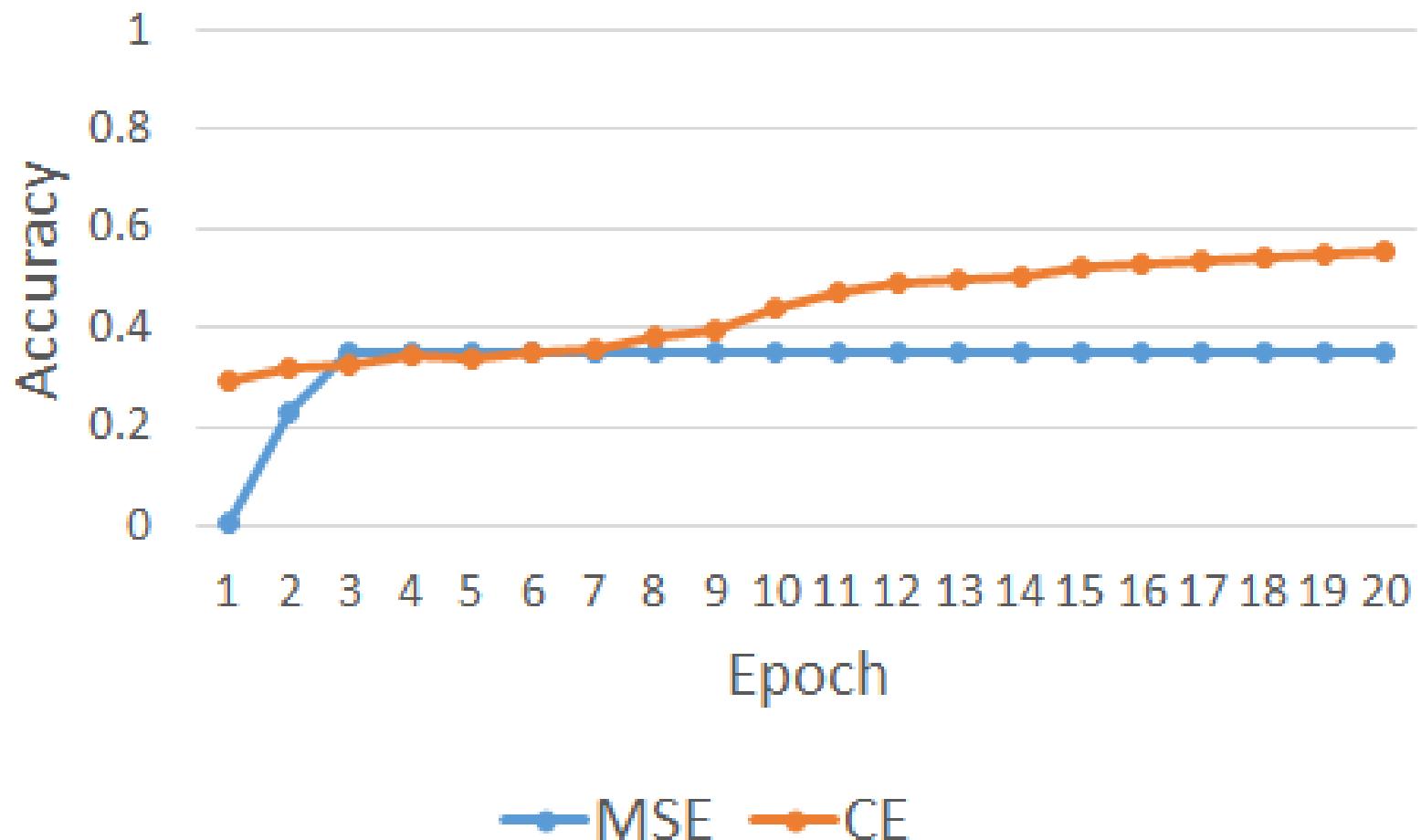
```
In [11]: y_test.shape  
Out[11]: (2246, 46)
```

```
In [13]: y_train[0]
```

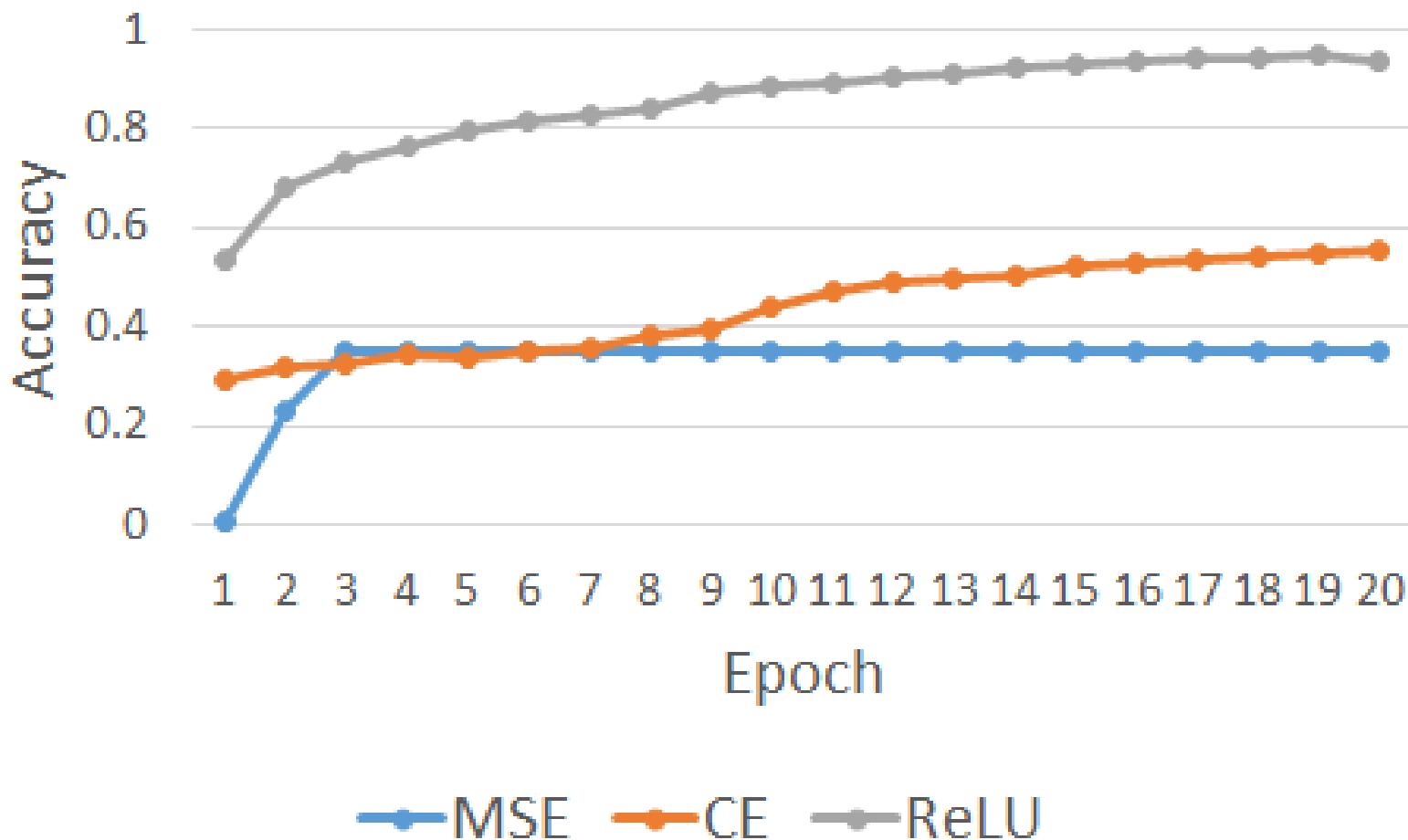
[ 1 ]

```
array([ 0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  
      0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  
      0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  
      0.,  0.,  0.,  0.,  0.,  0.])
```

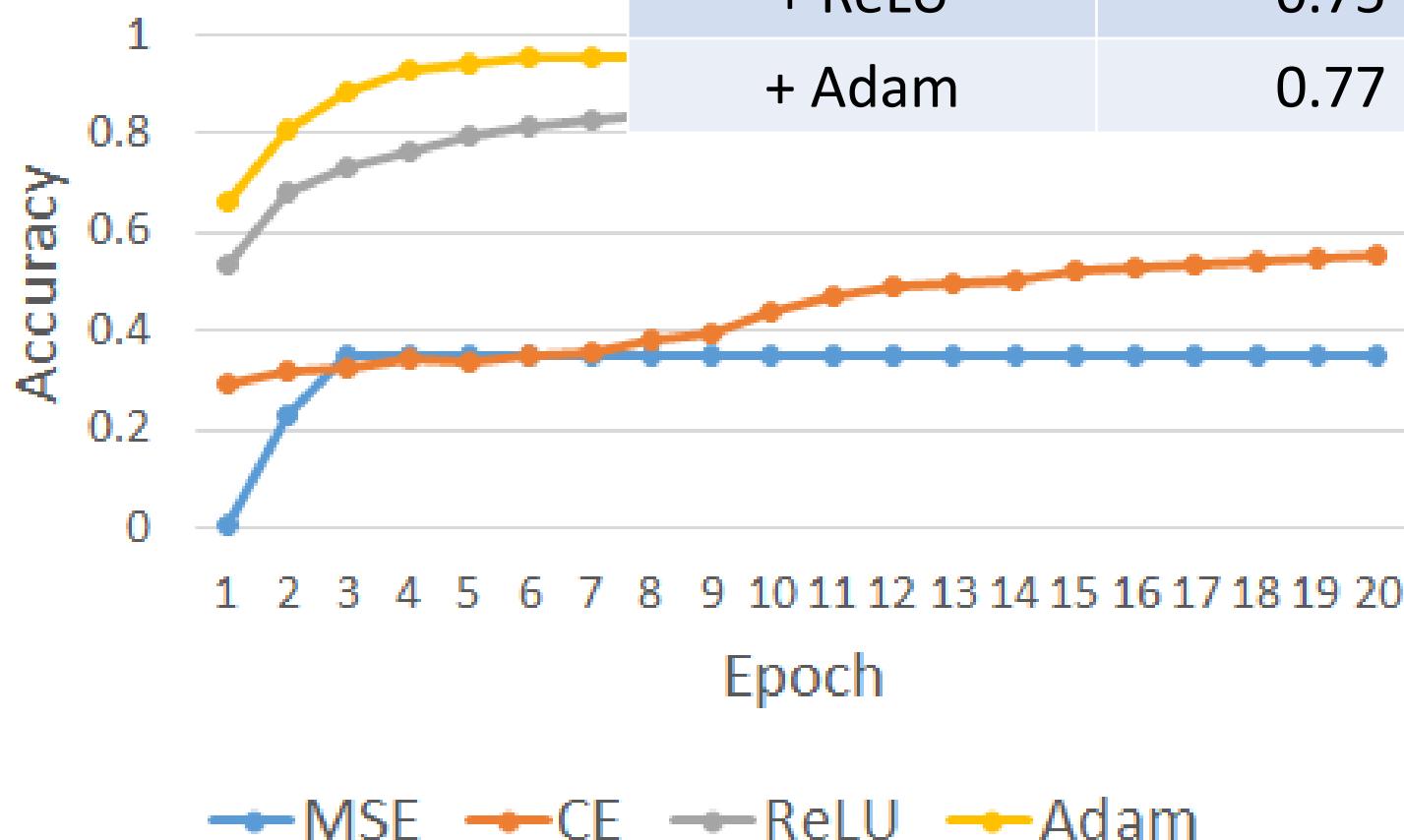
# MSE



# ReLU

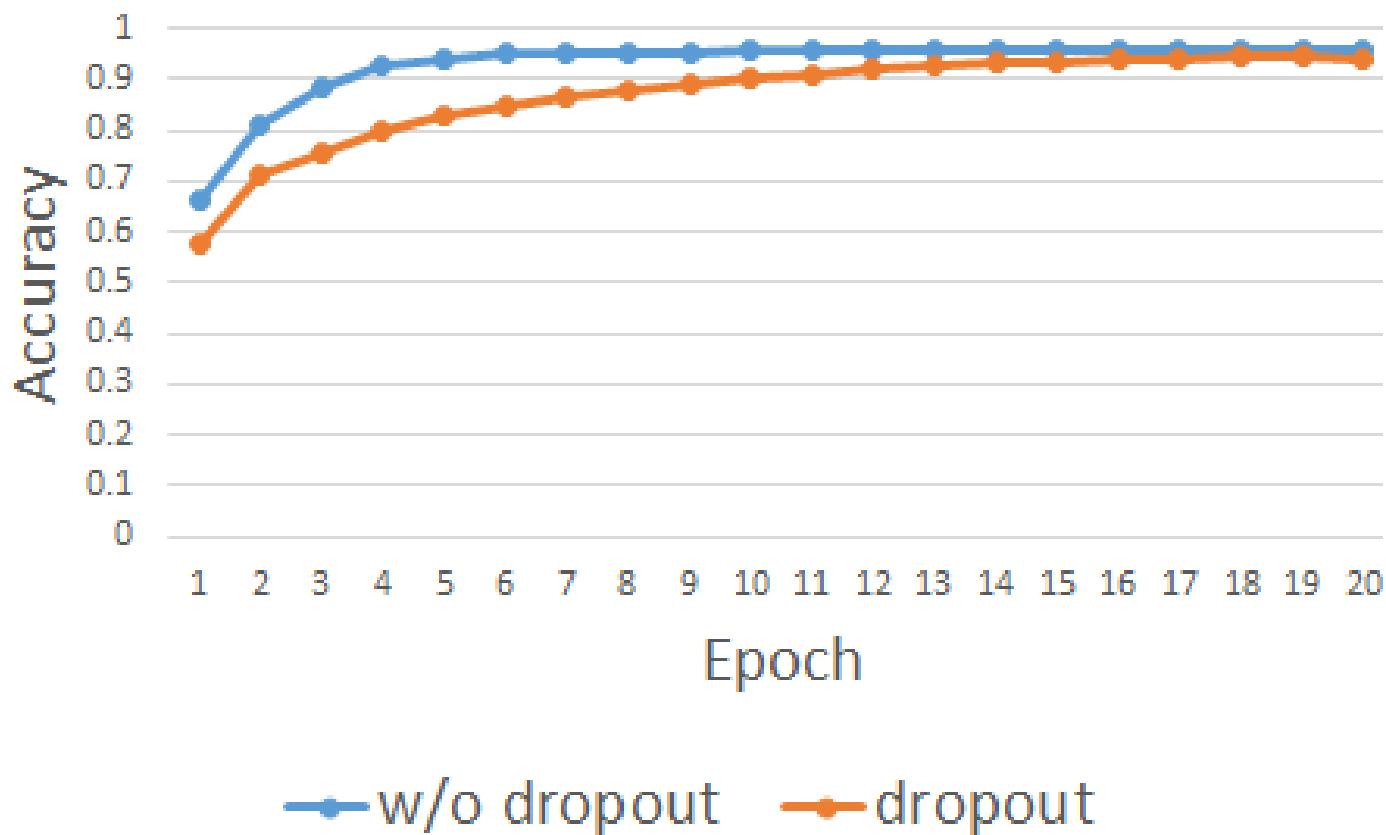


# Adaptive Learn



# Dropout

	Accuracy
Adam	0.77
+ dropout	0.79



# Lecture III:

## Variants of Neural Networks

# Variants of Neural Networks

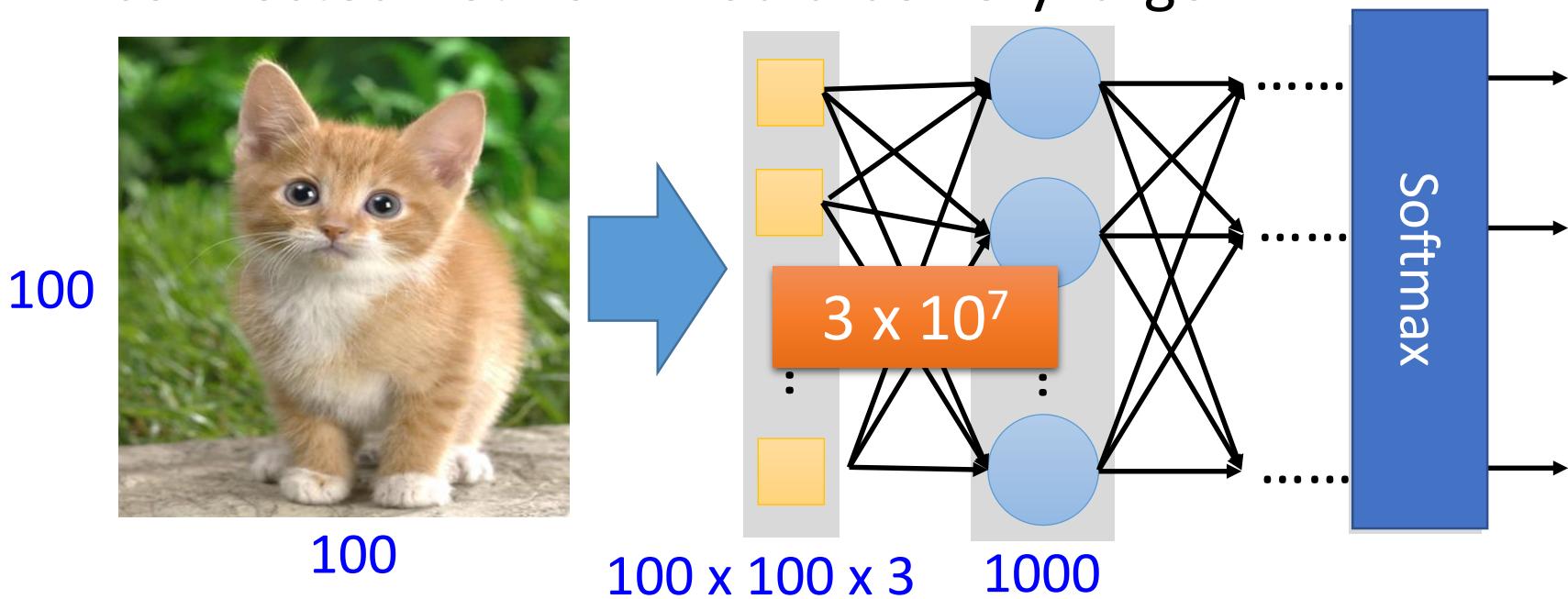
Convolutional Neural  
Network (CNN)

Widely used in  
image processing

Recurrent Neural Network  
(RNN)

# Why CNN for Image?

- When processing image, the first layer of fully connected network would be very large



Can the fully connected network be simplified by considering the properties of image recognition?

# Why CNN for Image

- Some patterns are much smaller than the whole image

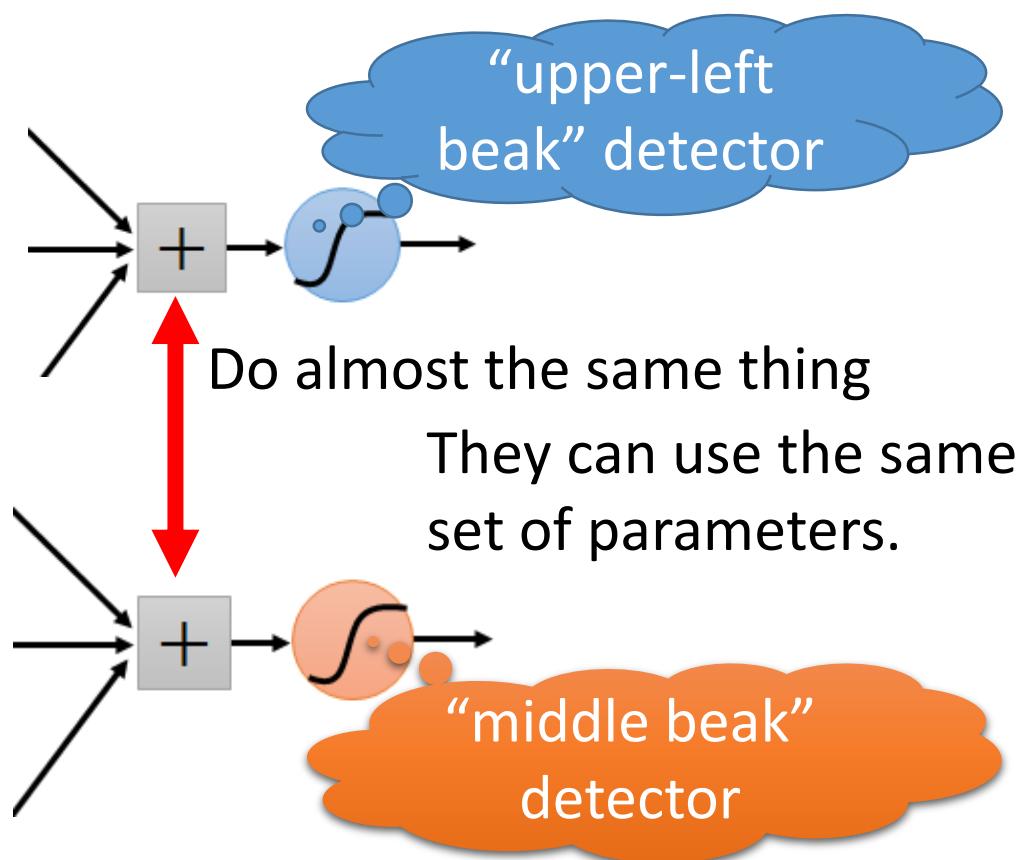
A neuron does not have to see the whole image to discover the pattern.

Connecting to small region with less parameters



# Why CNN for Image

- The same patterns appear in different regions.



# Why CNN for Image

- Subsampling the pixels will not change the object

bird



subsampling

bird



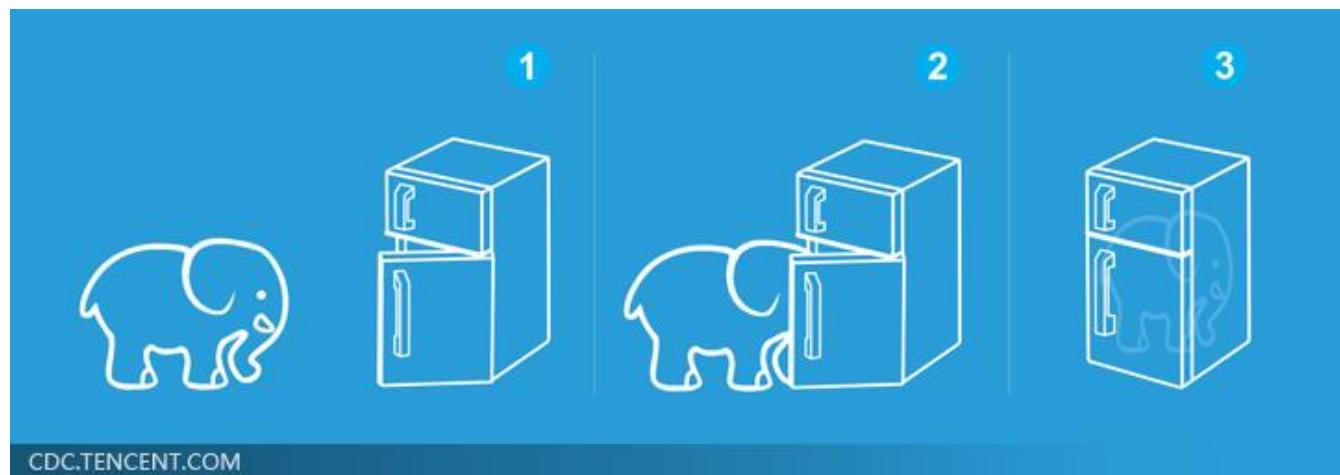
We can subsample the pixels to make image smaller

Less parameters for the network to process the image

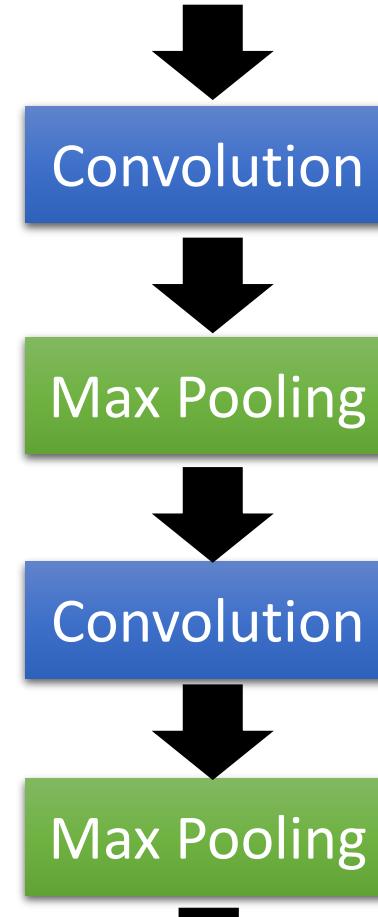
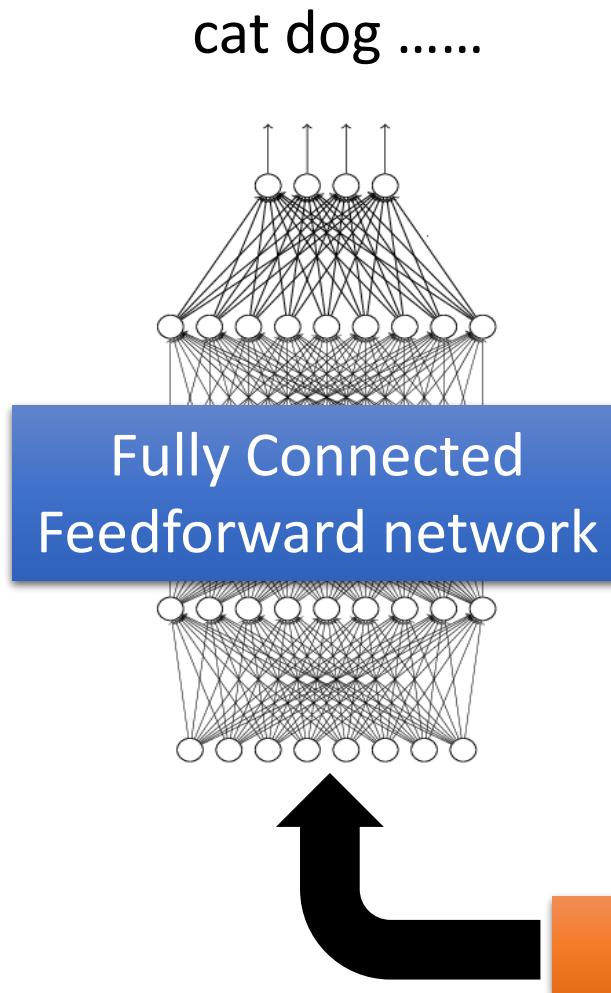
# Three Steps for Deep Learning



Deep Learning is so simple .....



# The whole CNN



Flatten

# The whole CNN

## Property 1

- Some patterns are much smaller than the whole image

## Property 2

- The same patterns appear in different regions.

## Property 3

- Subsampling the pixels will not change the object



Convolution

Max Pooling

Convolution

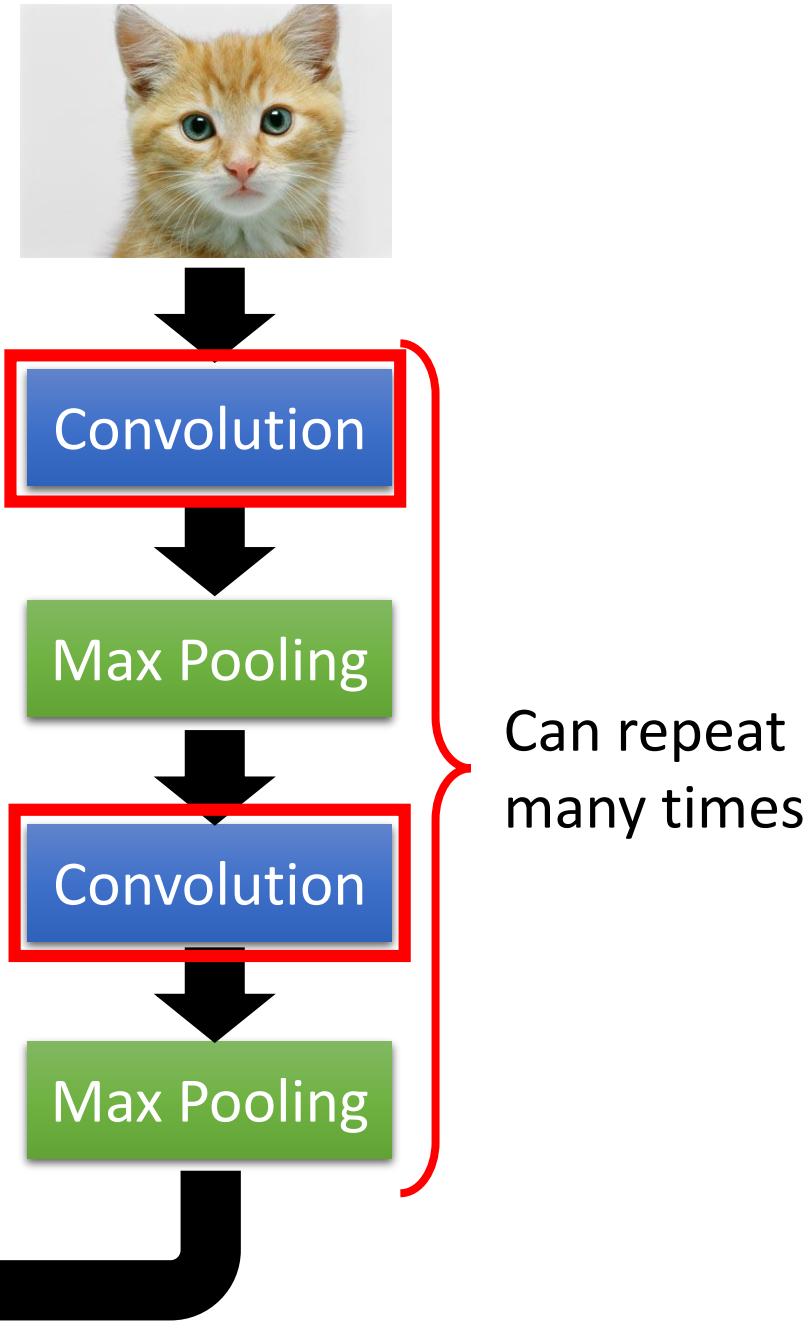
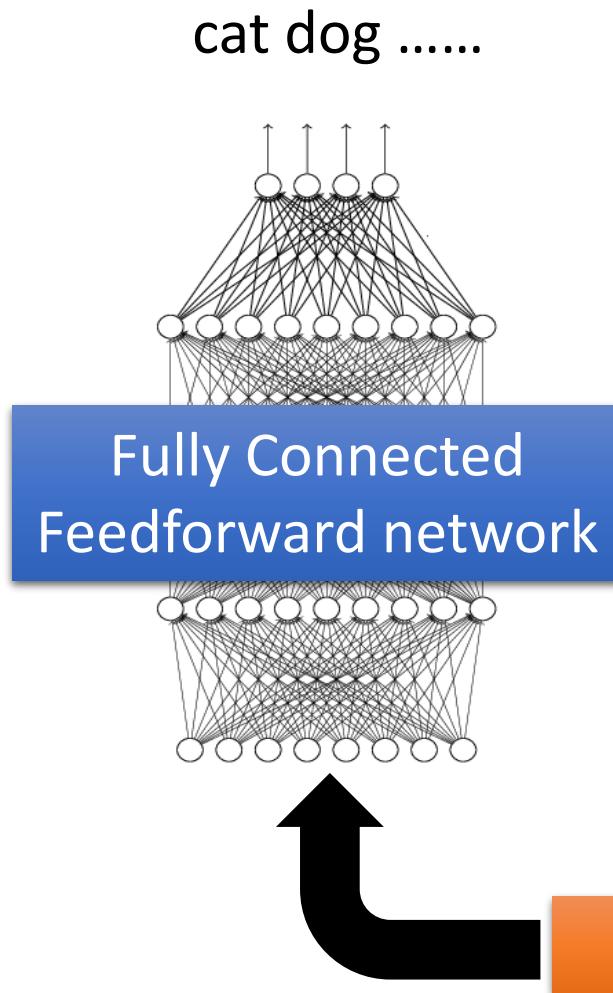
Max Pooling

Flatten



Can repeat  
many times

# The whole CNN



# CNN – Convolution

Those are the network parameters to be learned.

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1  
Matrix

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2  
Matrix

⋮

Property 1

Each filter detects a small pattern (3 x 3).

# CNN – Convolution

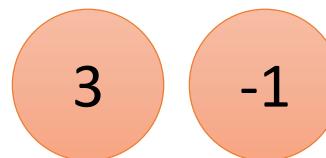
stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1



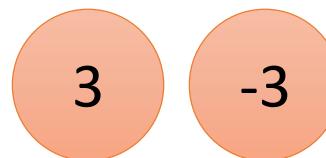
# CNN – Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

If stride=2

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

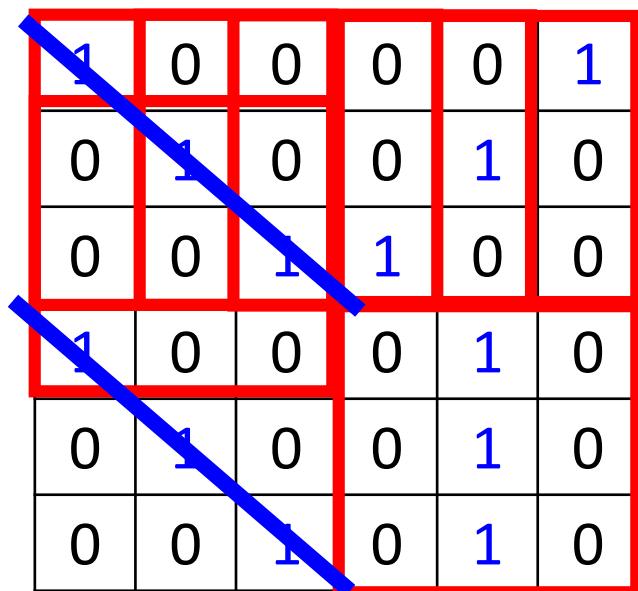


We set stride=1 below

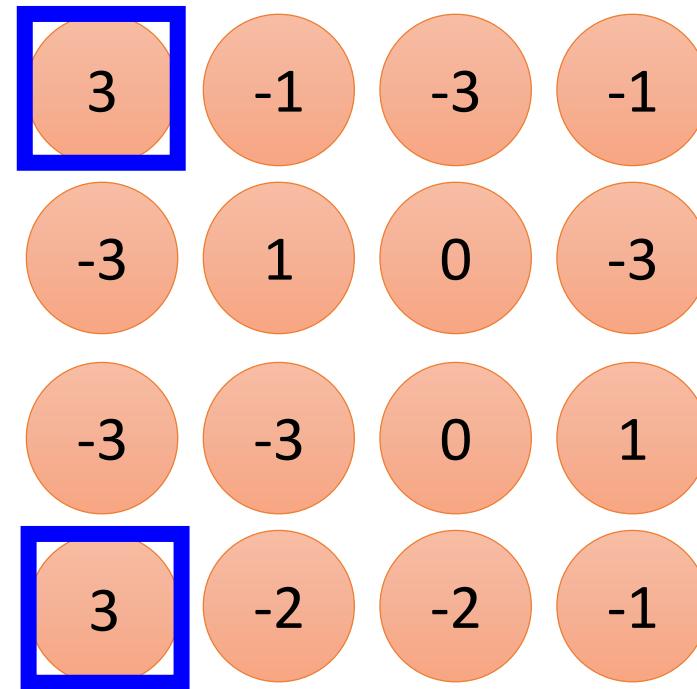
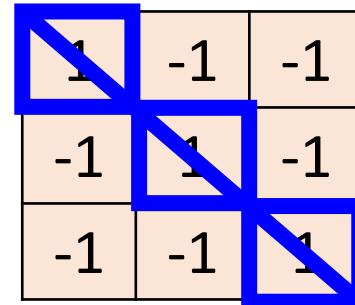
6 x 6 image

# CNN – Convolution

stride=1



6 x 6 image



# CNN – Convolution

stride=1

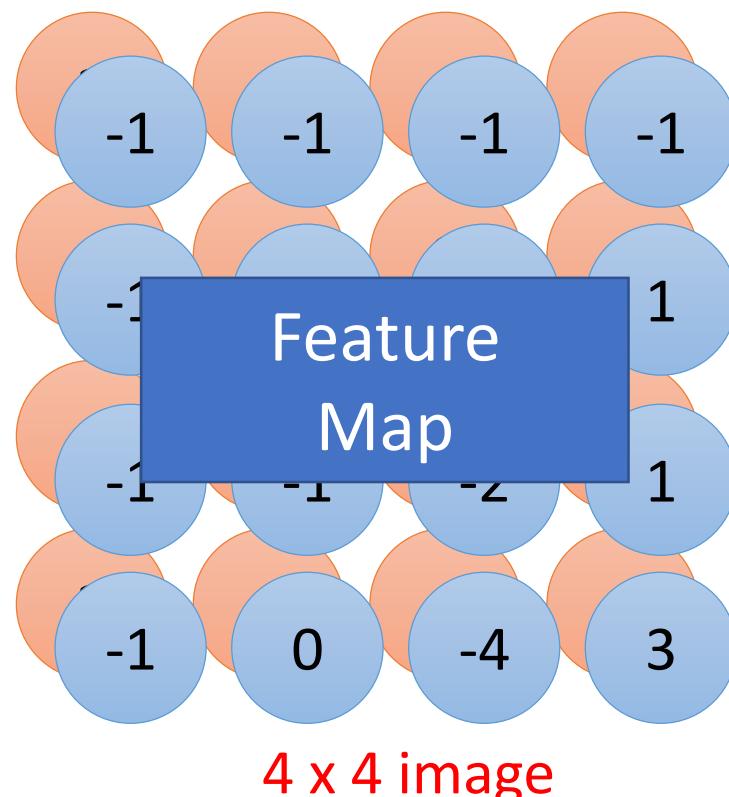
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

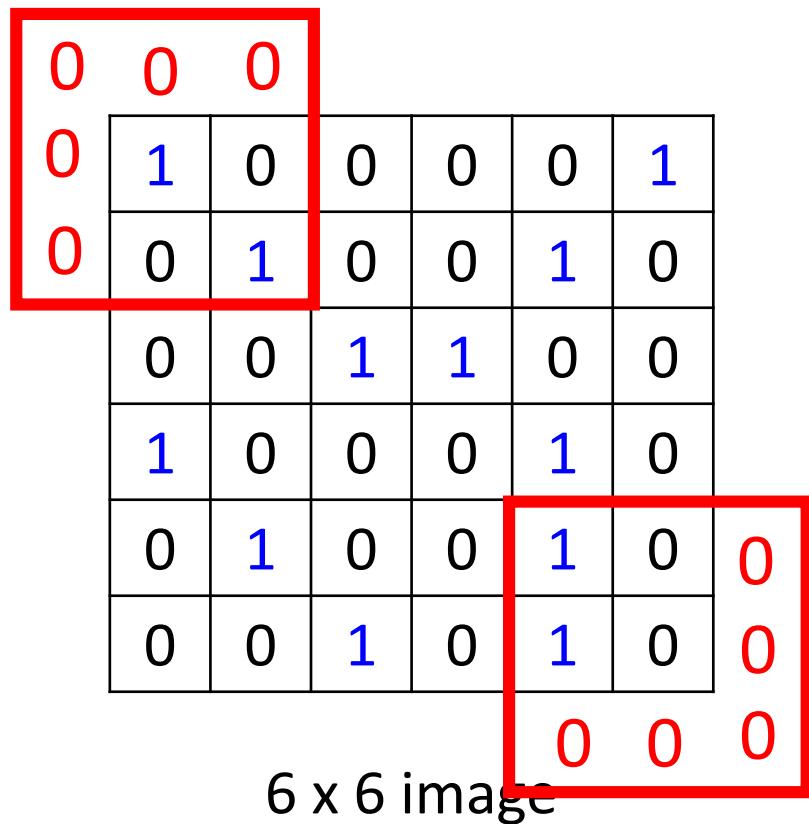
-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

Do the same process for every filter



# CNN – Zero Padding



1	-1	-1
-1	1	-1
-1	-1	1

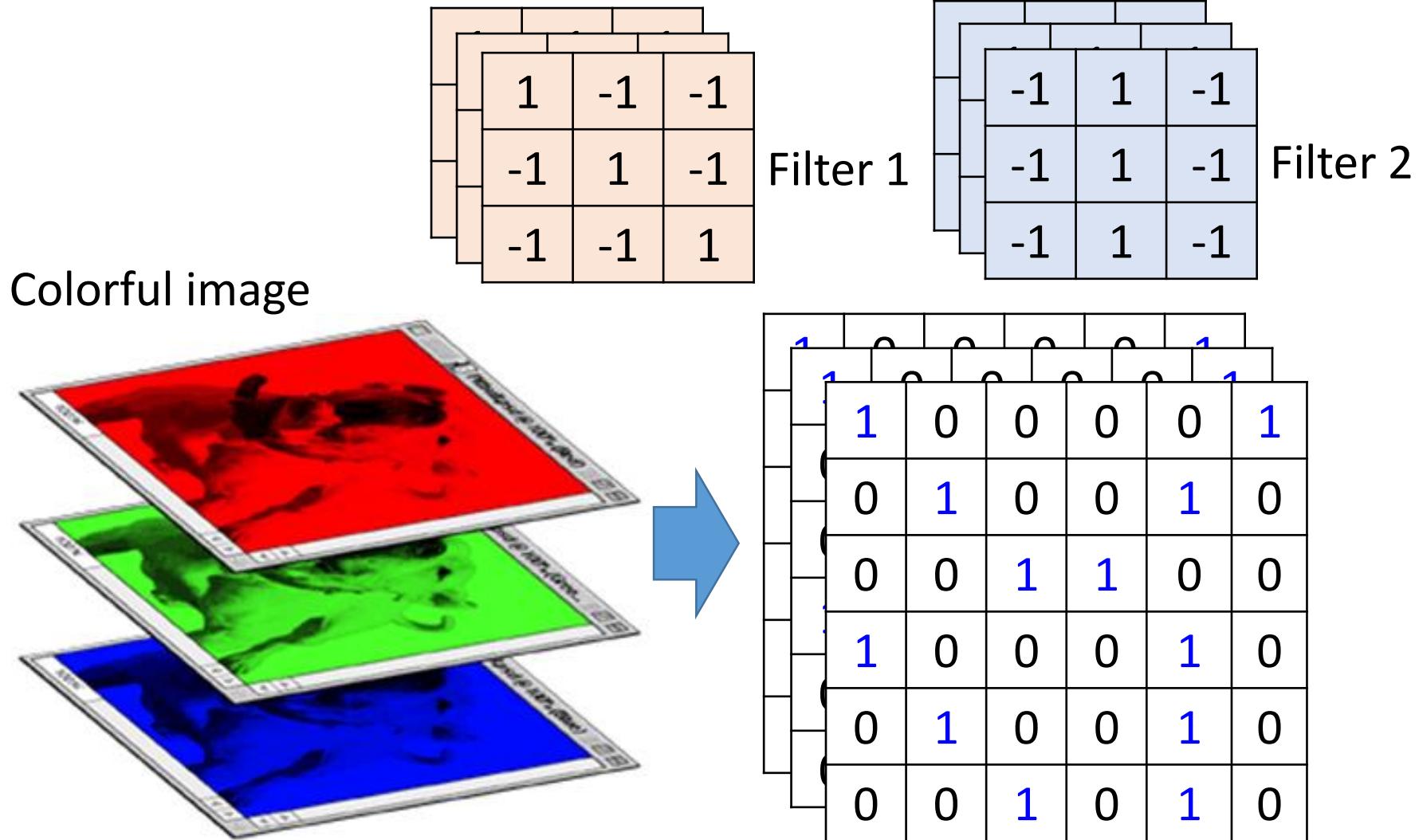
Filter 1

You will get another 6 x 6 images in this way

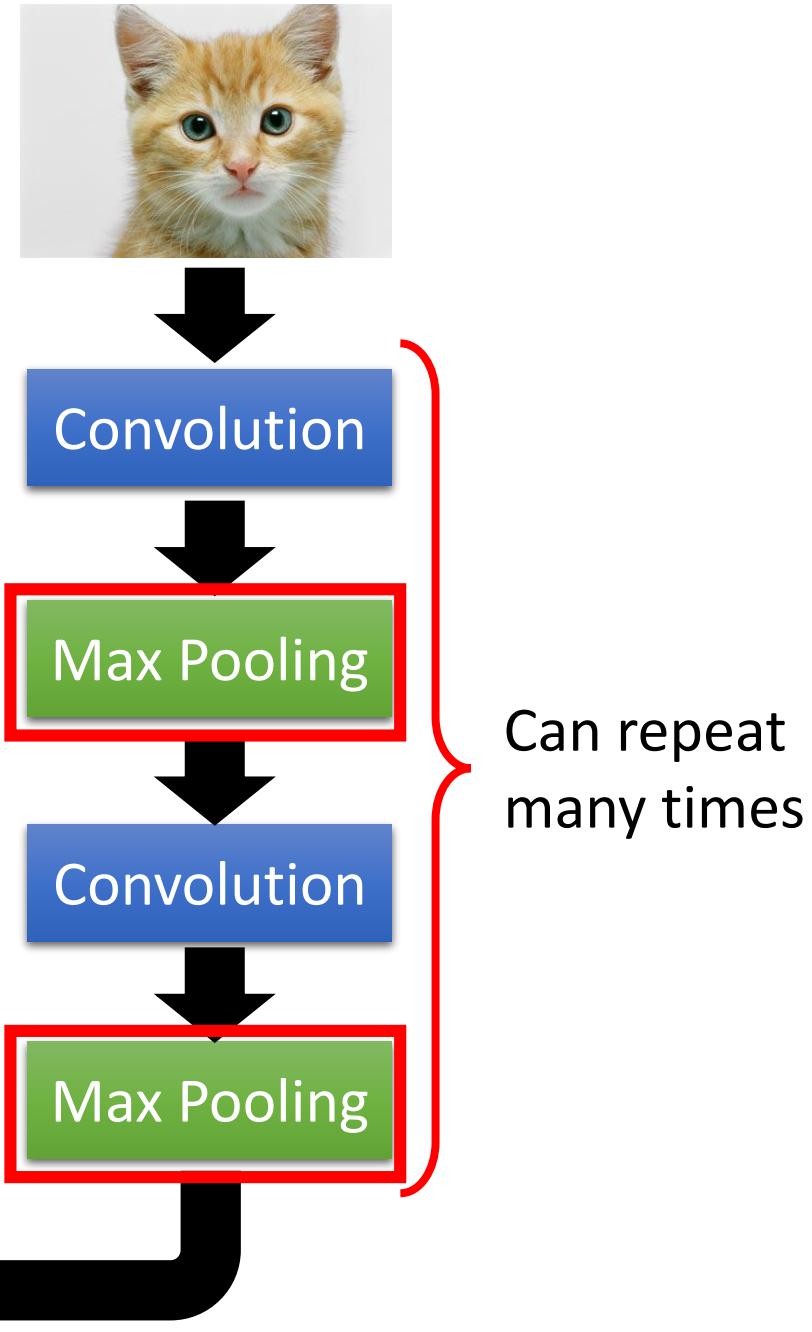
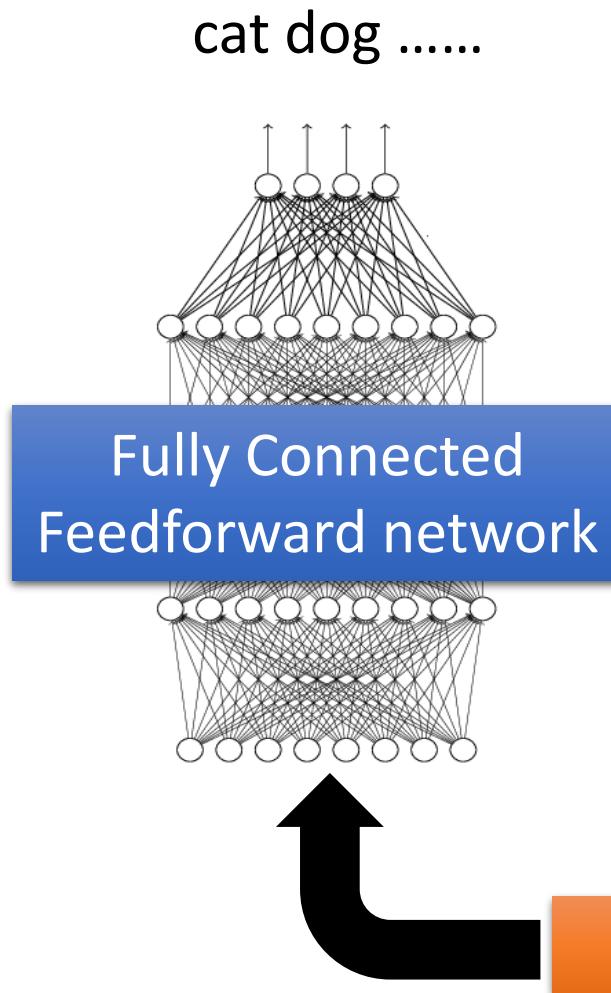


Zero padding

# CNN – Colorful image



# The whole CNN



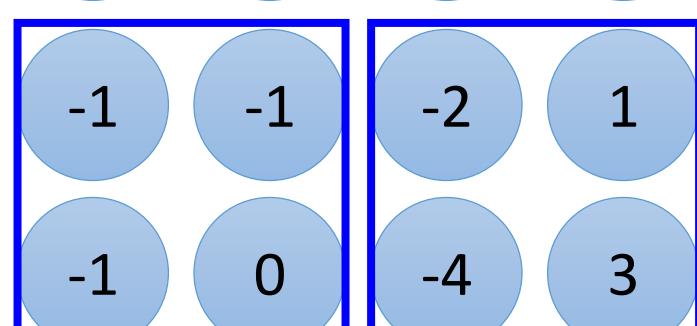
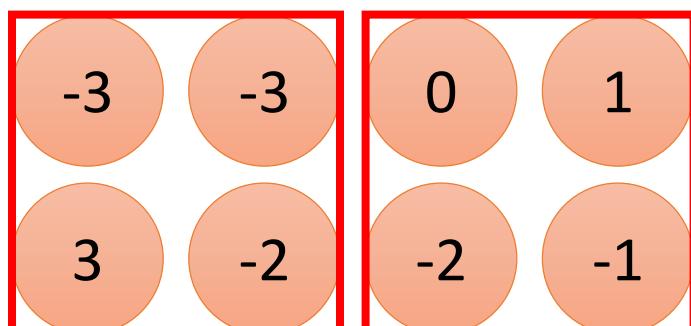
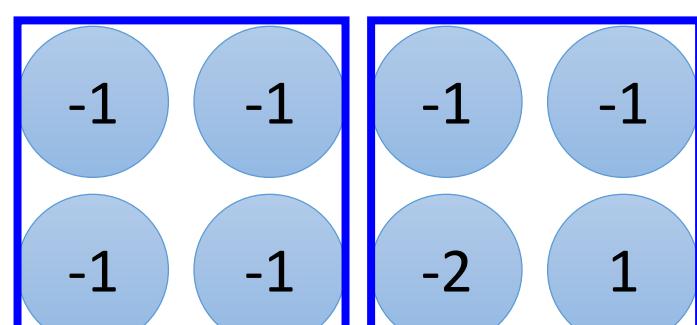
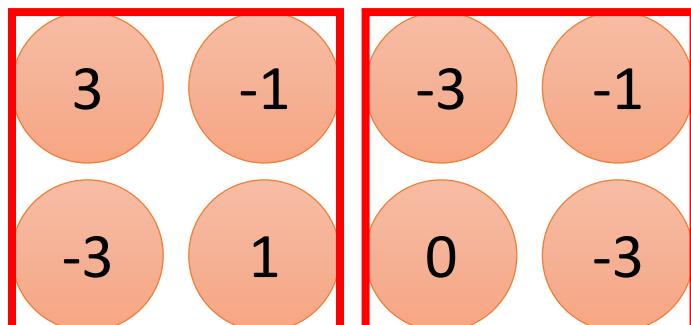
# CNN – Max Pooling

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

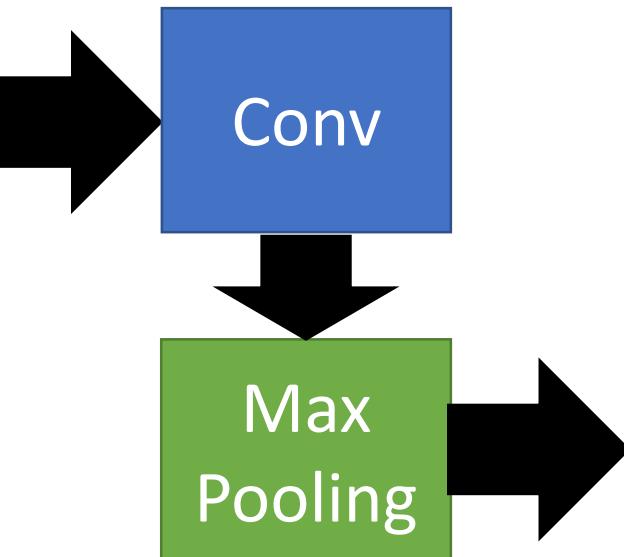
Filter 2



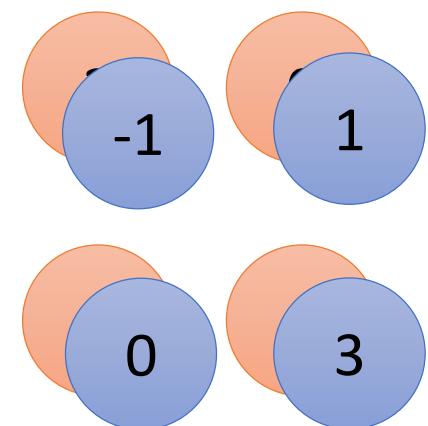
# CNN – Max Pooling

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image



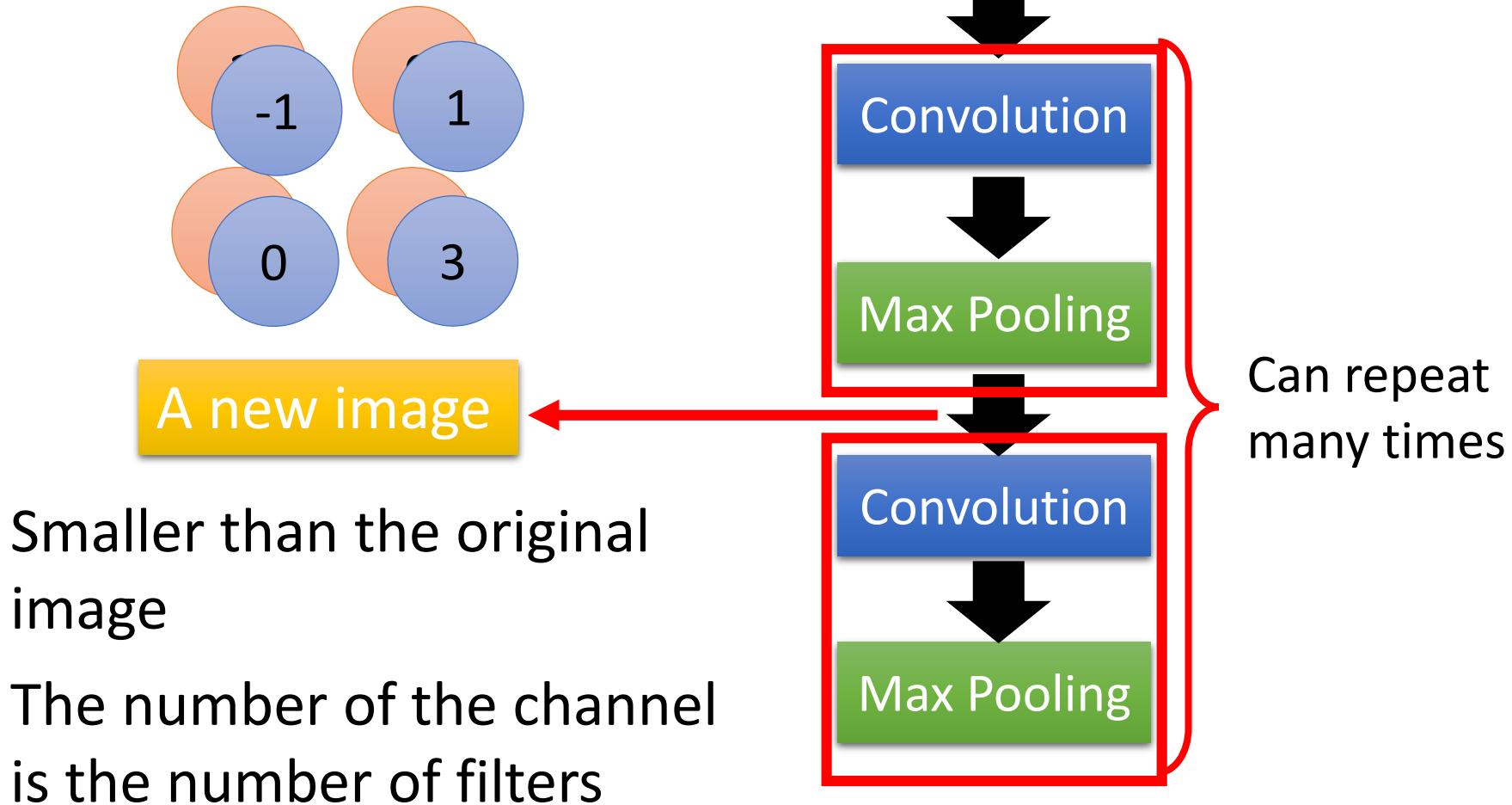
New image  
but smaller



2 x 2 image

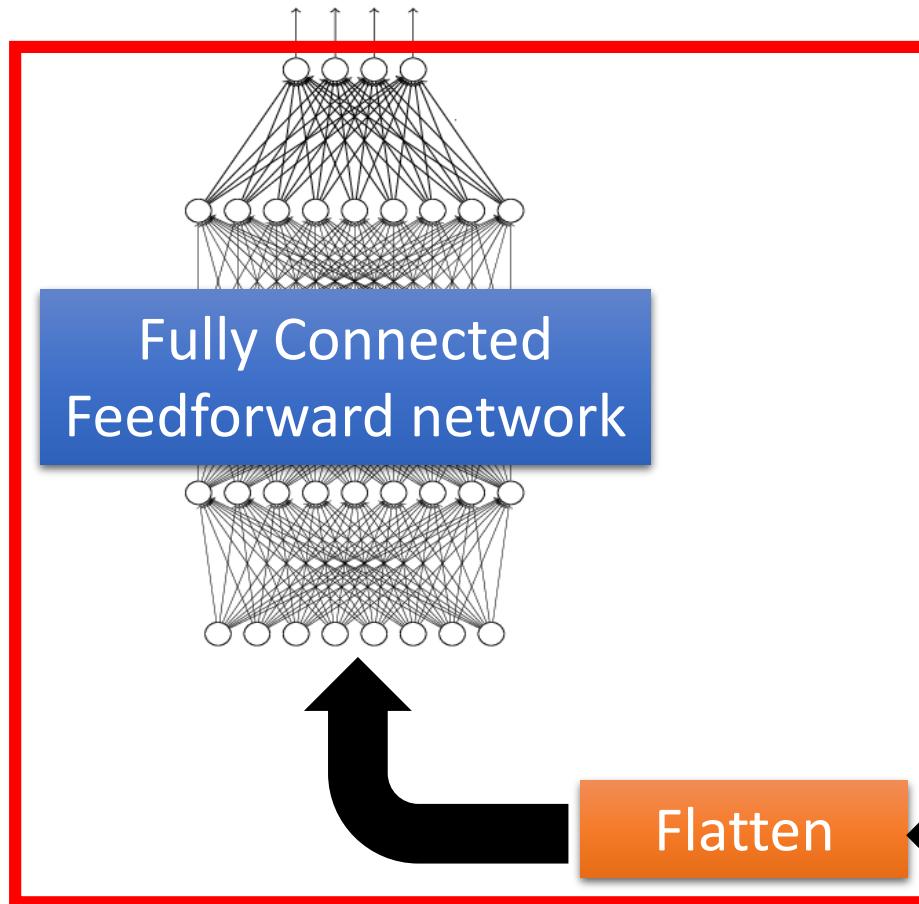
Each filter  
is a channel

# The whole CNN



# The whole CNN

cat dog .....



Convolution

Max Pooling

A new image

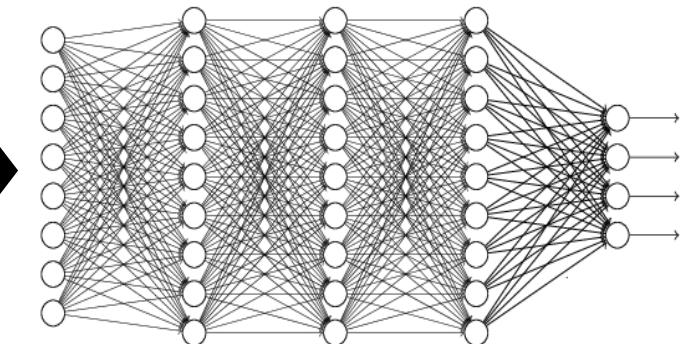
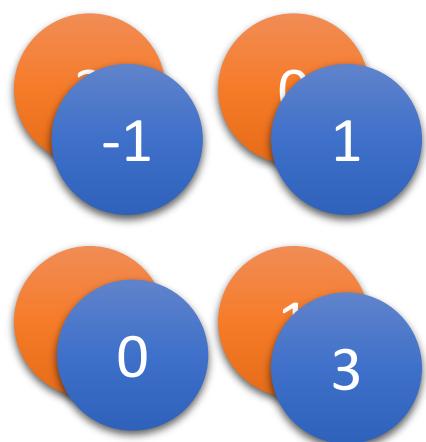
Convolution

Max Pooling

A new image

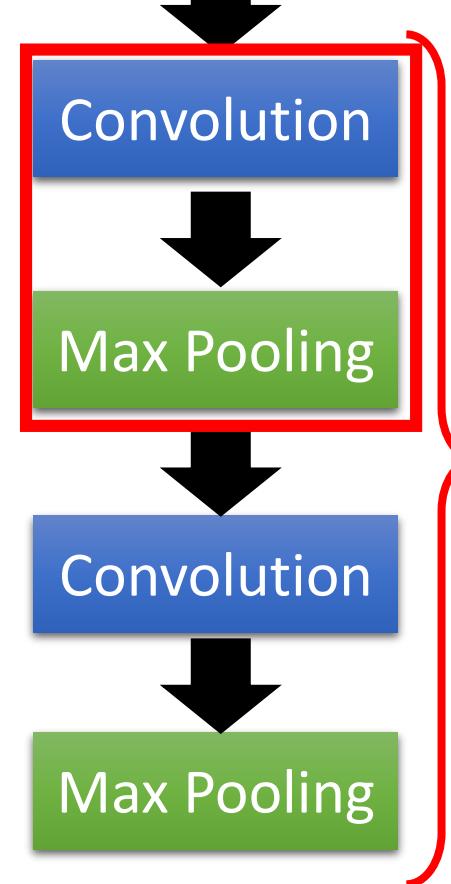
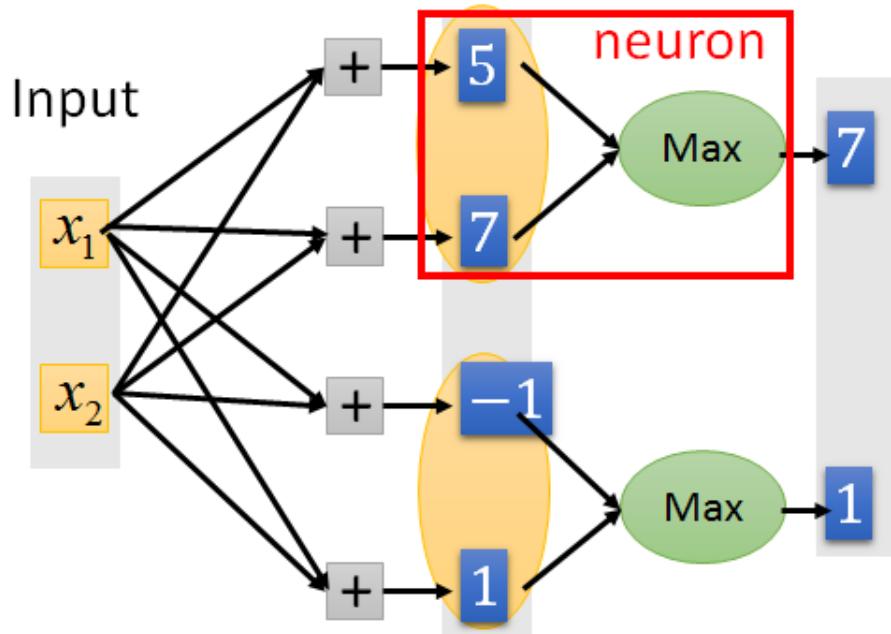
Flatten

# Flatten

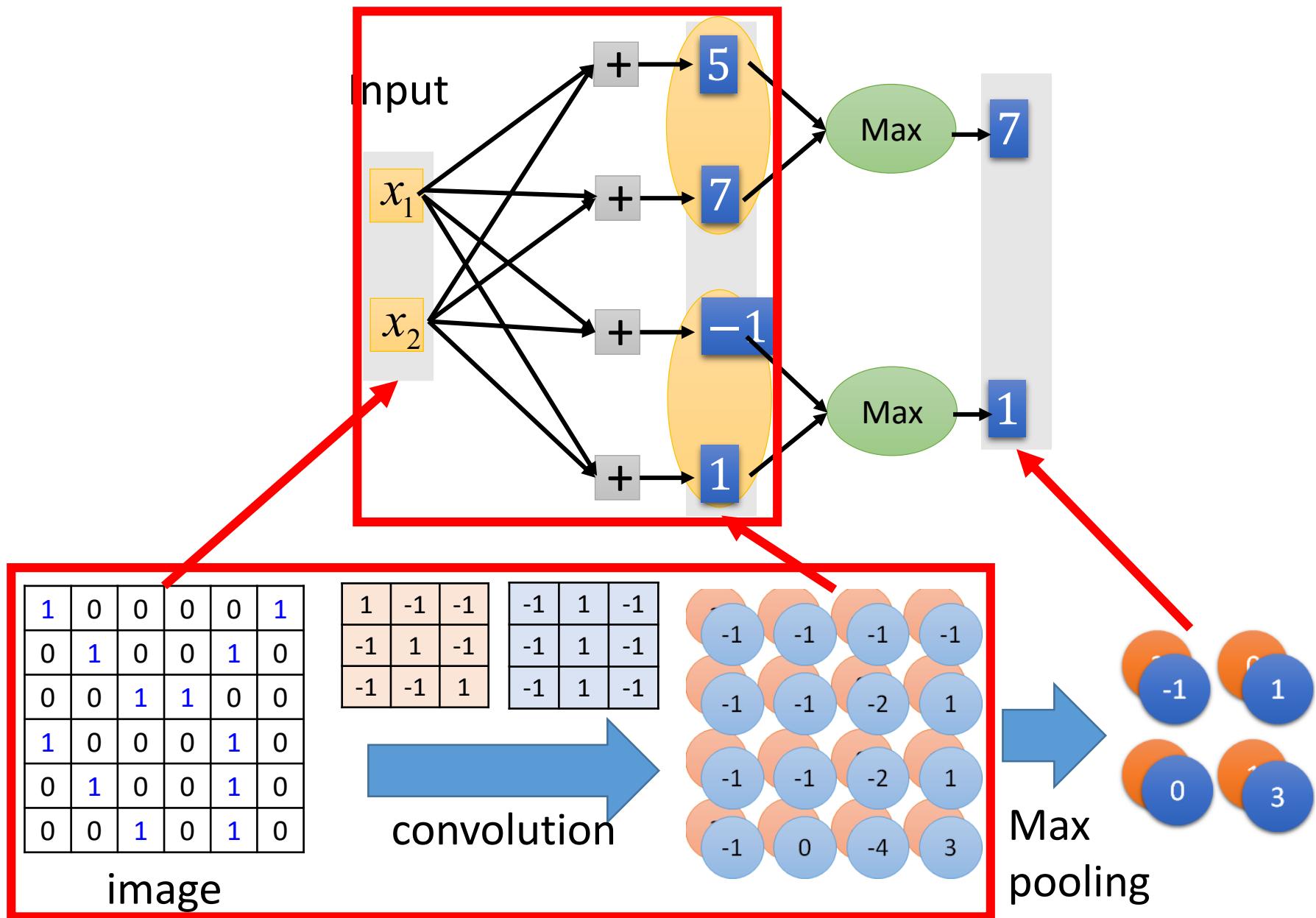


Fully Connected  
Feedforward network

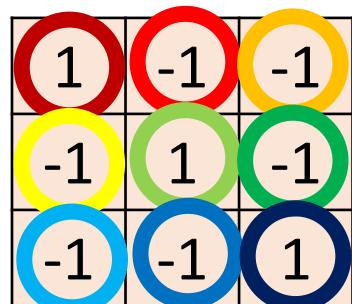
# The whole CNN



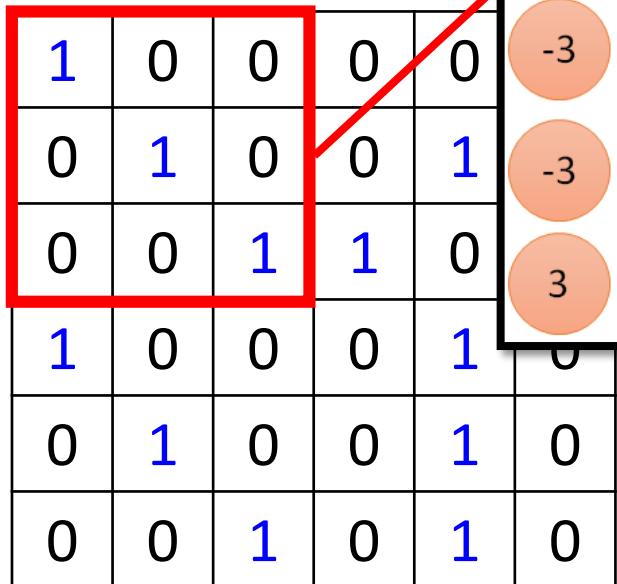
Can repeat  
many times



(Ignoring the non-linear activation function after the convolution.)

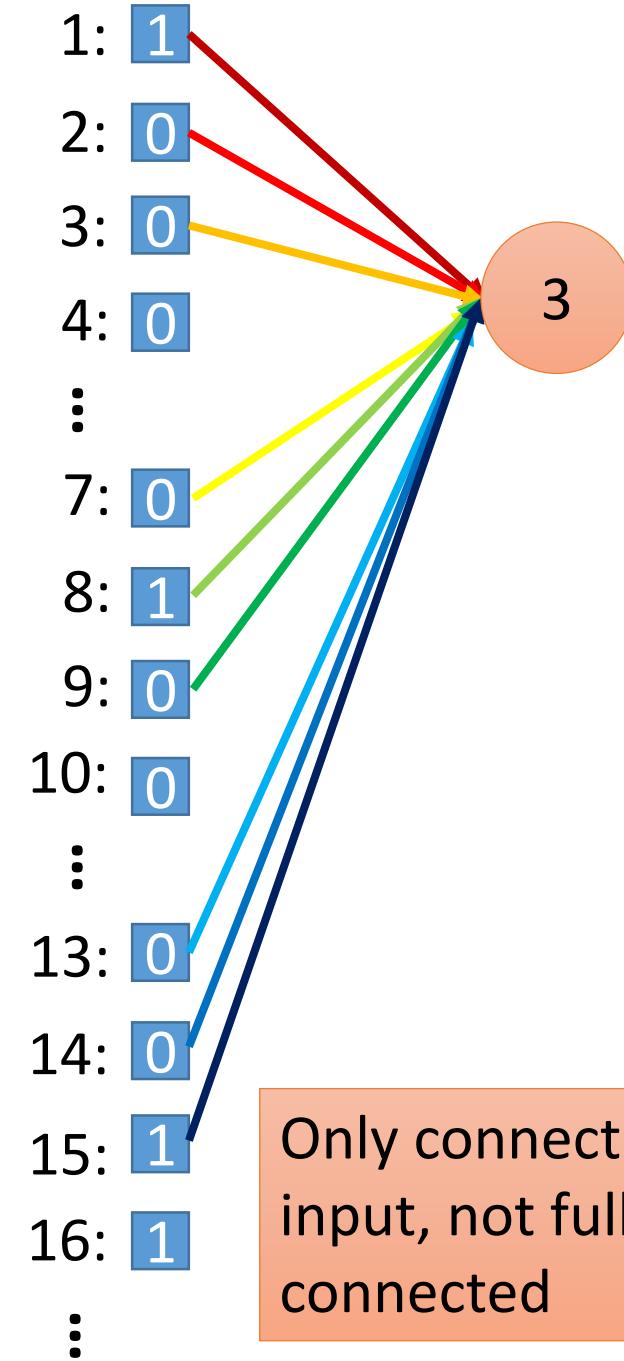
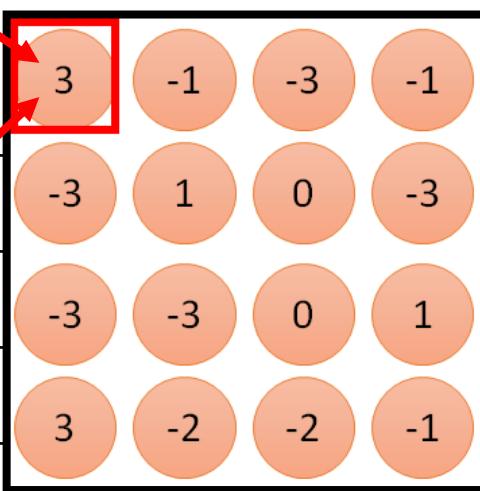


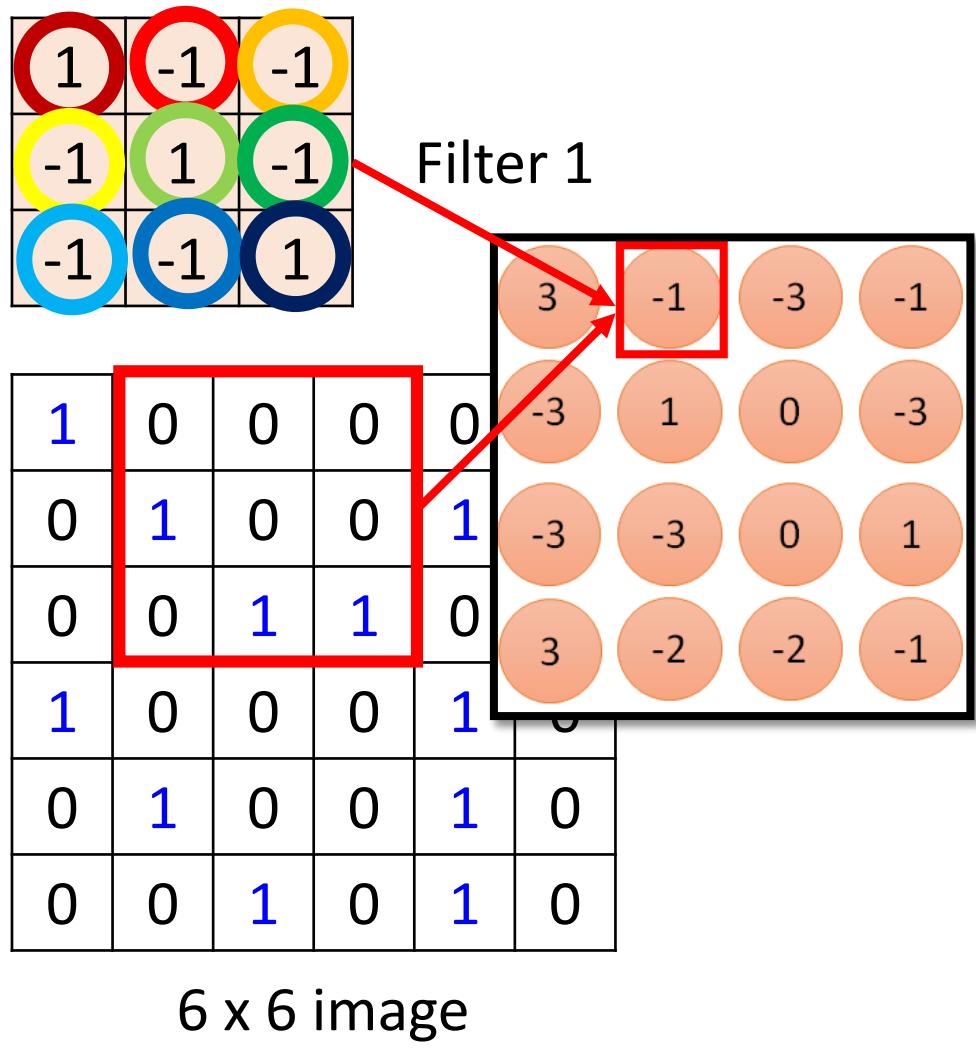
Filter 1



6 x 6 image

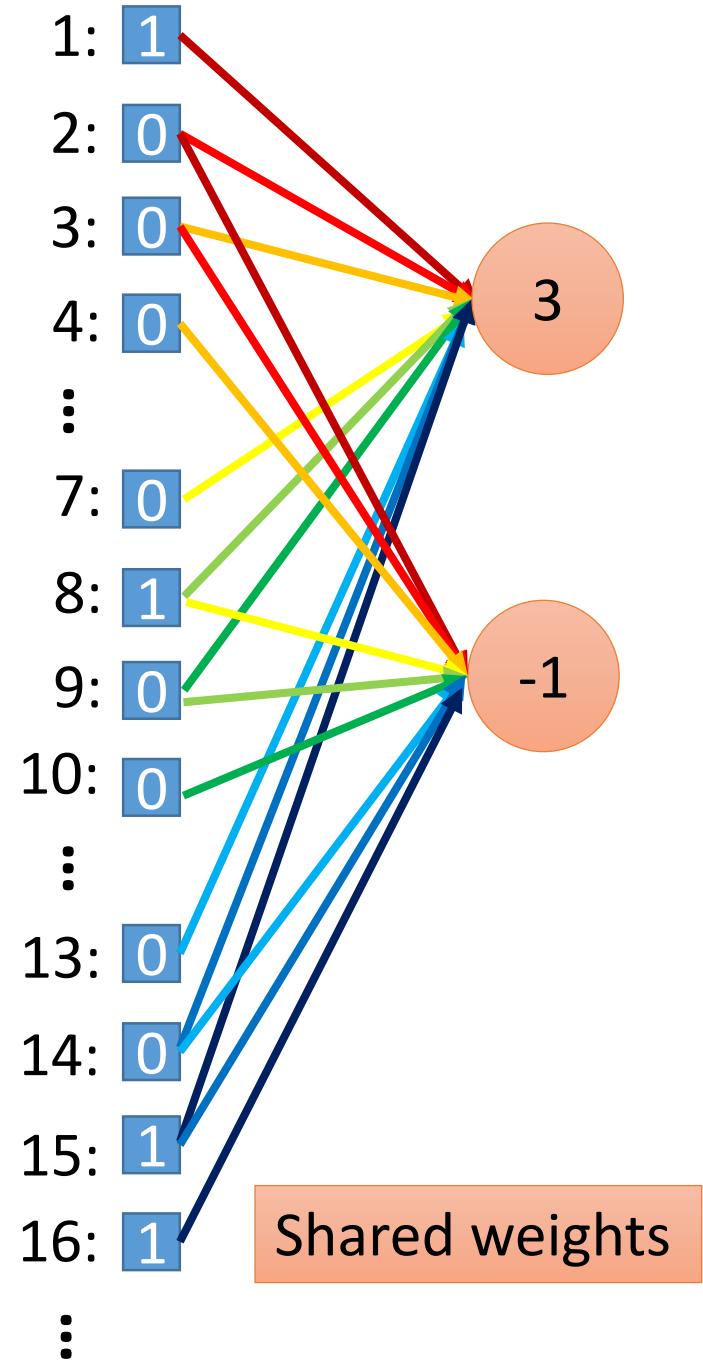
Less parameters!

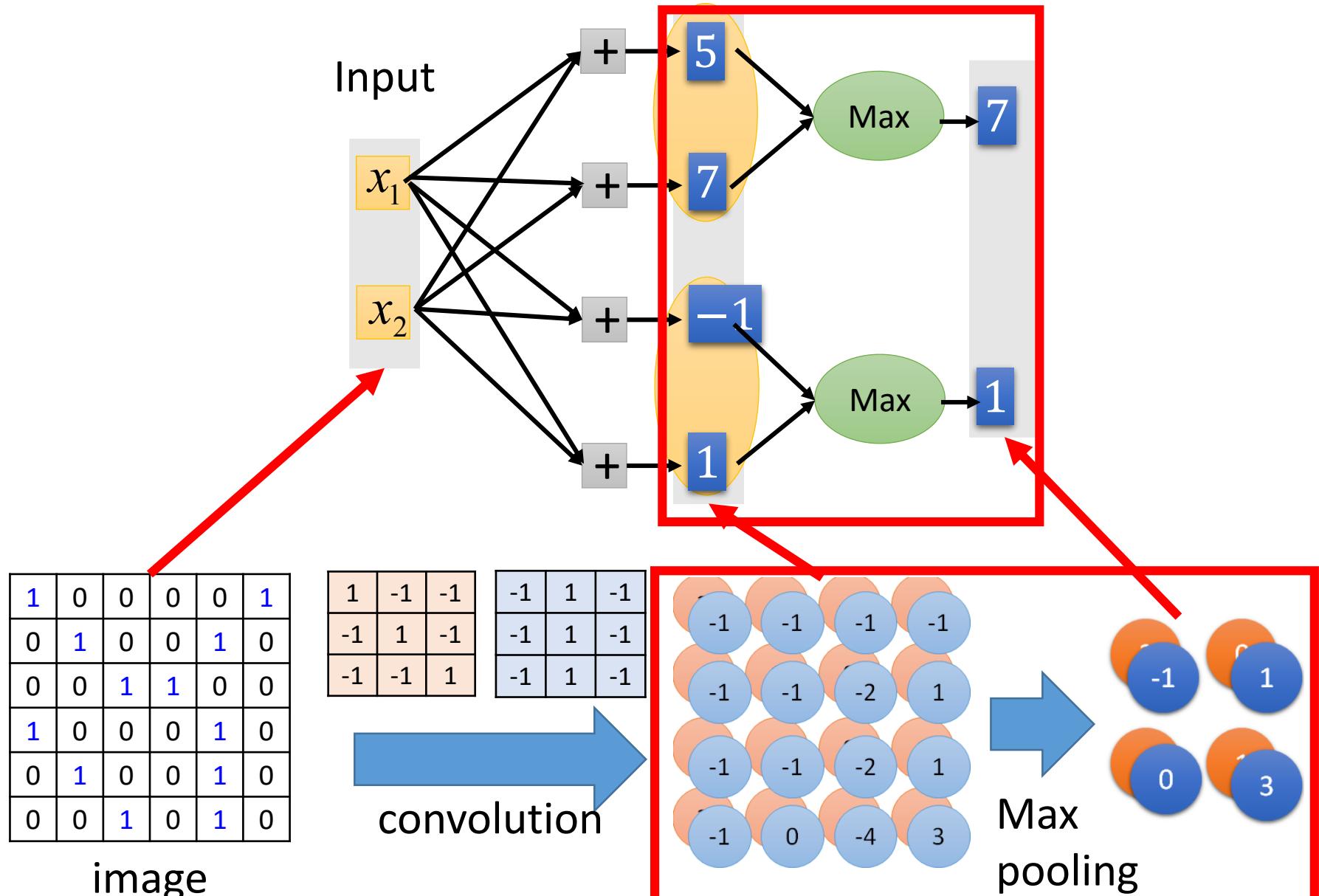




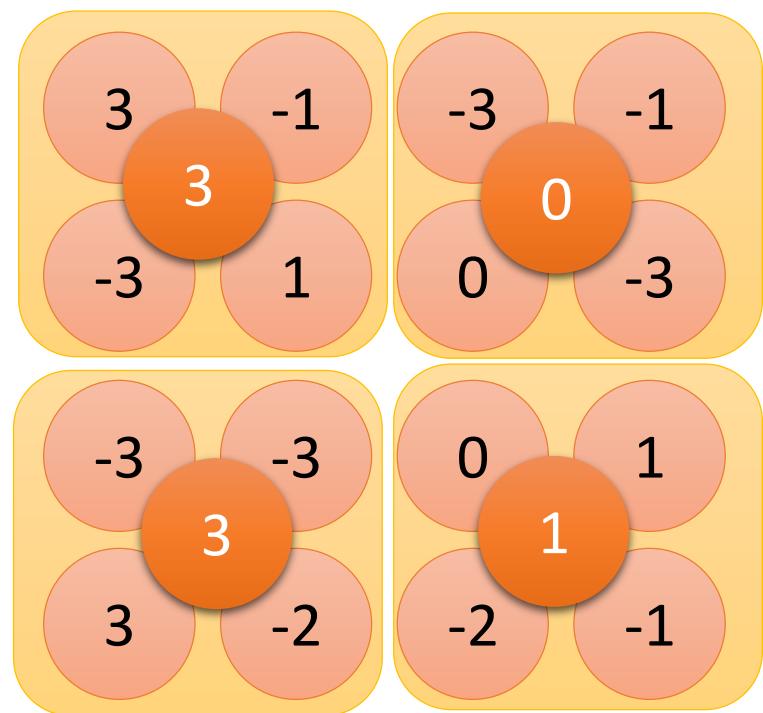
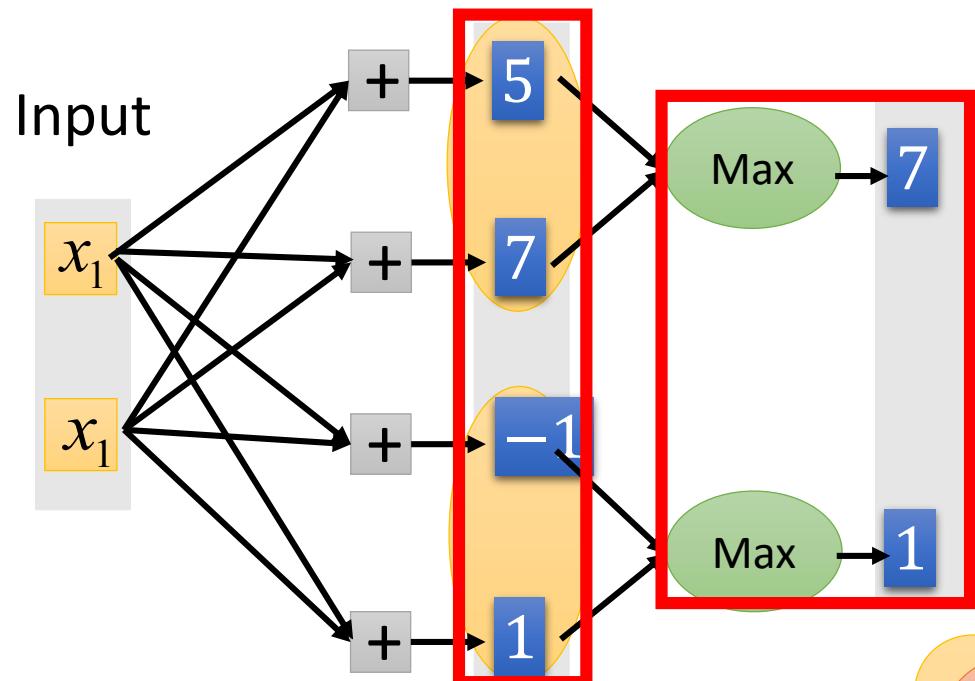
Less parameters!

Even less parameters!

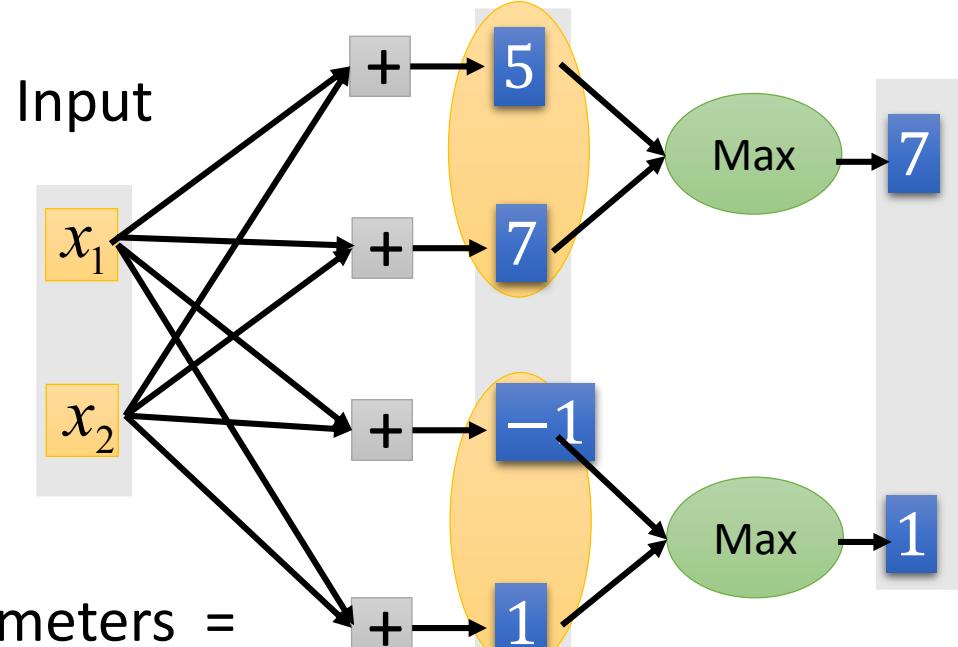




(Ignoring the non-linear activation function after the convolution.)



Dim =  $6 \times 6 = 36$



parameters =  
 $36 \times 32 = 1152$

Dim =  $4 \times 4 \times 2$   
 $= 32$

1	0	0	0	0	0	1
0	1	0	0	1	0	0
0	0	1	1	0	0	0
1	0	0	0	0	1	0
0	1	0	0	0	1	0
0	0	1	0	0	1	0

image

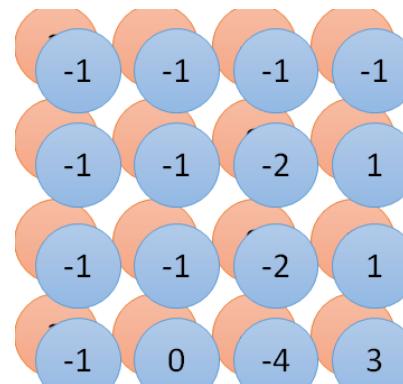
1	-1	-1
-1	1	-1
-1	-1	1

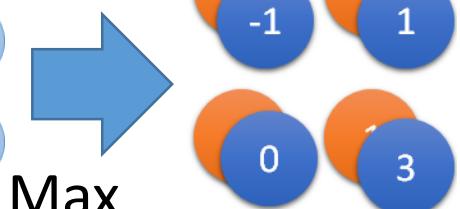
-1	1	-1
-1	1	-1
-1	1	-1

convolution

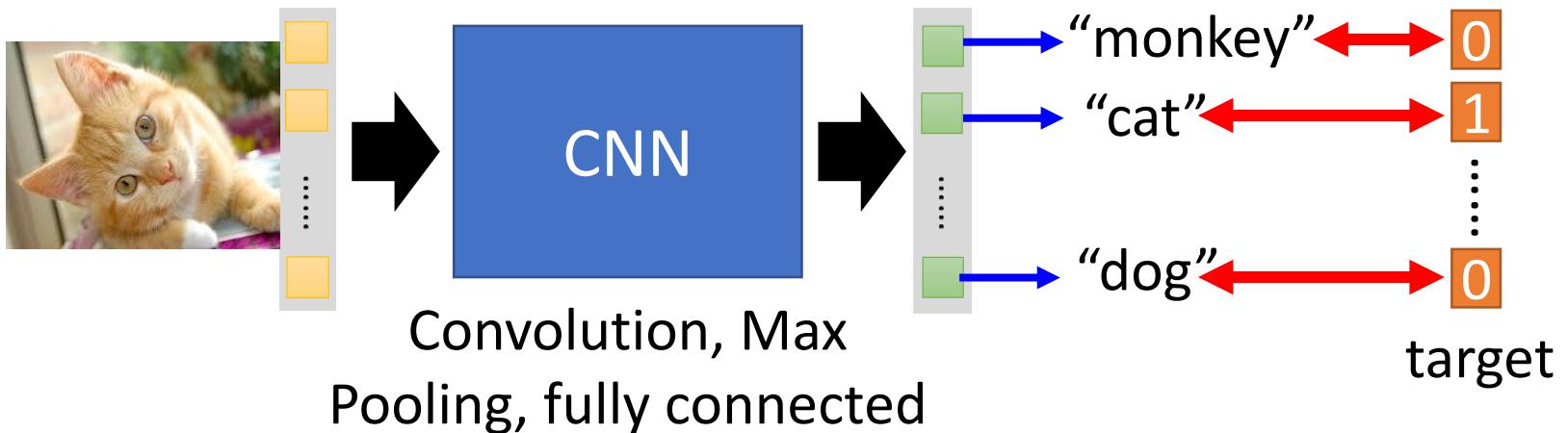
Only  $9 \times 2 = 18$   
parameters



Max  
pooling



# Convolutional Neural Network



Learning: Nothing special, just gradient descent .....

# Playing Go

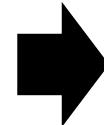


19 x 19 matrix  
(image)

Black: 1  
white: -1  
none: 0



Network



Next move  
(19 x 19  
positions)

19 x 19 vector

Fully-connected feedword  
network can be used

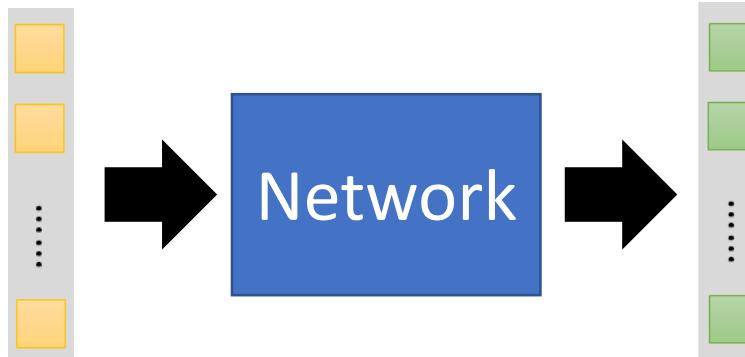
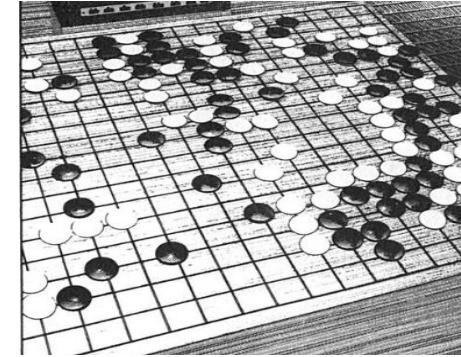
But CNN performs much better.

# Playing Go

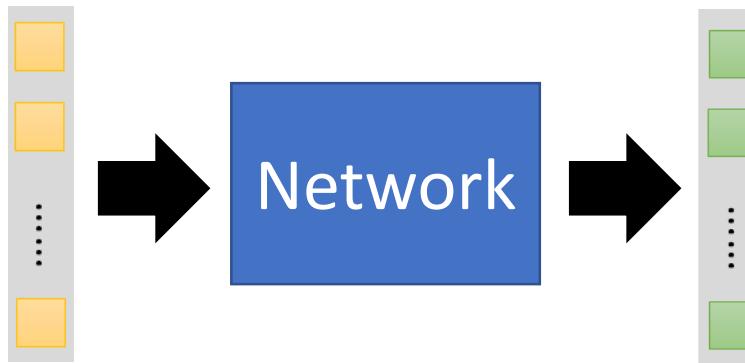
Training:



record of previous plays



Target:  
“天元” = 1  
else = 0



Target:  
“五之5” = 1  
else = 0

進藤光 v.s. 社青春

黒: 5之五

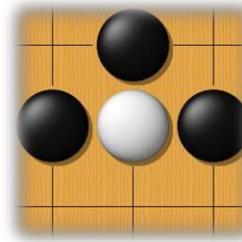
→ 白: 天元

→ 黑: 五之5

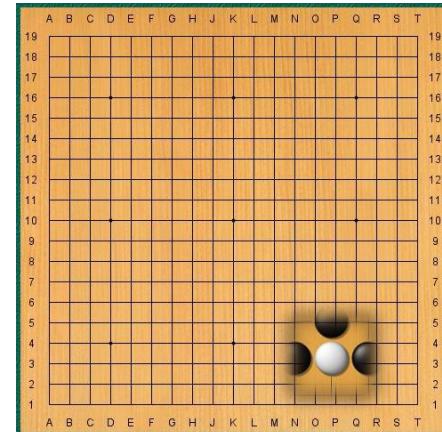
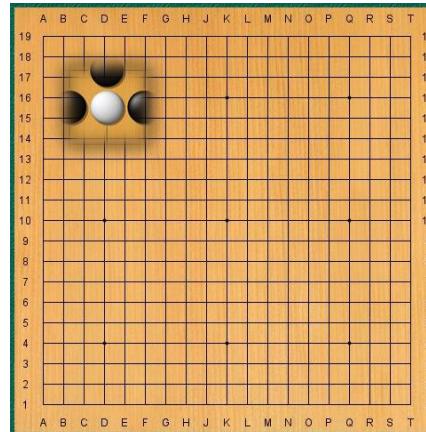
# Why CNN for playing Go?

- Some patterns are much smaller than the whole image

Alpha Go uses  $5 \times 5$  for first layer



- The same patterns appear in different regions.



# Why CNN for playing Go?

- Subsampling the pixels will not change the object



Max Pooling

How to explain this???

**Neural network architecture.** The input to the policy network is a  $19 \times 19 \times 48$  image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a  $23 \times 23$  image, then convolves  $k$  filters of kernel size  $5 \times 5$  with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a  $21 \times 21$  image, then convolves  $k$  filters of kernel size  $3 \times 3$  with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size  $1 \times 1$  with stride 1 with a different bias for each position and applies a softmax function. The Alpha Go does not use Max Pooling ..... Extended Data Table 3 additionally show the results of training with  $k = 128, 256$  and  $384$  filters.

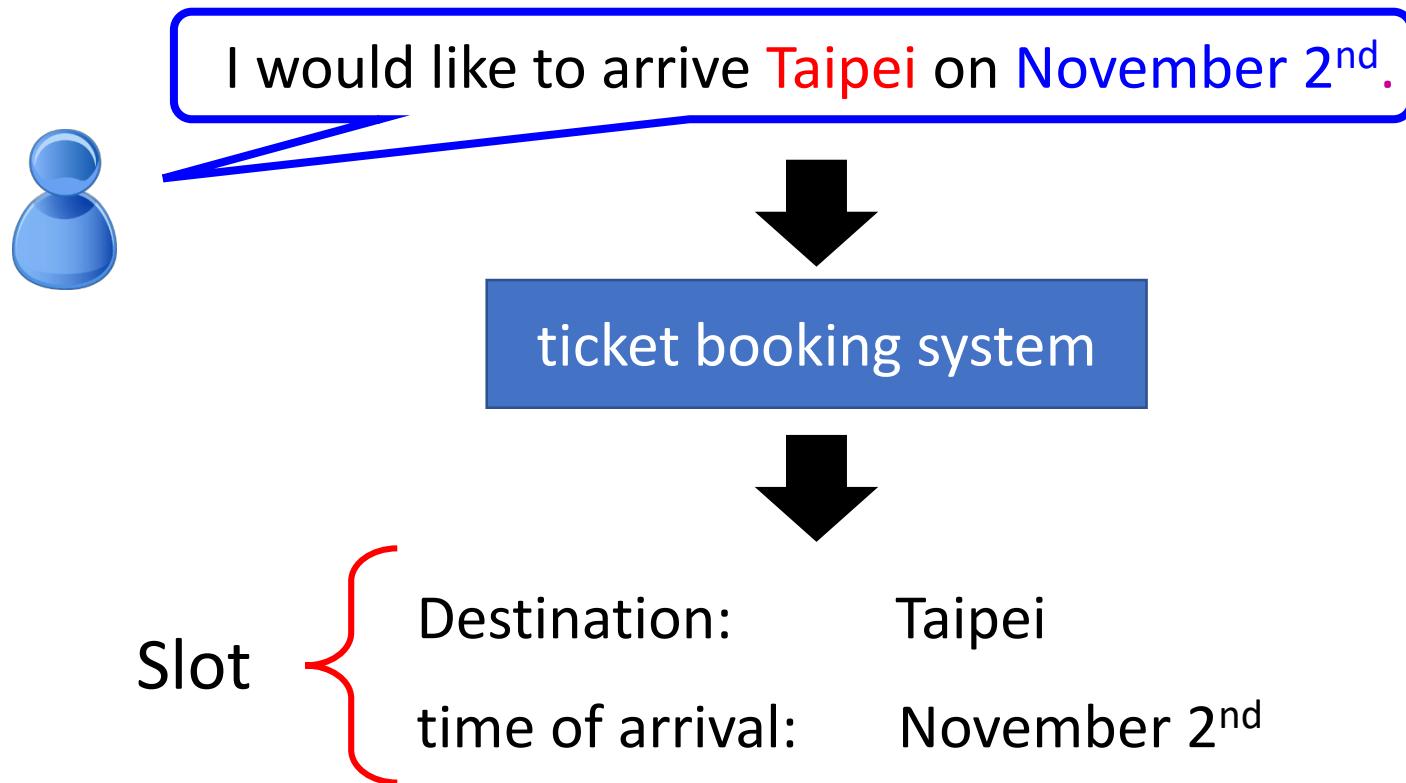
# Variants of Neural Networks

Convolutional Neural  
Network (CNN)

Recurrent Neural Network  
(RNN)      Neural Network with Memory

# Example Application

- Slot Filling



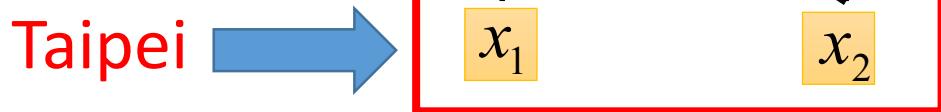
# Example Application

Solving slot filling by  
Feedforward network?

Input: a word

(Each word is represented  
as a vector)

Taipei



# 1-of-N encoding

How to represent each word as a vector?

**1-of-N Encoding** lexicon = {apple, bag, cat, dog, elephant}

The vector is lexicon size.

$$\text{apple} = [1 \ 0 \ 0 \ 0 \ 0]$$

Each dimension corresponds  
to a word in the lexicon

$$\text{bag} = [0 \ 1 \ 0 \ 0 \ 0]$$

The dimension for the word  
is 1, and others are 0

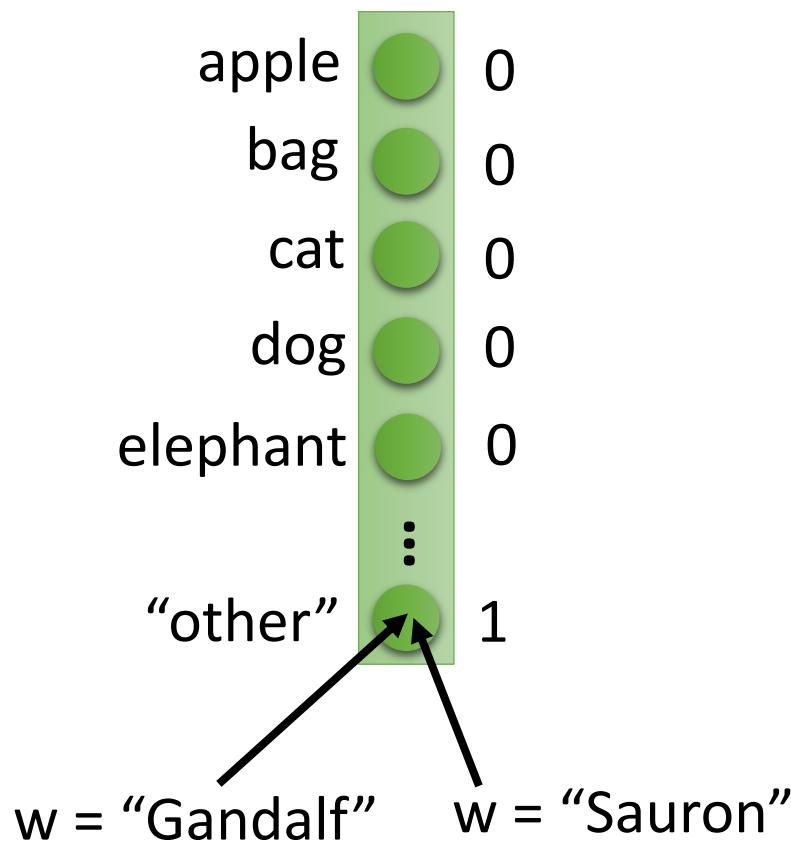
$$\text{cat} = [0 \ 0 \ 1 \ 0 \ 0]$$

$$\text{dog} = [0 \ 0 \ 0 \ 1 \ 0]$$

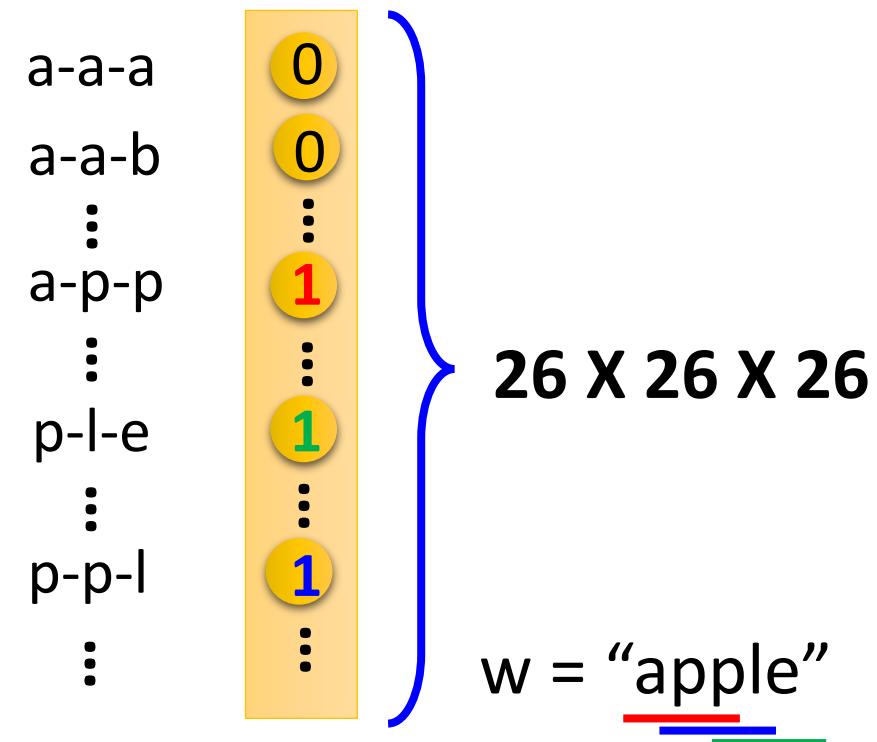
$$\text{elephant} = [0 \ 0 \ 0 \ 0 \ 1]$$

# Beyond 1-of-N encoding

## *Dimension for “Other”*



## *Word hashing*



# Example Application

Solving slot filling by  
Feedforward network?

Input: a word

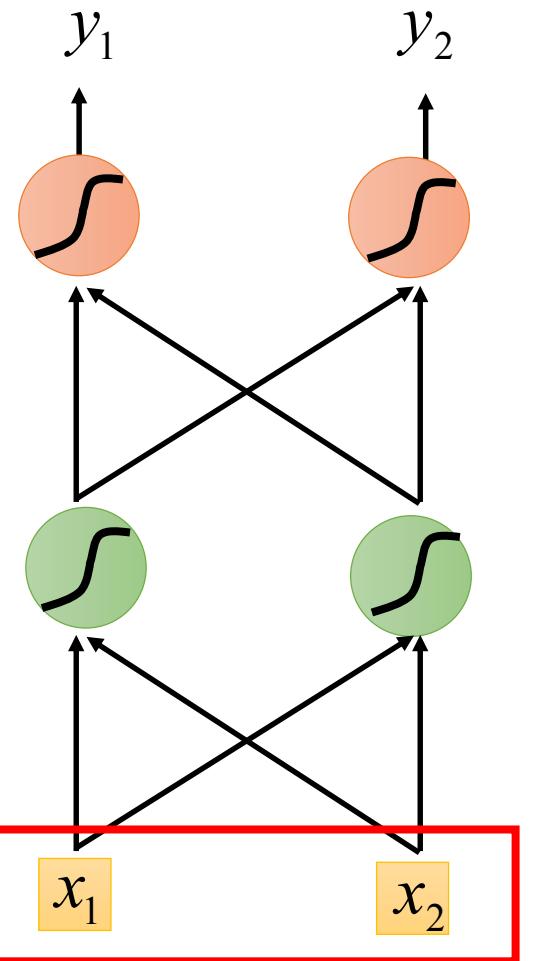
(Each word is represented  
as a vector)

Output:

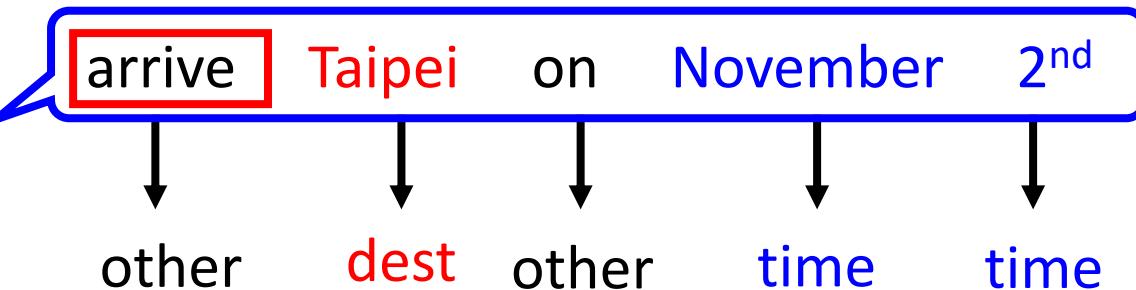
Probability distribution that  
the input word belonging to  
the slots

Taipei

dest  
time of  
departure



# Example Application

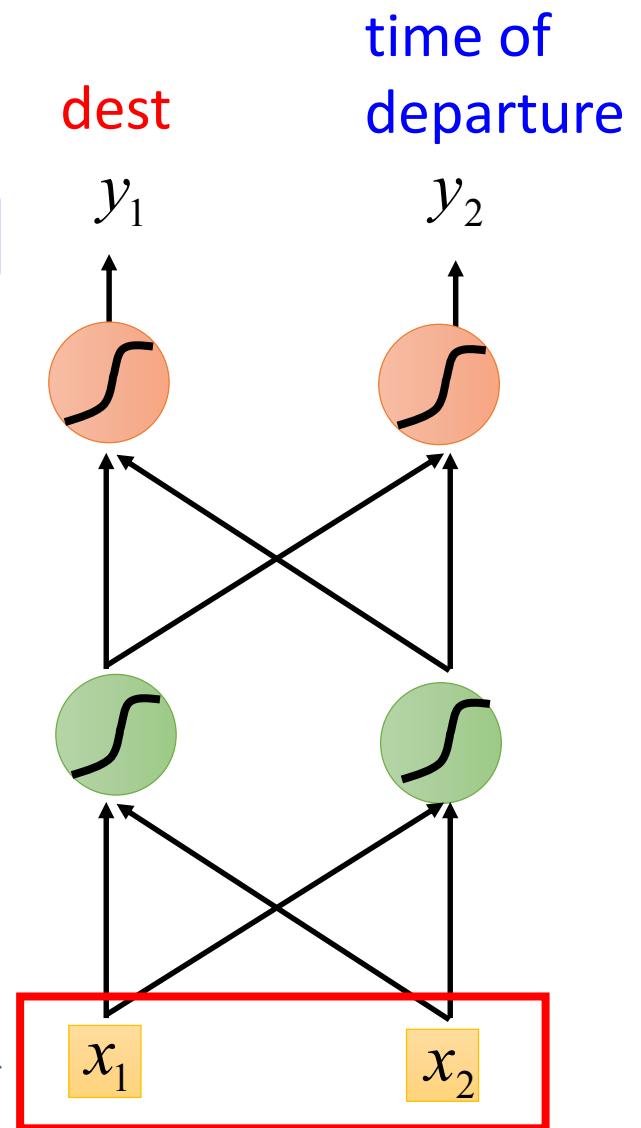


Problem?



Neural network  
needs memory!

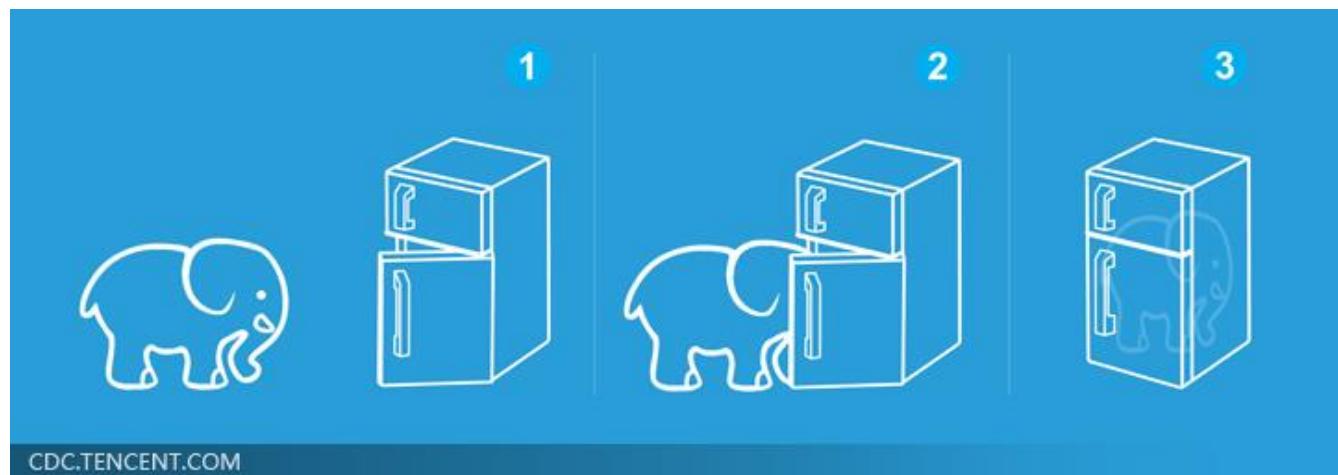
Taipei →



# Three Steps for Deep Learning

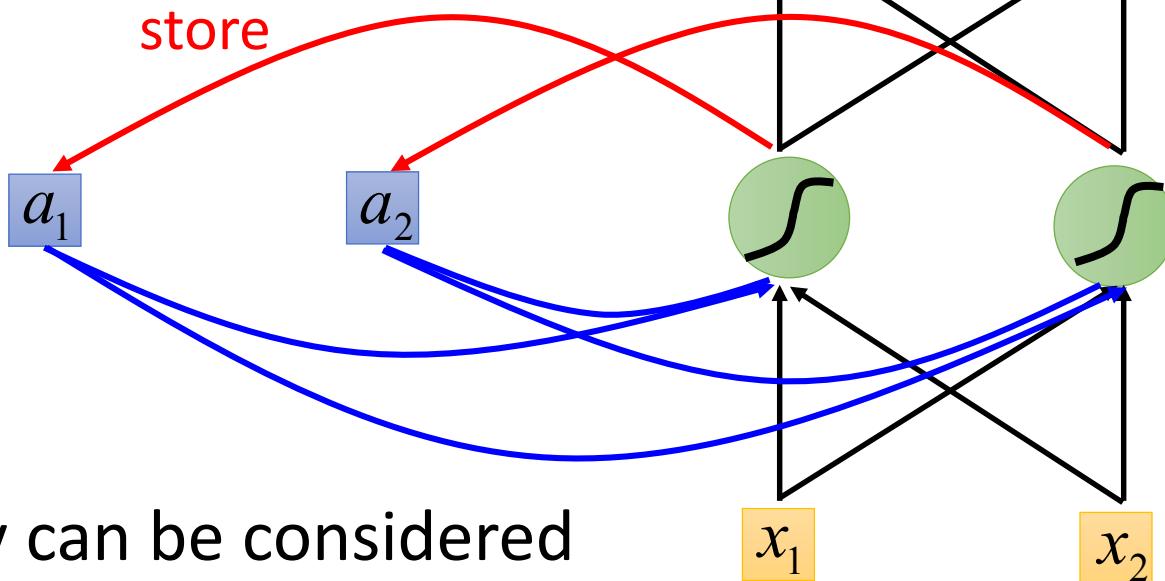


Deep Learning is so simple .....



# Recurrent Neural Network (RNN)

The output of hidden layer  
are stored in the memory.



Memory can be considered  
as another input.

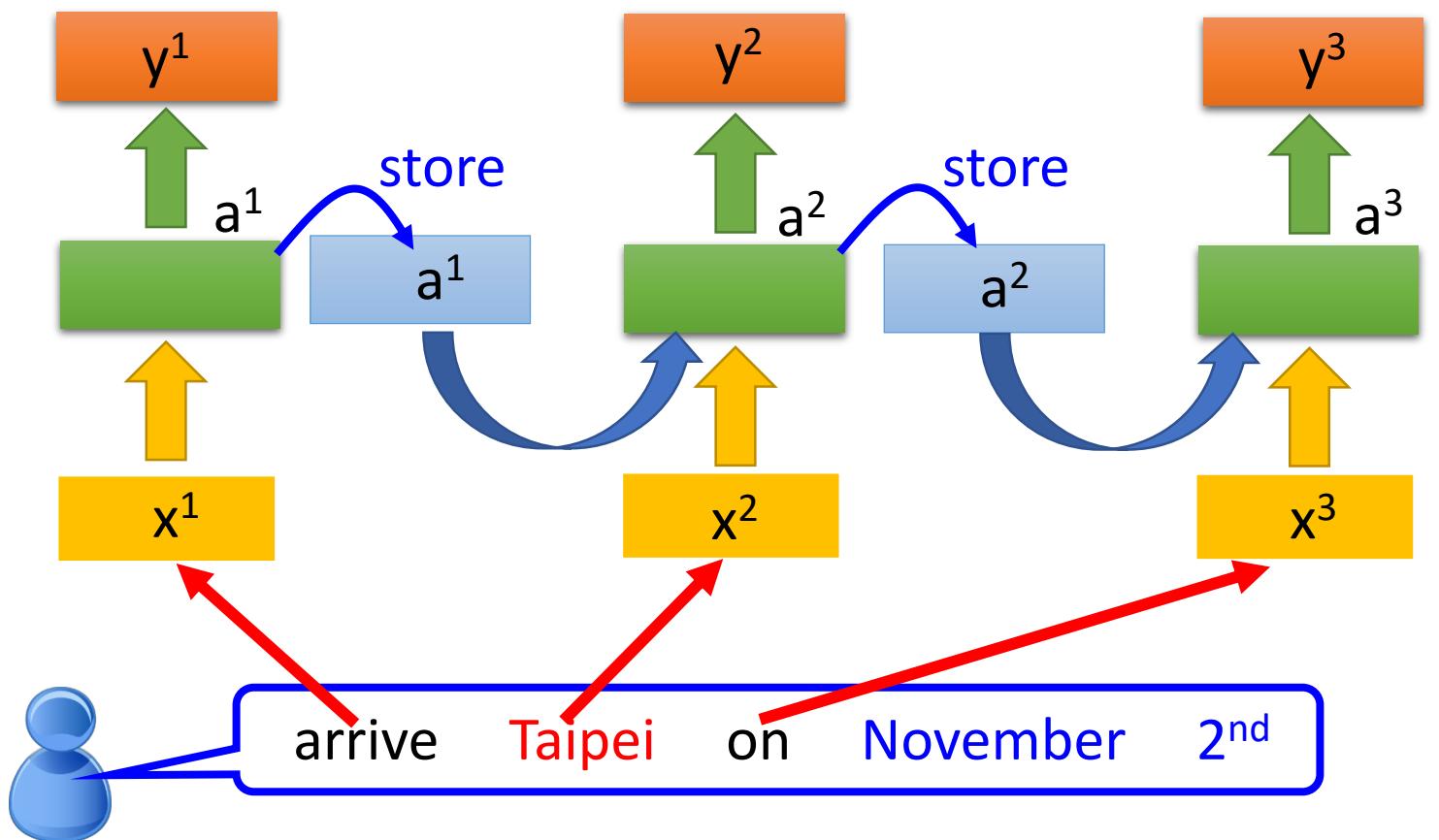
# RNN

The same network is used again and again.

Probability of  
“arrive” in each slot

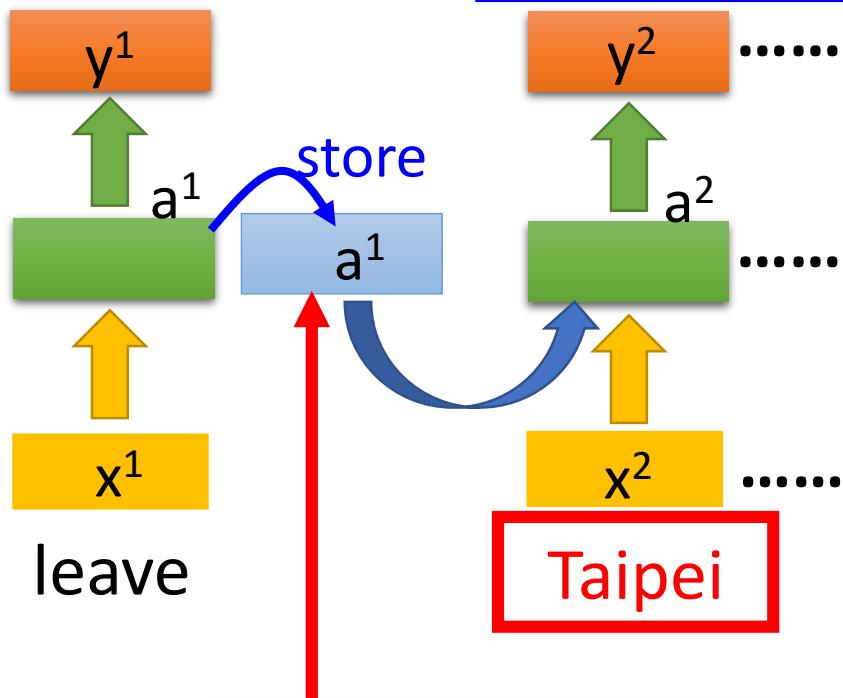
Probability of  
“Taipei” in each slot

Probability of  
“on” in each slot



# RNN

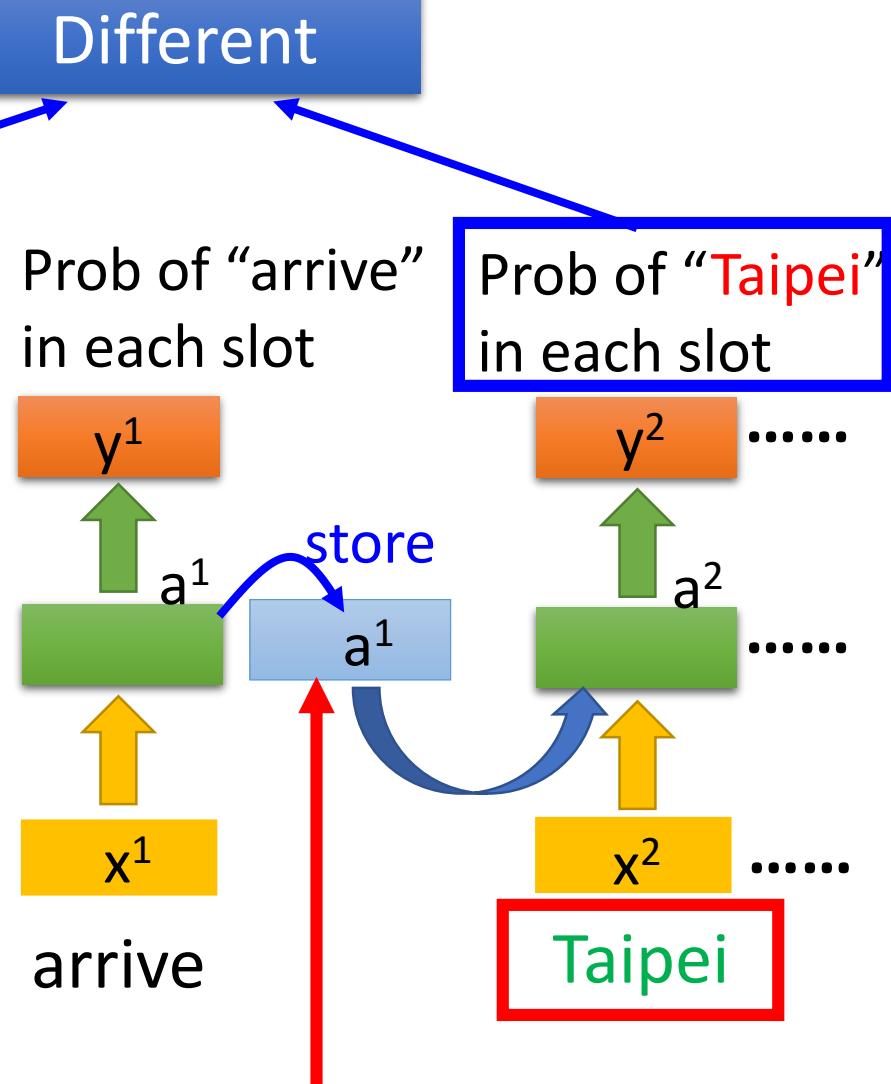
Prob of “leave”  
in each slot



Prob of “Taipei”  
in each slot

Different

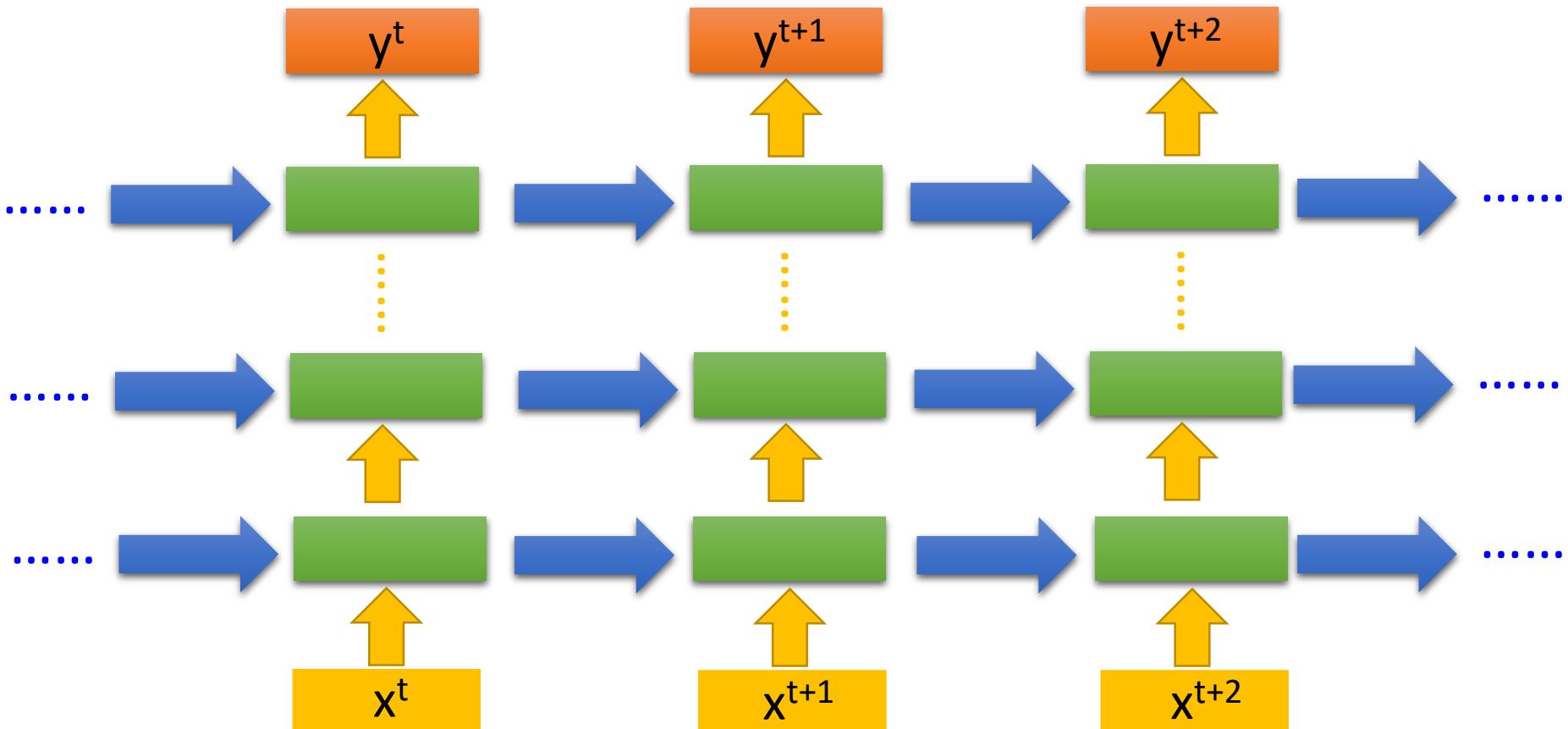
Prob of “arrive”  
in each slot



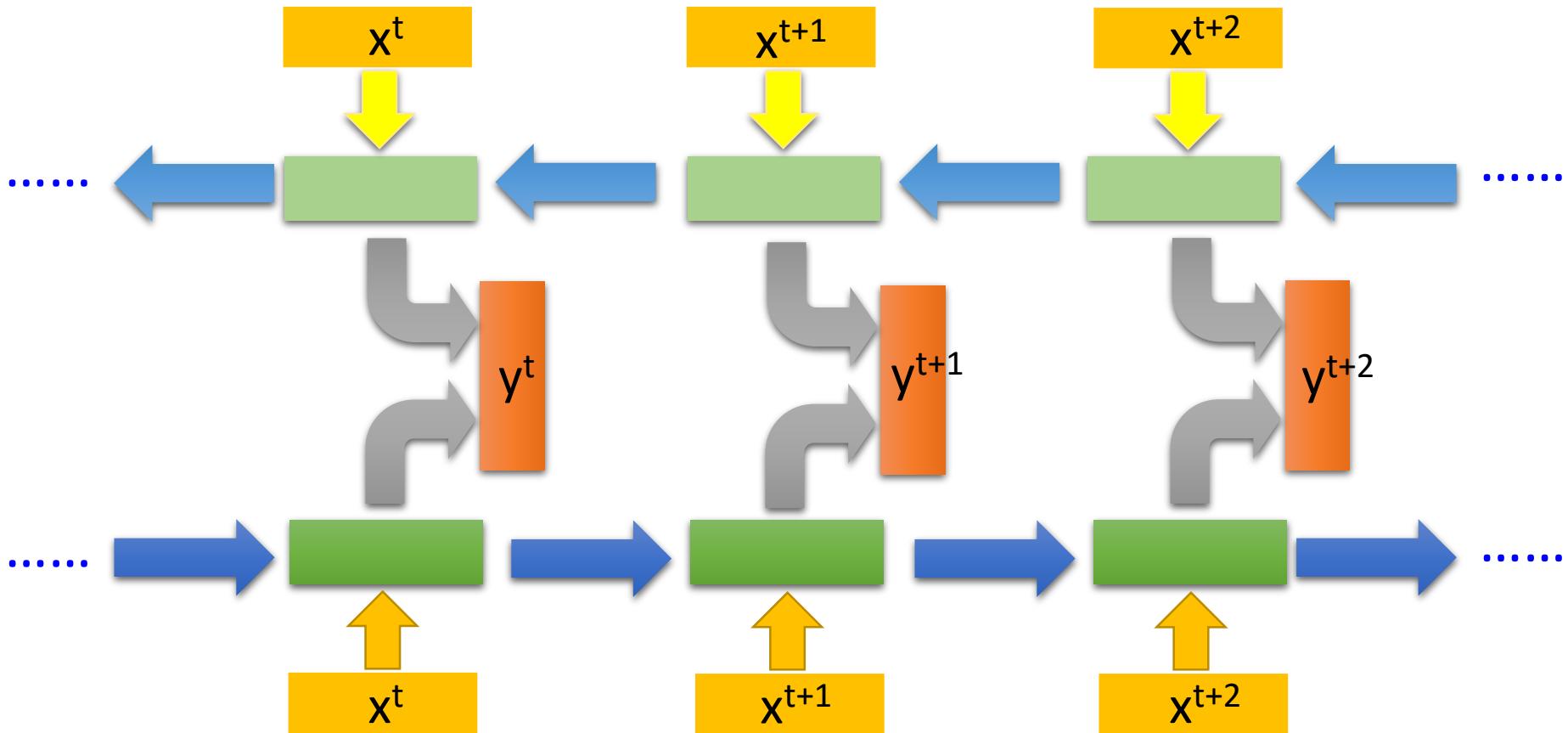
Prob of “Taipei”  
in each slot

The values stored in the memory is different.

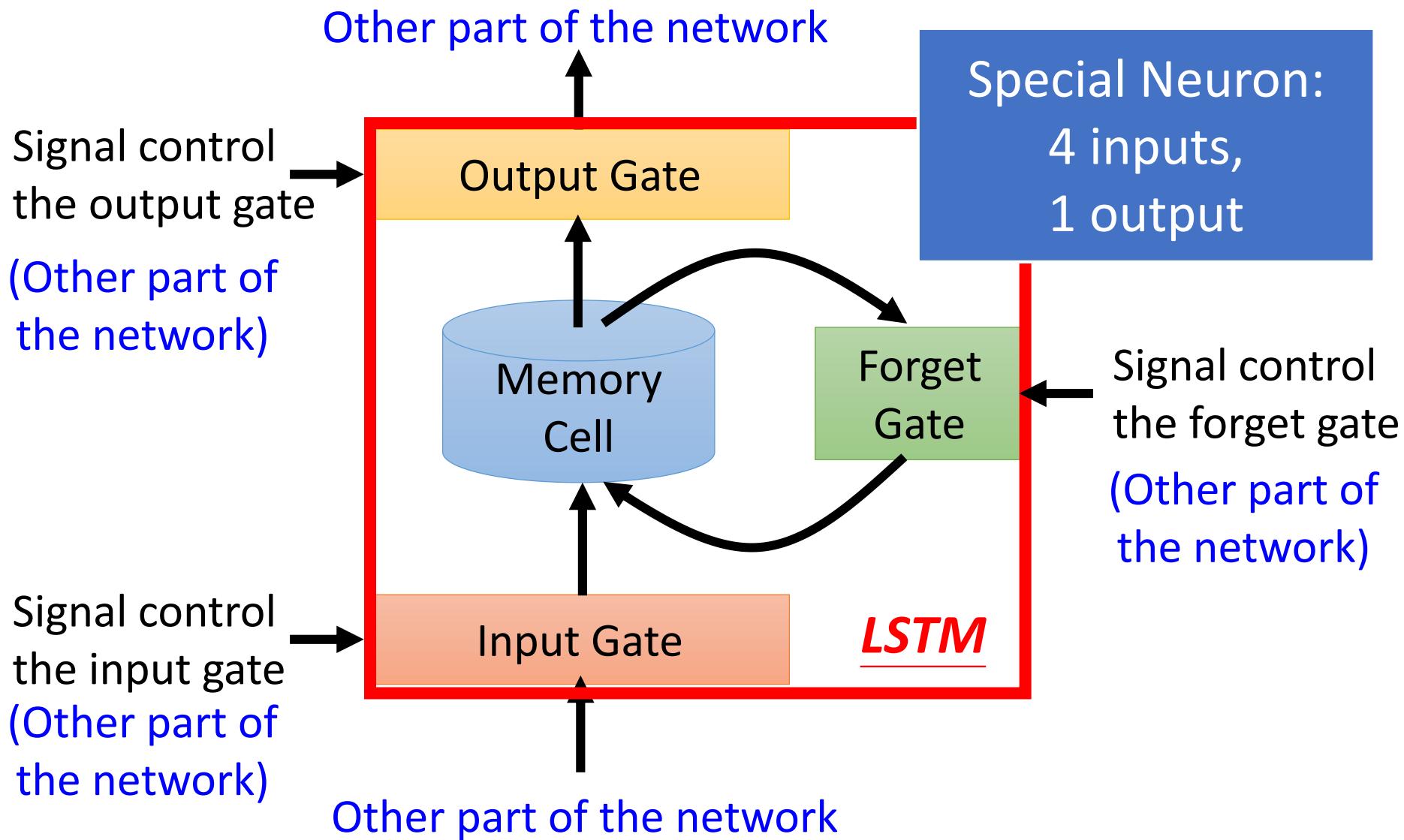
# Of course it can be deep ...

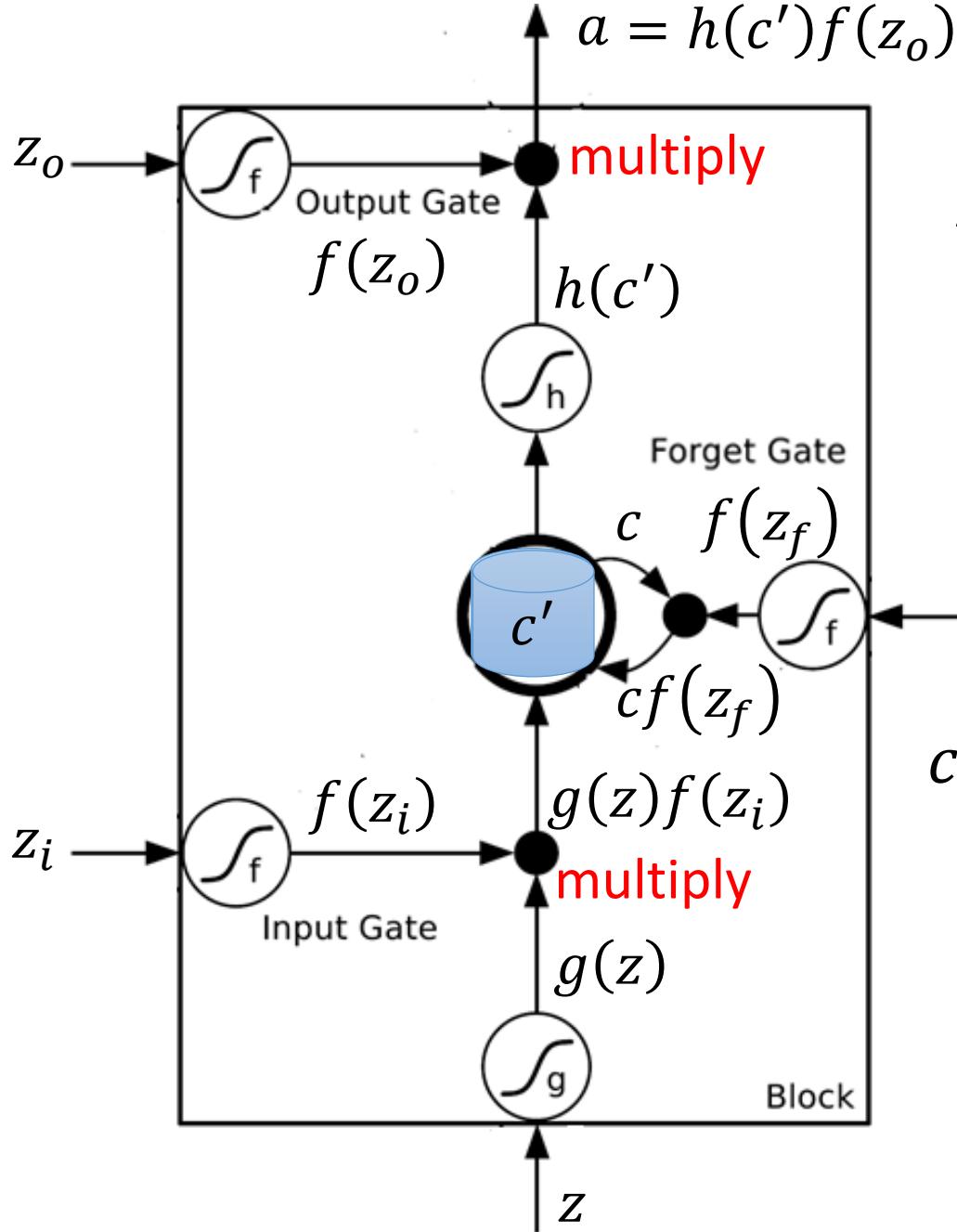


# Bidirectional RNN



# Long Short-term Memory (LSTM)



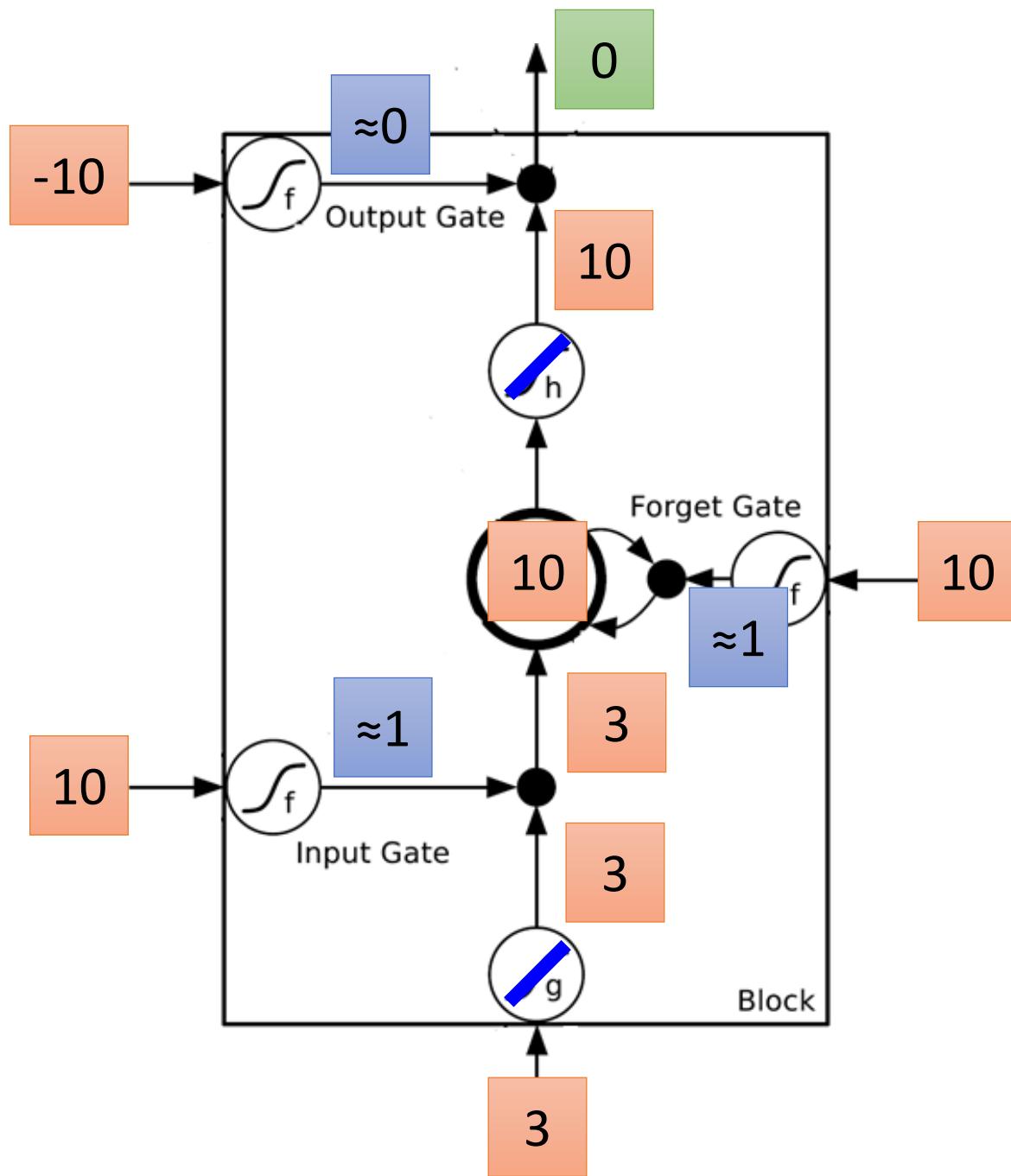


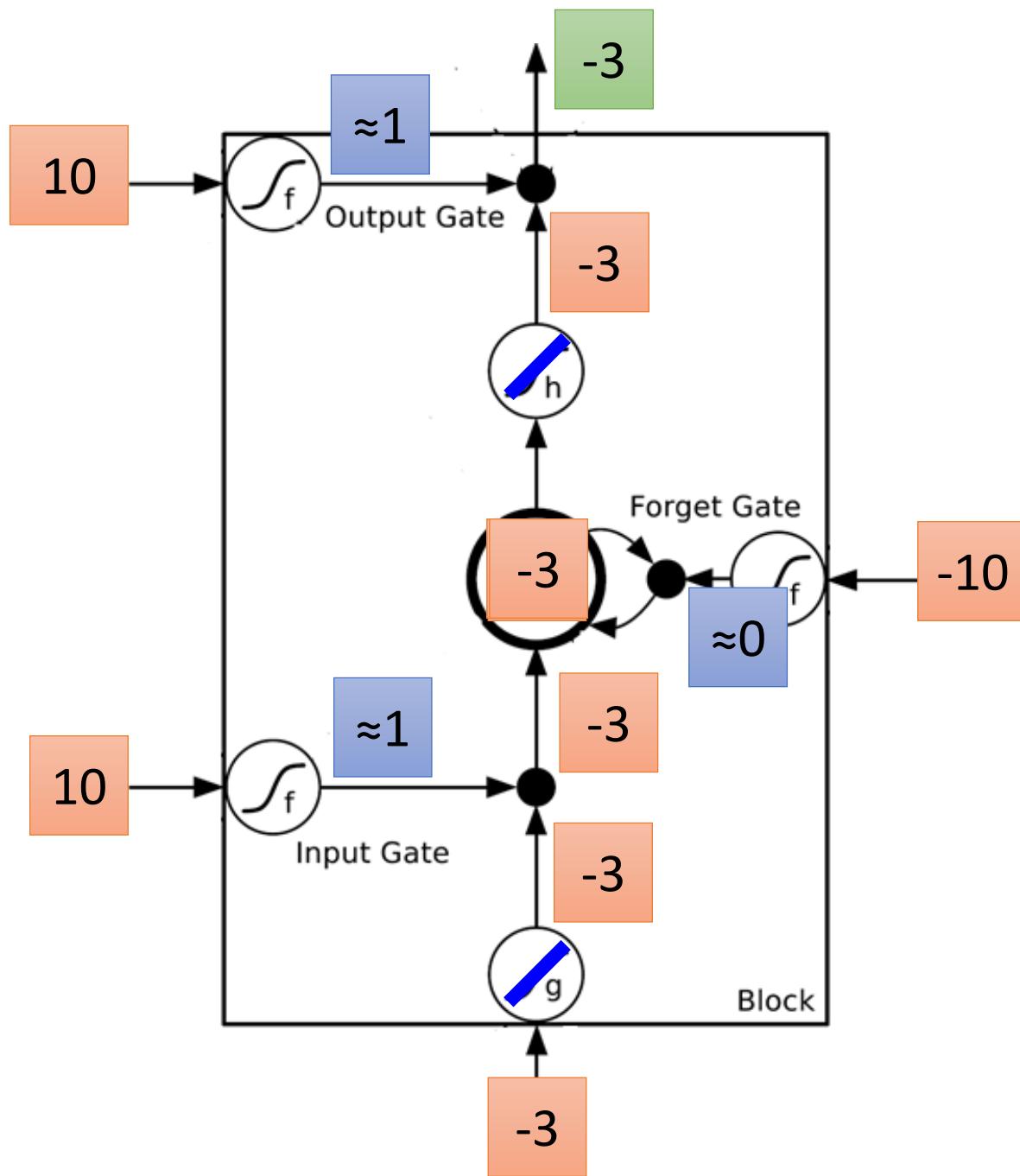
Activation function  $f$  is usually a sigmoid function

Between 0 and 1

Mimic open and close gate

$$c' = g(z)f(z_i) + cf(z_f)$$

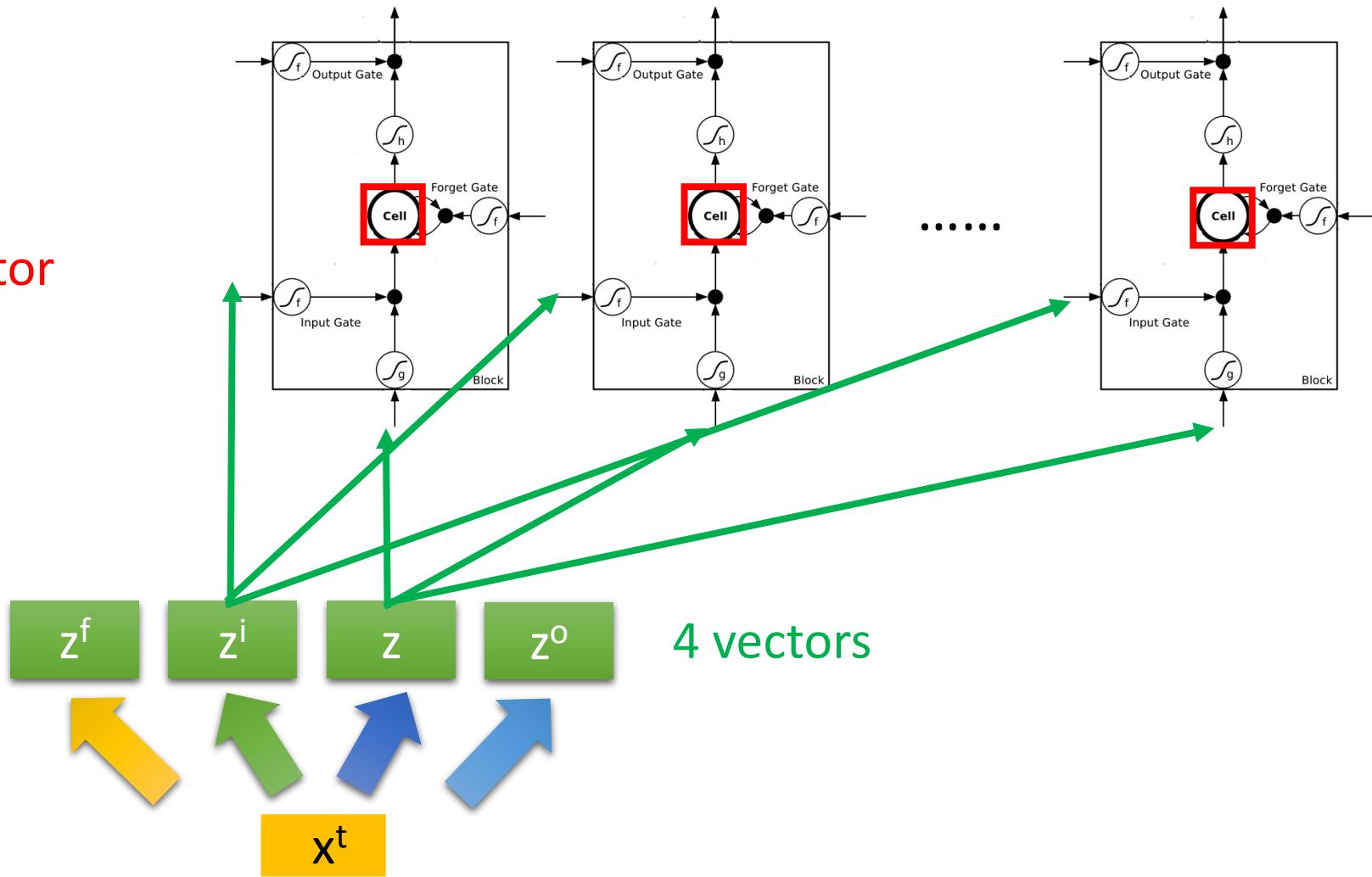




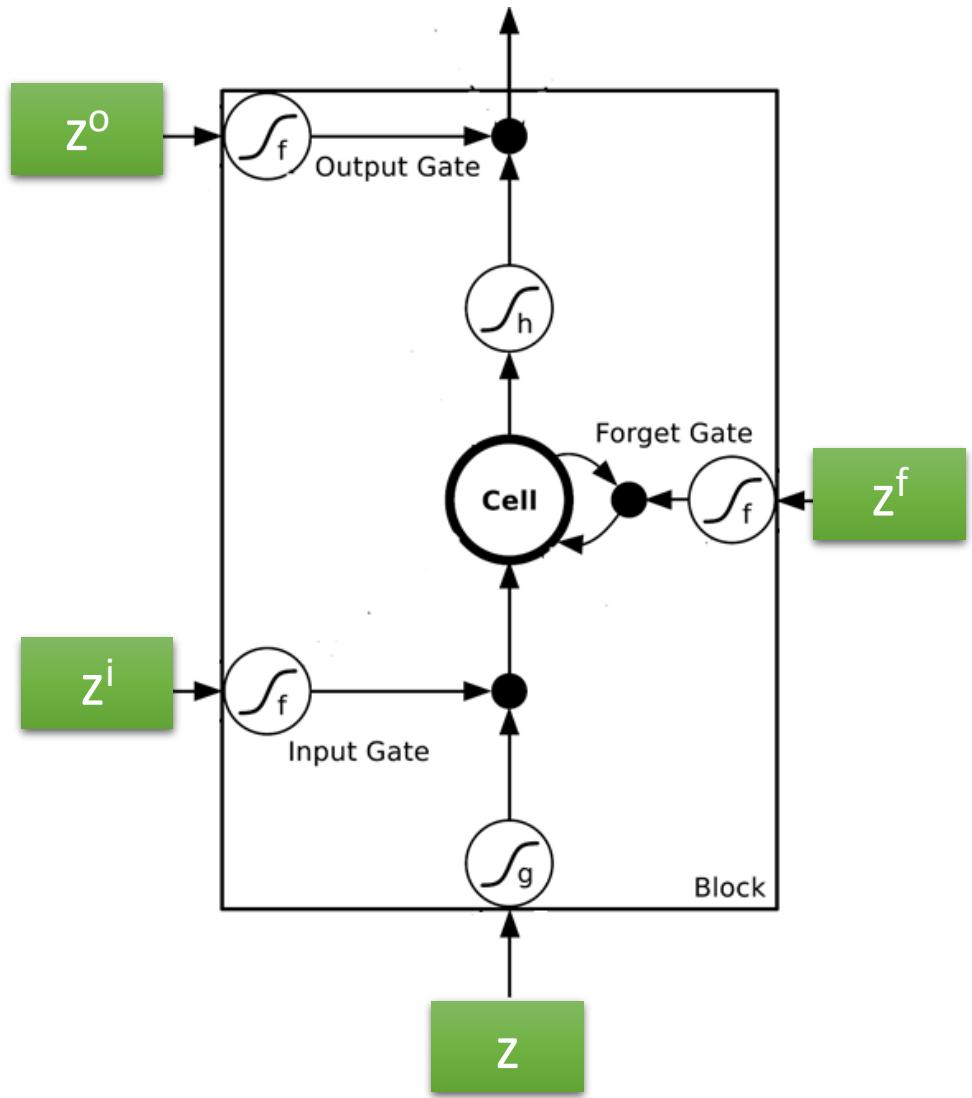
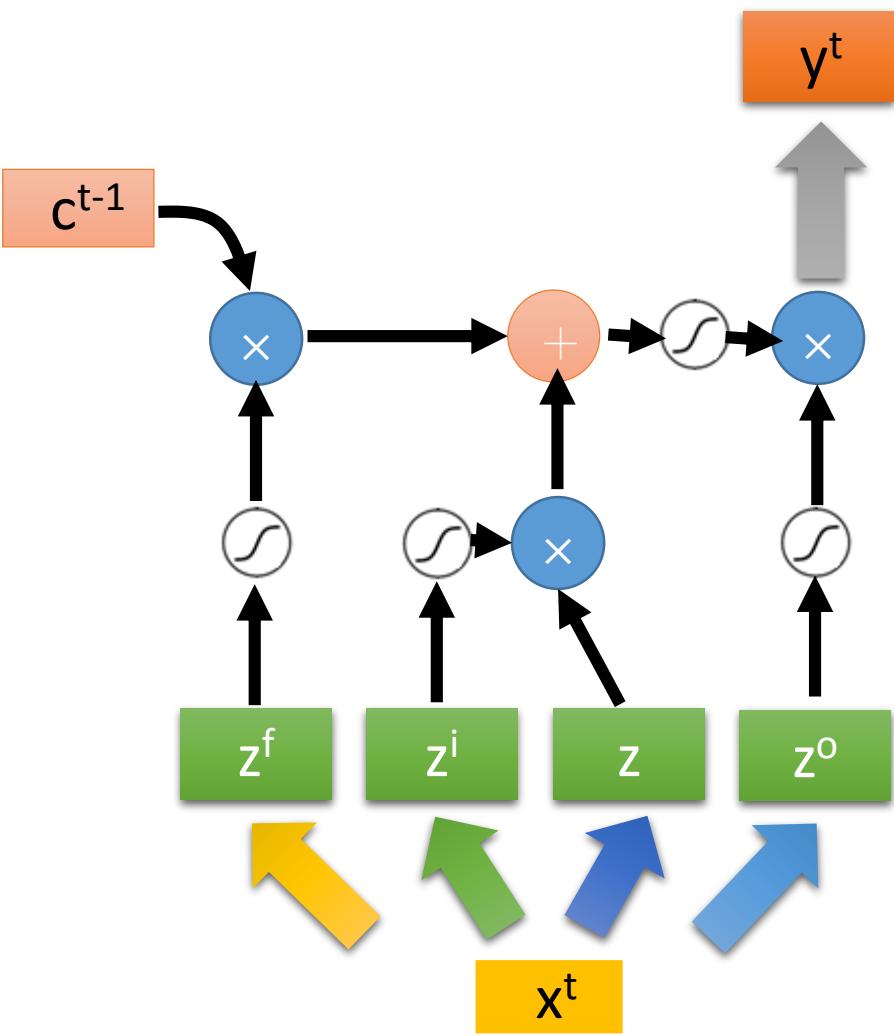
# LSTM

$C^{t-1}$

vector

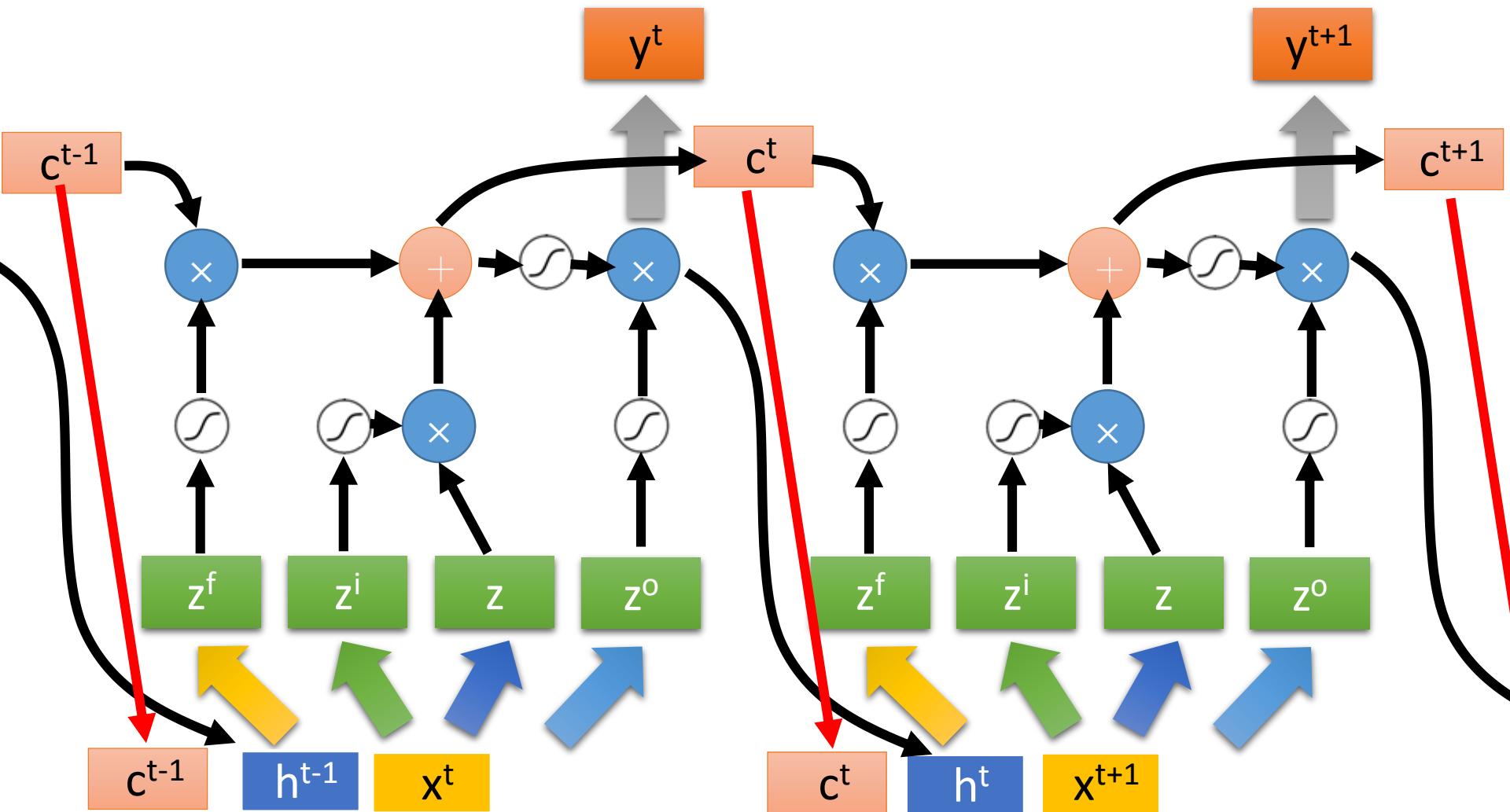


# LSTM

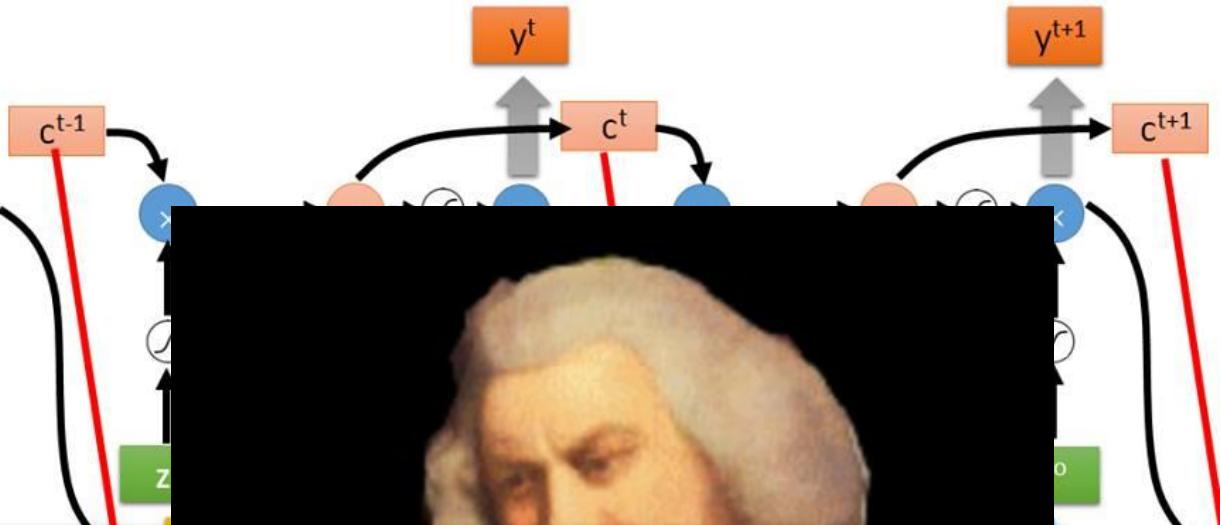


# LSTM

Extension: “peephole”



# Multiple-layer LSTM



Don't worry if you cannot understand this.  
Keras can handle it.

Keras supports  
“LSTM”, “GRU”, “SimpleRNN” layers

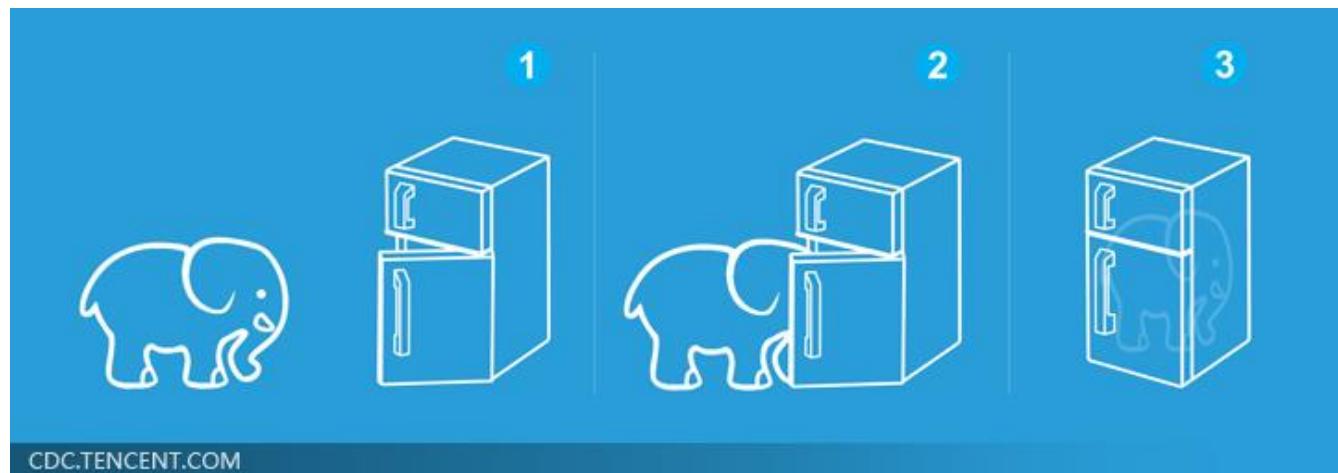
This is quite  
standard now.



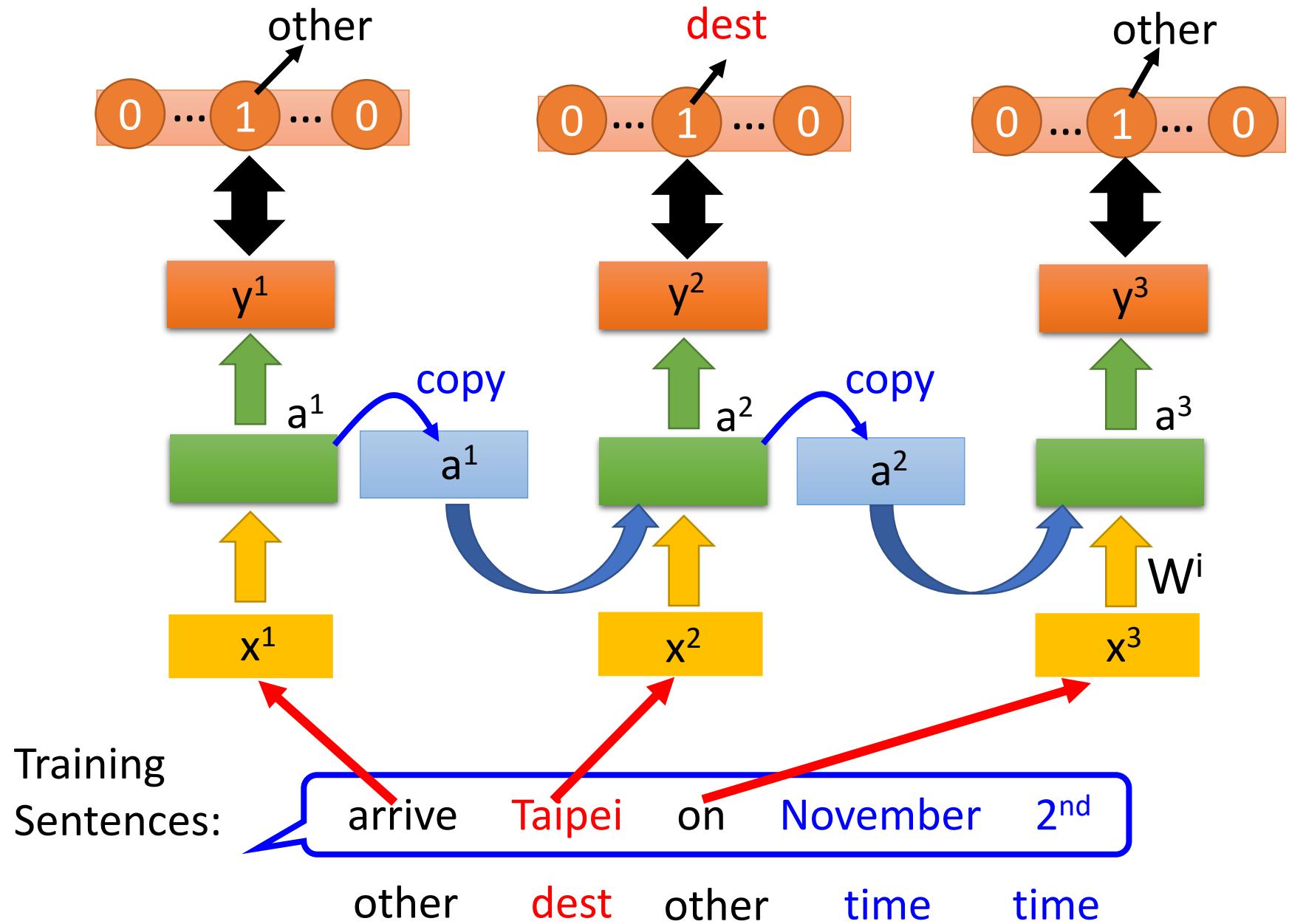
# Three Steps for Deep Learning



Deep Learning is so simple .....



# Learning Target



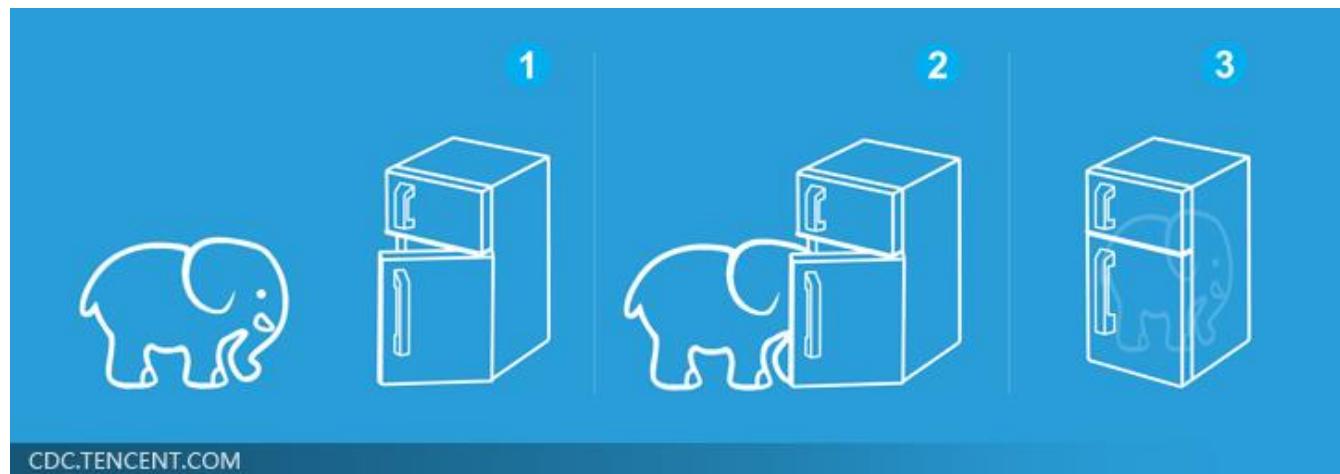
# Three Steps for Deep Learning

Step 1:  
define a set  
of function

Step 2:  
goodness of  
function

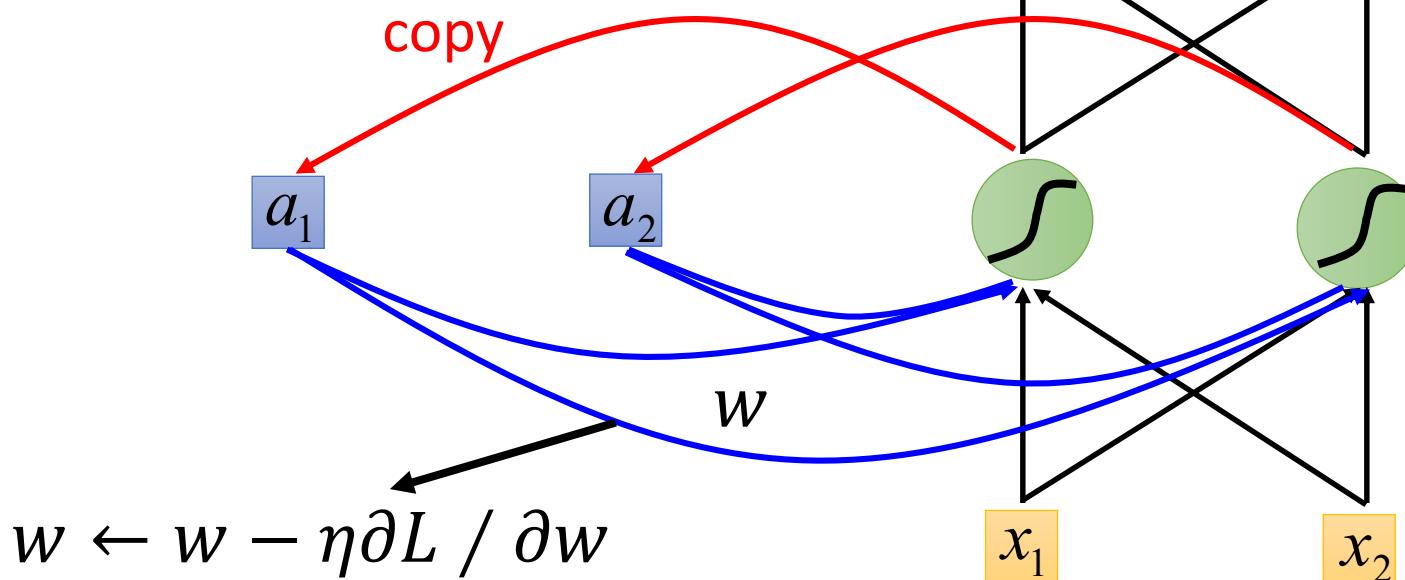
Step 3: pick  
the best  
function

Deep Learning is so simple .....



# Learning

Backpropagation  
through time (BPTT)

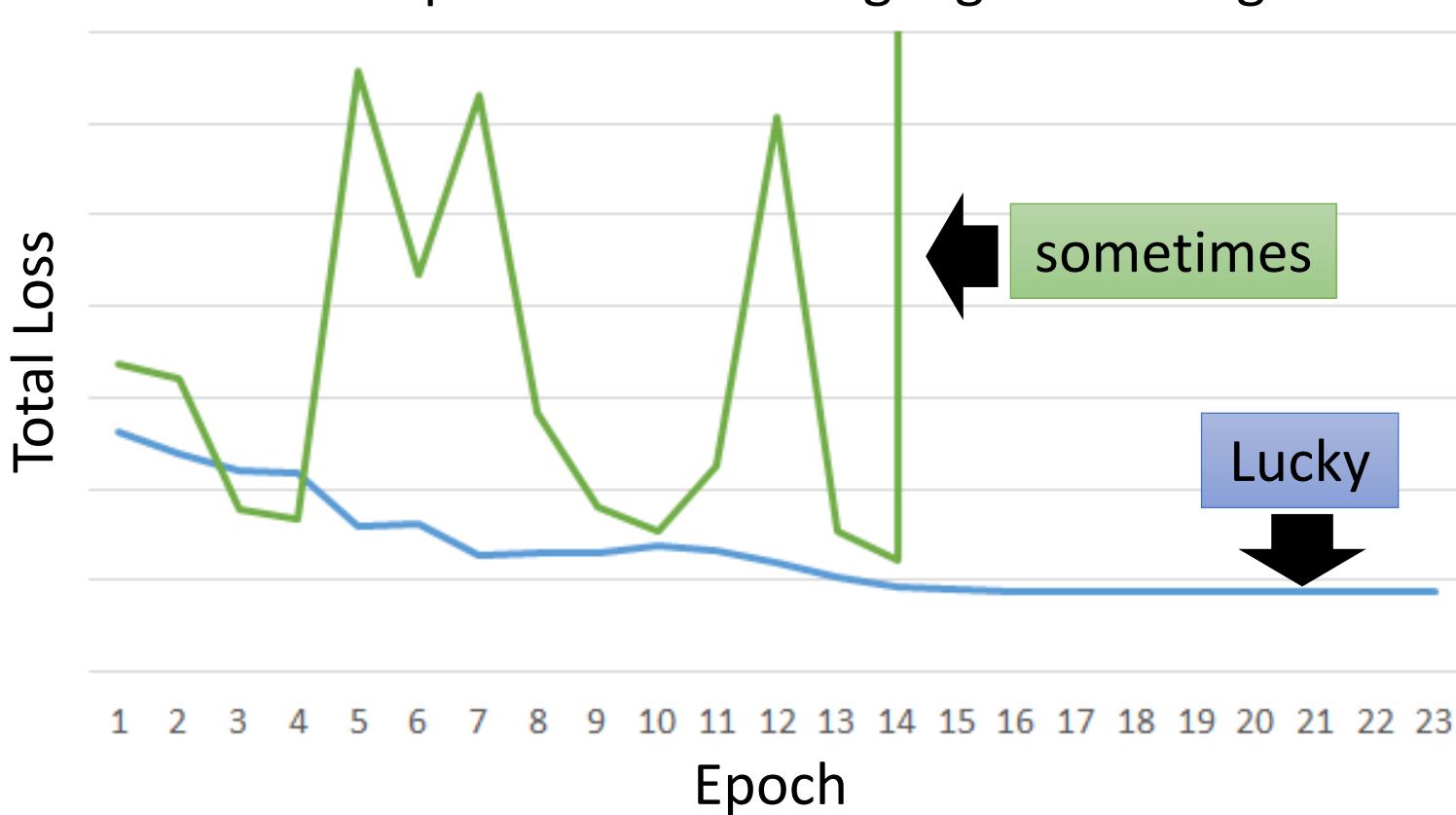


RNN Learning is very difficult in practice.

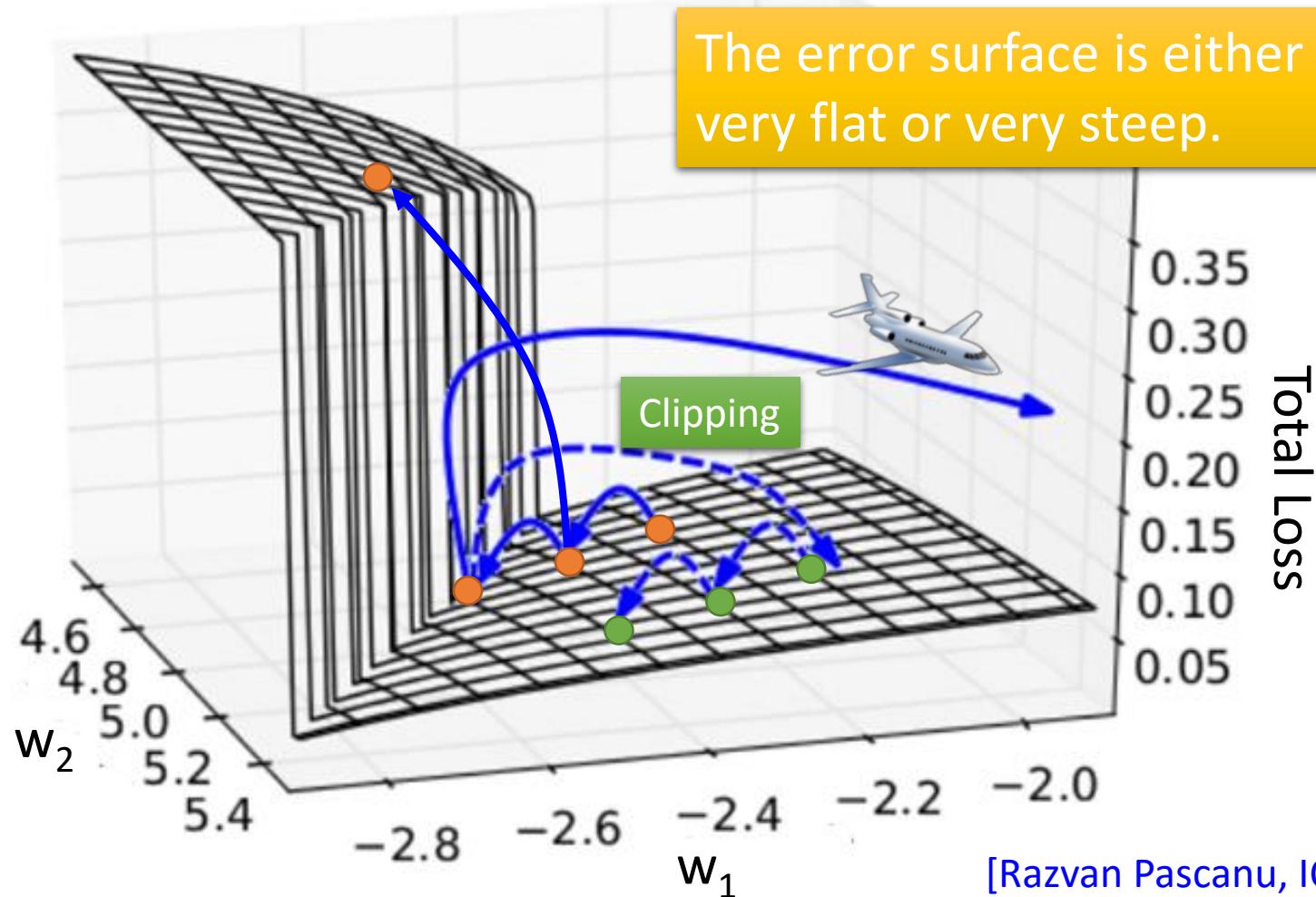
# Unfortunately .....

- RNN-based network is not always easy to learn

Real experiments on Language modeling



# The error surface is rough.



# Why?

$$w = 1 \quad \rightarrow \quad y^{1000} = 1$$

$$w = 1.01 \quad \rightarrow \quad y^{1000} \approx 20000$$

$$w = 0.99 \quad \rightarrow \quad y^{1000} \approx 0$$

$$w = 0.01 \quad \rightarrow \quad y^{1000} \approx 0$$

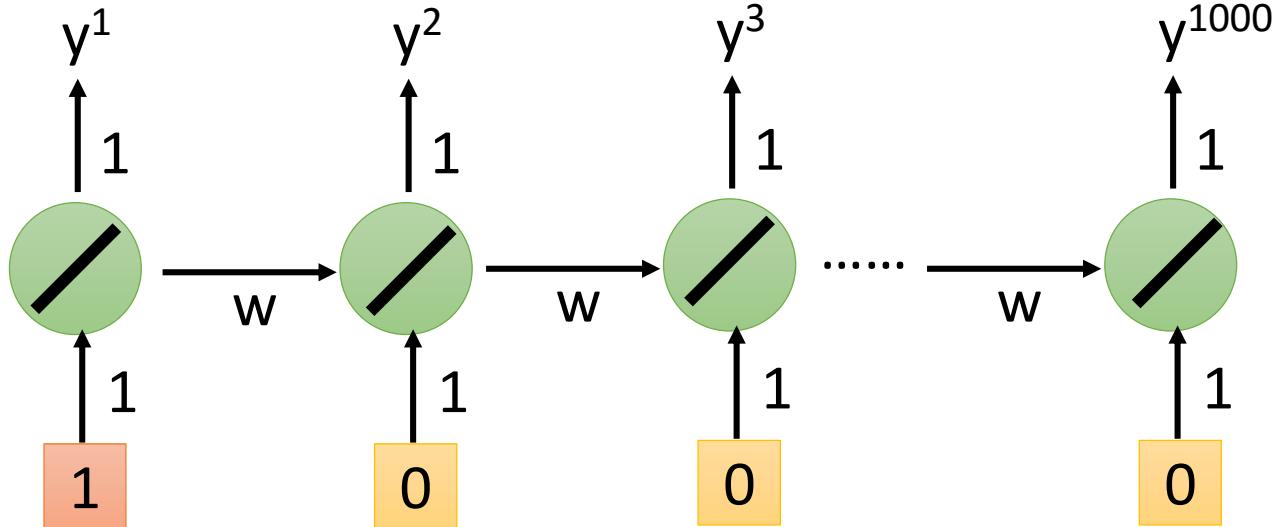
Large  
 $\partial L / \partial w$

Small  
Learning rate?

small  
 $\partial L / \partial w$

Large  
Learning rate?

## Toy Example

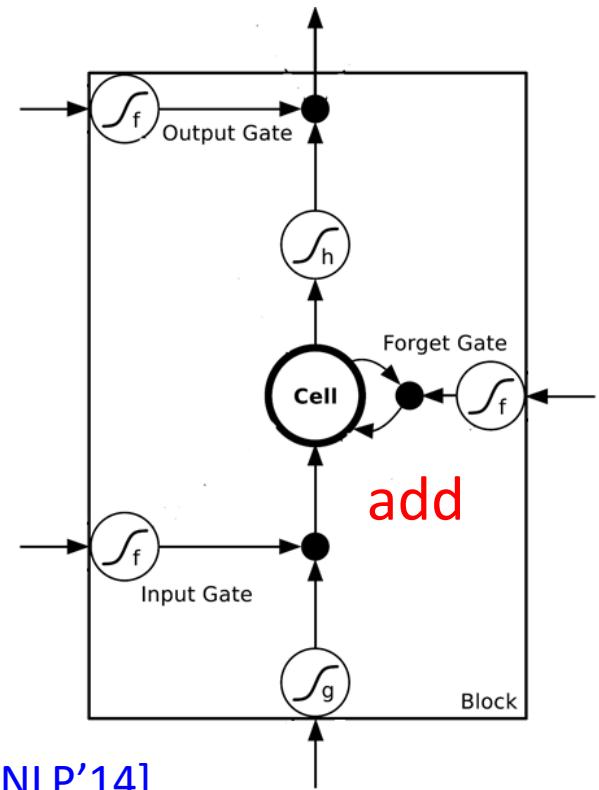


# Helpful Techniques

- Long Short-term Memory (LSTM)
  - Can deal with gradient vanishing (not gradient explode)
  - Memory and input are added
  - The influence never disappears unless forget gate is closed
  - No Gradient vanishing  
(If forget gate is opened.)

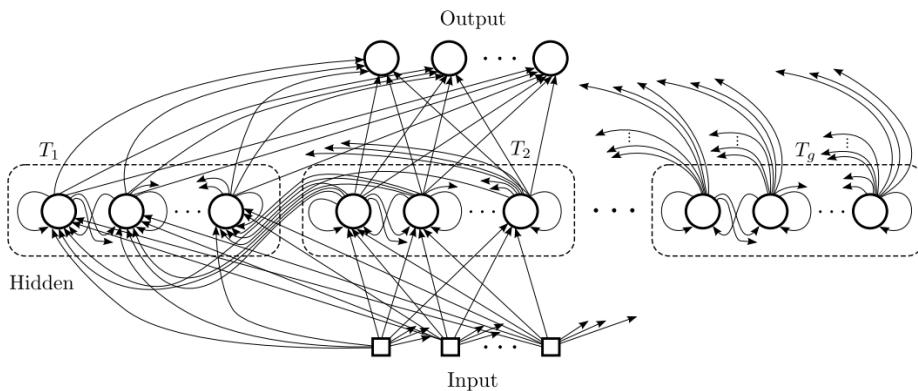
Gated Recurrent Unit (GRU):  
simpler than LSTM

[Cho, EMNLP'14]



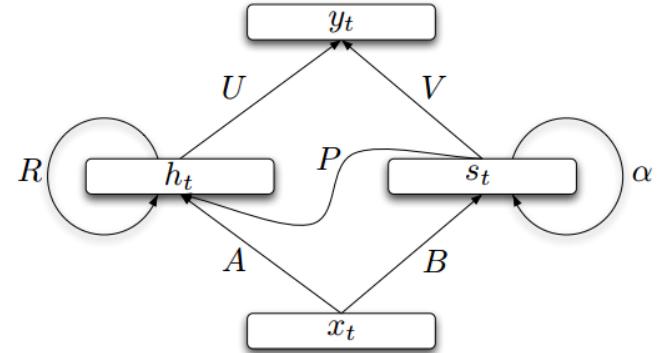
# Helpful Techniques

## Clockwise RNN



[Jan Koutnik, JMLR'14]

## Structurally Constrained Recurrent Network (SCRN)



[Tomas Mikolov, ICLR'15]

Vanilla RNN Initialized with Identity matrix + ReLU activation function [Quoc V. Le, arXiv'15]

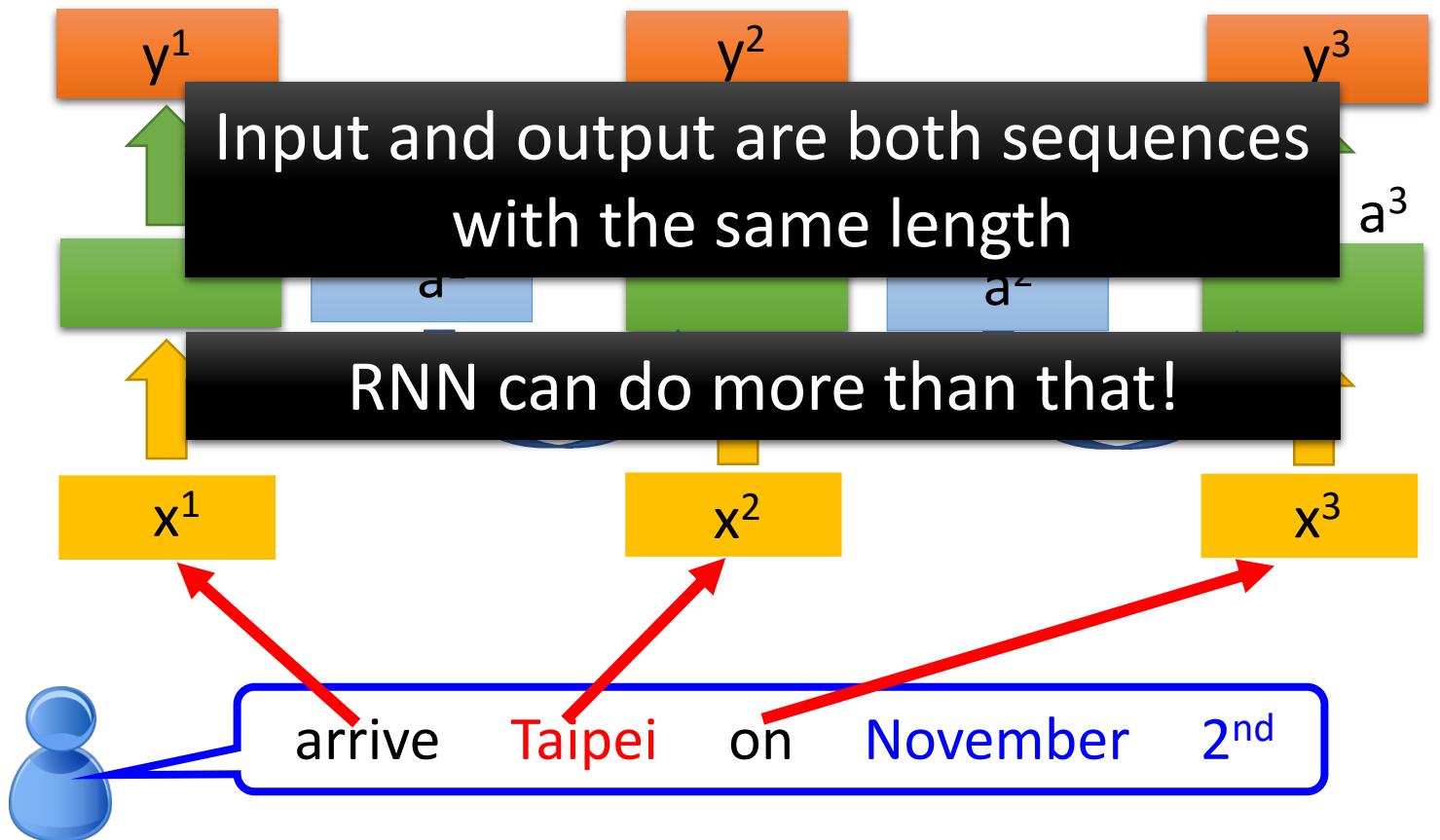
- Outperform or be comparable with LSTM in 4 different tasks

# More Applications .....

Probability of  
“arrive” in each slot

Probability of  
“Taipei” in each slot

Probability of  
“on” in each slot



# Many to one

Keras Example:

[https://github.com/fchollet/keras/blob/master/examples/imdb\\_lstm.py](https://github.com/fchollet/keras/blob/master/examples/imdb_lstm.py)

- Input is a vector sequence, but output is only one vector

## Sentiment Analysis

看了這部電影覺  
得很高興 .....

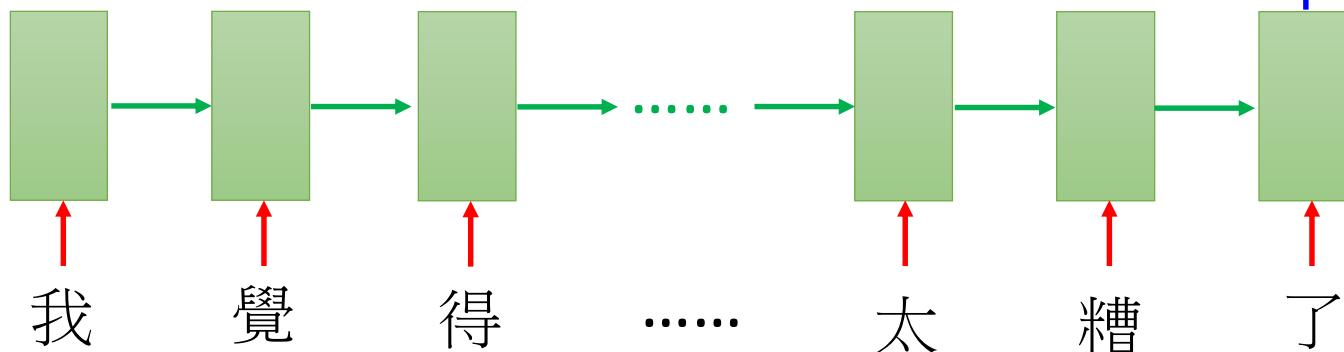
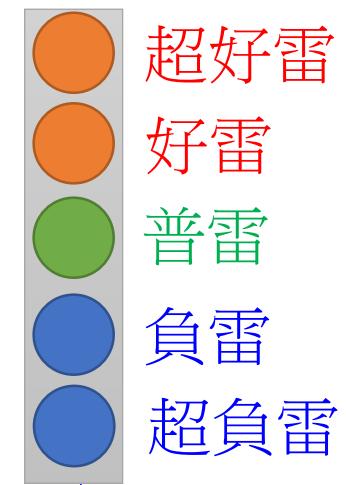
Positive (正雷)

這部電影太糟了  
.....

Negative (負雷)

這部電影很  
棒 .....

Positive (正雷)



# Many to Many (Output is shorter)

- Both input and output are both sequences, **but the output is shorter.**
  - E.g. **Speech Recognition**

Problem?

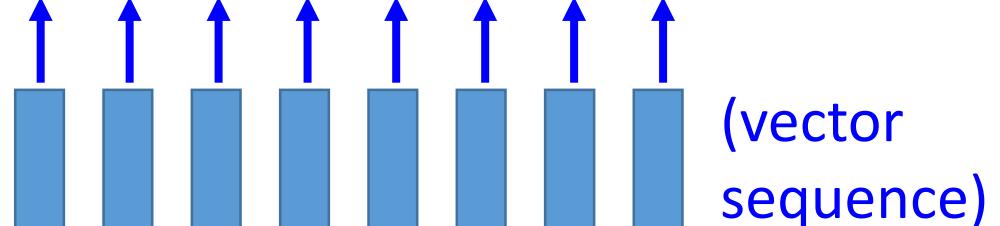
Why can't it be  
“好棒棒”

Output: “好棒” (character sequence)



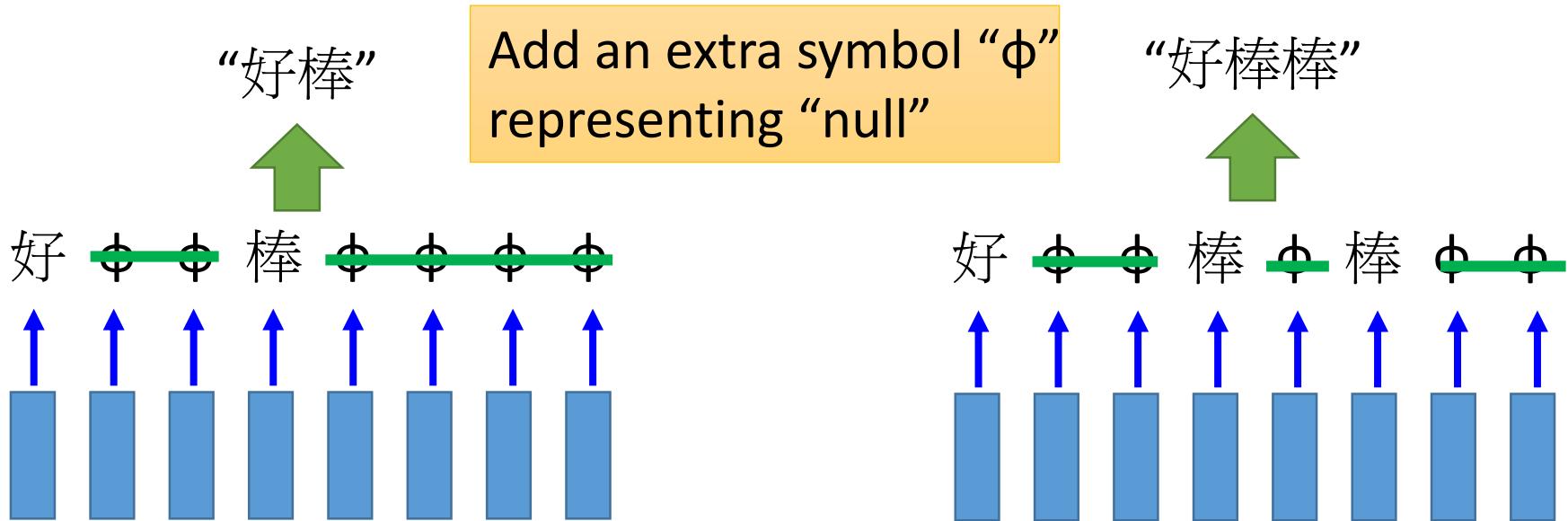
好 好 好 棒 棒 棒 棒 棒

Input:



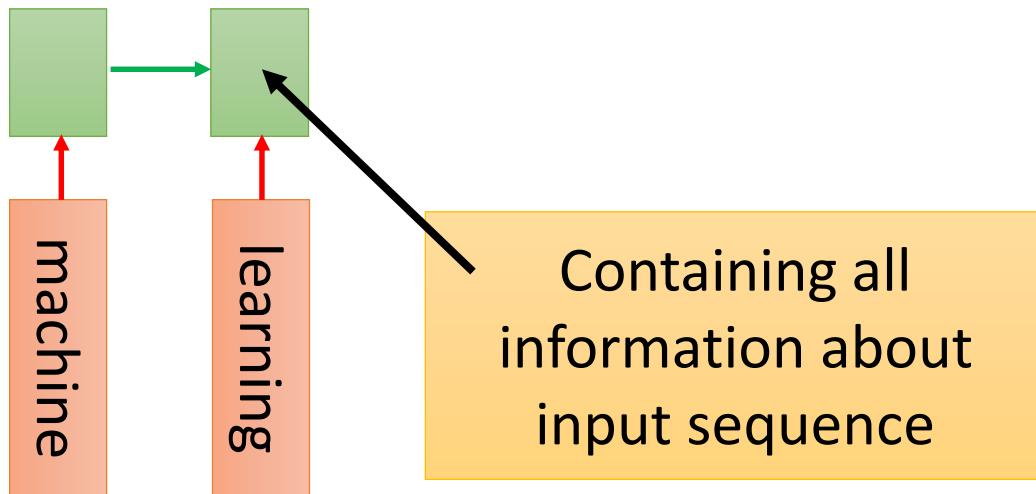
# Many to Many (Output is shorter)

- Both input and output are both sequences, **but the output is shorter.**
- Connectionist Temporal Classification (CTC) [Alex Graves, ICML'06][Alex Graves, ICML'14][Hasim Sak, Interspeech'15][Jie Li, Interspeech'15][Andrew Senior, ASRU'15]



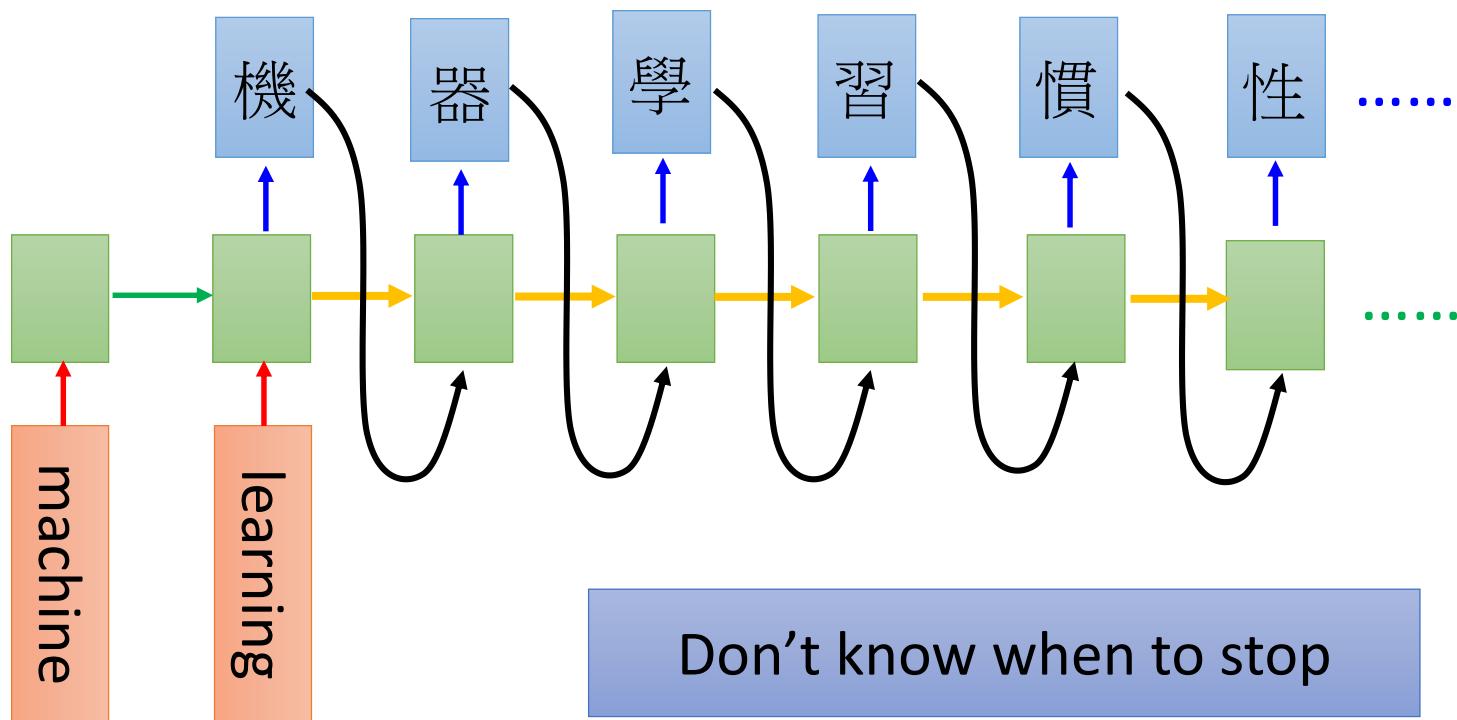
# Many to Many (No Limitation)

- Both input and output are both sequences with different lengths. → Sequence to sequence learning
  - E.g. Machine Translation (machine learning → 機器學習)



# Many to Many (No Limitation)

- Both input and output are both sequences with different lengths. → Sequence to sequence learning
  - E.g. Machine Translation (machine learning → 機器學習)



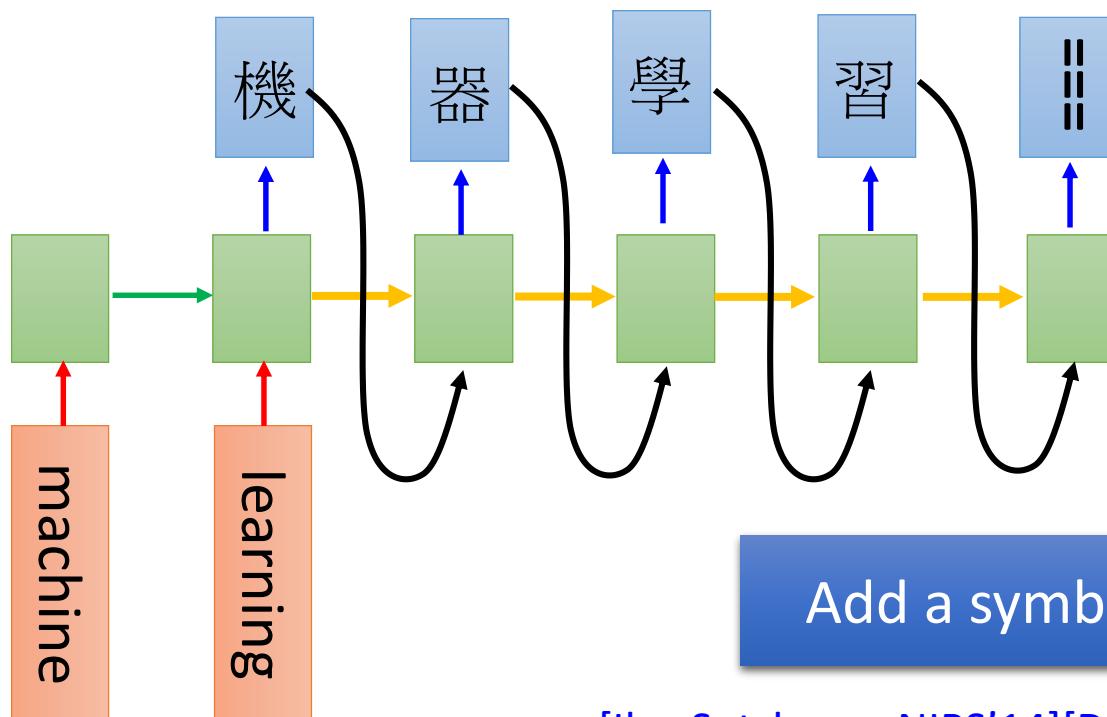
# Many to Many (No Limitation)

推	: 超	06/12 10:39
推	: n: 人	06/12 10:40
推	: tion: 正	06/12 10:41
→	: host: 大	06/12 10:47
推	: 中	06/12 10:59
推	: 403: 天	06/12 11:11
推	: 外	06/12 11:13
推	: 527: 飛	06/12 11:17
→	: 990b: 仙	06/12 11:32
→	: 512: 草	06/12 12:15

推 tlkagk: =====斷=====

# Many to Many (No Limitation)

- Both input and output are both sequences with different lengths. → Sequence to sequence learning
  - E.g. Machine Translation (machine learning→機器學習)



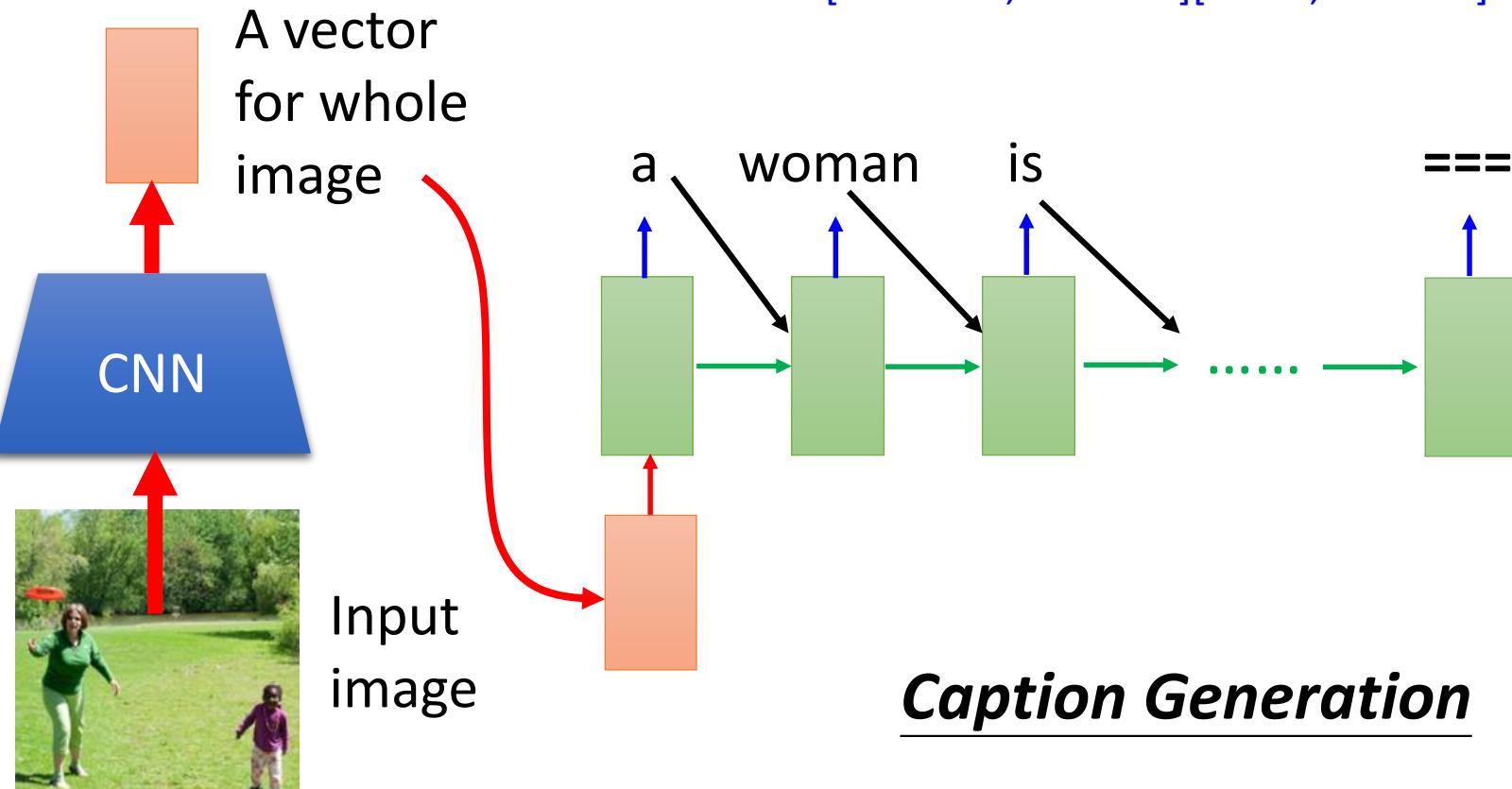
Add a symbol “==” (斷)

[Ilya Sutskever, NIPS'14][Dzmitry Bahdanau, arXiv'15]

# One to Many

- Input an image, but output a sequence of words

[Kelvin Xu, arXiv'15][Li Yao, ICCV'15]



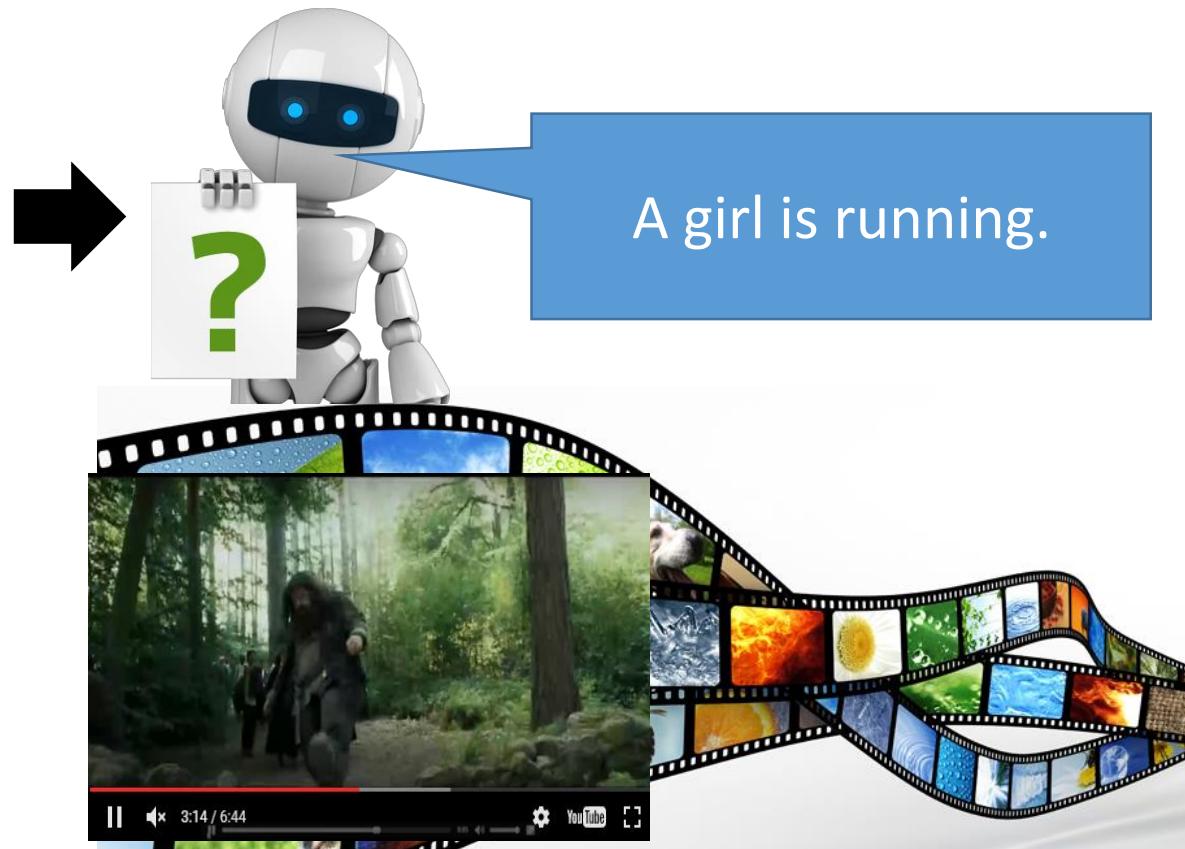
# Application: Video Caption Generation



Video



A group of people is  
knocked by a tree.



A group of people is  
walking in the forest.

# Video Caption Generation

- Can machine describe what it see from video?
- Demo: 曾柏翔、吳柏瑜、盧宏宗

# Concluding Remarks

Convolutional Neural  
Network (CNN)

Recurrent Neural Network  
(RNN)

# Lecture IV: Next Wave

# Outline

## Supervised Learning

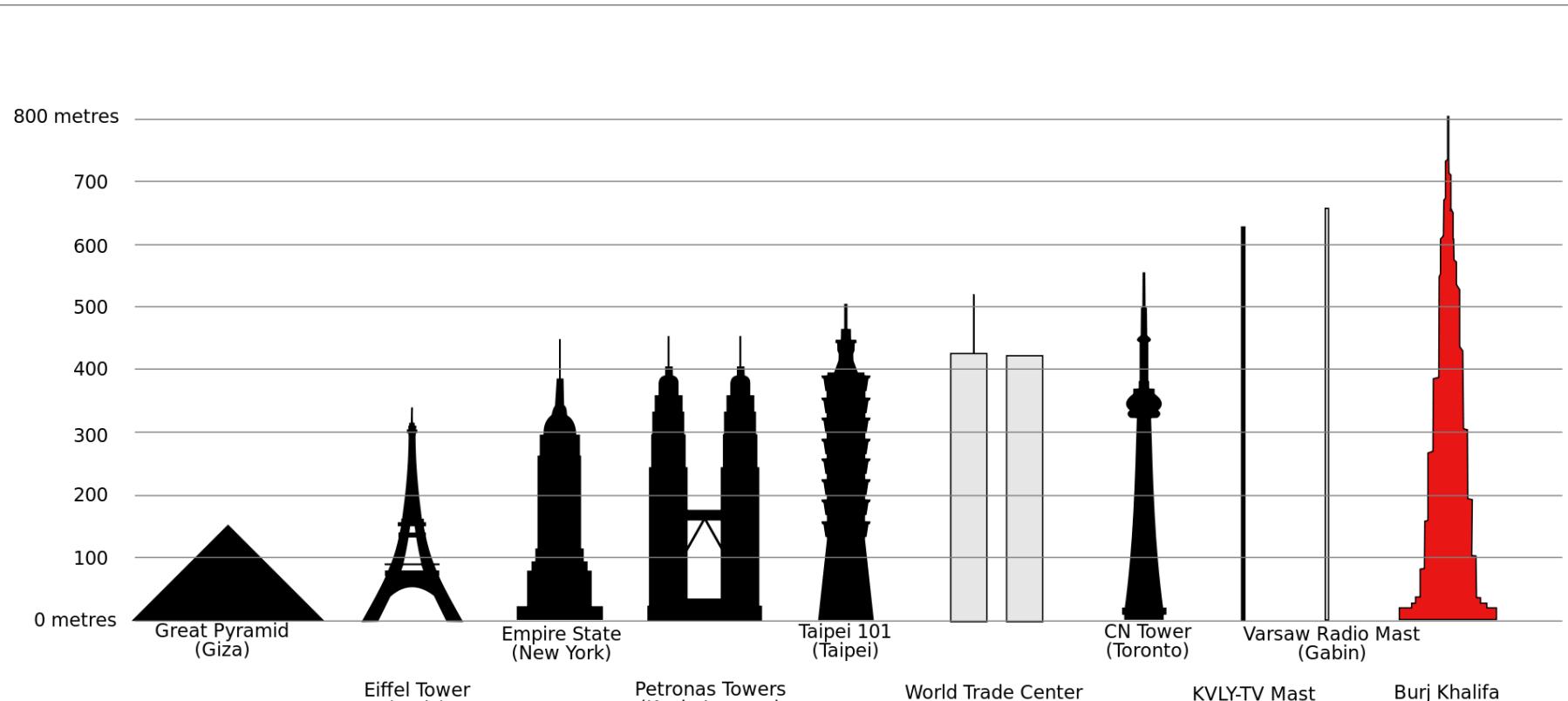
- Ultra Deep Network
  - Attention Model
- }
- New network structure

## Reinforcement Learning

## Unsupervised Learning

- Image: Realizing what the World Looks Like
- Text: Understanding the Meaning of Words
- Audio: Learning human language without supervision

# Skyscraper

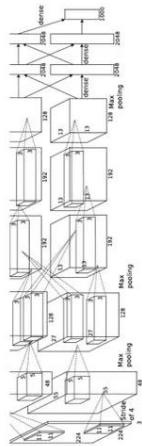


<https://zh.wikipedia.org/wiki/%E9%9B%99%E5%B3%B0%E5%A1%94#media/File:BurjDubaiHeight.svg>

# Ultra Deep Network

[http://cs231n.stanford.edu/slides/winter1516\\_lecuture8.pdf](http://cs231n.stanford.edu/slides/winter1516_lecuture8.pdf)

16.4%



AlexNet (2012)

8 layers

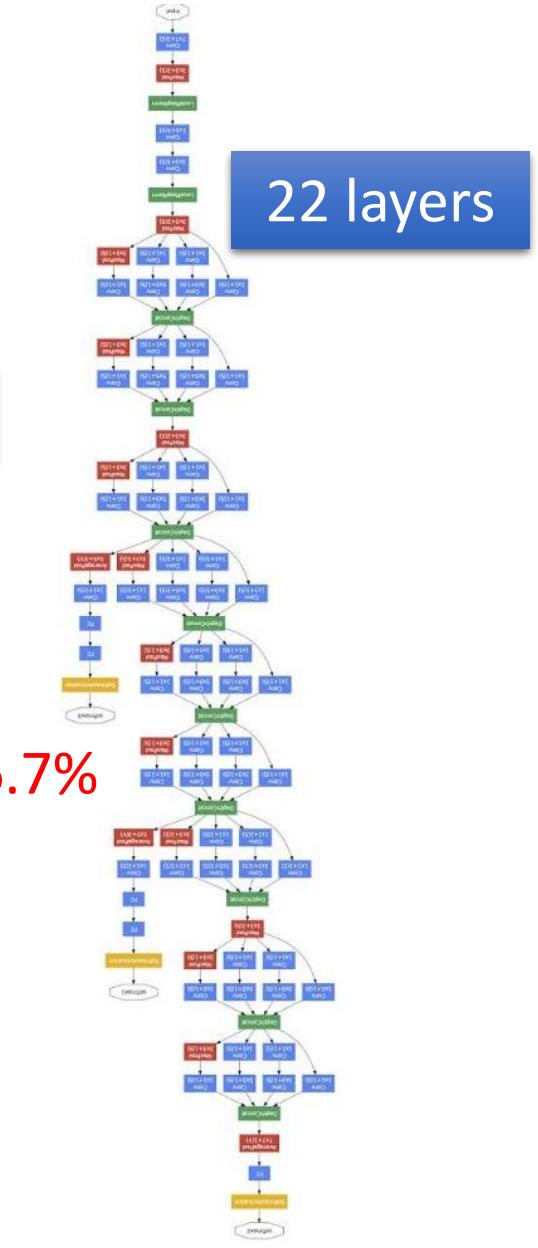
7.3%



VGG (2014)

19 layers

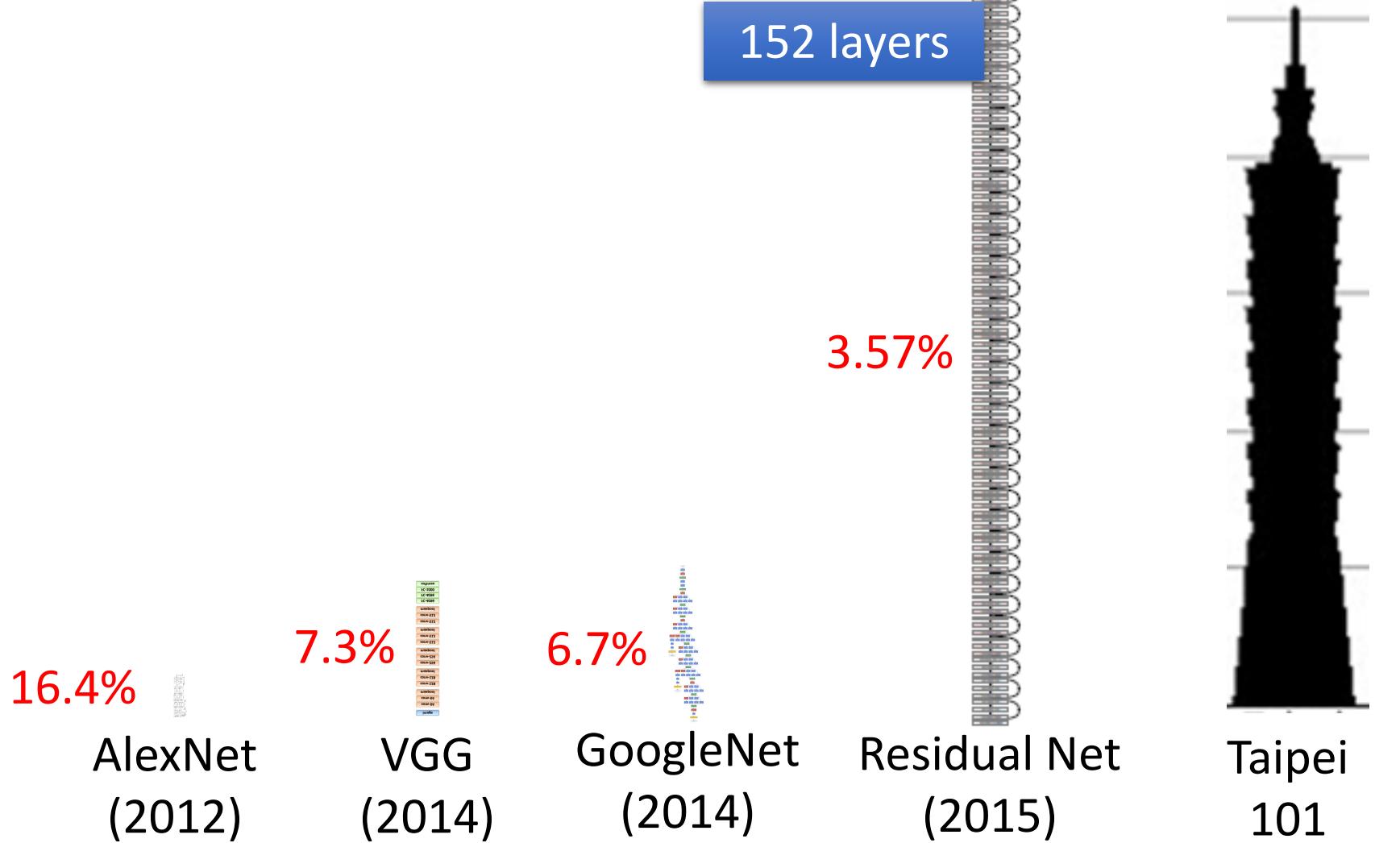
6.7%



GoogleNet (2014)

22 layers

# Ultra Deep Network



# Ultra Deep Network

Worry about overfitting?

152 layers

Worry about training  
first!

This ultra deep network  
have special structure.

3.57%

16.4%



AlexNet  
(2012)

7.3%



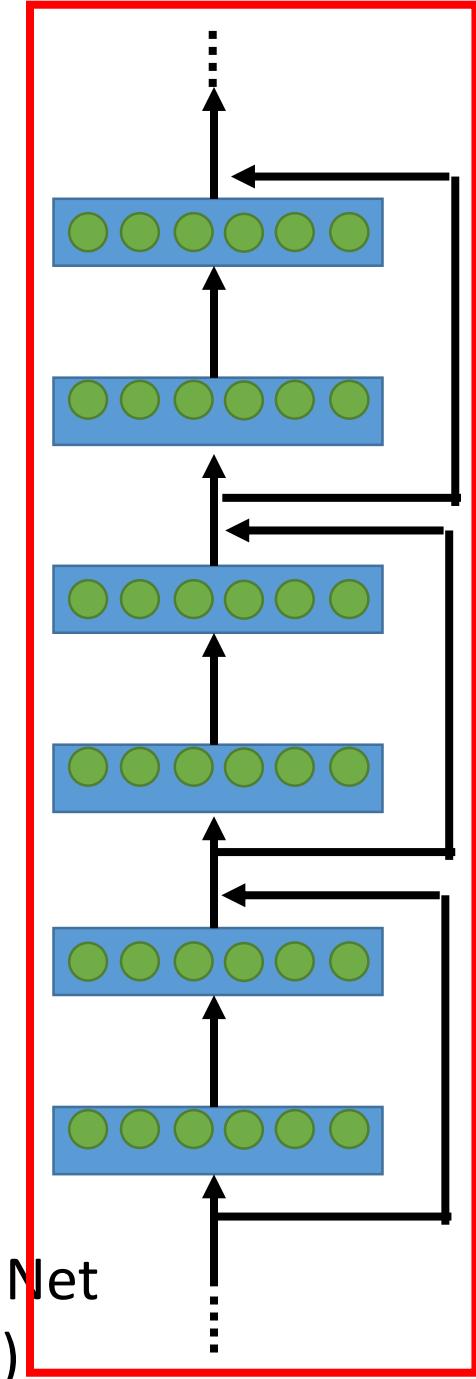
VGG  
(2014)

6.7%



GoogleNet  
(2014)

Residual Net  
(2015)



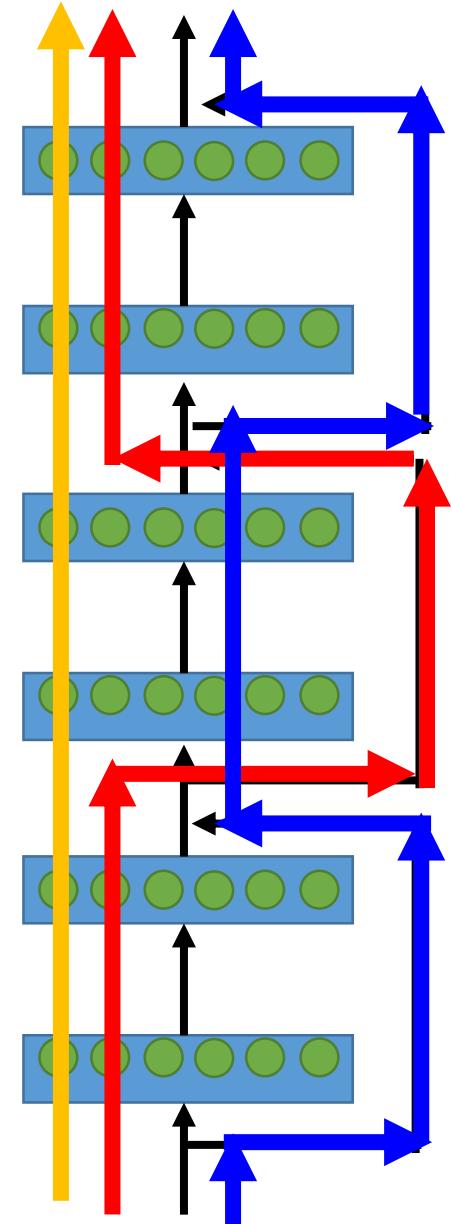
# Ultra Deep Network

- Ultra deep network is the ensemble of many networks with different depth.

Ensemble

6 layers  
4 layers  
2 layers

**Residual Networks are Exponential Ensembles of Relatively Shallow Networks**  
<https://arxiv.org/abs/1605.06431>



# Ultra Deep Network

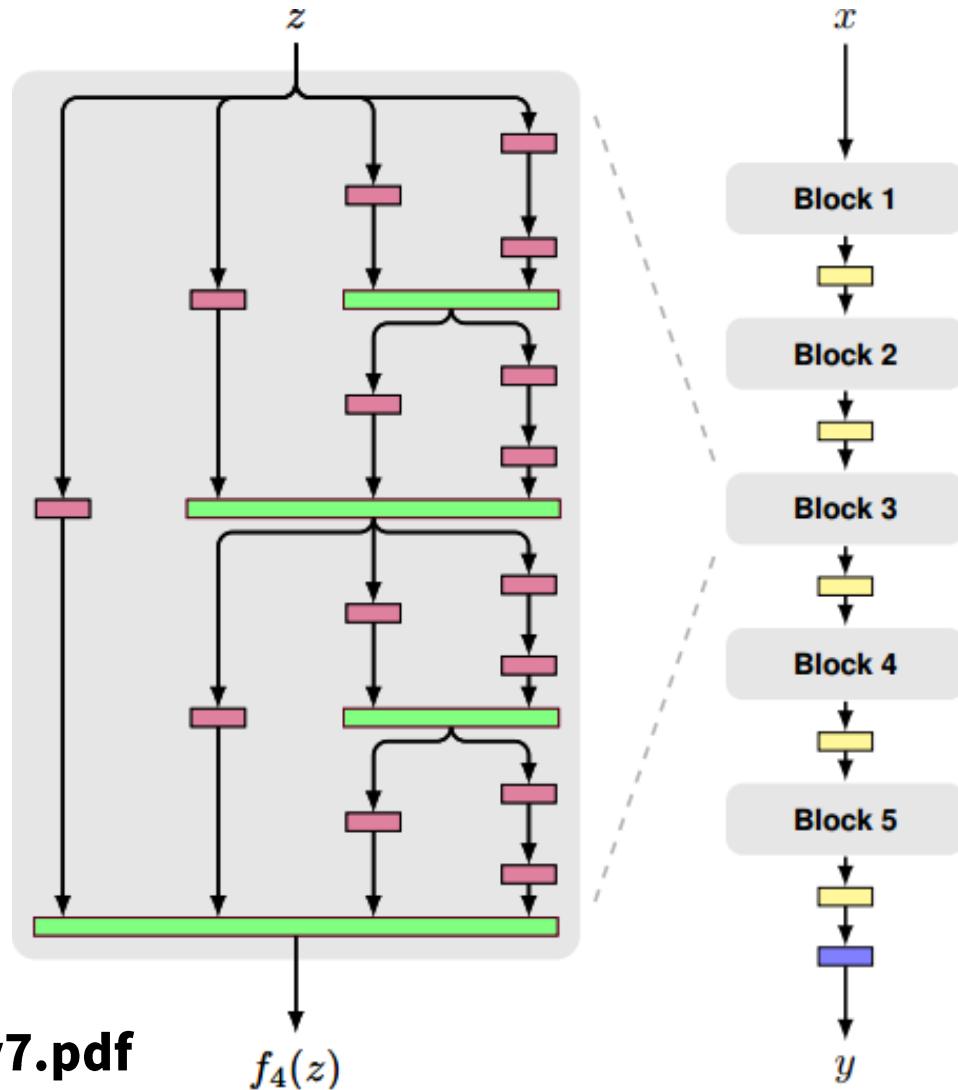
- FractalNet

**FractalNet: Ultra-Deep Neural Networks without Residuals**  
<https://arxiv.org/abs/1605.07648>  
Resnet in Resnet

**Resnet in Resnet: Generalizing Residual Architectures**  
<https://arxiv.org/abs/1603.08029>

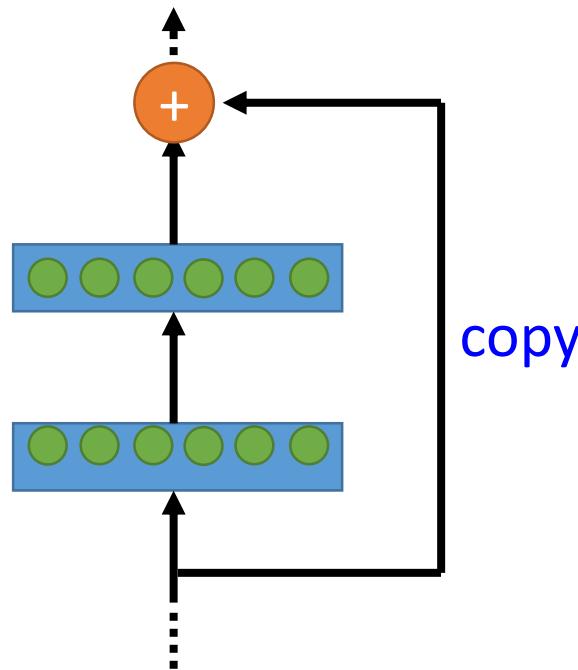
Good Initialization?

**All you need is a good init**  
<http://arxiv.org/pdf/1511.06422v7.pdf>

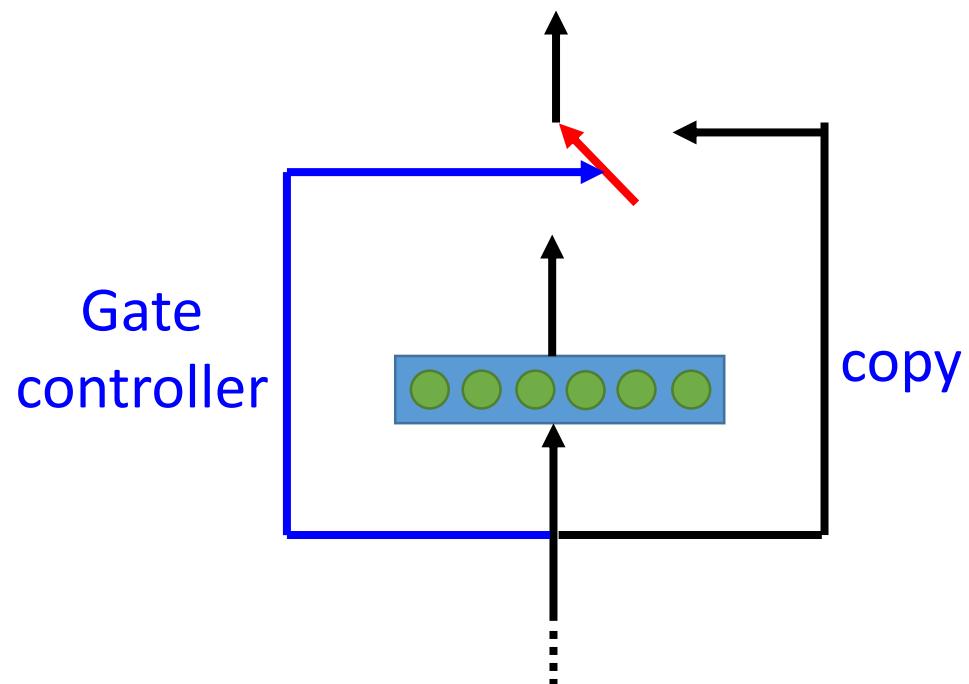


# Ultra Deep Network

- **Residual Network**
- **Highway Network**

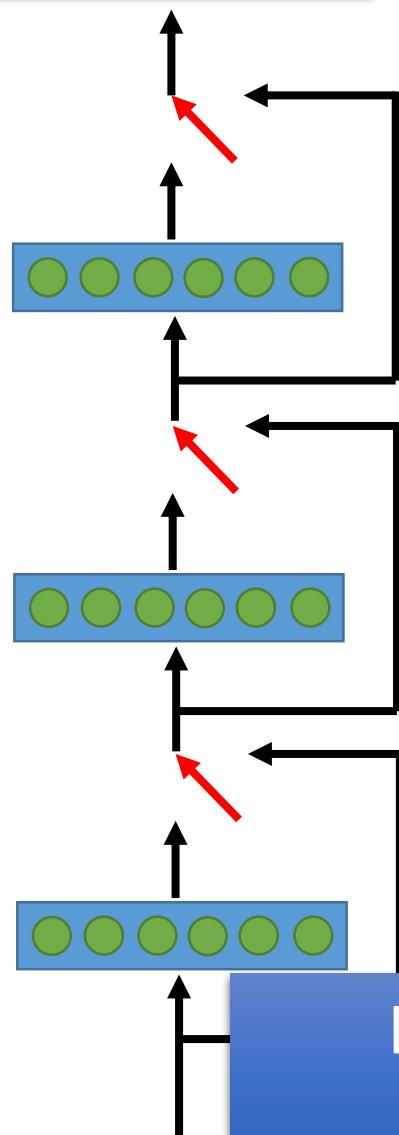


**Deep Residual Learning for Image Recognition**  
<http://arxiv.org/abs/1512.03385>

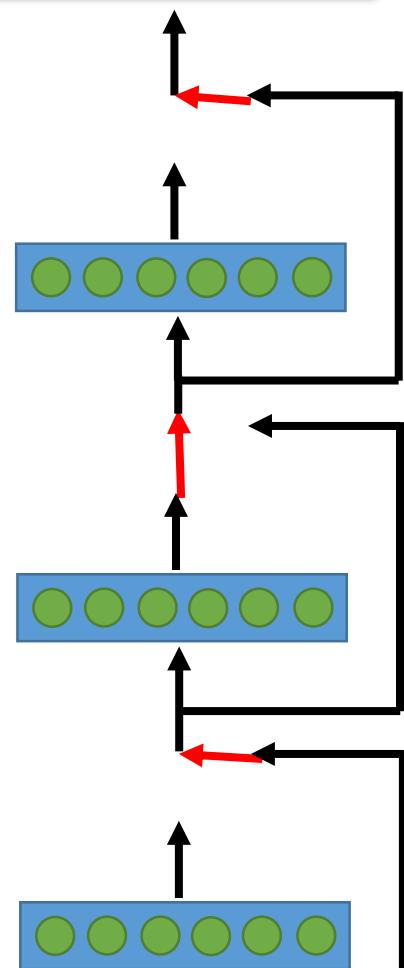


**Training Very Deep Networks**  
<https://arxiv.org/pdf/1507.06228v2.pdf>

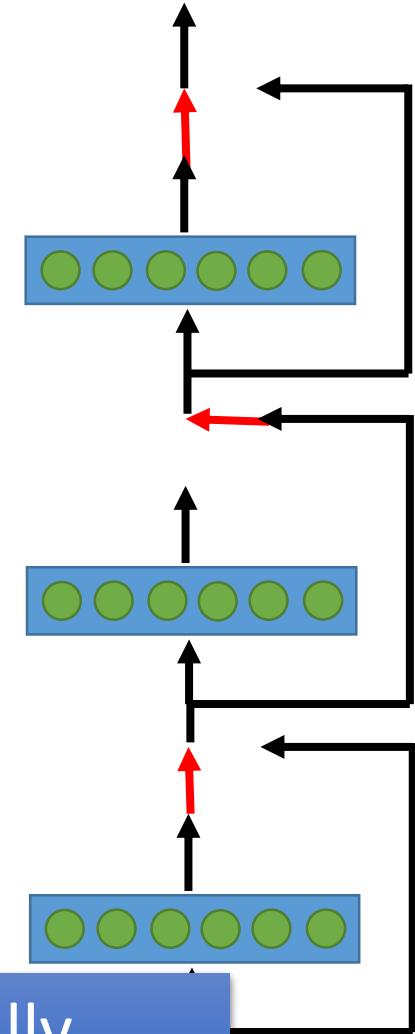
output layer



output layer



output layer



Highway Network automatically  
determines the layers needed!

Input layer

Input layer

Input layer

# Outline

## Supervised Learning

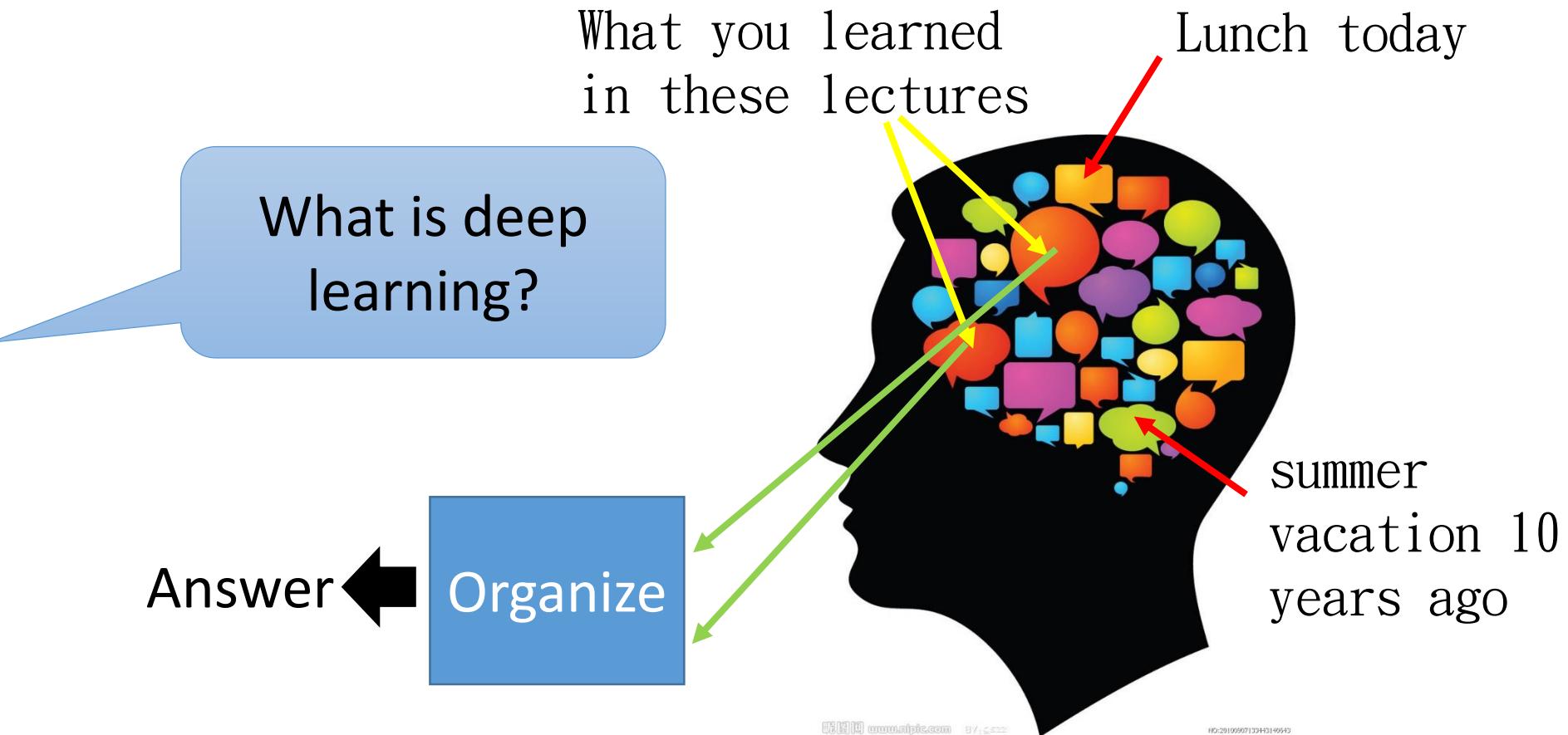
- Ultra Deep Network
  - Attention Model
- }
- New network structure

## Reinforcement Learning

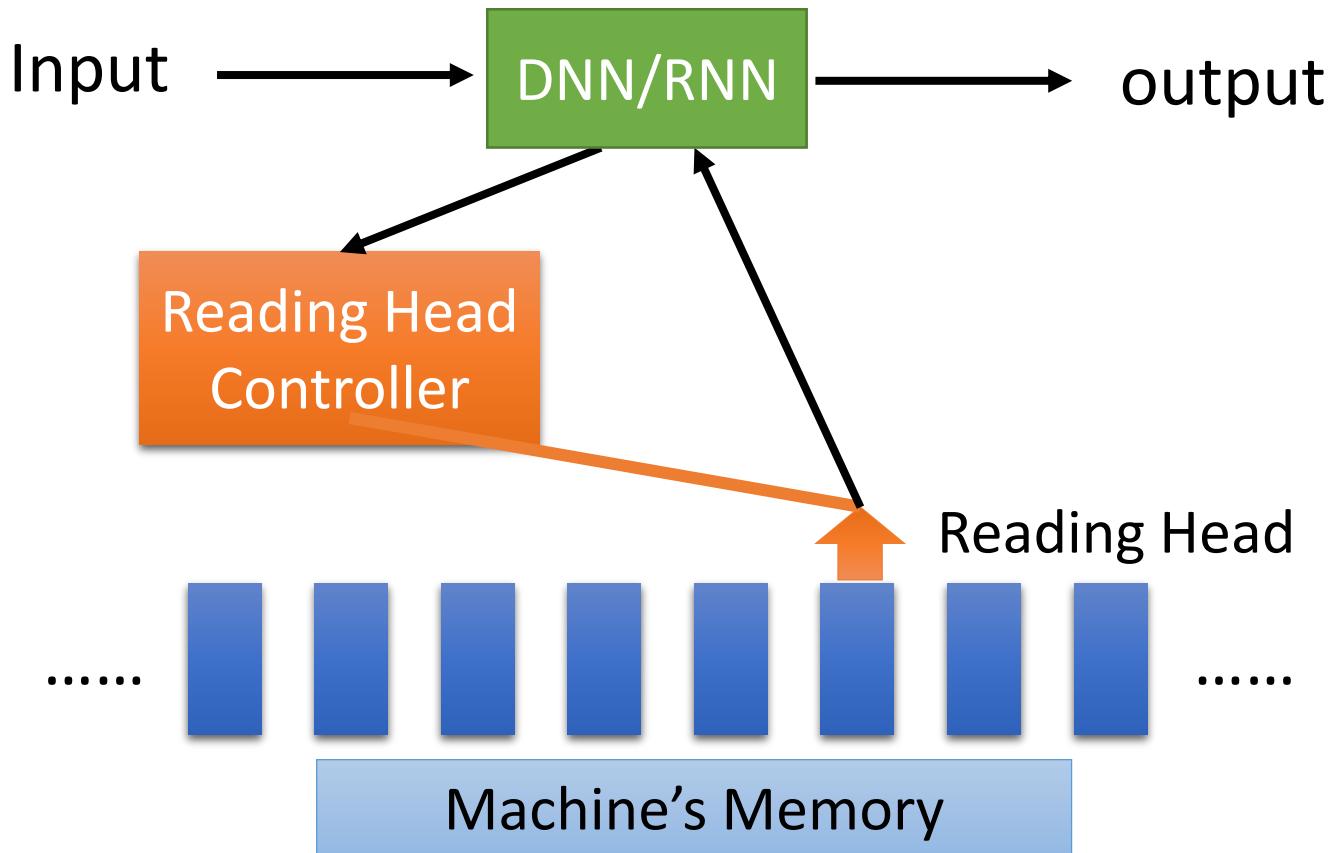
## Unsupervised Learning

- Image: Realizing what the World Looks Like
- Text: Understanding the Meaning of Words
- Audio: Learning human language without supervision

# Attention-based Model



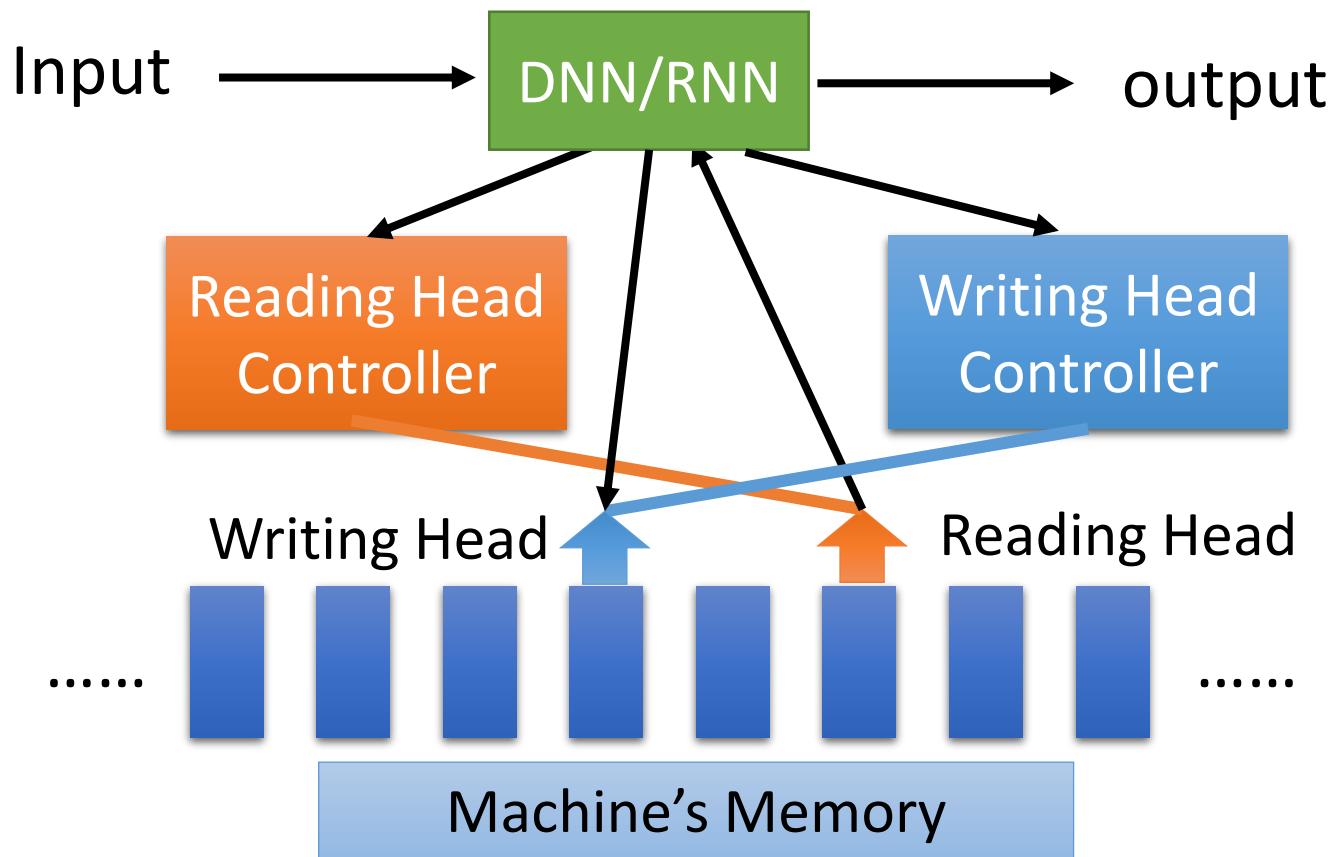
# Attention-based Model



Ref:

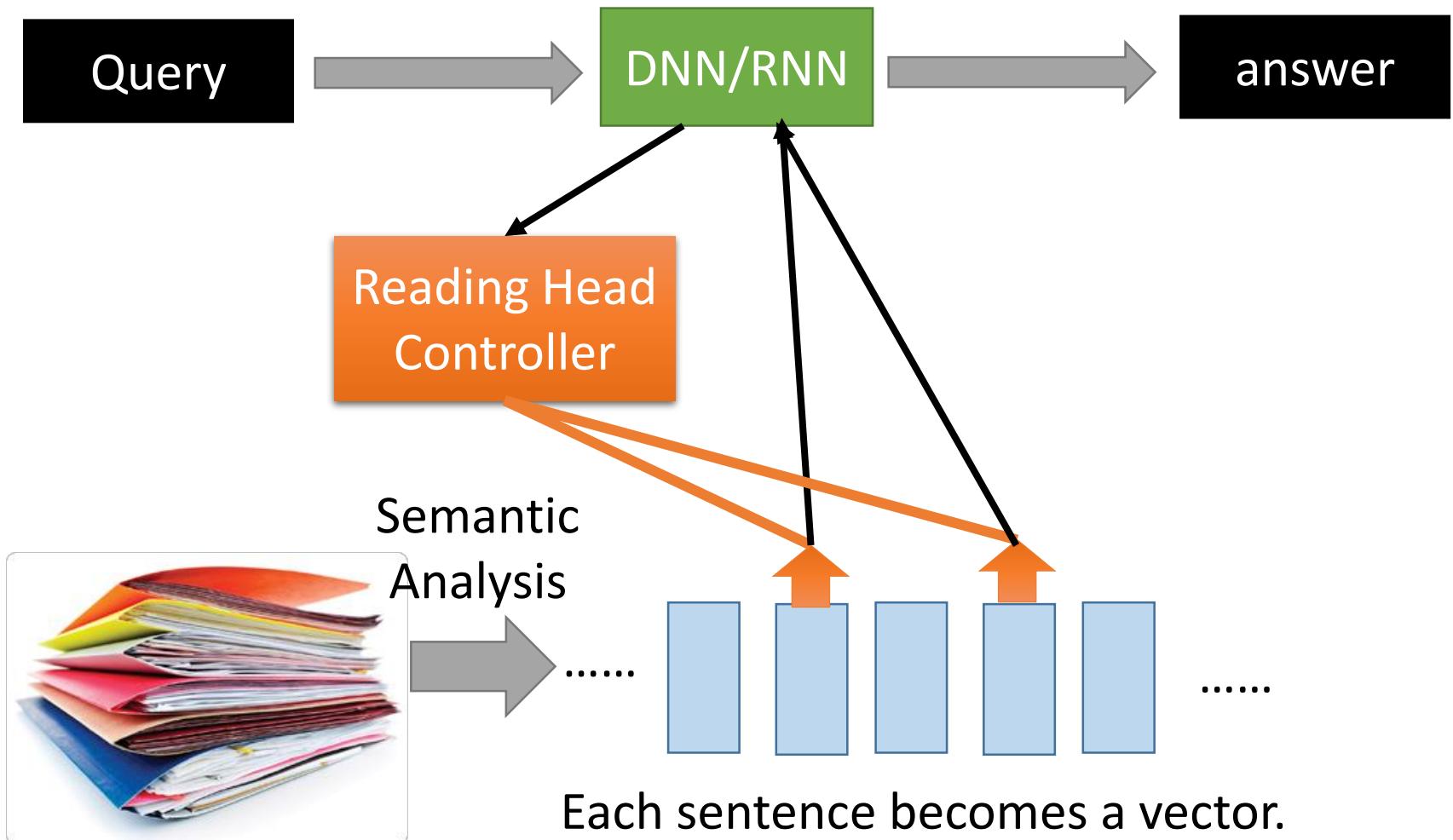
[http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS\\_2015\\_2/Lecture/Attain%20\(v3\).ecm.mp4/index.html](http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/Attain%20(v3).ecm.mp4/index.html)

# Attention-based Model v2



Neural Turing Machine

# Reading Comprehension



# Reading Comprehension

- End-To-End Memory Networks. S. Sukhbaatar, A. Szlam, J. Weston, R. Fergus. NIPS, 2015.

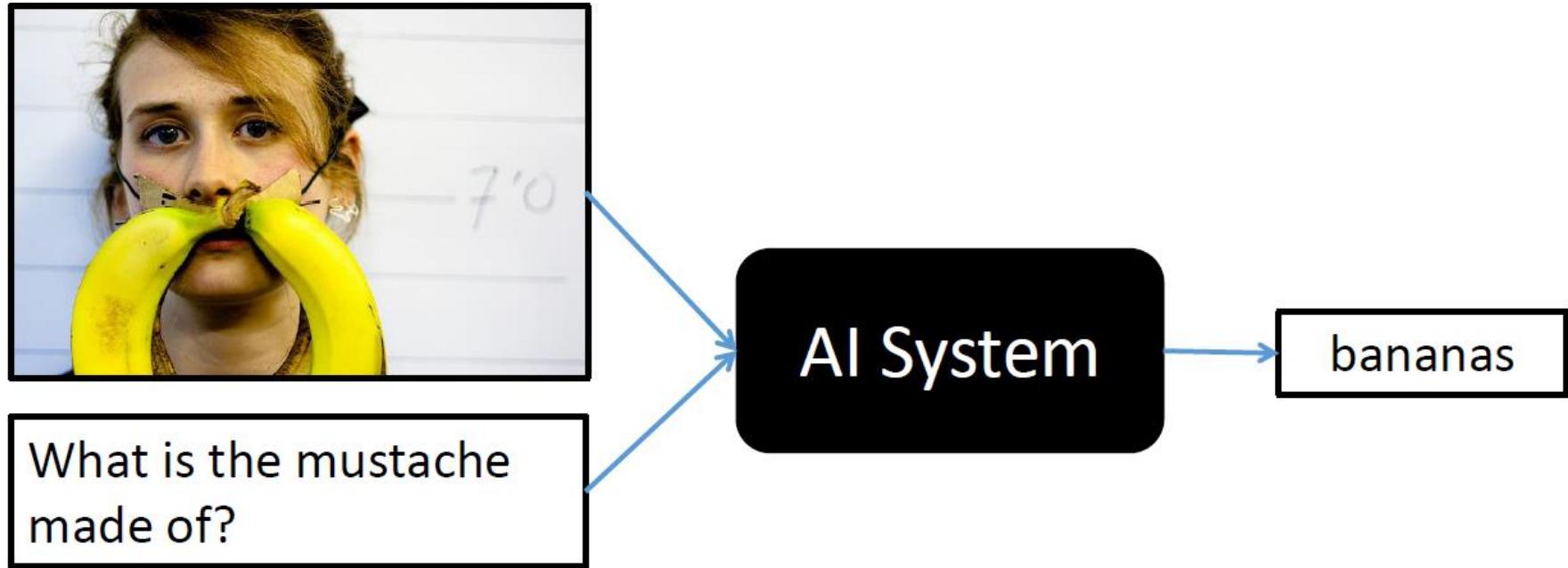
The position of reading head:

Story (16: basic induction)	Support	Hop 1	Hop 2	Hop 3
Brian is a frog.	yes	0.00	0.98	0.00
Lily is gray.		0.07	0.00	0.00
Brian is yellow.	yes	0.07	0.00	1.00
Julius is green.		0.06	0.00	0.00
Greg is a frog.	yes	0.76	0.02	0.00
What color is Greg? Answer: yellow		Prediction: yellow		

Keras has example:

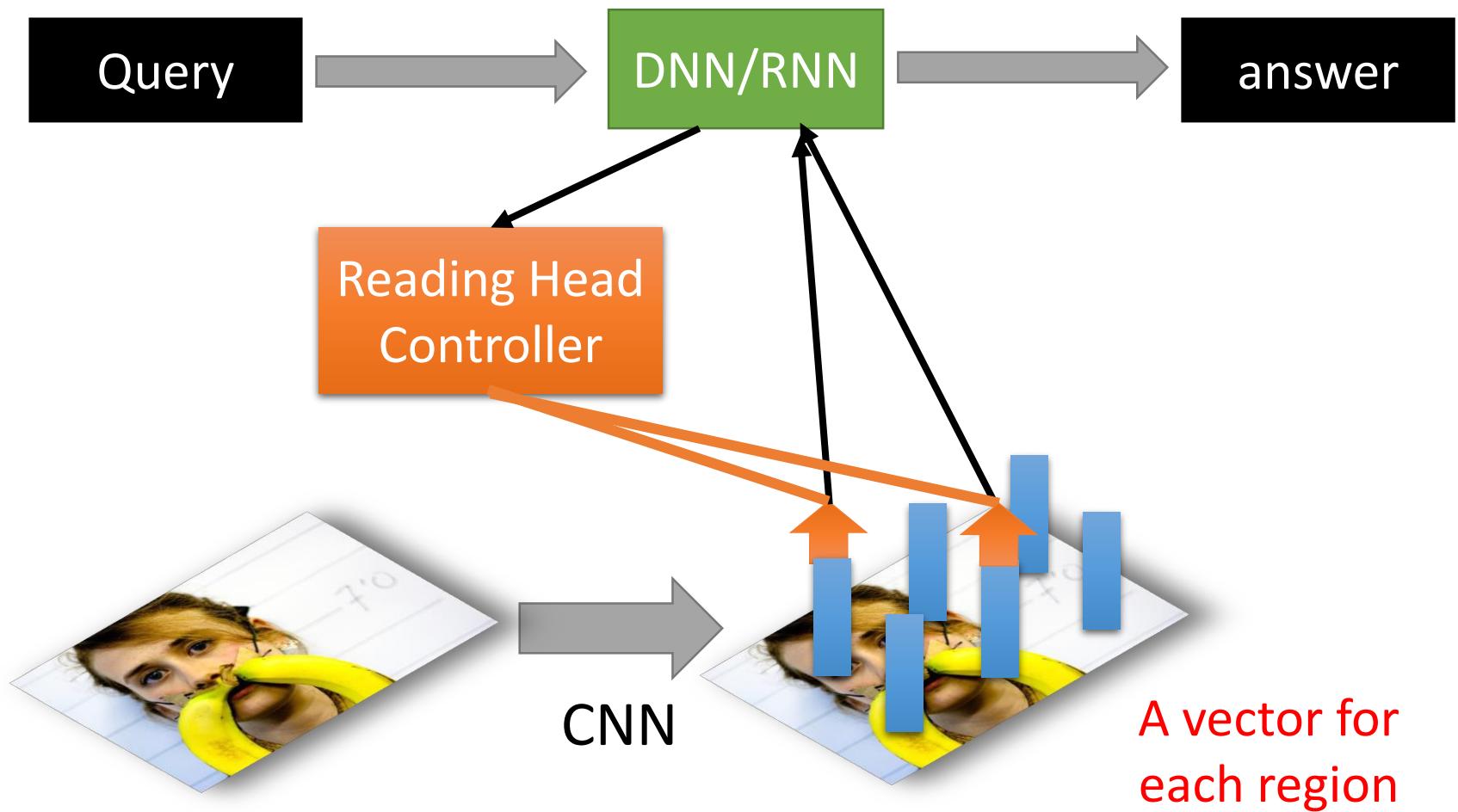
[https://github.com/fchollet/keras/blob/master/examples/babi\\_memnn.py](https://github.com/fchollet/keras/blob/master/examples/babi_memnn.py)

# Visual Question Answering



source: <http://visualqa.org/>

# Visual Question Answering



# Visual Question Answering

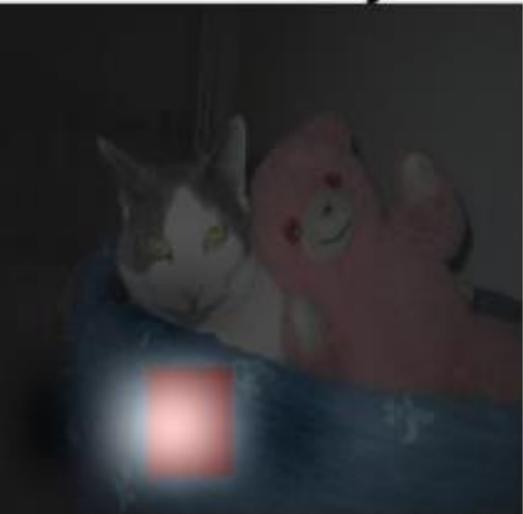
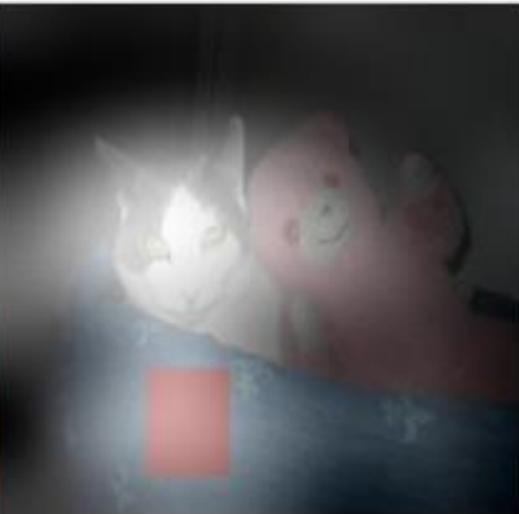
- Huijuan Xu, Kate Saenko. Ask, Attend and Answer: Exploring Question-Guided Spatial Attention for Visual Question Answering. arXiv Pre-Print, 2015

**Is there a red square on the bottom of the cat?**

**GT: yes**



**Prediction: yes**



# Speech Question Answering

- **TOEFL Listening Comprehension Test by Machine**
- Example:

Audio Story:  (The original story is 5 min long.)

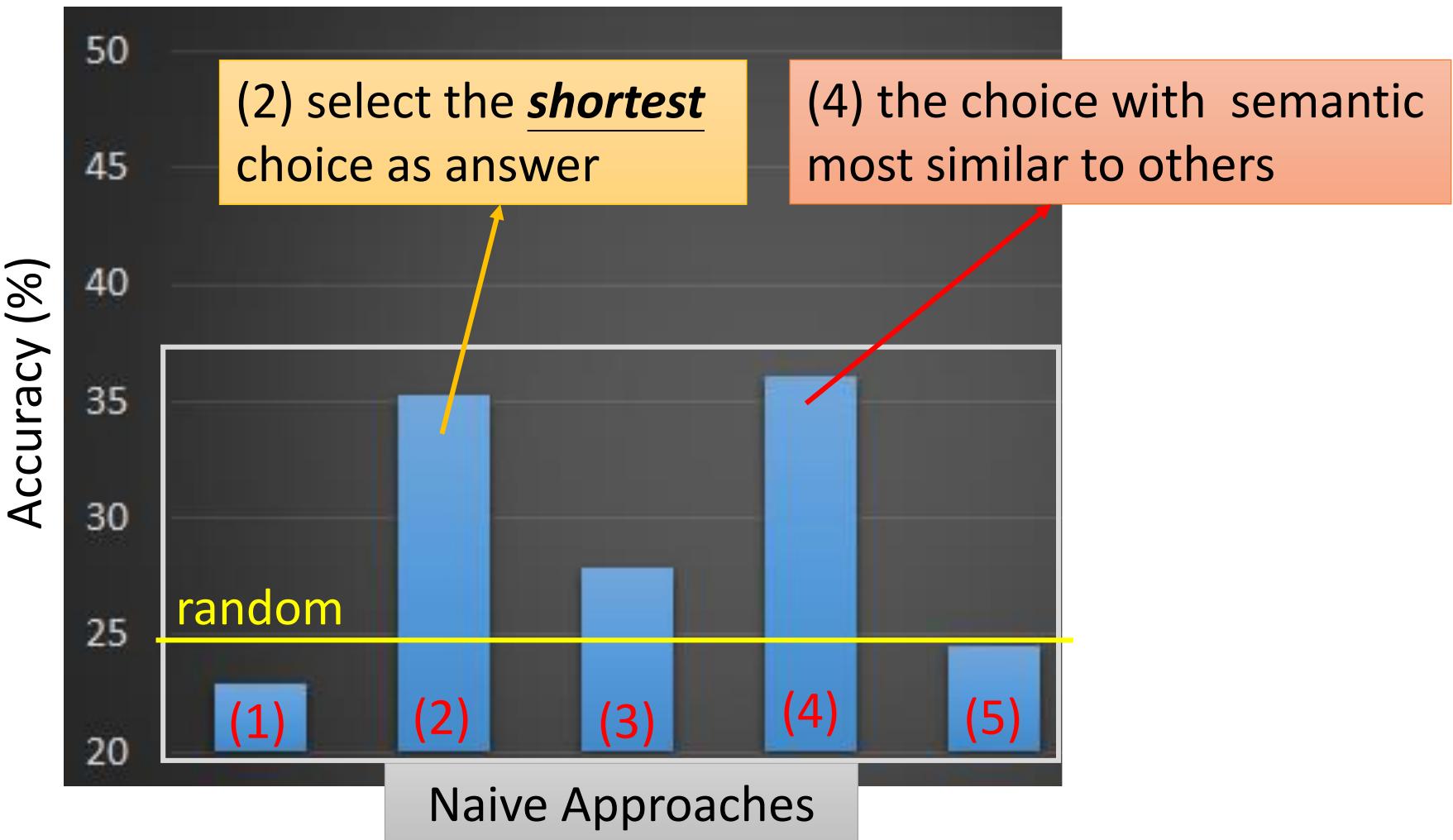
Question: “What is a possible origin of Venus’ clouds?”

Choices:

- (A) gases released as a result of volcanic activity
- (B) chemical reactions caused by high surface temperatures
- (C) bursts of radio energy from the planet's surface
- (D) strong winds that blow dust into the atmosphere

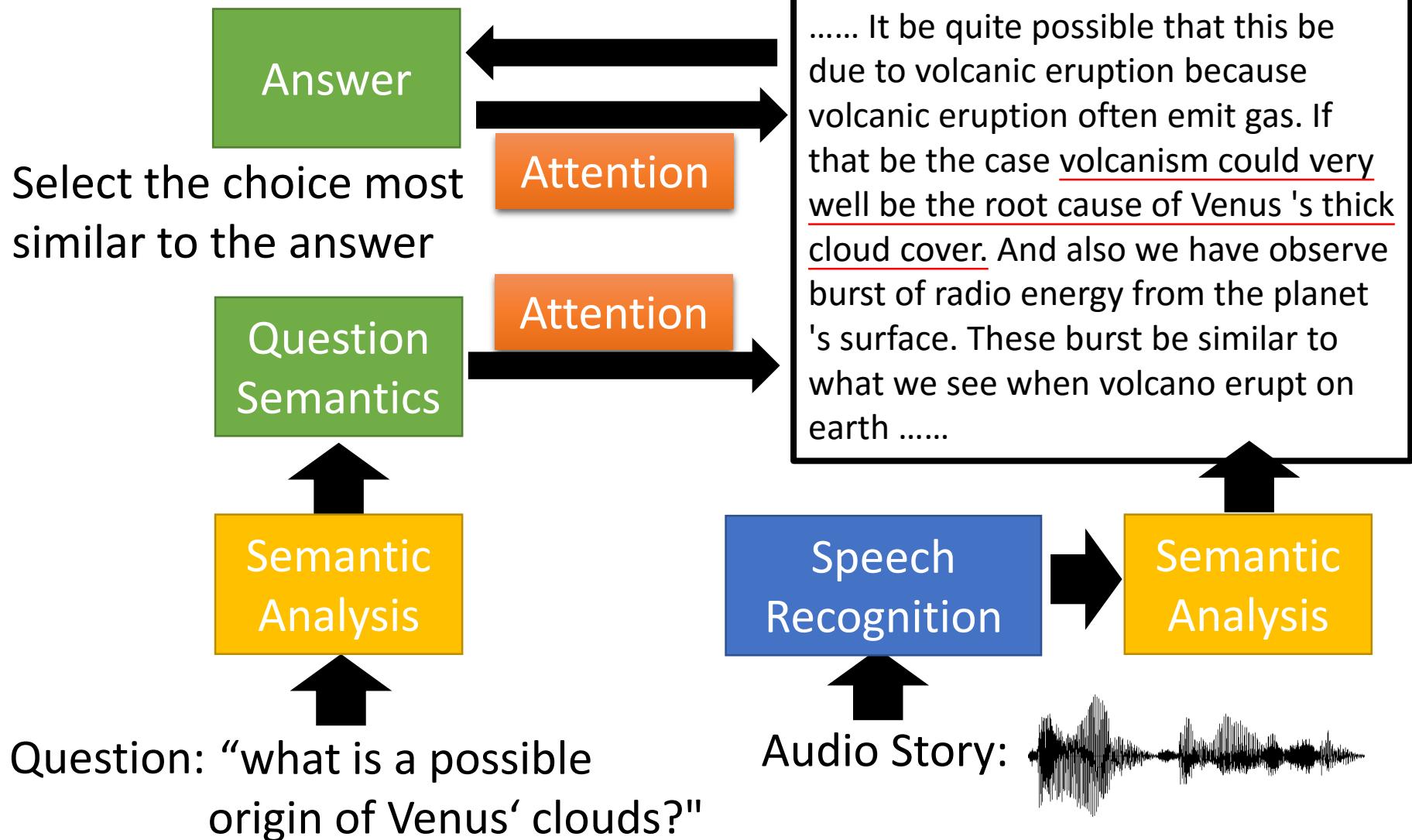
# Simple Baselines

Experimental setup:  
717 for training,  
124 for validation, 122 for testing



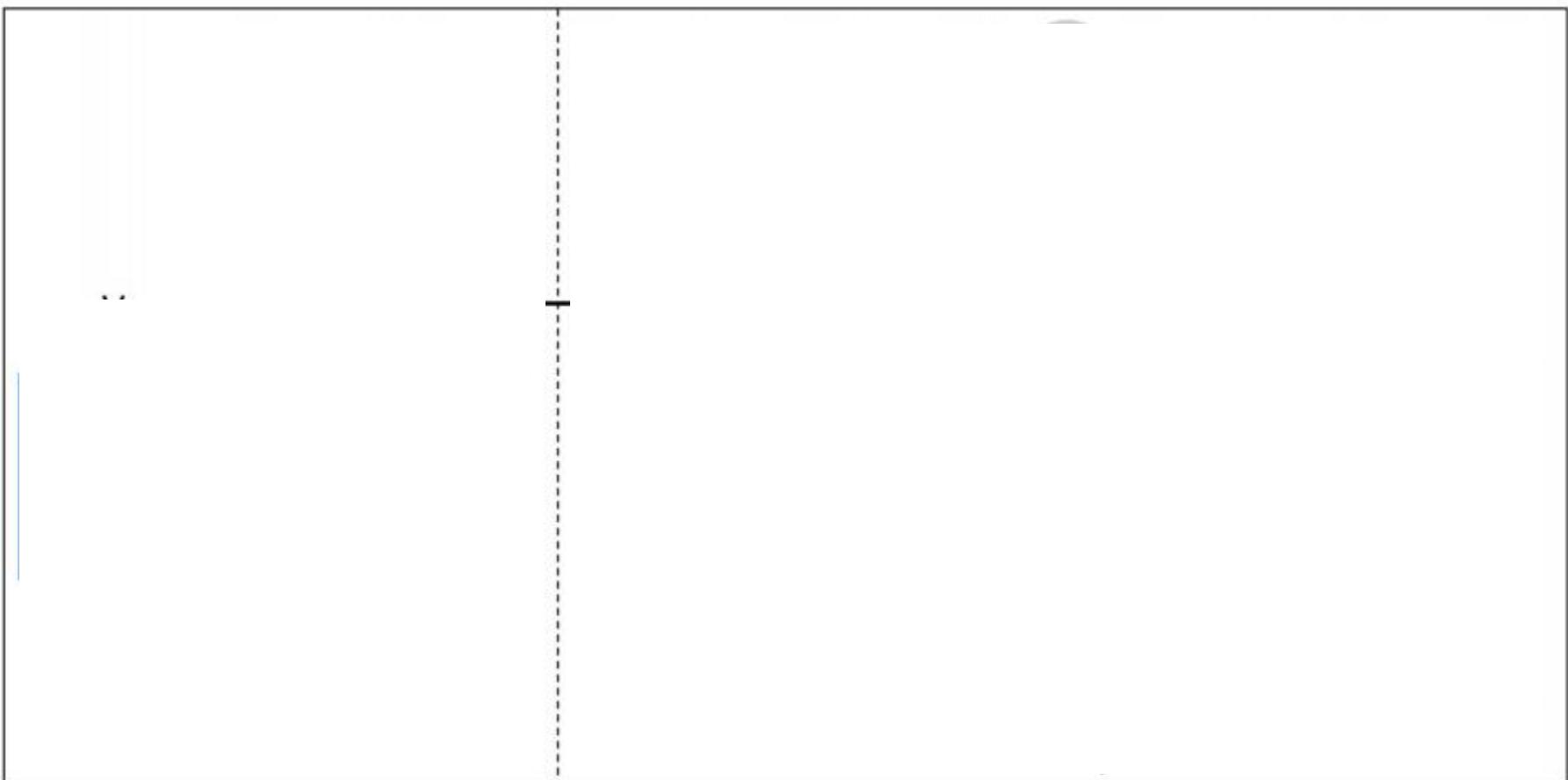
# Model Architecture

Everything is learned  
from training examples



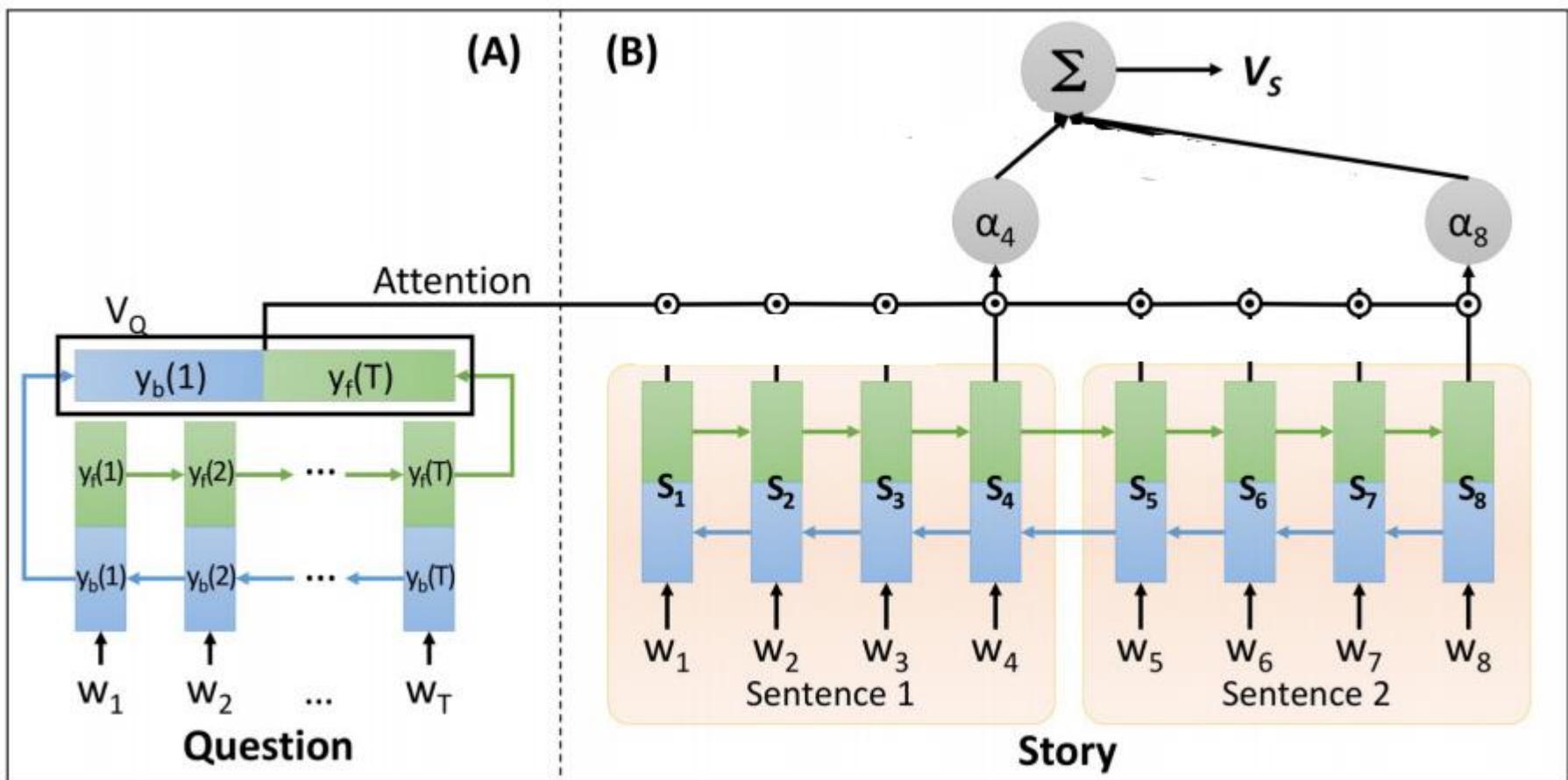
# Model Architecture

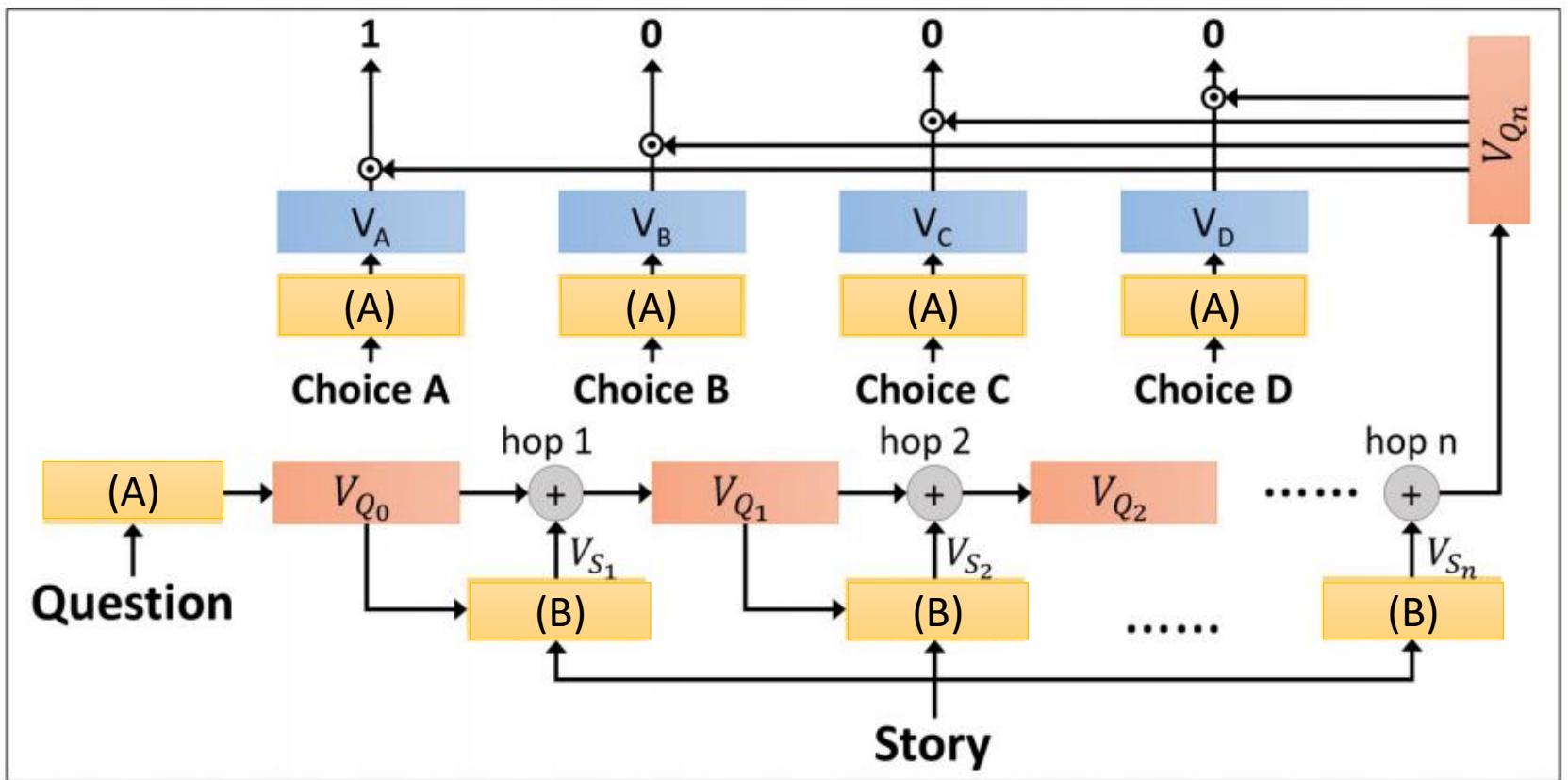
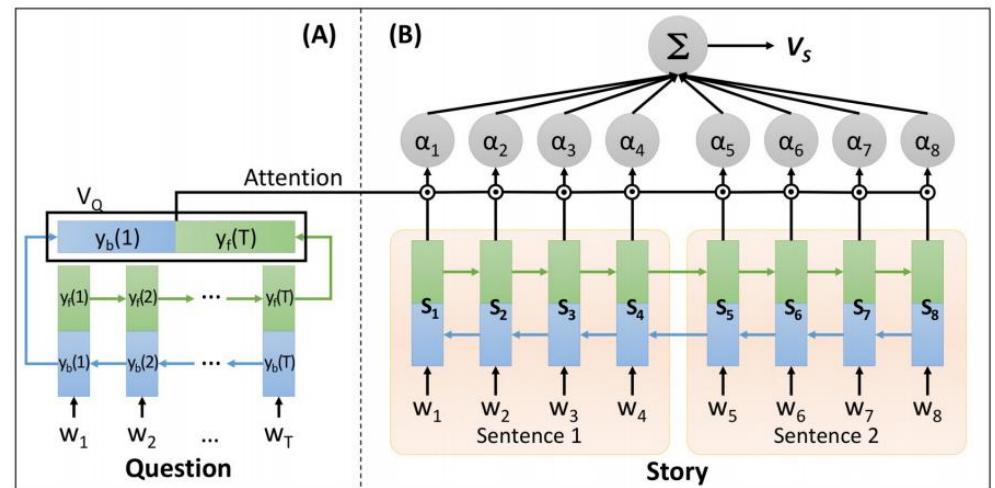
## Word-based Attention



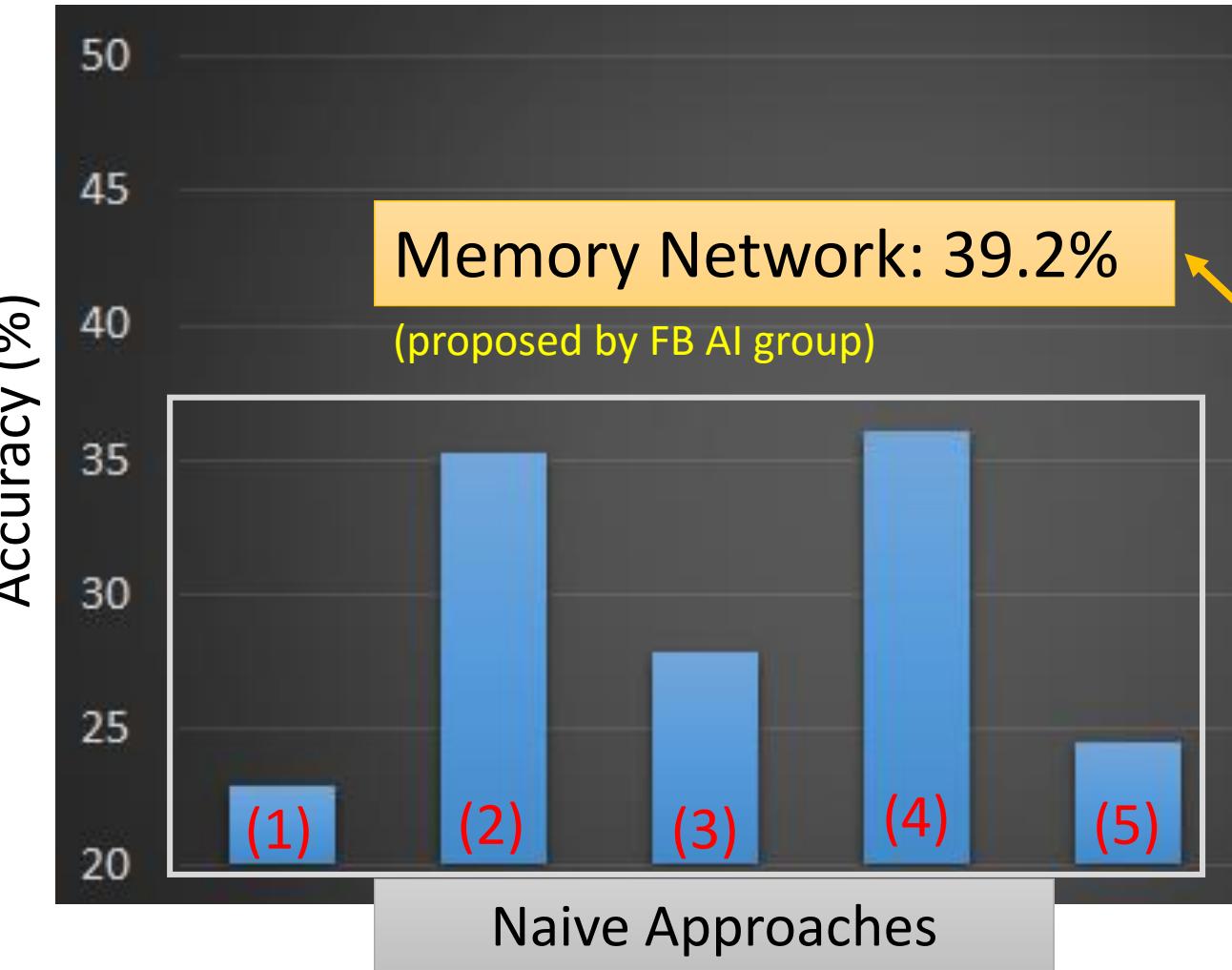
# Model Architecture

## Sentence-based Attention



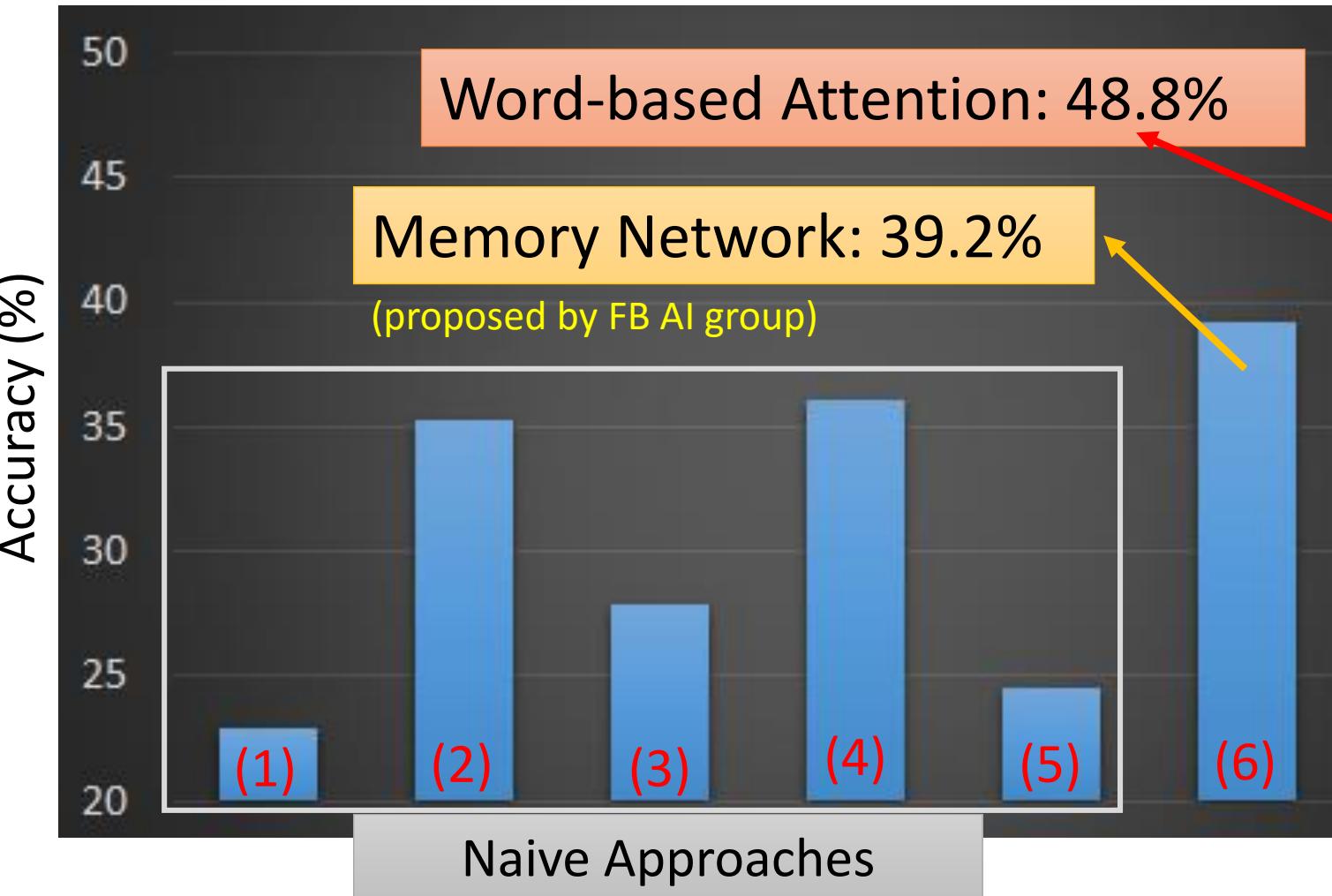


# Supervised Learning



# Supervised Learning

[Tseng & Lee, Interspeech 16]  
[Fang & Hsu & Lee, SLT 16]



# Outline

## Supervised Learning

- Ultra Deep Network
  - Attention Model
- }
- New network structure

## Reinforcement Learning

## Unsupervised Learning

- Image: Realizing what the World Looks Like
- Text: Understanding the Meaning of Words
- Audio: Learning human language without supervision

# Scenario of Reinforcement Learning



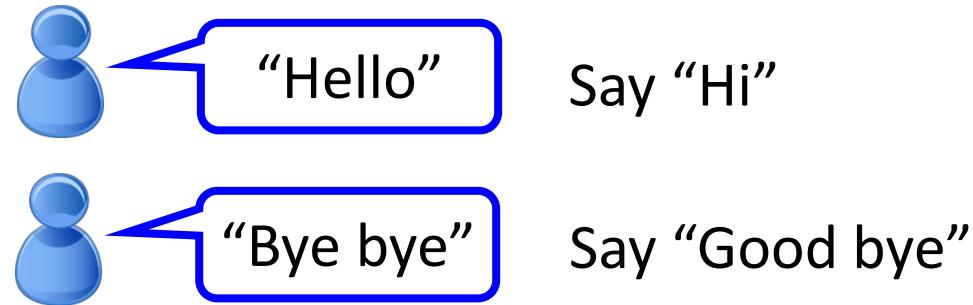
# Scenario of Reinforcement Learning



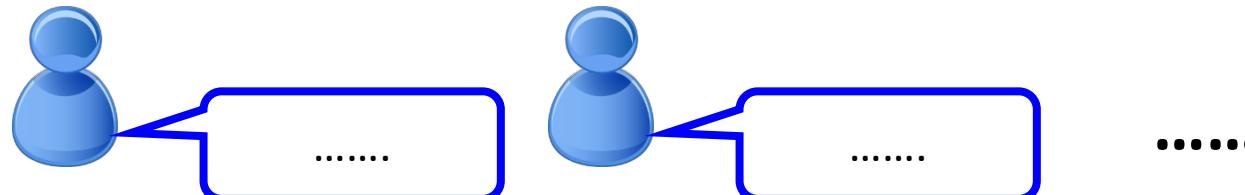
# Supervised v.s. Reinforcement

- Supervised

Learning from  
teacher



- Reinforcement



Bad

Learning from  
critics

Hello ☺

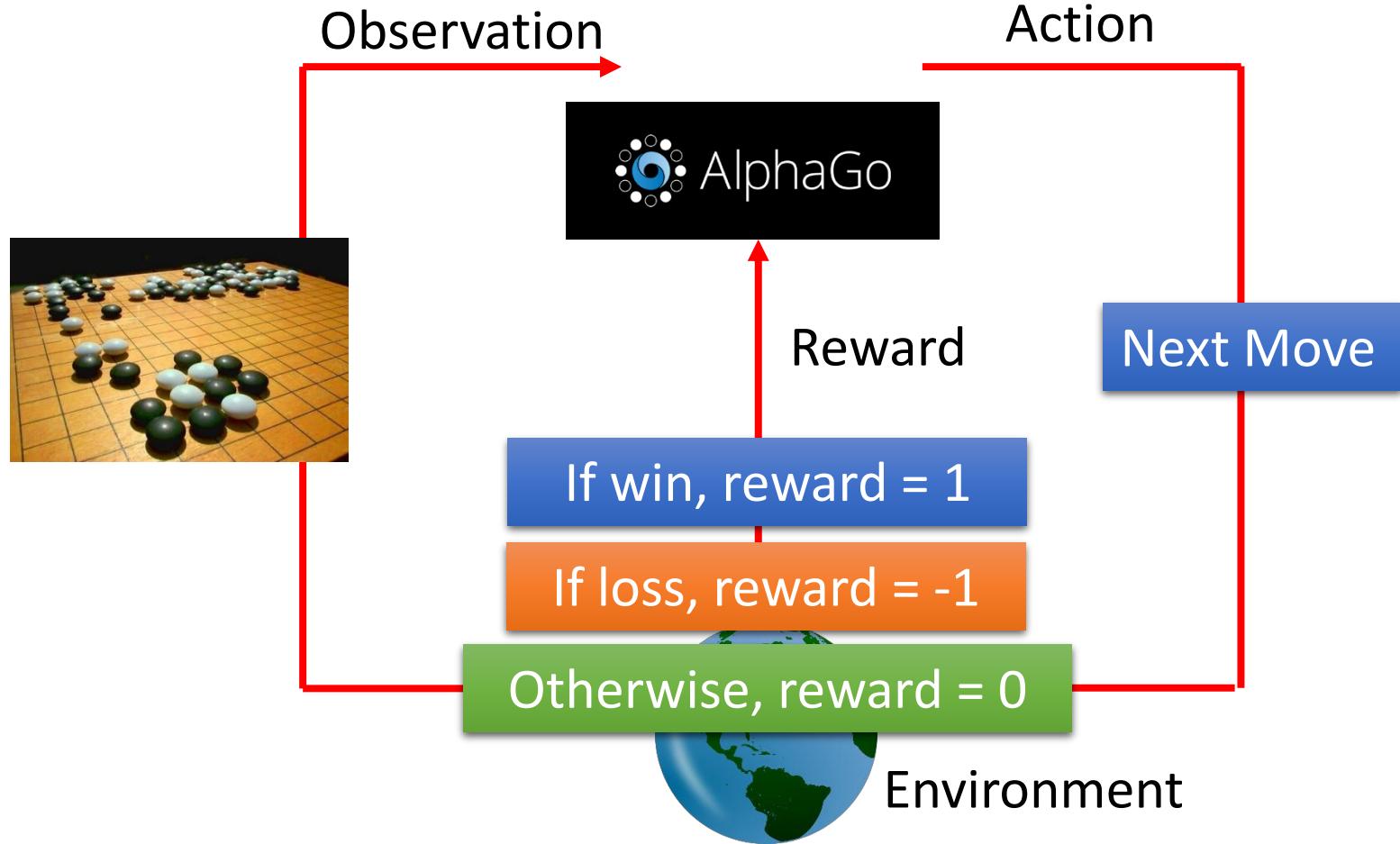
Agent

.....

Agent

# Scenario of Reinforcement Learning

Agent learns to take actions to maximize expected reward.



# Supervised v.s. Reinforcement

- Supervised:



Next move:  
“5-5”



Next move:  
“3-3”

- Reinforcement Learning

First move → ..... many moves ..... → Win!

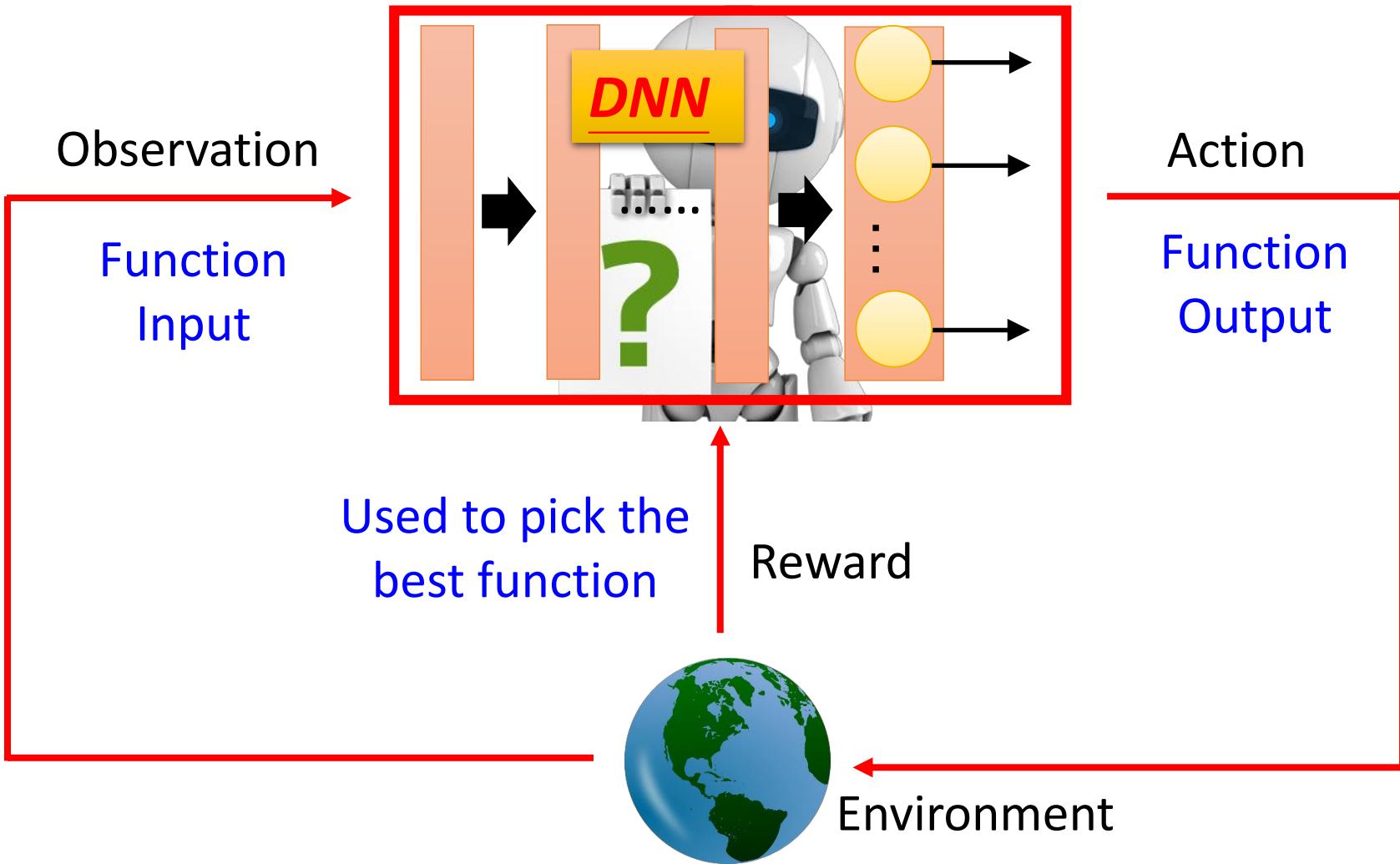
Alpha Go is supervised learning + reinforcement learning.

# Difficulties of Reinforcement Learning

- It may be better to sacrifice immediate reward to gain more long-term reward
  - E.g. Playing Go
- Agent's actions affect the subsequent data it receives
  - E.g. Exploration



# Deep Reinforcement Learning



# Application: Interactive Retrieval

- Interactive retrieval is helpful.

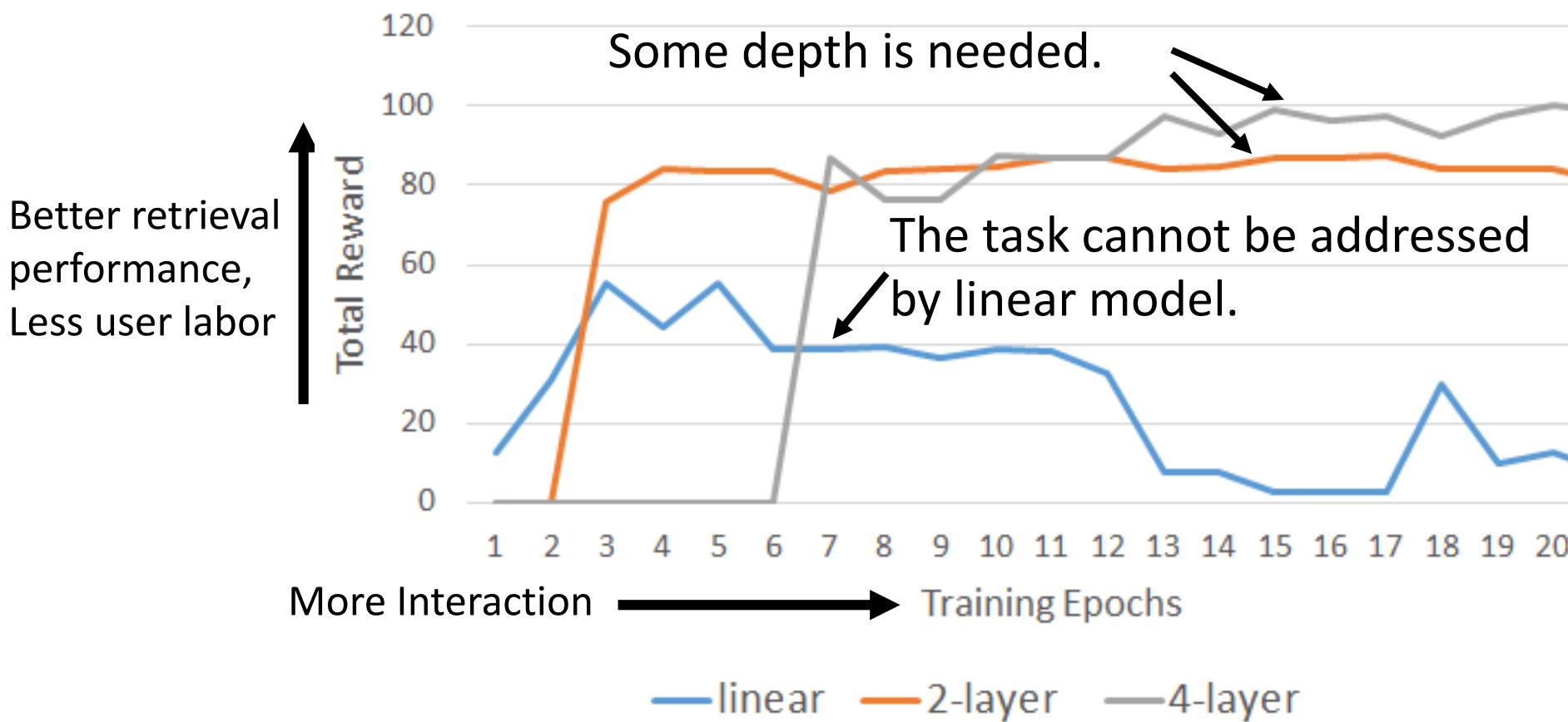
[Wu & Lee, INTERSPEECH 16]



“Deep Learning” related to Machine Learning?  
“Deep Learning” related to Education?

# Deep Reinforcement Learning

- Different network depth



# More applications

- Alpha Go, Playing Video Games, Dialogue
- Flying Helicopter
  - <https://www.youtube.com/watch?v=0JL04JJjocc>
- Driving
  - <https://www.youtube.com/watch?v=0xo1Ldx3L5Q>
- Google Cuts Its Giant Electricity Bill With DeepMind-Powered AI
  - <http://www.bloomberg.com/news/articles/2016-07-19/google-cuts-its-giant-electricity-bill-with-deepmind-powered-ai>

# To learn deep reinforcement learning .....

- Lectures of David Silver
  - <http://www0.cs.ucl.ac.uk/staff/D.Silver/web/Taching.html>
  - 10 lectures (1:30 each)
- Deep Reinforcement Learning
  - [http://videolectures.net/rldm2015\\_silver\\_reinforcement\\_learning/](http://videolectures.net/rldm2015_silver_reinforcement_learning/)

# Outline

## Supervised Learning

- Ultra Deep Network
  - Attention Model
- }
- New network structure

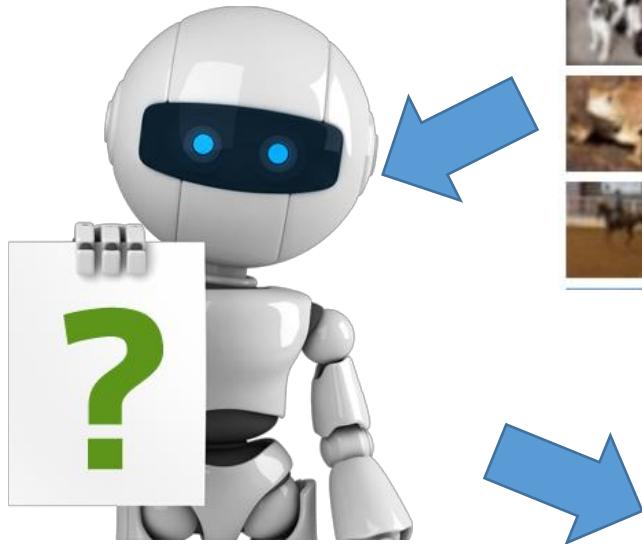
## Reinforcement Learning

## Unsupervised Learning

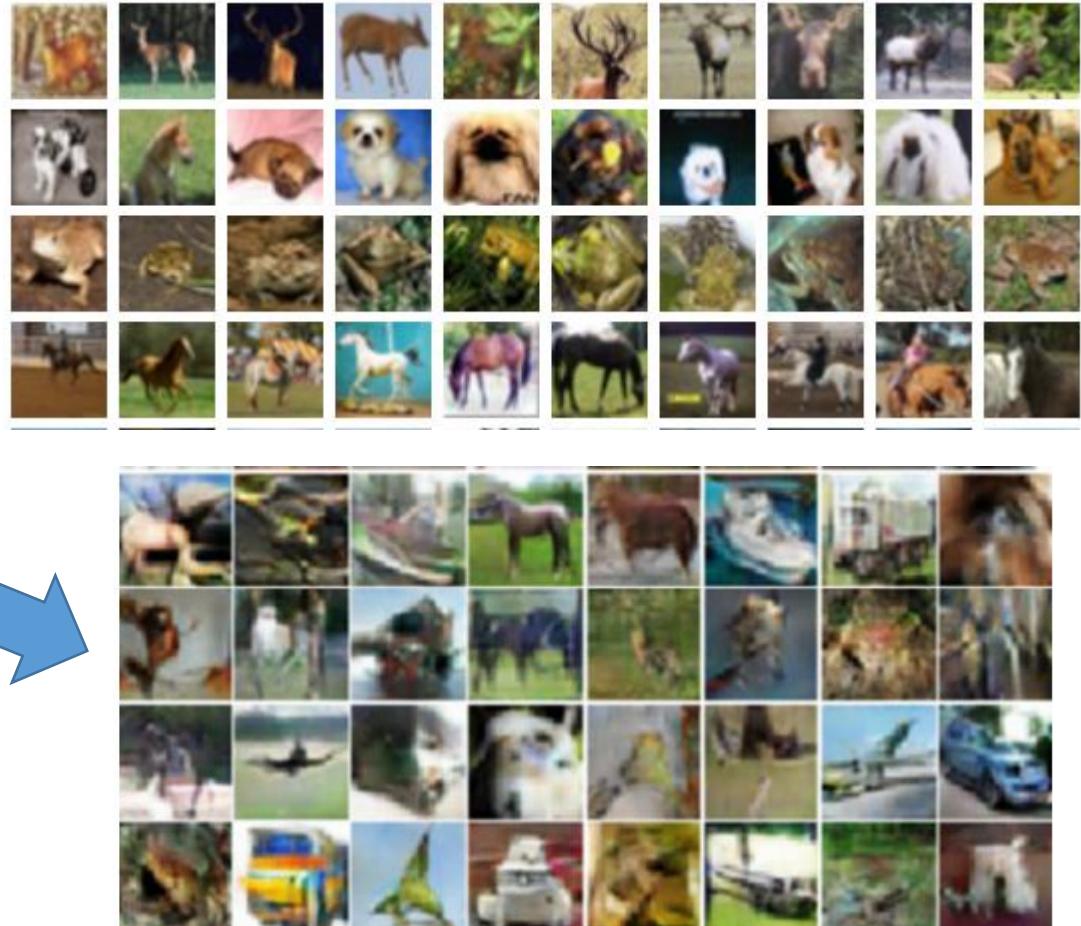
- Image: Realizing what the World Looks Like
- Text: Understanding the Meaning of Words
- Audio: Learning human language without supervision

# Does machine know what the world look like?

Ref: <https://openai.com/blog/generative-models/>



Draw something!



# Deep Dream

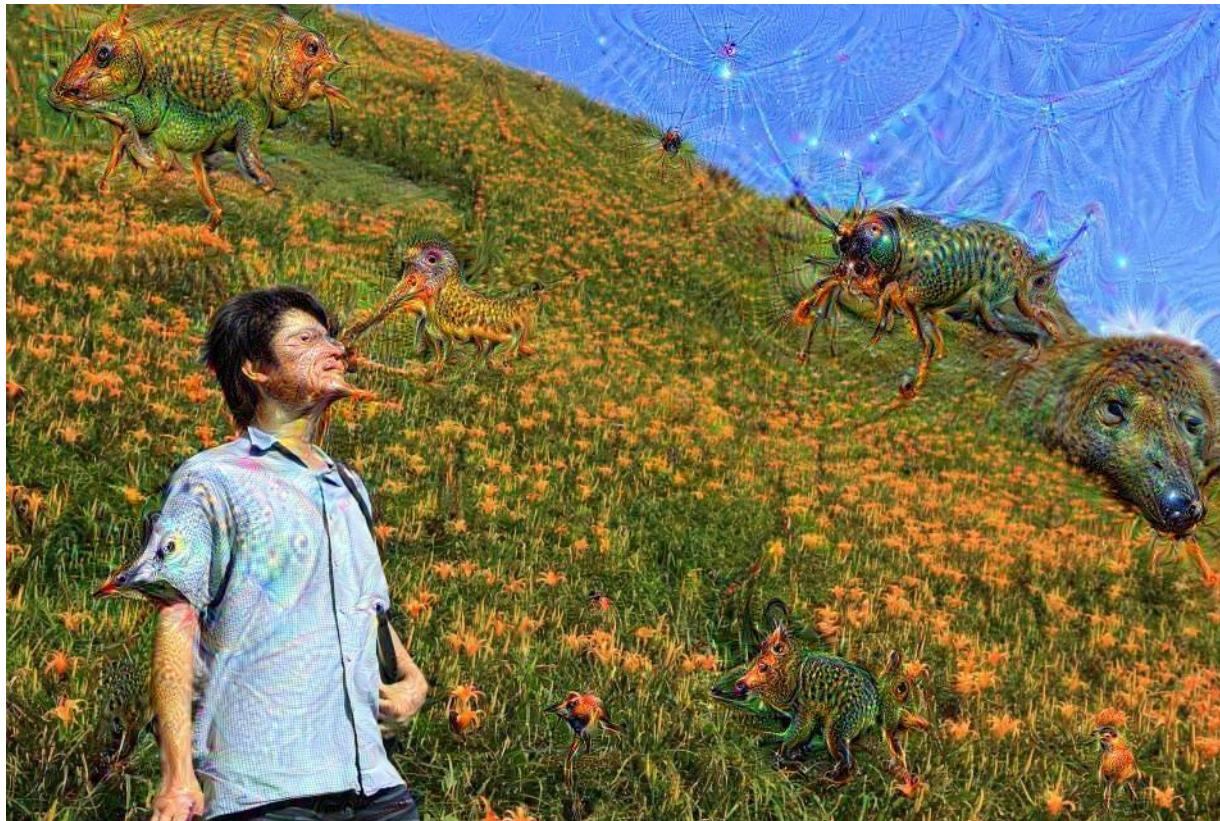
- Given a photo, machine adds what it sees .....



<http://deepdreamgenerator.com/>

# Deep Dream

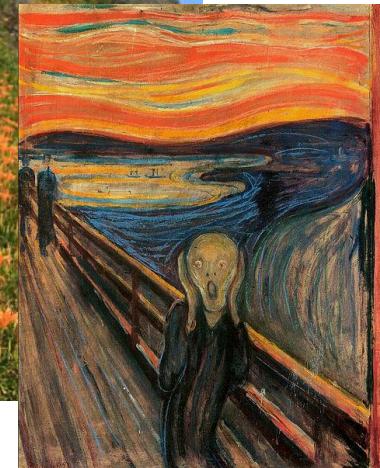
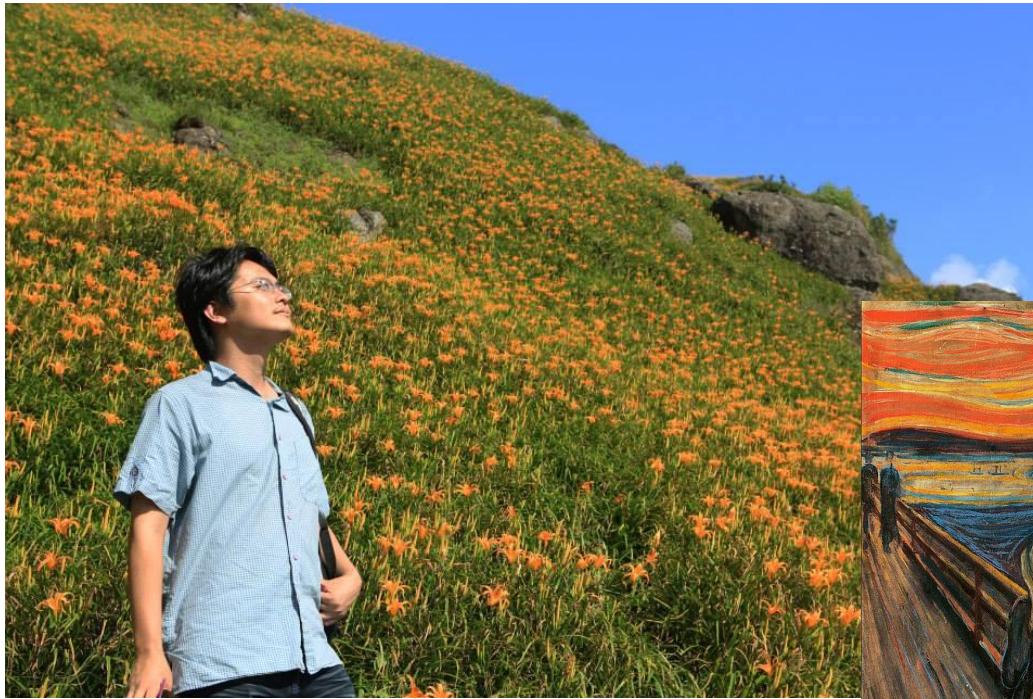
- Given a photo, machine adds what it sees .....



<http://deepdreamgenerator.com/>

# Deep Style

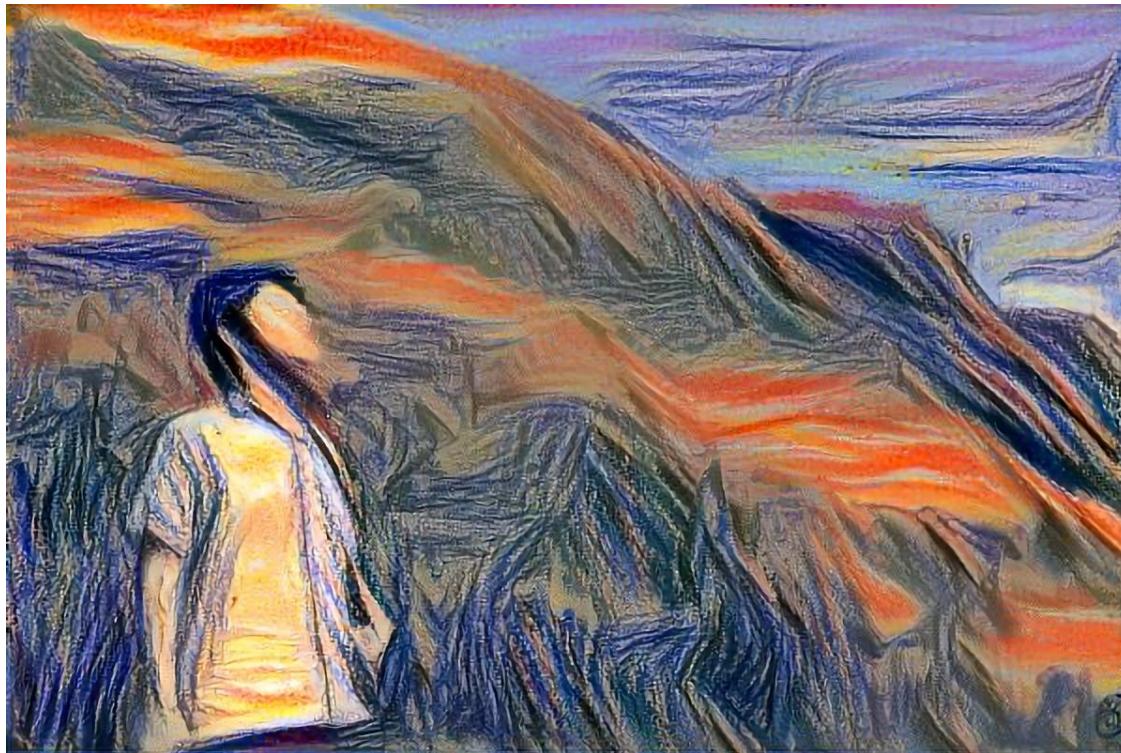
- Given a photo, make its style like famous paintings



<https://dreamscopeapp.com/>

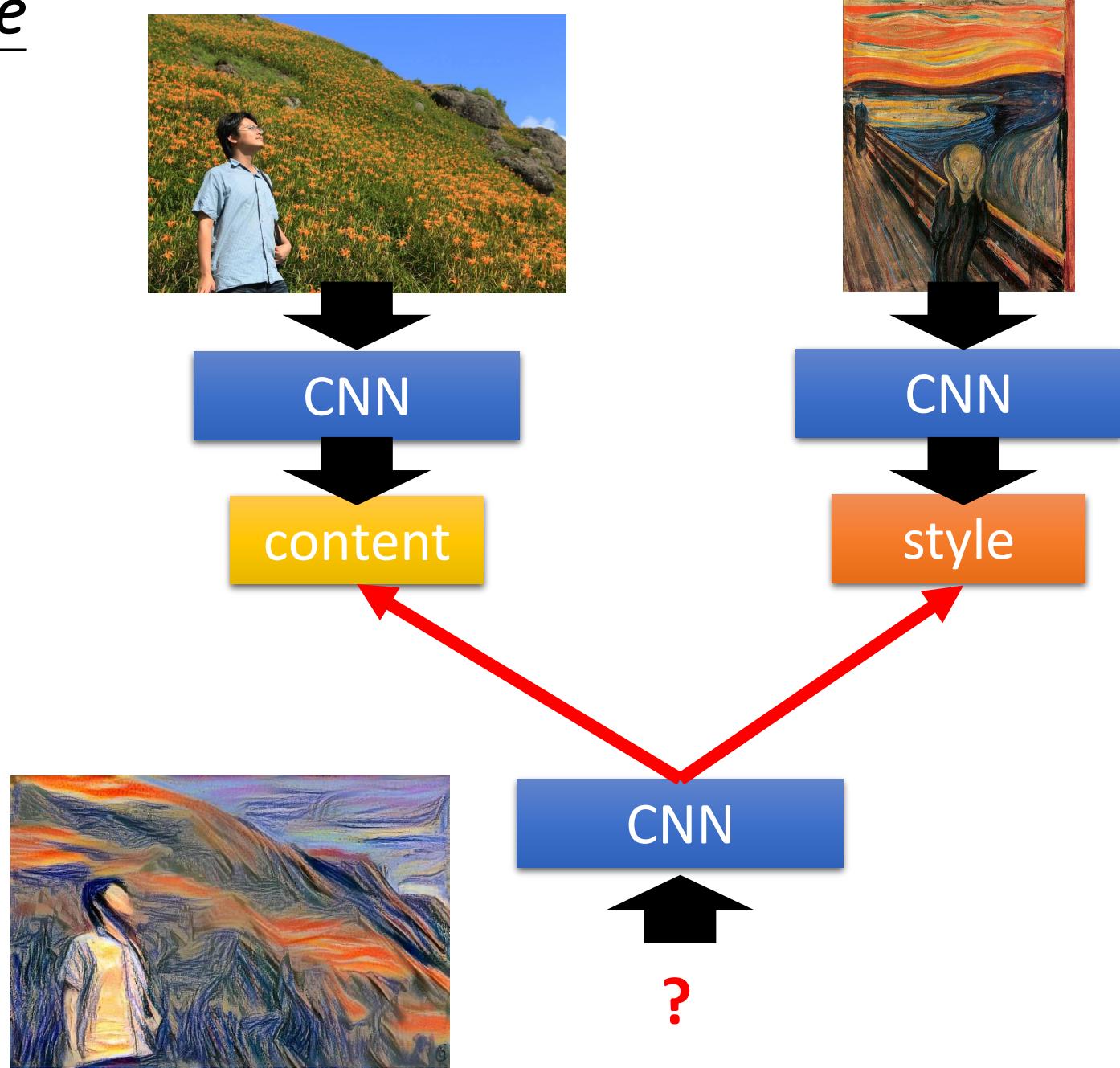
# Deep Style

- Given a photo, make its style like famous paintings



<https://dreamscopeapp.com/>

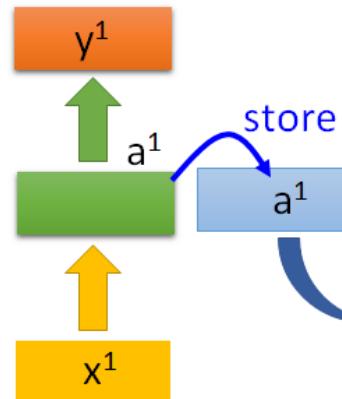
# Deep Style



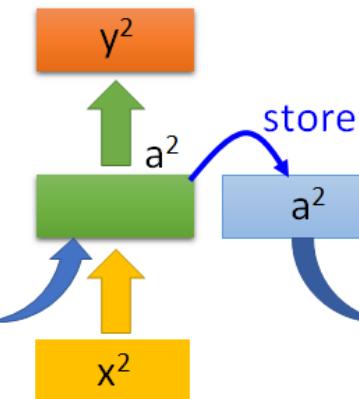
# Generating Images by RNN



color of  
2<sup>nd</sup> pixel



color of  
3rd pixel



color of  
4<sup>th</sup> pixel



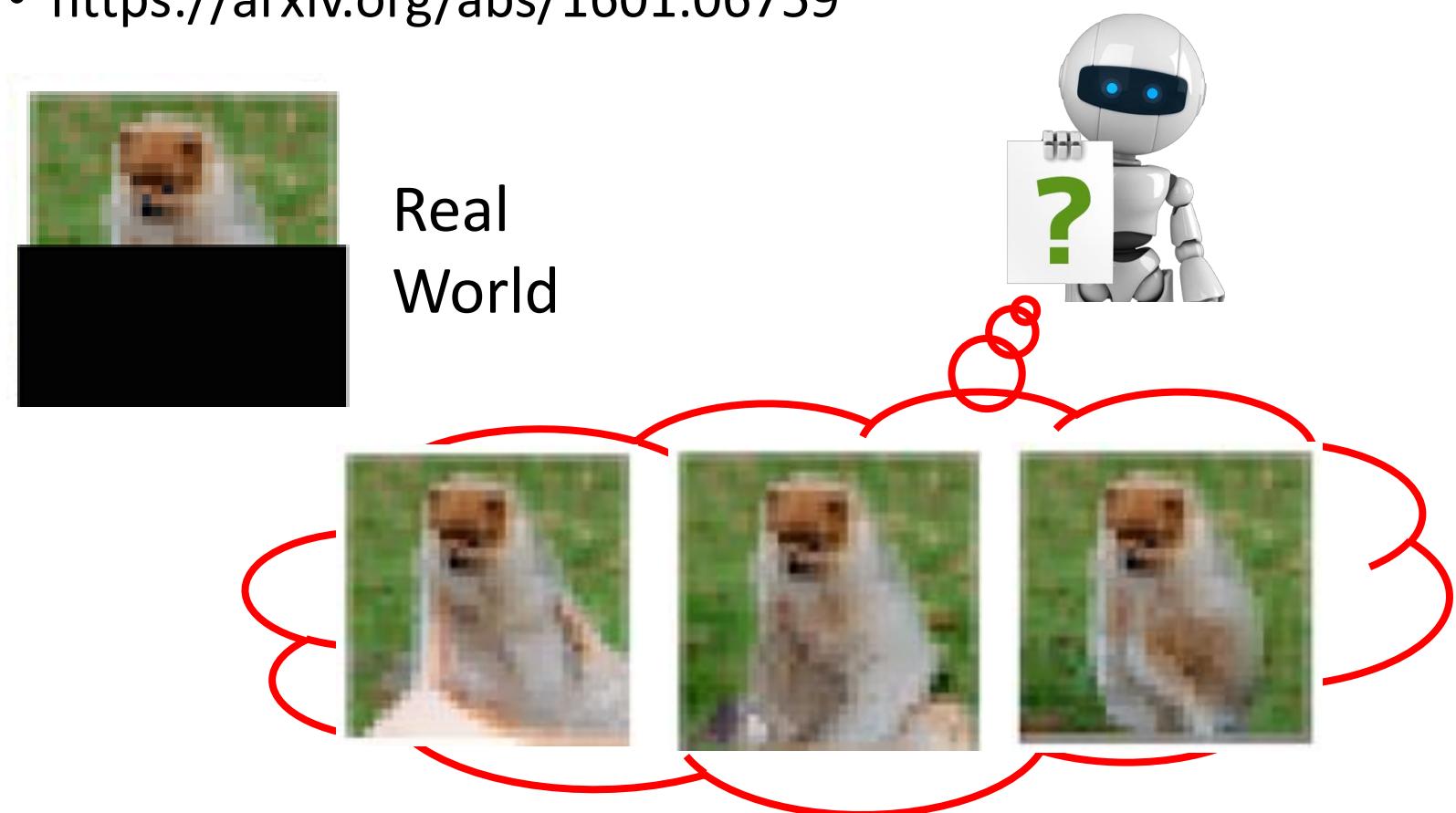
color of  
1<sup>st</sup> pixel

color of  
2<sup>nd</sup> pixel

color of  
3rd pixel

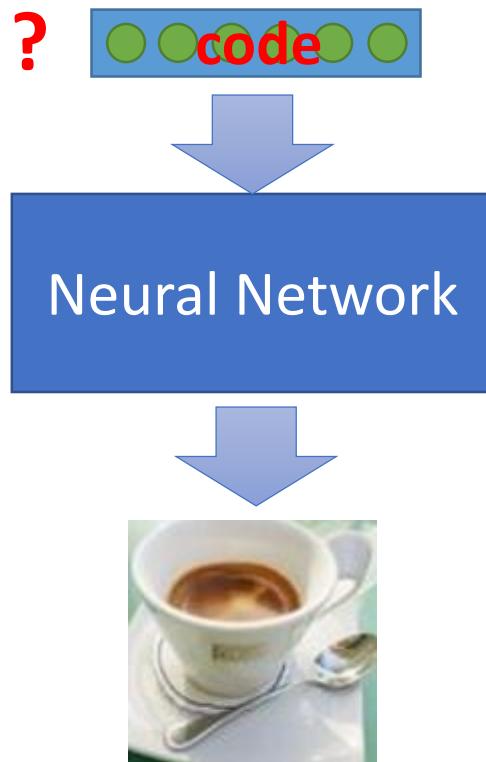
# Generating Images by RNN

- **Pixel Recurrent Neural Networks**
  - <https://arxiv.org/abs/1601.06759>

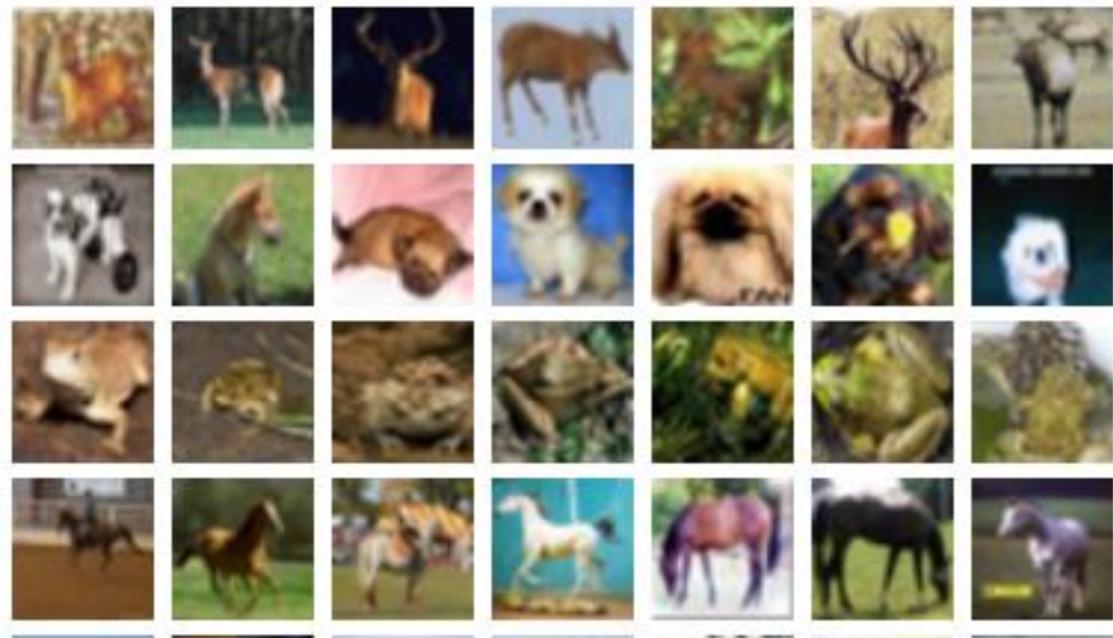


# Generating Images

- Training a decoder to generate images is **unsupervised**

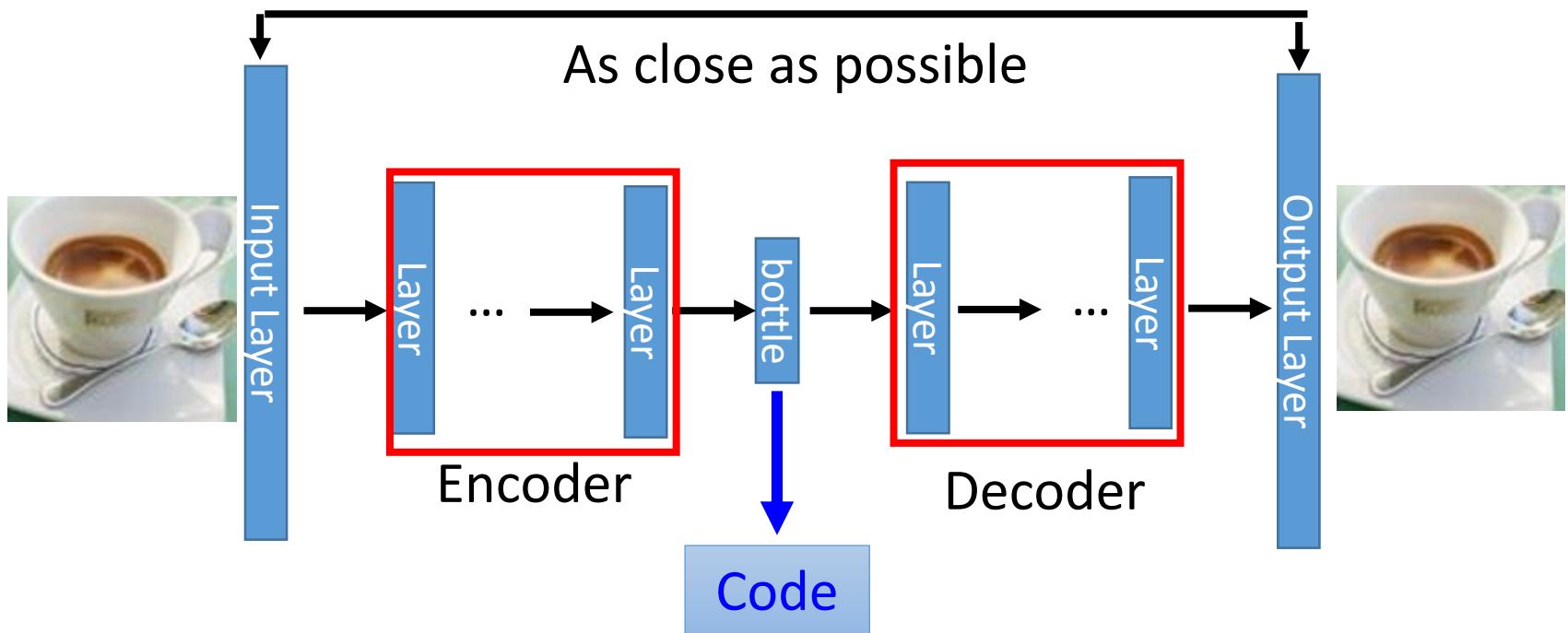
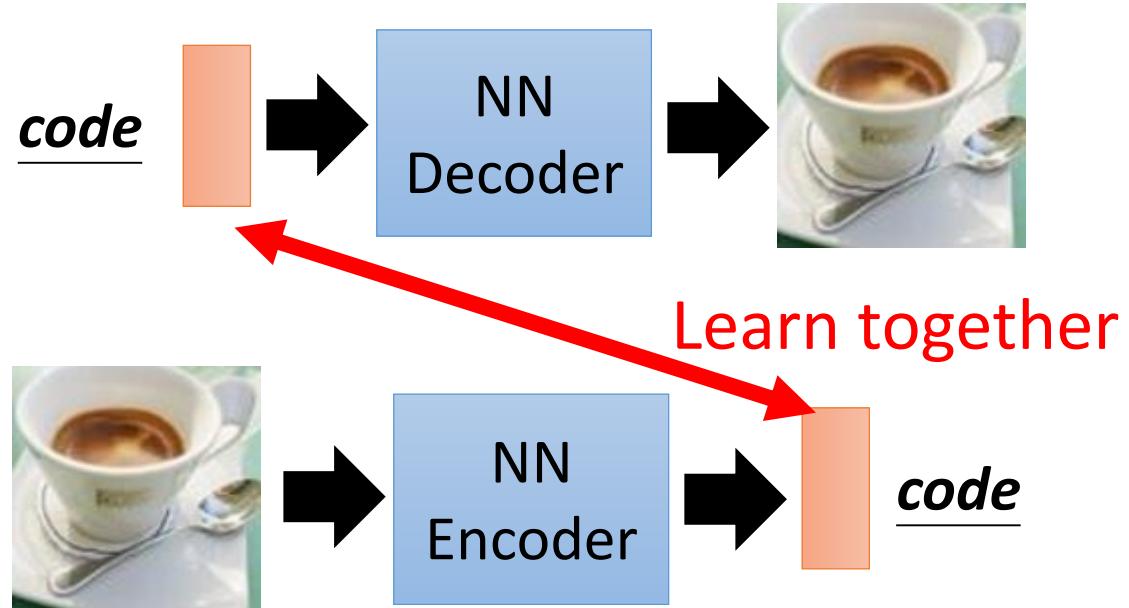


Training data is a lot of images



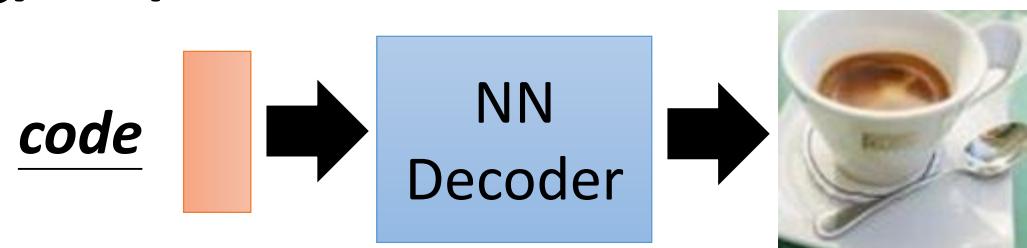
# Auto-encoder

Not state-of-the-art approach

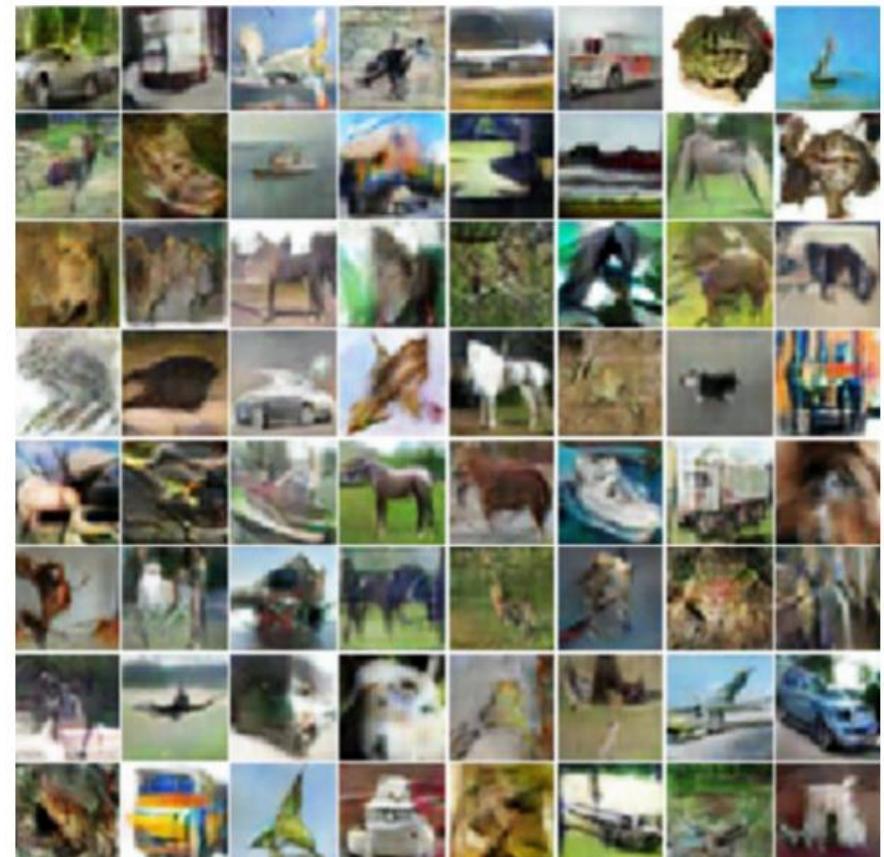
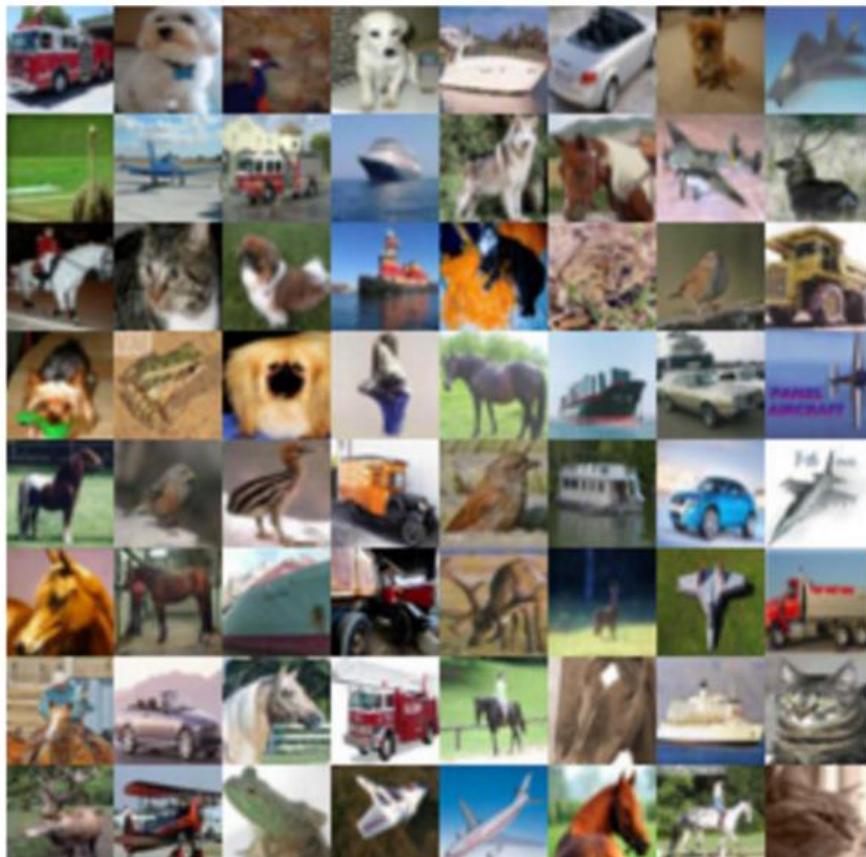


# Generating Images

- Training a decoder to generate images is **unsupervised**
- Variation Auto-encoder (VAE)
  - Ref: **Auto-Encoding Variational Bayes**,  
<https://arxiv.org/abs/1312.6114>
- Generative Adversarial Network (GAN)
  - Ref: **Generative Adversarial Networks**,  
<http://arxiv.org/abs/1406.2661>



Which one is machine-generated?



Ref: <https://openai.com/blog/generative-models/>

# 畫漫畫!!!

<https://github.com/mattyachainer-DCGAN>



# Outline

## Supervised Learning

- Ultra Deep Network
  - Attention Model
- }
- New network structure

## Reinforcement Learning

## Unsupervised Learning

- Image: Realizing what the World Looks Like
- Text: Understanding the Meaning of Words
- Audio: Learning human language without supervision

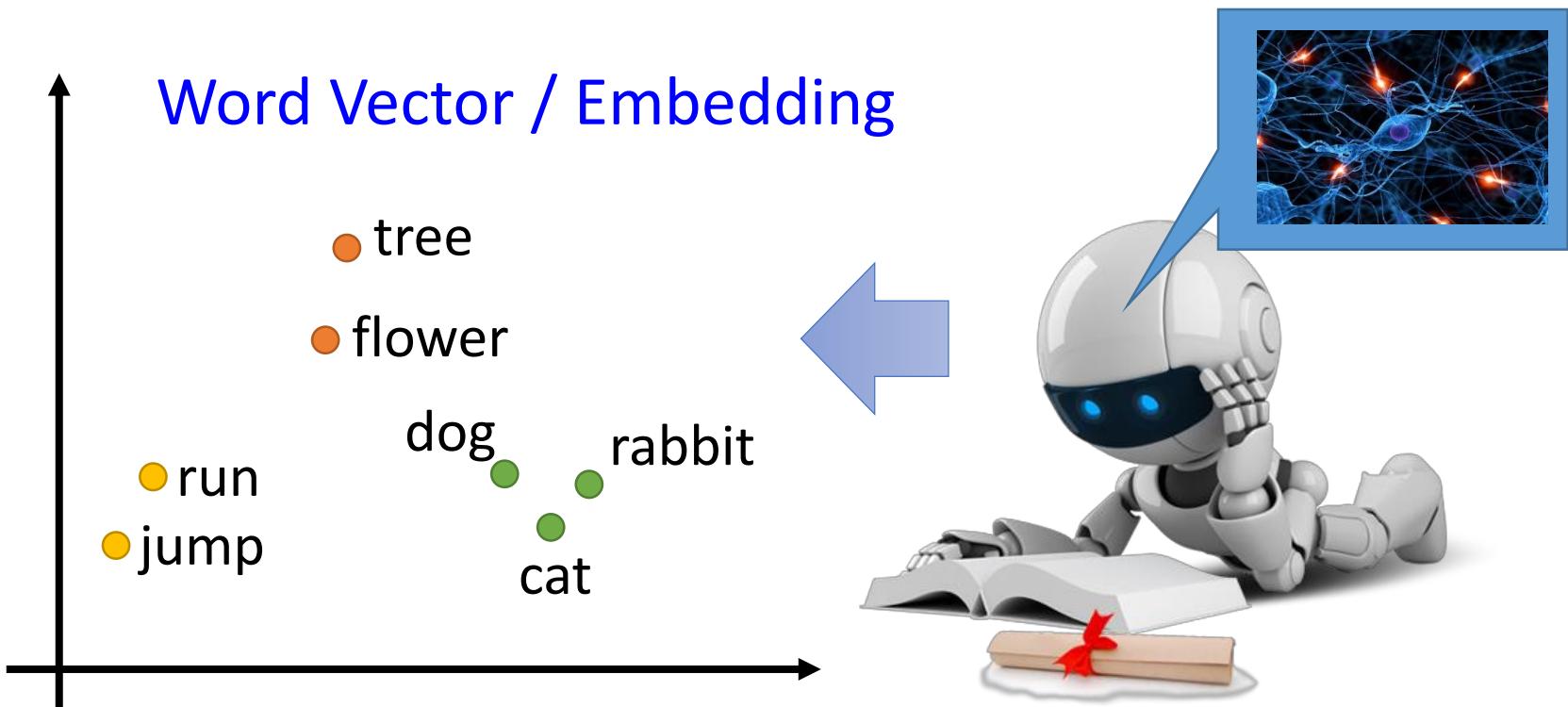
# Machine Reading

- Machine learn the meaning of words from reading a lot of documents without supervision



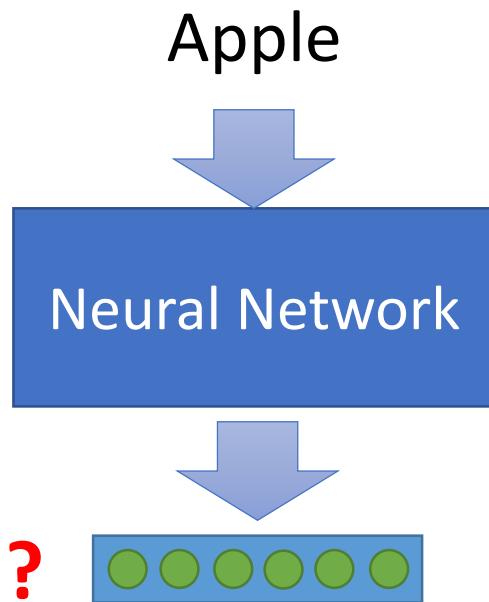
# Machine Reading

- Machine learn the meaning of words from reading a lot of documents without supervision



# Machine Reading

- Generating Word Vector/Embedding is **unsupervised**



Training data is a lot of text



# Machine Reading

- Machine learn the meaning of words from reading a lot of documents without supervision
- A word can be understood by its context

蔡英文、馬英九 are something very similar

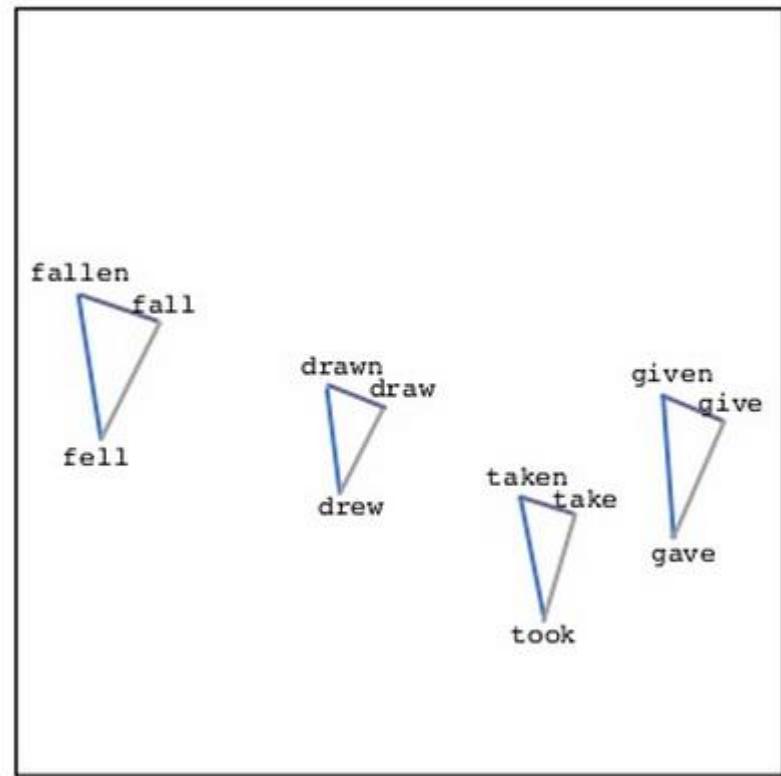
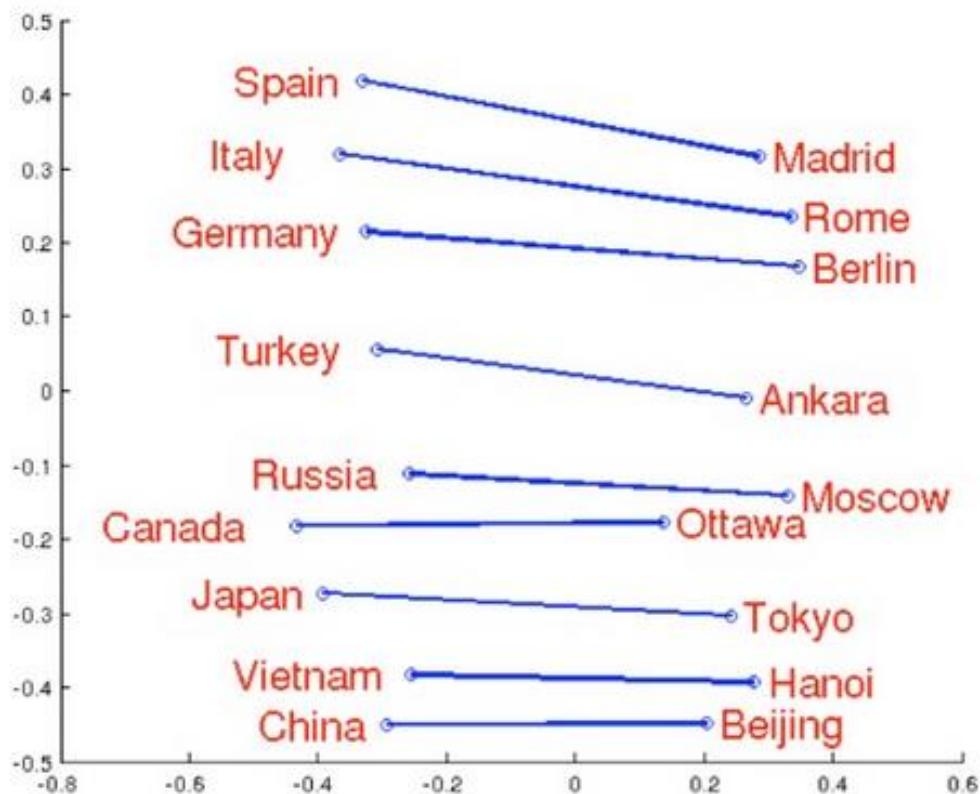
You shall know a word by the company it keeps

馬英九 520宣誓就職

蔡英文 520宣誓就職



# Word Vector



Source: <http://www.slideshare.net/hustwj/cikm-keynotenov2014>

# Word Vector

$$\approx V(Berlin) - V(Rome) + V(Italy)$$

- Characteristics

$$V(hotter) - V(hot) \approx V(bigger) - V(big)$$

$$V(Rome) - V(Italy) \approx V(Berlin) - V(Germany)$$

$$V(king) - V(queen) \approx V(uncle) - V(aunt)$$

- Solving analogies

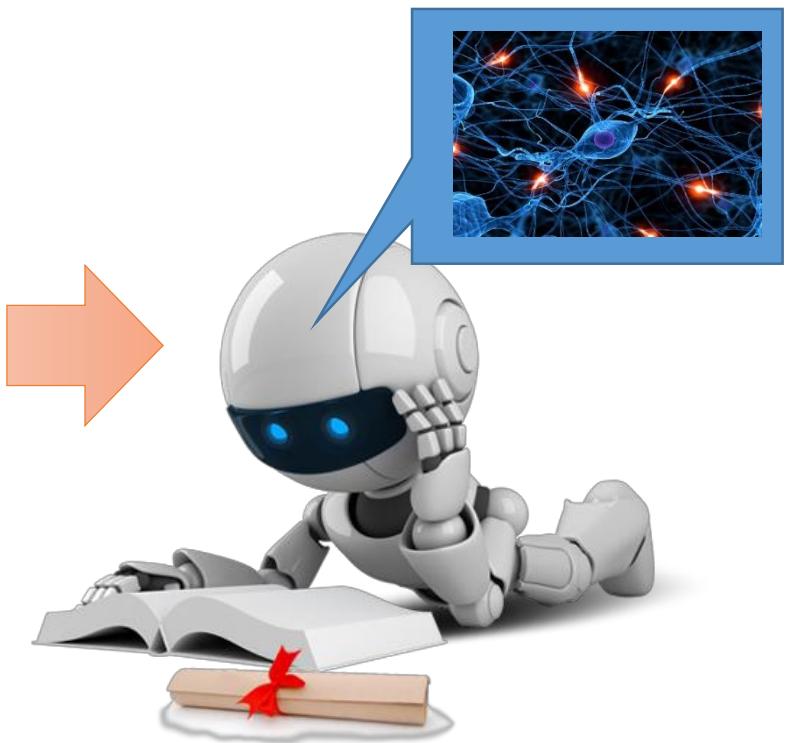
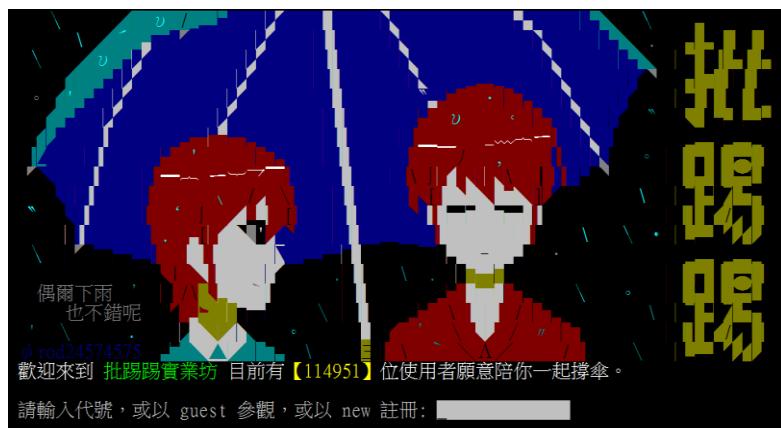
Rome : Italy = Berlin : ?

Compute  $V(Berlin) - V(Rome) + V(Italy)$

Find the word w with the closest  $V(w)$

# Machine Reading

- Machine learn the meaning of words from reading a lot of documents without supervision



# Demo

- Model used in demo is provided by 陳仰德
- Part of the project done by 陳仰德、林資偉
- TA: 劉元銘
- Training data is from PTT (collected by 葉青峰)

# Outline

## Supervised Learning

- Ultra Deep Network
  - Attention Model
- }
- New network structure

## Reinforcement Learning

## Unsupervised Learning

- Image: Realizing what the World Looks Like
- Text: Understanding the Meaning of Words
- Audio: Learning human language without supervision

# Learning from Audio Book



Machine does not have  
any prior knowledge

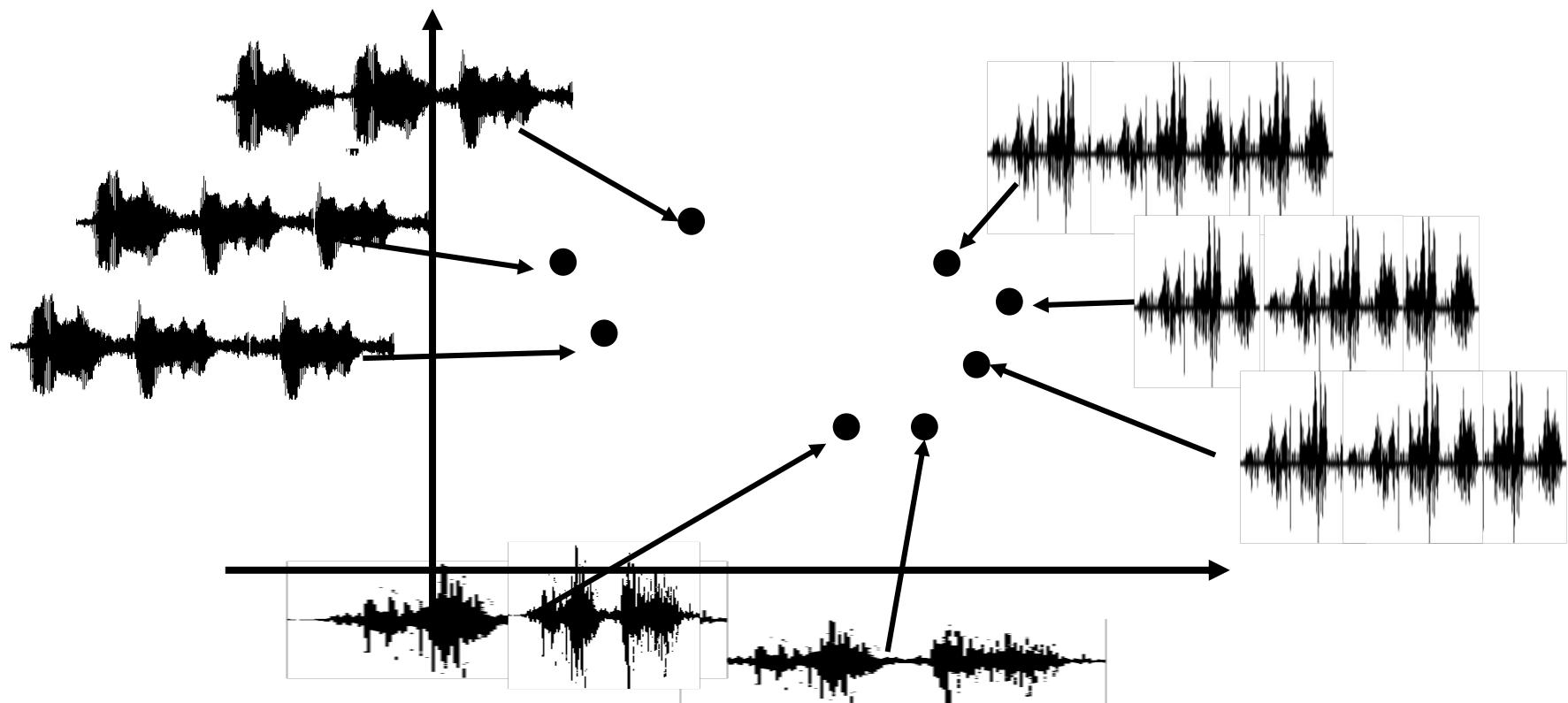
Machine listens to lots of  
audio book

Like an infant

# Audio Word to Vector

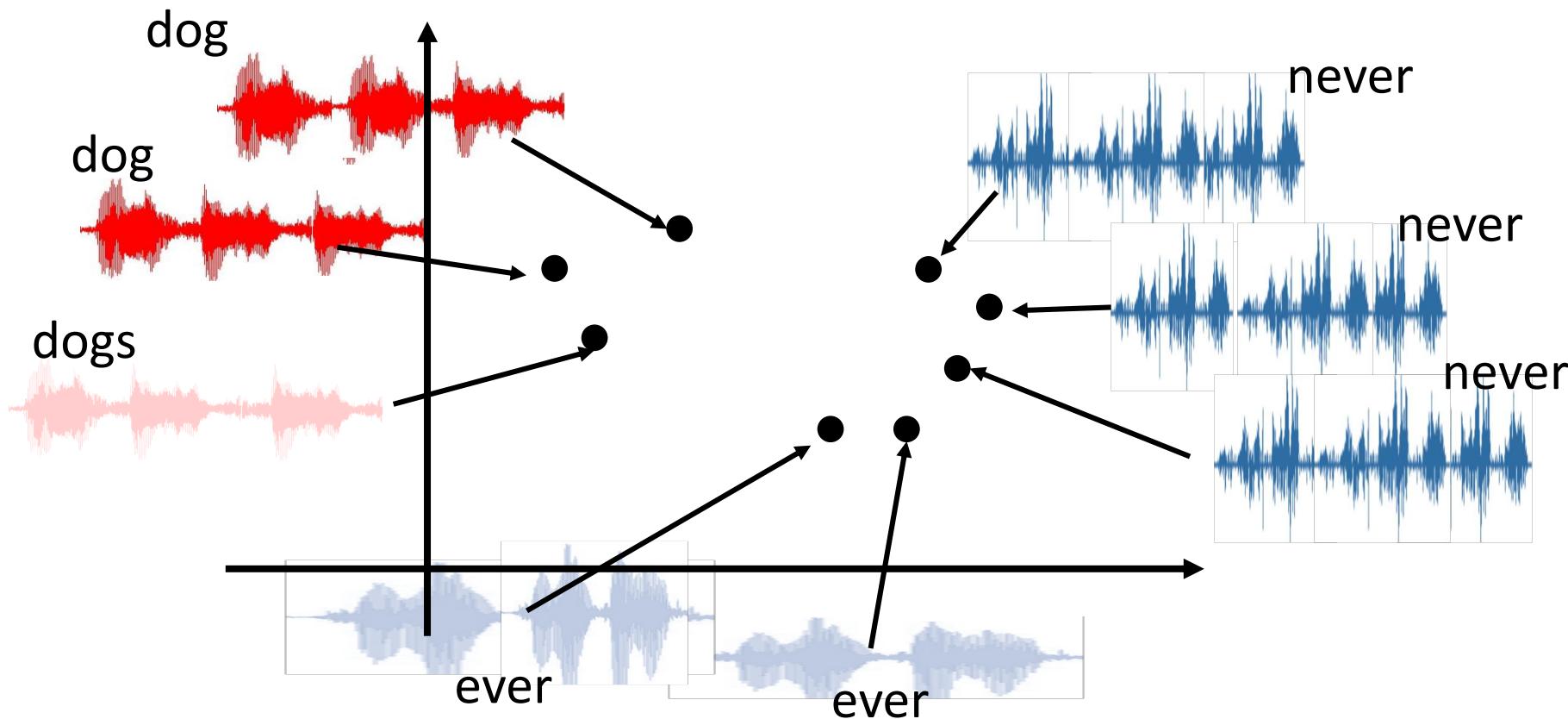
- Audio segment corresponding to an unknown word

→ Fixed-length vector

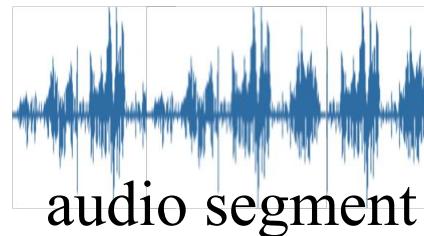


# Audio Word to Vector

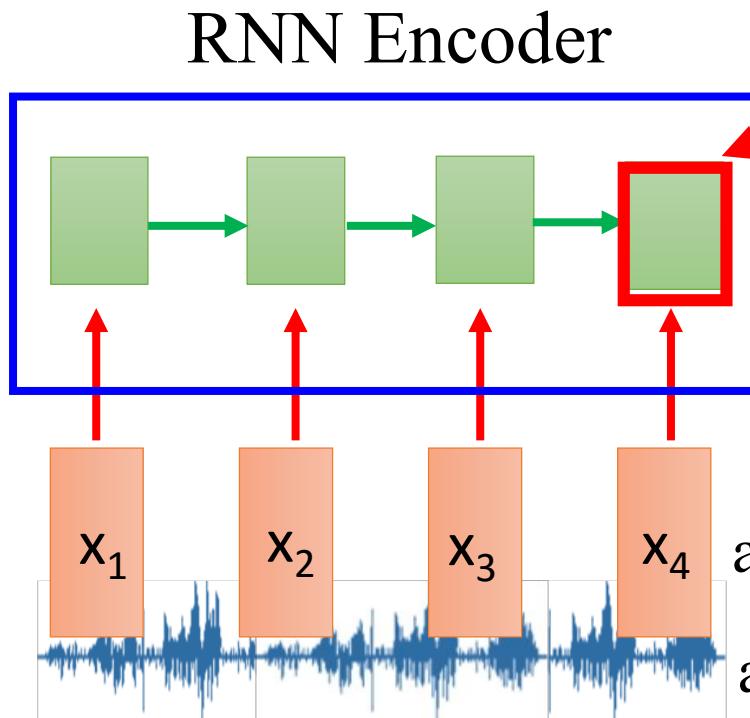
- The audio segments corresponding to words with similar pronunciations are close to each other.



# Sequence-to-sequence Auto-encoder



vector



RNN Encoder

The values in the memory  
represent the whole audio  
segment

The vector we want

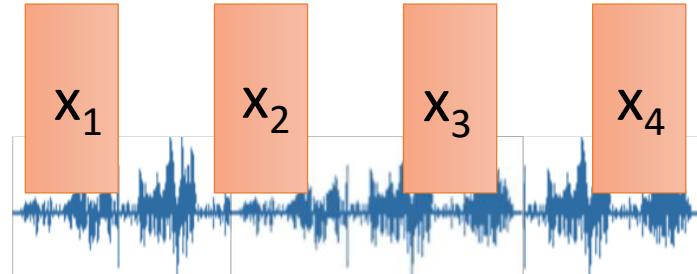
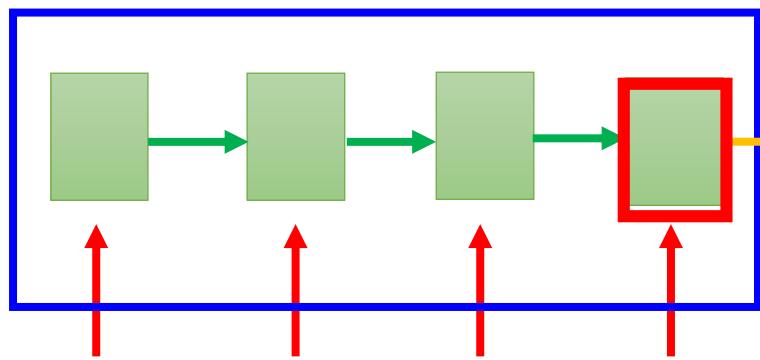
How to train RNN Encoder?

acoustic features  
audio segment

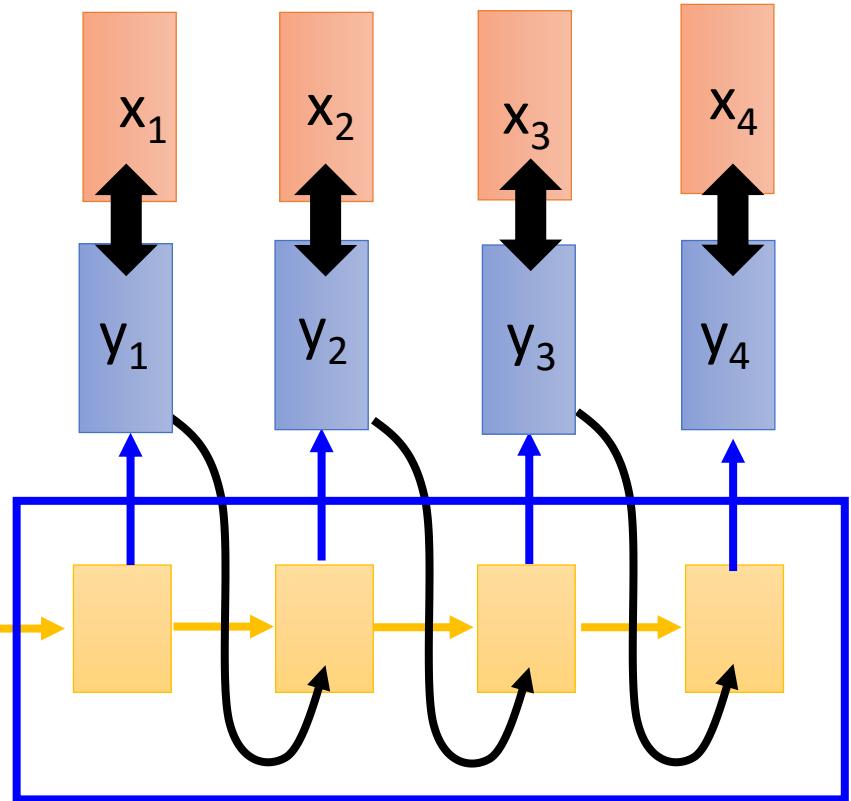
# Sequence-to-sequence Auto-encoder

The RNN encoder and  
decoder are jointly trained.

RNN Encoder



Input acoustic features



RNN Decoder

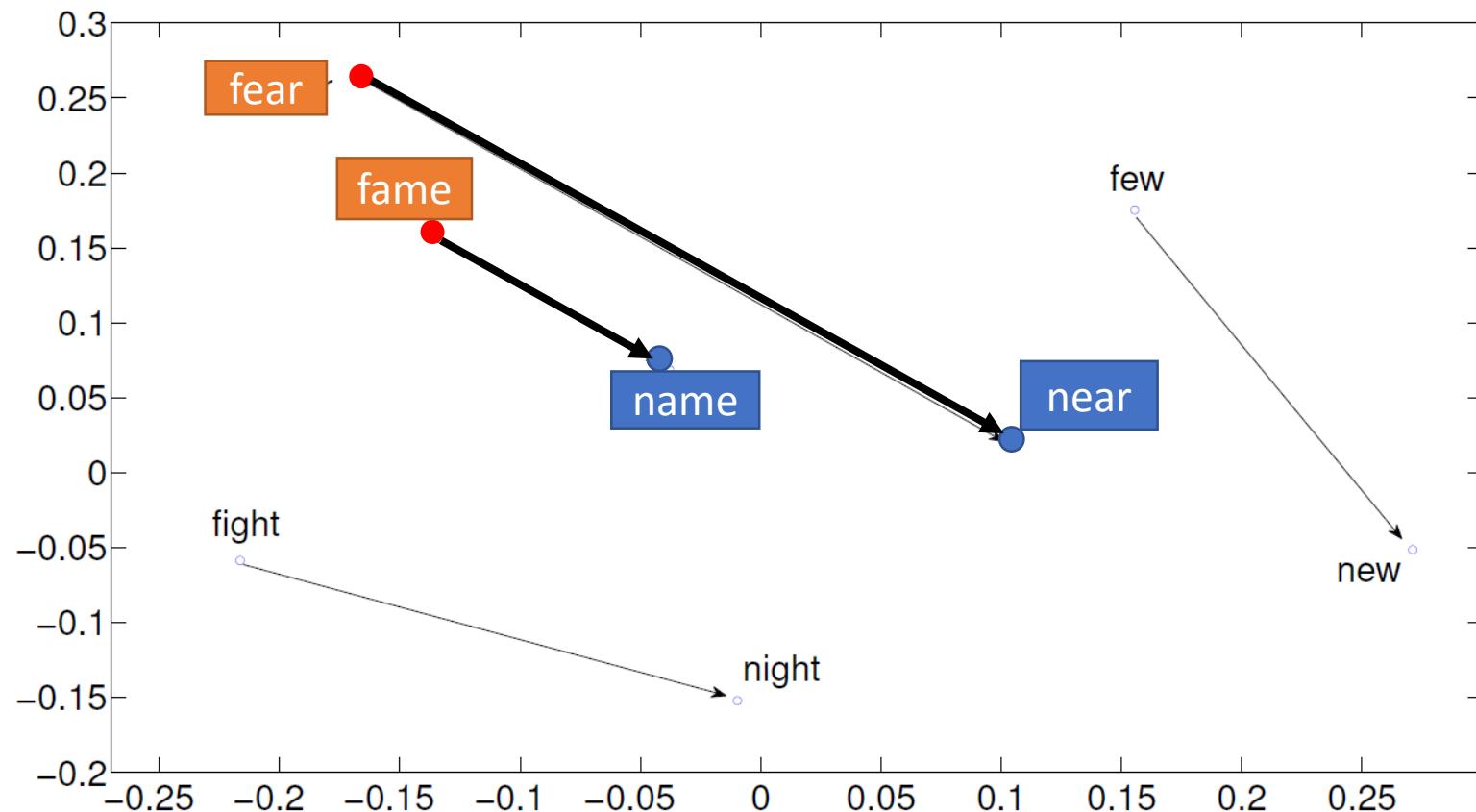
acoustic features

audio segment

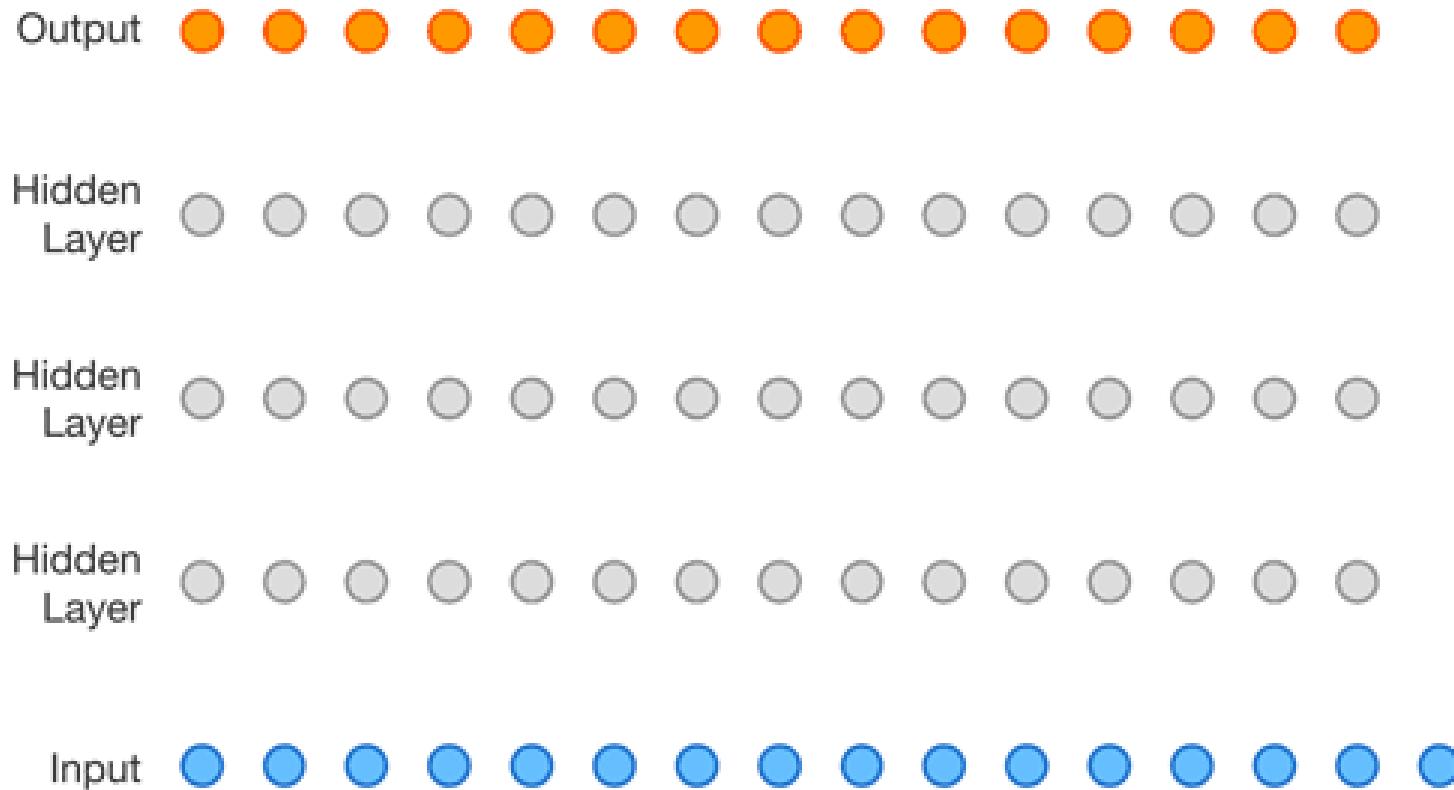
# Audio Word to Vector

## - Results

- Visualizing embedding vectors of the words



# WaveNet (DeepMind)



<https://deepmind.com/blog/wavenet-generative-model-raw-audio/>

# Concluding Remarks

# Concluding Remarks

Lecture I: Introduction of Deep Learning



Lecture II: Tips for Training Deep Neural Network



Lecture III: Variants of Neural Network



Lecture IV: Next Wave

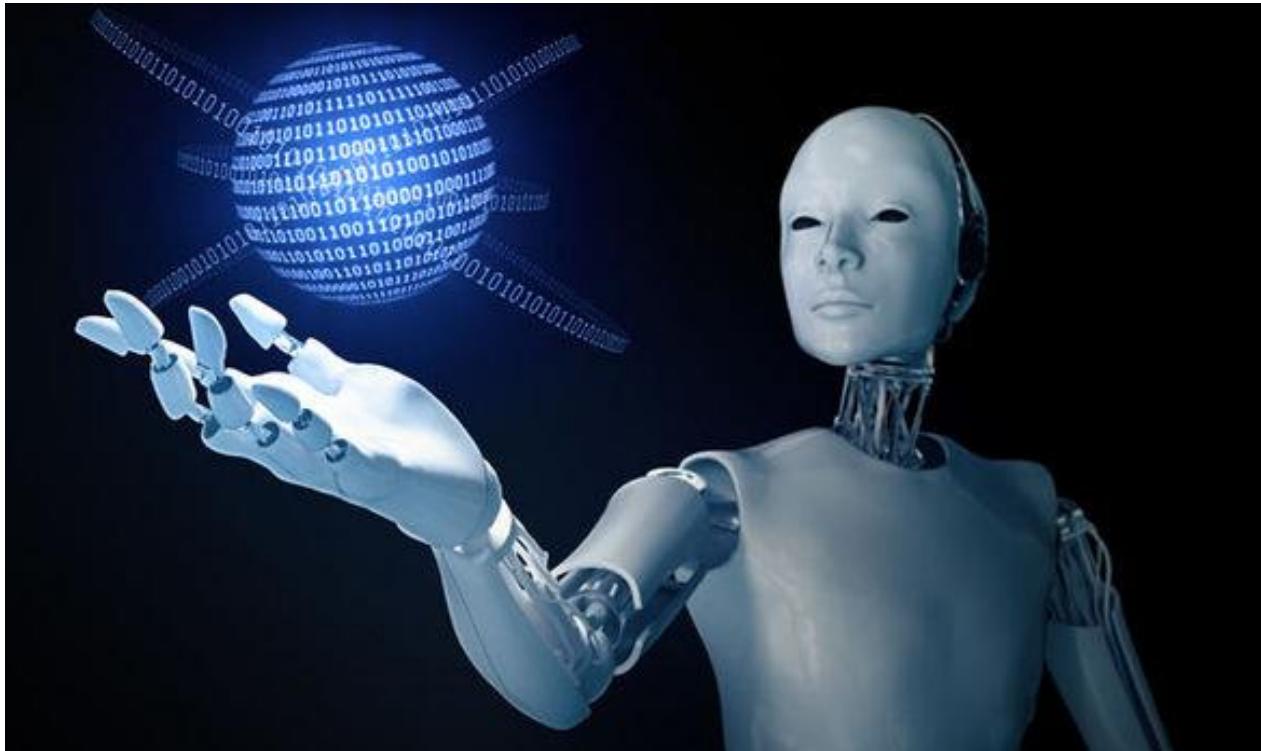
# AI 即將取代多數的工作？

- New Job in AI Age



AI 訓練師

(機器學習專家、  
資料科學家)



<http://www.express.co.uk/news/science/651202/First-step-towards-The-Terminator-becoming-reality-AI-beats-champ-of-world-s-oldest-game>

# AI 訓練師



機器不是自己會學嗎？  
為什麼需要 AI 訓練師

戰鬥是寶可夢在打，  
為什麼需要寶可夢訓練師？

# AI 訓練師



## 寶可夢訓練師

- 寶可夢訓練師要挑選適合的寶可夢來戰鬥
  - 寶可夢有不同的屬性
- 召喚出來的寶可夢不一定能操控
  - E.g. 小智的噴火龍
  - 需要足夠的經驗

## AI 訓練師

- 在 step 1，AI 訓練師要挑選合適的模型
  - 不同模型適合處理不同的問題
- 不一定能在 step 3 找出 best function
  - E.g. Deep Learning
  - 需要足夠的經驗

# AI 訓練師

- 厲害的 AI ， AI 訓練師功不可沒
- 讓我們一起朝 AI 訓練師之路邁進



以神奇寶貝大師為目標一直進行著修煉



台大電機系 資料科學與智慧網路組 首屆招生：

碩士生甄試20名，考試入學10名

博士生甄試2名，考試入學1名



招生公告：105.09.20



報名期間：105.10.04 ~ 10.12



招生網址：

<https://comm.ntu.edu.tw/new/Master.php>



招生說明會：

時間：105.09.28 12:20

地點：台大博理館 112 R



國立臺灣大學  
National Taiwan University  
電信工程學研究所丙組  
資料科學與智慧網路組

Pythagorean  
Learning Center  
Department of  
Electrical Engineering  
National Taiwan University  
NTU-GICE  
Data Science  
Smart Networking  
Machine Learning & Deep learning  
Cloud Computing  
Blockchain  
Big Data