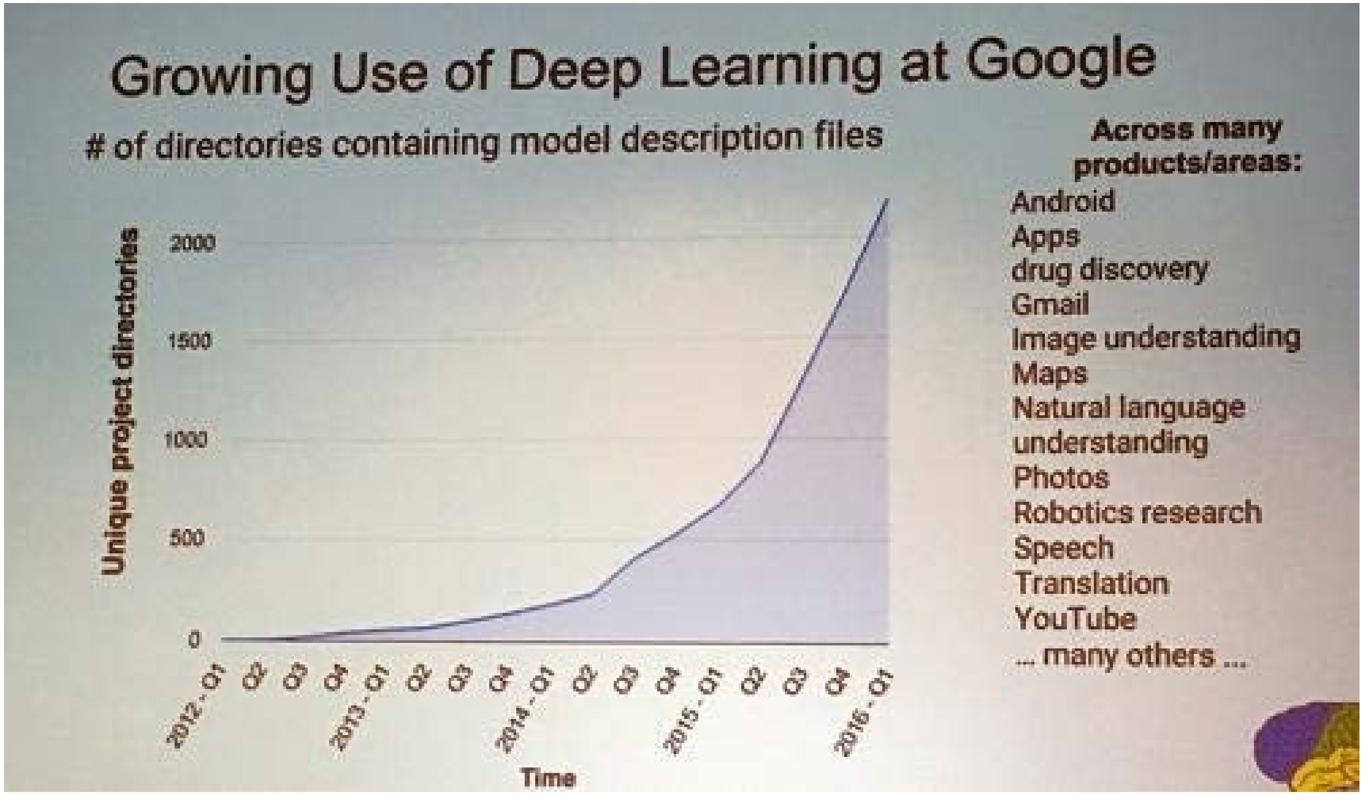
Deep Learning Tutorial

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

Three Steps for Deep Learning

Step 1: Network Structure

Step 2: Learning Target

Step 3: Learn!

天生的腦

http://onepiece1234567890.blogspot.tw/2013/12/blog-post_8.html

Three Steps for Deep Learning

Step 1: Network Structure

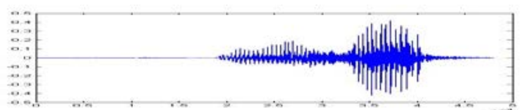
Step 2: Learning Target

Step 3: Learn!

based on
training data

Three Steps for Deep Learning

- Speech Recognition

$f*($  $)=$ “你好”

- Handwritten Recognition

$f*($  $)=$ “2”

- Playing Go

$f*($  $)=$ “5-5”
(step)

- Dialogue System

$f*($ “Hi” $)=$ “Hello”
(what the user said) (system response)

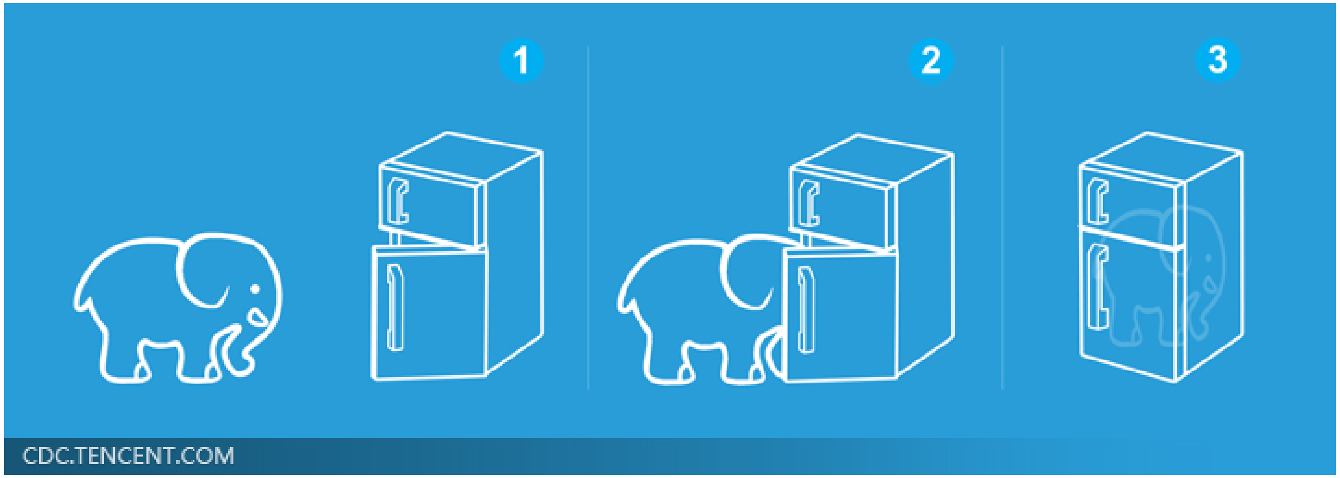
Step 3: Learn!

Pick the best function f*

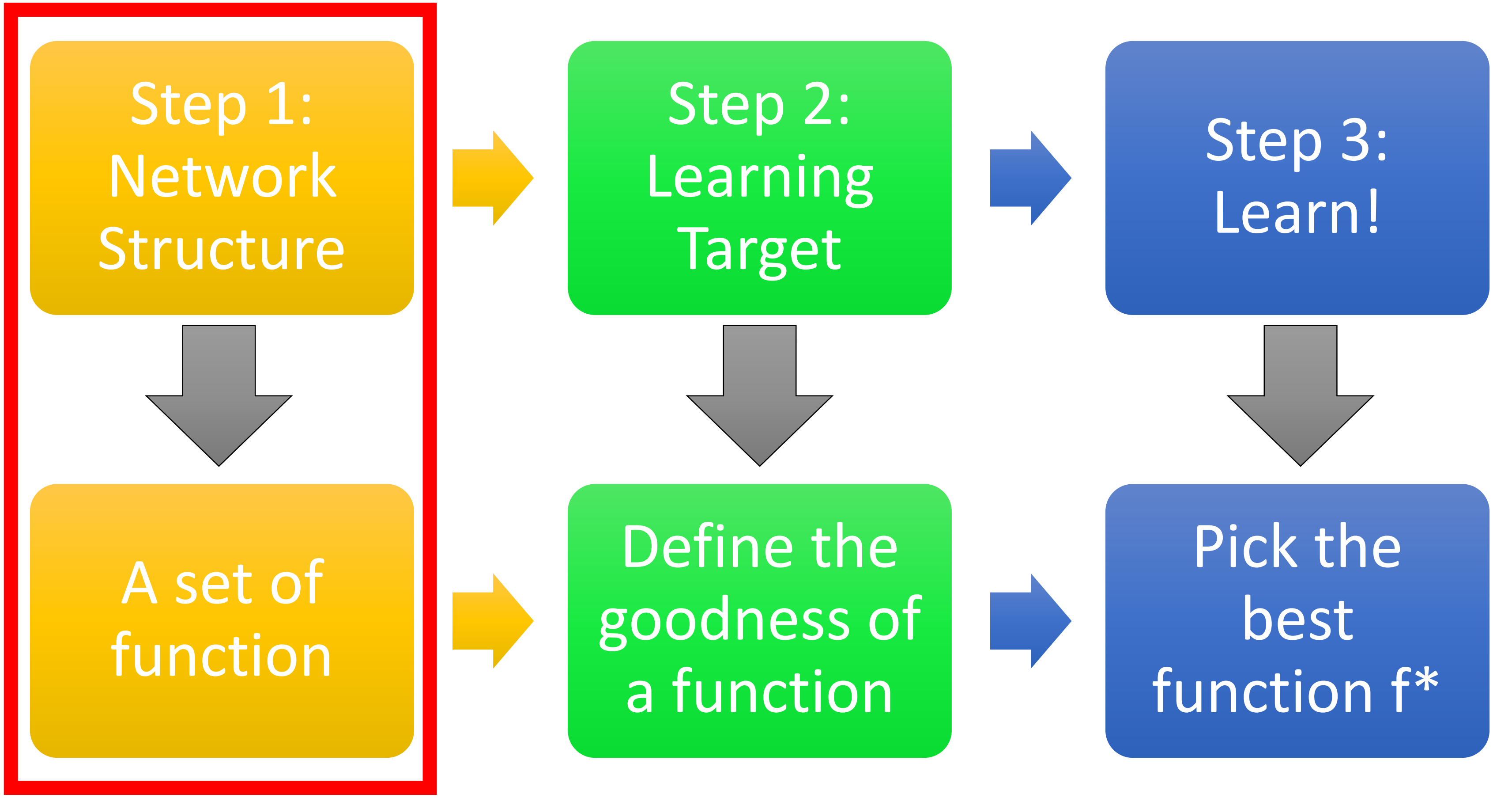
Three Steps for Deep Learning

Step 1: Network Structure → Step 2: Learning Target → Step 3: Learn!

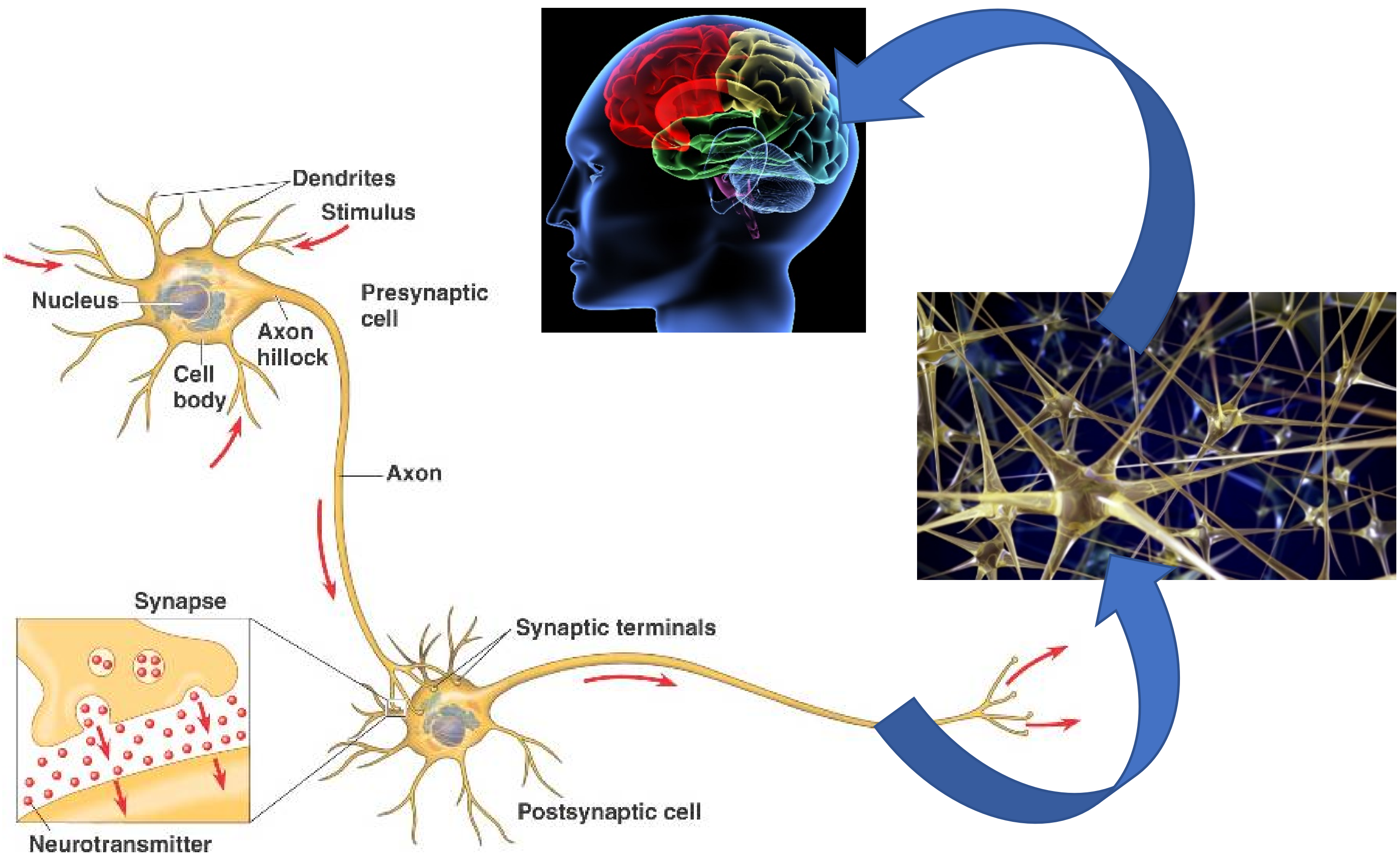
Deep Learning is so simple ……



Three Steps for Deep Learning



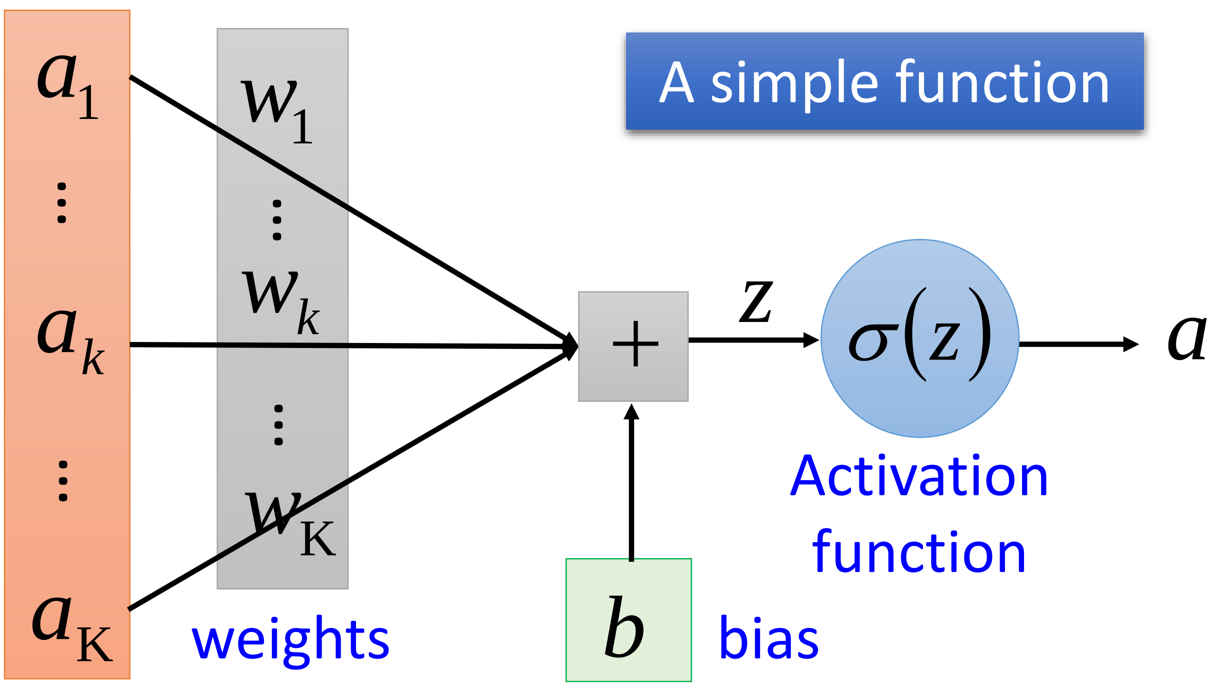
Human Brains



Neural Network

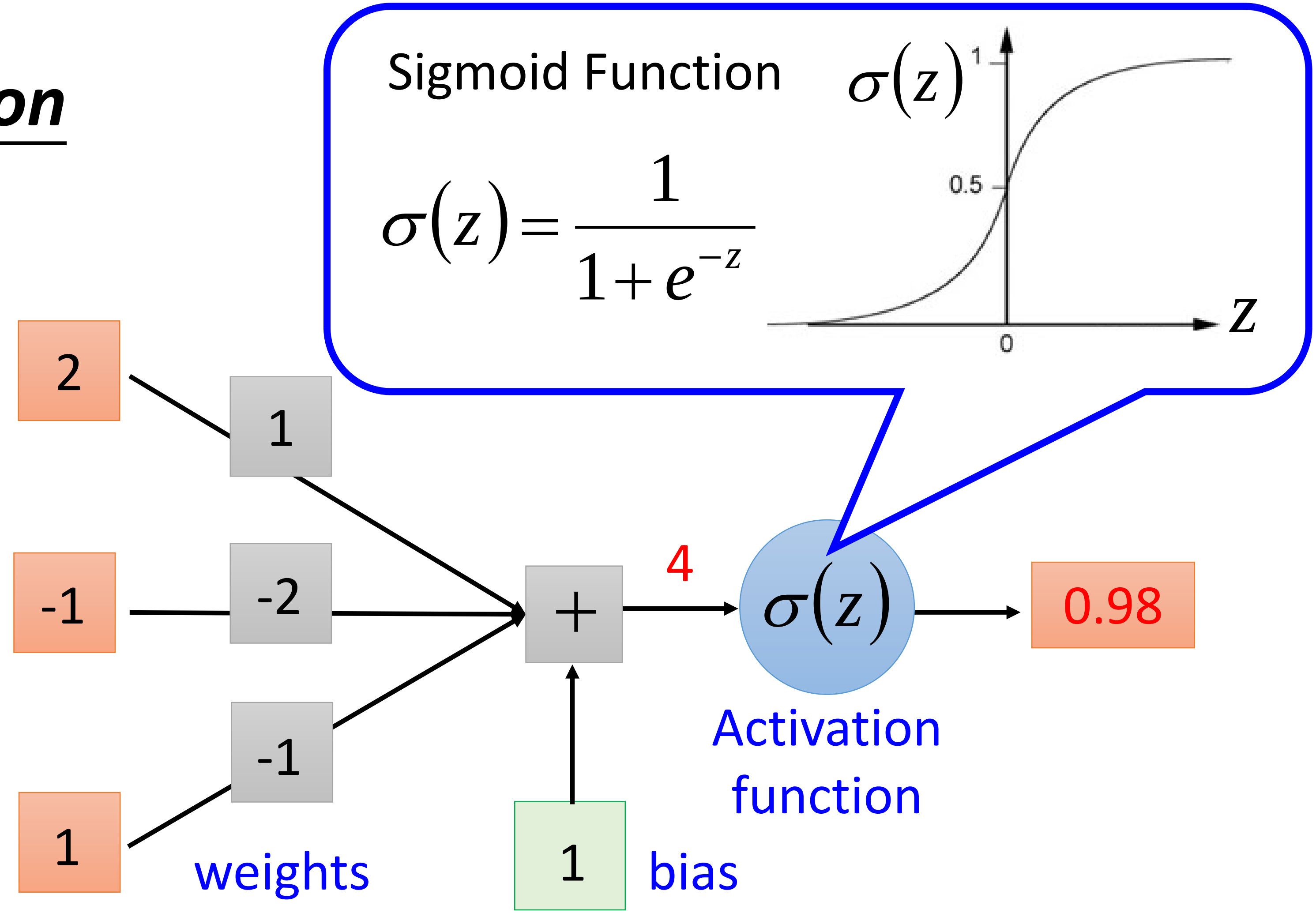
Neuron

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

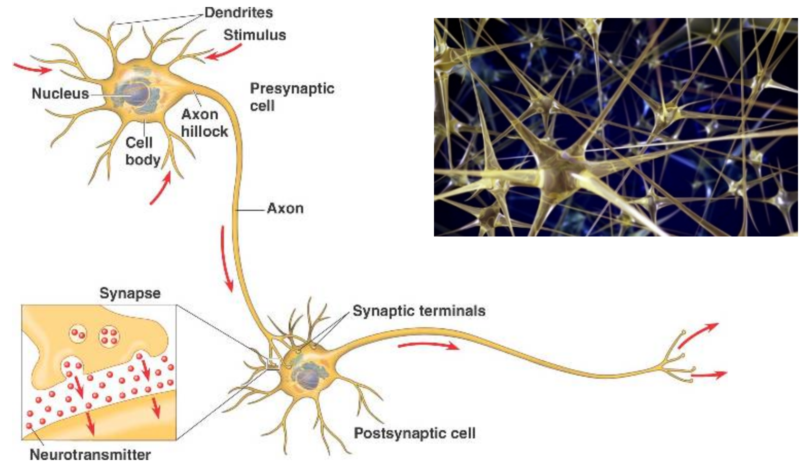


Neural Network

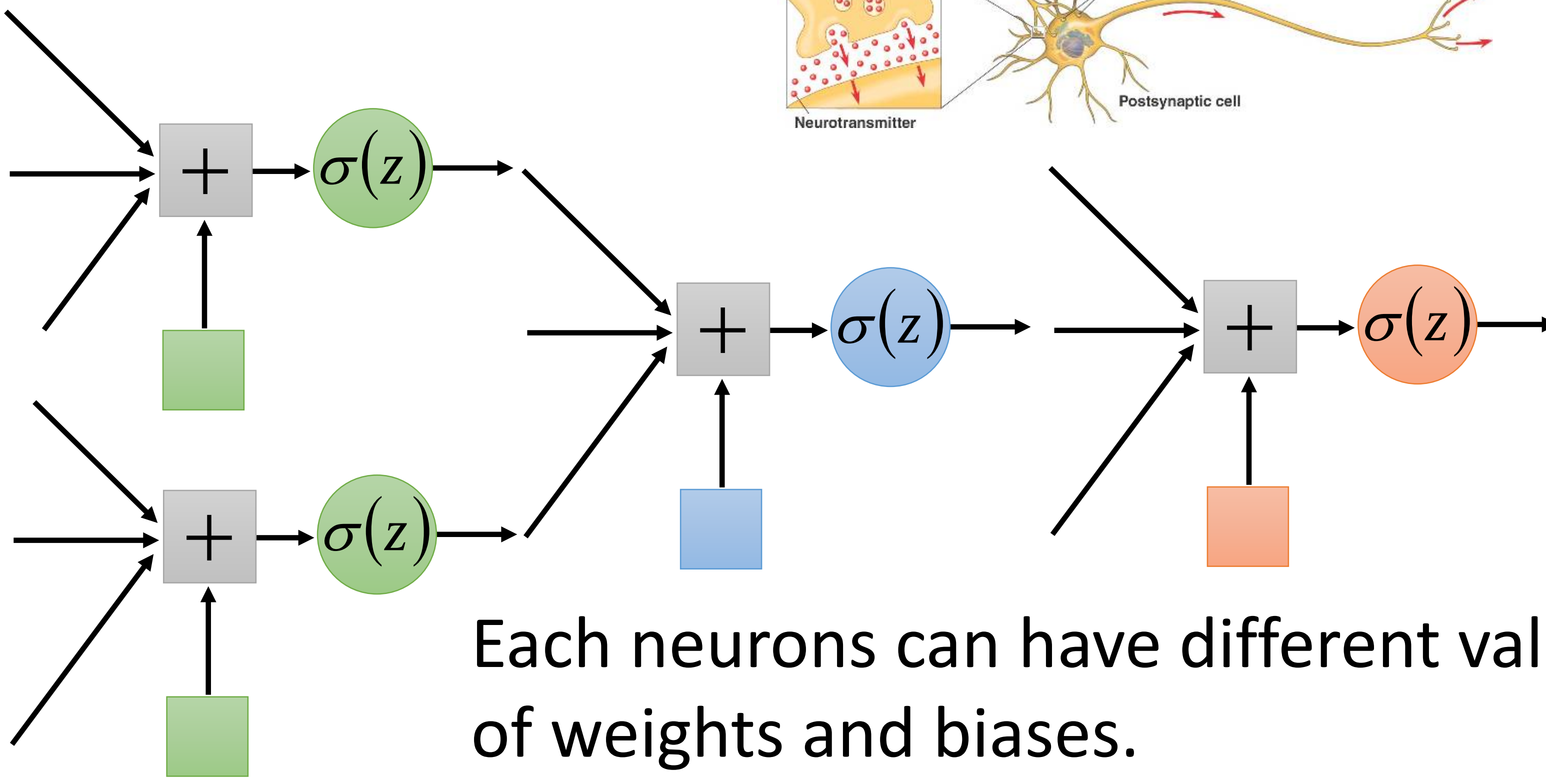
Neuron



Neural Network



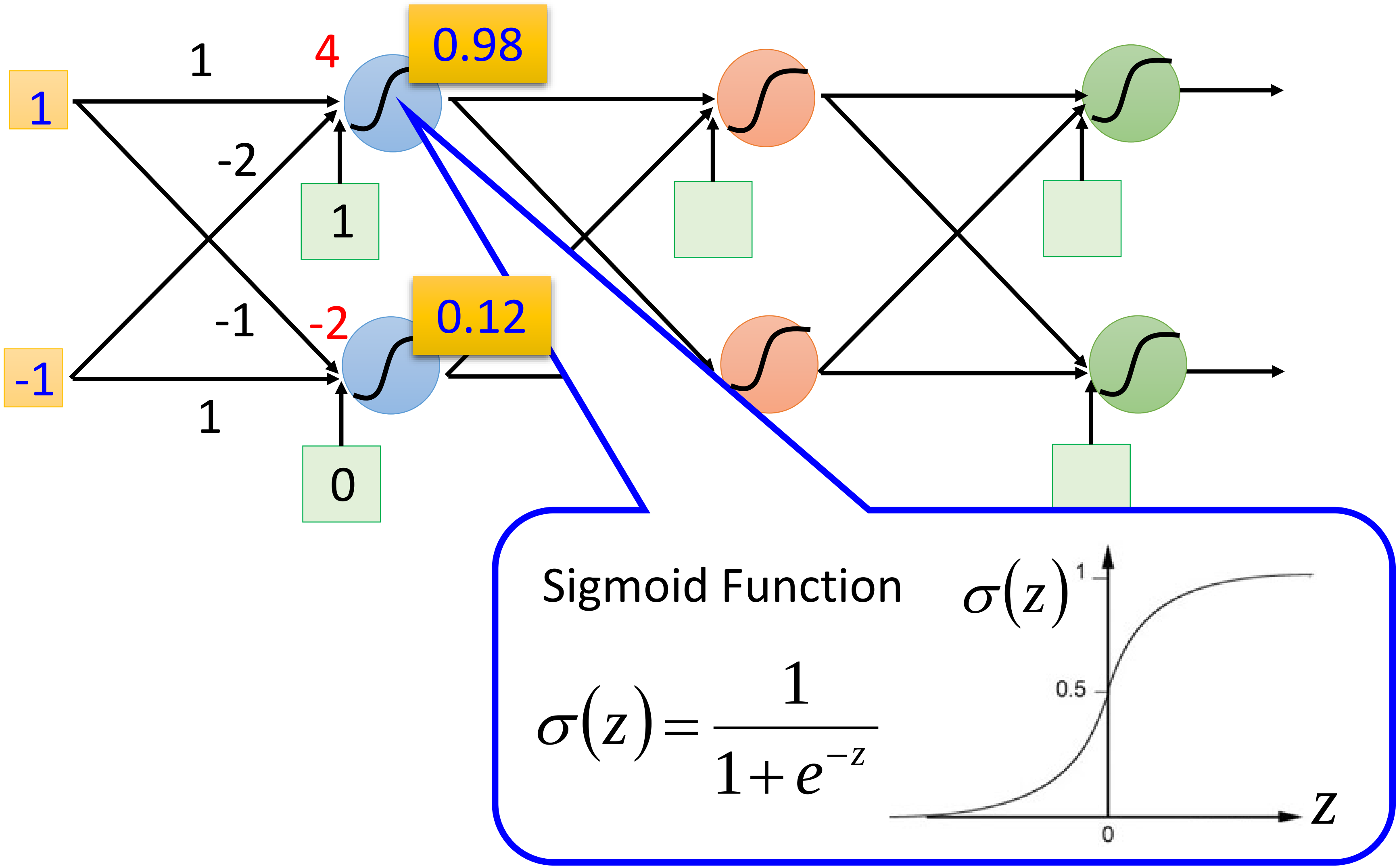
Different connections leads to different network structured



Each neurons can have different values of weights and biases.

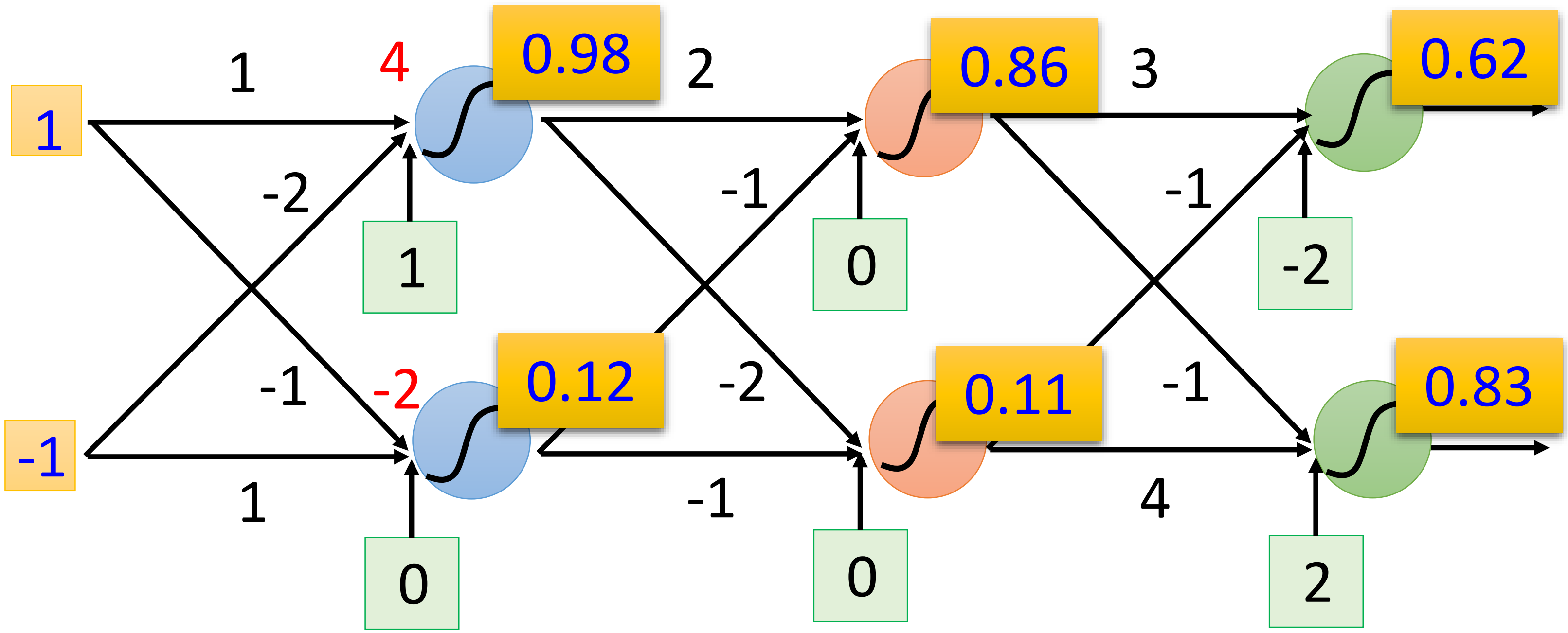
Weights and biases are network parameters θ

Fully Connect Feedforward Network

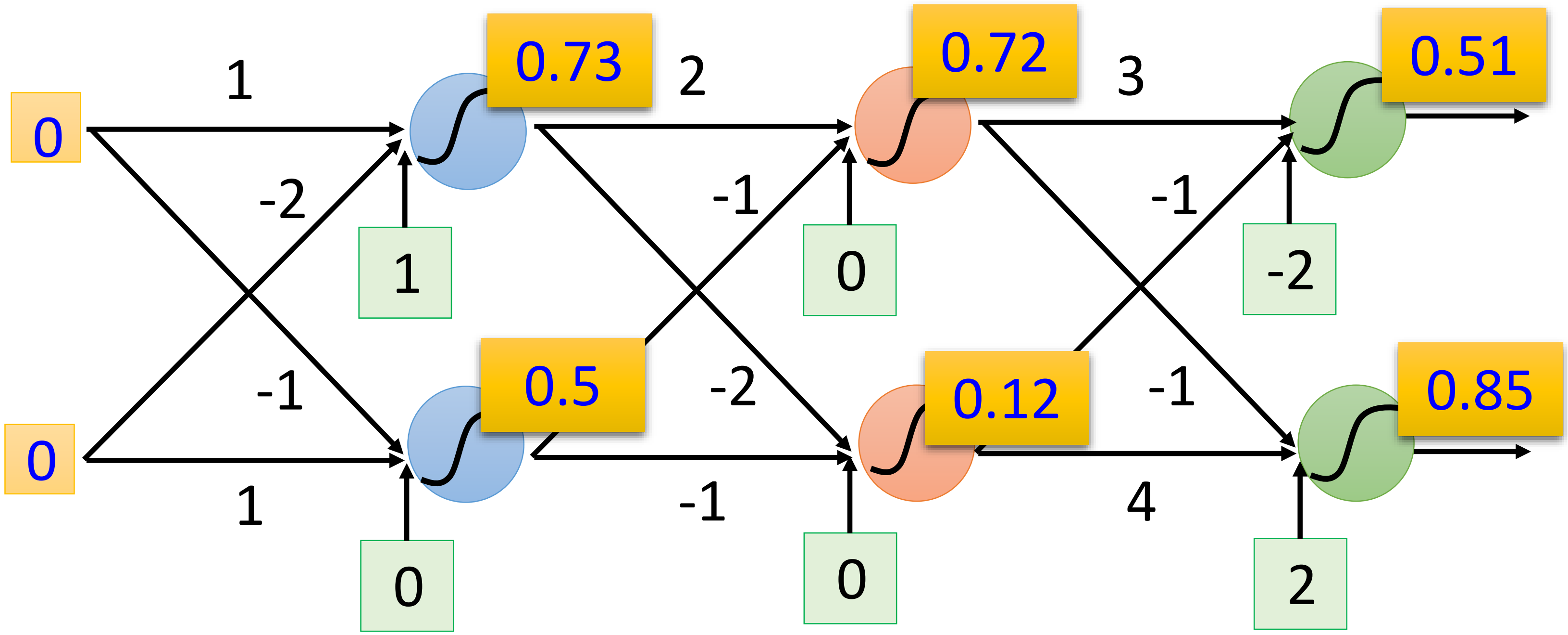


$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Fully Connect Feedforward Network



Fully Connect Feedforward Network



Network is a function.

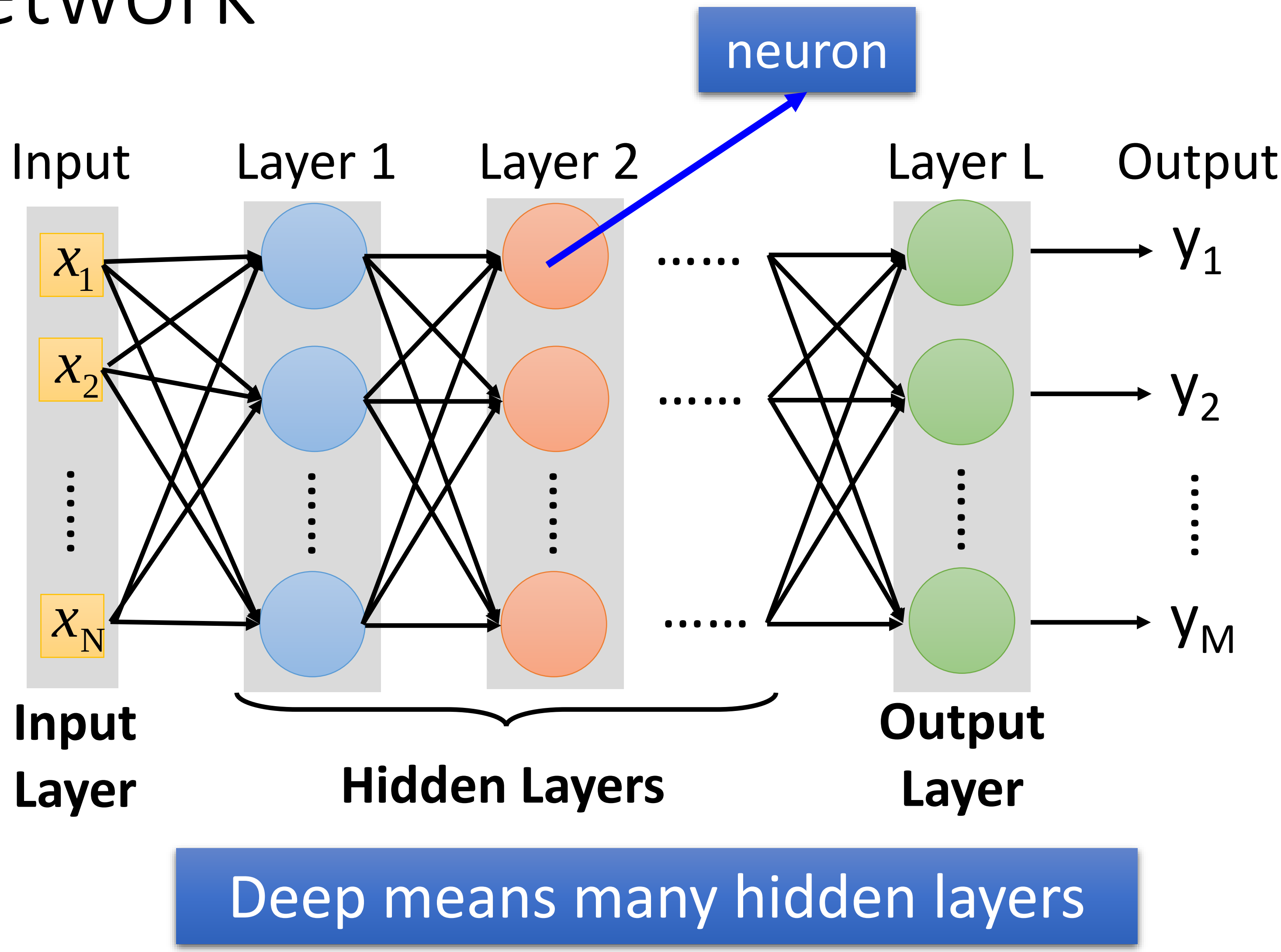
Input vector, output vector

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

Given parameters θ, define a function

Given network structure, define ***a function set***

Fully Connect Feedforward Network

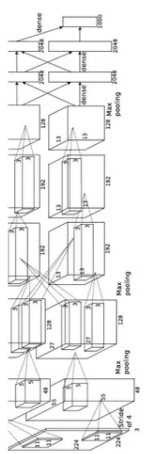


Ultra Deep Network

http://cs231n.stanford.edu/slides/winter1516_lecture8.pdf

8 layers

16.4%



AlexNet (2012)

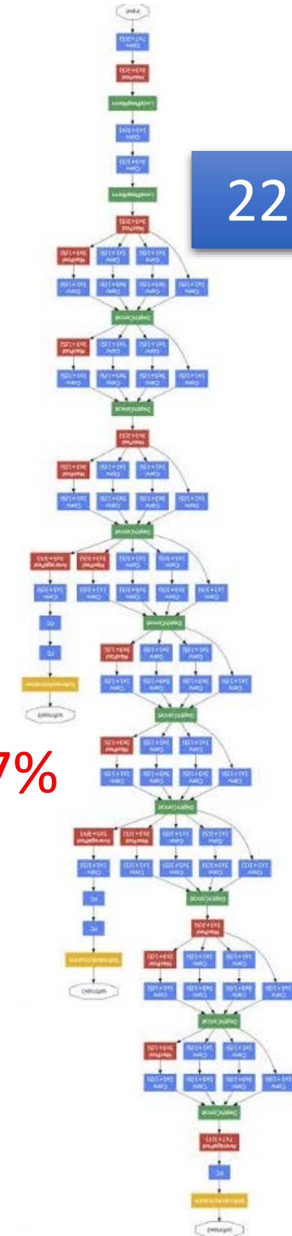
19 layers

7.3%

VGG (2014)

22 layers

6.7%

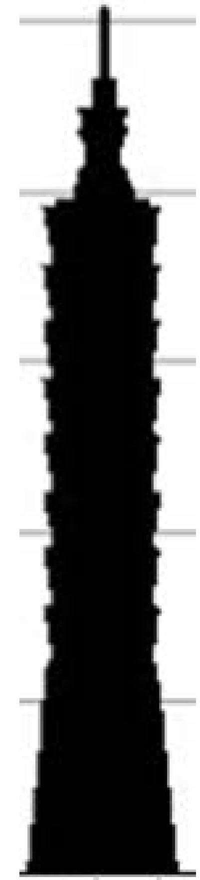


GoogleNet (2014)

Ultra Deep Network

This ultra deep network
have special structure.

(Lecture IV)



152 layers

101 layers

3.57%

16.4%

7.3%

6.7%

AlexNet (2012)

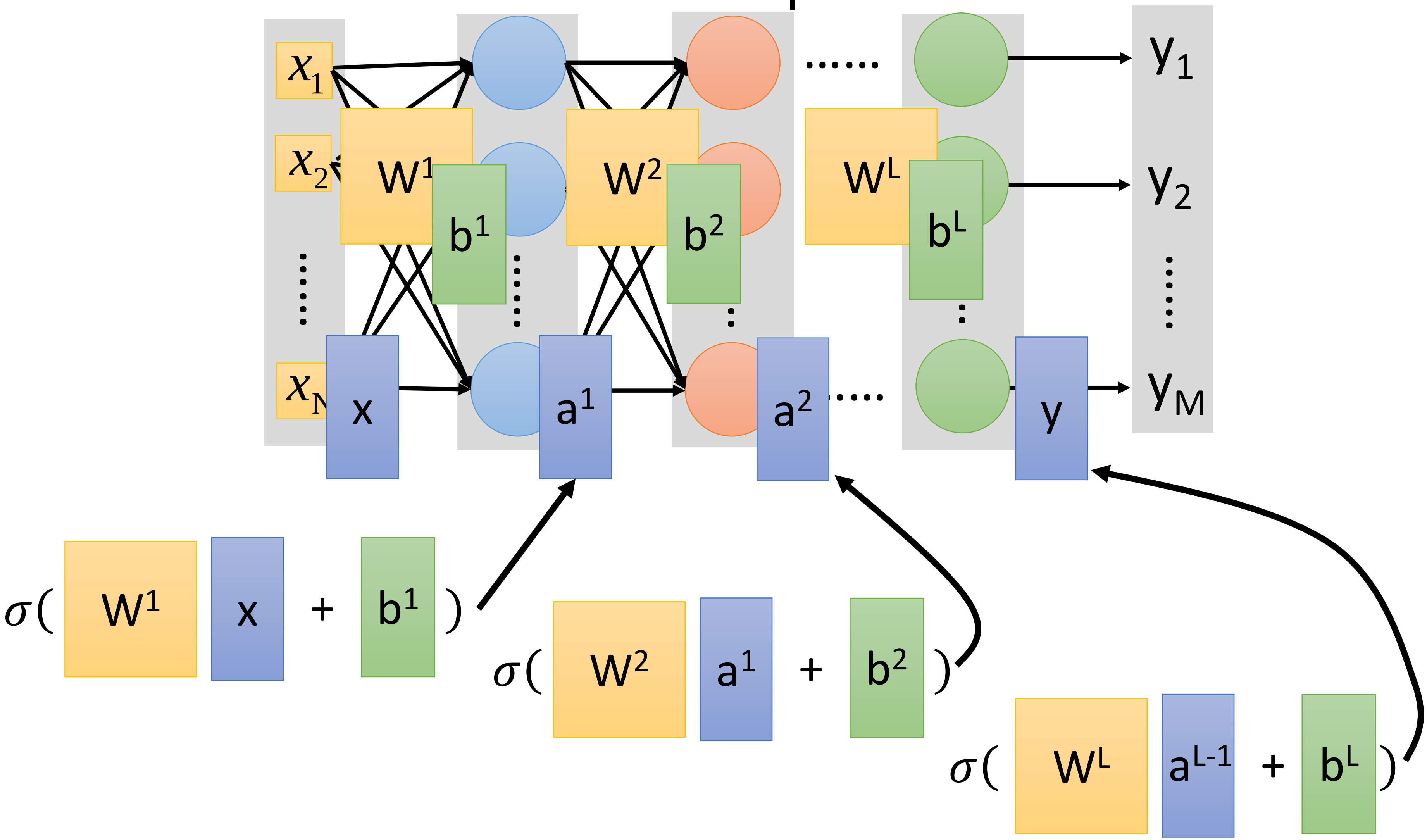
VGG (2014)

GoogleNet (2014)

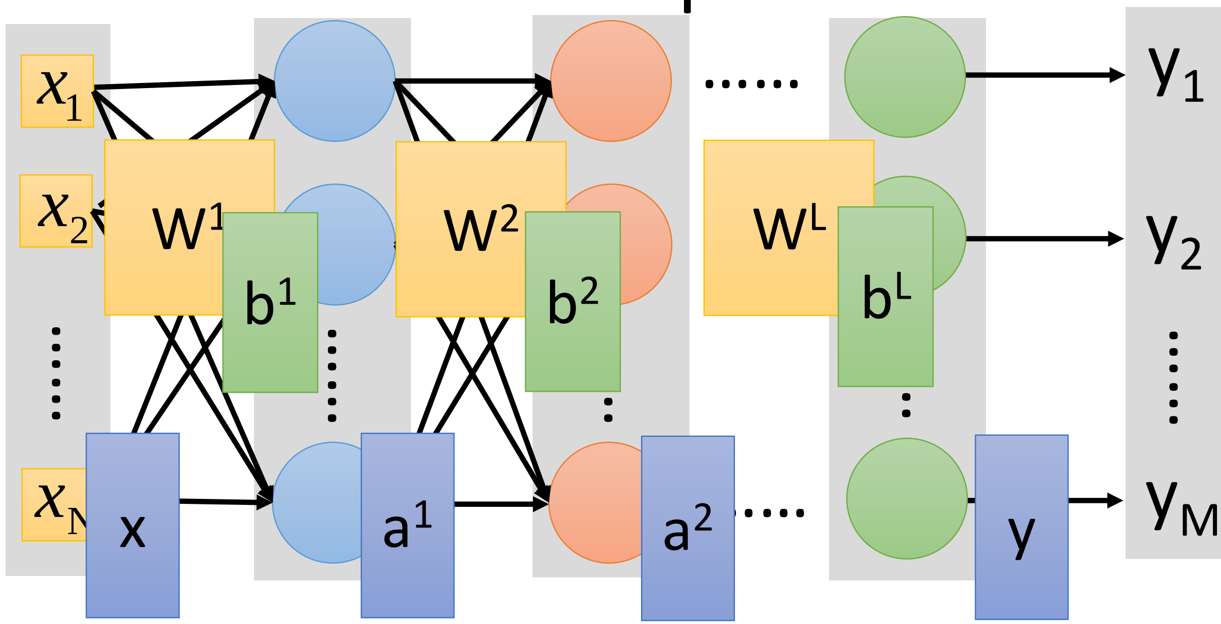
Residual Net (2015)

Taipei 101

Fully Connect Feedforward Network - Matrix Operation



Fully Connect Feedforward Network - Matrix Operation



$$\mathrm{y} = f(\mathrm{x})$$

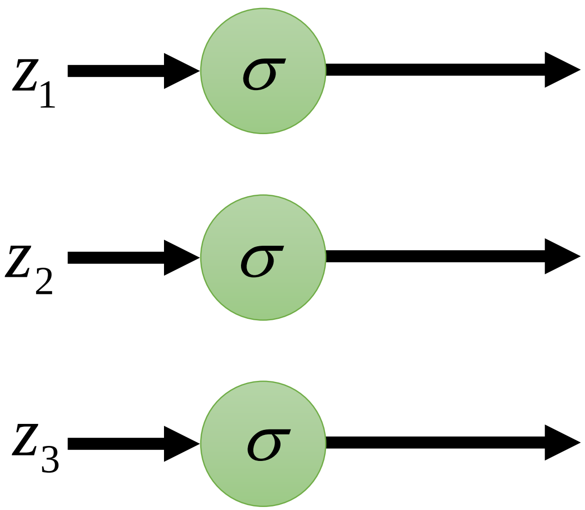
Using parallel computing techniques (e.g. GPU) to speed up matrix operation

$$= \sigma(W^L \cdots \sigma(W^2 \; \sigma(W^1 \; \mathrm{x} + b^1) + b^2) \cdots + b^L)$$

Output Layer (Option)

- Softmax layer as the output layer

<u>Ordinary Layer</u>



$$y_1 = \sigma(z_1)$$

$$y_2 = \sigma(z_2)$$

$$y_3 = \sigma(z_3)$$

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

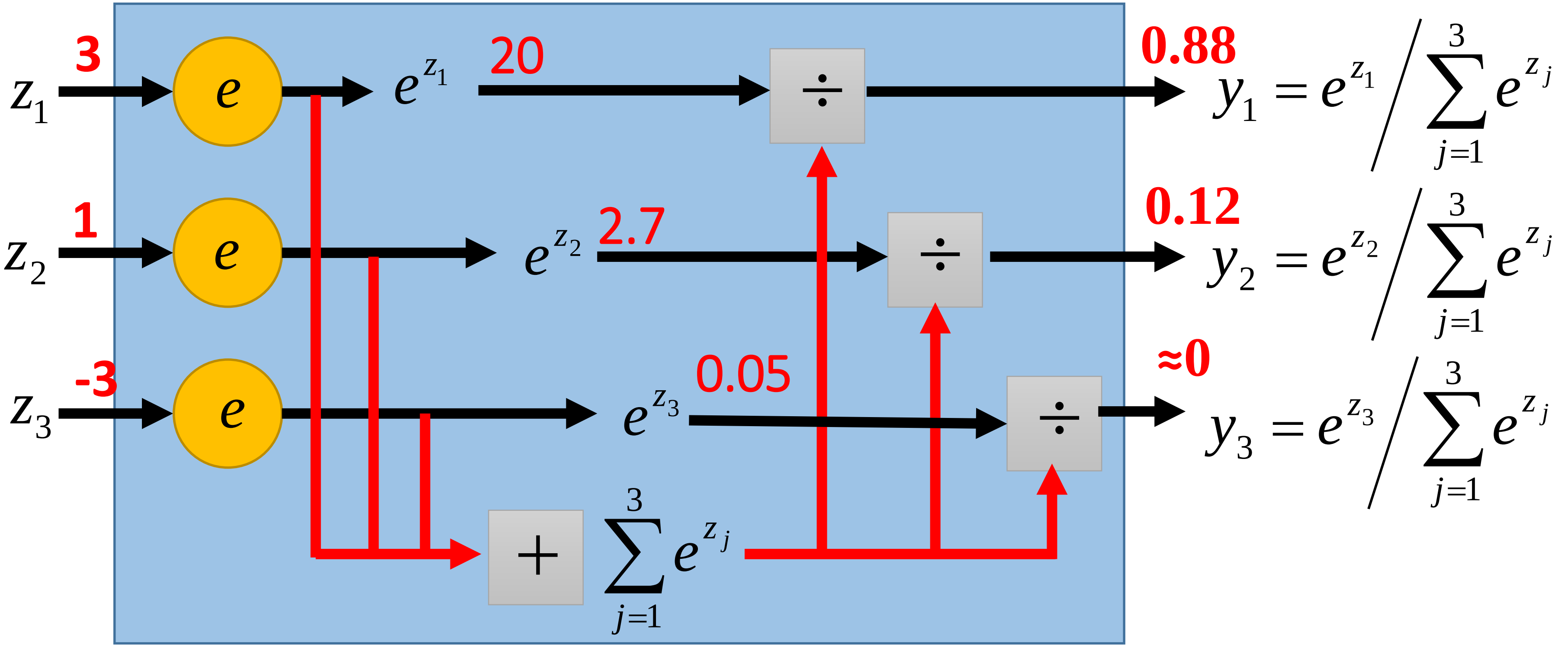
May not be easy to interpret

Output Layer (Option)

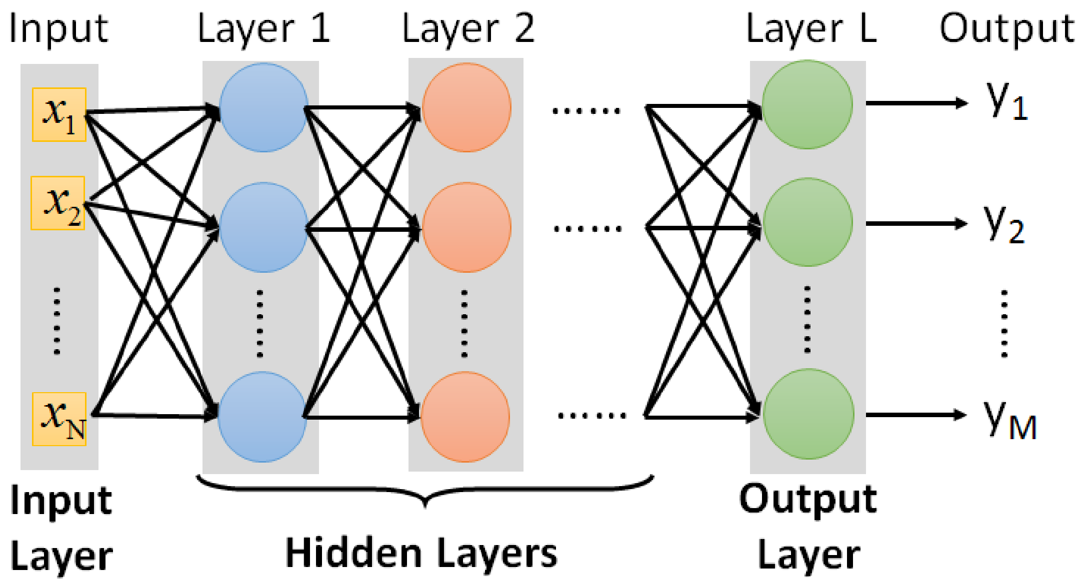
- Softmax layer as the output layer

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

Softmax Layer



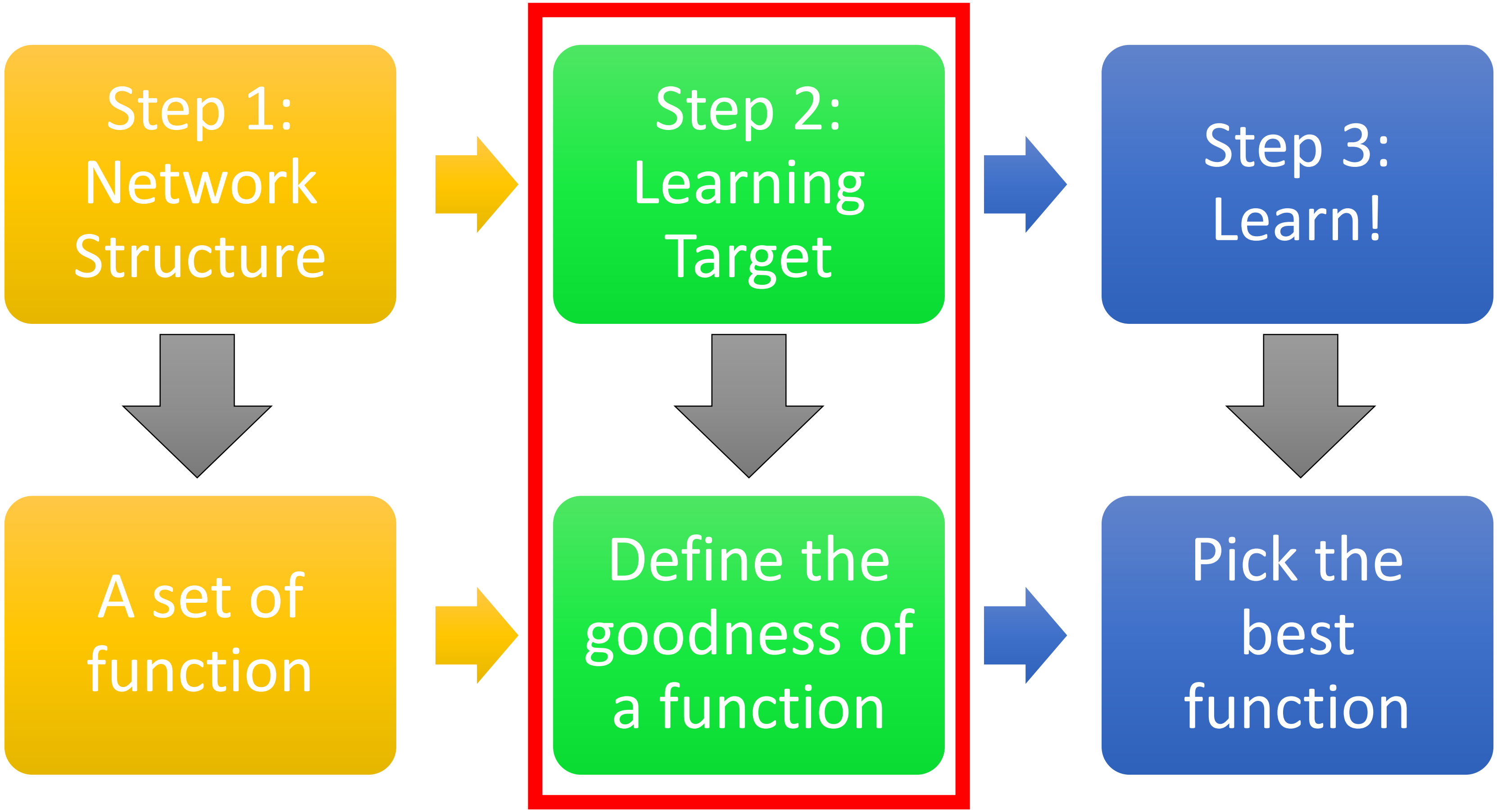
FAQ



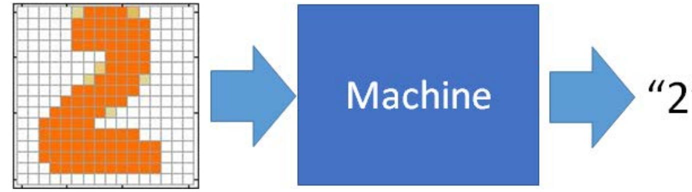
- Q: How many layers? How many neurons for each layer?

- Q: Can the structure be automatically determined?

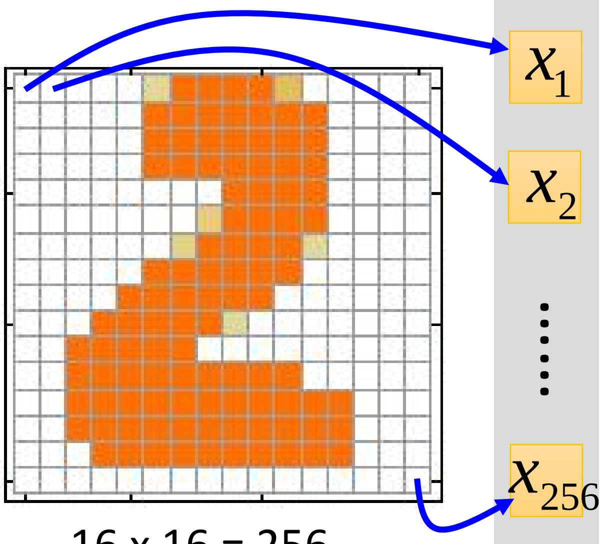
Three Steps for Deep Learning



Example Application



Input

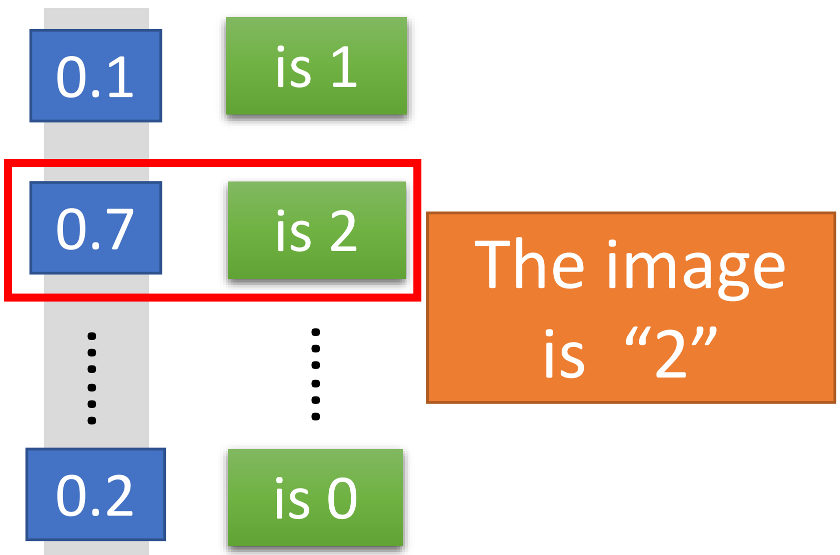


16 x 16 = 256

Ink → 1

No ink → 0

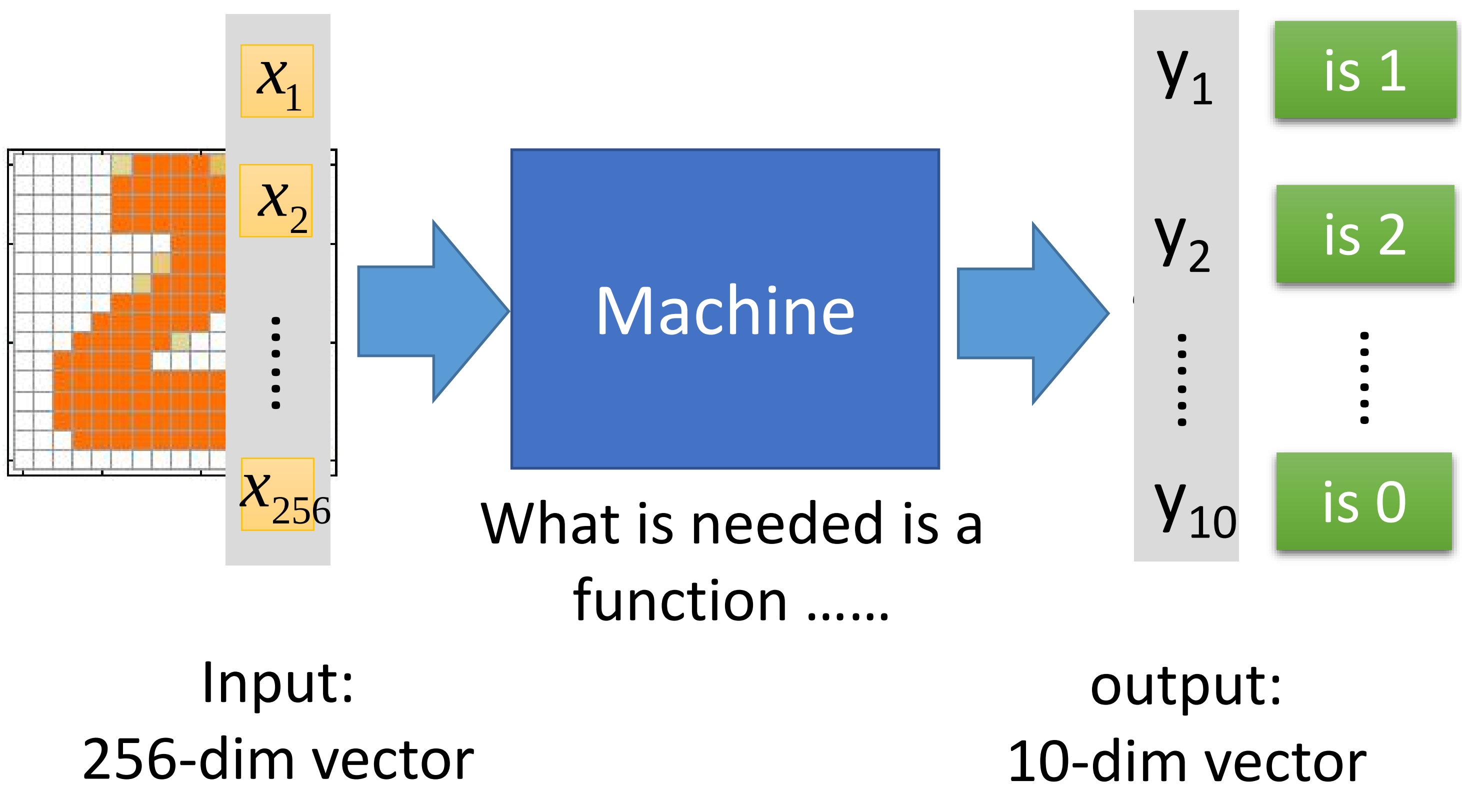
Output



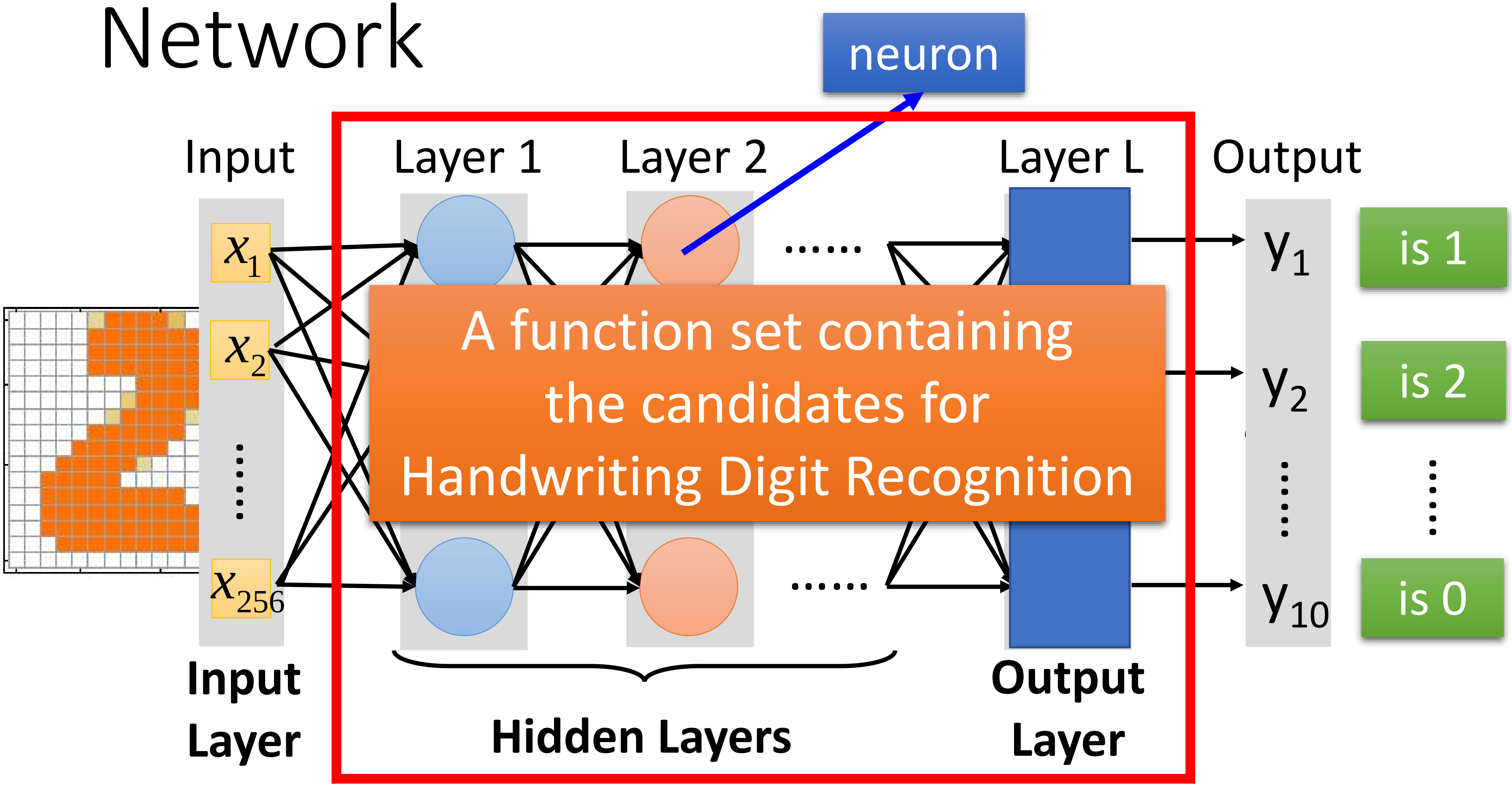
Each dimension represents the confidence of a digit.

Example Application

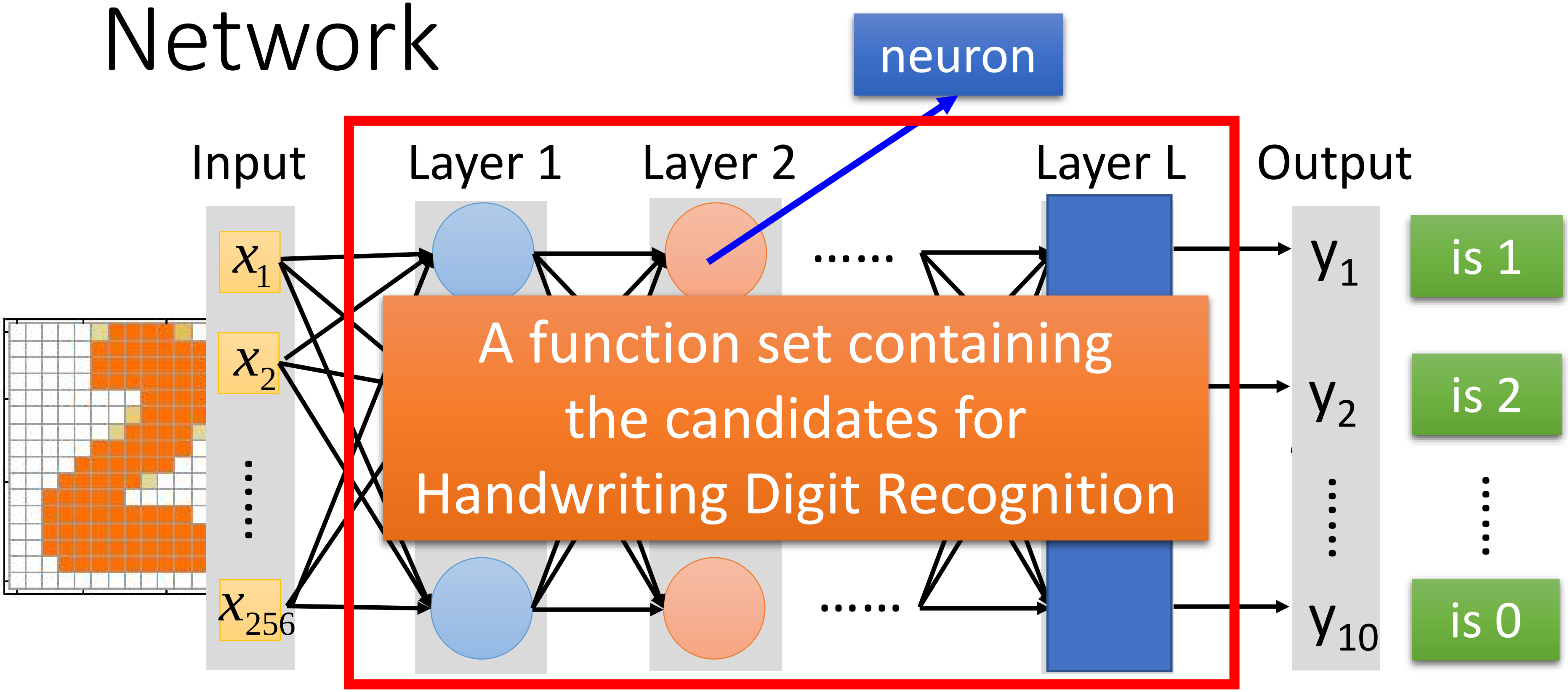
- Handwriting Digit Recognition



Fully Connect Feedforward Network



Fully Connect Feedforward Network

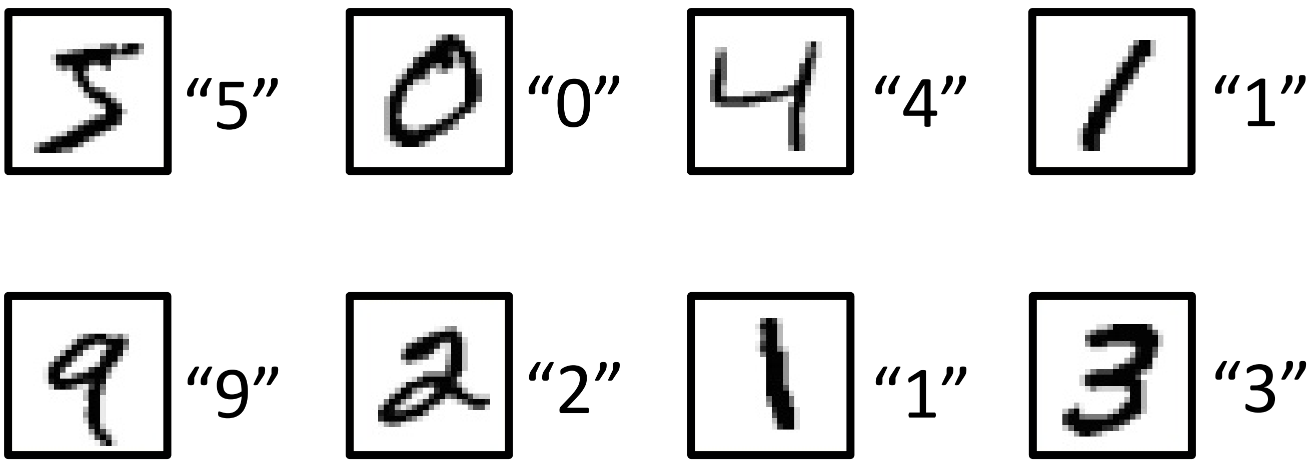


Step 2 Define the goodness of function based on training data

Step 3 Pick the best function

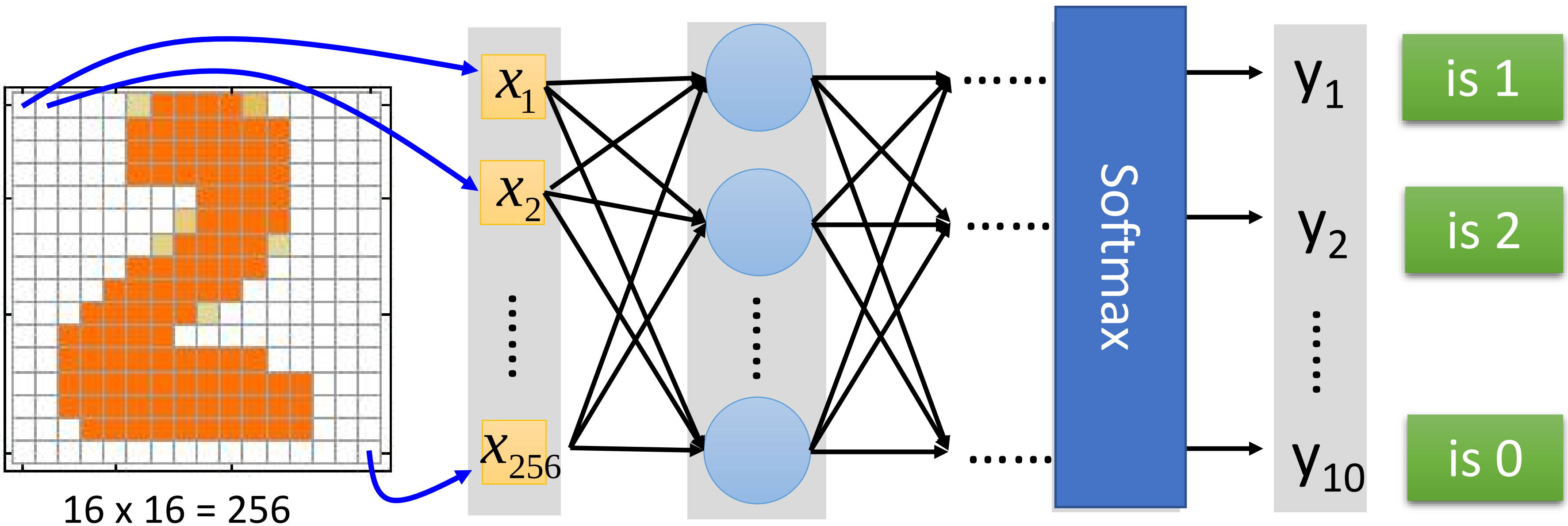
Training Data

- Preparing training data: images and their labels



The learning target is defined on the training data.

Learning Target



Ink → 1

No ink → 0

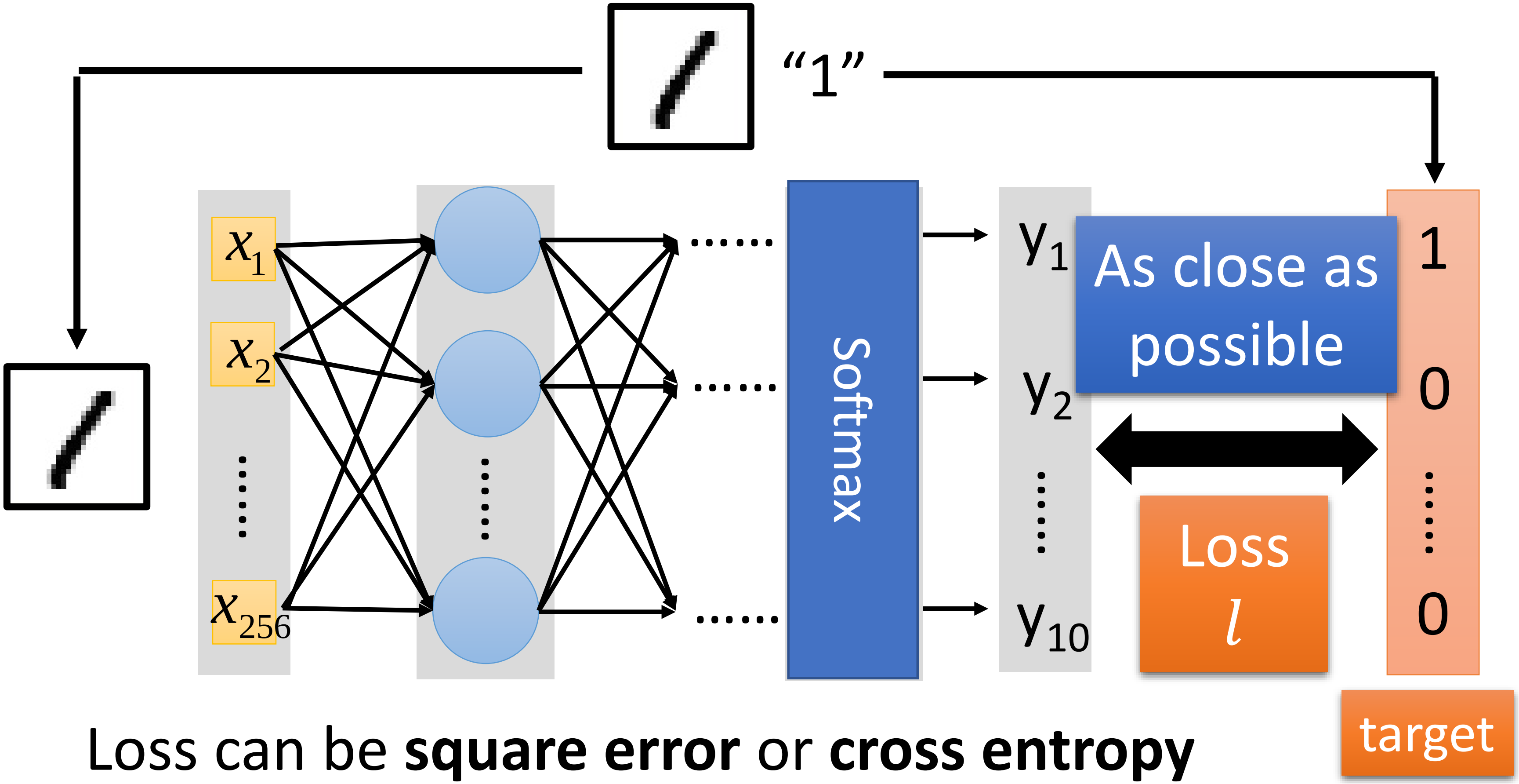
The learning target is ……

Input:  → y_1 has the maximum value

Input:  → y_2 has the maximum value

Loss

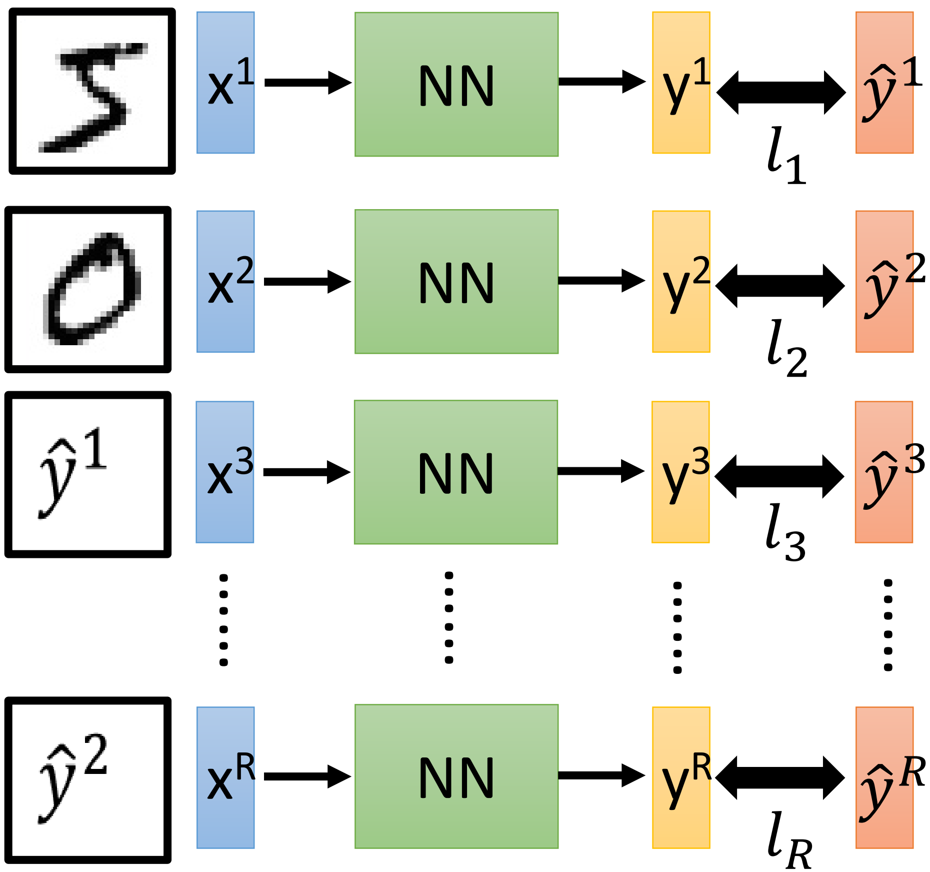
A good function should make the loss of all examples as small as possible.



Loss can be **square error** or **cross entropy** between the network output and target

Total Loss

For all training data …



Total Loss:

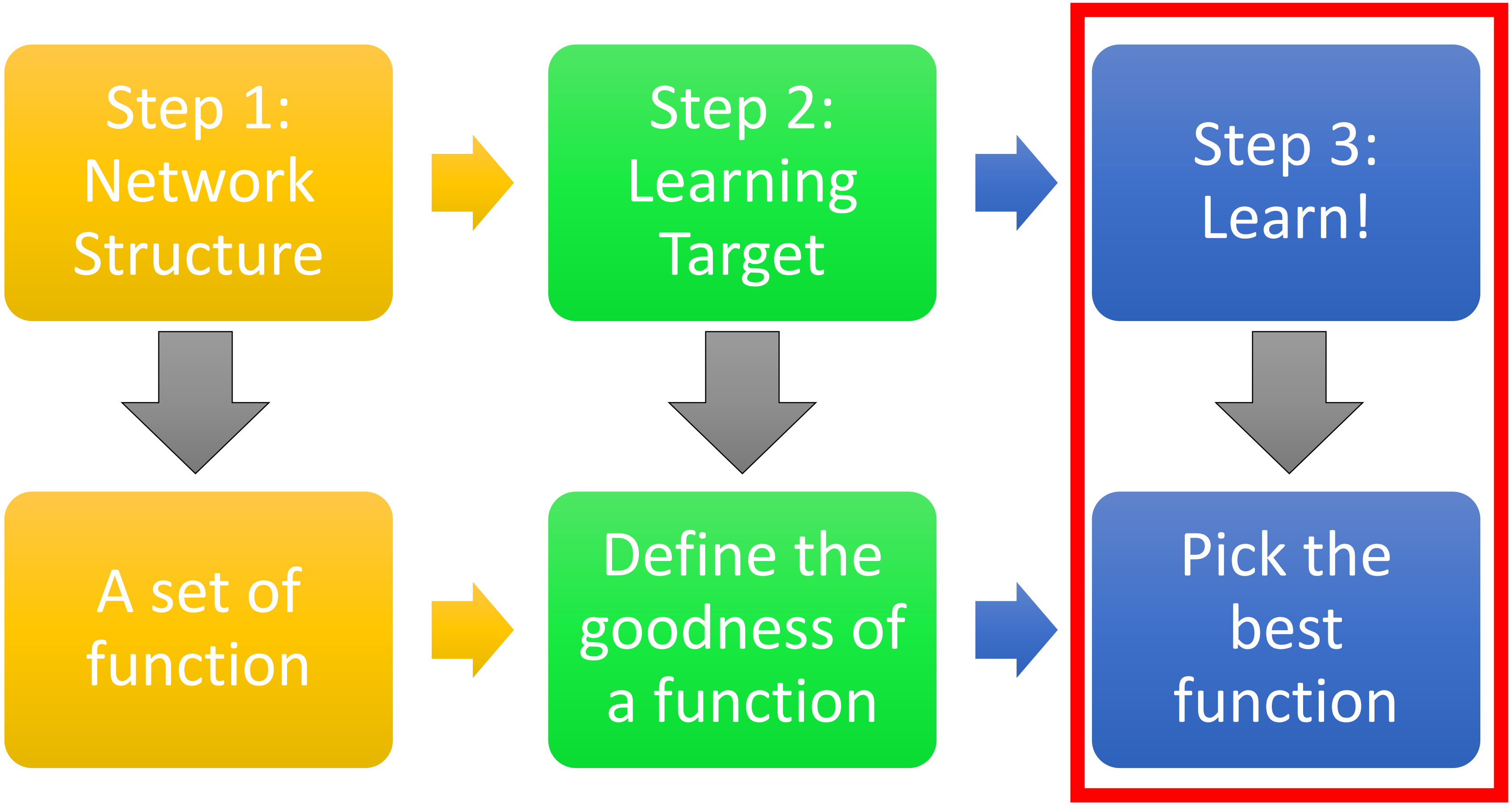
$$L = \sum_{r=1}^{R} l_r$$

As small as possible

Find ***a function in function set*** that minimize total loss L

Find ***the network parameters θ^**** that minimize total loss L

Three Steps for Deep Learning



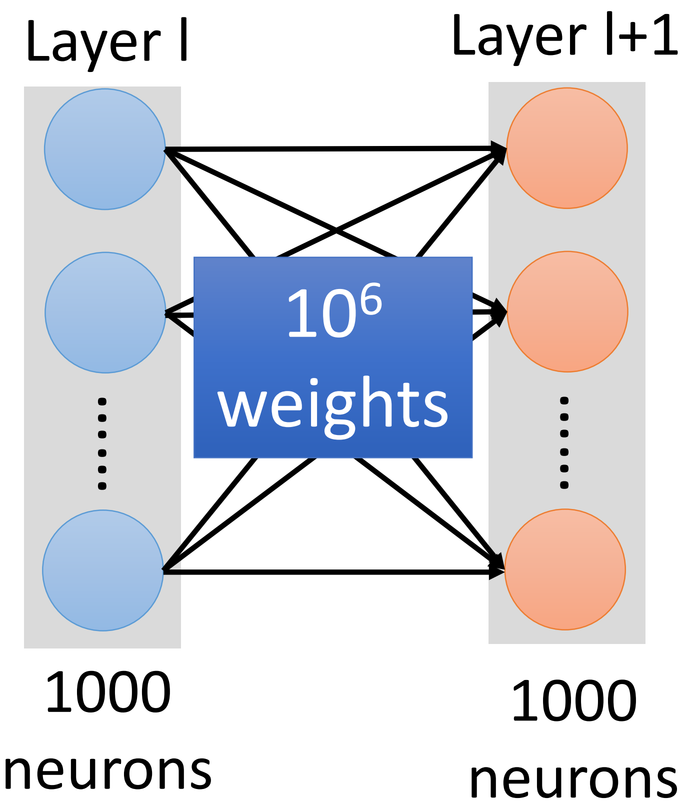
How to pick the best function

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

Enumerate all possible values

~~Network parameters~~ $\theta = \underbrace{\{w_1, w_2, w_3, \cdots, b_1, b_2, b_3, \cdots\}}_{\text{Millions of parameters}}$

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



Gradient Descent

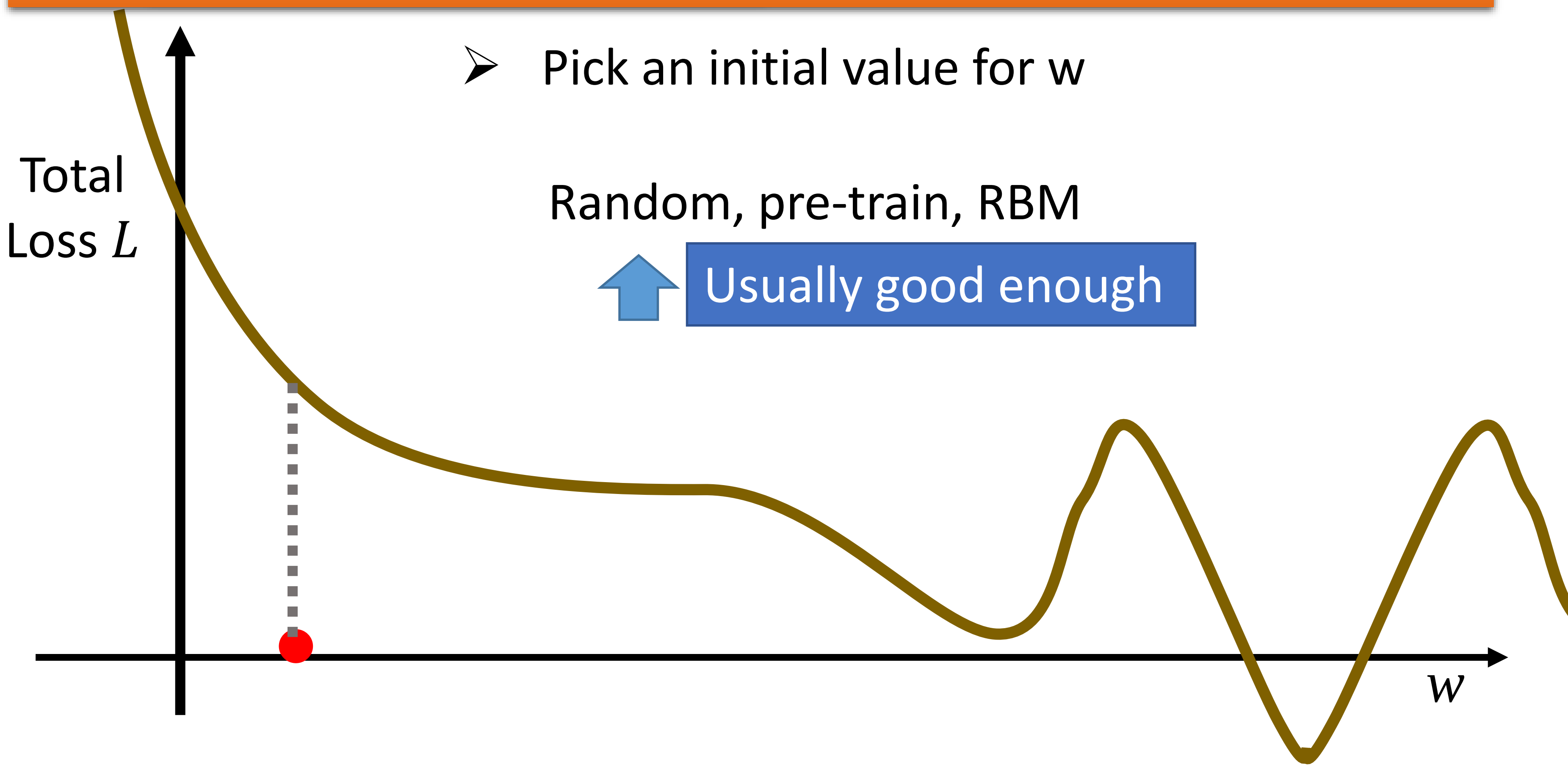
Network parameters $\theta = \{w_1, w_2, \cdots, b_1, b_2, \cdots\}$

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

- Pick an initial value for w

Random, pre-train, RBM

Usually good enough



Gradient Descent

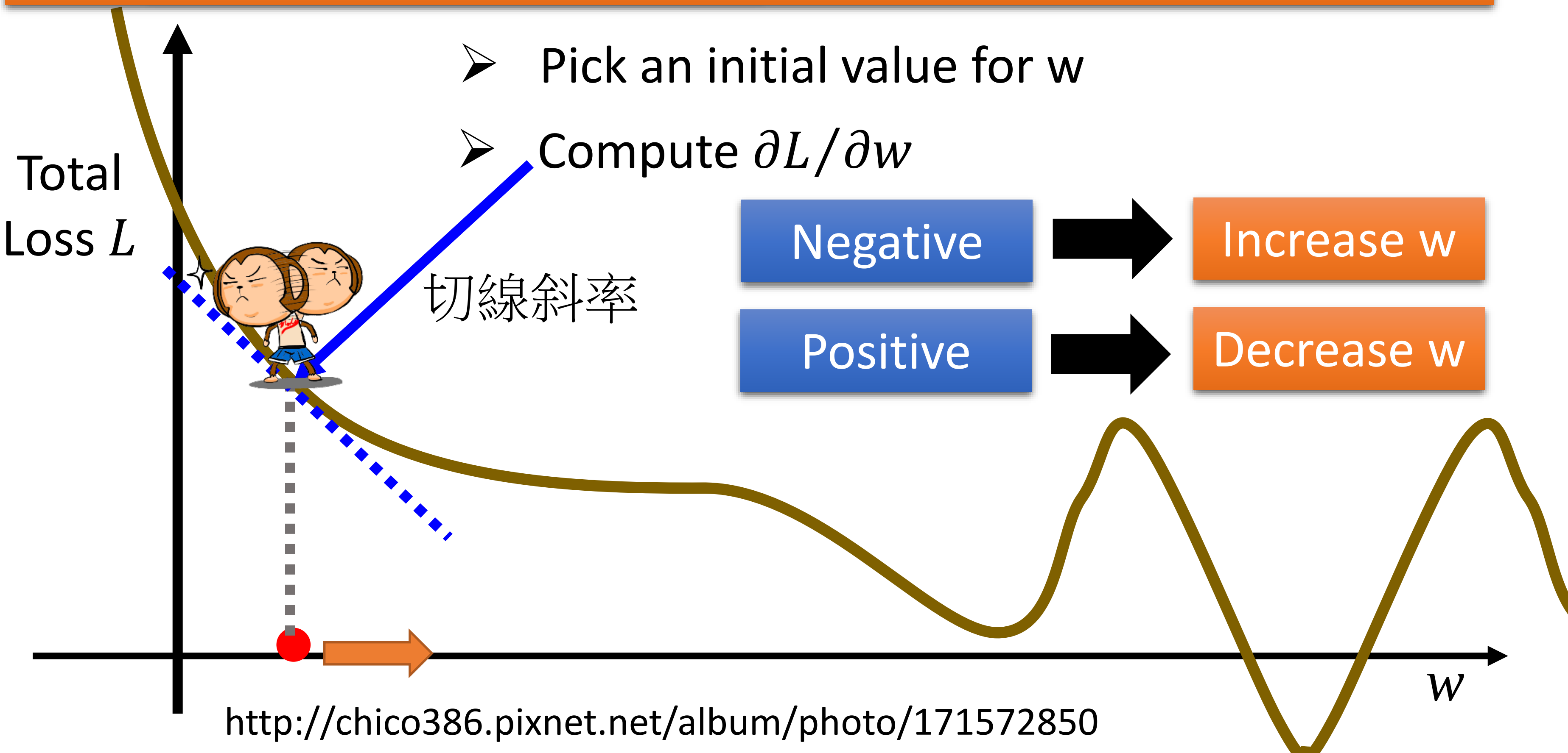
Network parameters $\theta = \{w_1, w_2, \cdots, b_1, b_2, \cdots\}$

Find ***network parameters*** θ^* that minimize total loss L

- Pick an initial value for w
- Compute $\partial L / \partial w$

Negative → Increase w

Positive → Decrease w



http://chico386.pixnet.net/album/photo/171572850

Gradient Descent

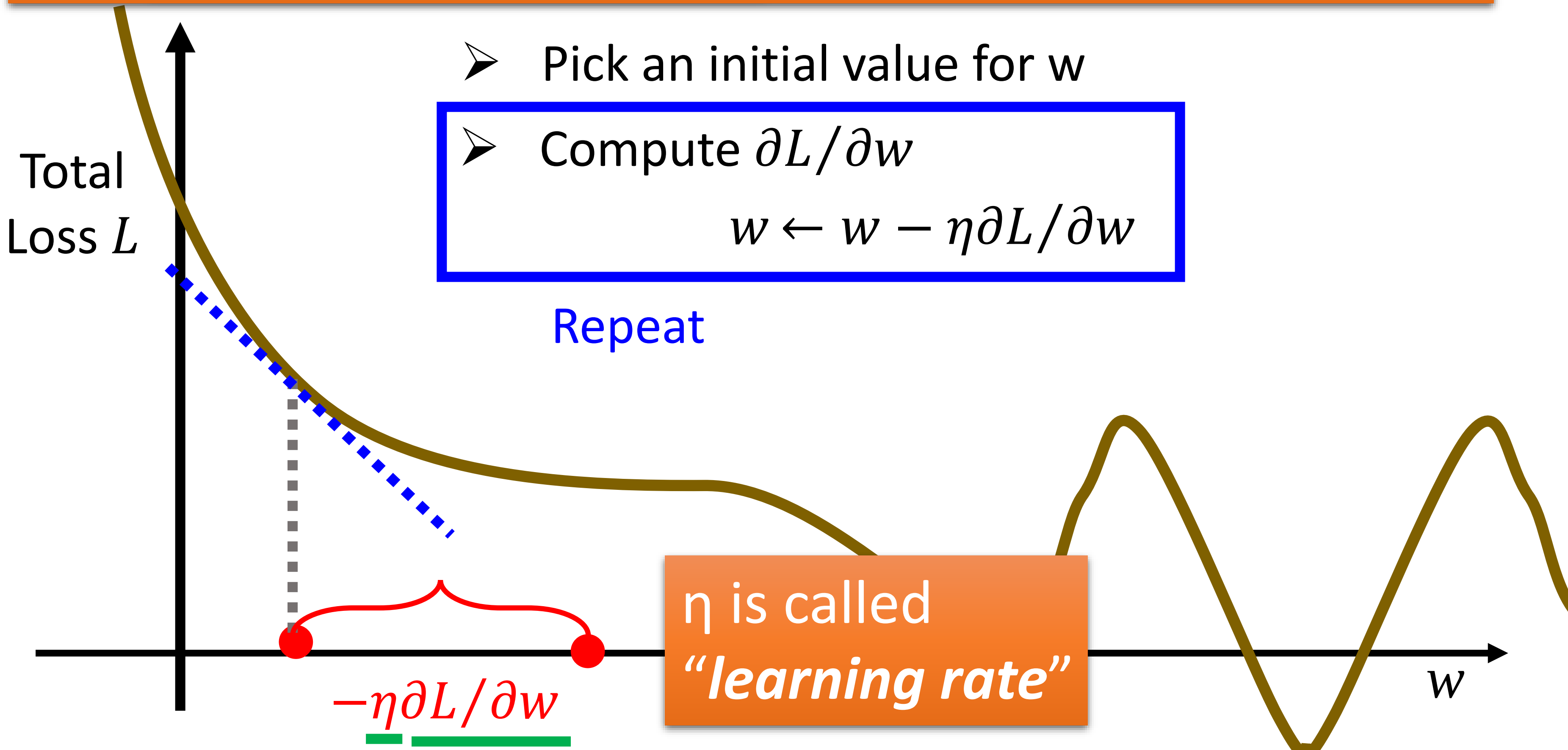
Network parameters $\theta = \{w_1, w_2, \cdots, b_1, b_2, \cdots\}$

Find ***network parameters*** θ^* that minimize total loss L

- Pick an initial value for w
- Compute $\partial L / \partial w$

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

Repeat



Gradient Descent

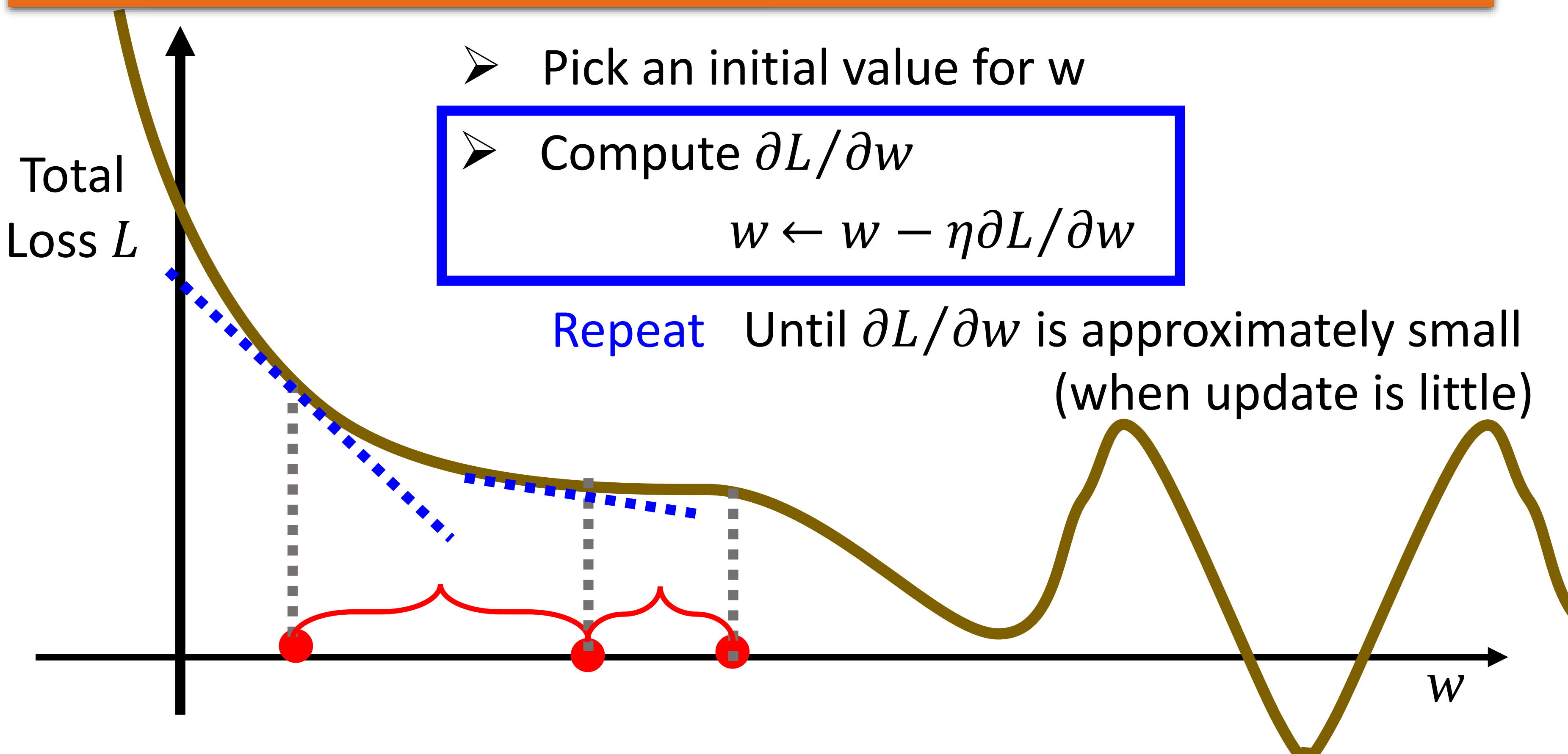
Network parameters $\theta = \{w_1, w_2, \cdots, b_1, b_2, \cdots\}$

Find ***network parameters θ^**** that minimize total loss L

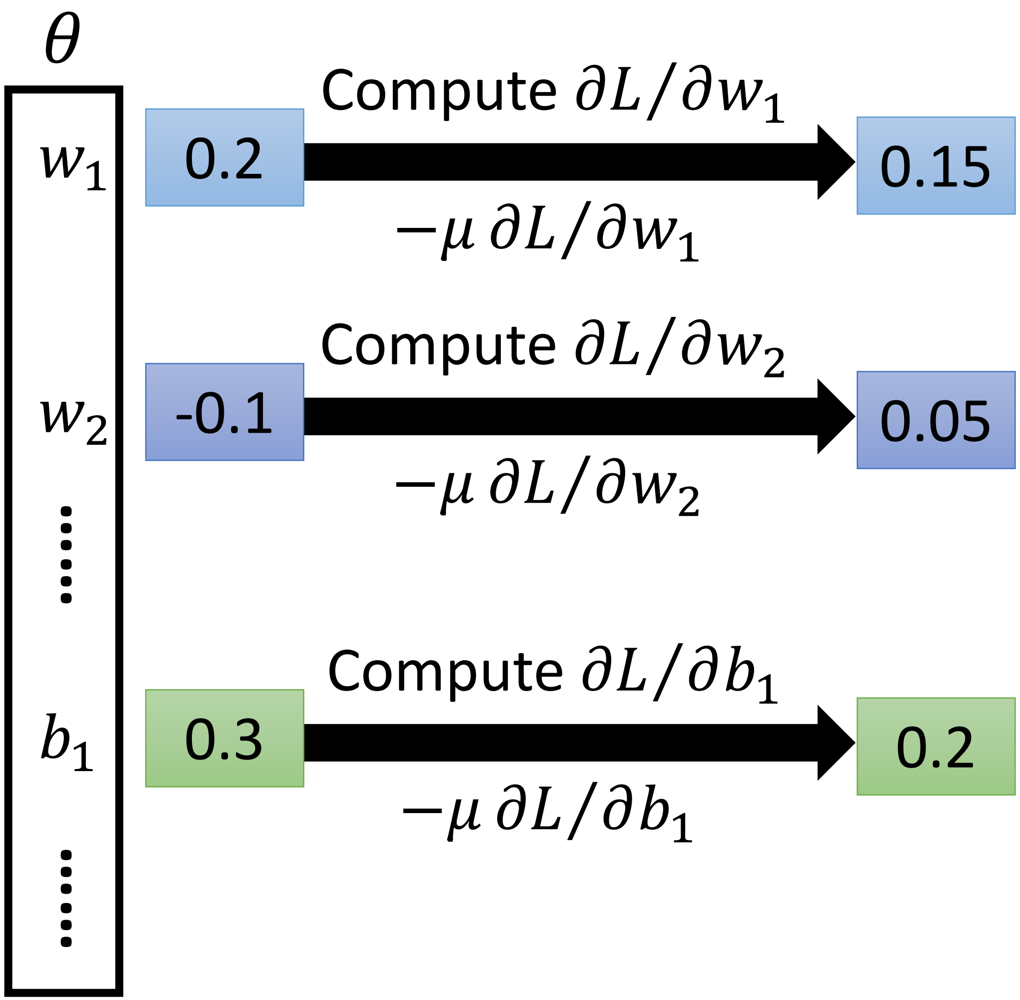
- Pick an initial value for w
- Compute $\partial L / \partial w$

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

Repeat Until $\partial L / \partial w$ is approximately small
(when update is little)



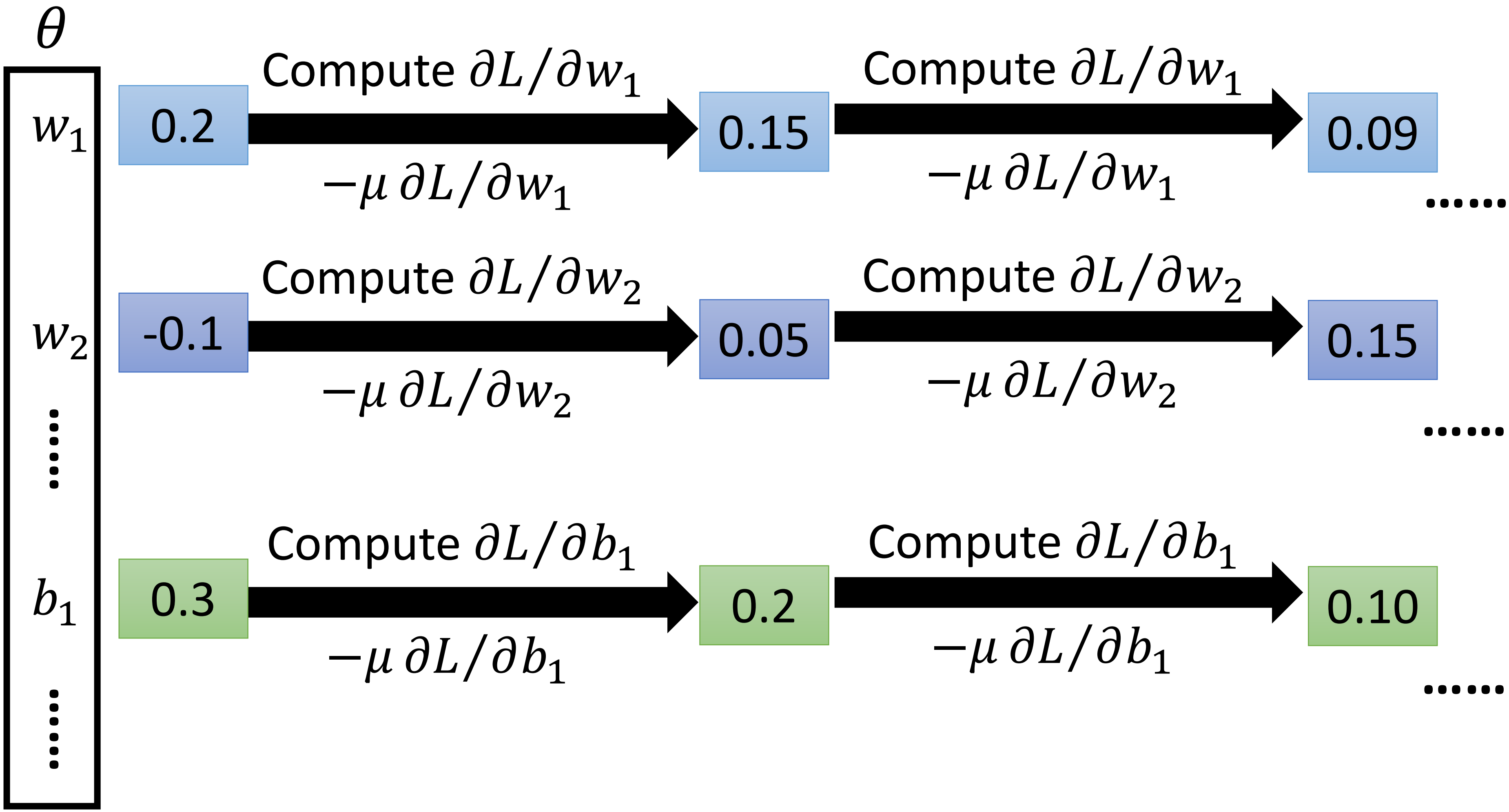
Gradient Descent



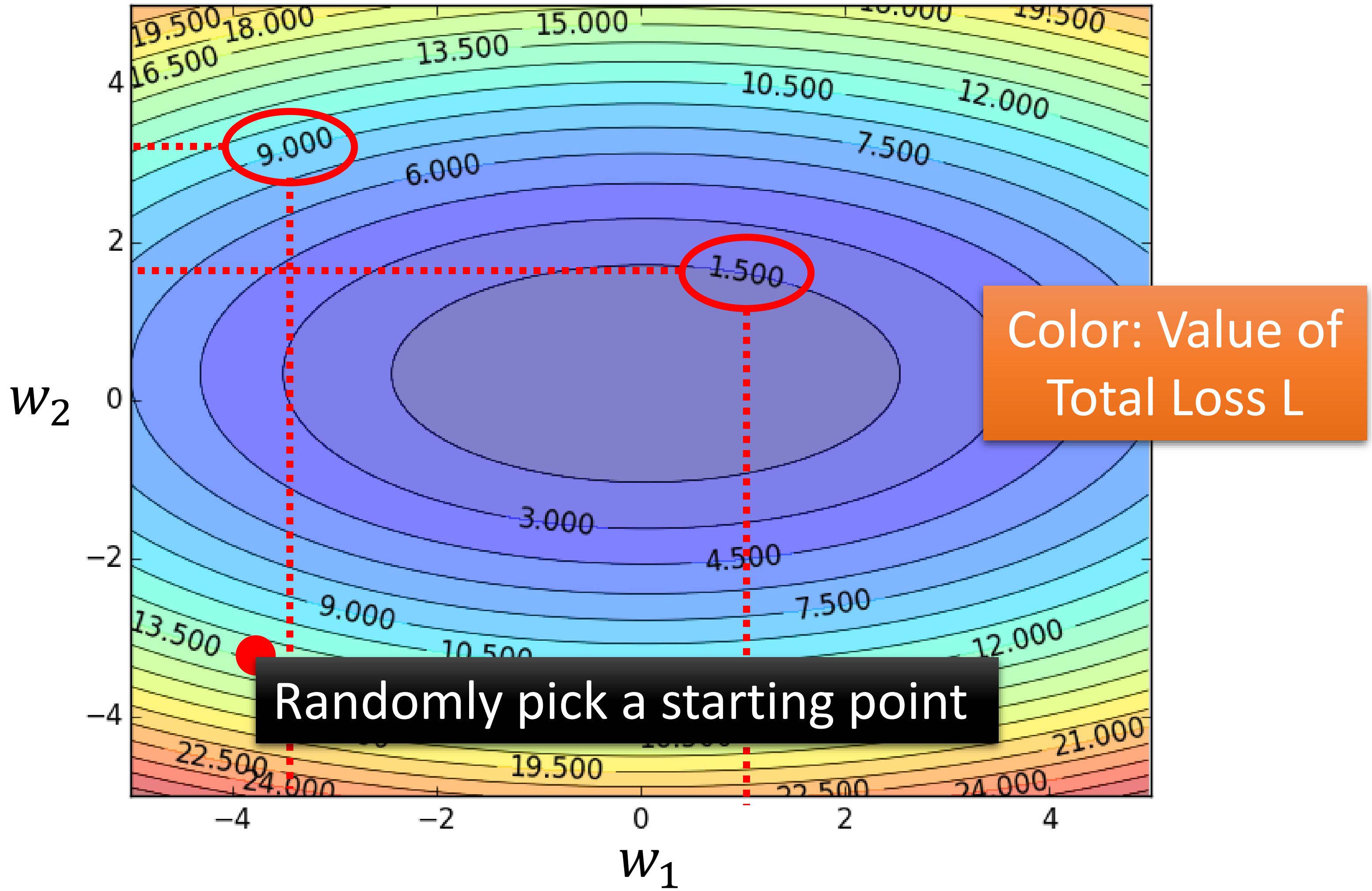
$$\nabla L = \begin{bmatrix} \dfrac{\partial L}{\partial w_1} \\ \dfrac{\partial L}{\partial w_2} \\ \vdots \\ \dfrac{\partial L}{\partial b_1} \\ \vdots \end{bmatrix}$$

gradient

Gradient Descent

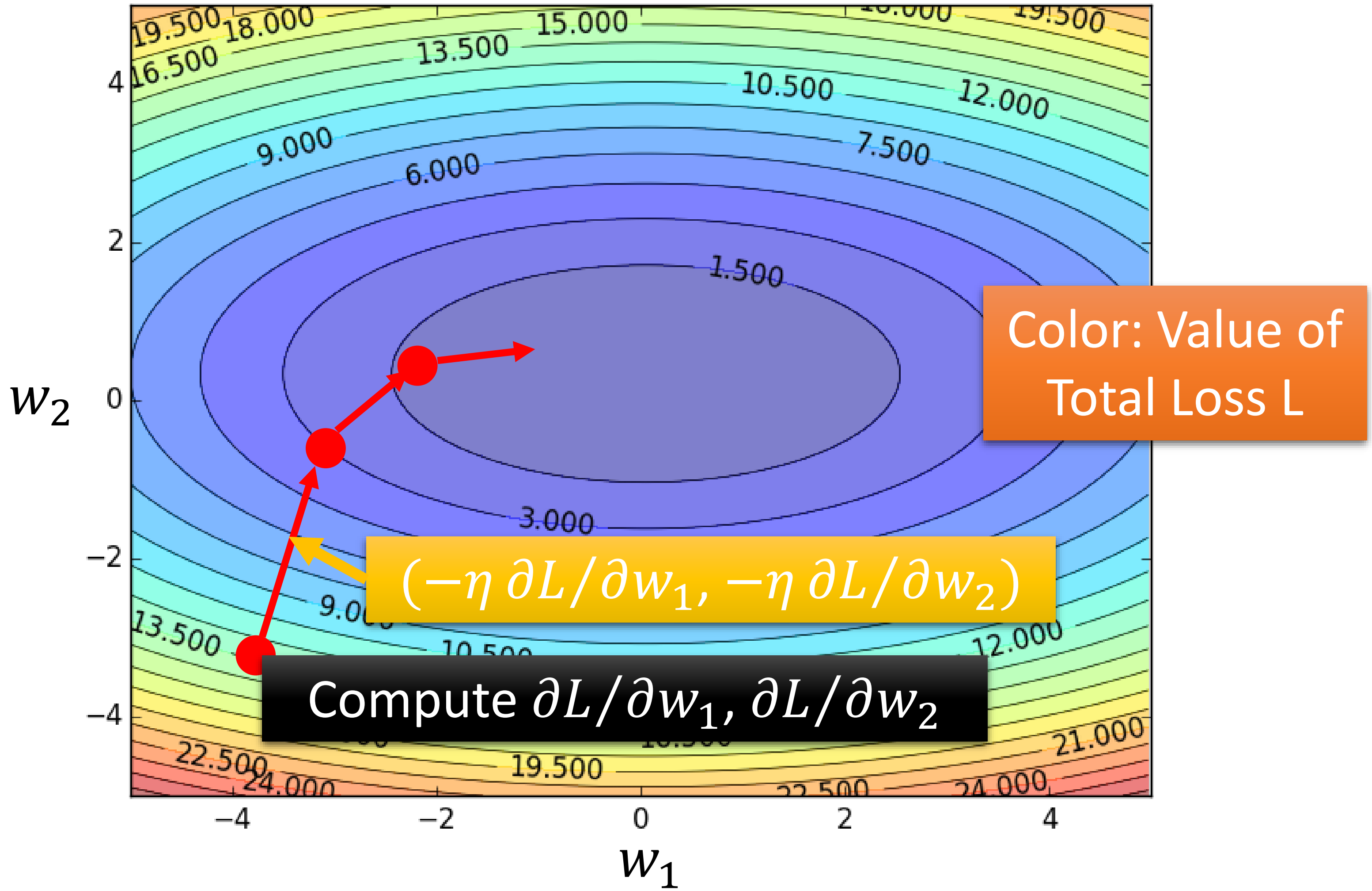


Gradient Descent



Gradient Descent

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



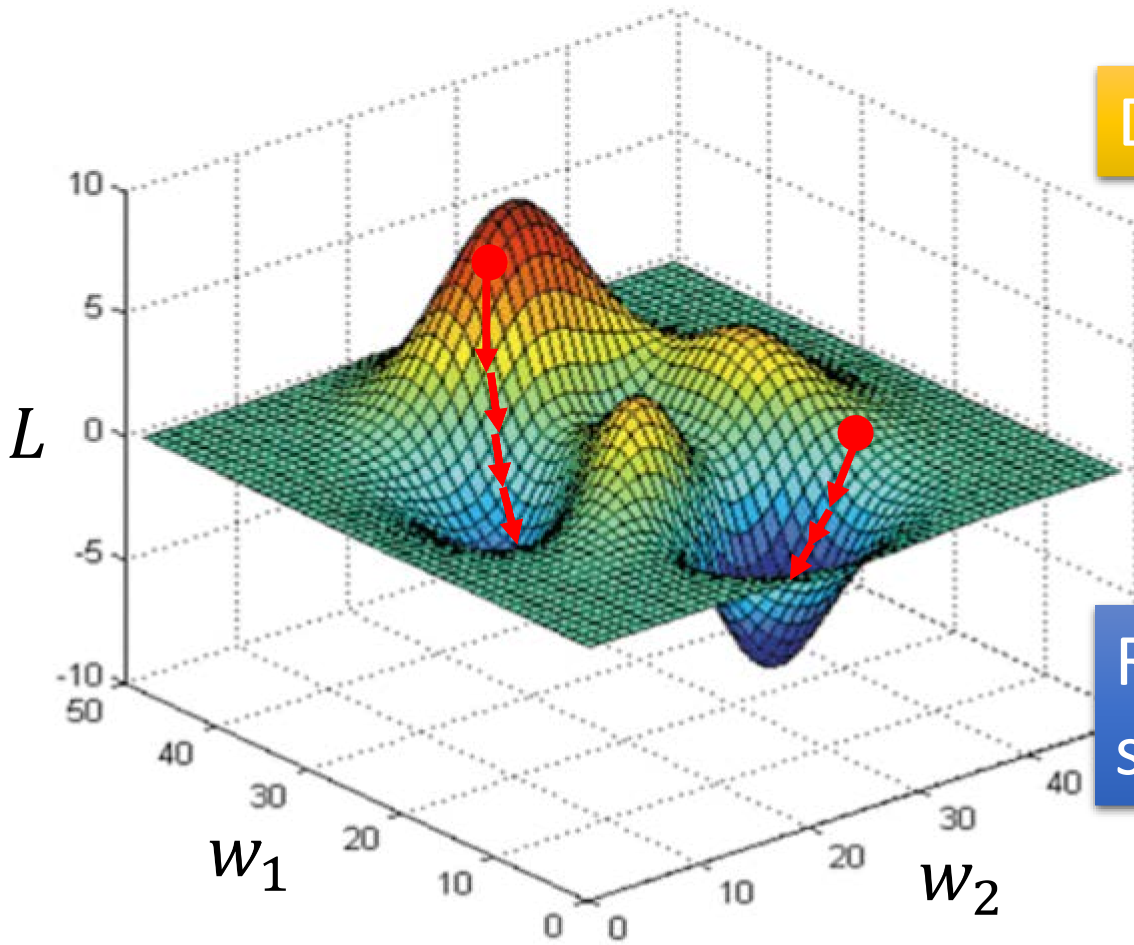
Gradient Descent

- When considering multiple parameters together, do you see any problem?

Local Minima

Who is Afraid of Non-Convex Loss Functions?
http://videolectures.net/eml07_lecun_wia/

- Gradient descent never guarantee global minima

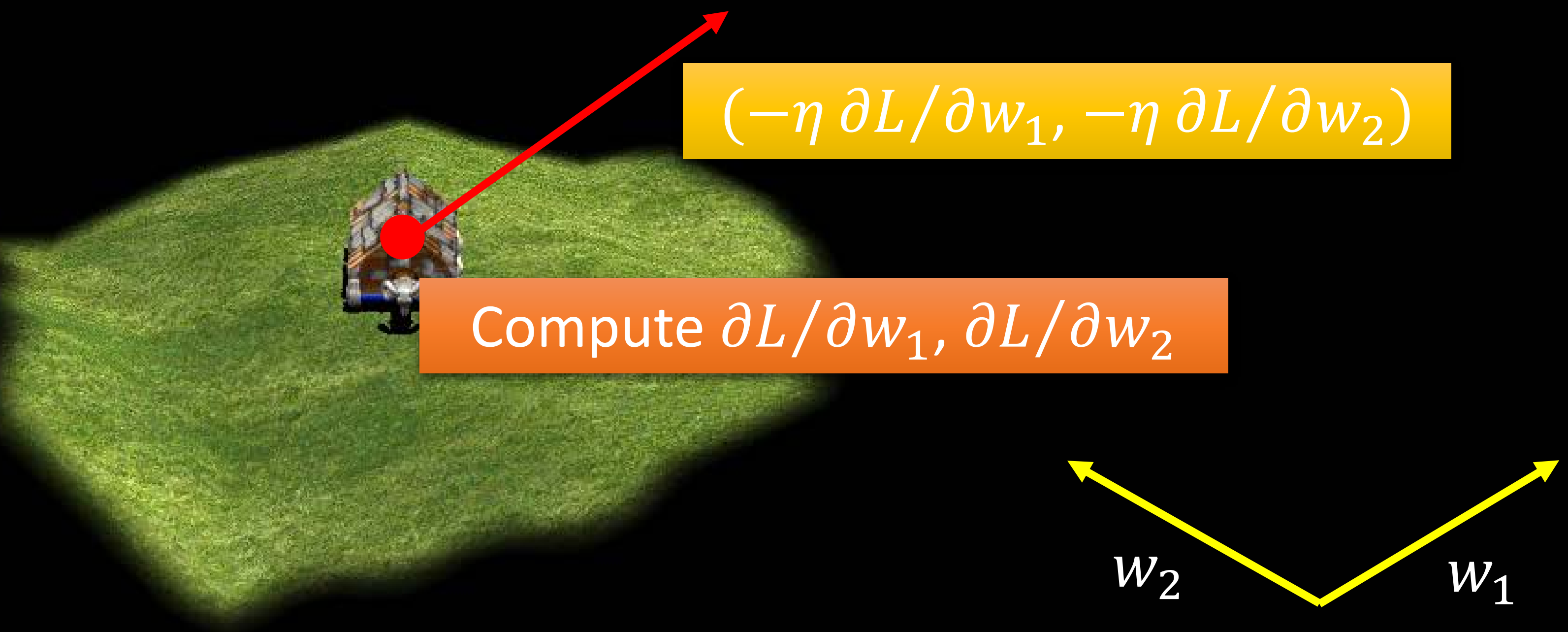


Different initial point

Reach different minima, so different results

想像你在玩世紀帝國……

沒有探索過的地方被戰霧覆蓋

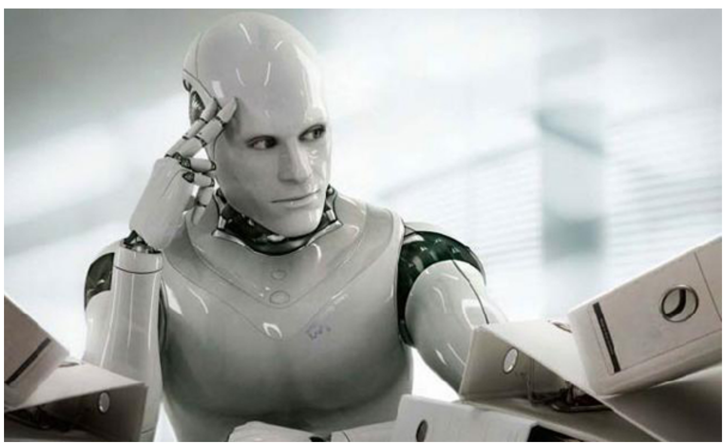


Gradient Descent

This is the "learning" of machines in deep learning ……

➡ Even alpha go using this approach.

大家以為 Learning 是 ……



其實 Learning 只是 ……

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

TensorFlow

torch

theano

libdnn
台大周伯威
同學開發

Caffe

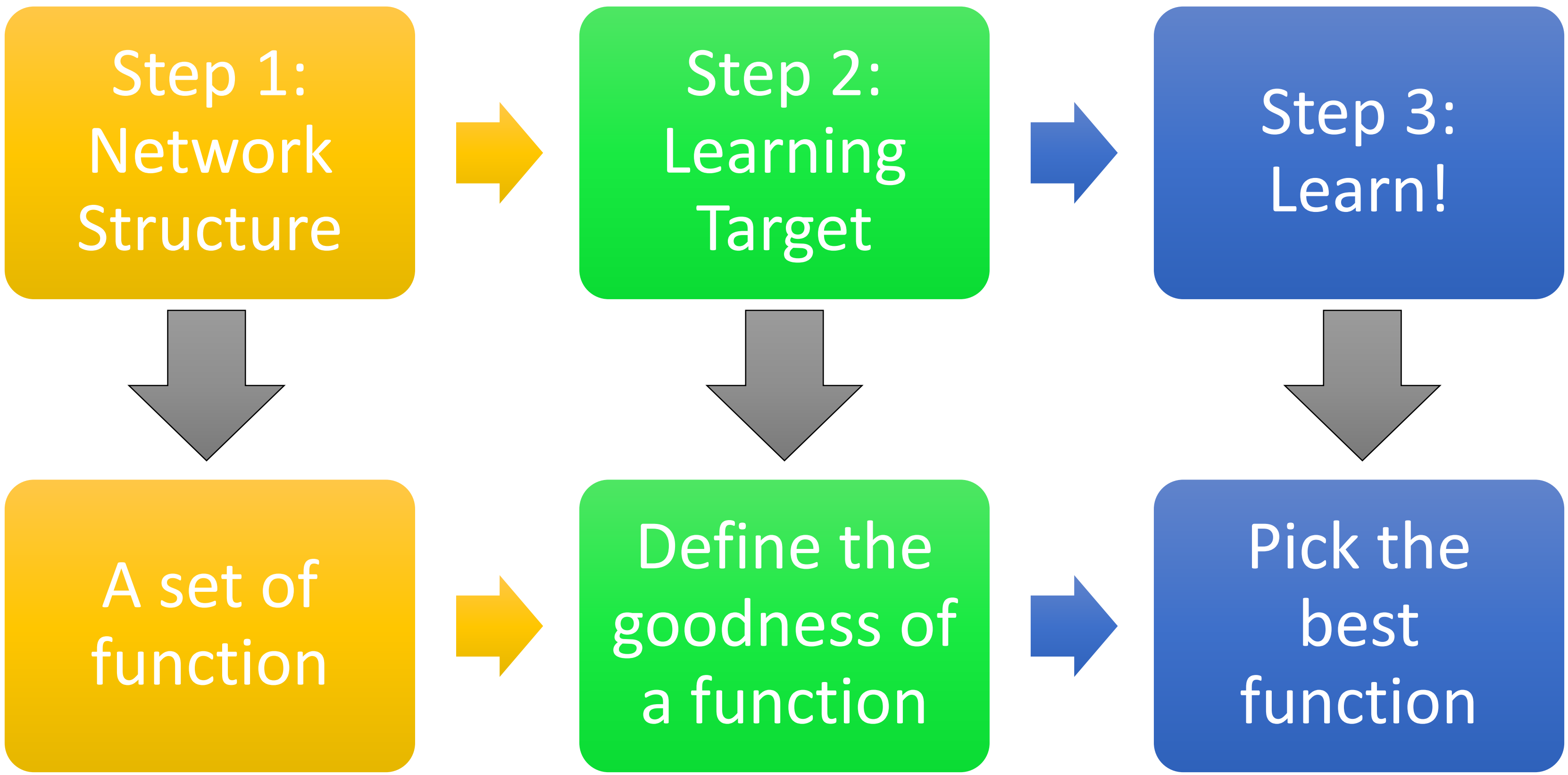
Microsoft CNTK

Chainer

mxnet

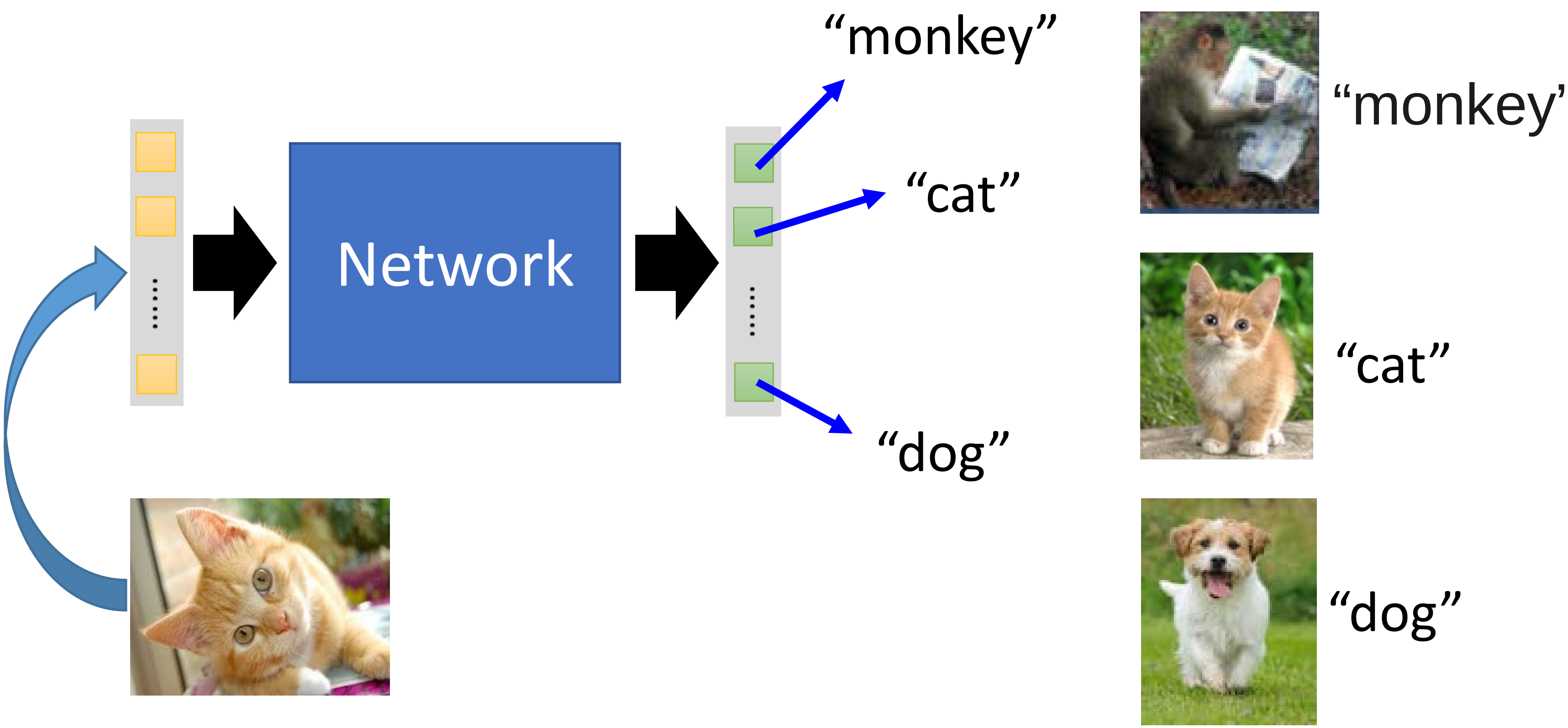
Don’t worry about $\partial L / \partial w$, the toolkits will handle it.

You can do lots of different things



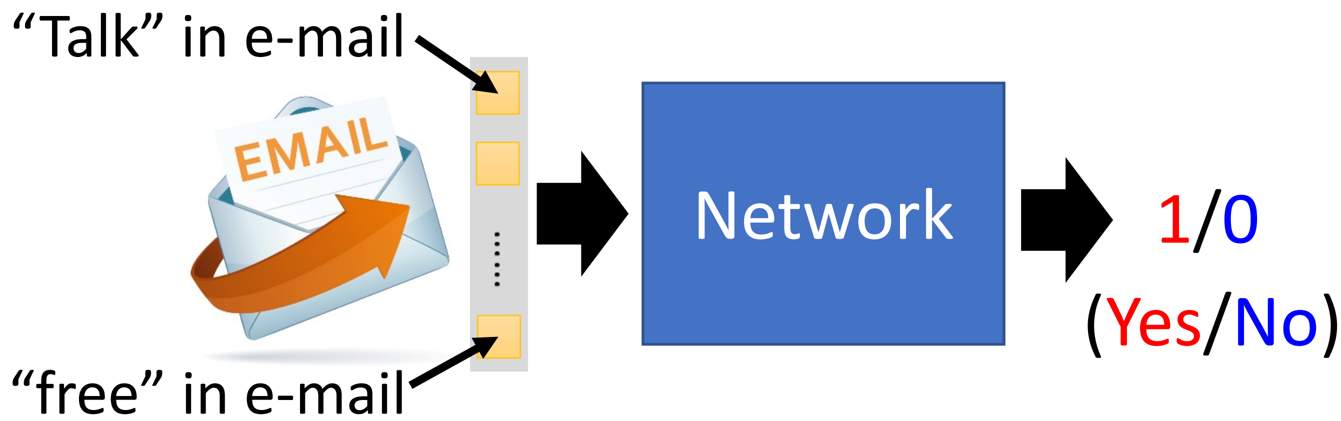
For example, you can do …….

- Image Recognition



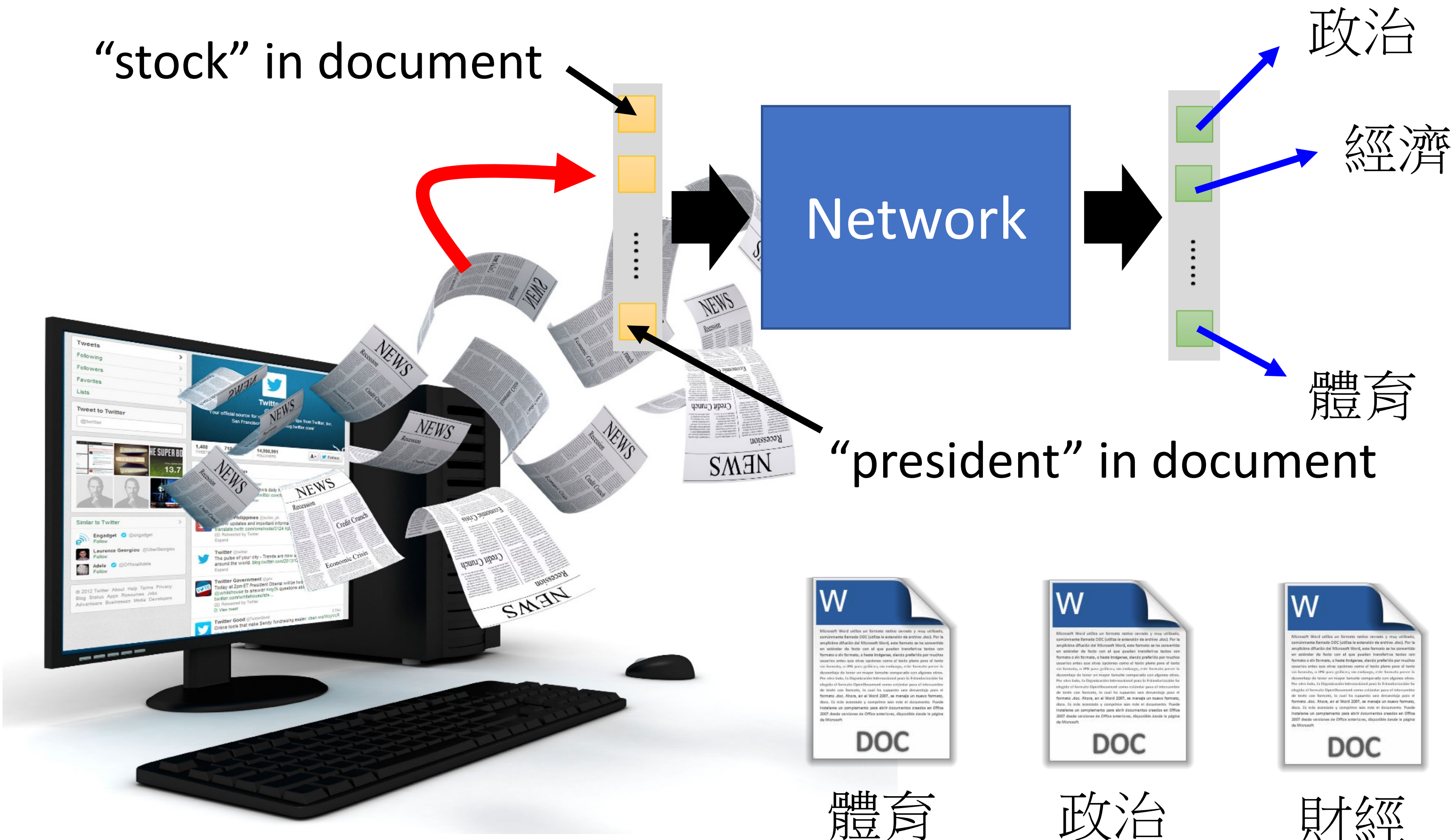
For example, you can do ....

Spam filtering



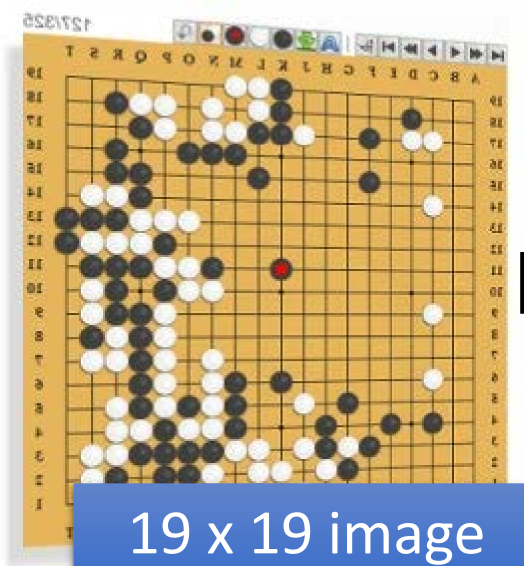
(http://spam-filter-review.toptenreviews.com/)

Document Classification



http://top-breaking-news.com/

Playing Go



19 x 19 image (matrix)

Black: 1

white: -1

none: 0

Network

Next move (19 x 19 positions)

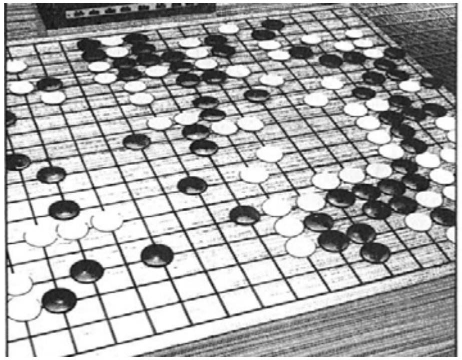
19 x 19 vector

Fully-connected feedword network can be used

But CNN performs much better.

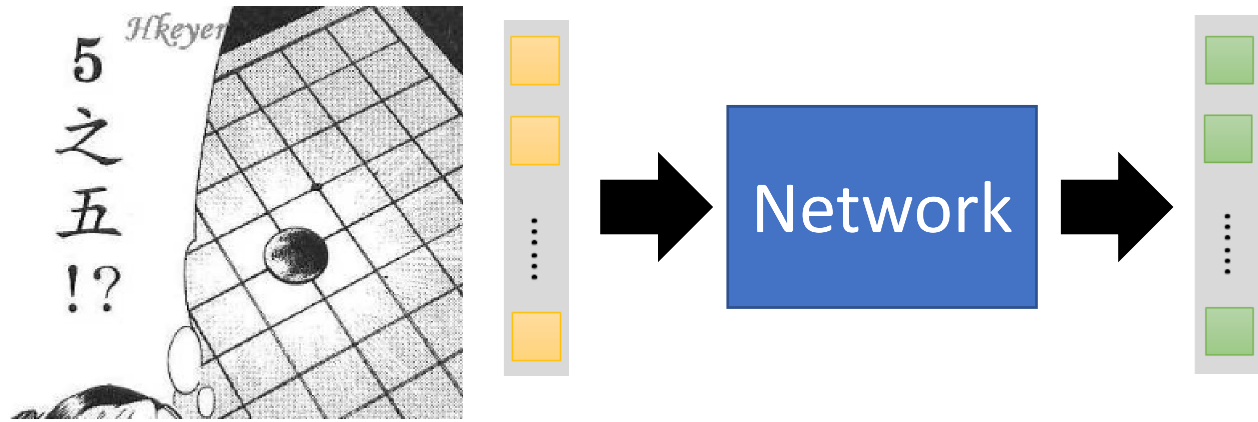
(Lecture III)

Playing Go

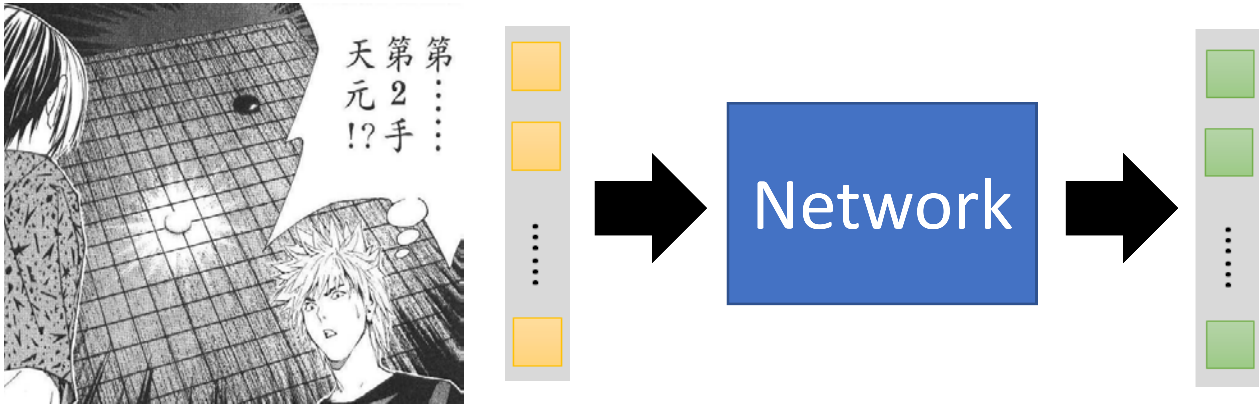


蒐集一堆棋譜

Training:



Target:
天元 = 1
其他都是 0



Target:
五之 5 = 1
其他都是 0

Concluding Remarks

- Deep Learning is simple & powerful!

但 Deep Learning 就像 雷神之槌

無法輕易被舉起來 ……

http://ent.ltn.com.tw/news/breakingnews/1144545

Lecture II:
Tips for Training DNN

Outline of Lecture II

"Hello World" for Deep Learning

Recipe of 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/RNN%20training%20(v6).ecm.mp4/index.html

TensorFlow or theano

Very flexible

Need some effort to learn

Interface of TensorFlow or Theano

keras

Easy to learn and use

(still have some flexibility)

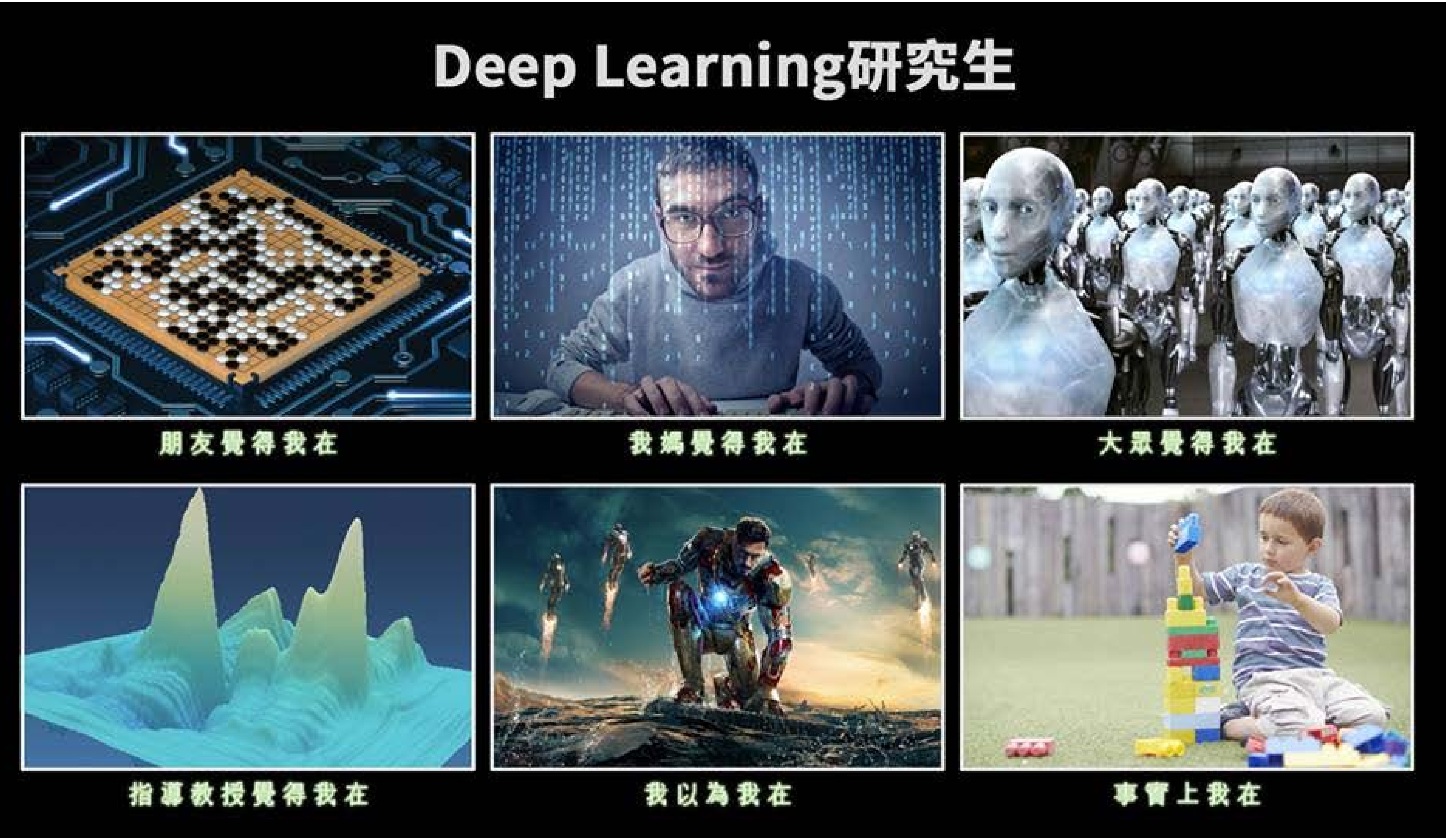
You can modify it if you can write TensorFlow or Theano

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

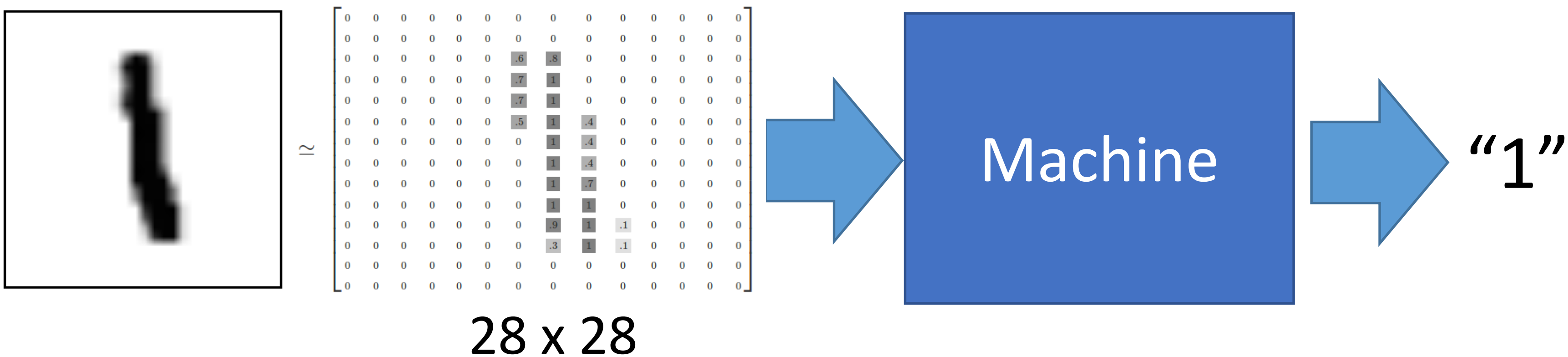
感謝 沈昇勳 同學提供圖檔

使用 Keras 心得



Example Application

- Handwriting Digit Recognition



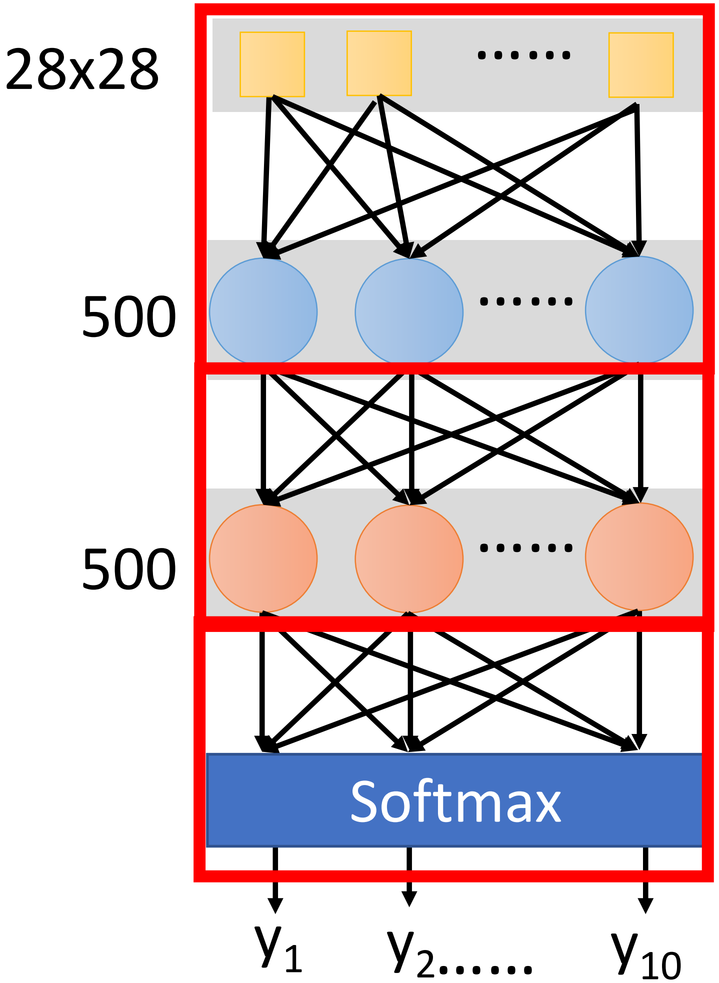
MNIST Data: http://yann.lecun.com/exdb/mnist/

“Hello world” for deep learning

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

Keras

Step 1: Network Structure → Step 2: Learning Target → Step 3: Learn!



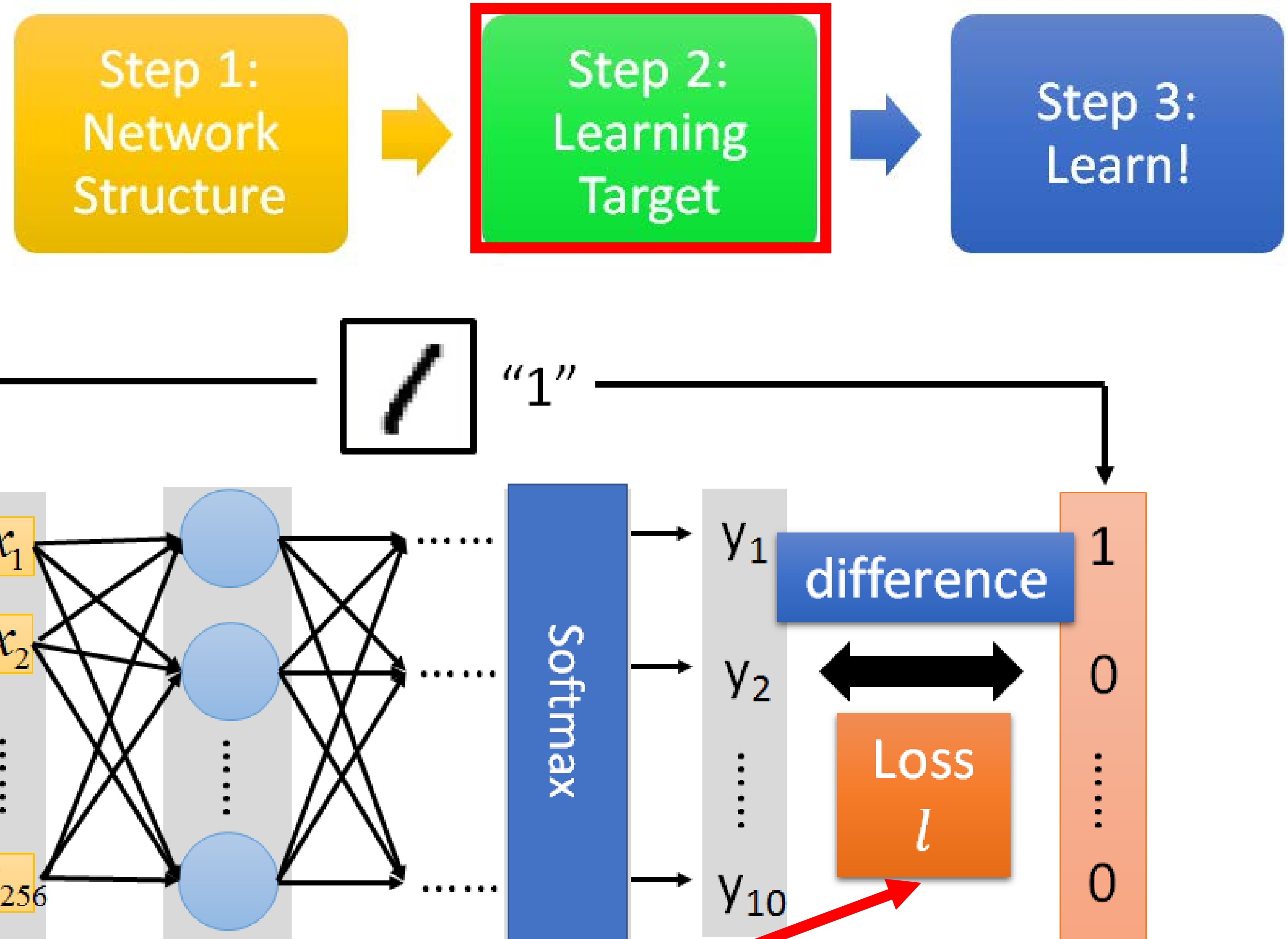

```
model = Sequential()
```

```
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




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

Keras

Step 1: Network Structure → Step 2: Learning Target → Step 3: Learn!

Step 3.1: Configuration

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

$$w \leftarrow w - \underset{0.1}{\eta}\,\partial L / \partial w$$

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)

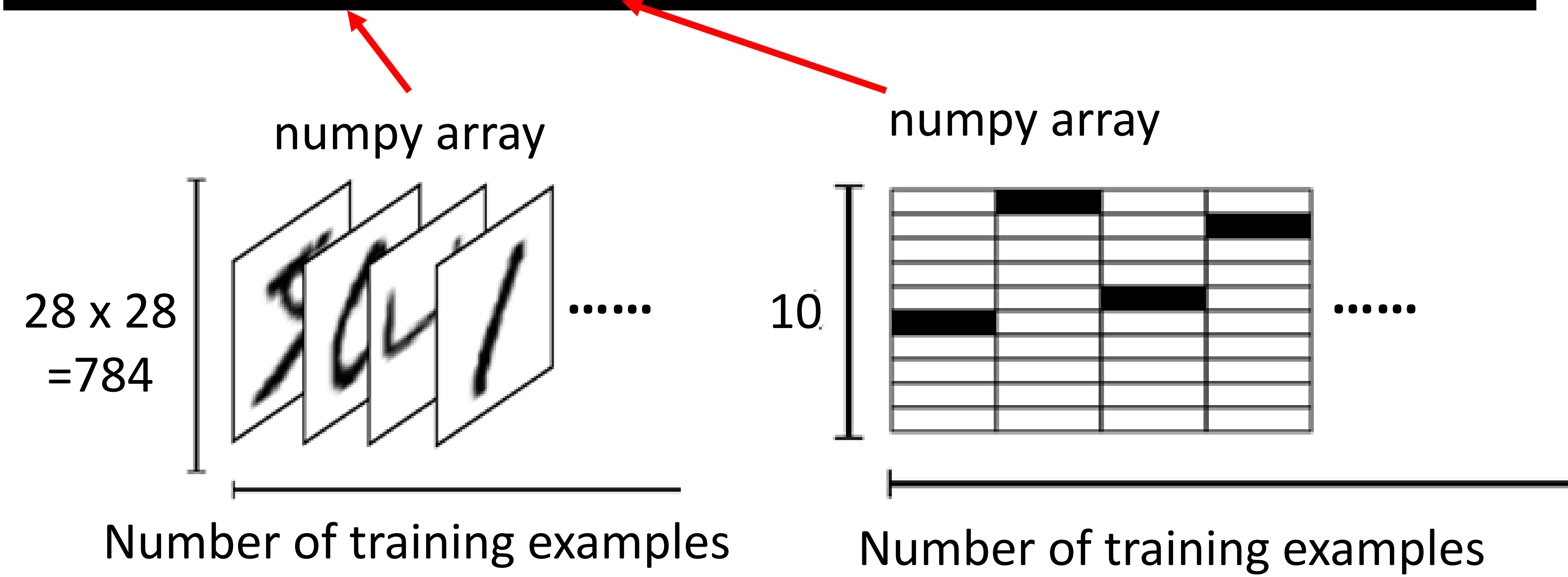
等一下再講

Keras

Step 1: Network Structure → Step 2: Learning Target → **Step 3: Learn!**

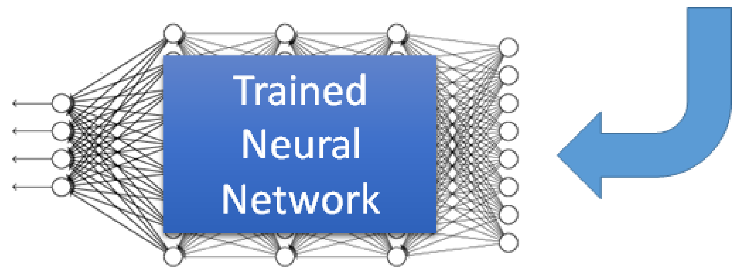
Step 3.2: Find the optimal network parameters

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



https://www.tensorflow.org/versions/r0.8/tutorials/mnist/beginners/index.html

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):

case 1:

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

case 2:

```
result = model.predict(x_test)
```

Keras

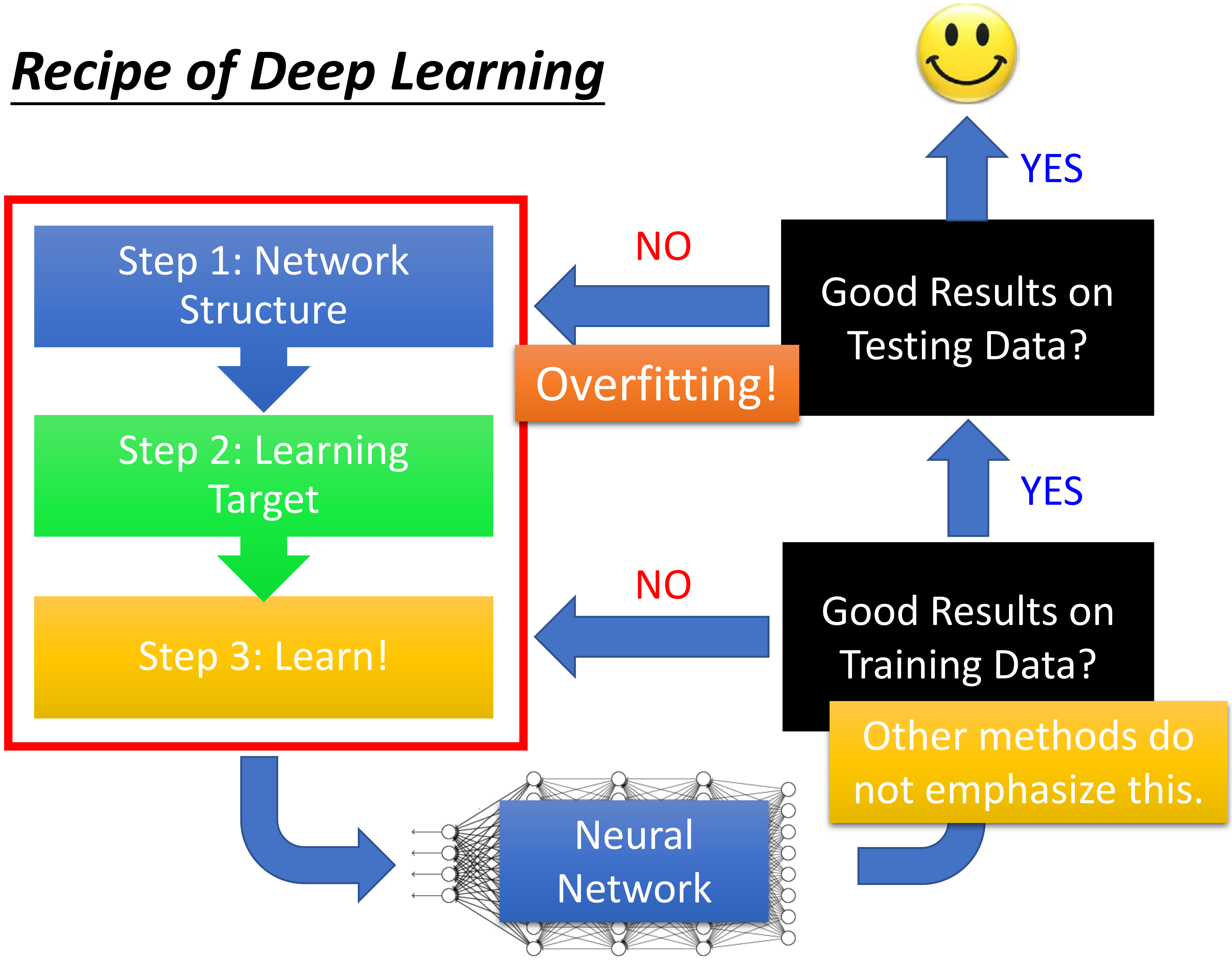
- Using GPU to speed training
  - Way 1
    - THEANO_FLAGS=device=gpu0 python YourCode.py
  - Way 2 (in your code)
    - import os
    - os.environ["THEANO_FLAGS"] = "device=cpu"

Outline of Lecture II

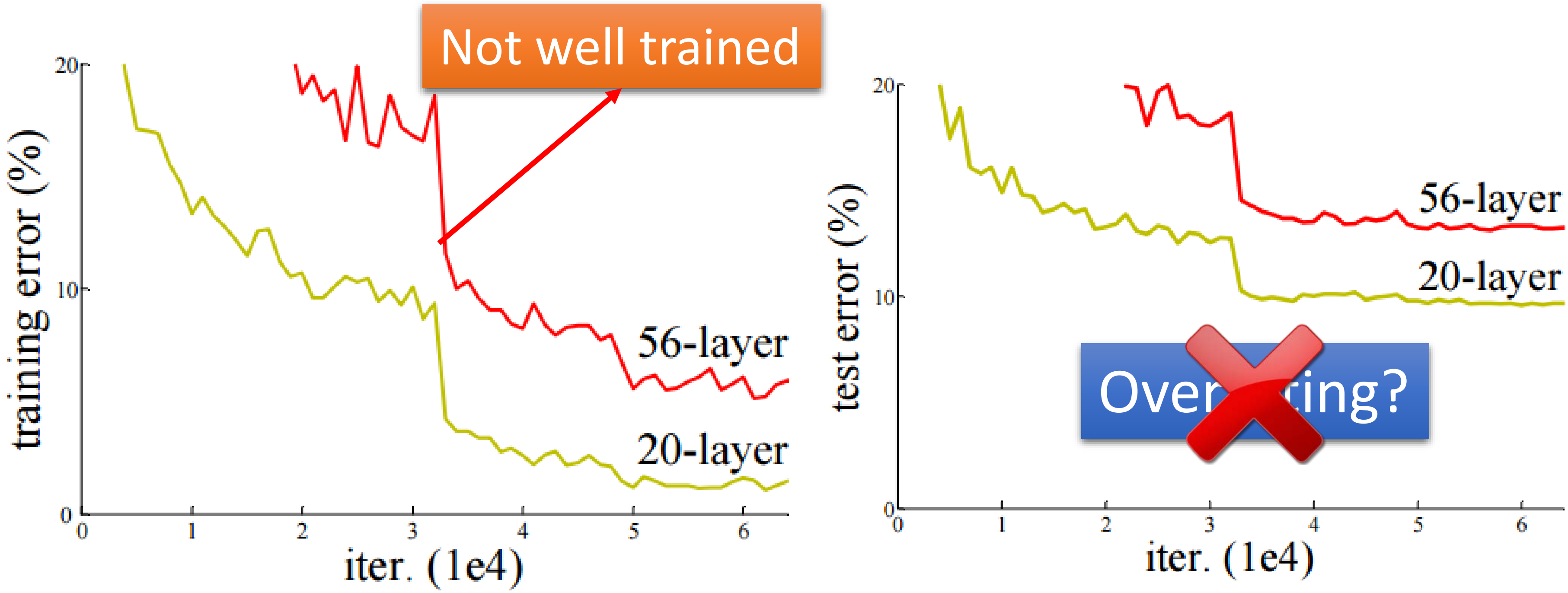
"Hello World" for Deep Learning

Recipe of Deep Learning

Recipe of Deep Learning



Do not always blame Overfitting



Training Data

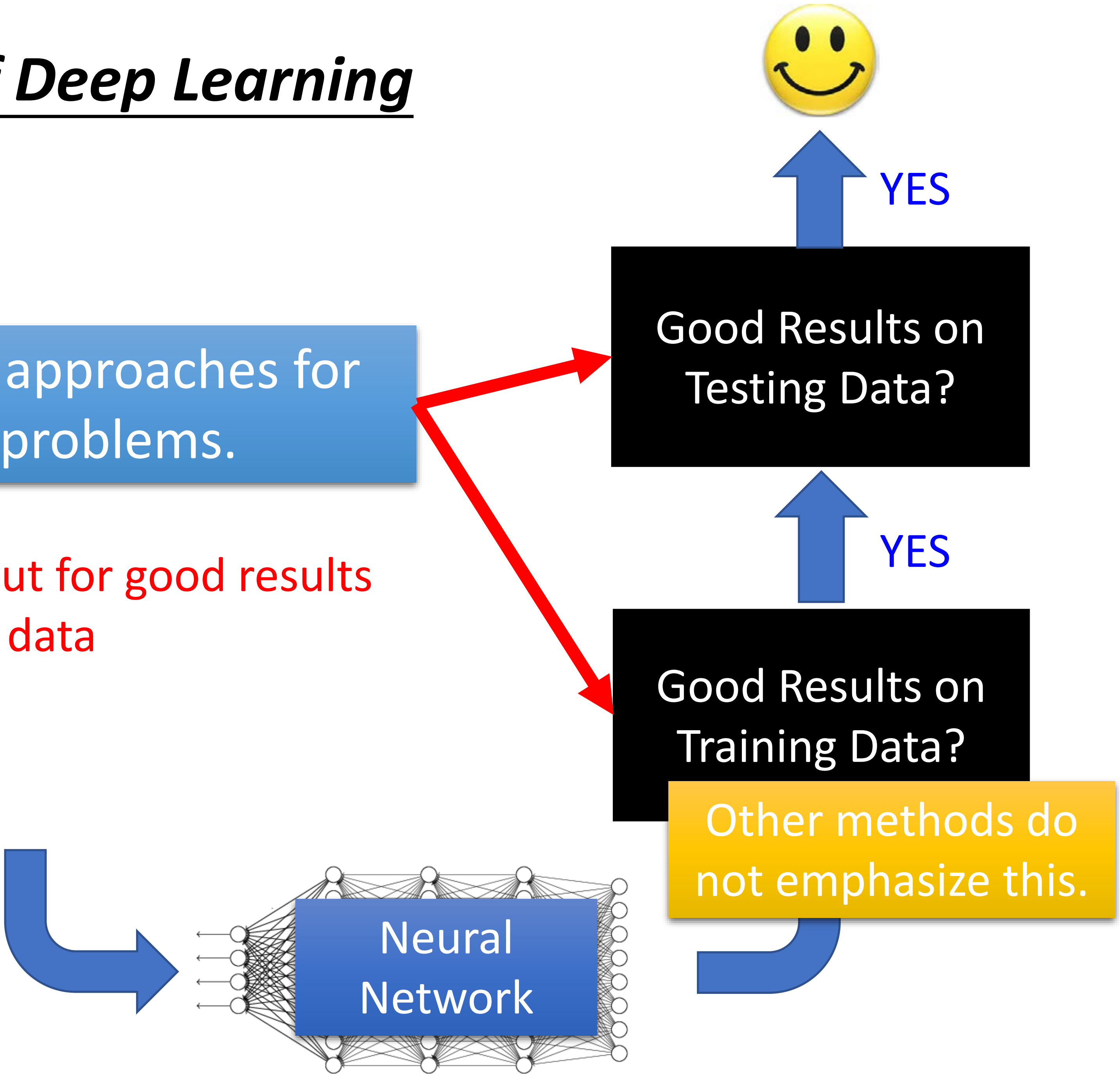
Testing Data

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

Recipe of Deep Learning

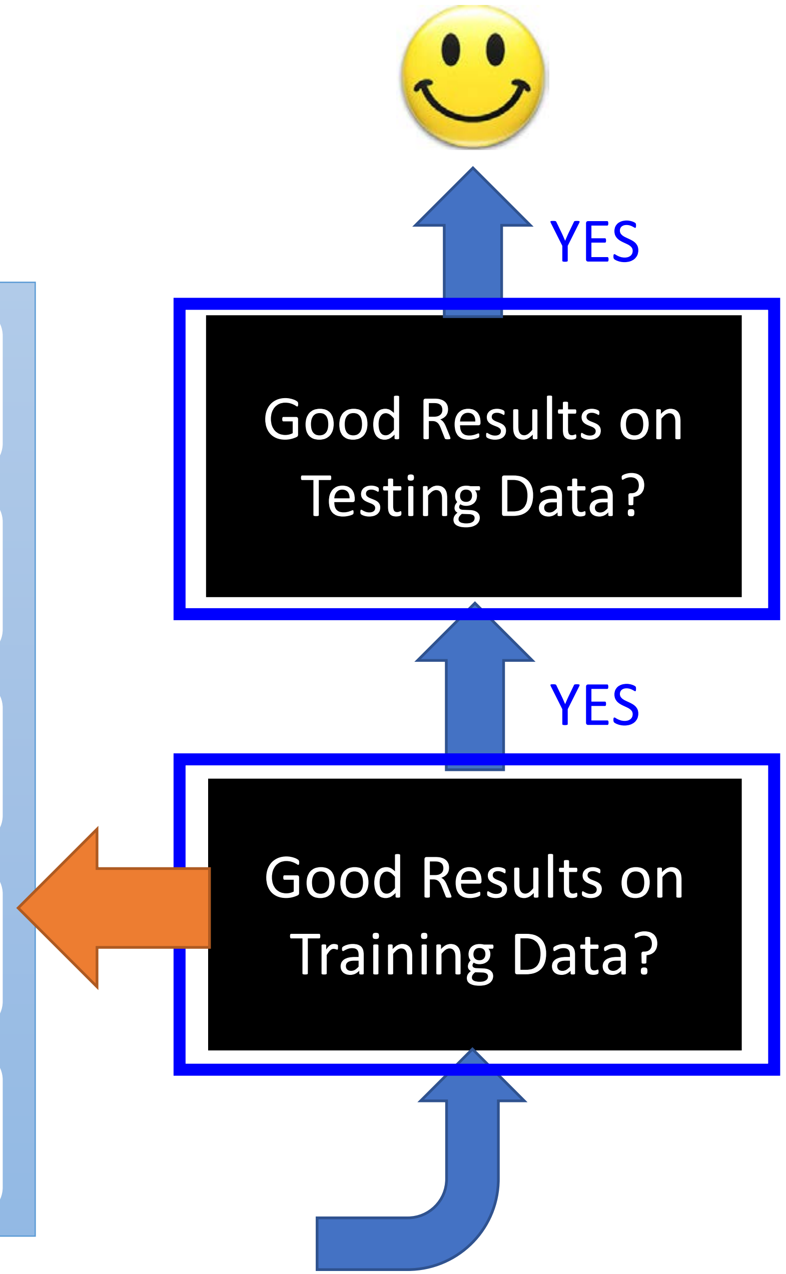
Different approaches for different problems.

e.g. dropout for good results on testing data

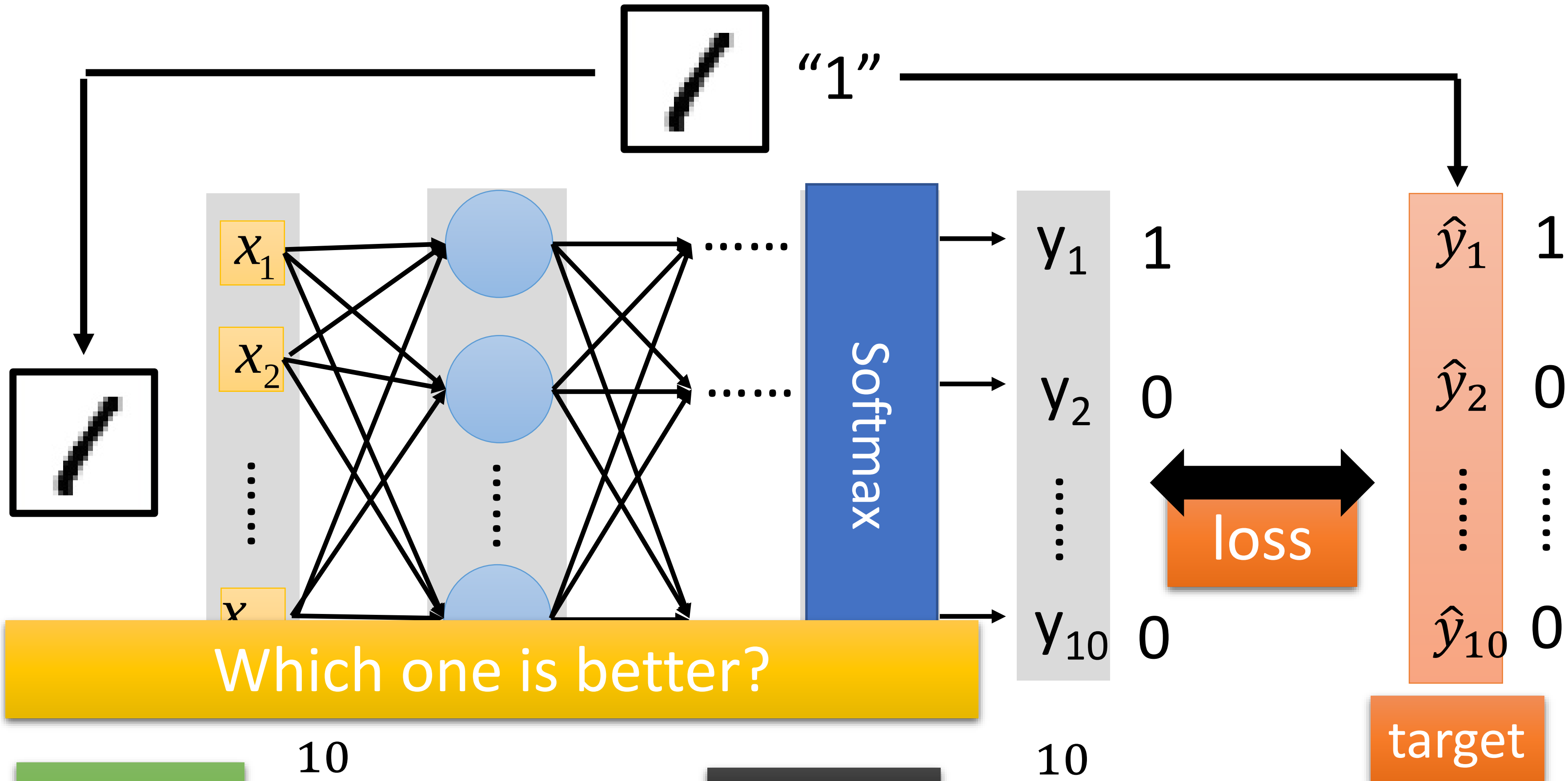


Recipe of Deep Learning

- Choosing proper loss
- Mini-batch
- New activation function
- Adaptive Learning Rate
- Momentum



Choosing Proper Loss



Which one is better?

Square Error

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

=0

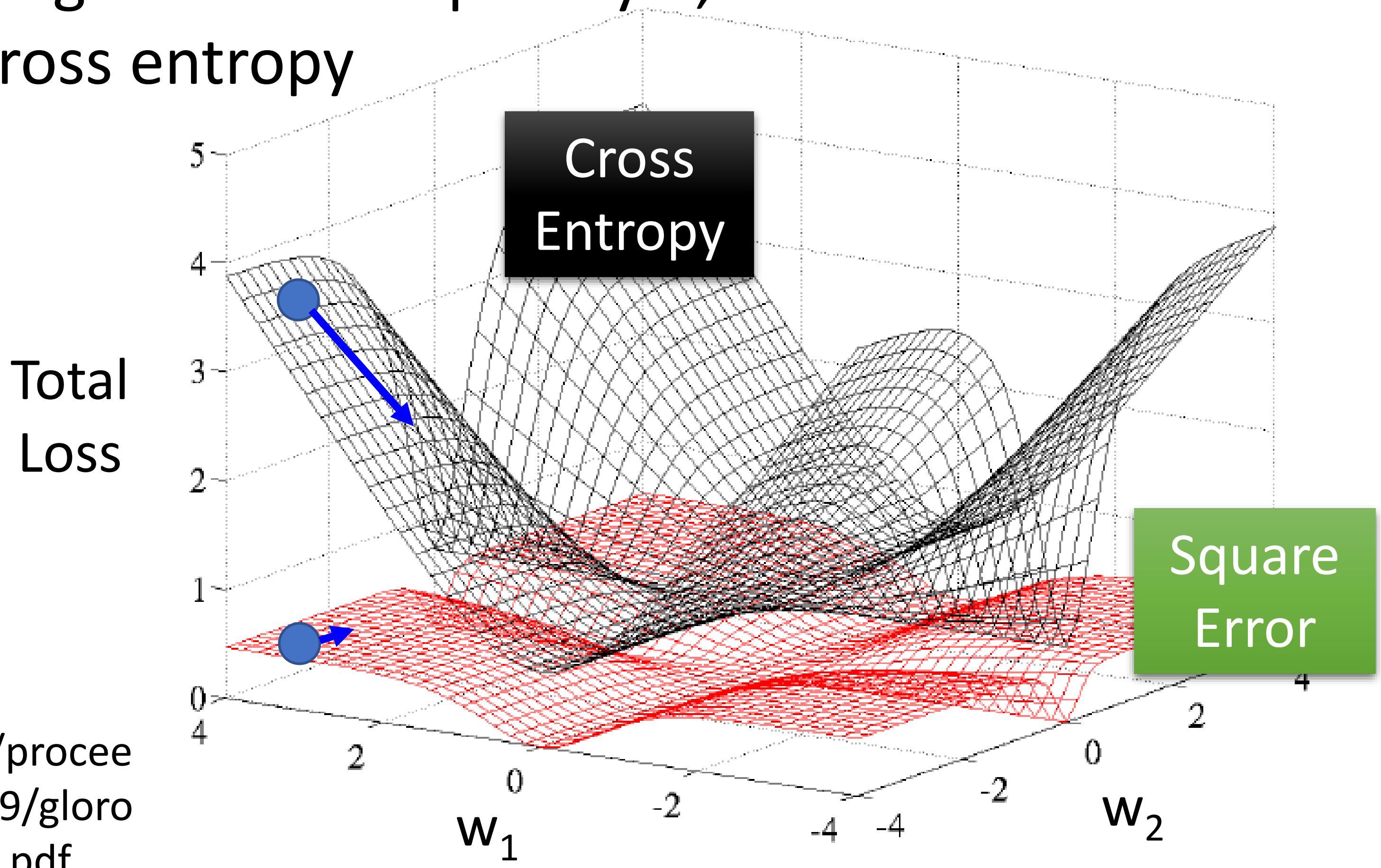
Cross Entropy

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

=0

Choosing Proper Loss

When using softmax output layer, choose cross entropy



http://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf

Let’s try it

Square Error

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

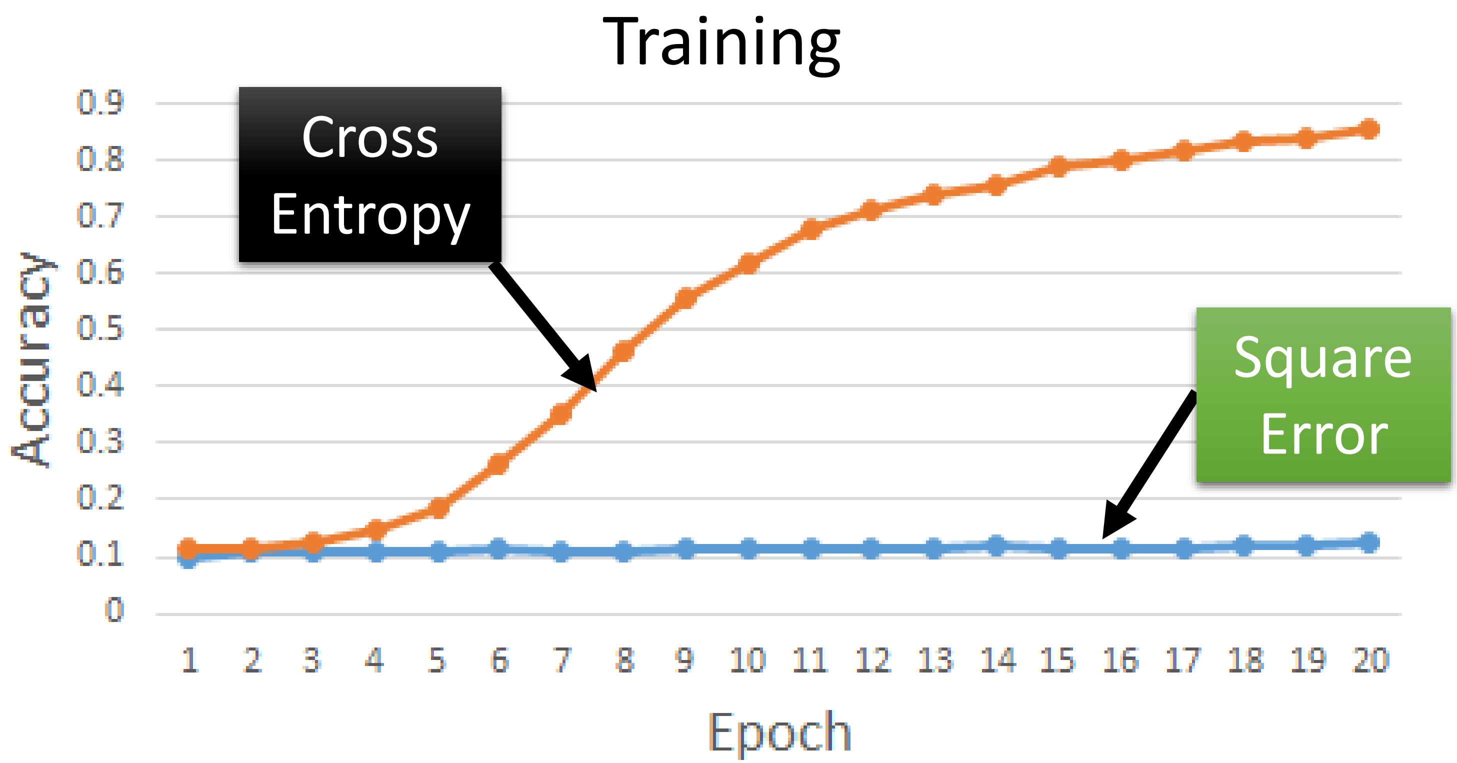
Cross Entropy

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

Let’s try it

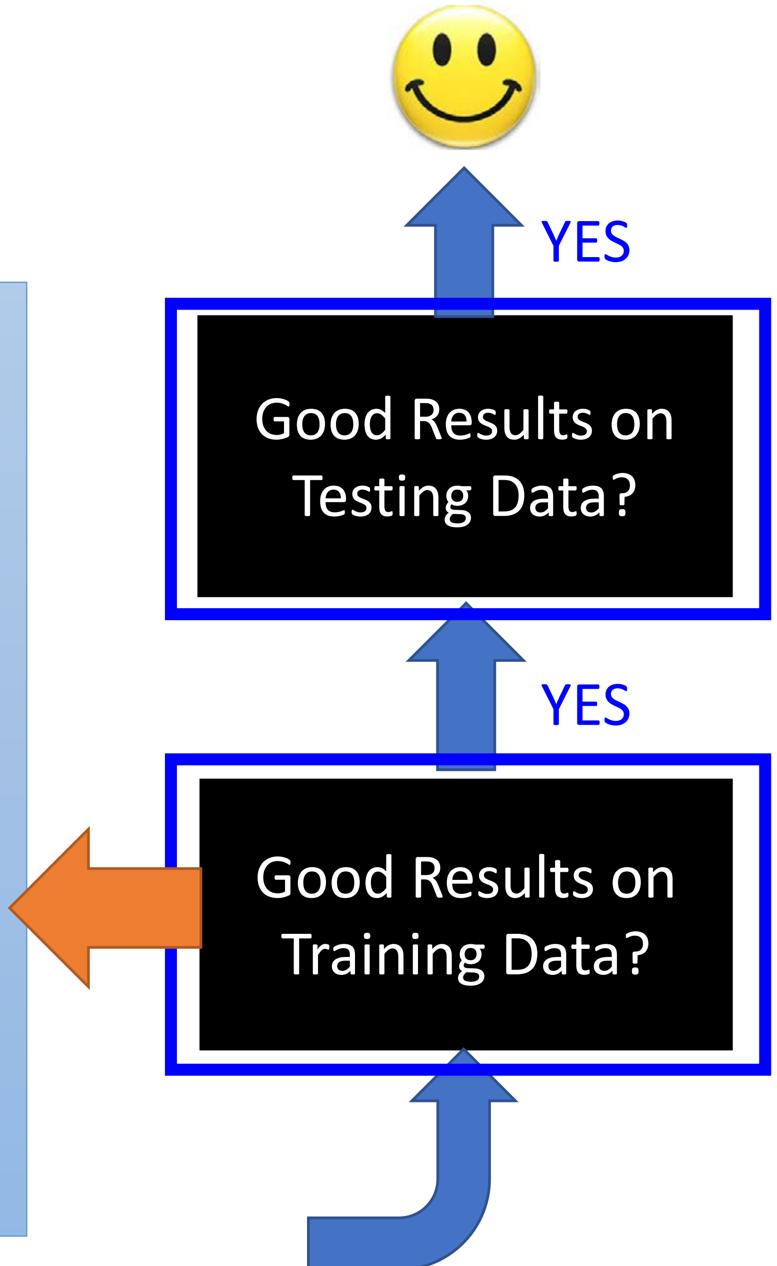
Testing:

	Accuracy
Square Error	0.11
Cross Entropy	0.84



Recipe of Deep Learning

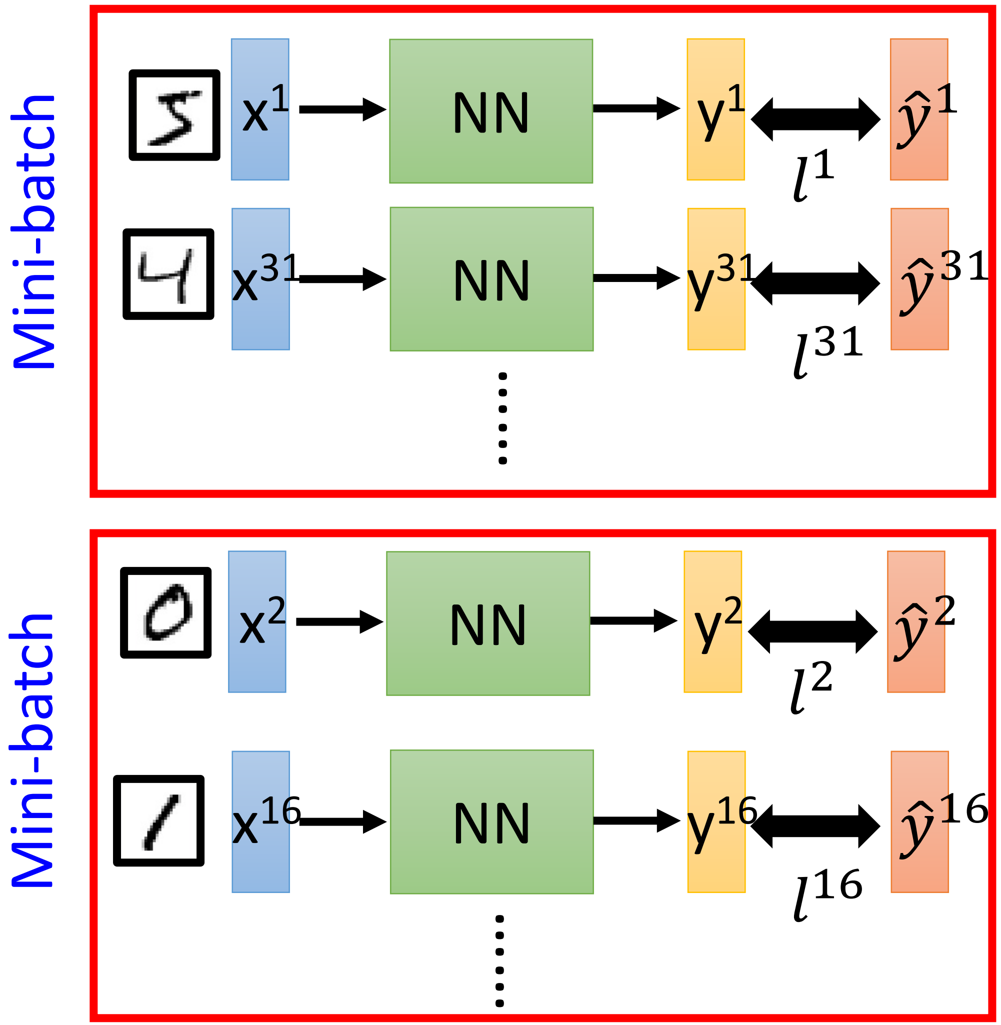
- Choosing proper loss
- Mini-batch
- New activation function
- Adaptive Learning Rate
- Momentum




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

We do not really minimize total loss!

Mini-batch



- Randomly initialize network parameters
- Pick the 1st batch
  $L' = l^1 + l^{31} + \cdots$
  Update parameters once
- Pick the 2nd batch
  $L'' = l^2 + l^{16} + \cdots$
  Update parameters once

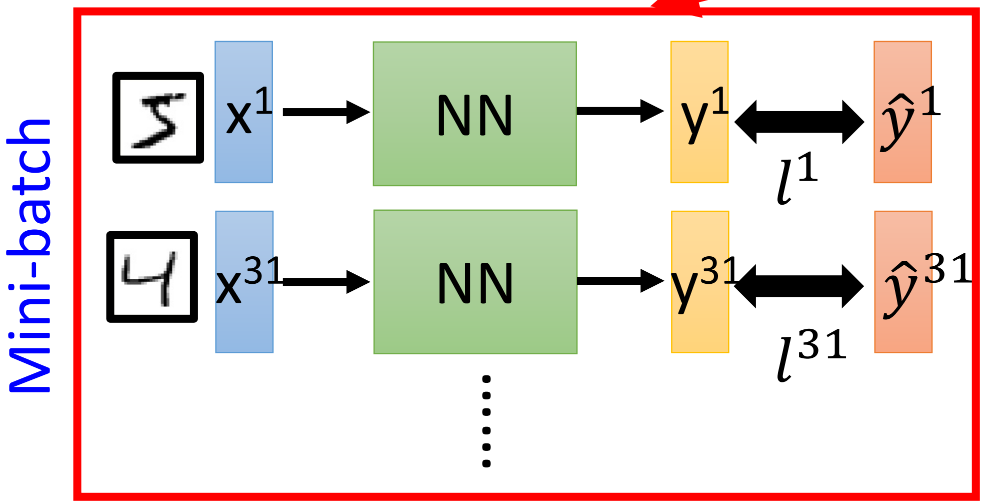
  ⋮
- Until all mini-batches have been picked

one epoch

Repeat the above process

Mini-batch

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



100 examples in a mini-batch

- Pick the 1st batch

  $L' = l^1 + l^{31} + \cdots$

  Update parameters once

- Pick the 2nd batch

  $L'' = l^2 + l^{16} + \cdots$

  Update parameters once

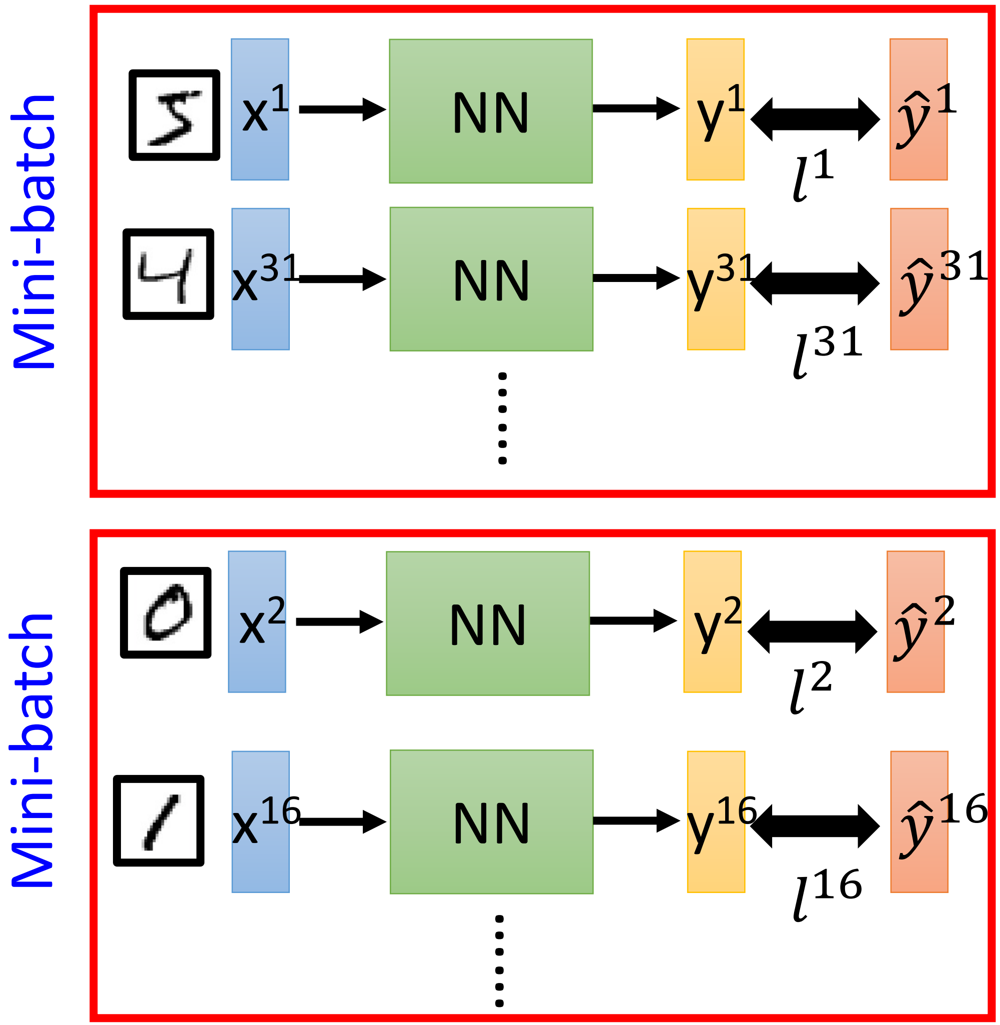
  ⋮

- Until all mini-batches have been picked

one epoch

Repeat 20 times

Mini-batch



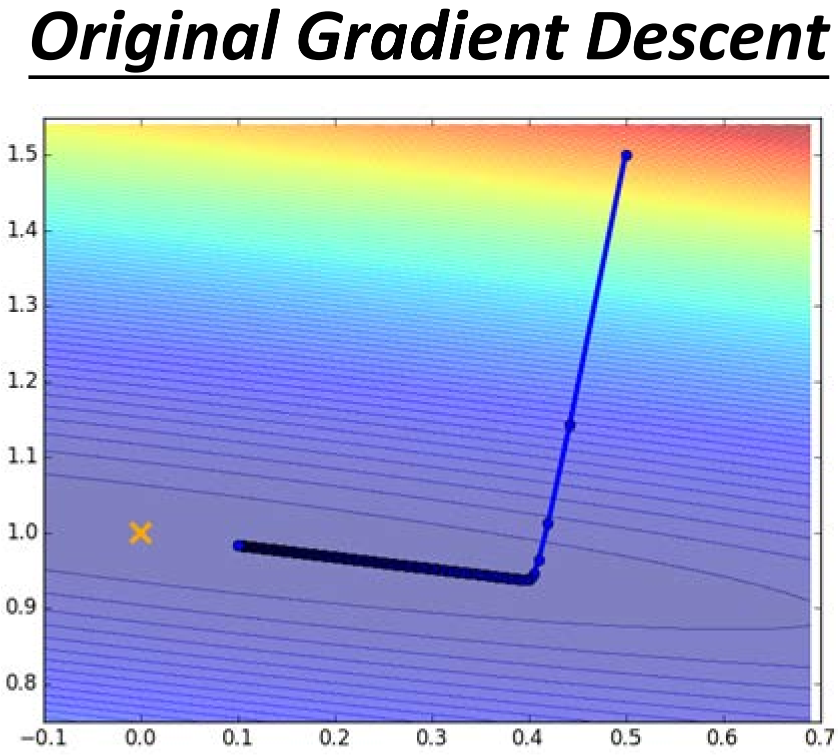
- Randomly initialize network parameters
- Pick the 1st batch
  $L' = l^1 + l^{31} + \cdots$
  Update parameters once
- Pick the 2nd batch
  $L'' = l^2 + l^{16} + \cdots$
  Update parameters once

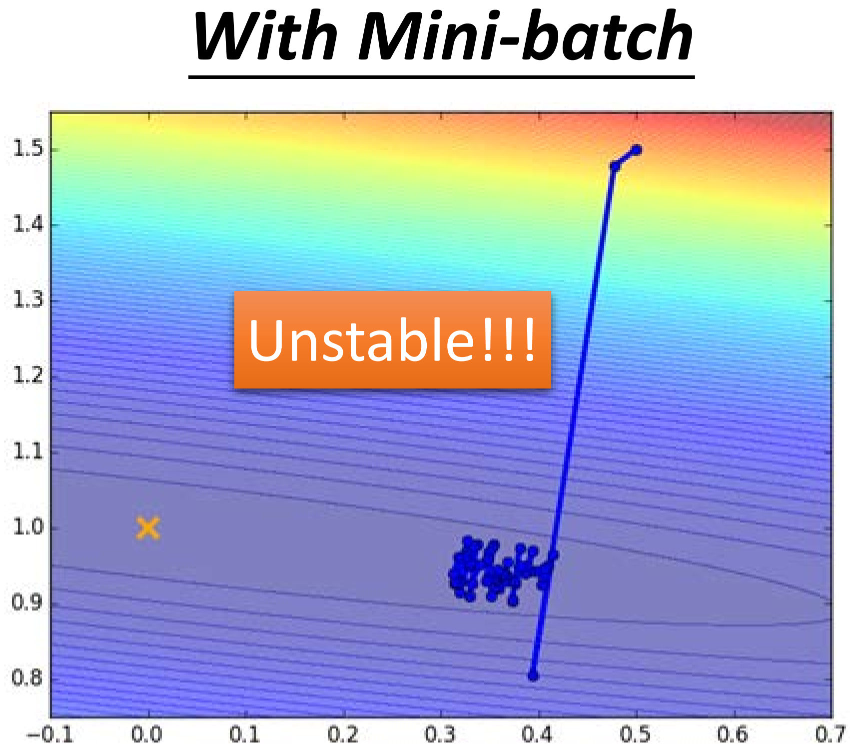
⋮

L is different each time when we update parameters!

目標換來換去?!

Mini-batch





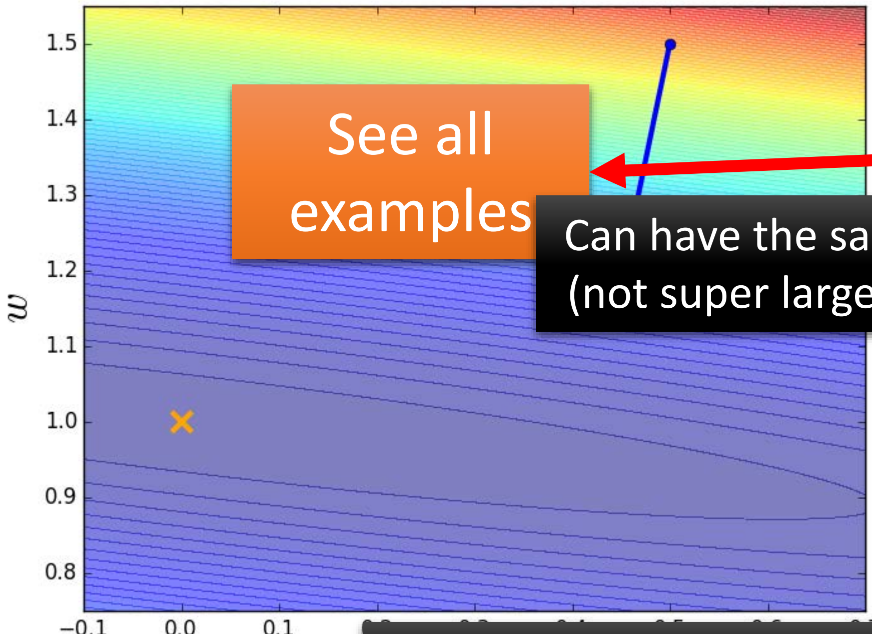
The colors represent the total loss.

Mini-batch is Faster

Not always true with parallel computing.

Original Gradient Descent

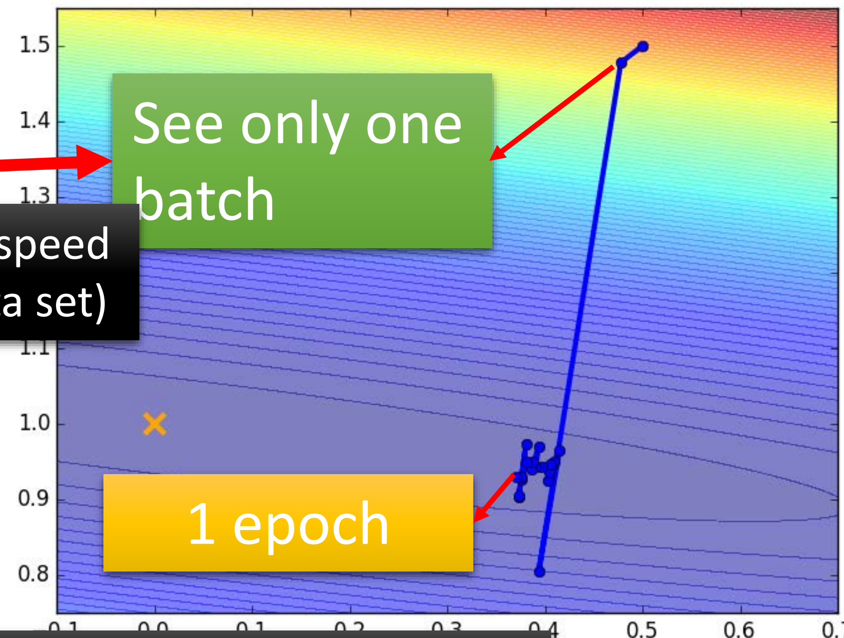
Update after seeing all examples



See all examples

With Mini-batch

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



See only one batch

1 epoch

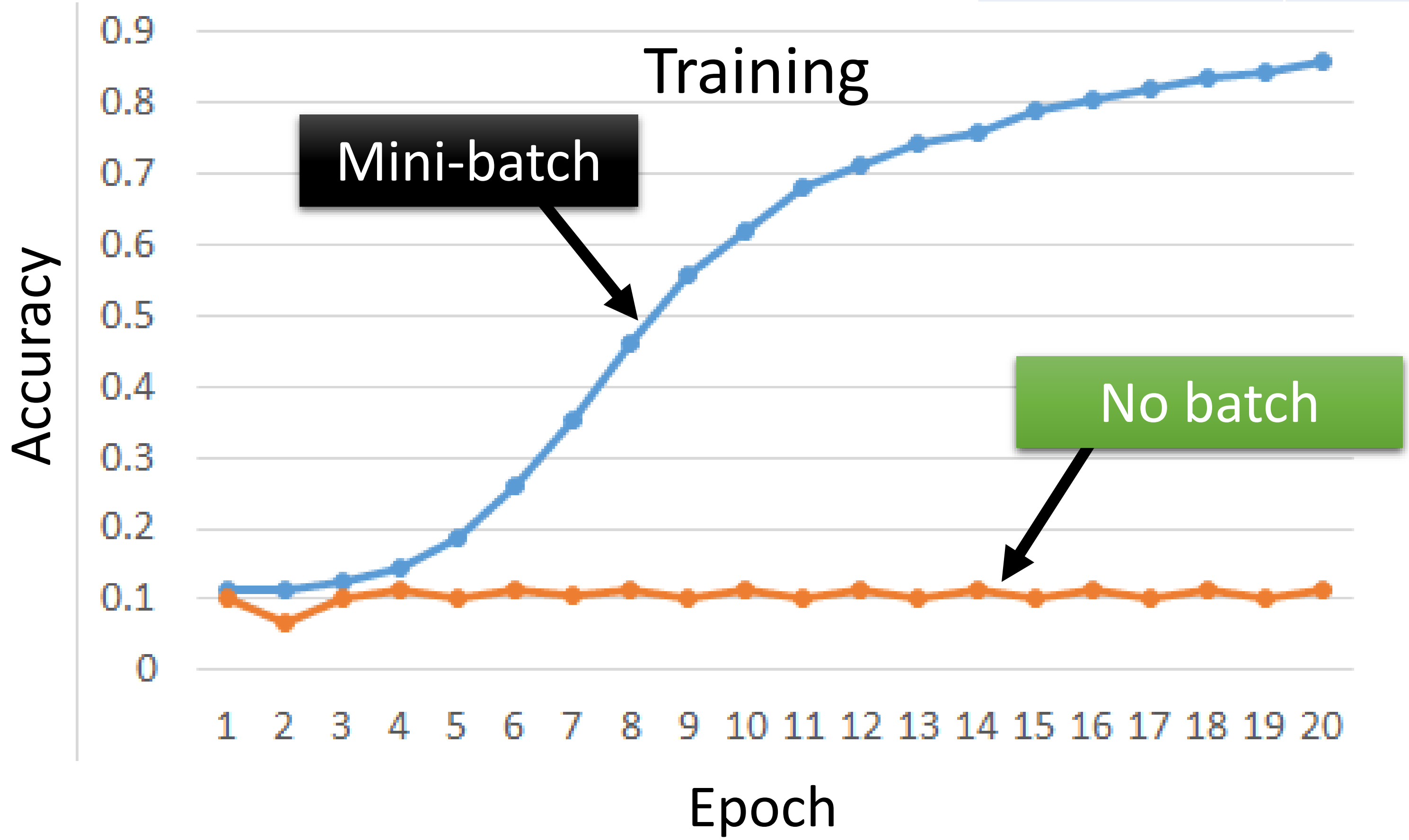
Can have the same speed (not super large data set)

Mini-batch has better performance!

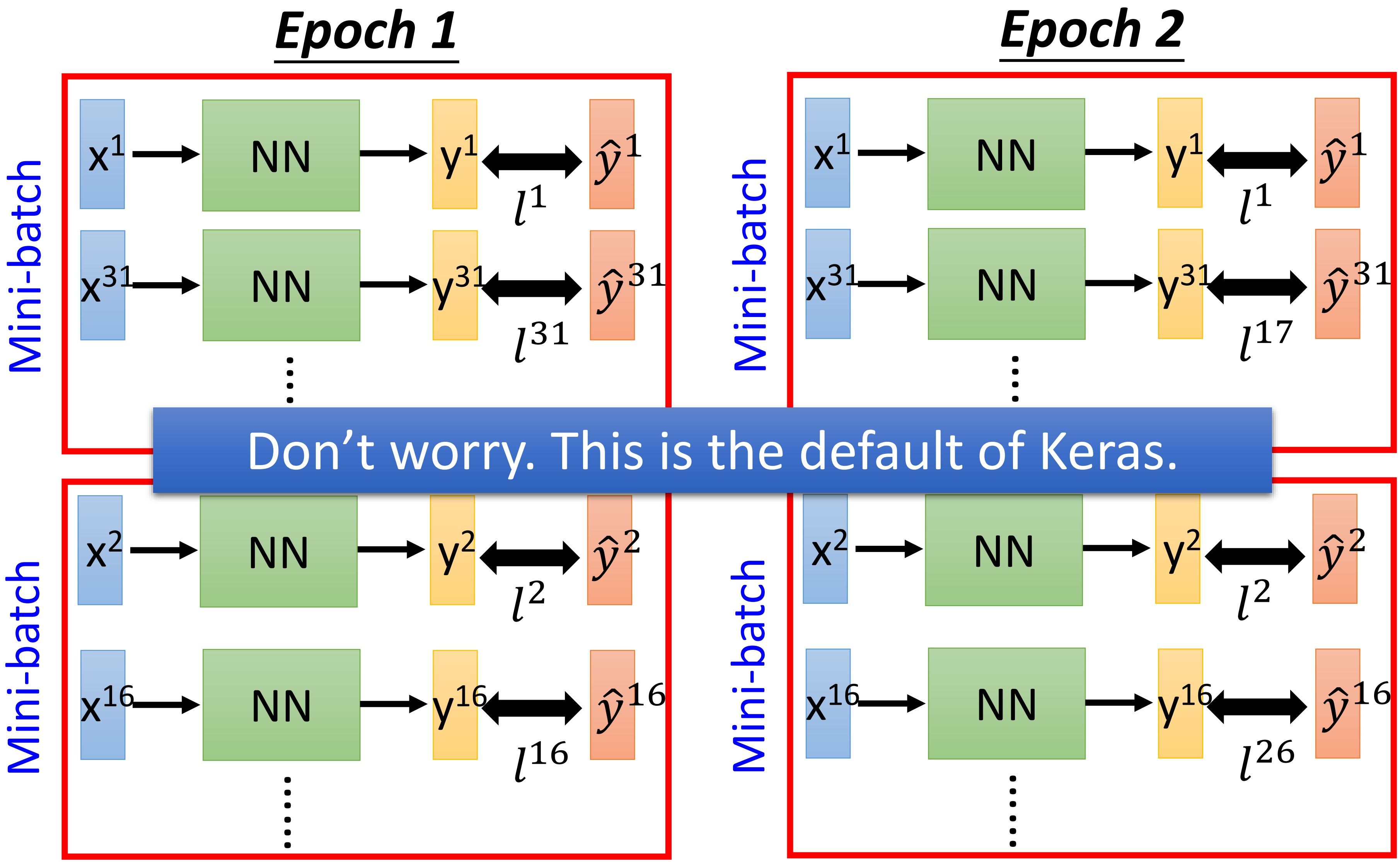
Mini-batch is Better!

Testing:

	Accuracy
Mini-batch	0.84
No batch	0.12

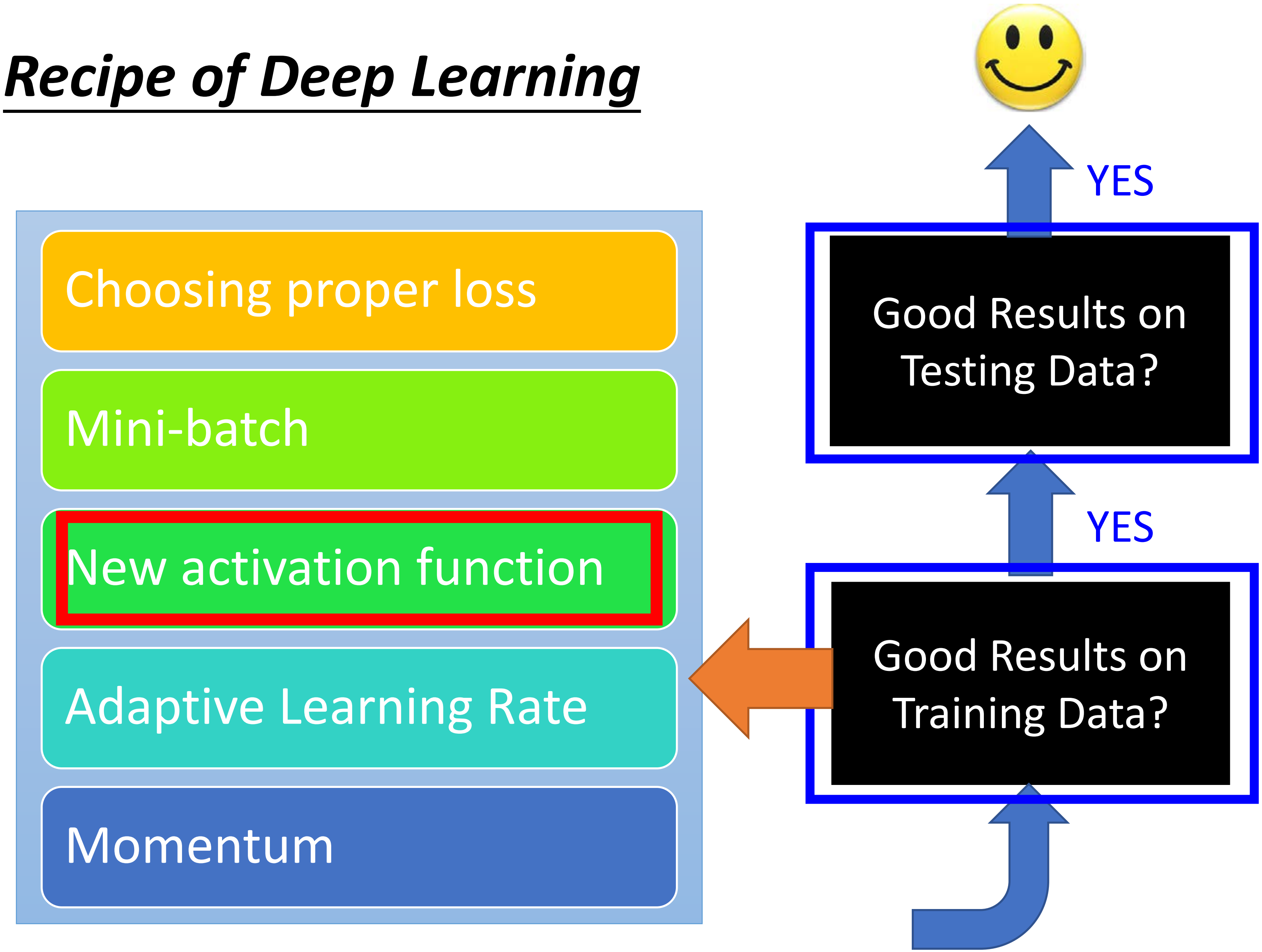


Shuffle the training examples for each epoch

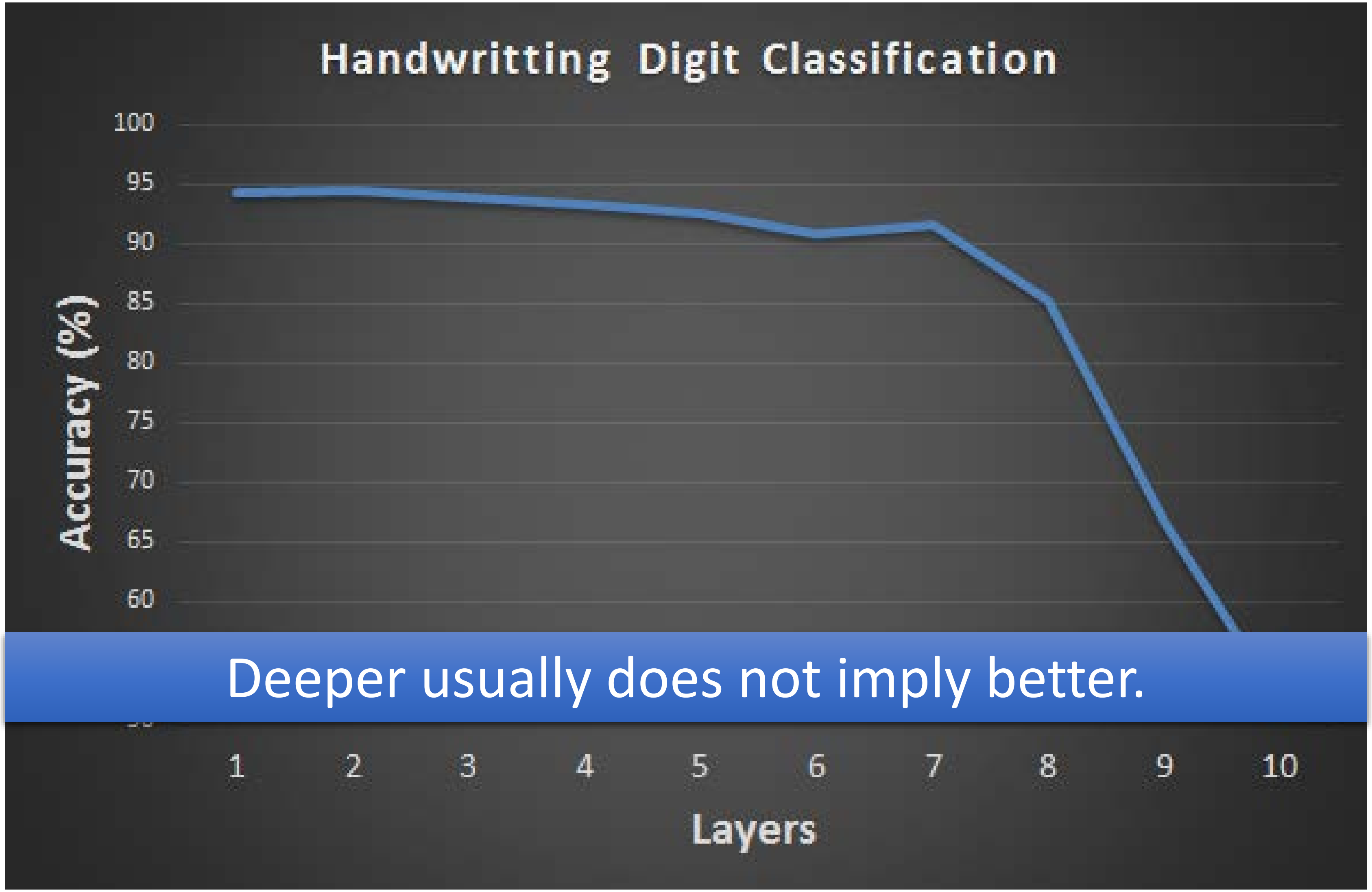


Don't worry. This is the default of Keras.

Recipe of Deep Learning



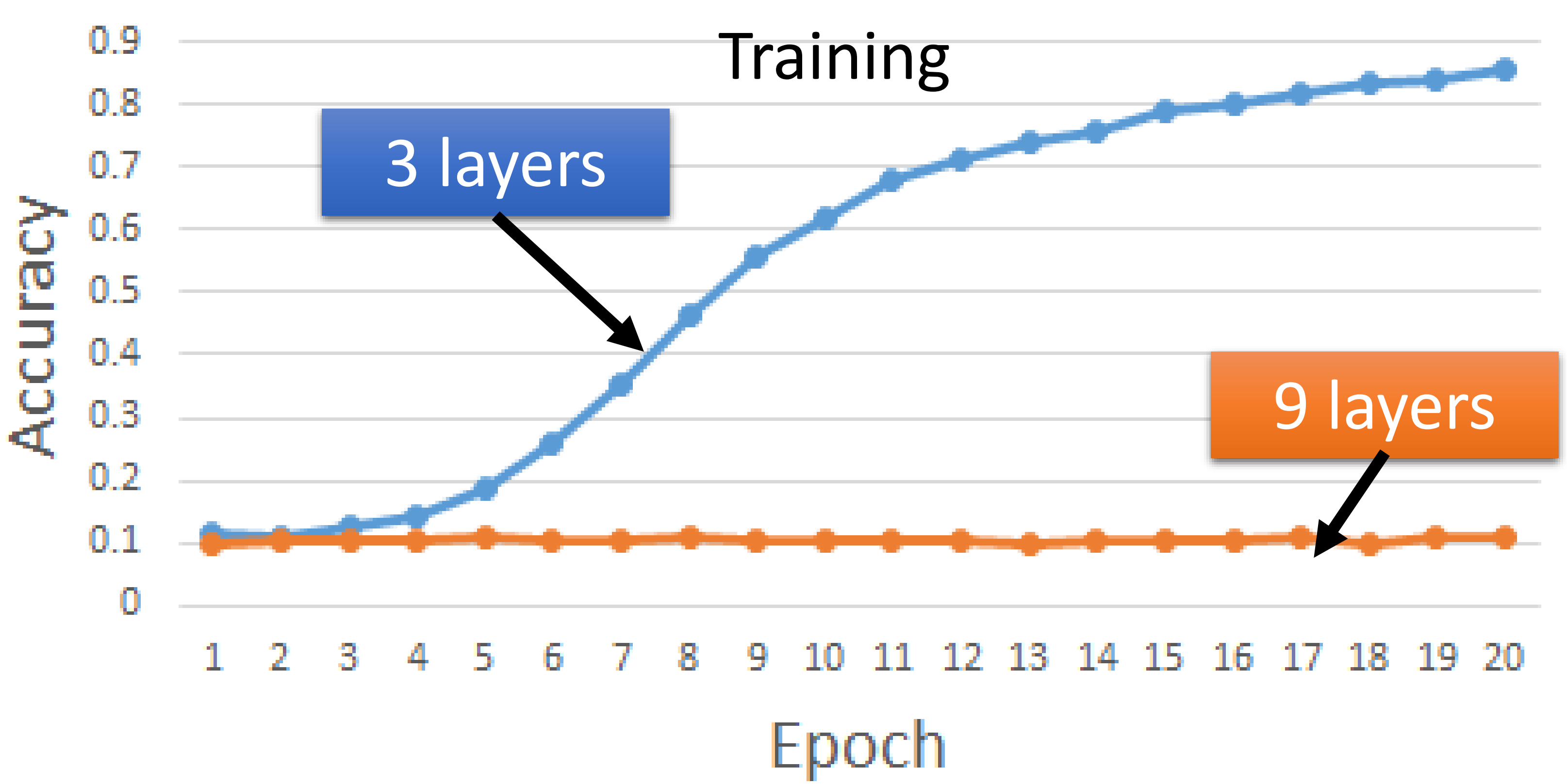
Hard to get the power of Deep …



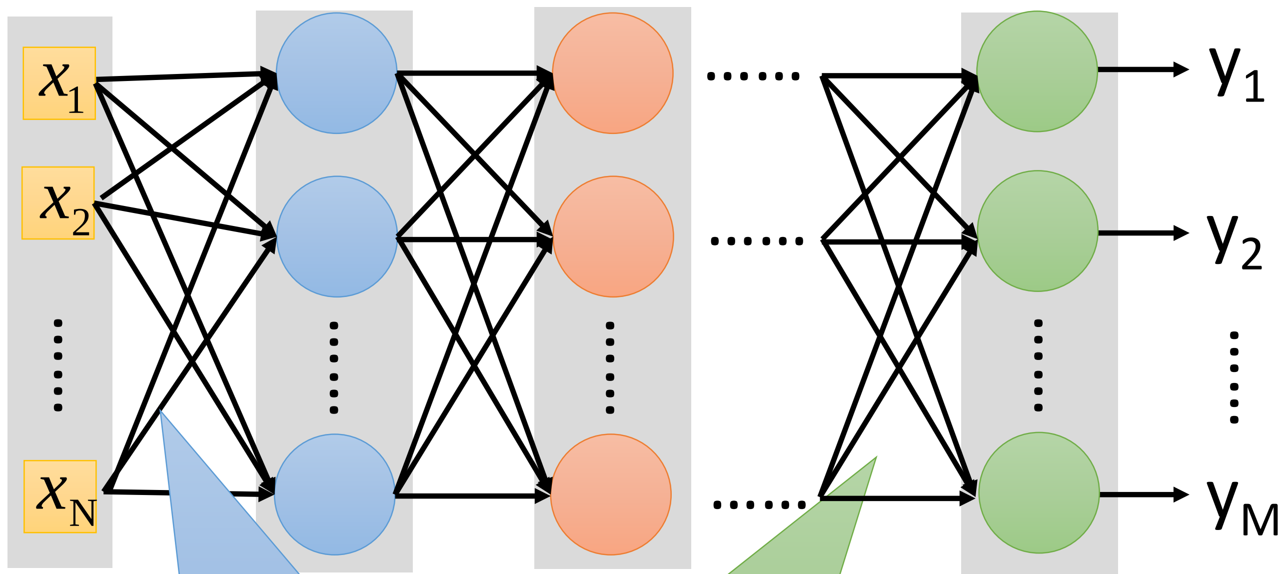
Let's try it

Testing:

	Accuracy
3 layers	0.84
9 layers	0.11



Vanishing Gradient Problem



Smaller gradients

Learn very slow

Almost random

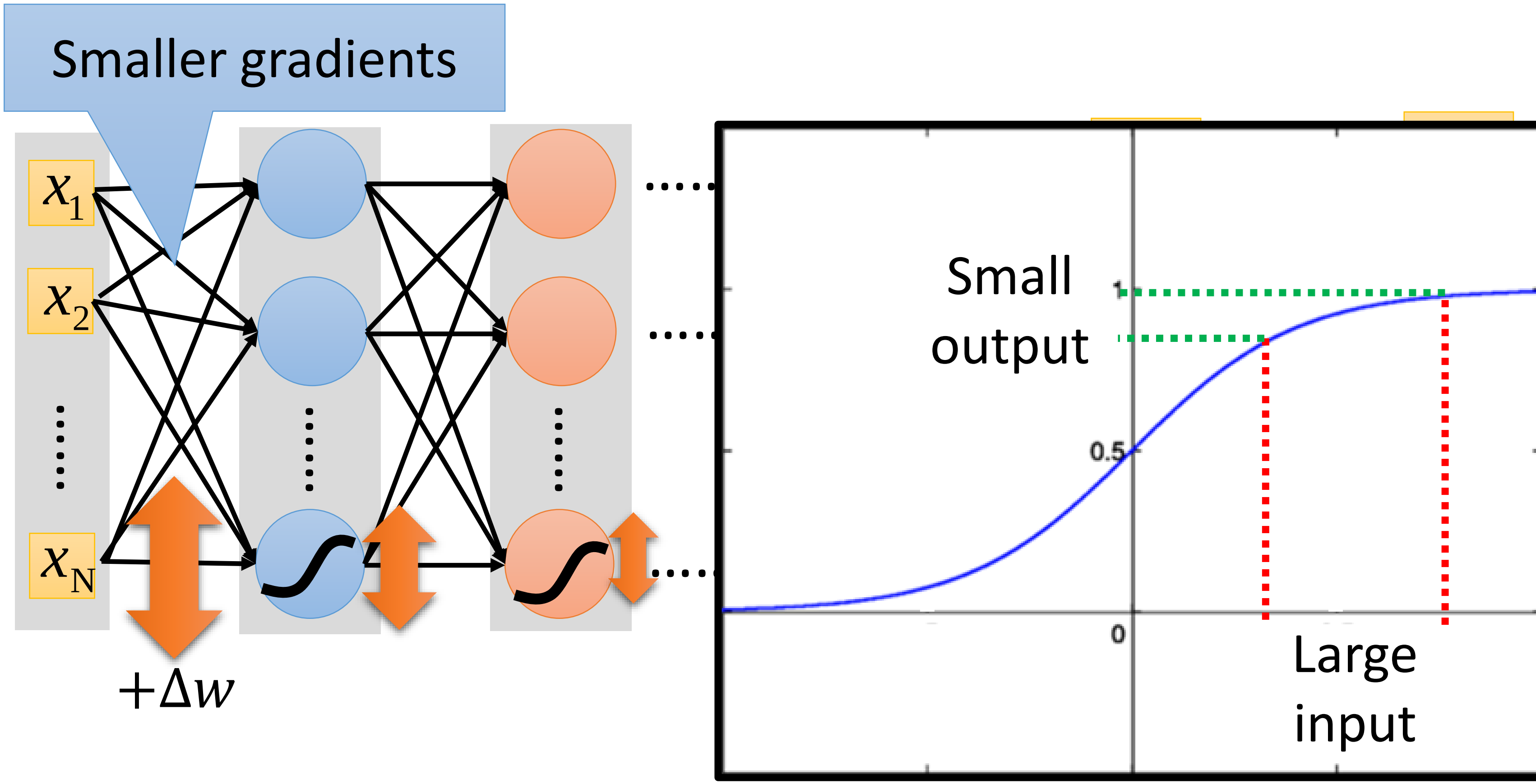
Larger gradients

Learn very fast

Already converge

based on random!?

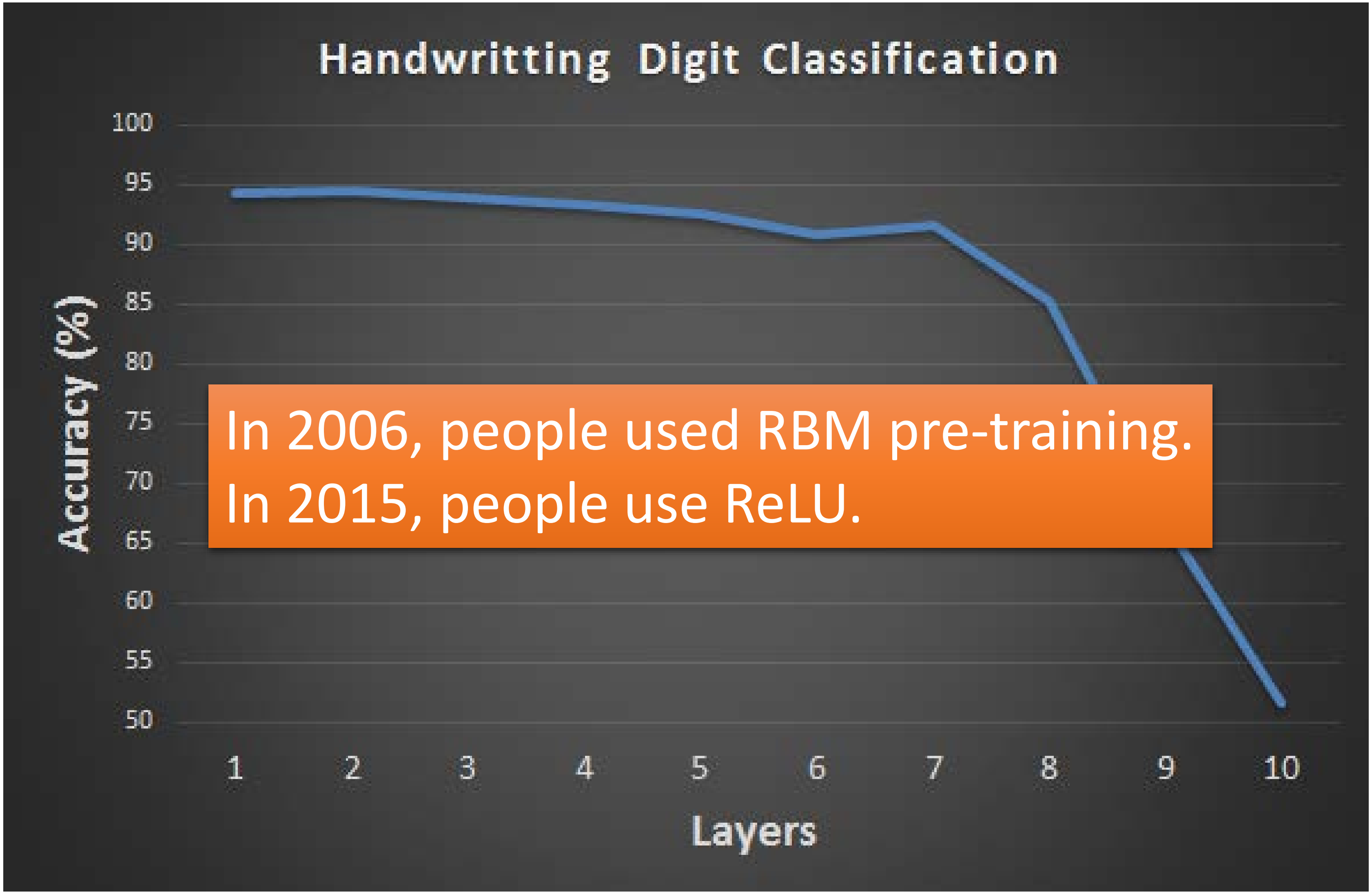
Vanishing Gradient Problem



Intuitive way to compute the derivatives …

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

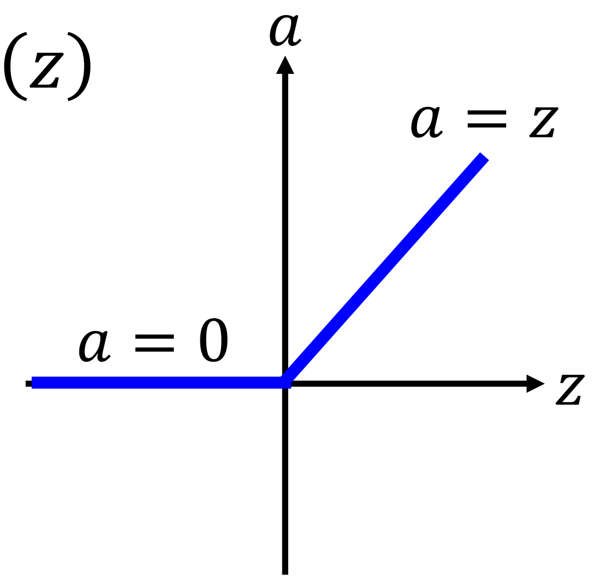
Hard to get the power of Deep …



ReLU

- Rectified Linear Unit (ReLU)

$\sigma(z)$

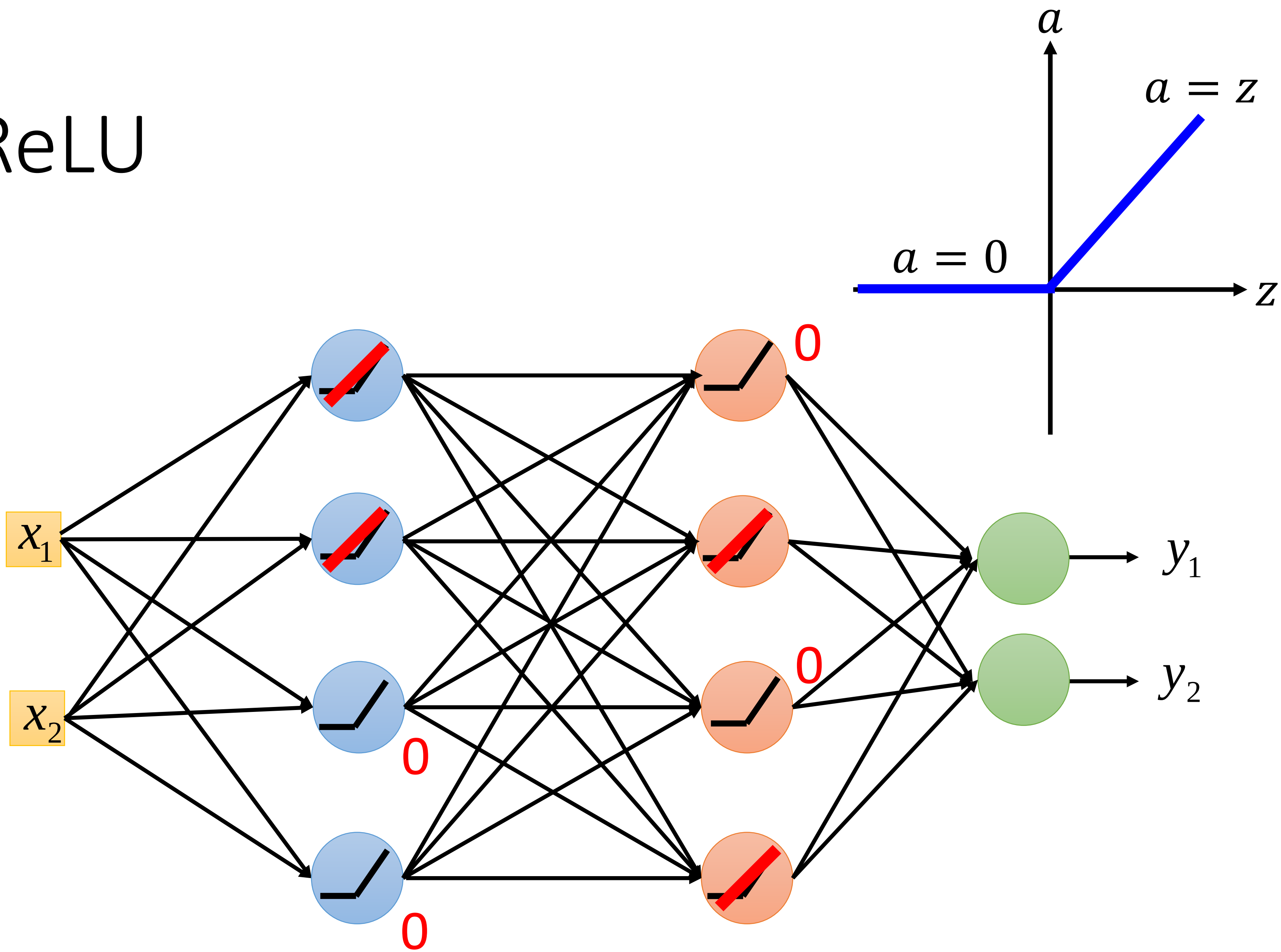


[Xavier Glorot, AISTATS'11]
[Andrew L. Maas, ICML'13]
[Kaiming He, arXiv'15]

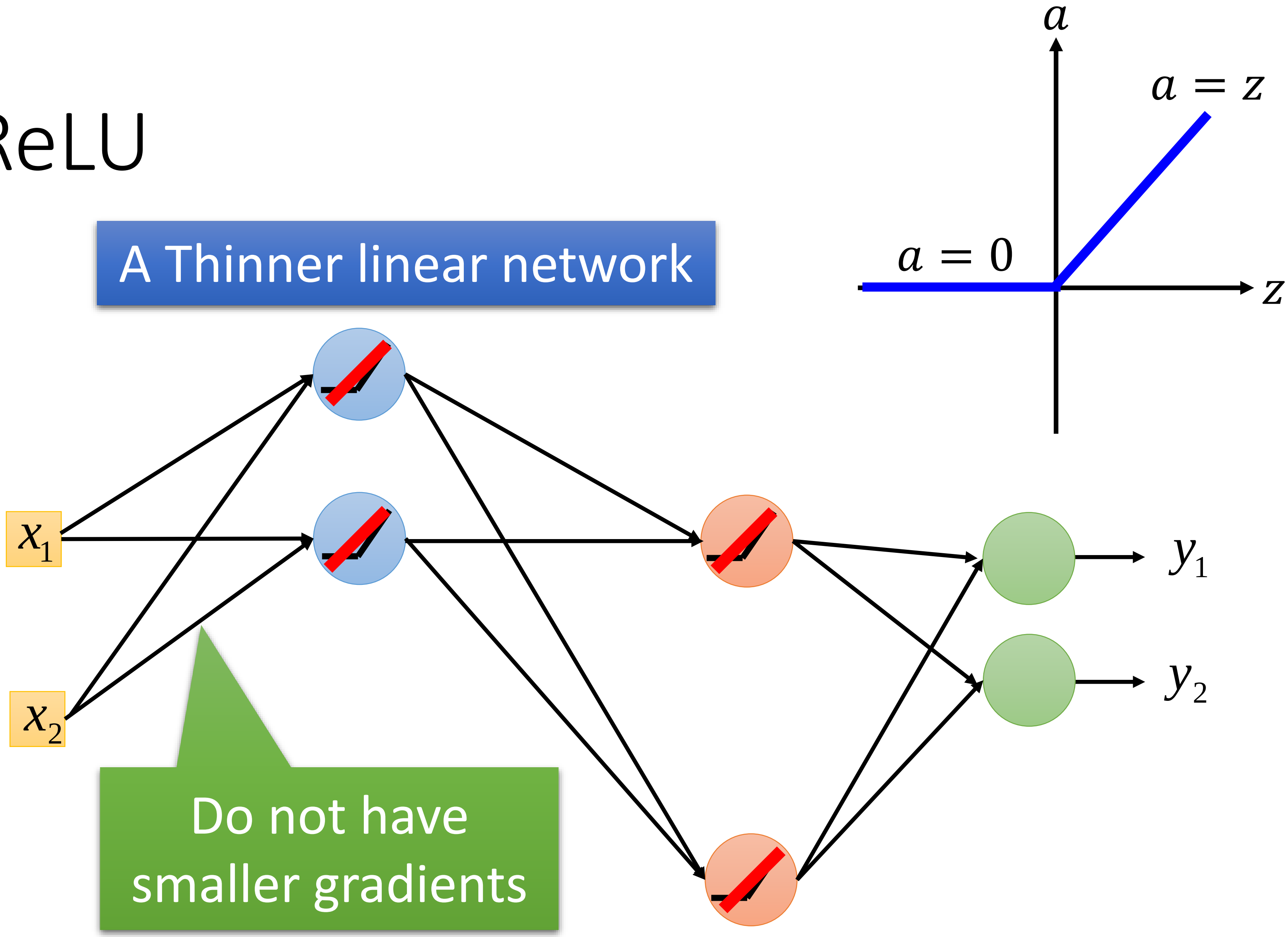
Reason:

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

ReLU



ReLU



A Thinner linear network
Do not have smaller gradients
a
$a = z$
$a = 0$
z
x_1
x_2
y_1
y_2

Let’s try it

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

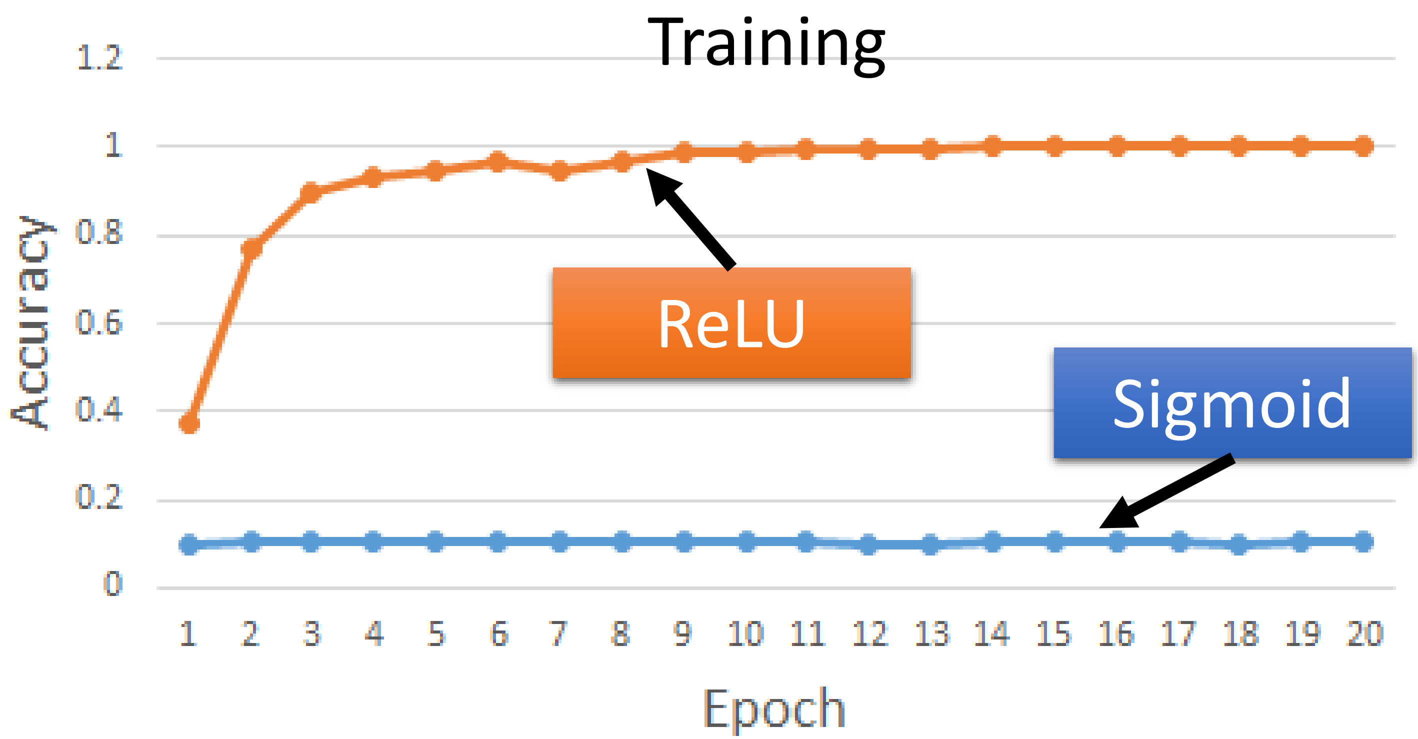
```
model.add( Activation('relu') )
```

Let's try it

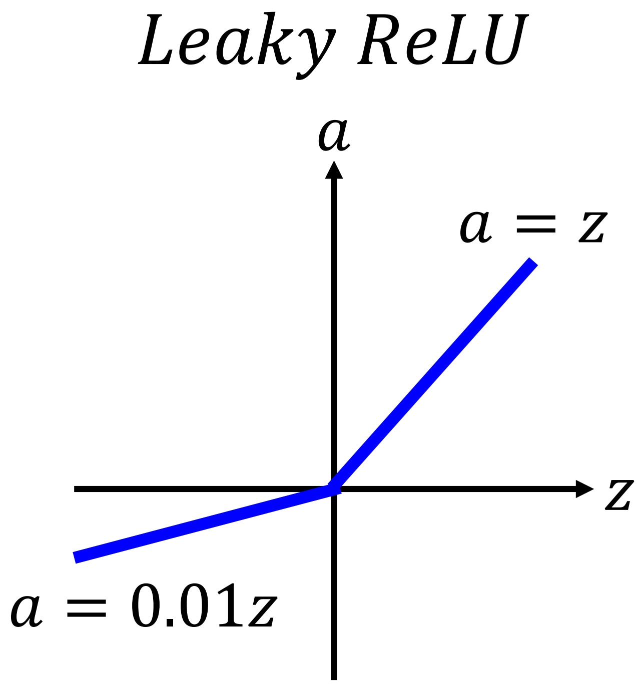
• 9 layers

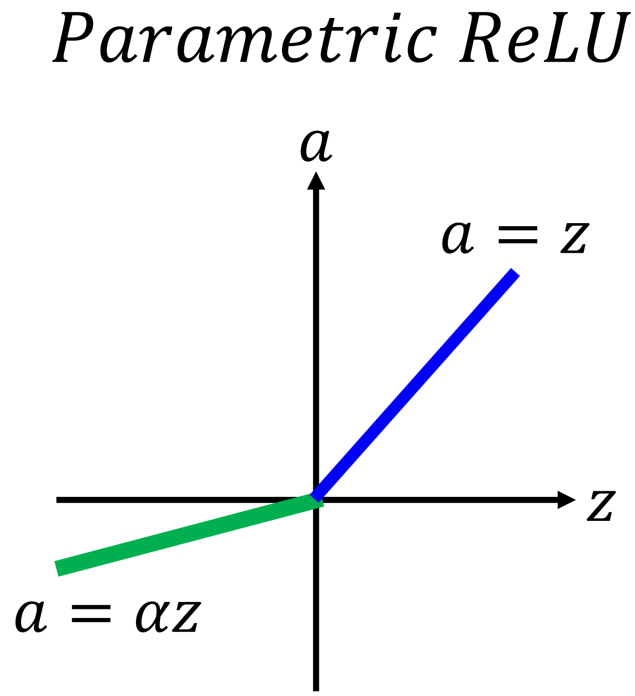
Testing:

9 layers	Accuracy
Sigmoid	0.11
ReLU	0.96



ReLU - variant



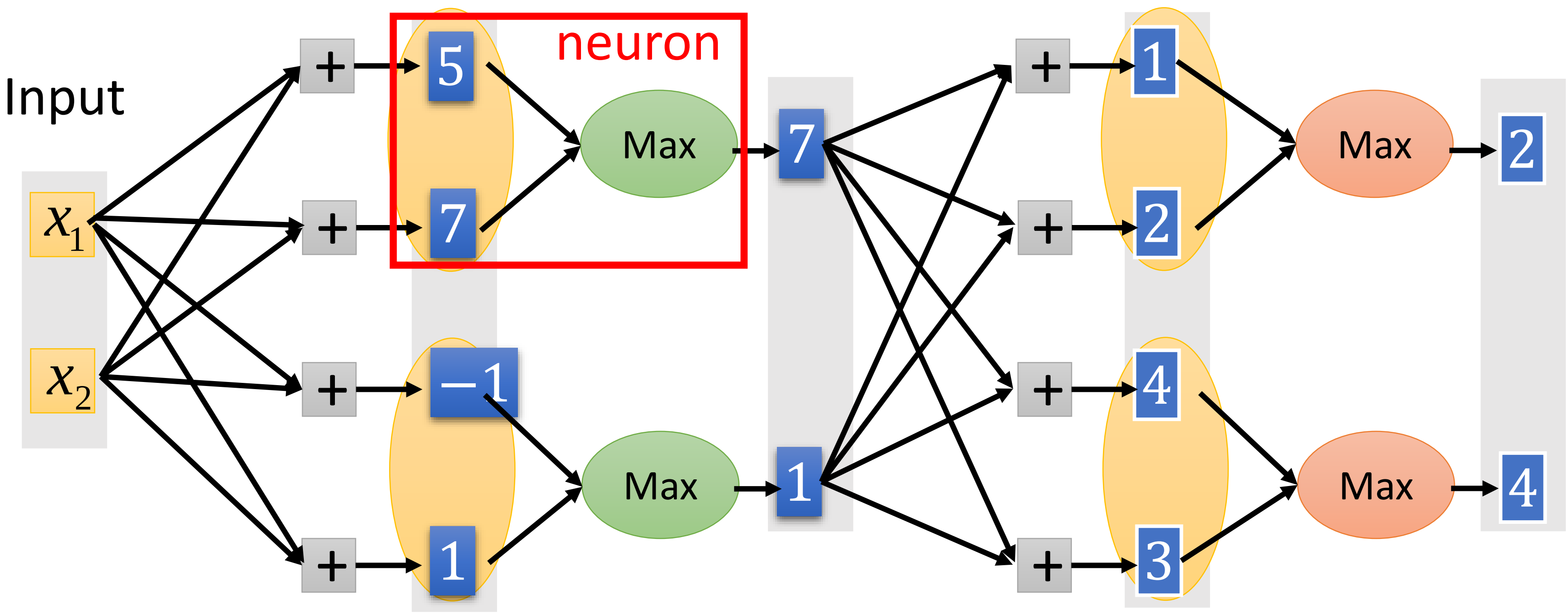


α also learned by gradient descent

Maxout

ReLU is a special cases of Maxout

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



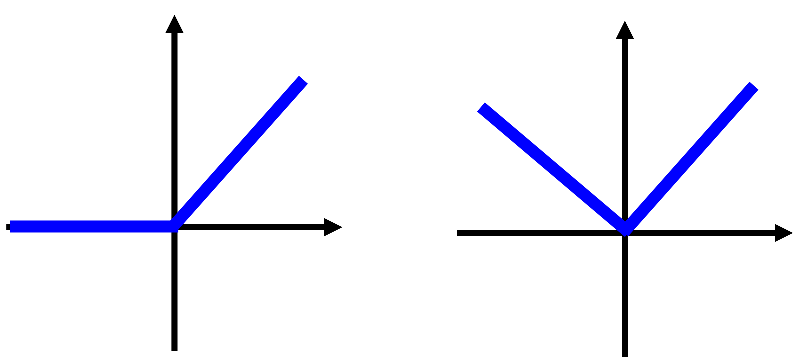
You can have more than 2 elements in a group.

Maxout

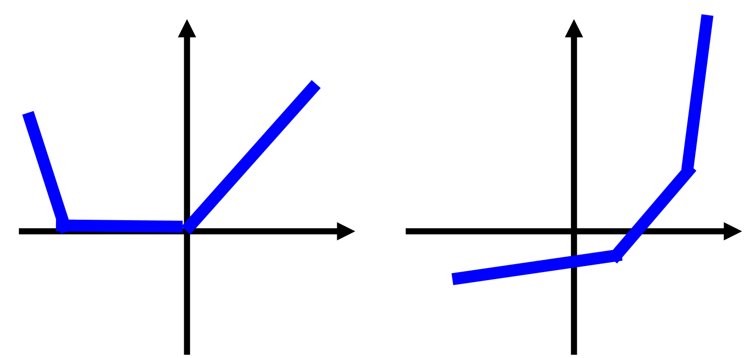
ReLU is a special cases of Maxout

- 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

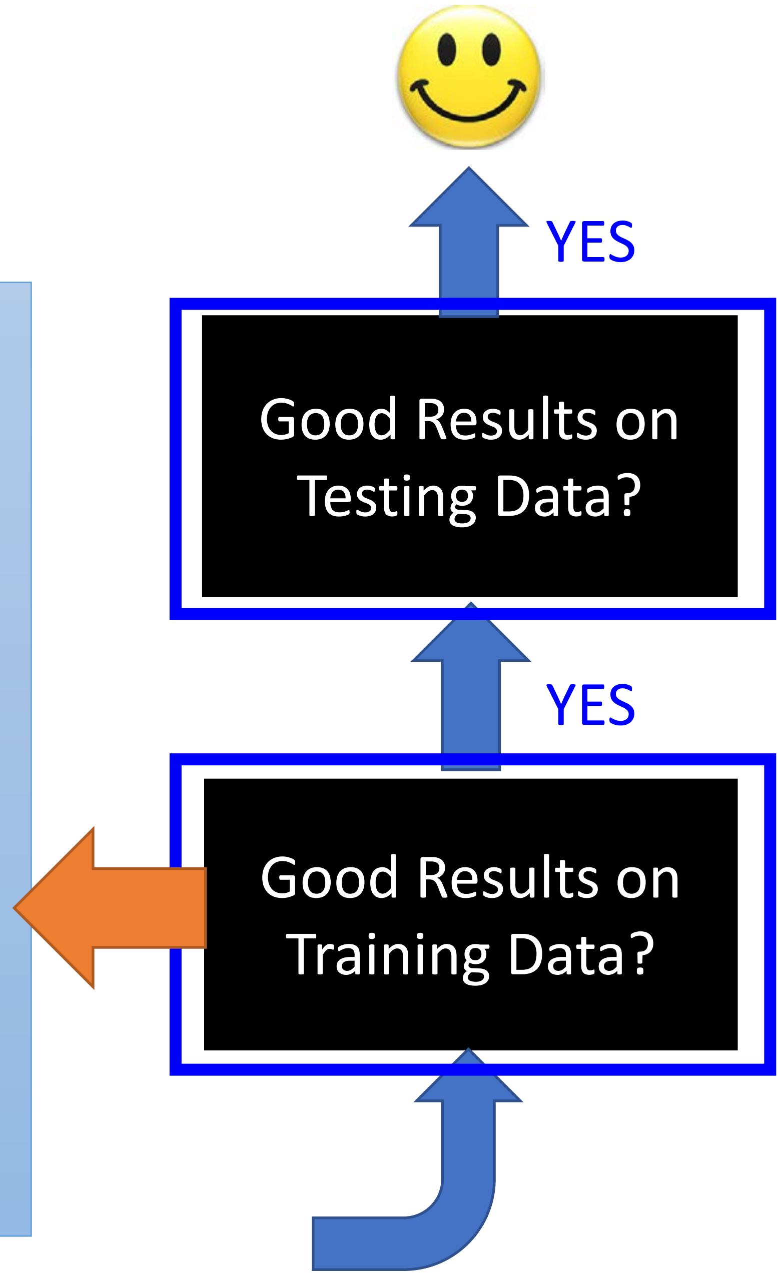


3 elements in a group



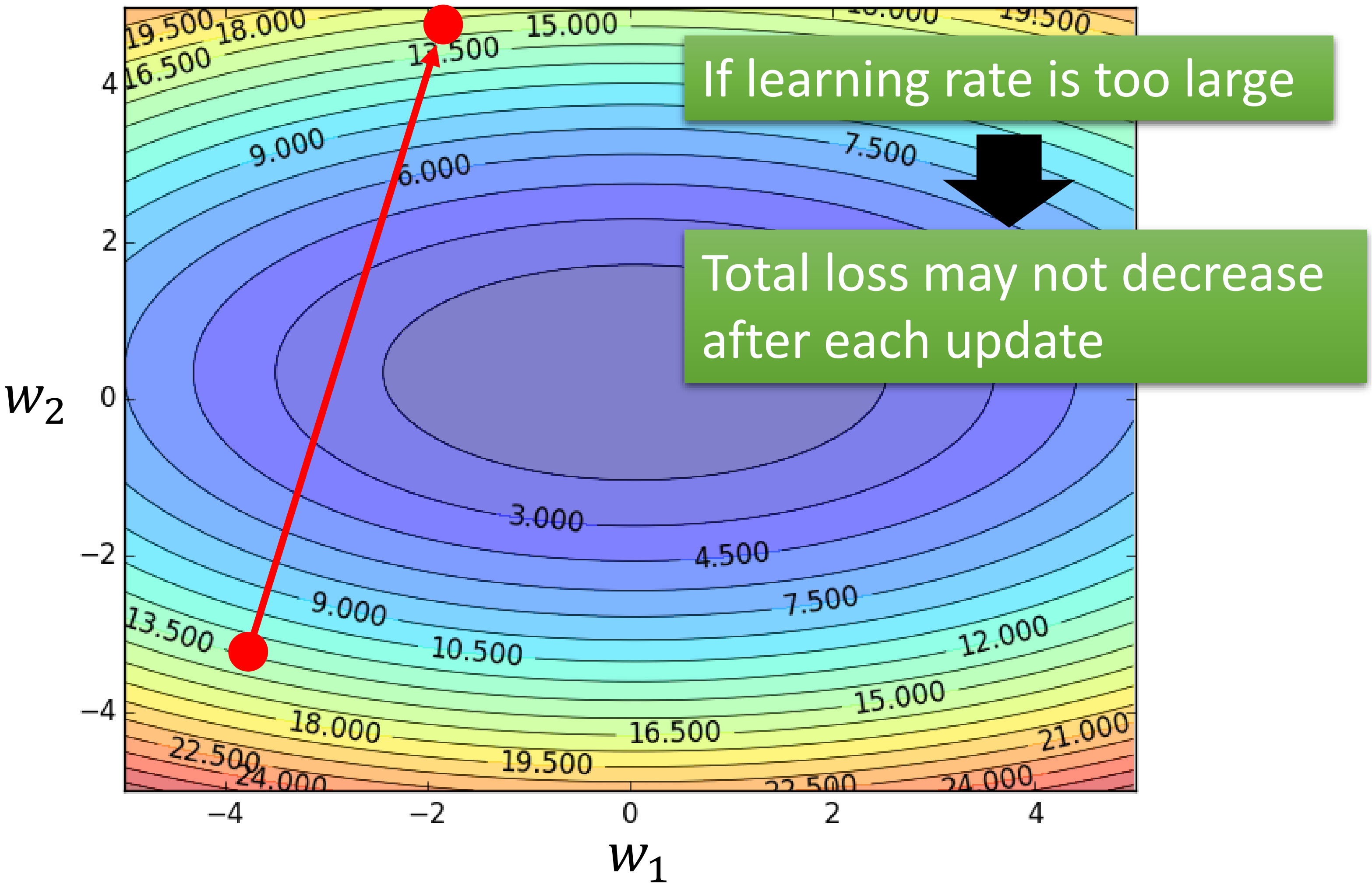
Recipe of Deep Learning

- Choosing proper loss
- Mini-batch
- New activation function
- Adaptive Learning Rate
- Momentum



Learning Rates

Set the learning rate η carefully



Learning Rates

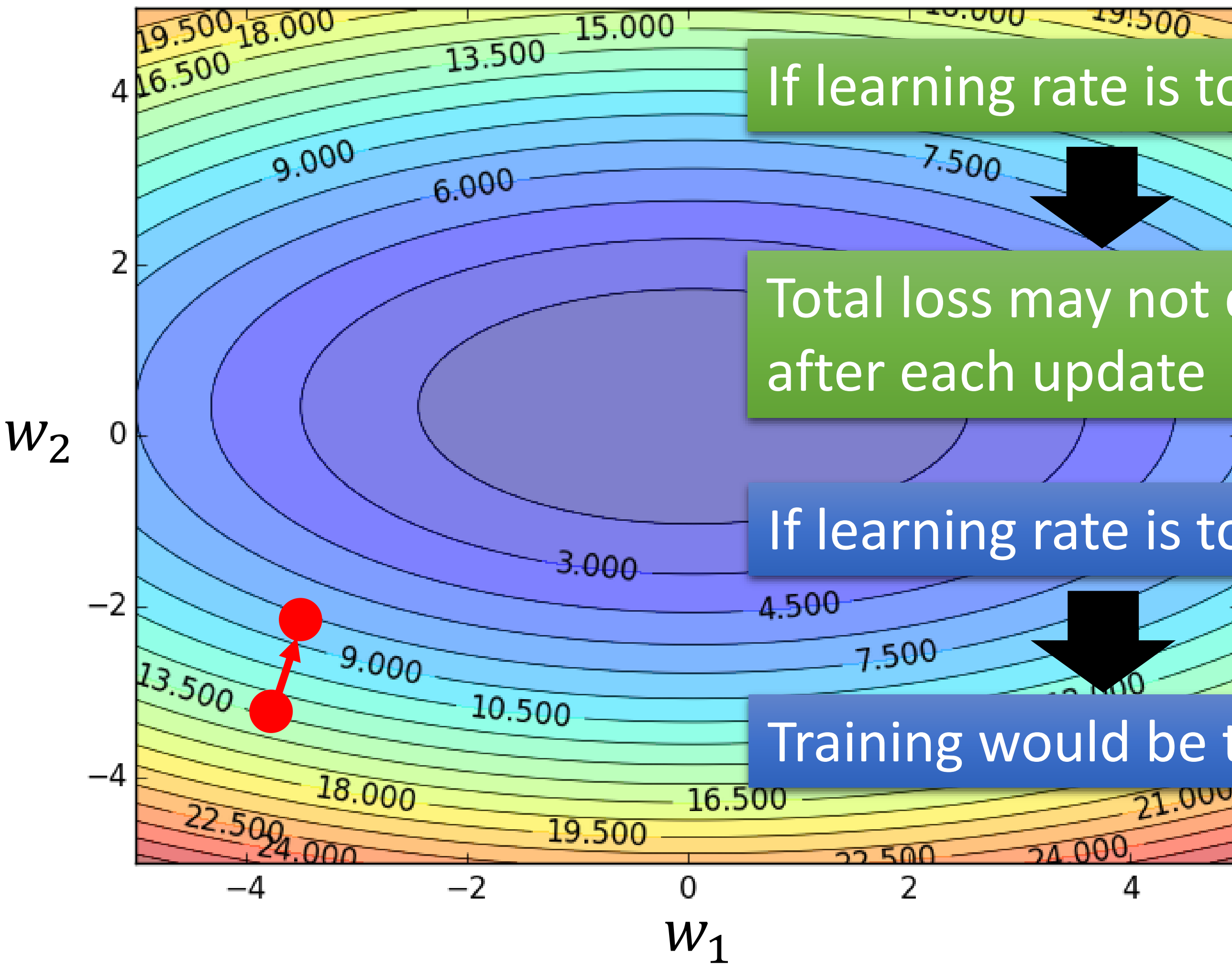
Set the learning rate η carefully

If learning rate is too large

Total loss may not decrease after each update

If learning rate is too small

Training would be too slow



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

Parameter dependent learning rate

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

constant

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

Summation of the square of the previous derivatives

Adagrad

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

w_1	g^0
	0.1

w_2	g^0
	20.0

Learning rate:

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

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

Learning rate:

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

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

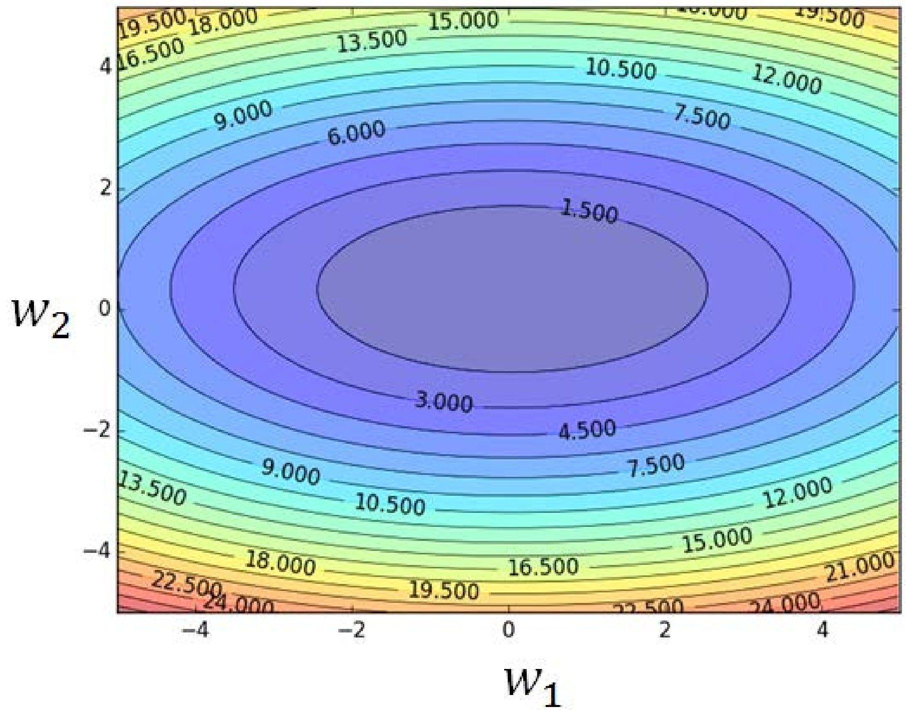
Observation: 1. Learning rate is smaller and smaller for all parameters

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

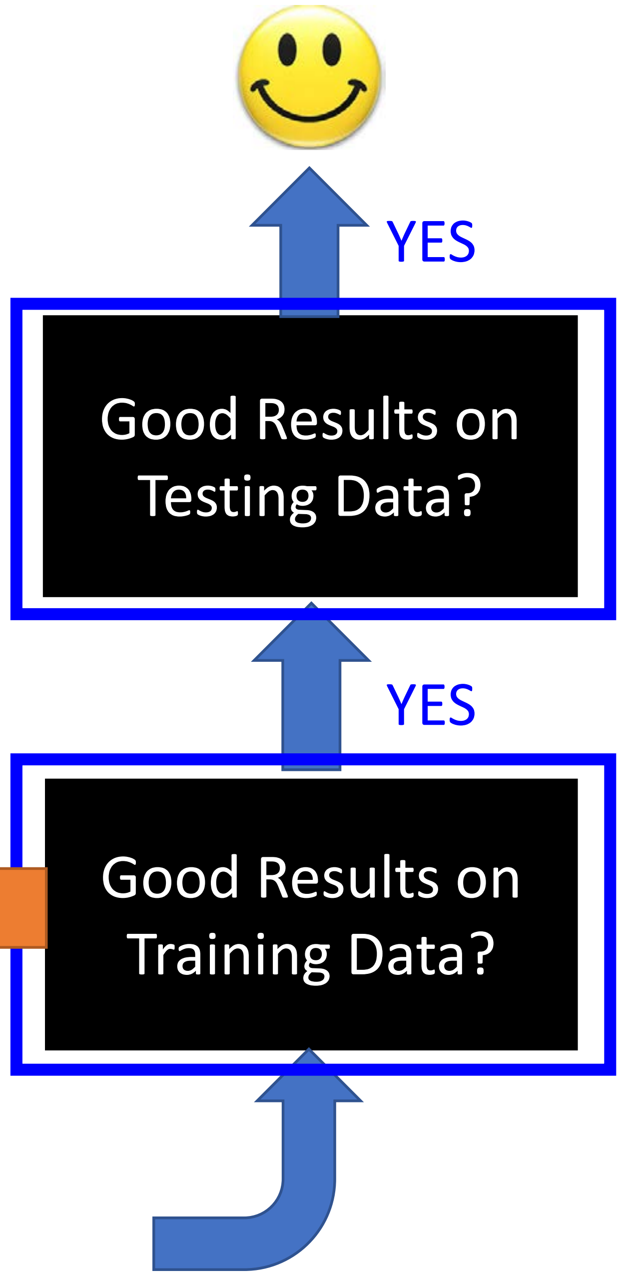
Why?

Not the whole story ……

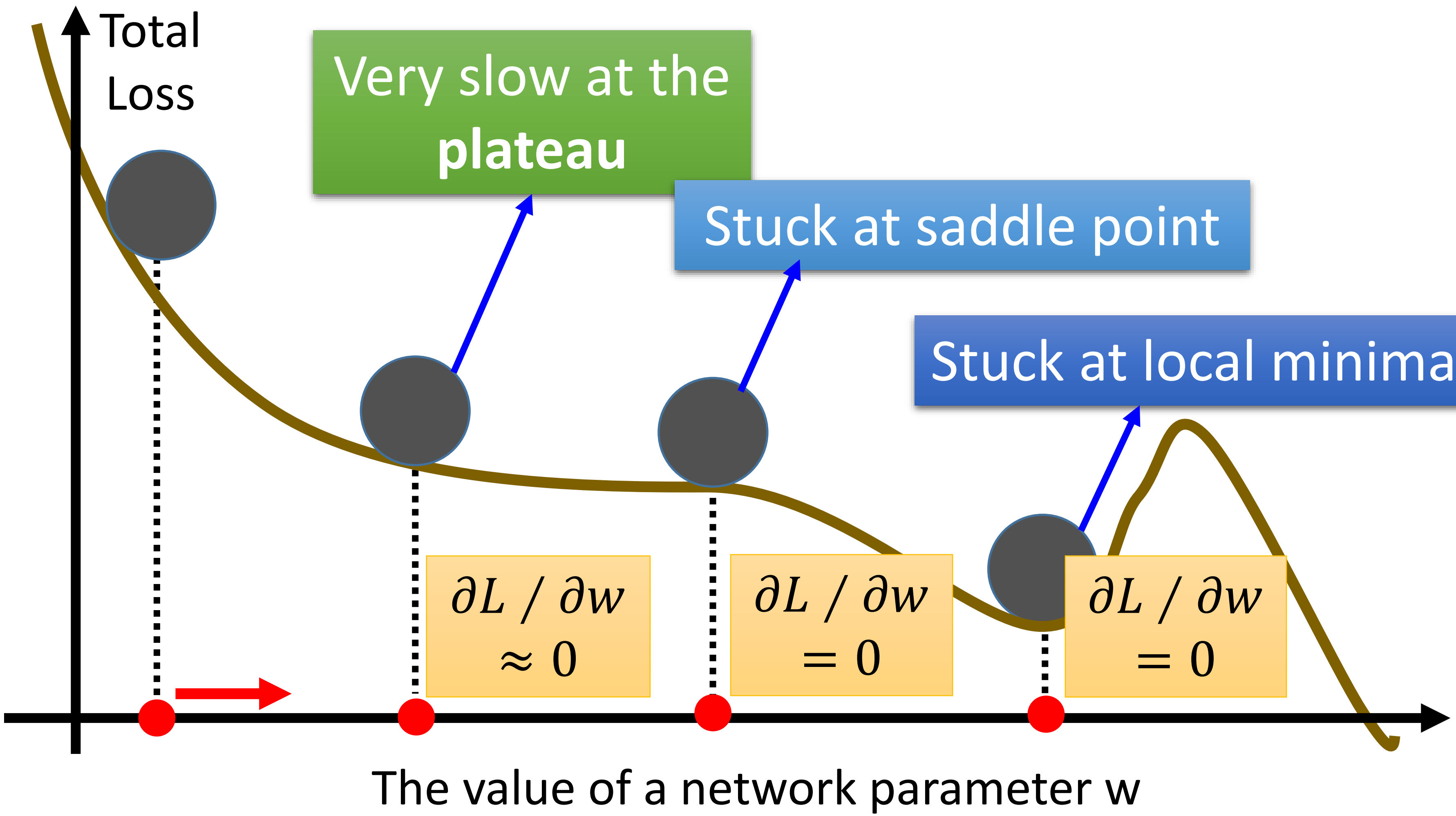
- Adagrad [John Duchi, JMLR'11]
- RMSprop
  - https://www.youtube.com/watch?v=O3sxAc4hxZU
- Adadelta [Matthew D. Zeiler, arXiv'12]
- "No more pesky learning rates" [Tom Schaul, arXiv'12]
- AdaSecant [Caglar Gulcehre, arXiv'14]
- Adam [Diederik P. Kingma, ICLR'15]
- Nadam
  - http://cs229.stanford.edu/proj2015/054_report.pdf

Recipe of Deep Learning

- Choosing proper loss
- Mini-batch
- New activation function
- Adaptive Learning Rate
- Momentum



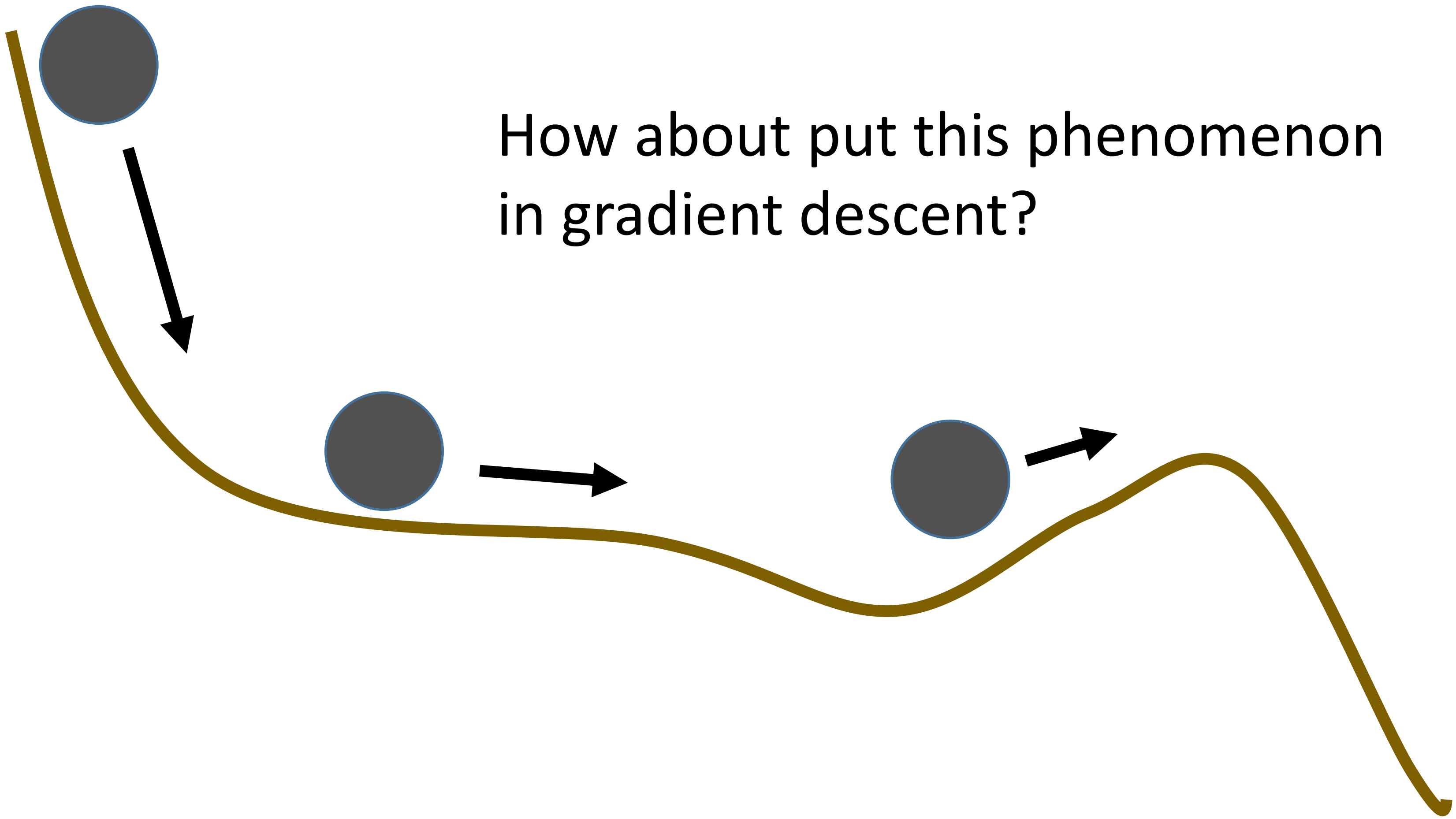
Hard to find optimal network parameters



In physical world ……

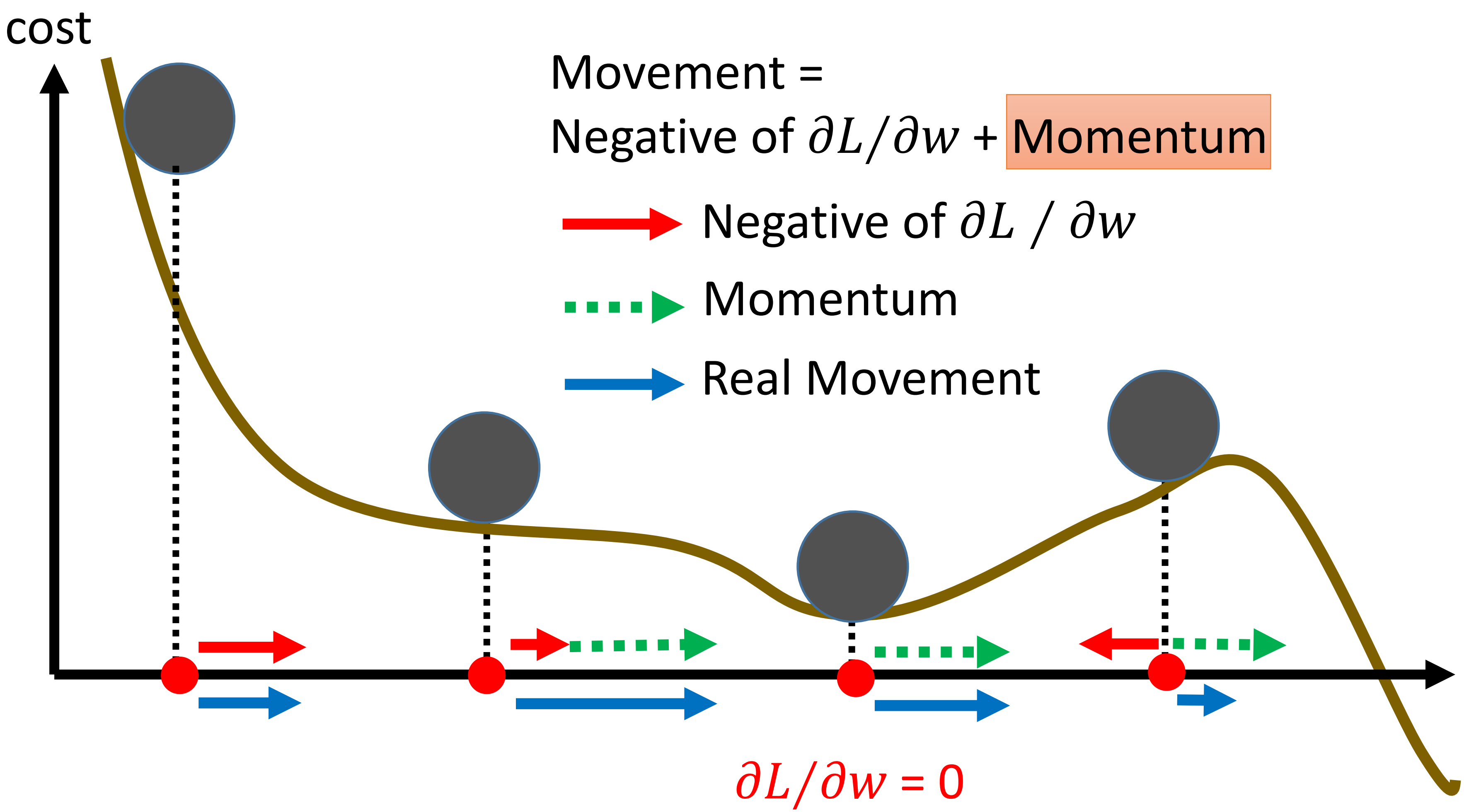
- Momentum

How about put this phenomenon in gradient descent?



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 β_1^t and β_2^t we denote β_1 and β_2 to the power t.

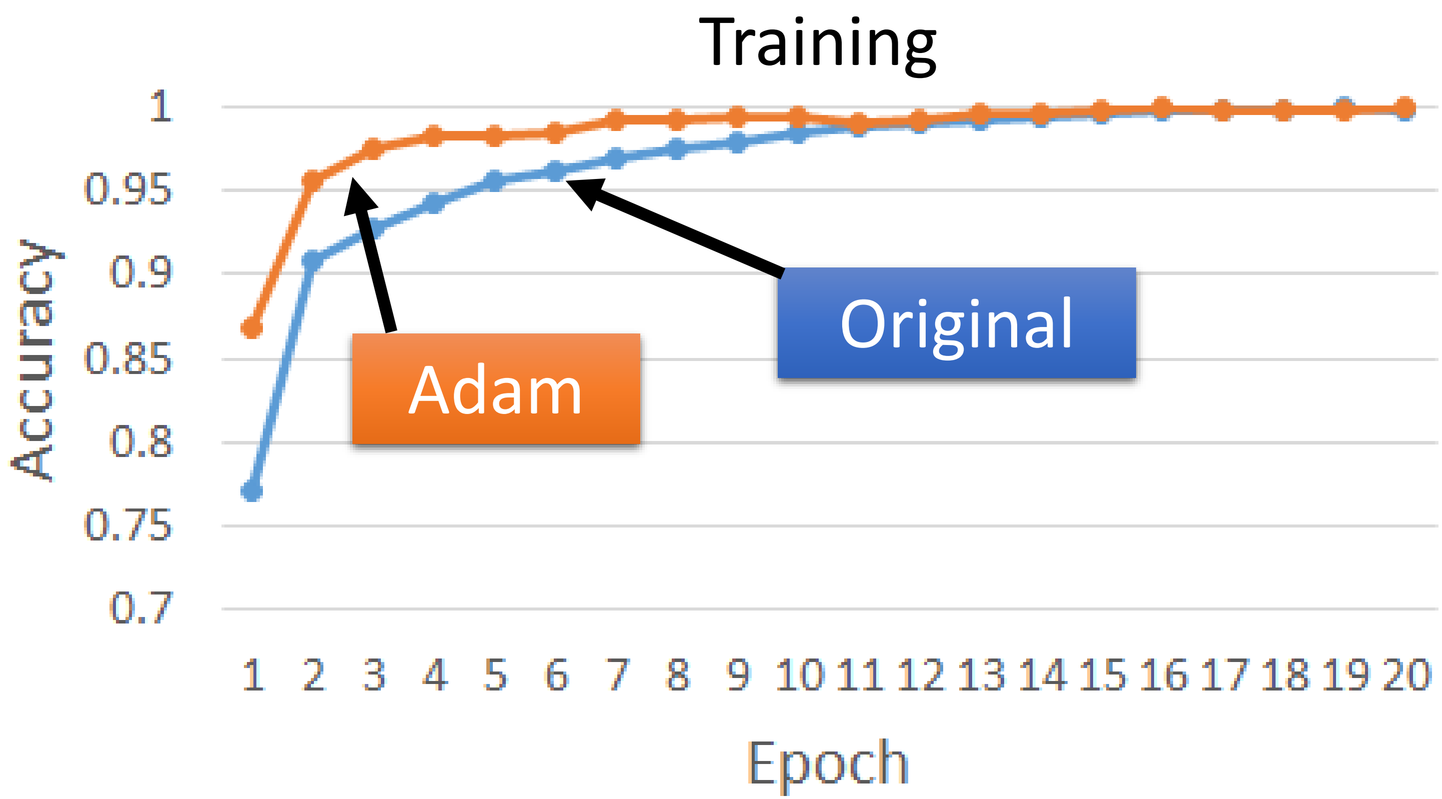
Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_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)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Let's try it

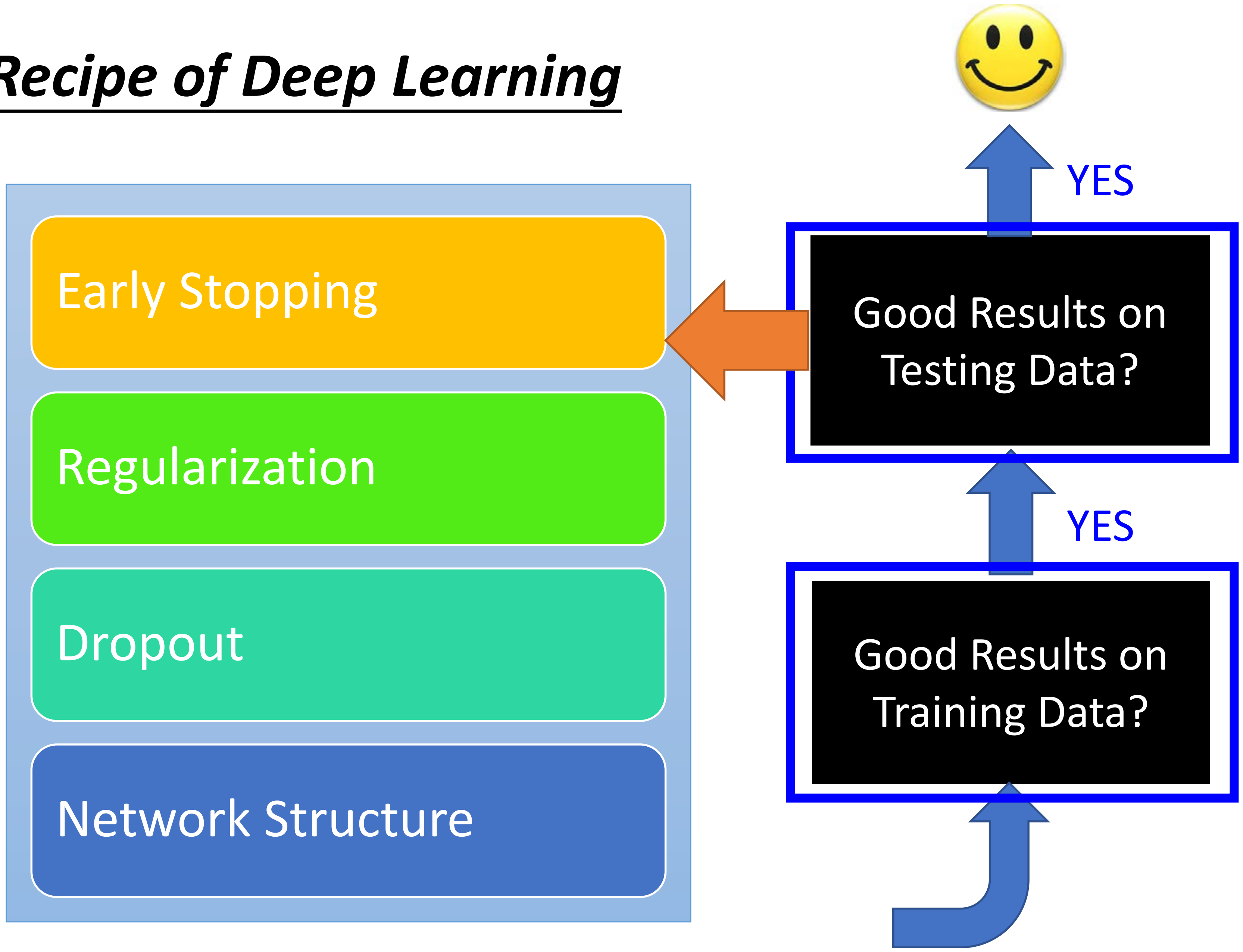
Testing:

	Accuracy
Original	0.96
Adam	0.97

- ReLU, 3 layer



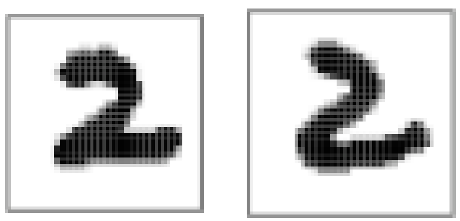
Recipe of Deep Learning



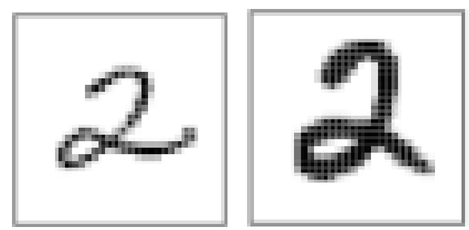
Why Overfitting?

- Training data and testing data can be different.

Training Data:



Testing Data:



Learning target is defined by the training data.

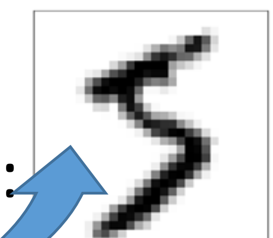
The parameters achieving the learning target do not necessary have good results on the testing data.

Panacea for Overfitting

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

Handwriting recognition:

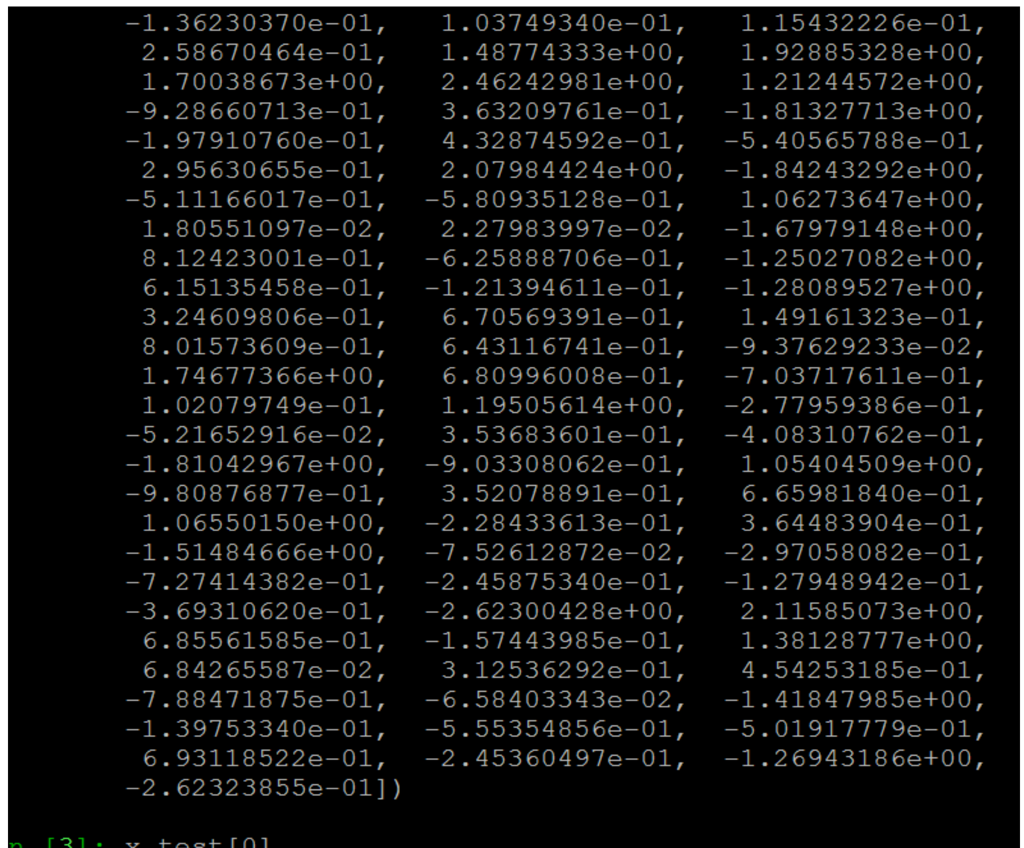
Original Training Data: 

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?

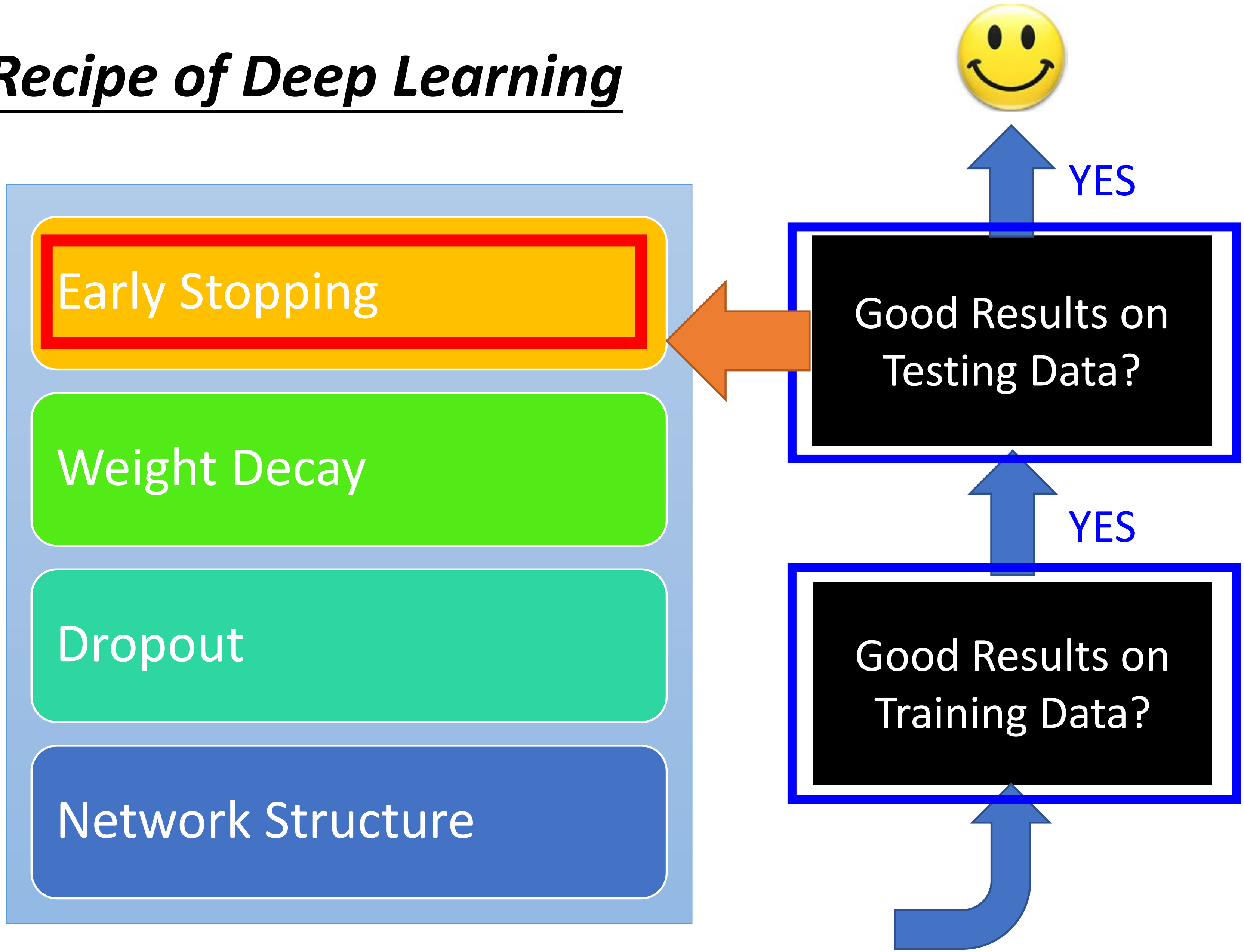
- For experiments, we added some noises to the testing data

Testing:

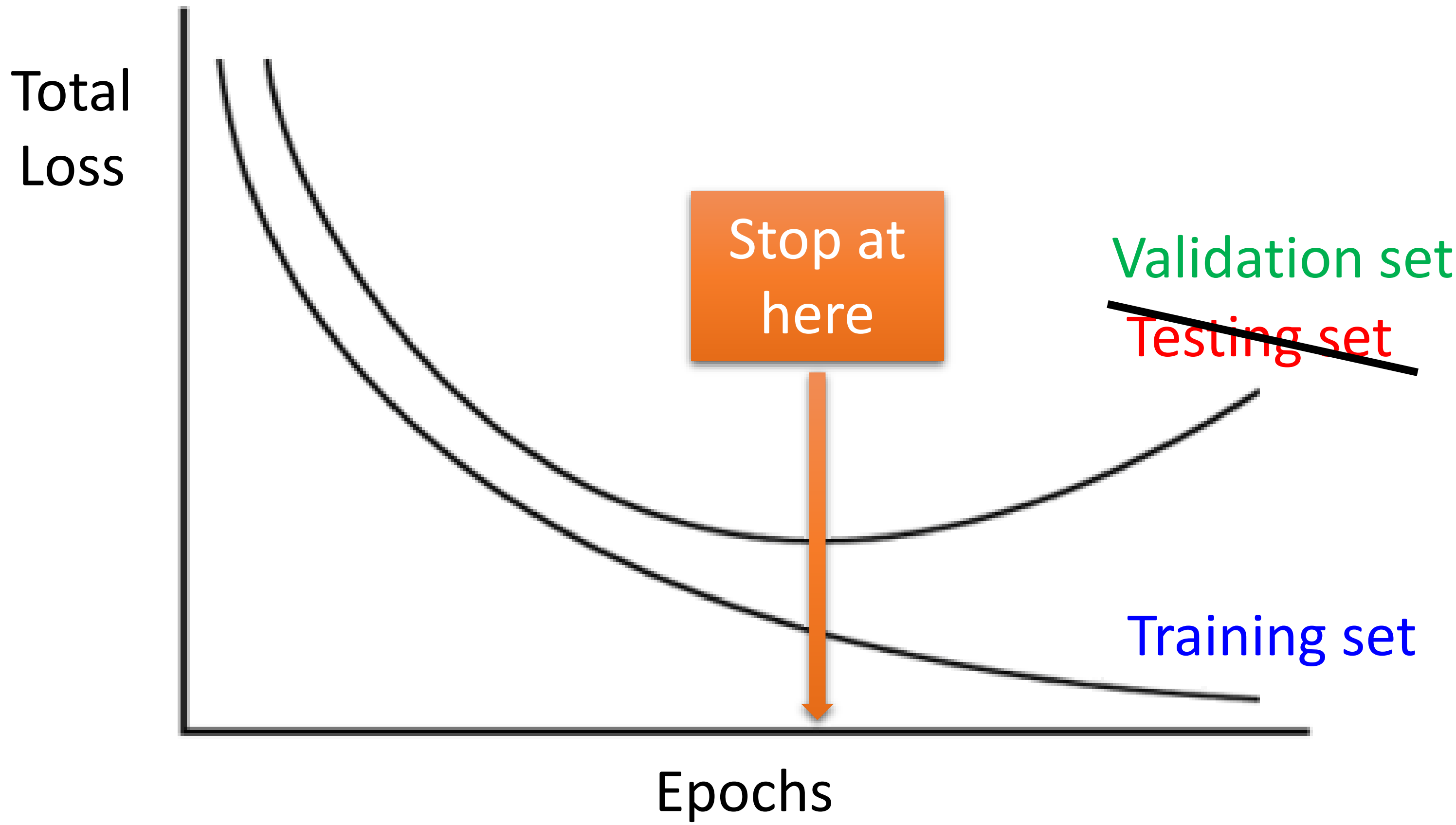
	Accuracy
Clean	0.97
Noisy	0.50

Training is not influenced.

Recipe of Deep Learning

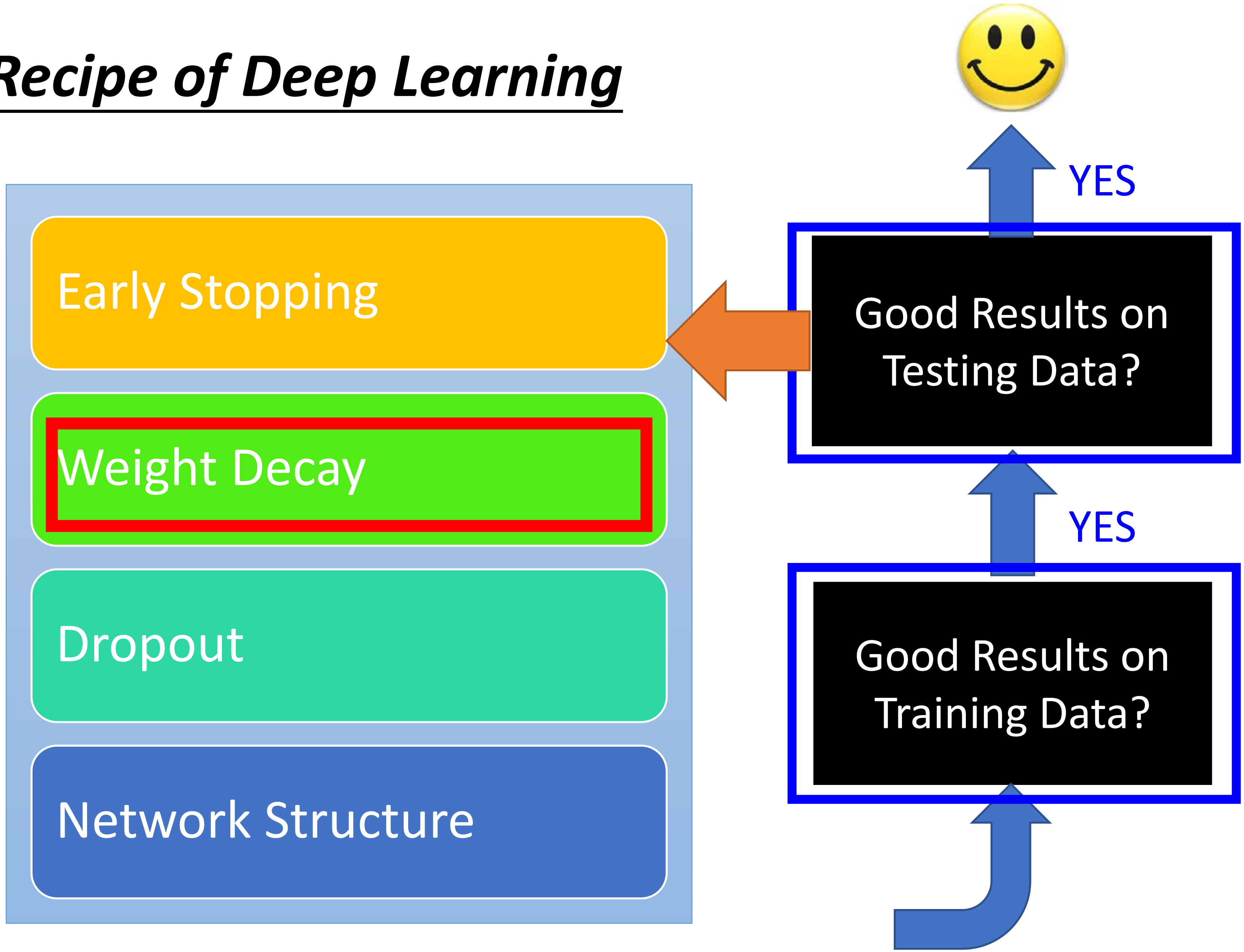


Early Stopping



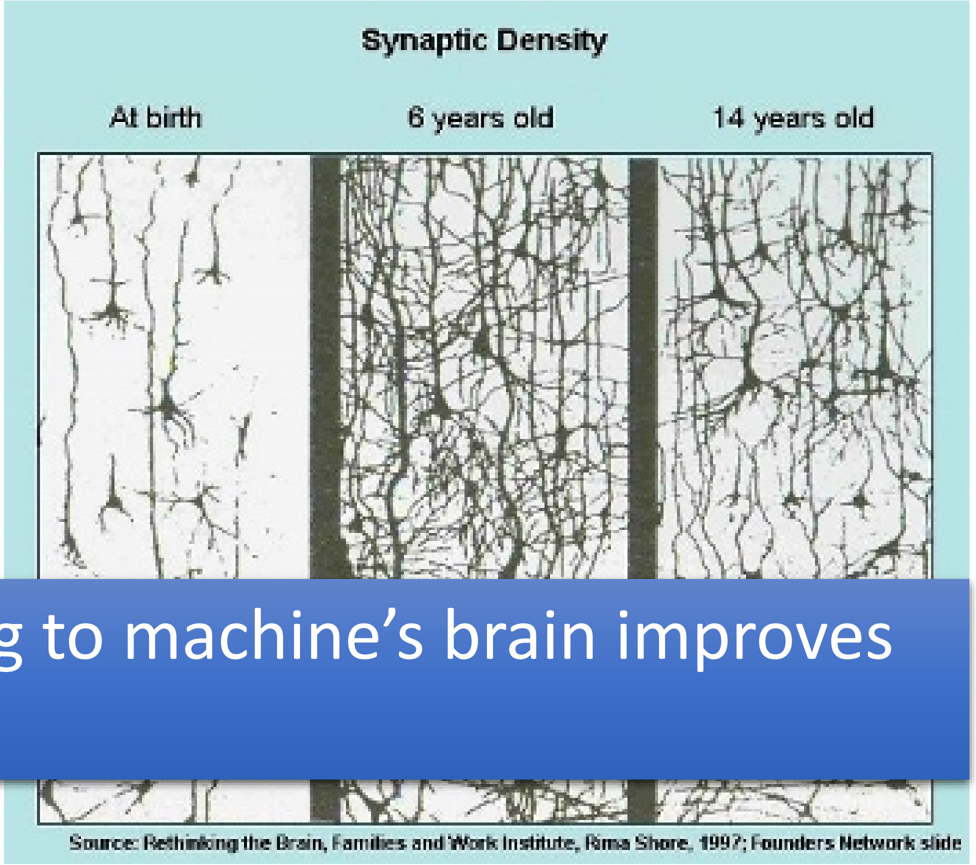
Keras: http://keras.io/getting-started/faq/#how-can-i-interrupt-training-when-the-validation-loss-isnt-decreasing-anymore

Recipe of Deep Learning



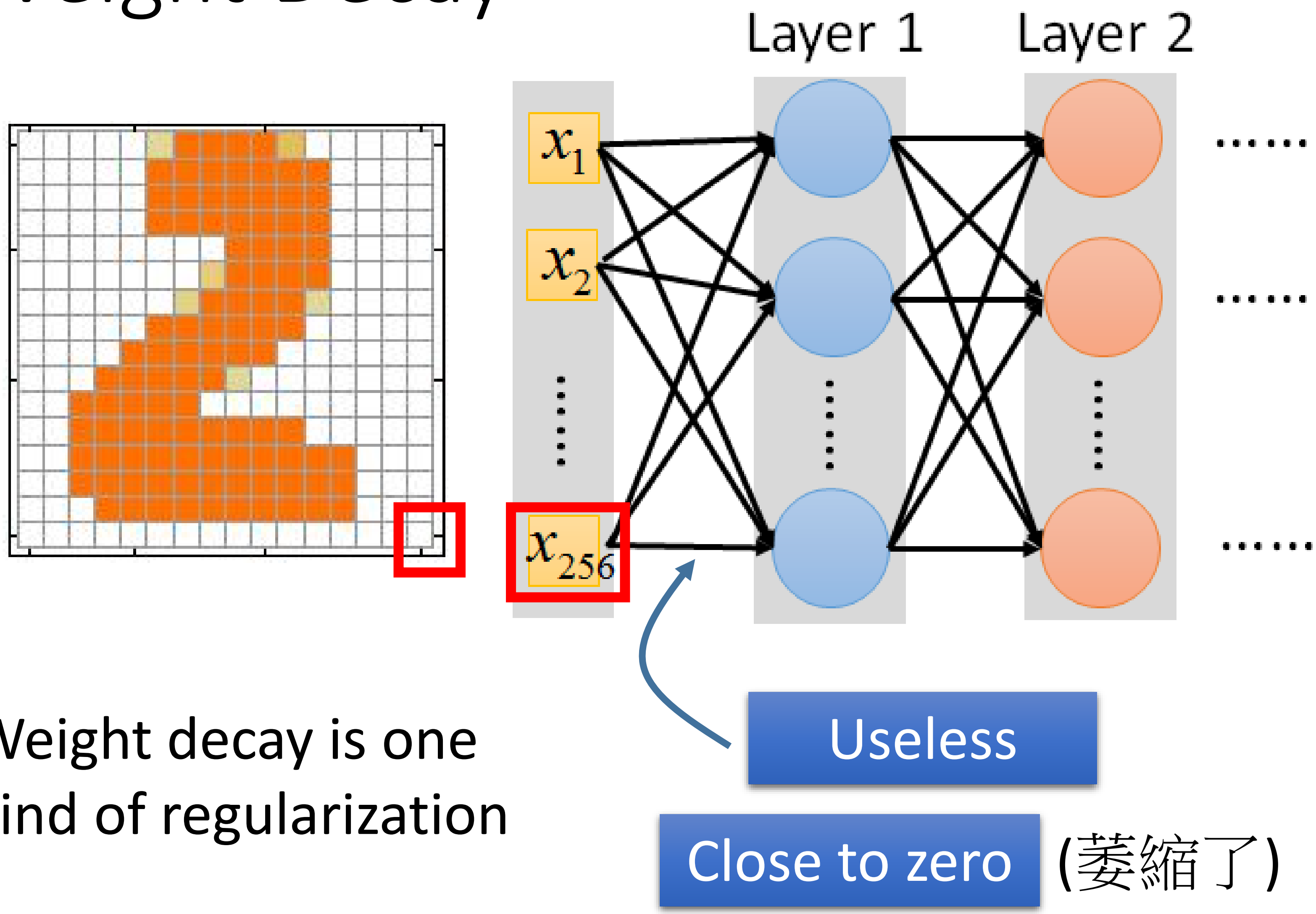
Weight Decay

- Our brain prunes out the useless link between neurons.



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

Weight Decay



Weight decay is one kind of regularization

Weight Decay

- Implementation

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

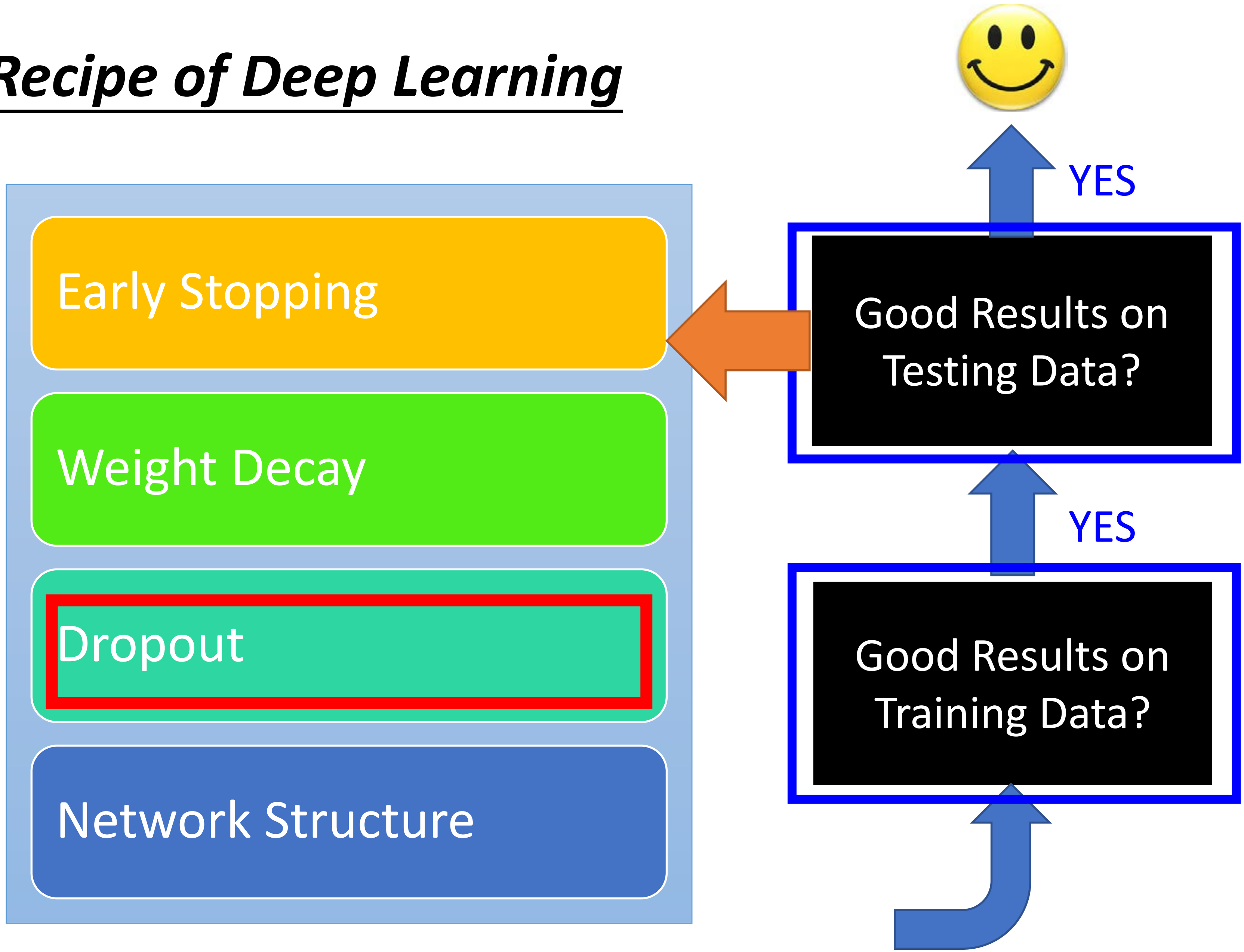
$$\lambda = 0.01$$

Weight Decay: $w \leftarrow \underline{\boxed{0.99}}\, w - \eta \dfrac{\partial L}{\partial w}$

Smaller and smaller

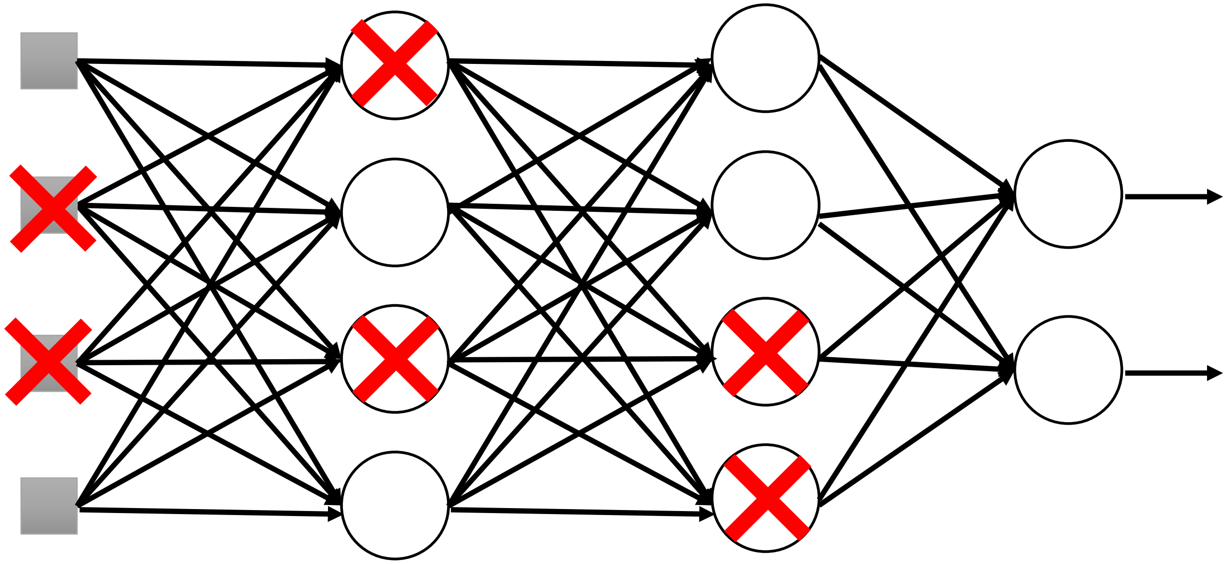
Keras: http://keras.io/regularizers/

Recipe of Deep Learning



Dropout

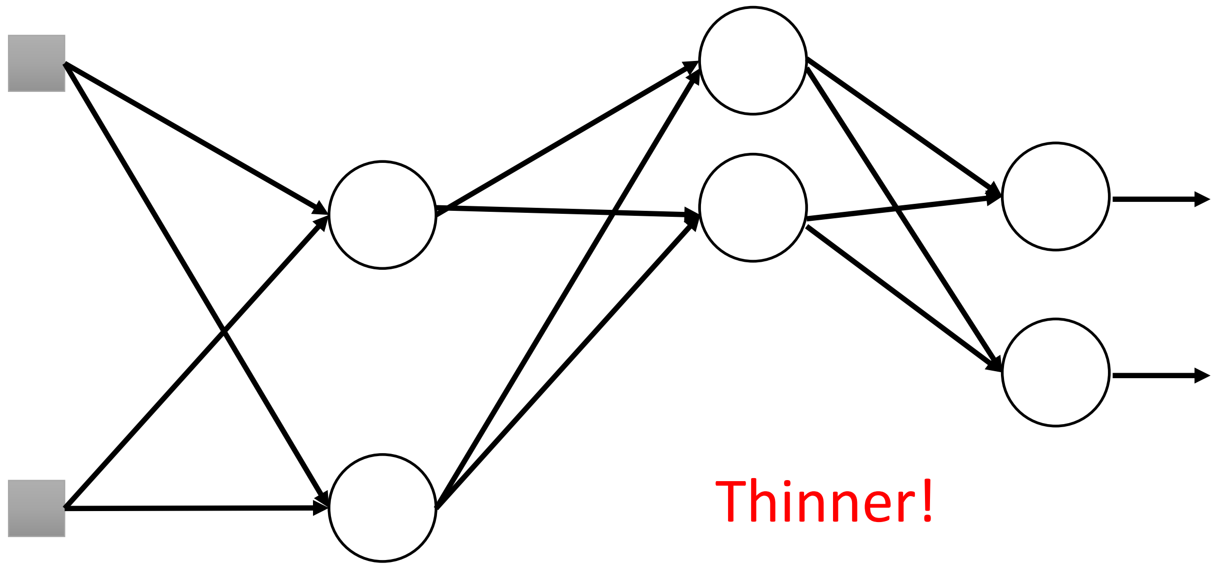
Training:



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

Dropout

Training:

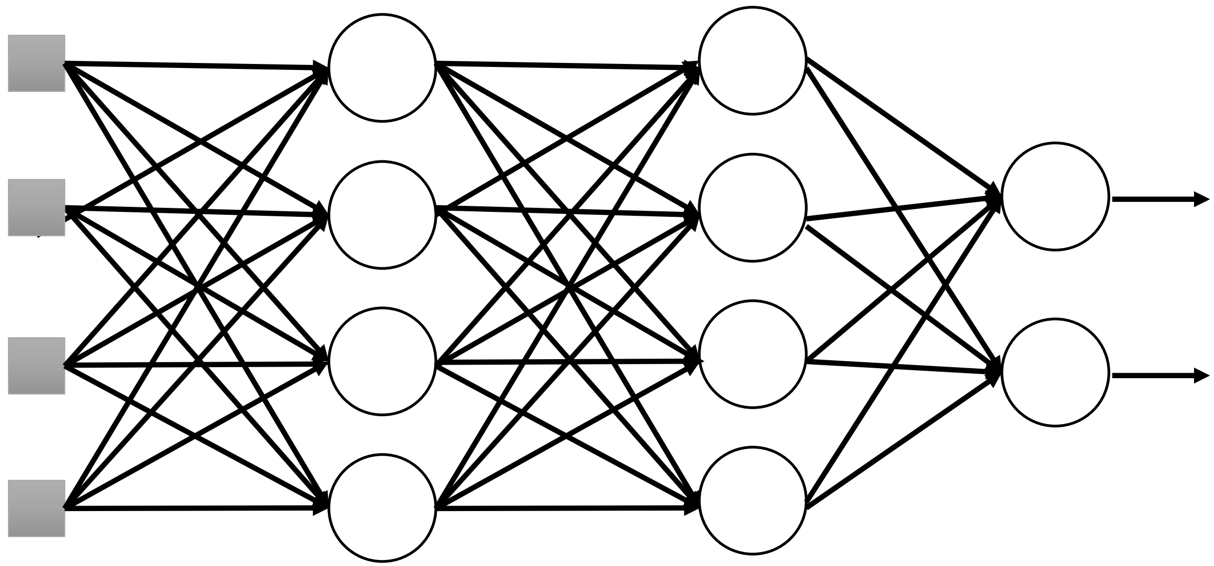


- **Each time before updating the parameters**
  - Each neuron has p% to dropout
    - **The structure of the network is changed.**
  - Using the new network for training

For each mini-batch, we resample the dropout neurons

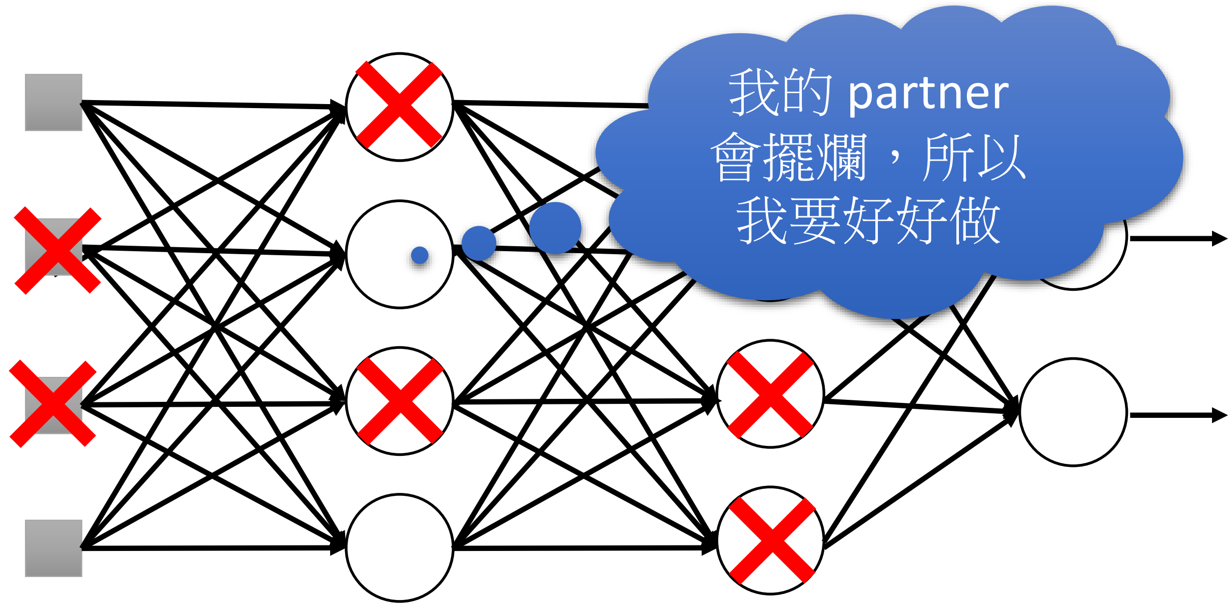
Dropout

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



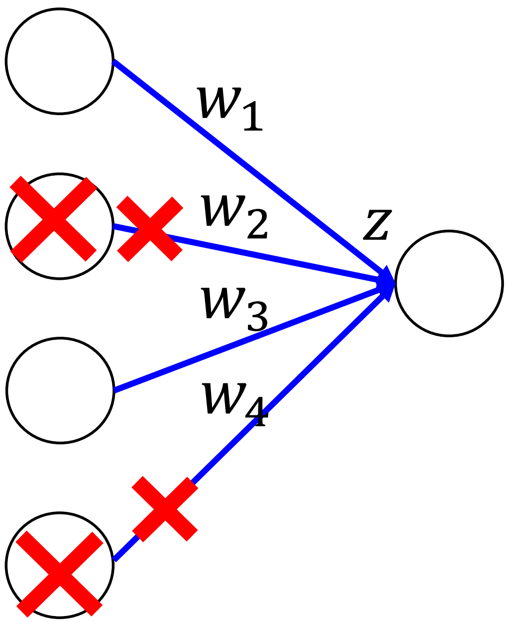
- When teams up, if everyone expect the partner will do the work, nothing will be done finally.
- However, if you know your partner will dropout, you will do better.
- When testing, no one dropout actually, so obtaining good results eventually.

Dropout - Intuitive Reason

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

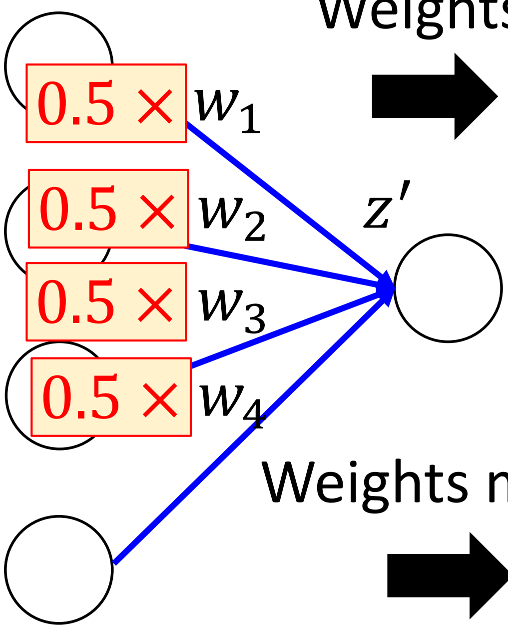
<u>Training of Dropout</u>

Assume dropout rate is 50%



<u>Testing of Dropout</u>

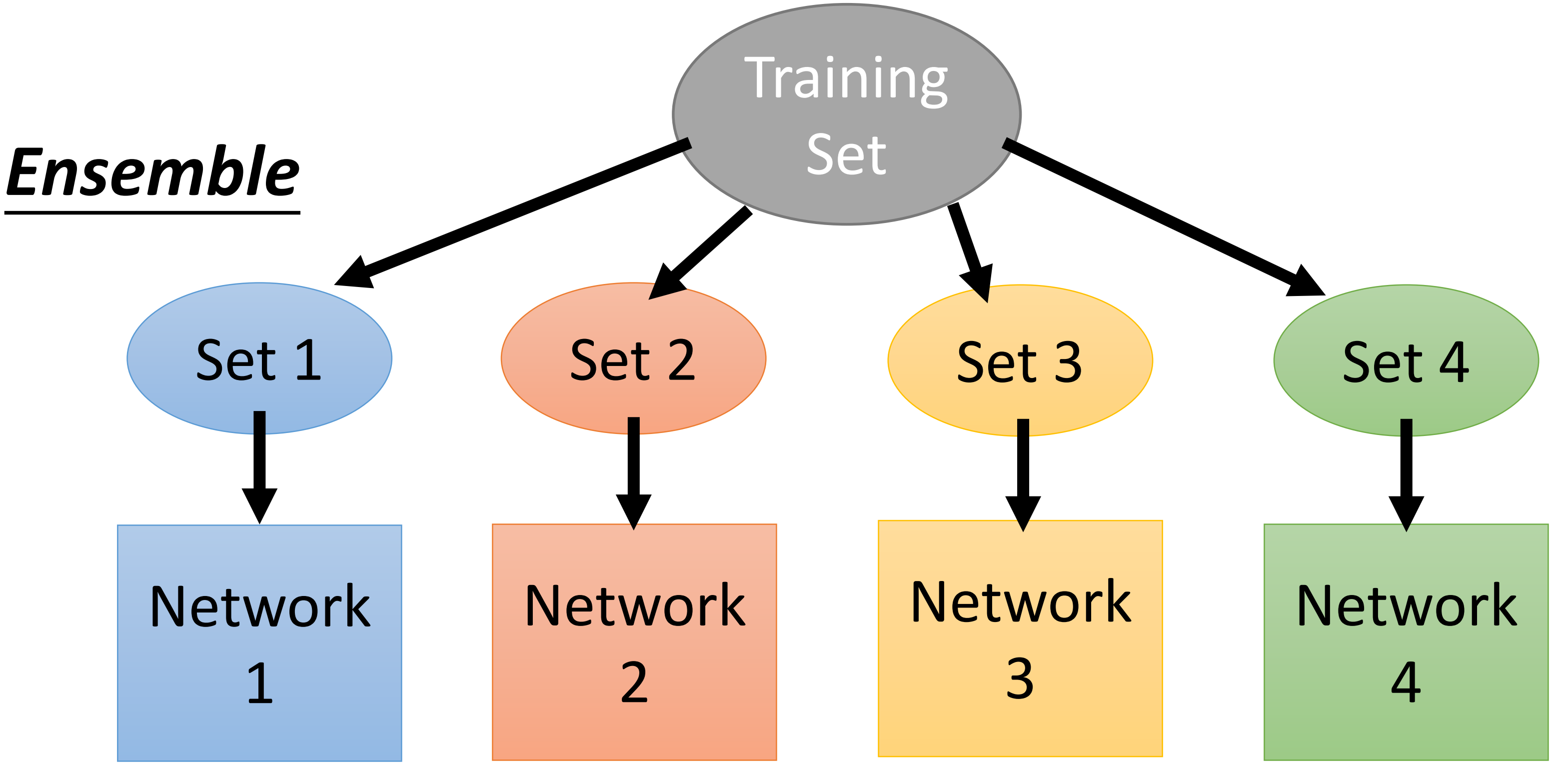
No dropout



Weights from training $\Rightarrow z' \approx 2z$

Weights multiply (1-p)% $\Rightarrow z' \approx z$

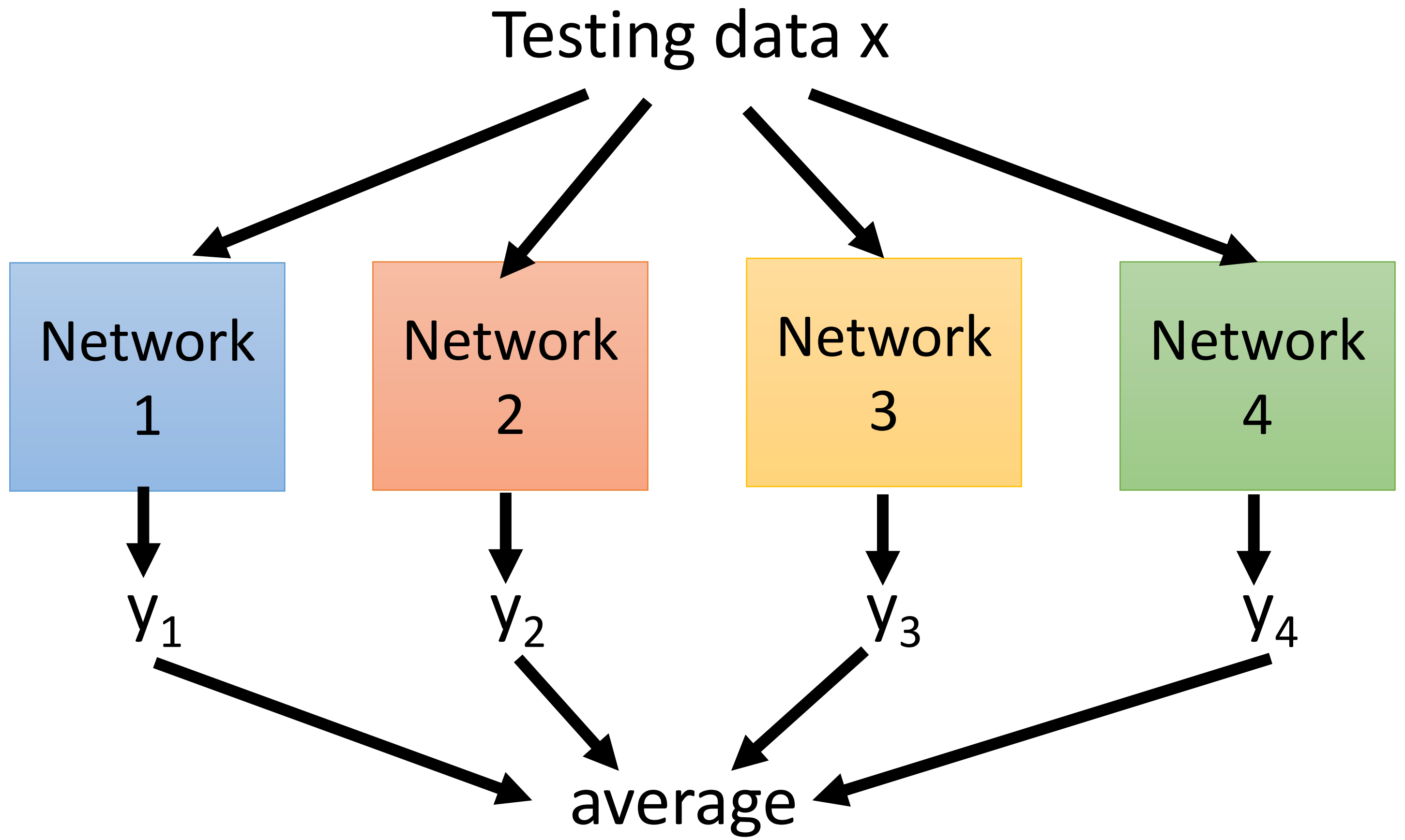
Dropout is a kind of ensemble.



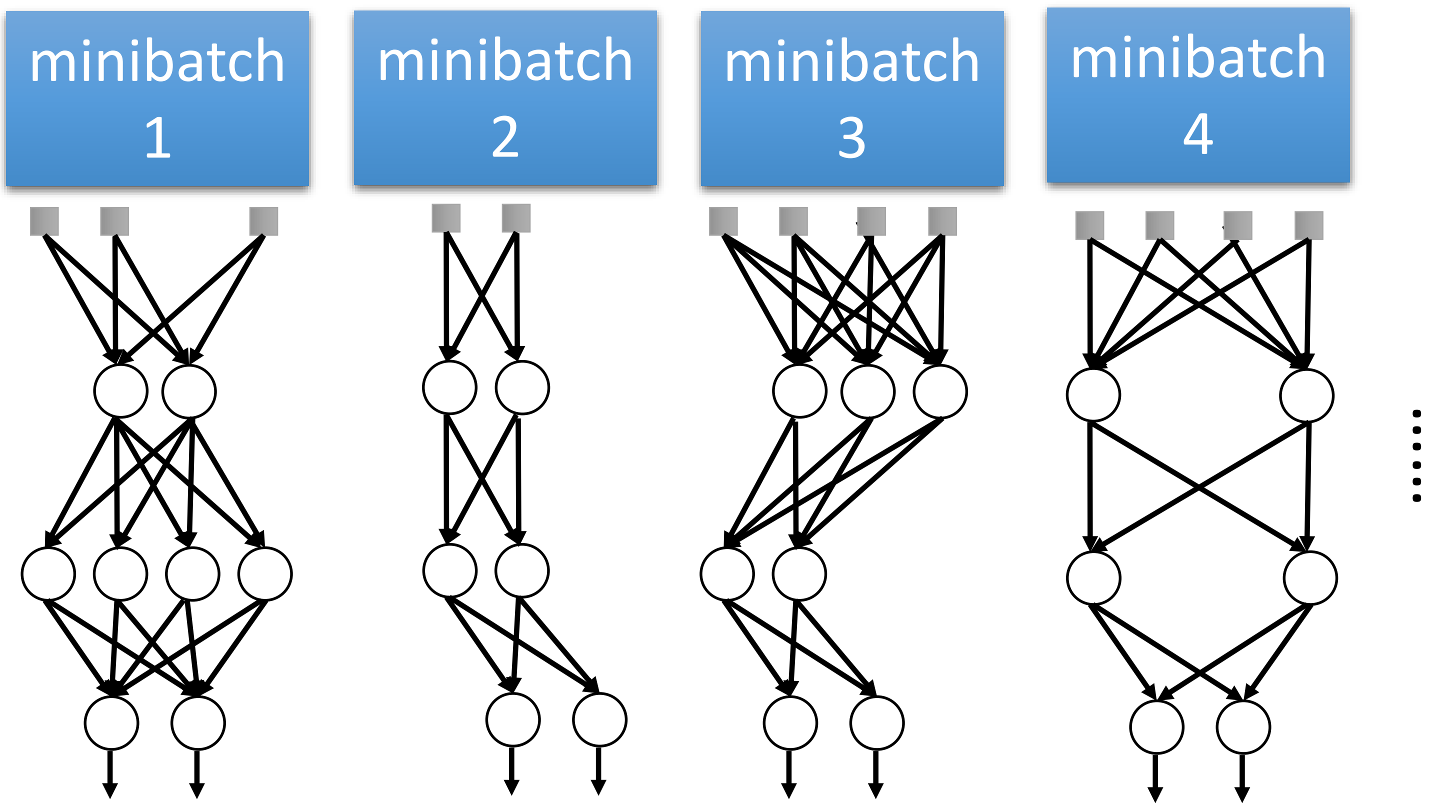
Train a bunch of networks with different structures

Dropout is a kind of ensemble.

Ensemble



Dropout is a kind of ensemble.



<u>Training of Dropout</u>

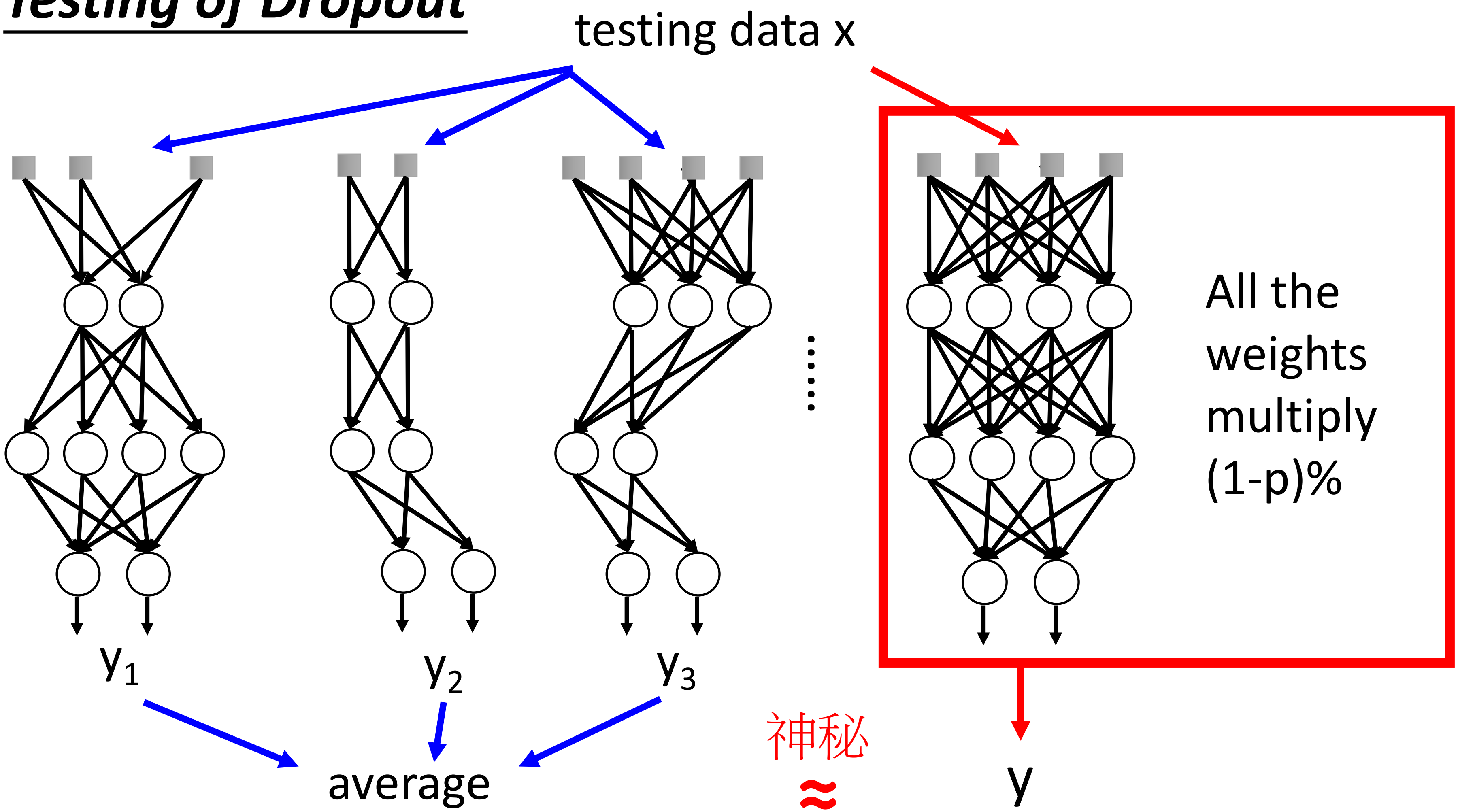
M neurons

2^M possible networks

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

Dropout is a kind of ensemble.

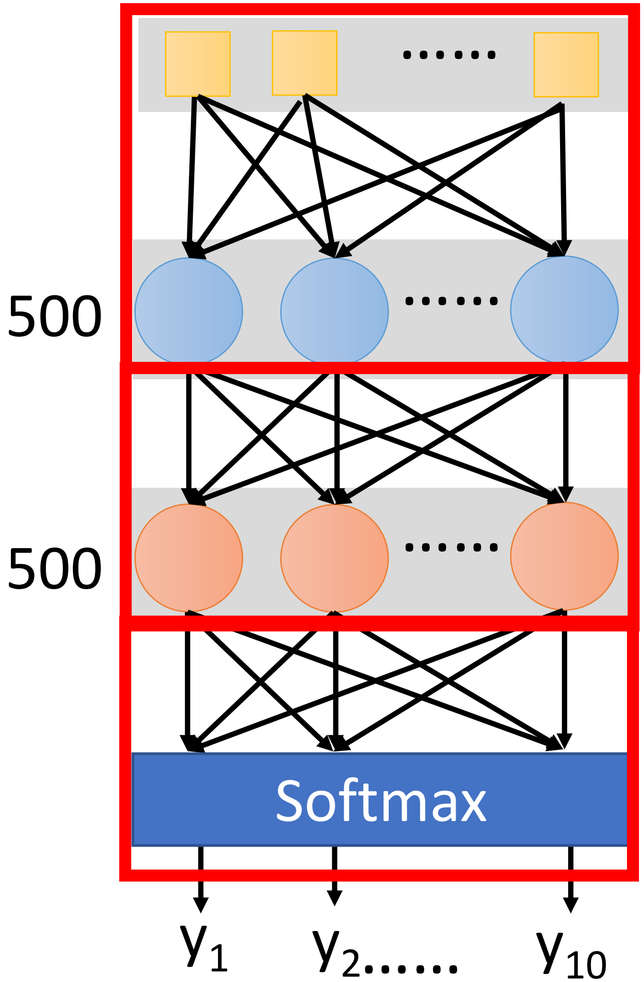
Testing of Dropout



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') )
```

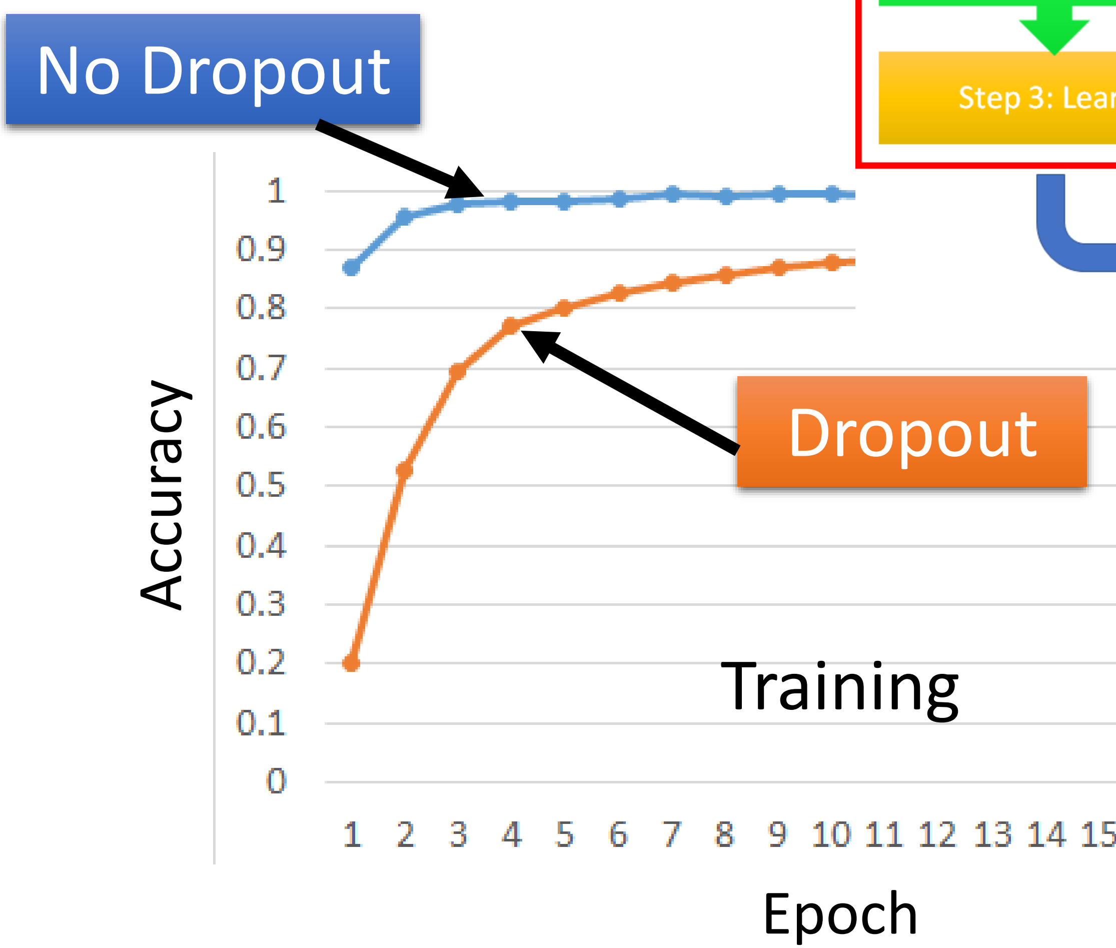
model.add(dropout(0.8))

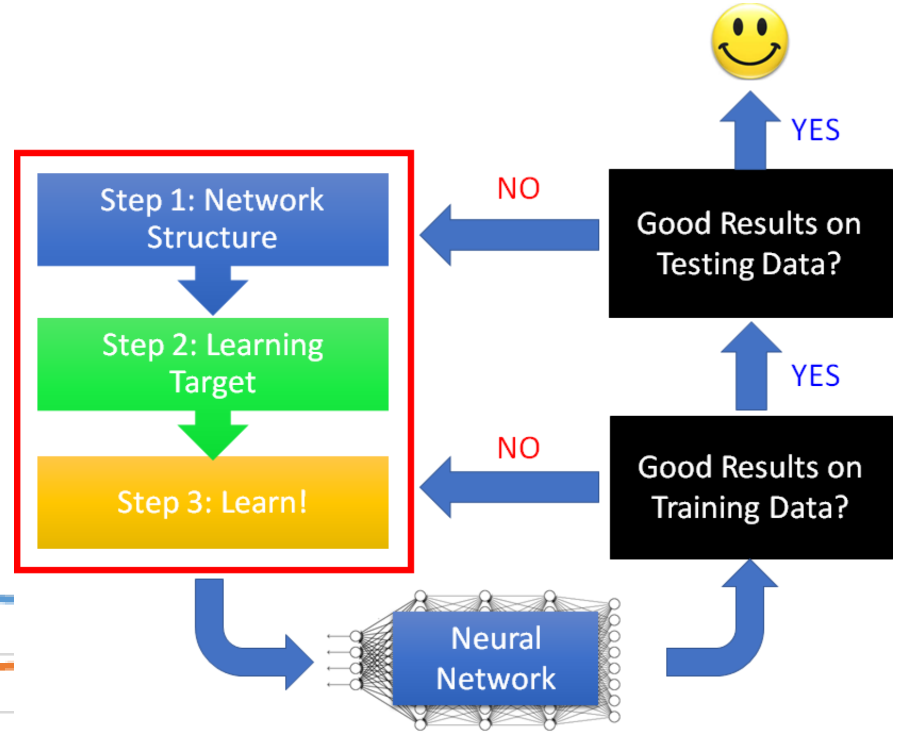
```
model.add( Dense( output_dim=500 ) )
model.add( Activation('sigmoid') )
```

model.add(dropout(0.8))

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

Let's try it





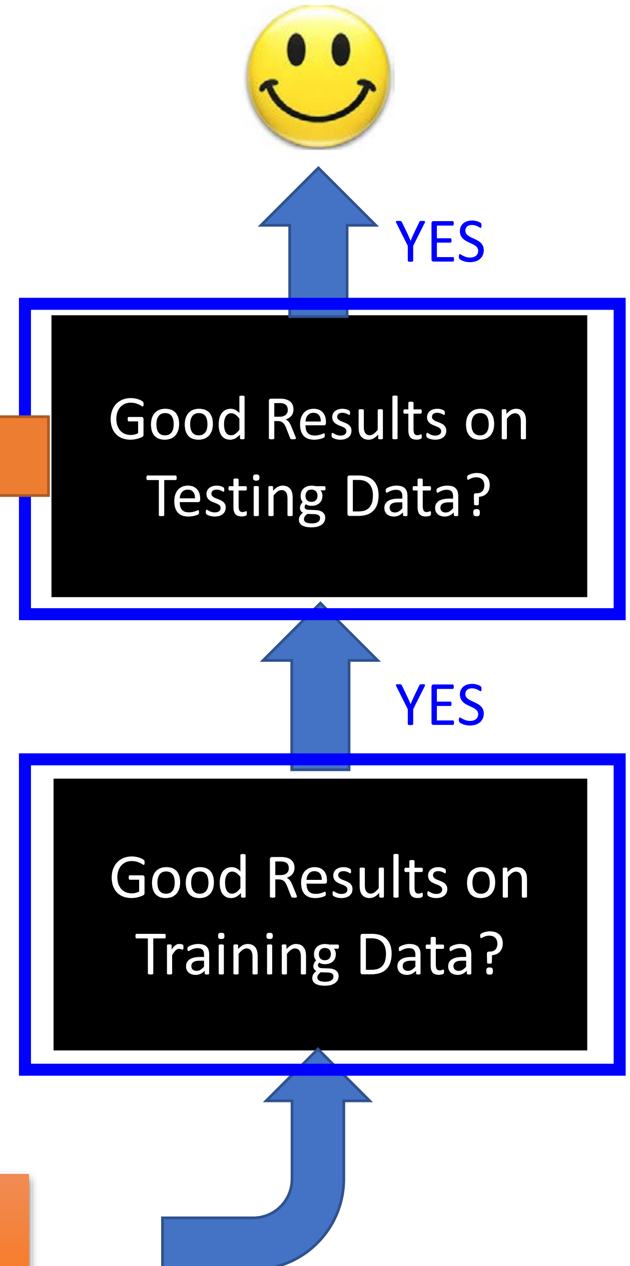
Testing:

	Accuracy
Noisy	0.50
+ dropout	0.63

Recipe of Deep Learning

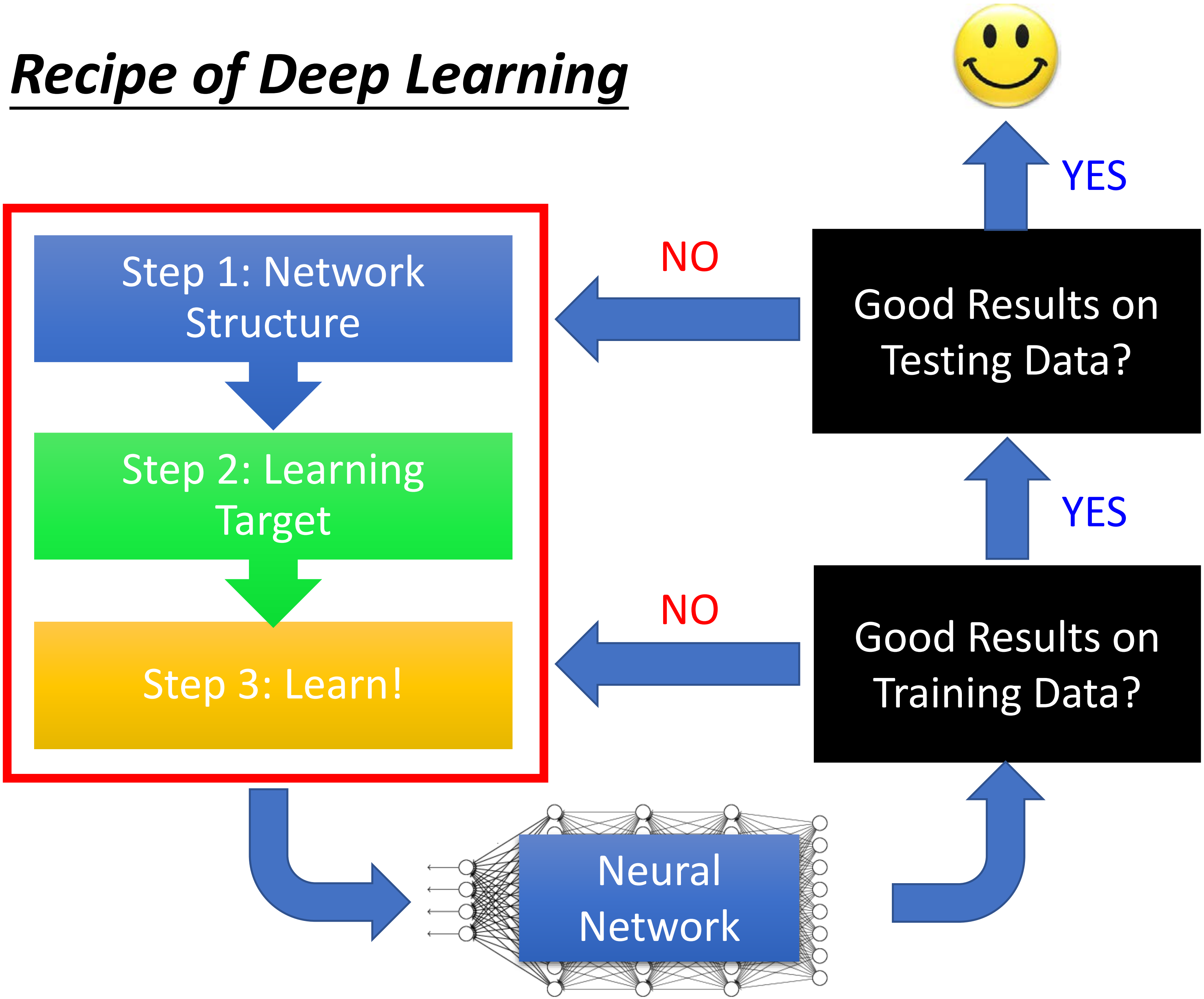
- Early Stopping
- Regularization
- Dropout
- Network Structure

CNN is a very good example!
(next lecture)



Concluding Remarks of Lecture II

Recipe of Deep Learning



Lecture III: Variants of Neural Networks

Variants of Neural Networks

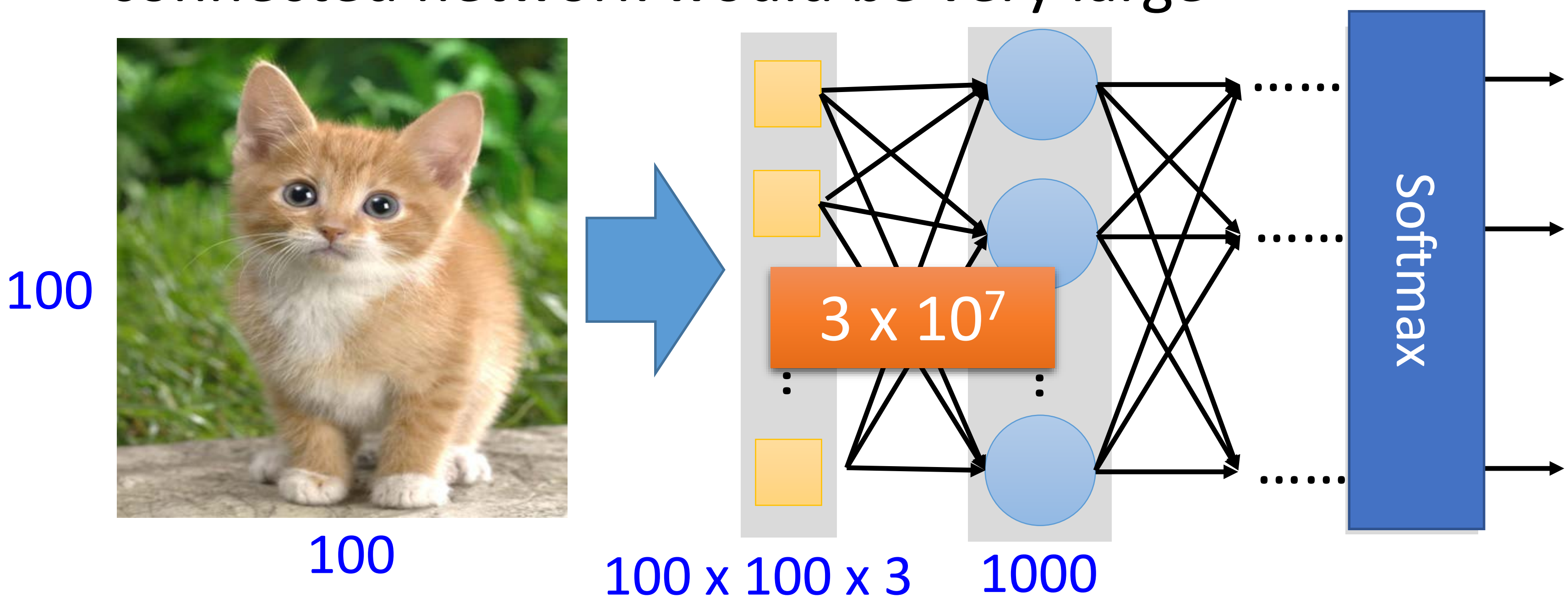
Convolutional Neural Network (CNN)

Consdiering the property of images

Recurrent Neural Network (RNN)

Why CNN for Image?

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



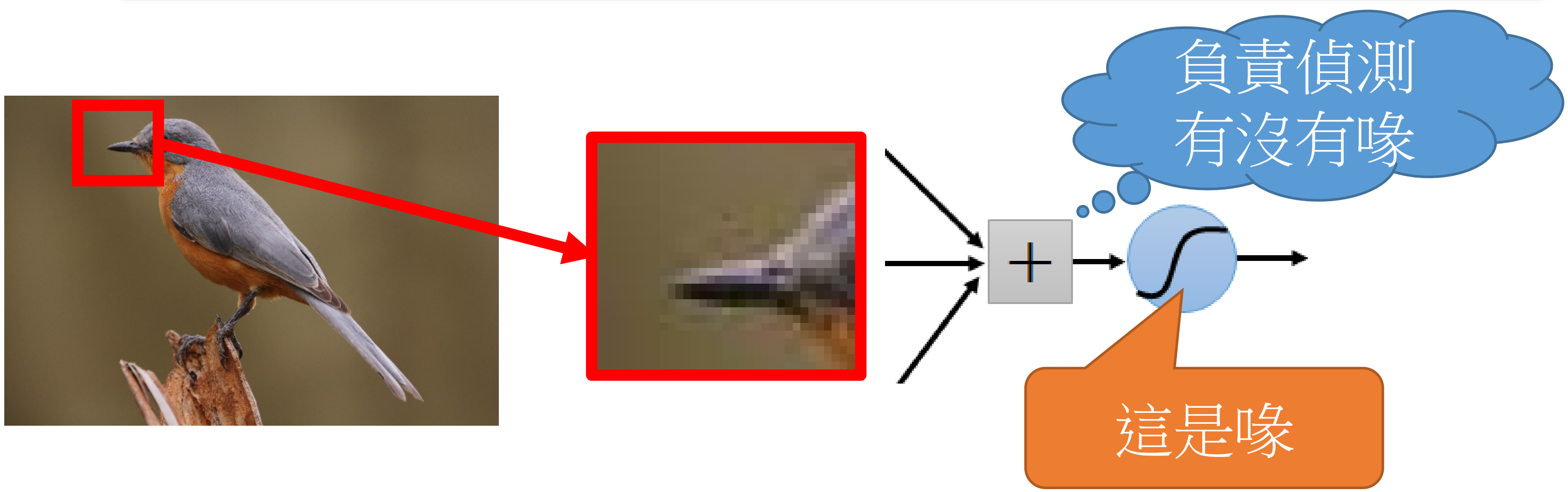
Can the fully connected network be simplified by considering the properties of image recognition?

Why CNN for Image

- Some patterns are much smaller than the whole image

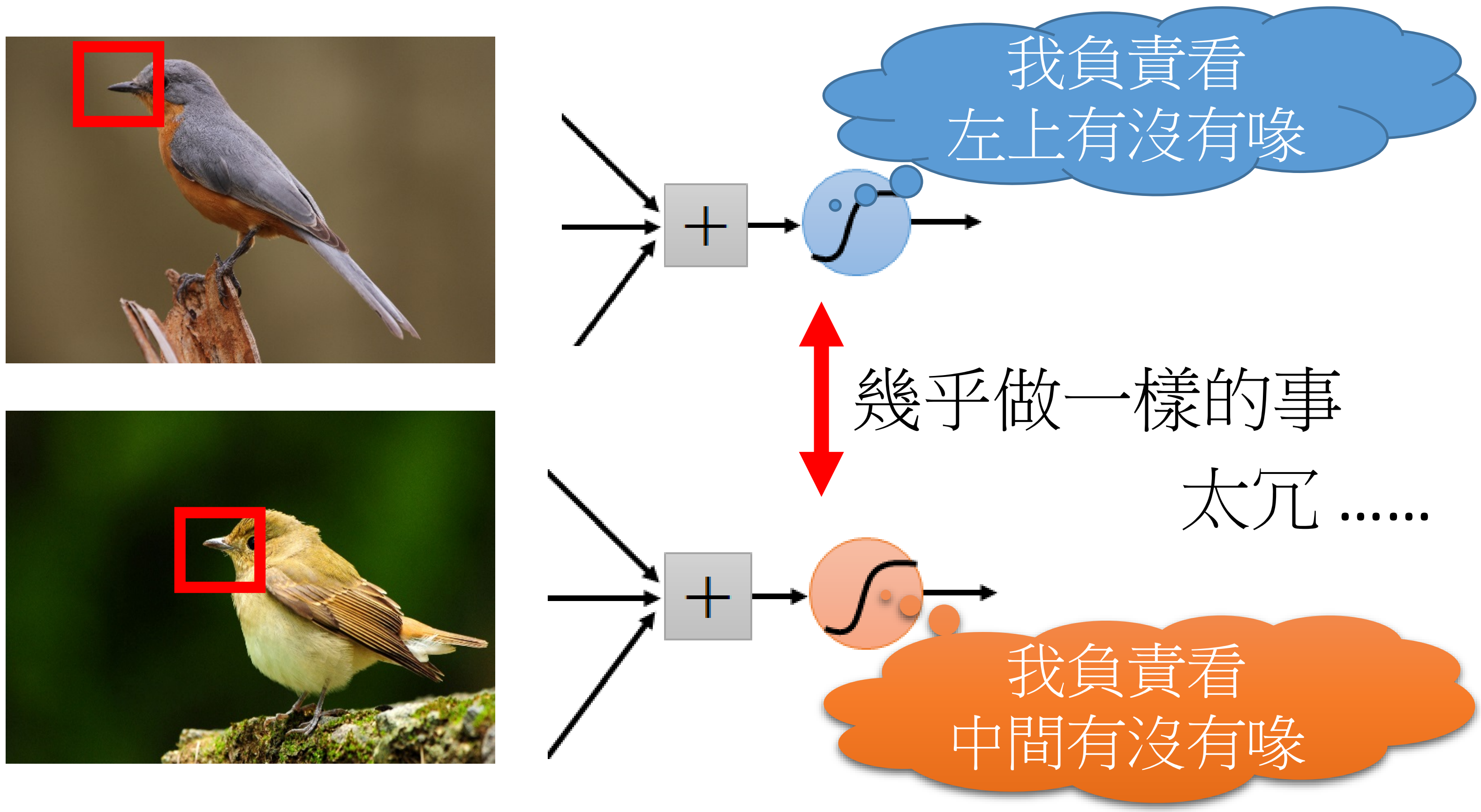
A neuron does not have to see the whole image to discover the pattern.

Connecting to small region with less parameters



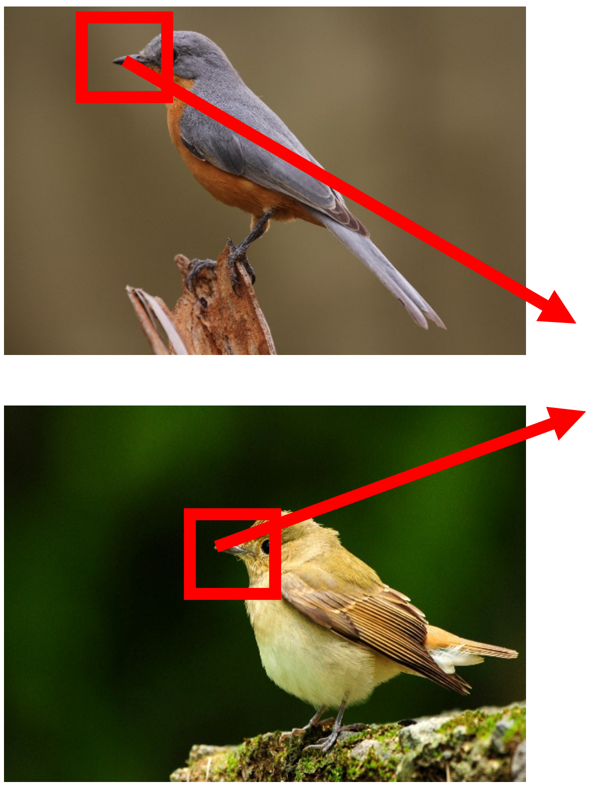
Why CNN for Image

- The same patterns appear in different regions.



Why CNN for Image

- The same patterns appear in different regions.



一個 neuron 就夠了 ......

Why CNN for Image

- Subsampling the pixels will not change the object

除非太過頭

bird

subsampling

bird

We can subsample the pixels to make image smaller

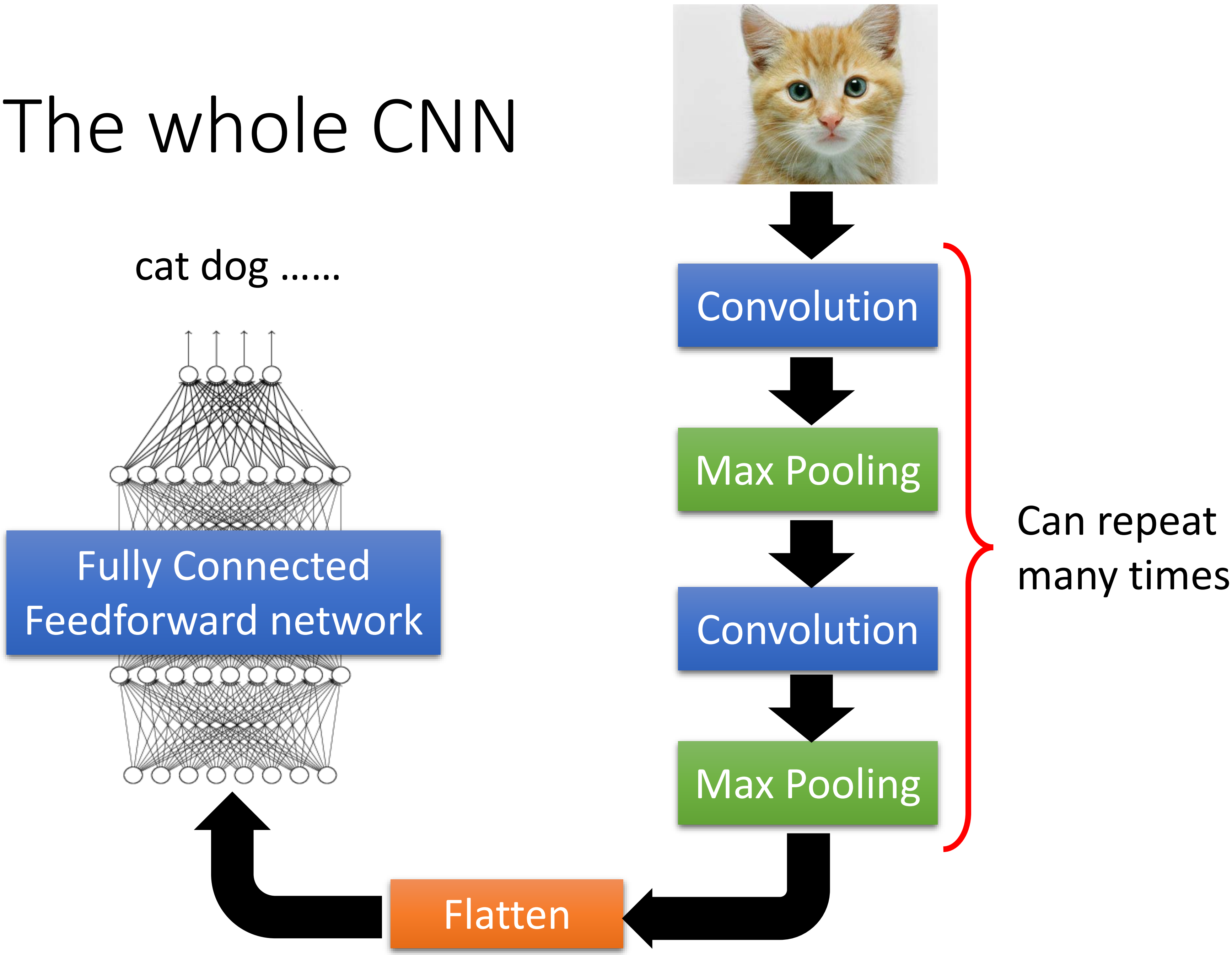
Less parameters for the network to process the image

Convolutional Neural Network

Step 1: Neural Network → Step 2: Learning Target → Step 3: Learn!

天生的腦

The whole CNN



The whole CNN

Property 1

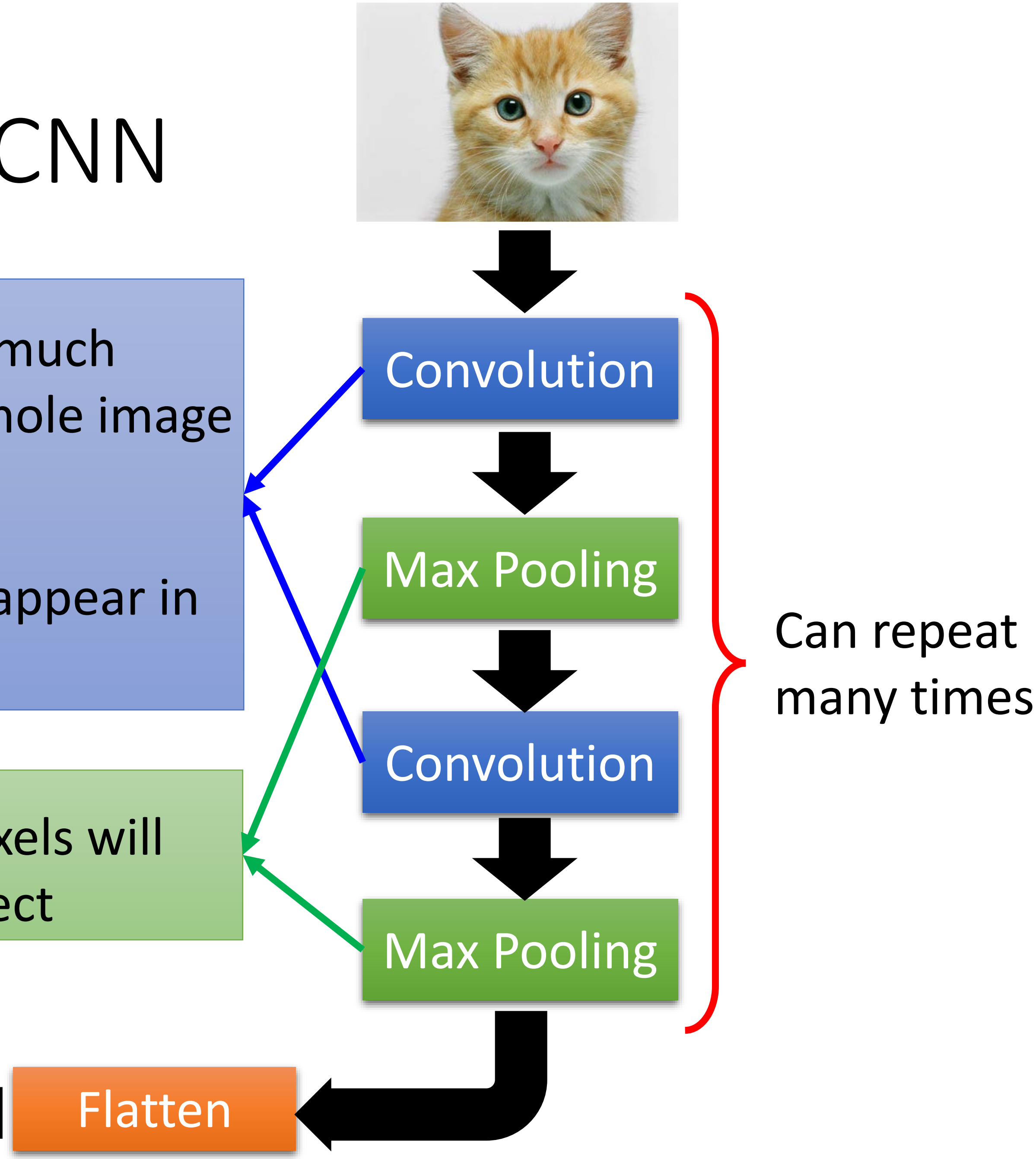
- Some patterns are much smaller than the whole image

Property 2

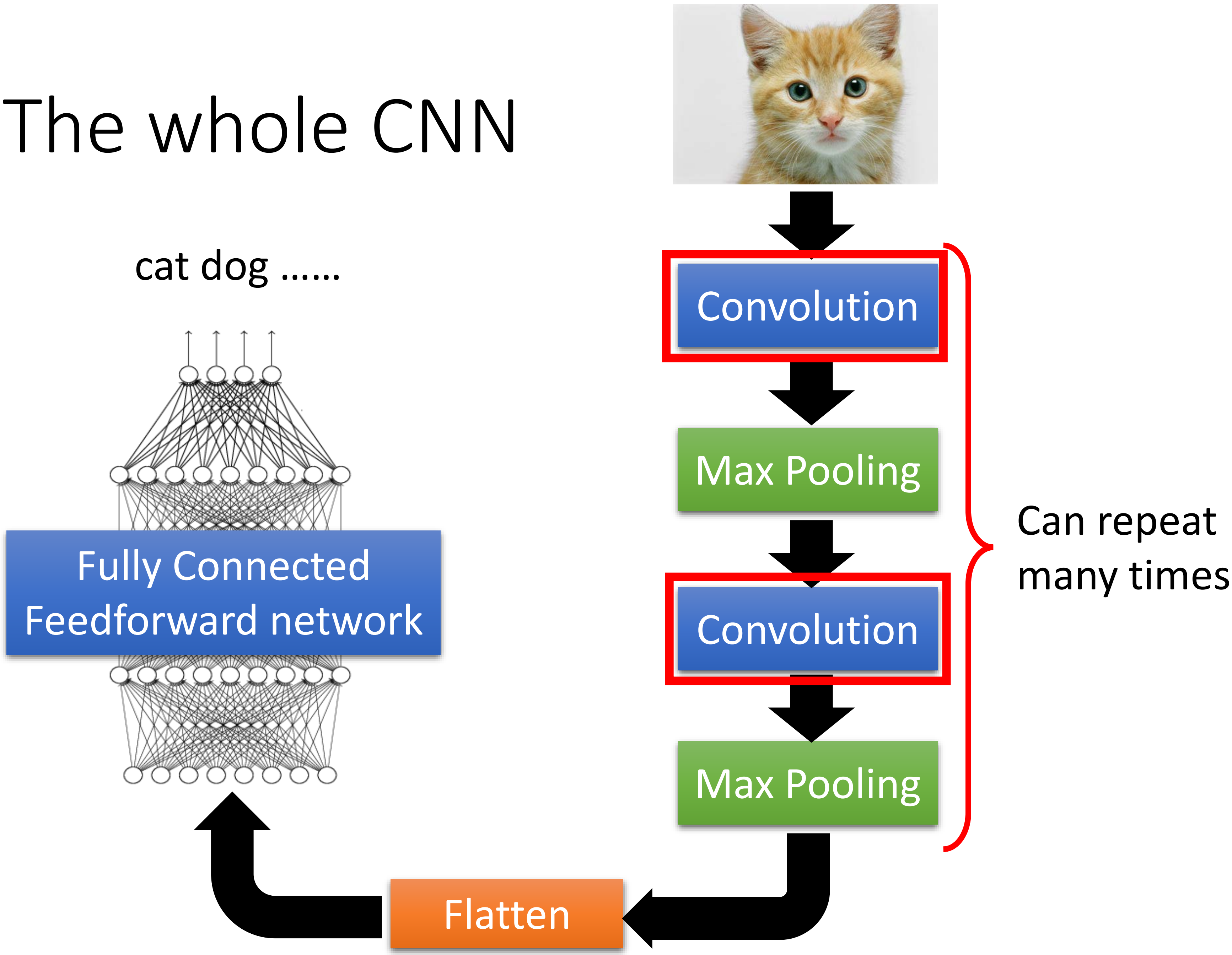
- The same patterns appear in different regions.

Property 3

- Subsampling the pixels will not change the object



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

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

Filter 1

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

3 -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

6 x 6 image

3 -3

We set stride=1 below

CNN – Convolution

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

Filter 1

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

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

Property 2

CNN – Convolution

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

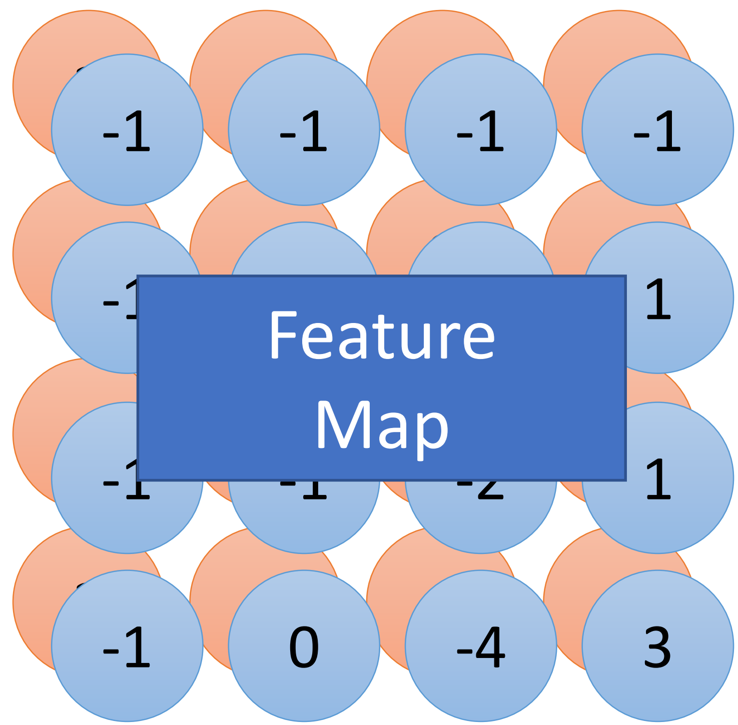
Filter 2

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

Do the same process for every filter



4 x 4 image

CNN – Zero Padding

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

Filter 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

You will get another 6 x 6 images in this way

Zero padding

CNN – Colorful image

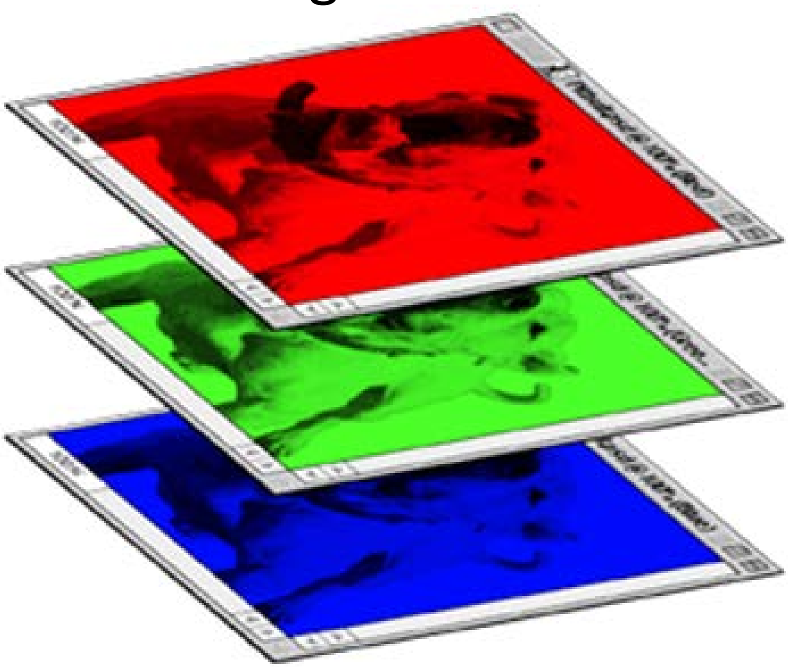
Filter 1

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

Filter 2

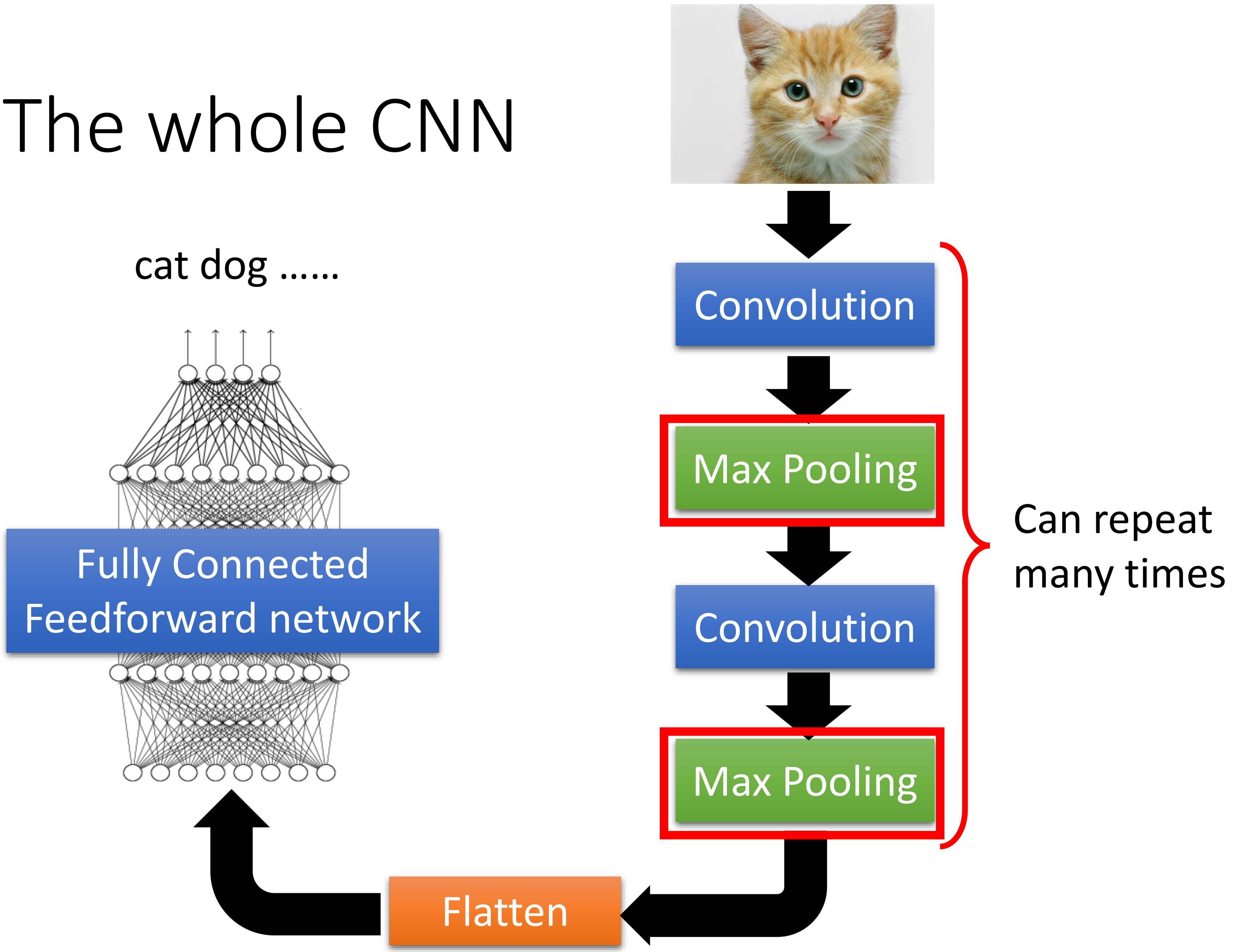
-1	1	-1
-1	1	-1
-1	1	-1

Colorful image



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

The whole CNN



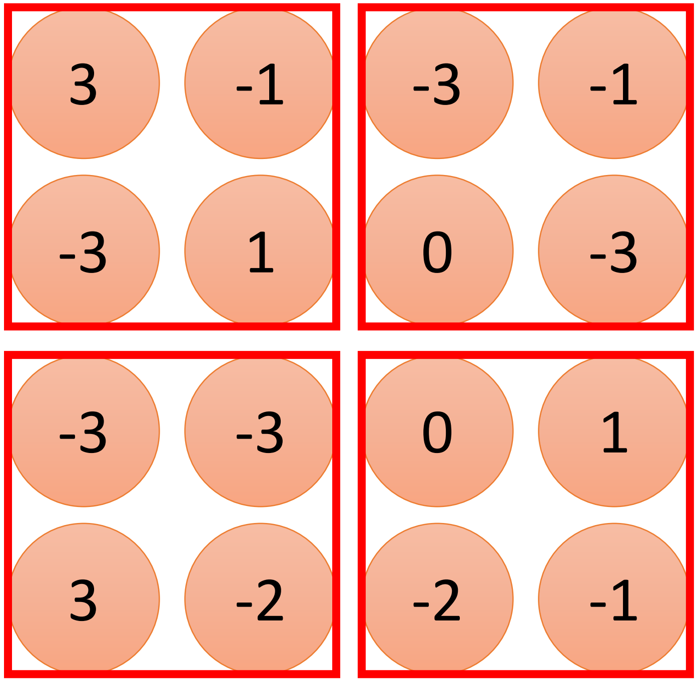
CNN – Max Pooling

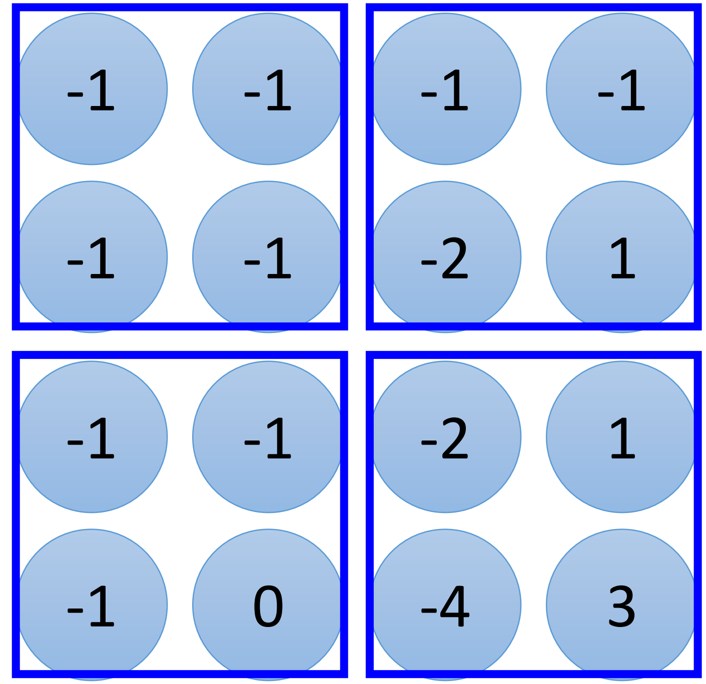
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

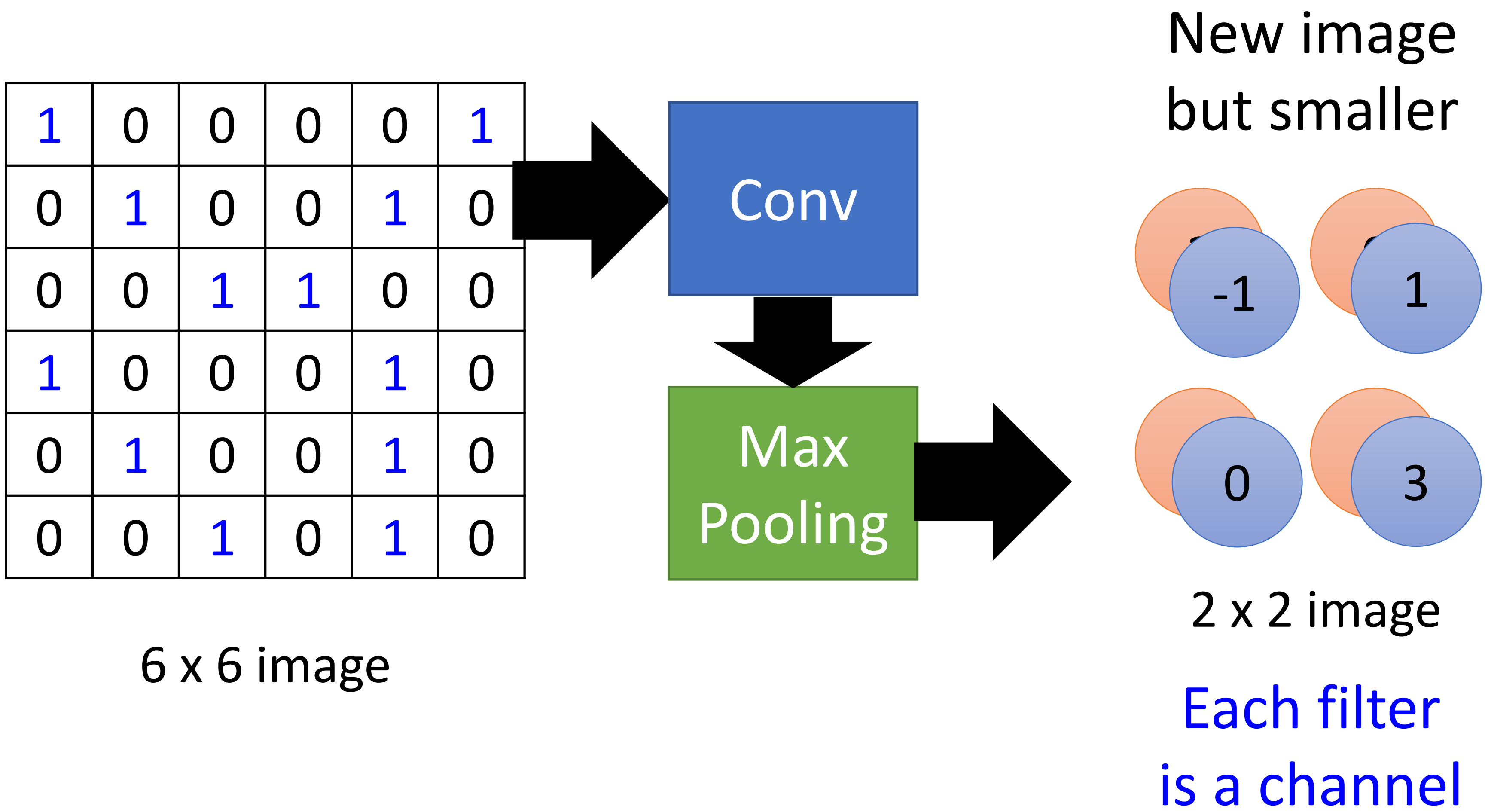
-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

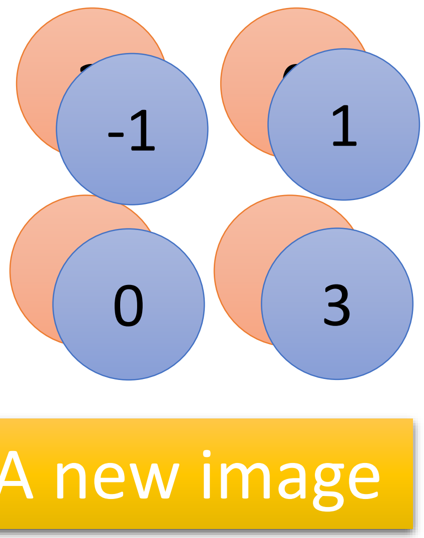




CNN – Max Pooling

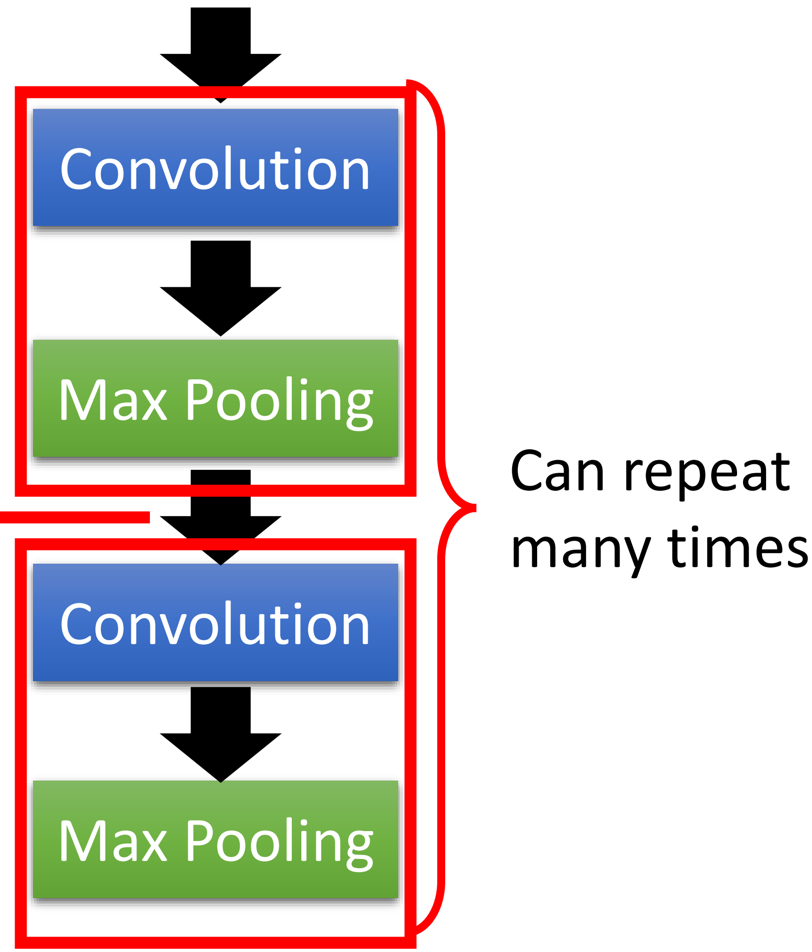


The whole CNN

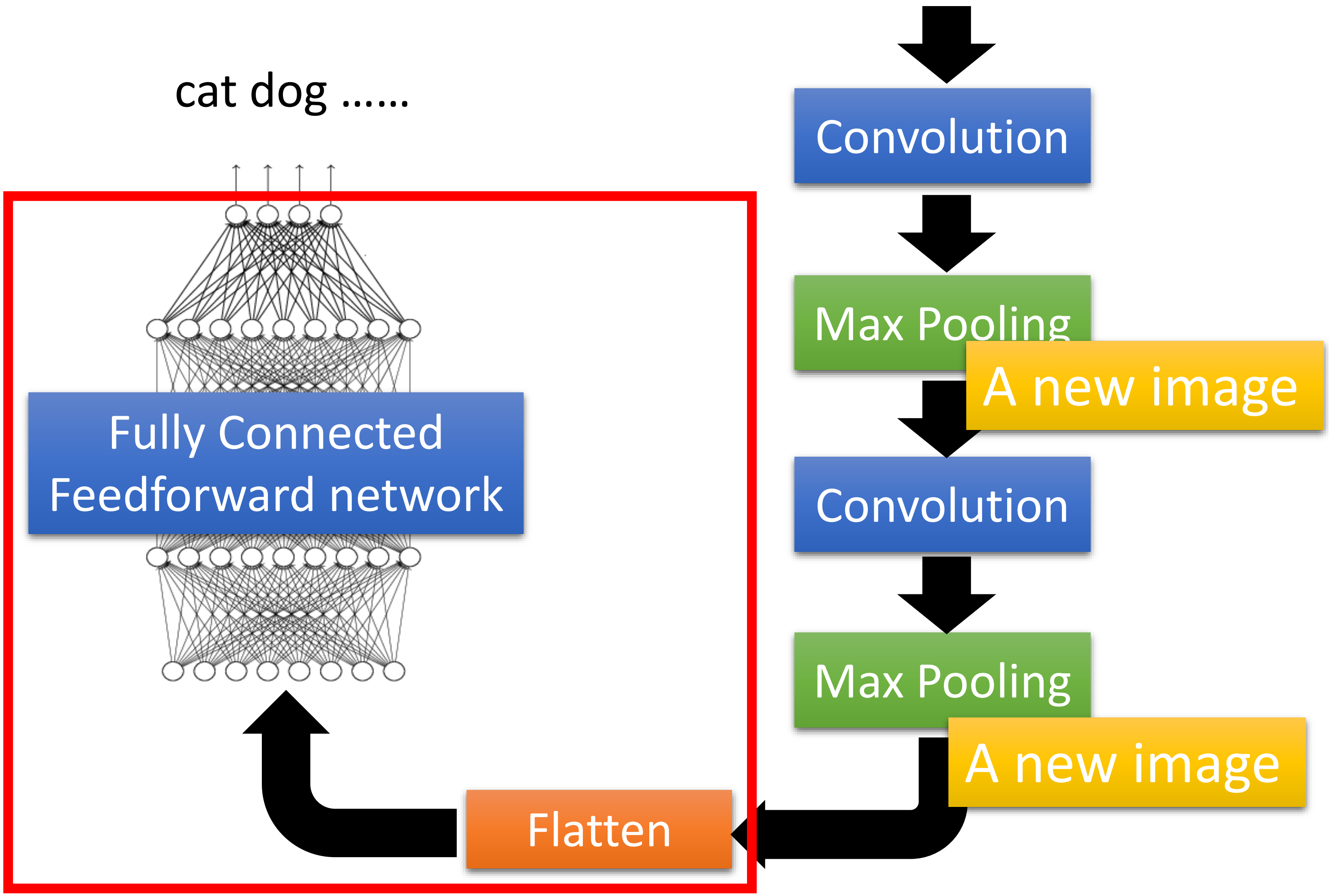


Smaller than the original image

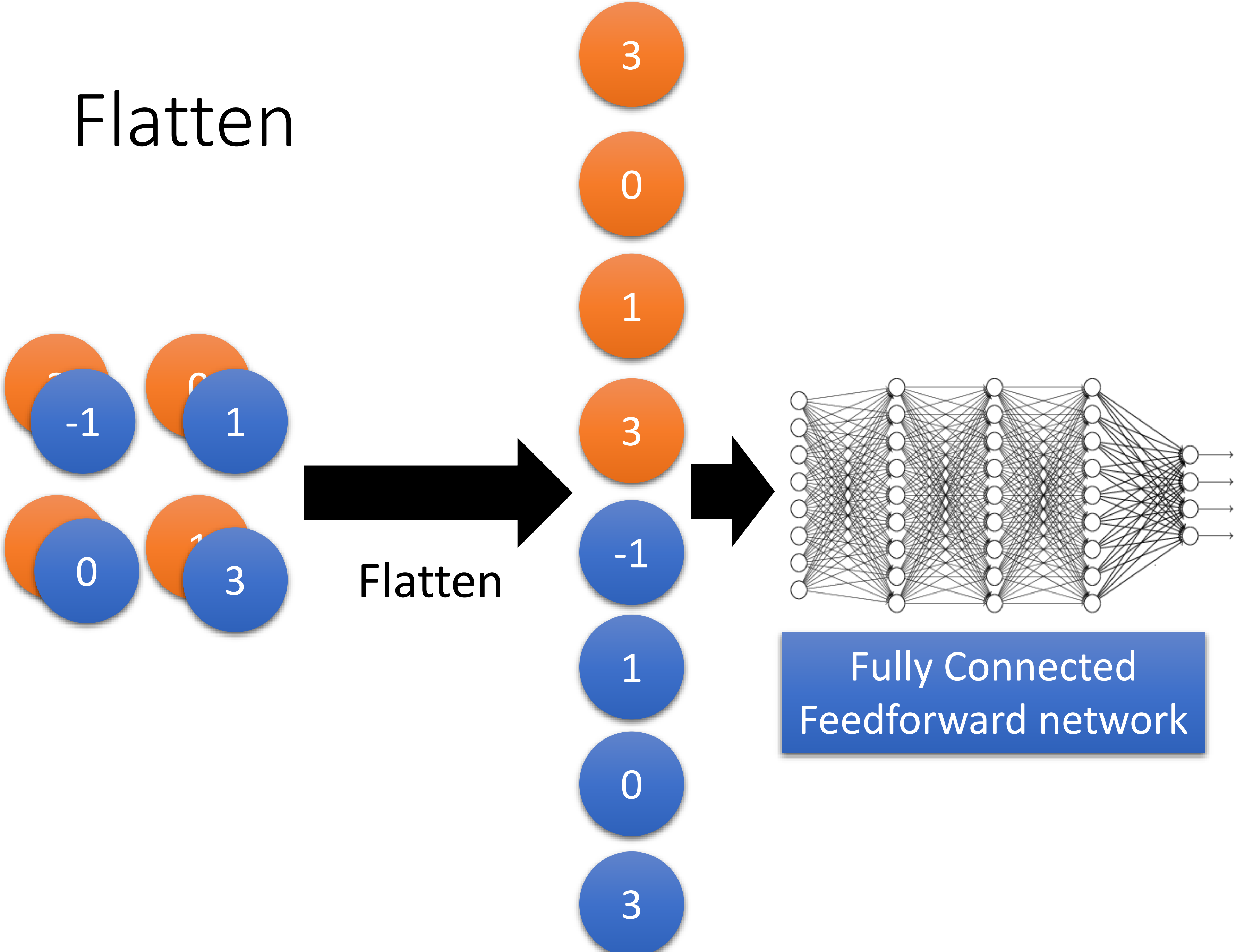
The number of the channel is the number of filters



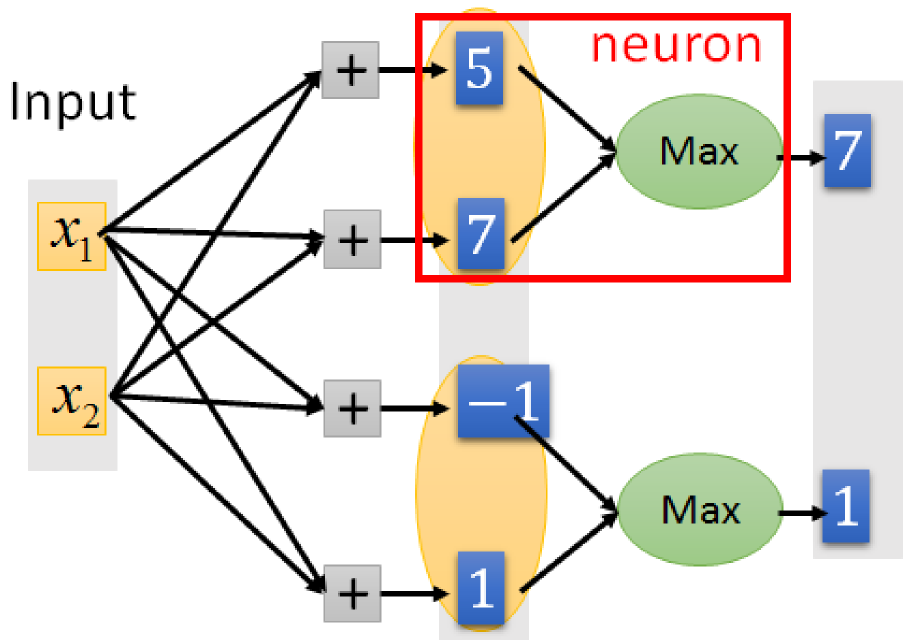
The whole CNN

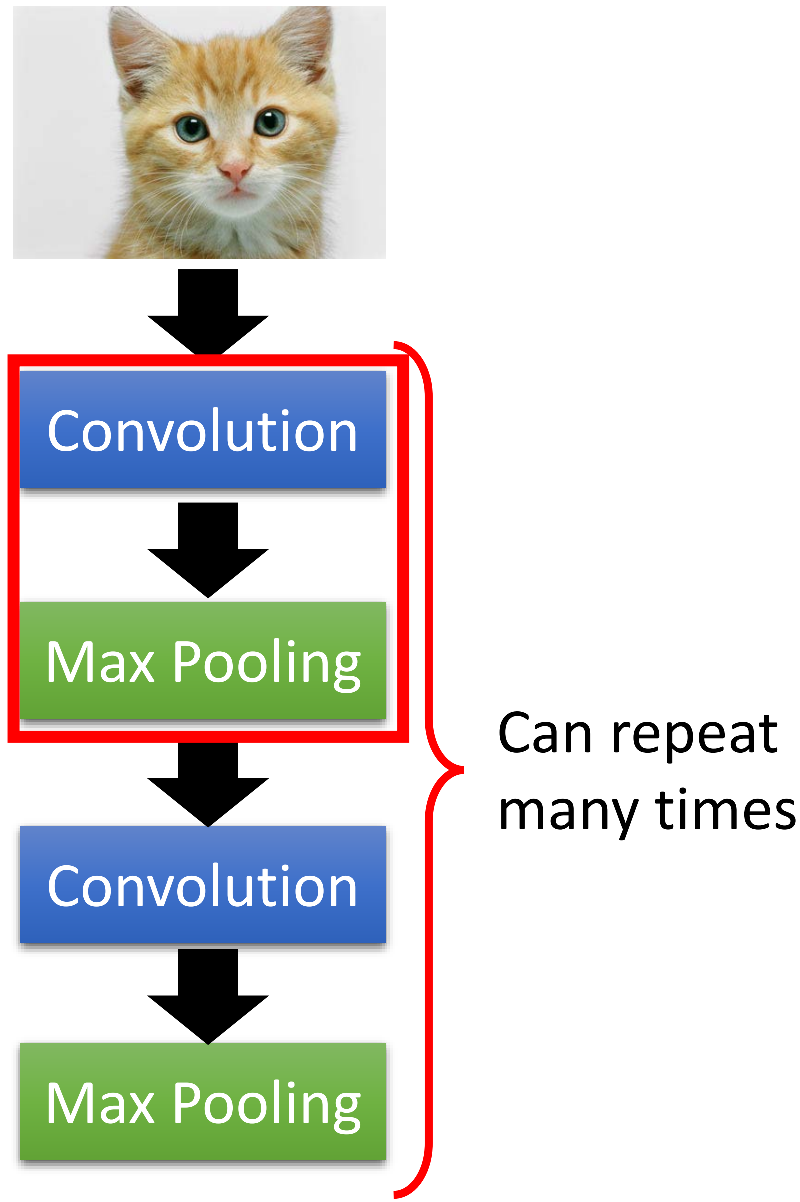


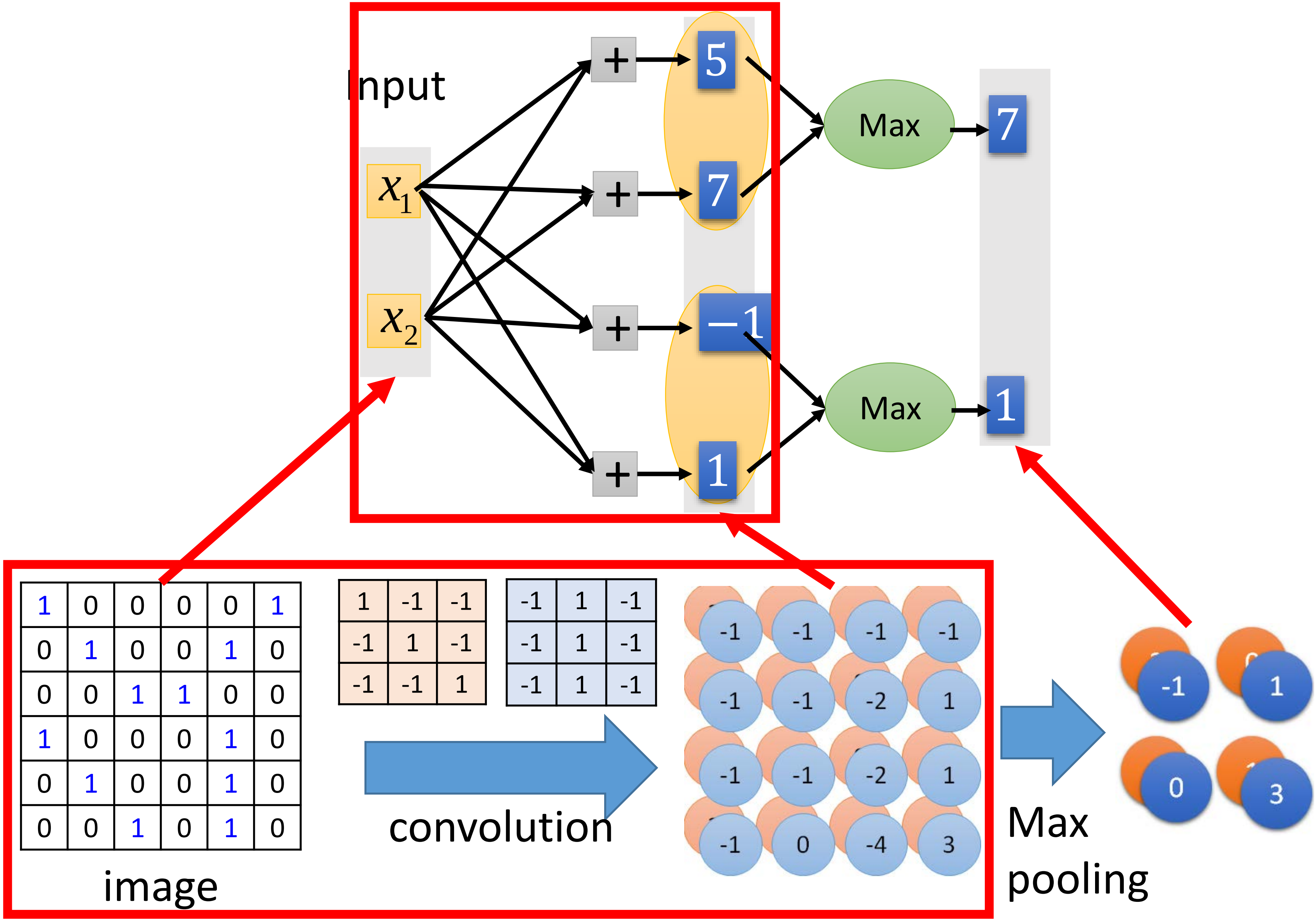
Flatten


-1
1
0
3
Flatten
3
0
1
3
-1
1
0
3
Fully Connected
Feedforward network

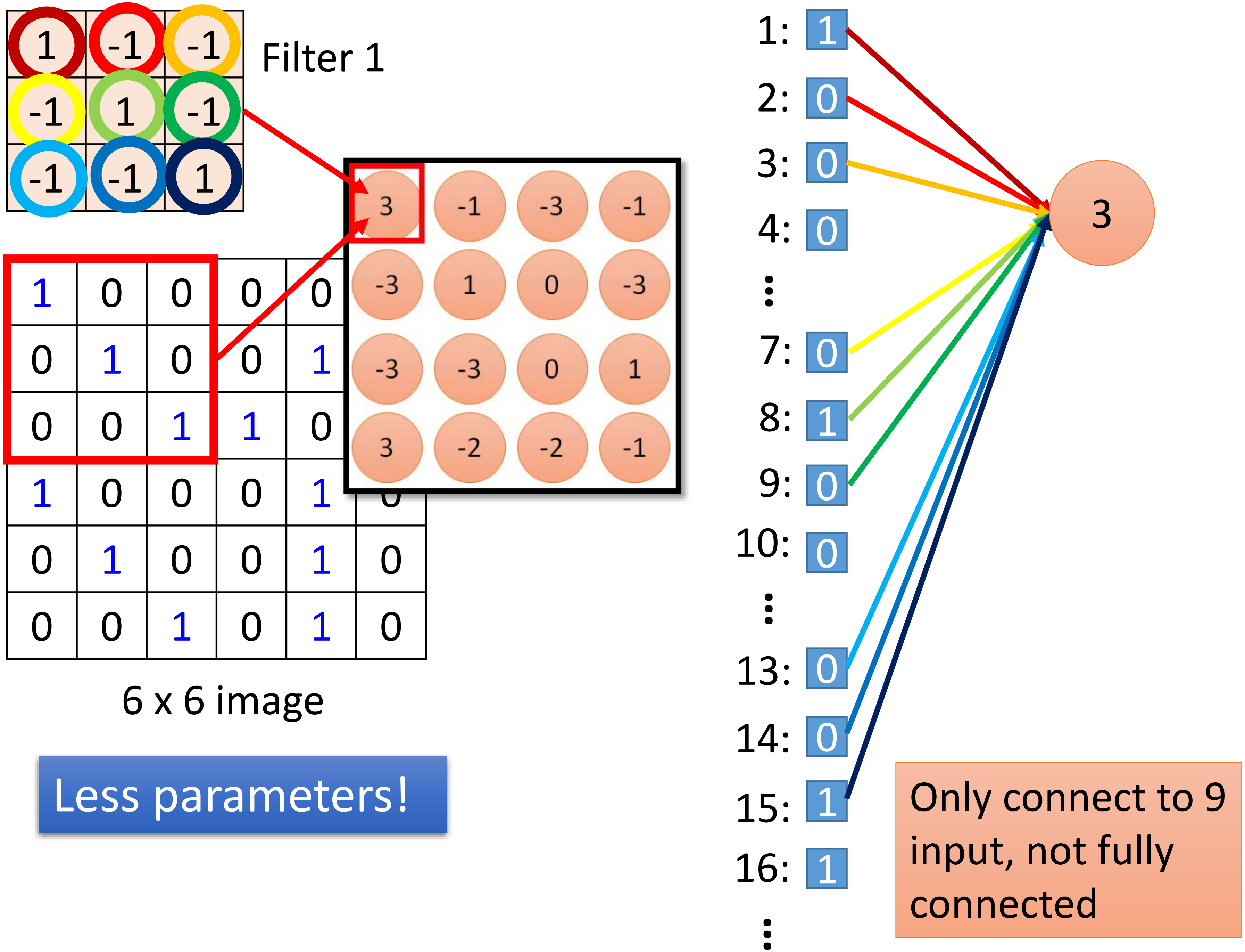
The whole CNN



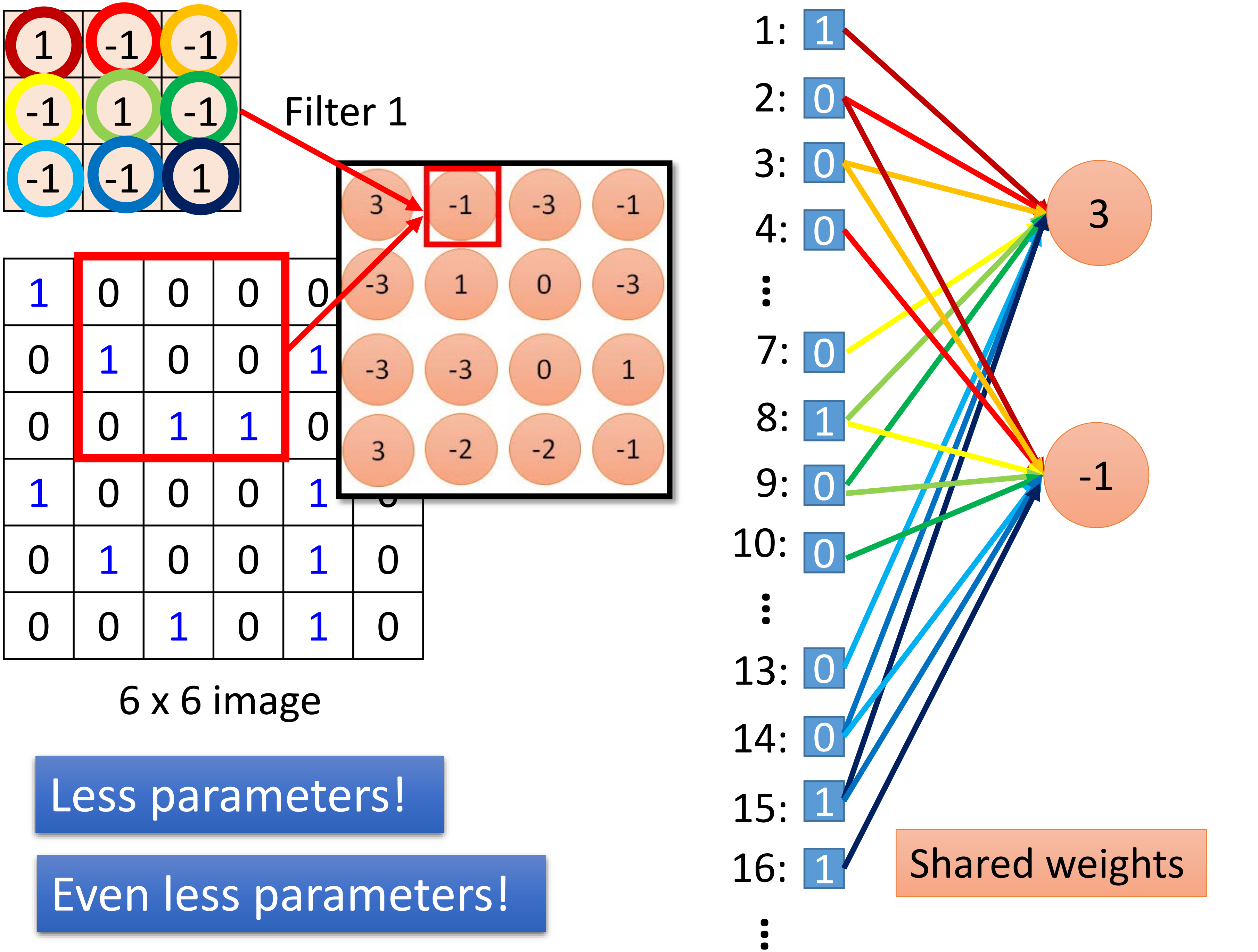




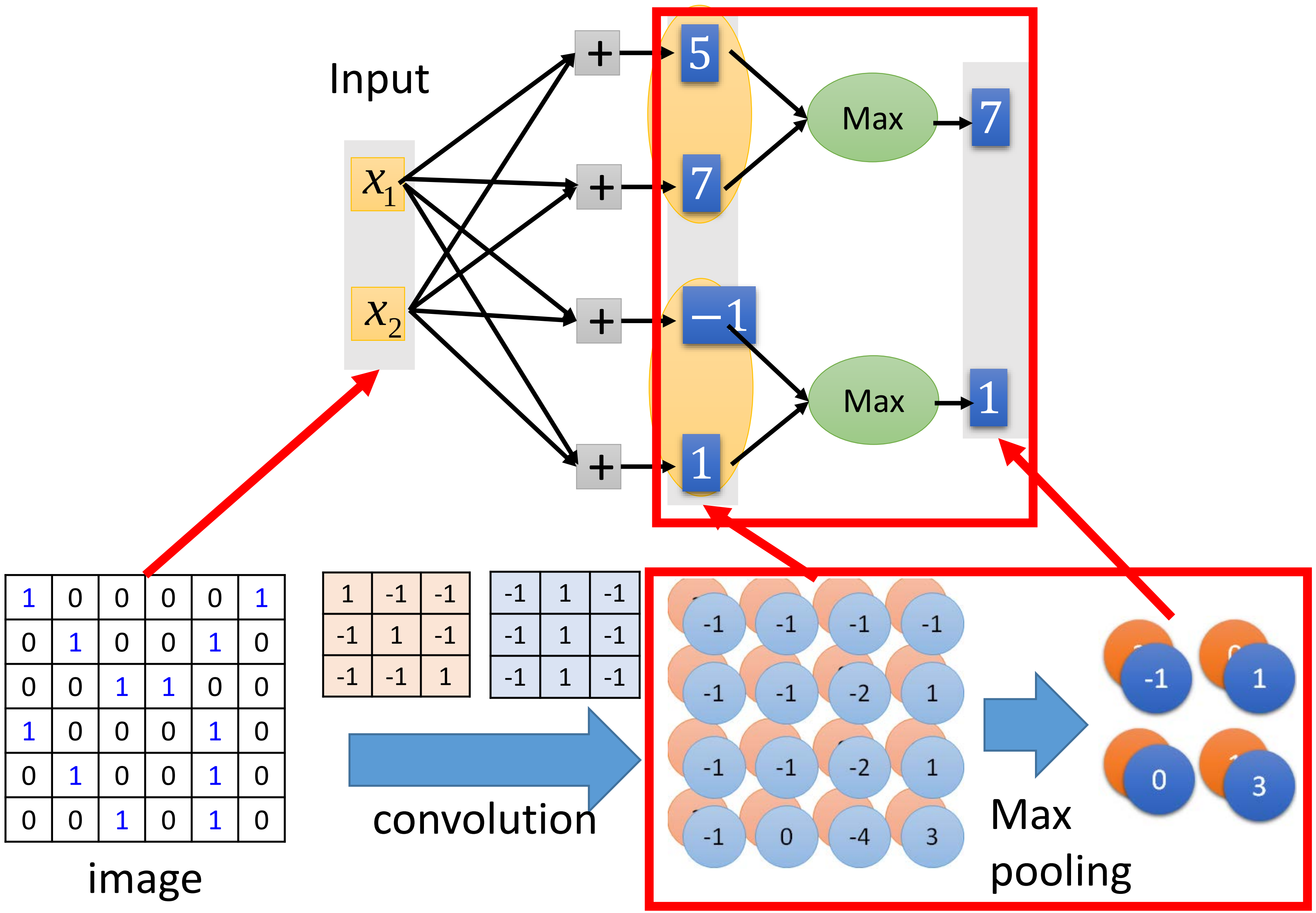
(Ignoring the non-linear activation function after the convolution.)



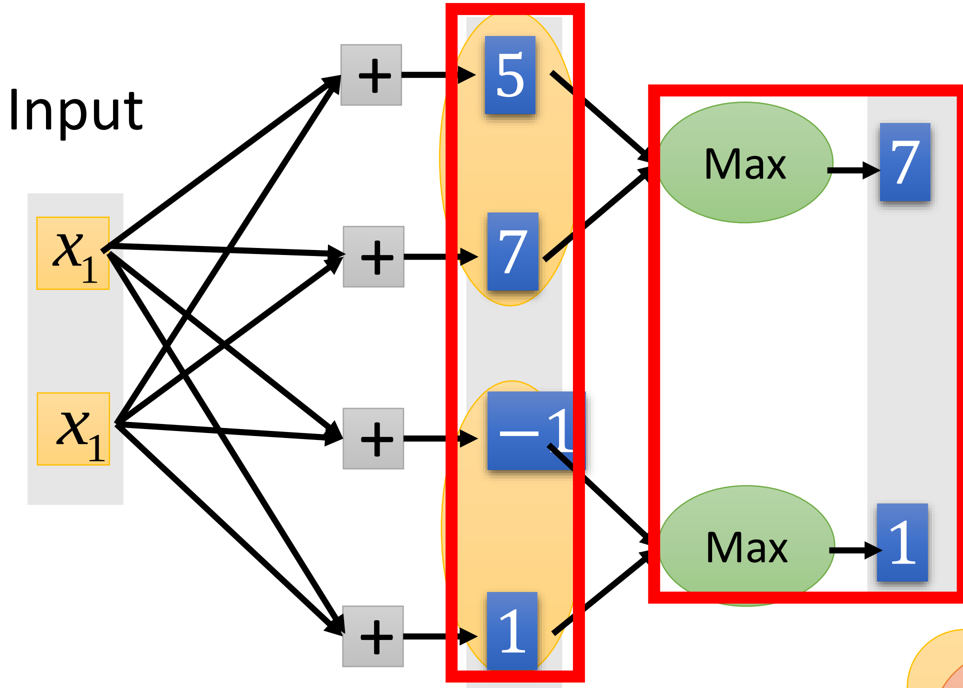
Filter 1
1 -1 -1
-1 1 -1
-1 -1 1
3 -1 -3 -1
-3 1 0 -3
-3 -3 0 1
3 -2 -2 -1
6 x 6 image
Less parameters!
1: 1
2: 0
3: 0
4: 0
7: 0
8: 1
9: 0
10: 0
13: 0
14: 0
15: 1
16: 1
3
Only connect to 9 input, not fully connected

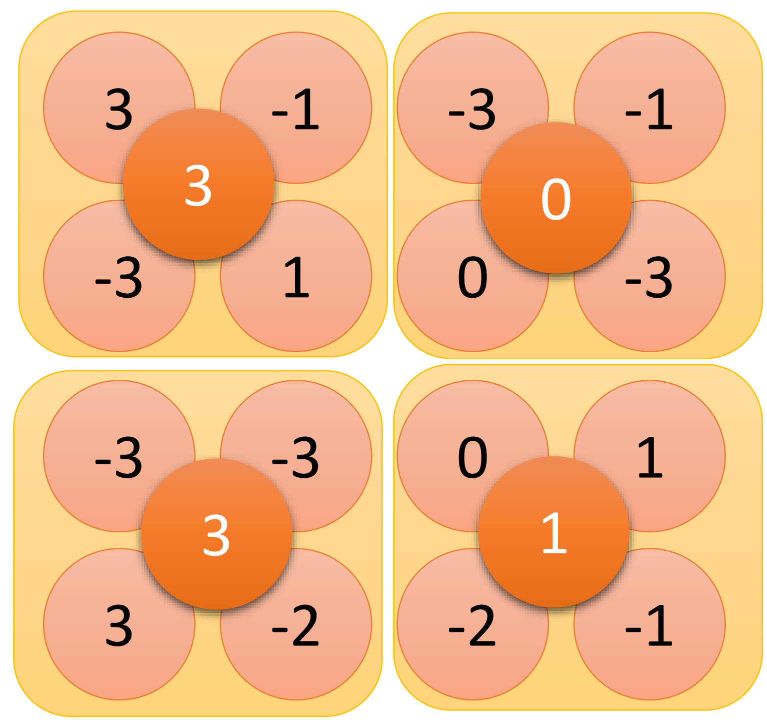


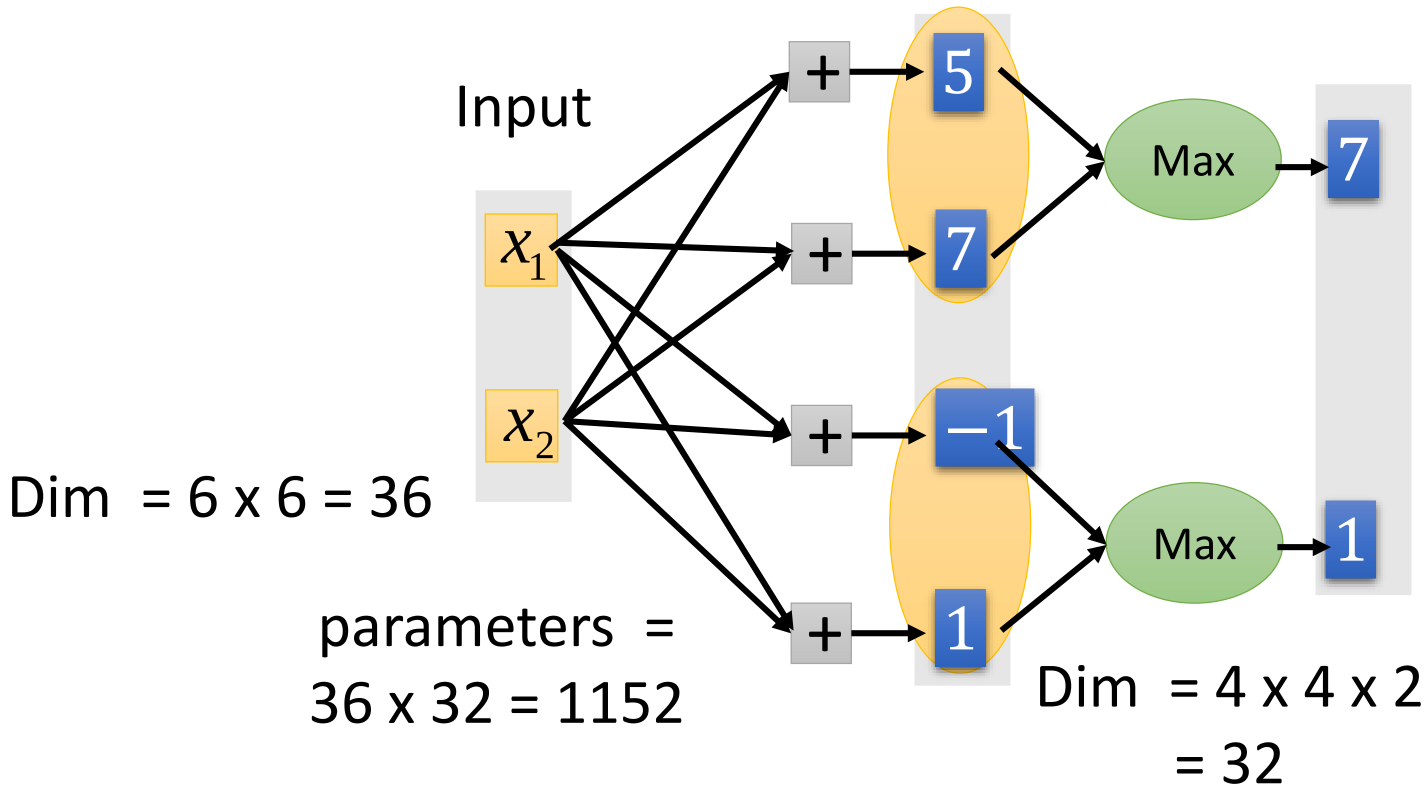
Filter 1
6 x 6 image
Less parameters!
Even less parameters!
1: 1
2: 0
3: 0
4: 0
7: 0
8: 1
9: 0
10: 0
13: 0
14: 0
15: 1
16: 1
3
-1
Shared weights



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


Input
x_1
x_1
+
+
+
+
5
7
−1
1
Max
Max
7
1


3 -1 -3 1
3
-3 -1 0 -3
0
-3 -3 3 -2
3
0 1 -2 -1
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

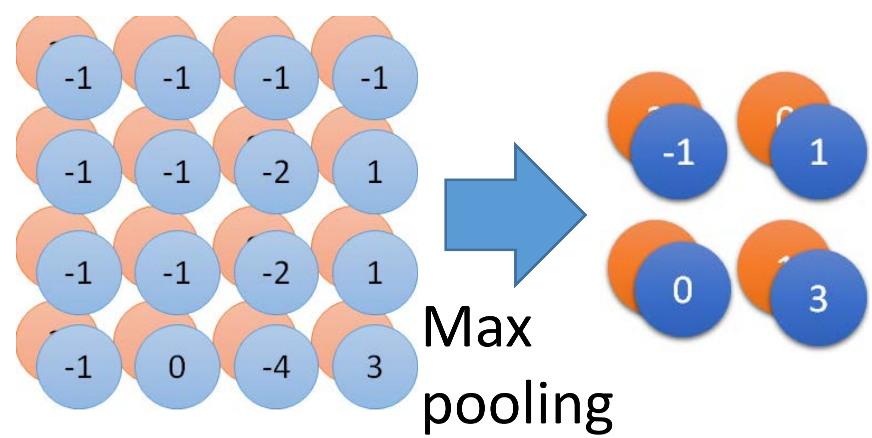
image

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

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

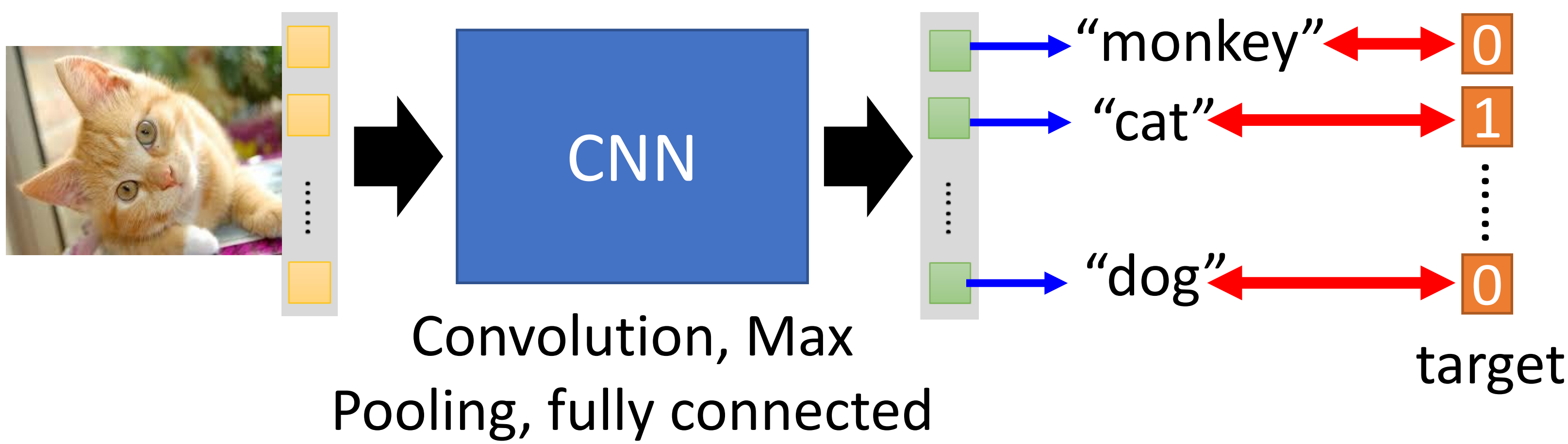
convolution

Only 9 x 2 = 18 parameters



Convolutional Neural Network

Step 1: Neural Network → Step 2: Learning Target → Step 3: Learn!



Learning: Nothing special, just gradient descent ……

CNN in Keras

Only modified the network structure

```
model.add(Convolution2D(32, 3, 3,
          border_mode='same',
          input_shape=(3,32,32)))
model.add(Activation('relu'))
```

```
model.add(Convolution2D(32, 3, 3))
model.add(Activation('relu'))
```

```
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

```
model.add(Convolution2D(64, 3, 3,
          border_mode='same'))
model.add(Activation('relu'))
```

```
model.add(Convolution2D(64, 3, 3))
model.add(Activation('relu'))
```

```
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

```
model.add(Flatten())
```

```
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
```

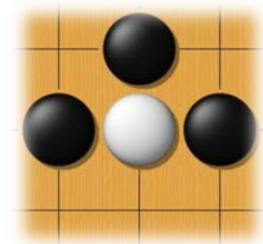
```
model.add(Dense(10))
model.add(Activation('softmax'))
```

Code: https://github.com/fchollet/keras/blob/master/examples/cifar10_cnn.py

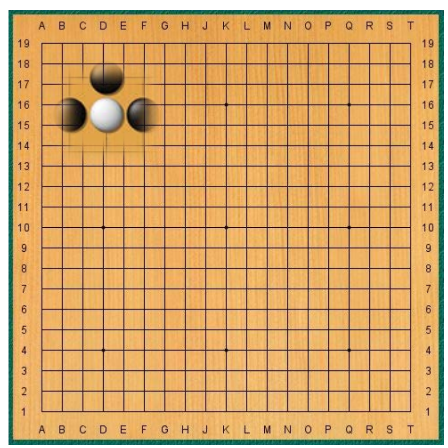
Why CNN for playing Go?

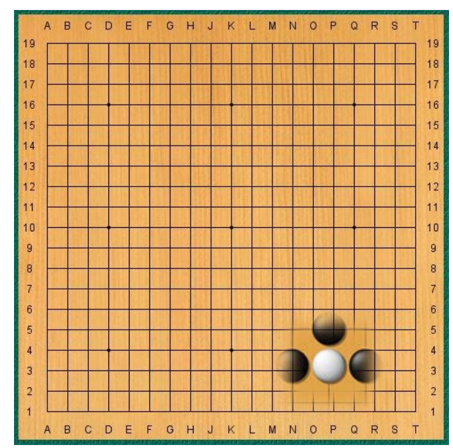
- Some patterns are much smaller than the whole image

Alpha Go uses 5 x 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 23×23 image, then convolves k filters of kernel size 5×5 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 21×21 image, then convolves k filters of kernel size 3×3 with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size 1×1 with stride 1, with a different bias for each position, and applies a softmax func-
tion. The … Extended
Data Table 3 additionally show the results of training with $k = 128$, 256 and 384 filters.

Alpha Go does not use Max Pooling ……

Variants of Neural Networks

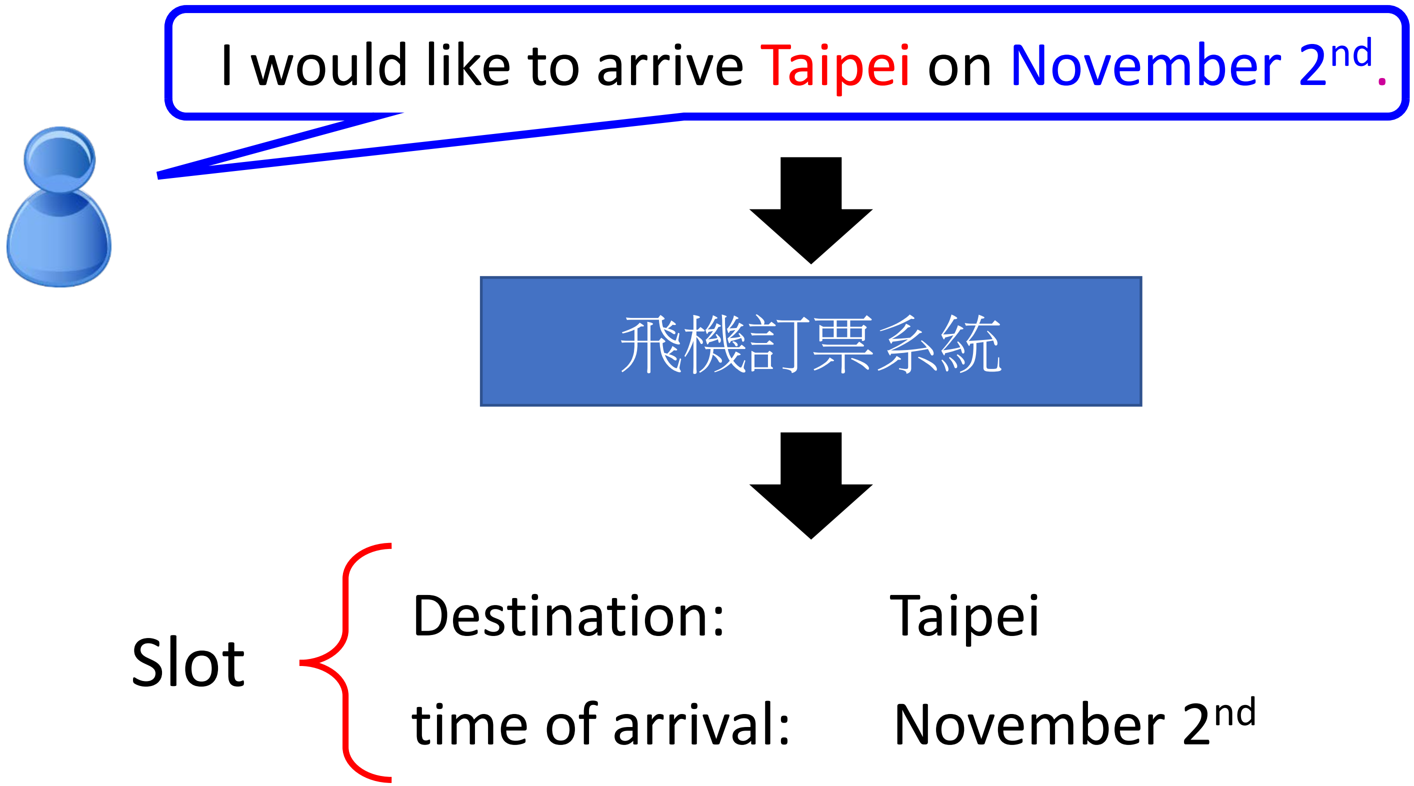
Convolutional Neural Network (CNN)

Recurrent Neural Network (RNN)

Neural Network with Memory

Example Application

- Slot Filling

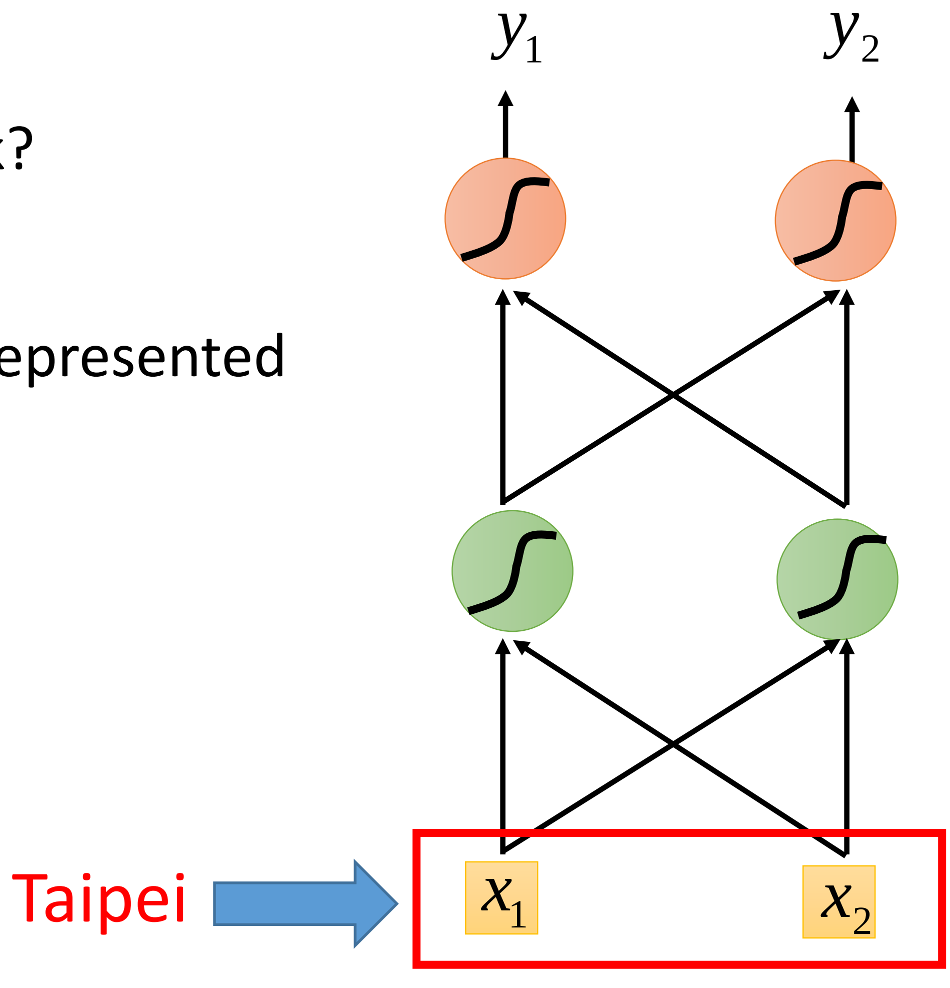


Example Application

Solving slot filling by Feedforward network?

Input: a word

(Each word is represented as a vector)



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.

Each dimension corresponds to a word in the lexicon

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

apple = [1 0 0 0 0]

bag = [0 1 0 0 0]

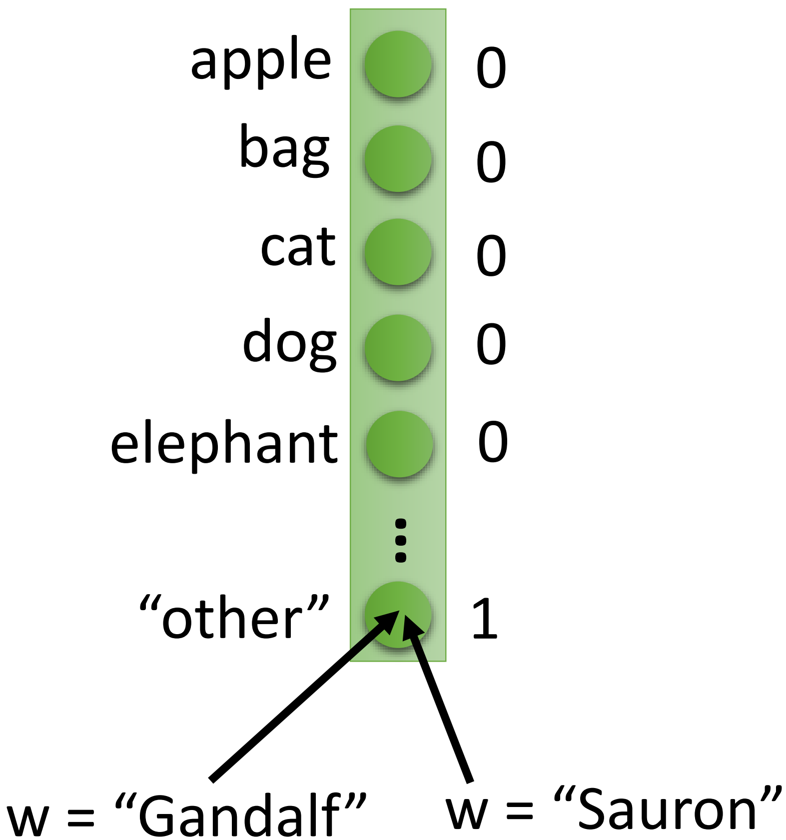
cat = [0 0 1 0 0]

dog = [0 0 0 1 0]

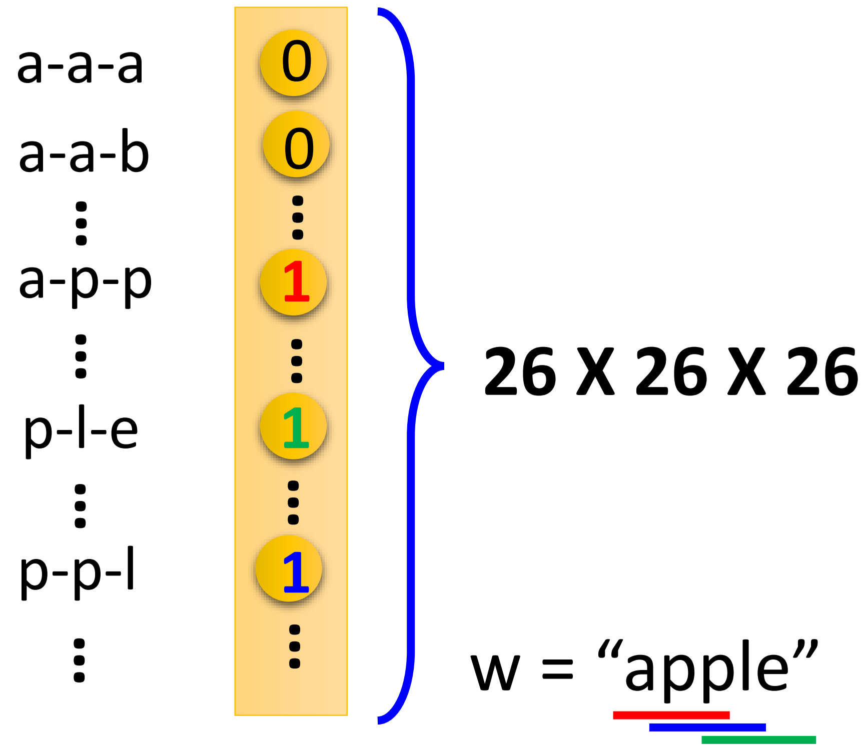
elephant = [0 0 0 0 1]

Beyond 1-of-N encoding

Dimension for "Other"



Word hashing



Example Application

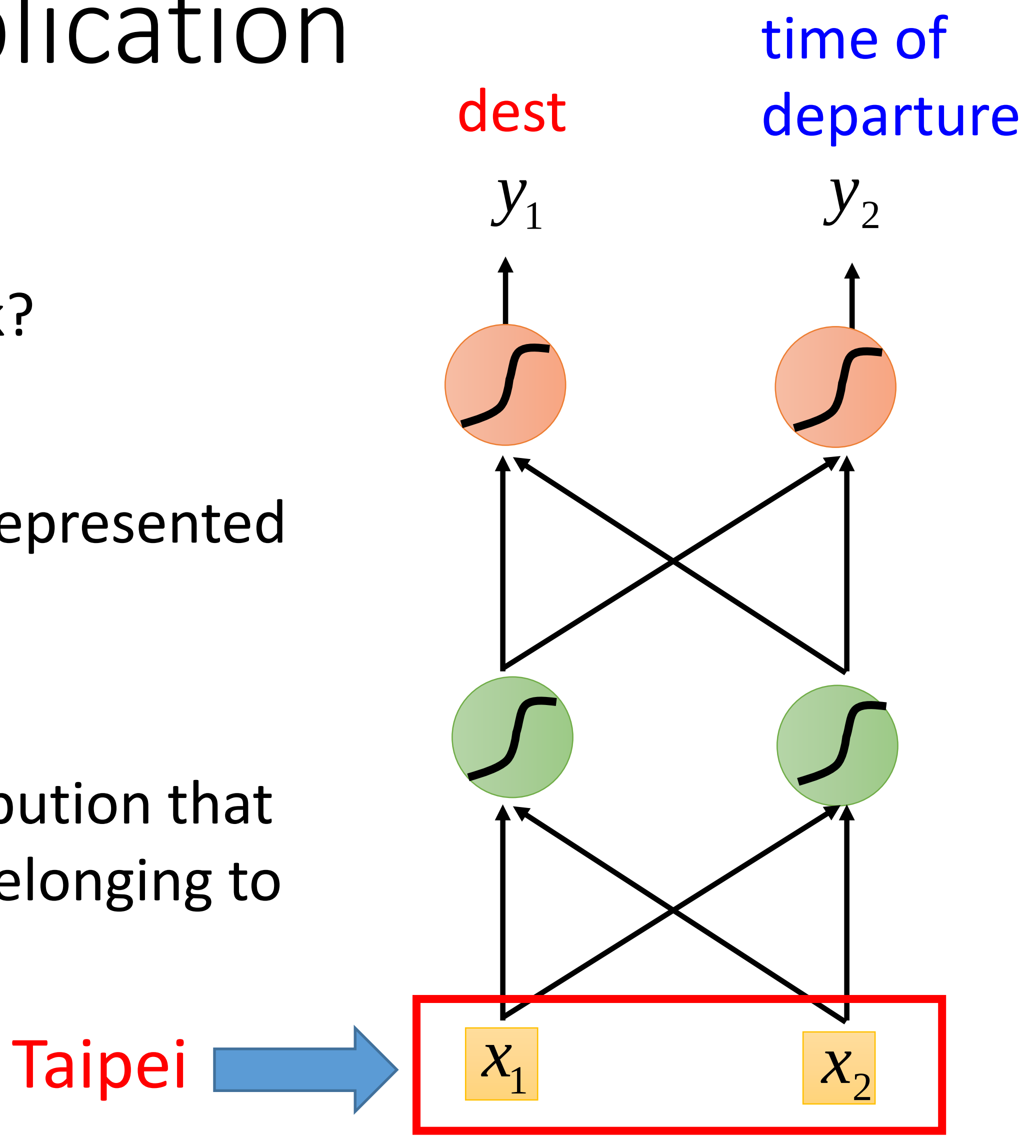
Solving slot filling by Feedforward network?

Input: a word

(Each word is represented as a vector)

Output:

Probability distribution that the input word belonging to the slots



Example Application

arrive Taipei on November 2nd

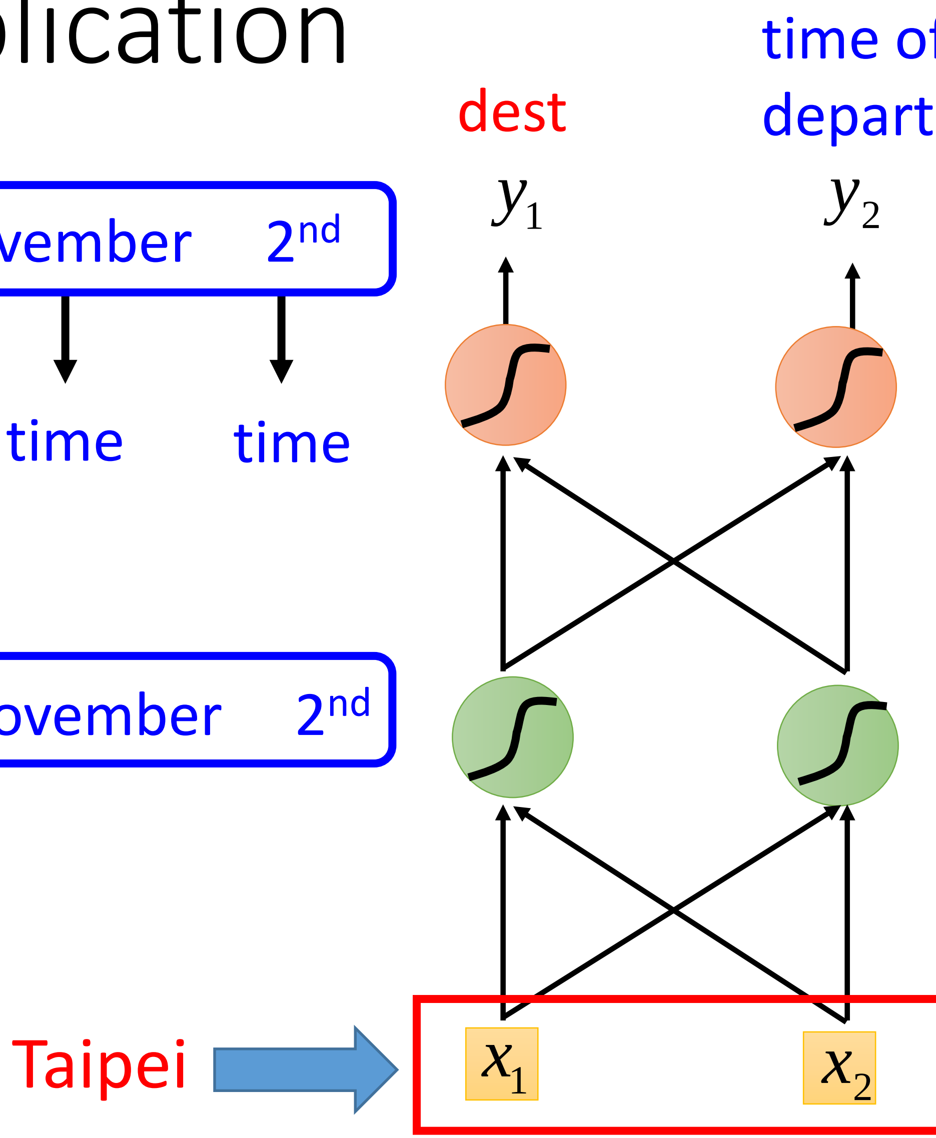
other dest other time time

Problem?

leave Taipei on November 2nd

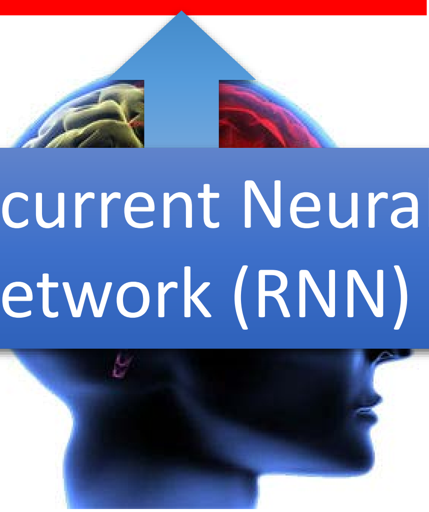
place of departure

Neural network needs memory!



Recurrent Neural Network

Step 1: Neural Network → Step 2: Learning Target → Step 3: Learn!



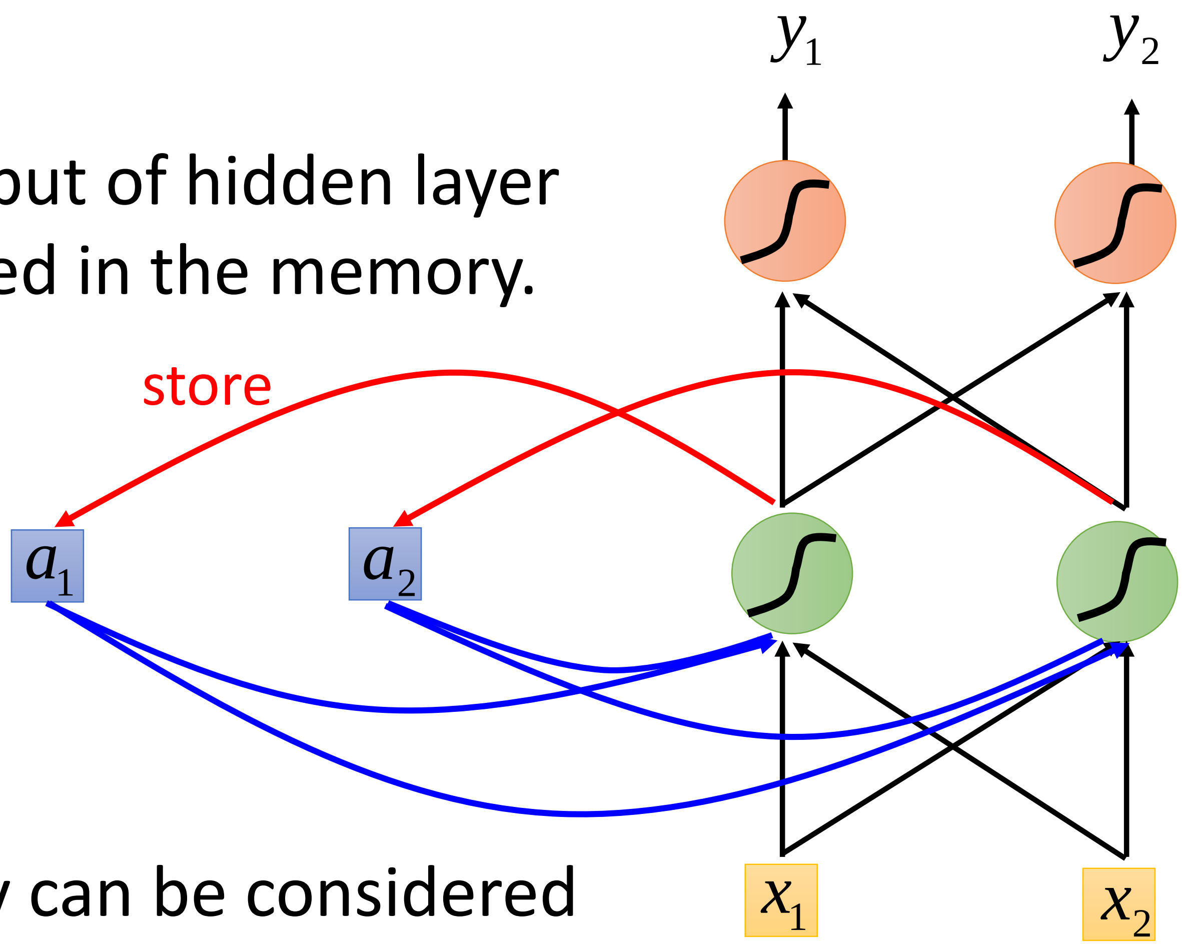
Recurrent Neural Network (RNN)

天生的腦

http://onepiece1234567890.blogspot.tw/2013/12/blog-post_8.html

Recurrent Neural Network (RNN)

The output of hidden layer are stored in the memory.



Memory can be considered as another input.

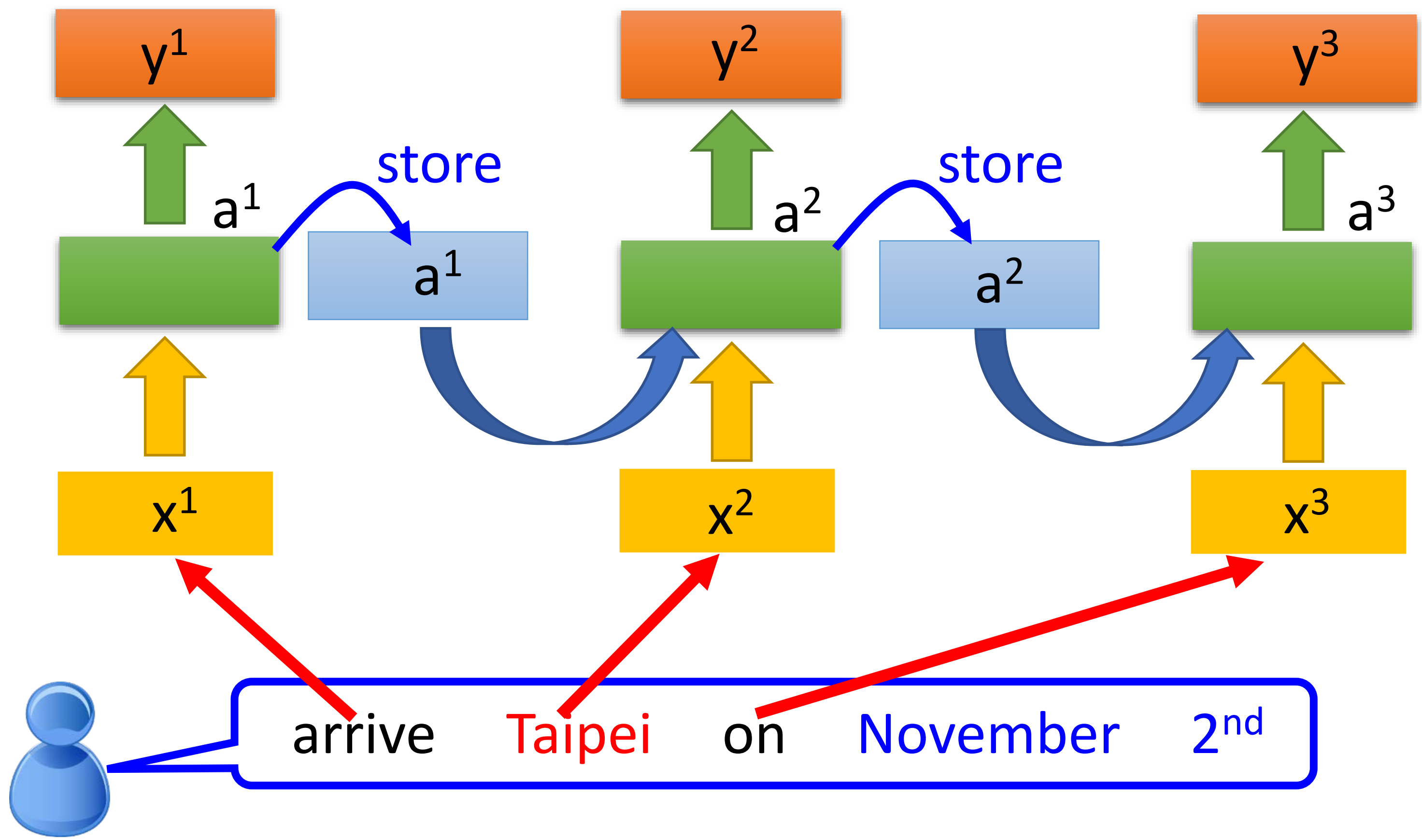
RNN

The same network is used again and again.

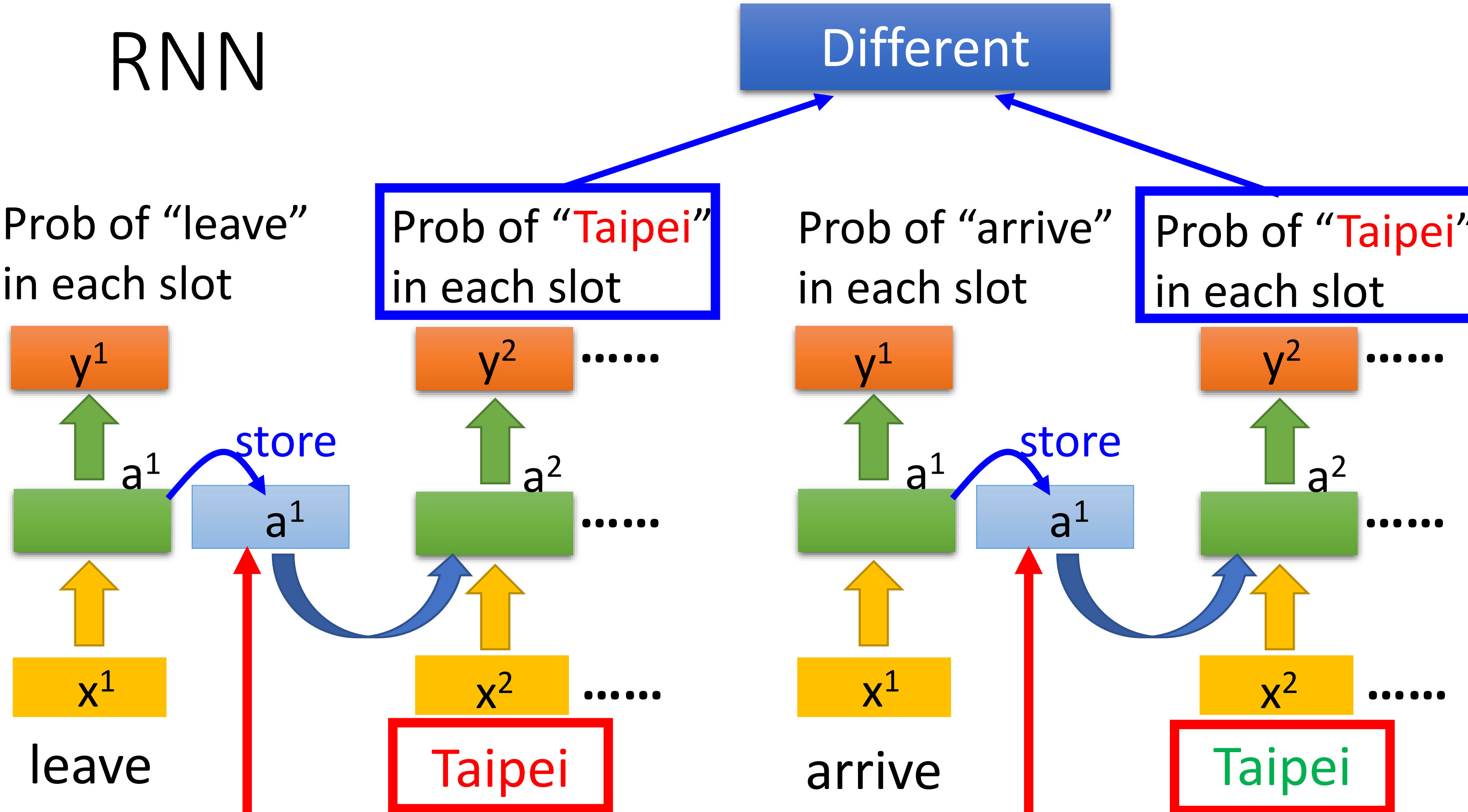
Probability of "arrive" in each slot

Probability of "Taipei" in each slot

Probability of "on" in each slot

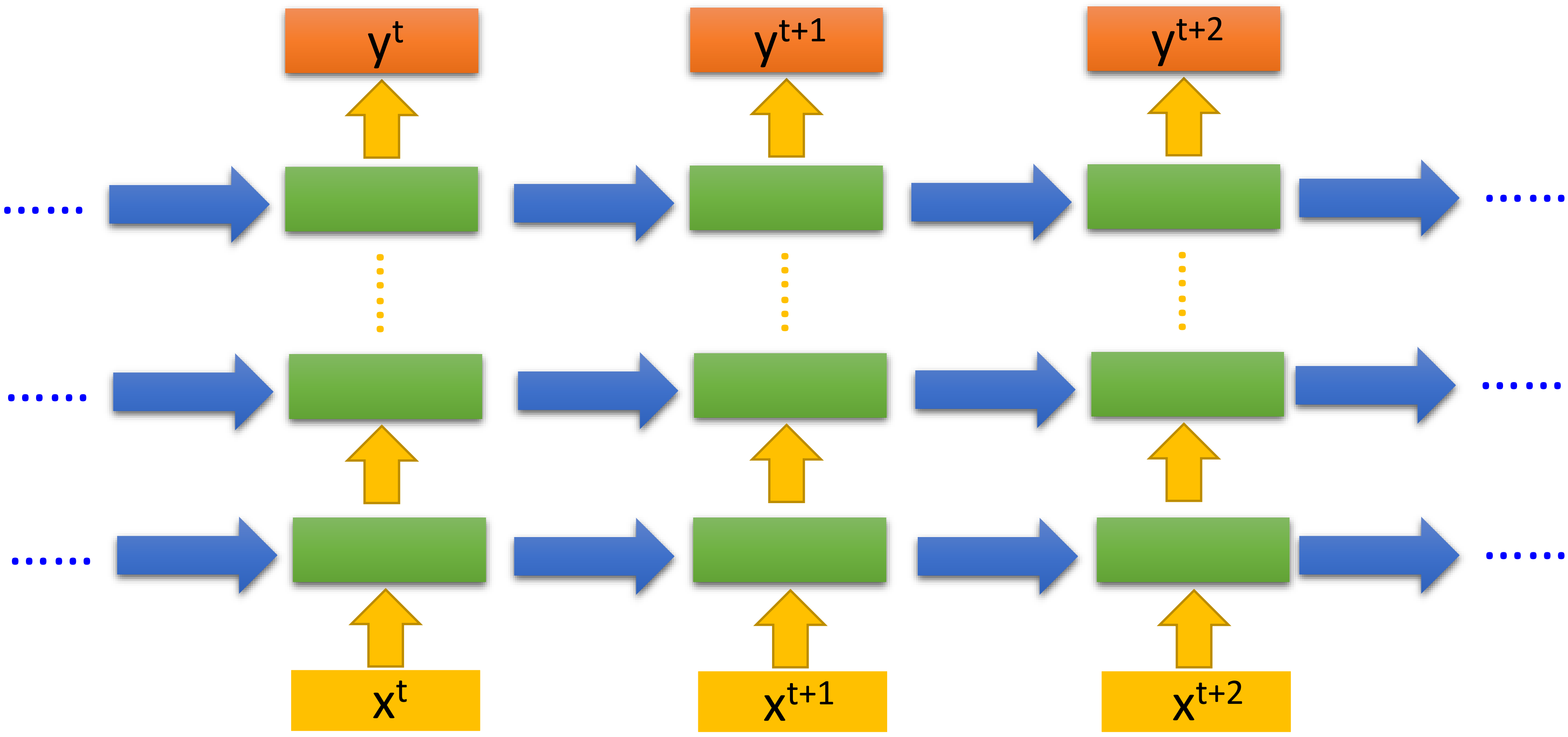


RNN

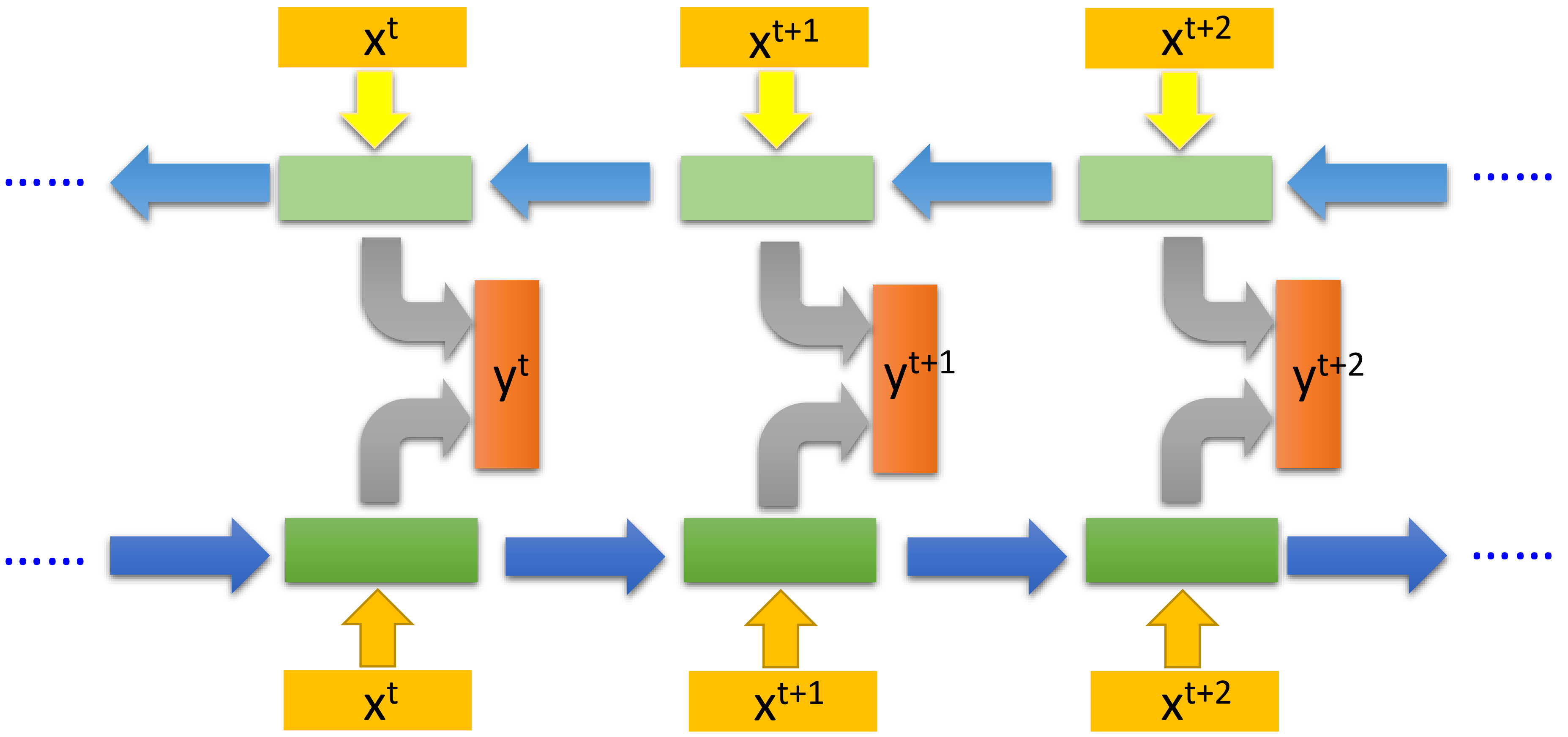


The values stored in the memory is different.

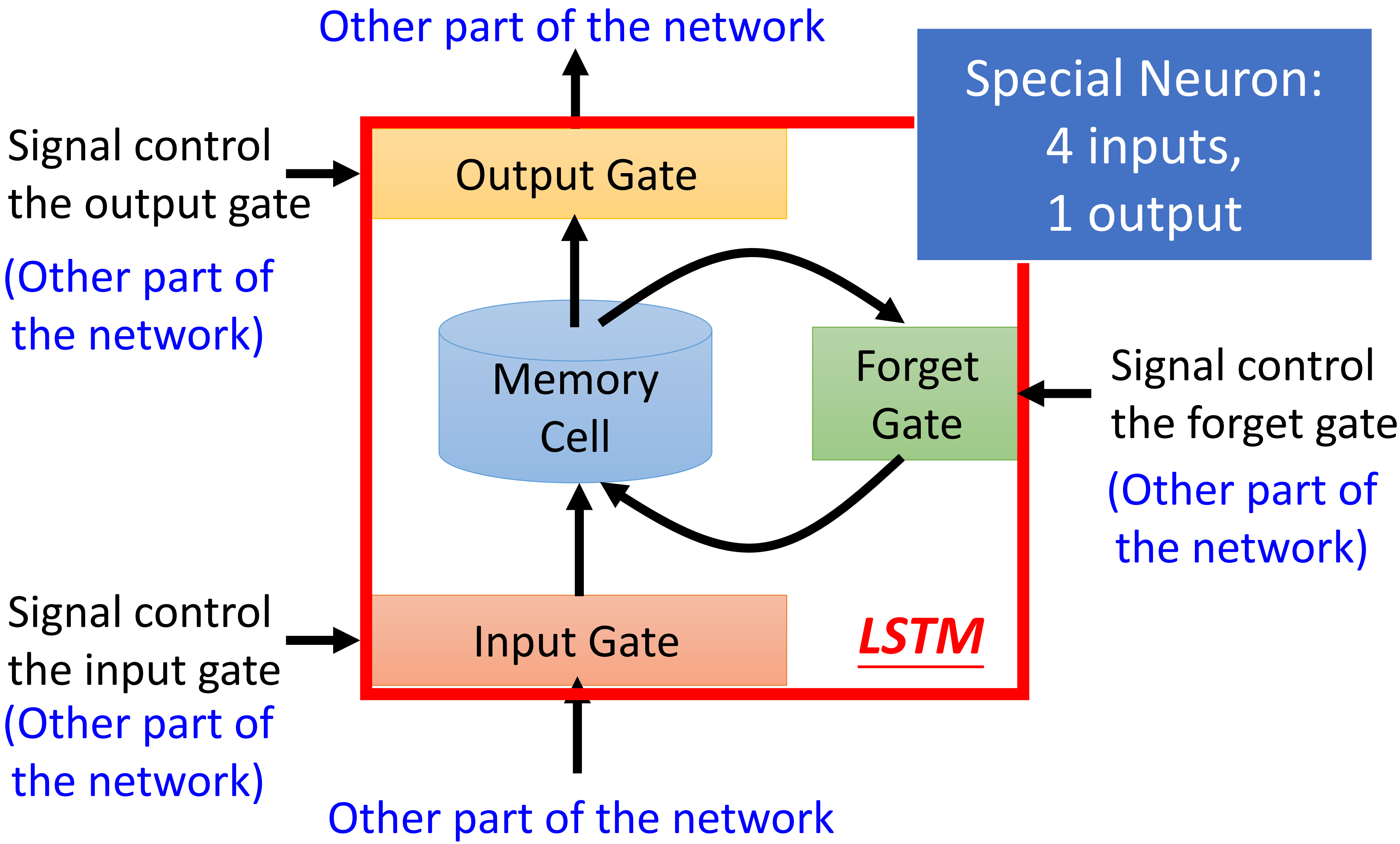
Of course it can be deep …

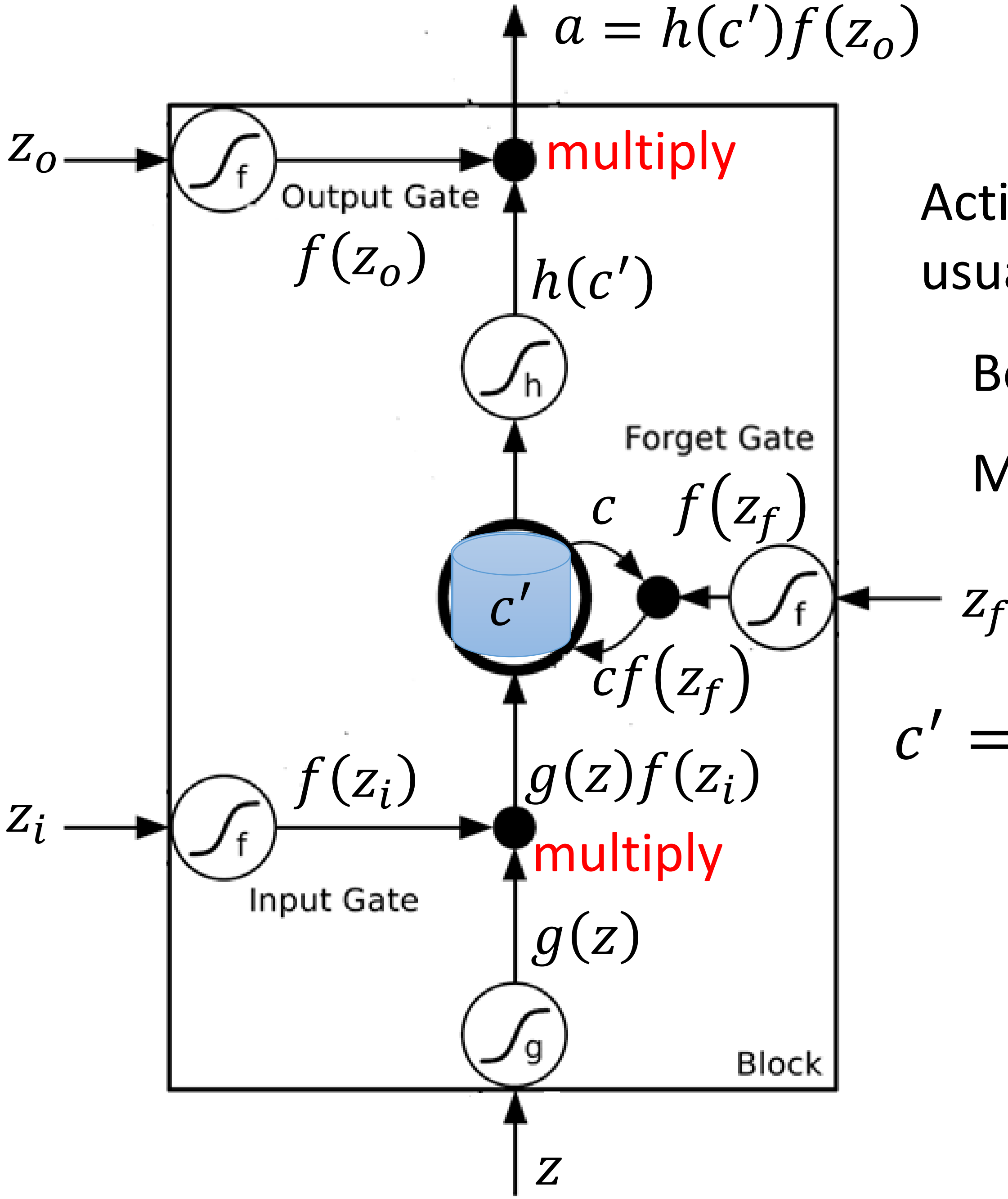


Bidirectional RNN



Long Short-term Memory (LSTM)



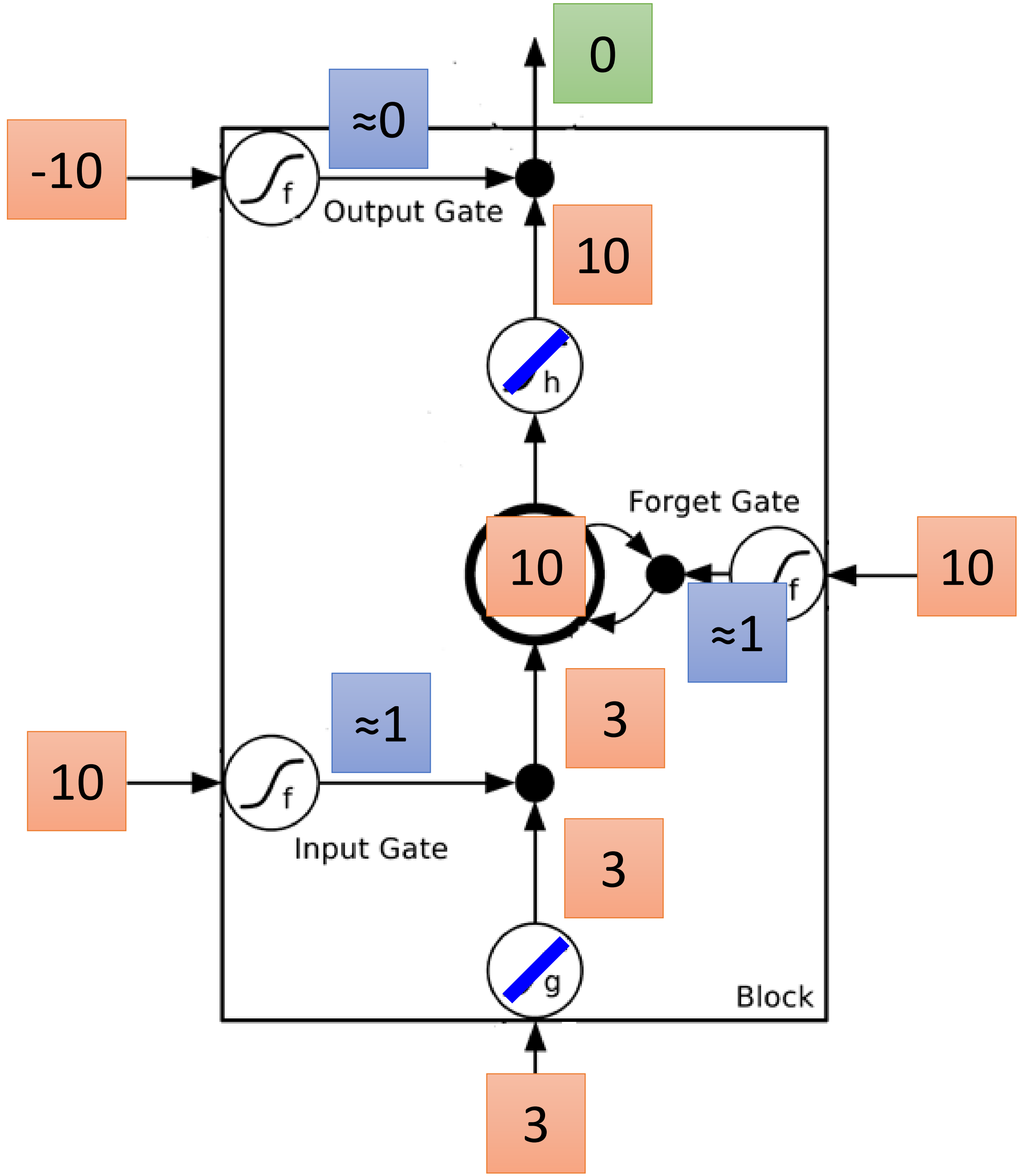


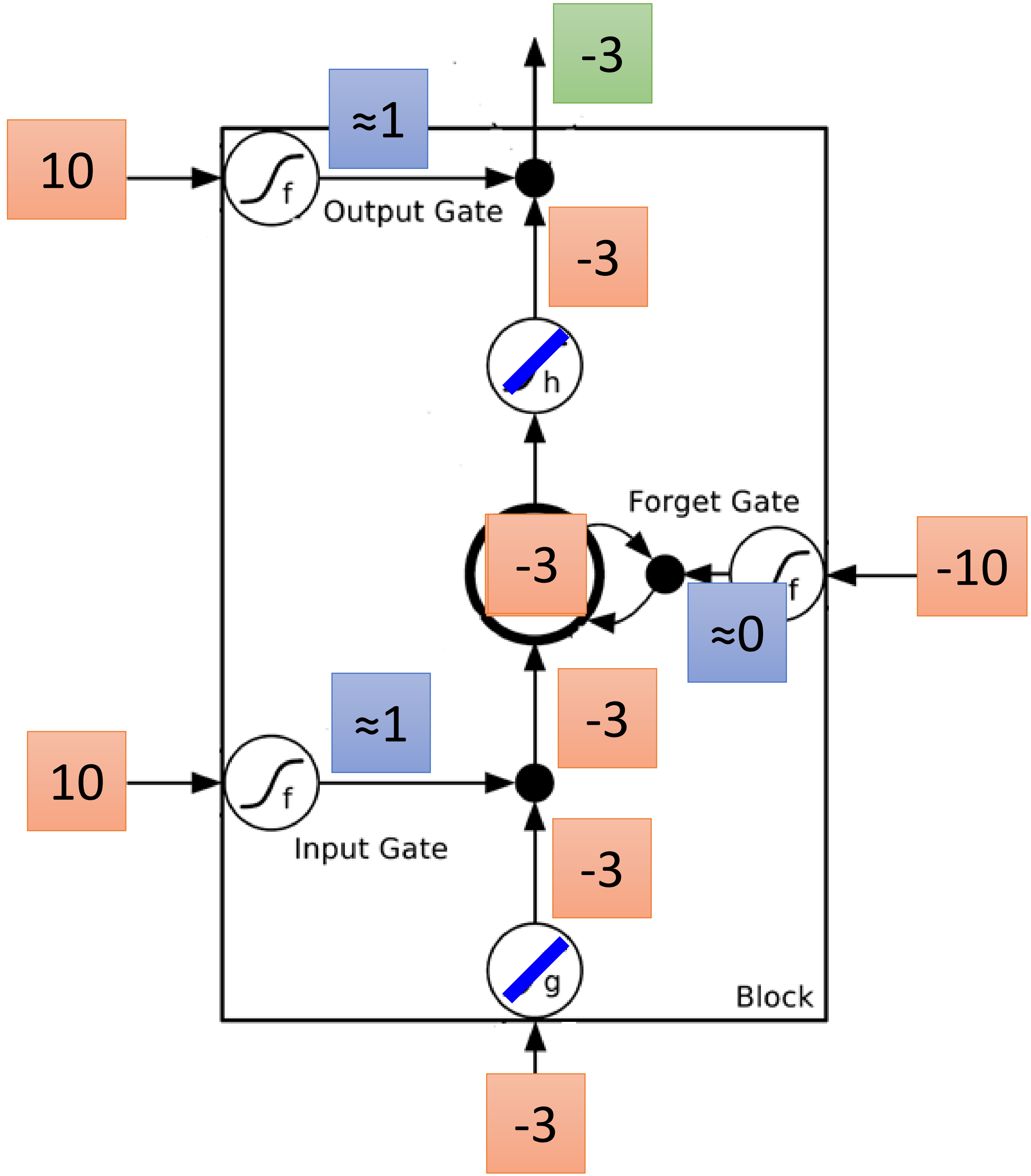
Activation function f is usually a sigmoid function

Between 0 and 1

Mimic open and close gate

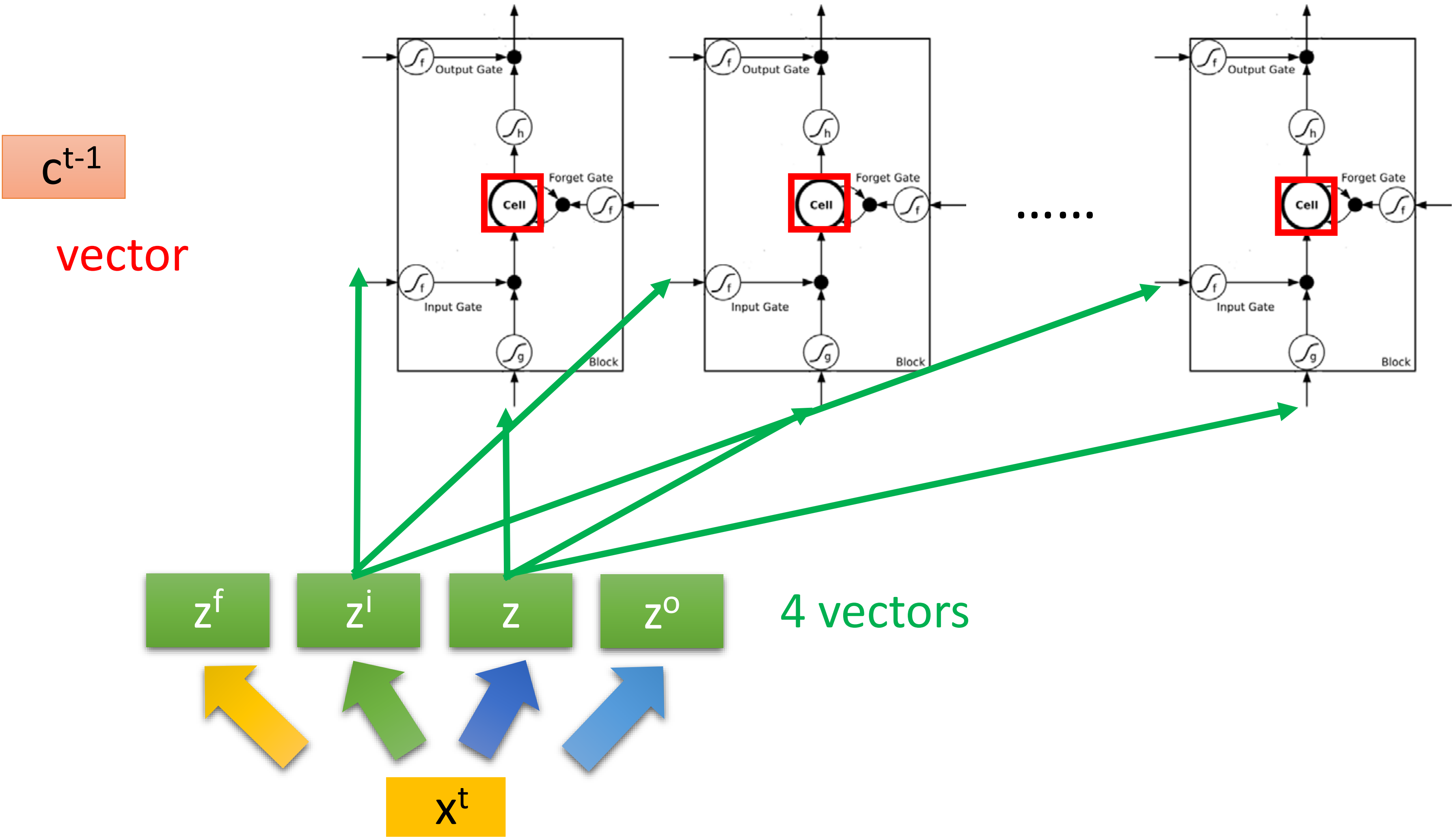
$$c' = g(z)f(z_i) + cf(z_f)$$


0
≈0
-10
Output Gate
10
h
Forget Gate
10
10
≈1
3
≈1
10
Input Gate
3
g
Block
3



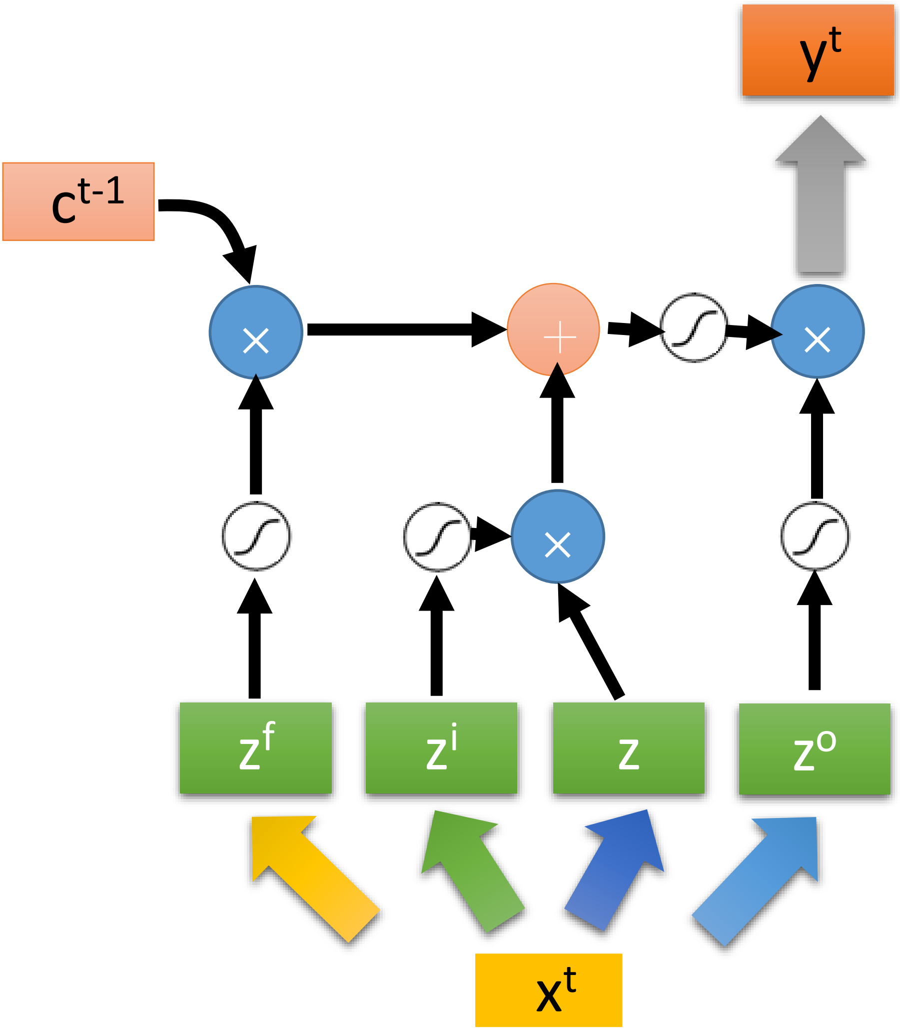
-3
≈1
10
Output Gate
-3
h
Forget Gate
-3
-10
≈0
≈1
-3
10
Input Gate
-3
g
Block
-3

LSTM

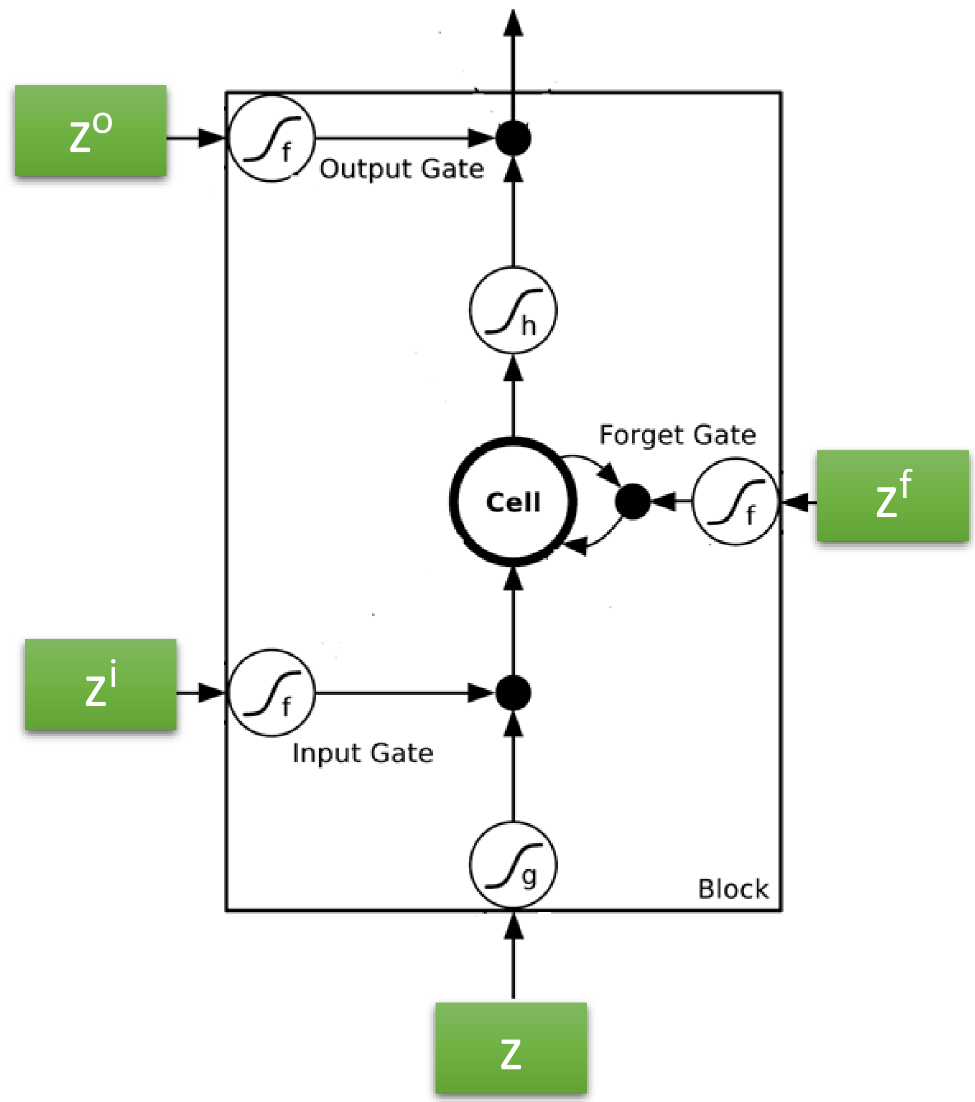


c^{t-1}
vector
Output Gate
Forget Gate
Cell
Input Gate
Block
……
z^f
z^i
z
z^o
4 vectors
x^t

LSTM



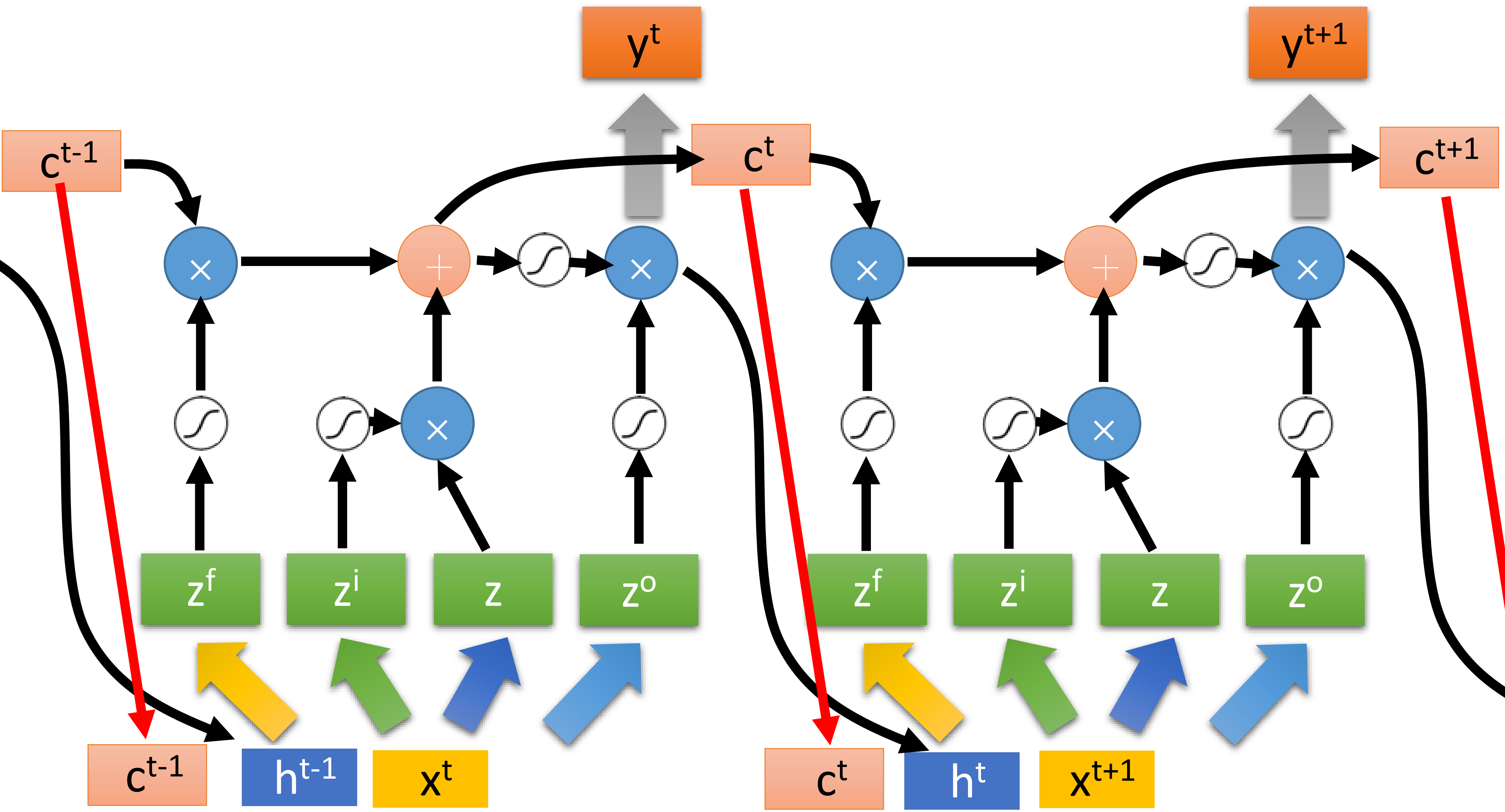
y^t
c^{t-1}
z^f
z^i
z
z^o
x^t



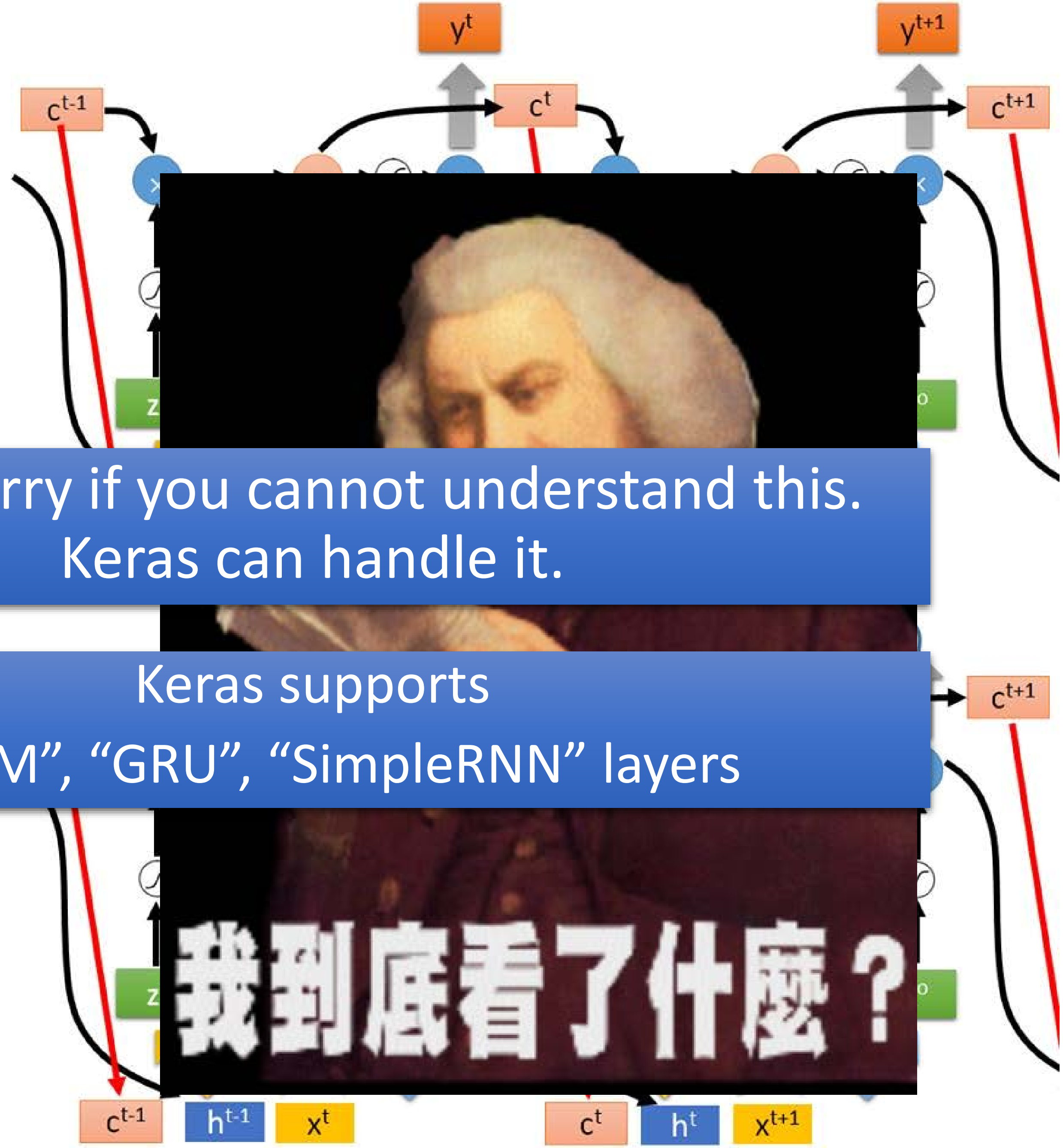
z^o
Output Gate
Forget Gate
Cell
z^f
z^i
Input Gate
Block
z

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.

https://img.komicolle.org/2015-09-20/src/1442967627131.gif

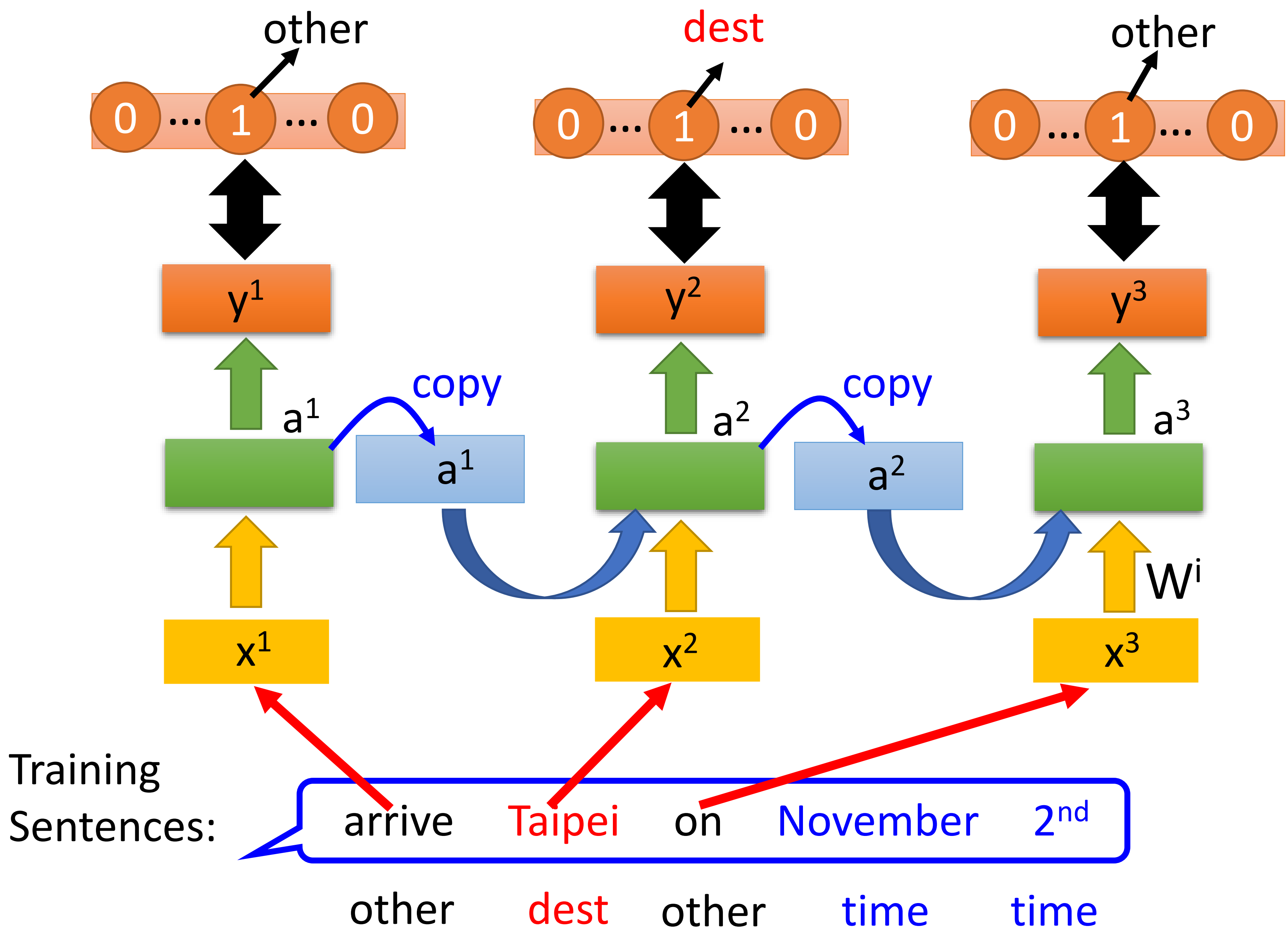
Recurrent Neural Network

Step 1: Neural Network → **Step 2: Learning Target** → **Step 3: Learn!**

天生的腦

http://onepiece1234567890.blogspot.tw/2013/12/blog-post_8.html

Learning Target



Recurrent Neural Network

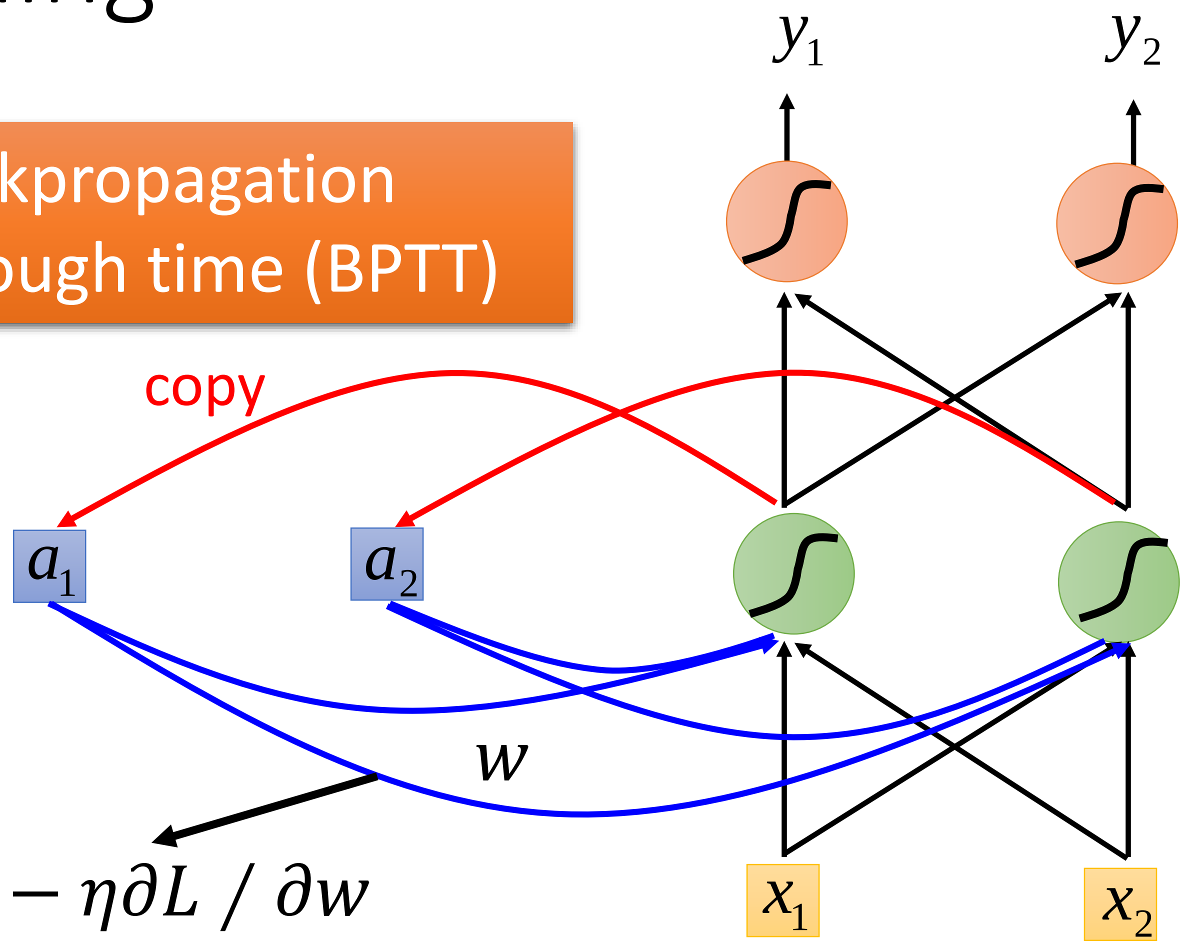
Step 1: Neural Network → Step 2: Learning Target → **Step 3: Learn!**

天生的腦

http://onepiece1234567890.blogspot.tw/2013/12/blog-post_8.html

Learning

Backpropagation through time (BPTT)



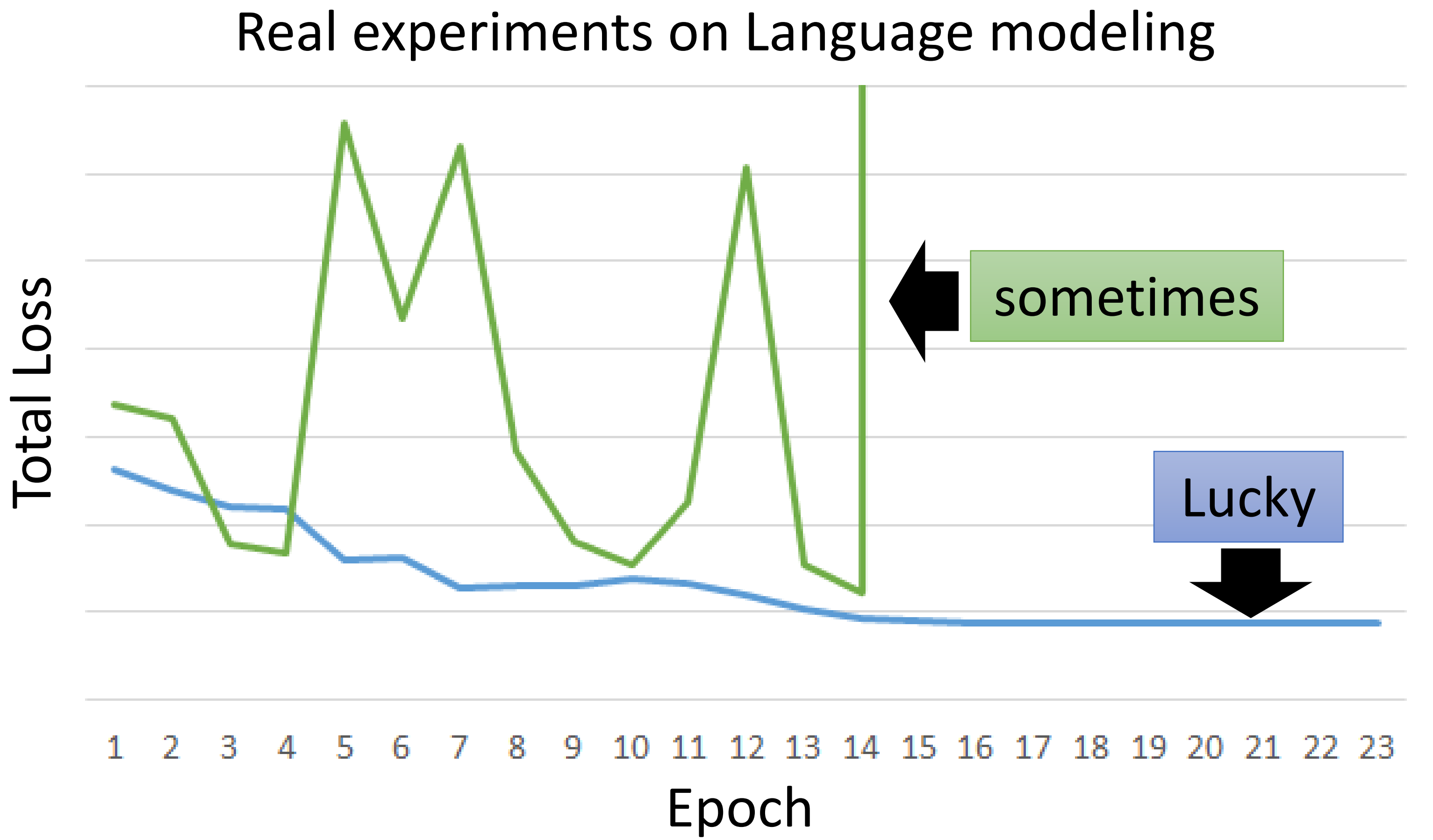
$$w \leftarrow w - \eta \partial L / \partial w$$

RNN Learning is very difficult in practice.

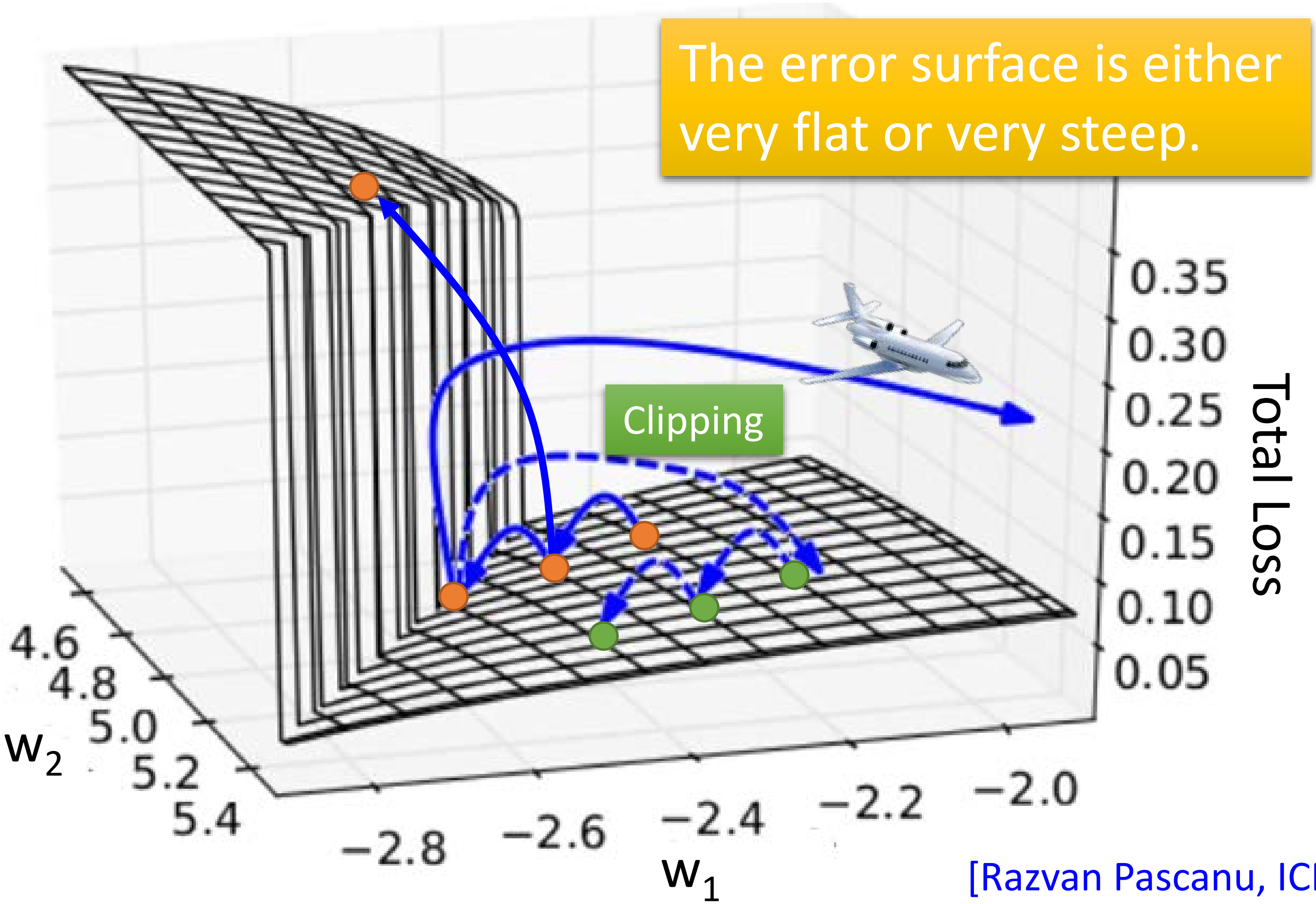
感謝 曾柏翔 同學
提供實驗結果

Unfortunately ……

- RNN-based network is not always easy to learn



The error surface is rough.



[Razvan Pascanu, ICML'13]

Why?

$w = 1 \Rightarrow y^{1000} = 1$

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

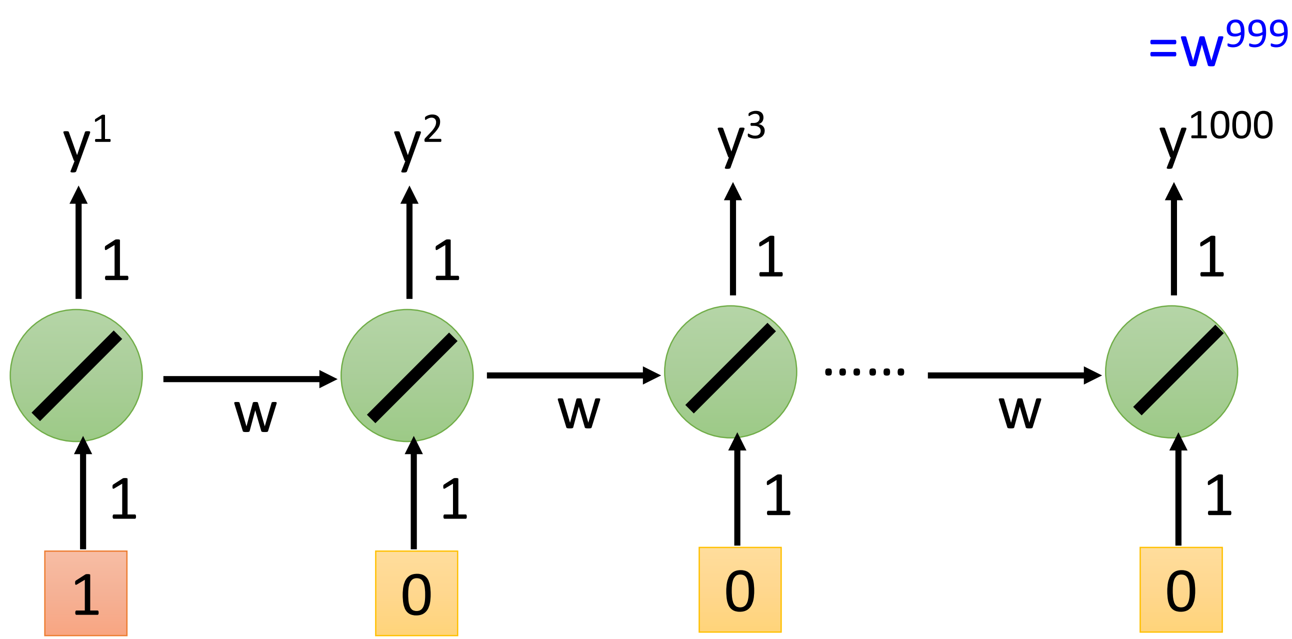
Large $\partial L / \partial w$ → Small Learning rate?

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

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

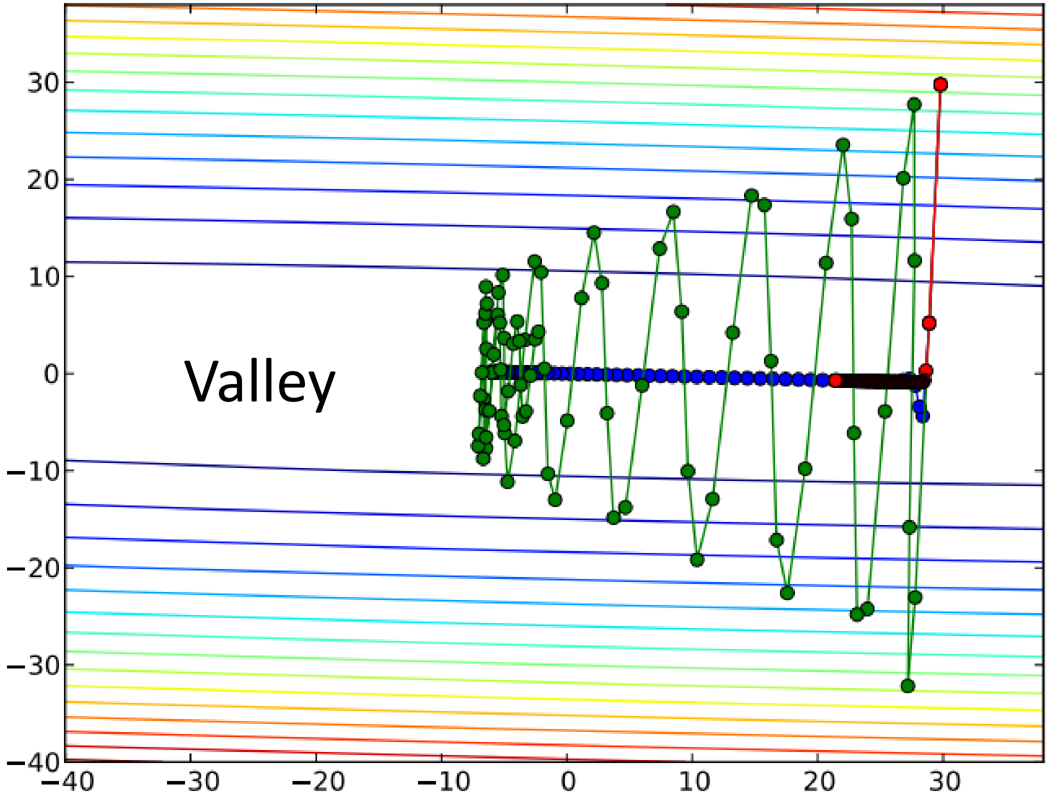
small $\partial L / \partial w$ → Large Learning rate?

Toy Example



Helpful Techniques

- Advance momentum method
  - Nesterov's Accelerated Gradient (NAG)



Methods:

- Gradient descent
- Momentum
- Nesterov's Accelerated Gradient (NAG)

Source: http://www.cs.toronto.edu/~fritz/absps/momentum.pdf

Helpful Techniques

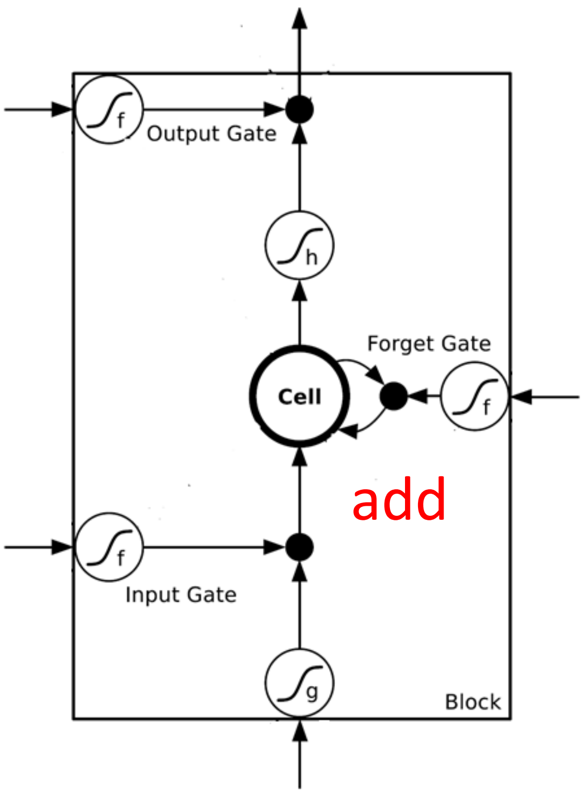
- Long Short-term Memory (LSTM)
  - Can deal with gradient vanishing (not gradient explode)

➢ Memory and input are ***added***

➢ The influence never disappears unless forget gate is closed

No Gradient vanishing

(If forget gate is opened.)



Helpful Techniques

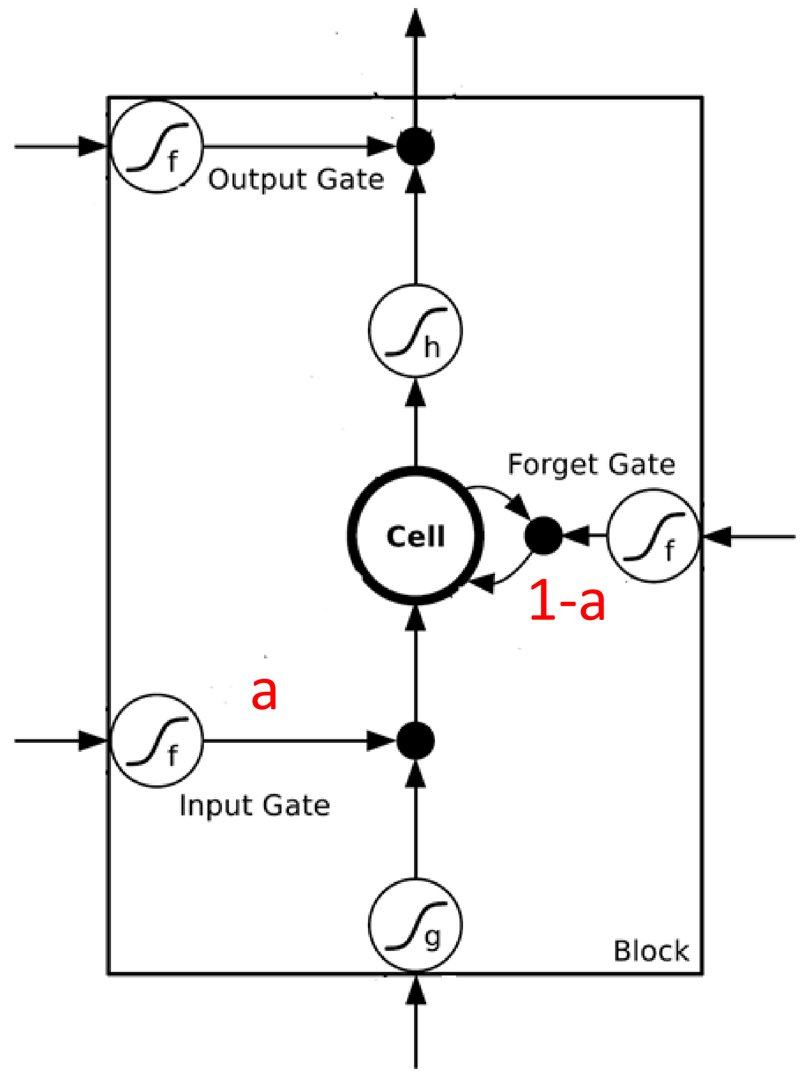
- Gated Recurrent Unit (GRU)

Simplified LSTM

[Cho, EMNLP’14]

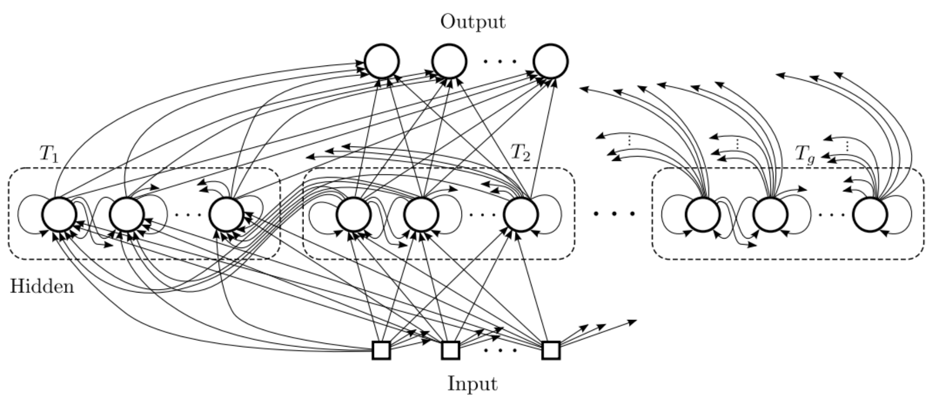
舊的不去、新的不來

GRU has less parameters than LSTM



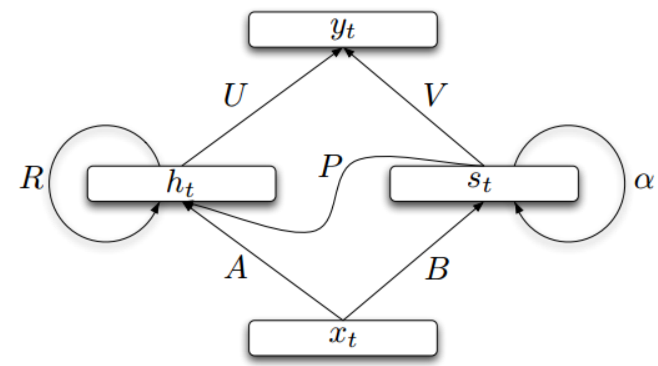
Helpful Techniques

Clockwise RNN



[Jan Koutnik, JMLR'14]

Structurally Constrained Recurrent Network (SCRN)

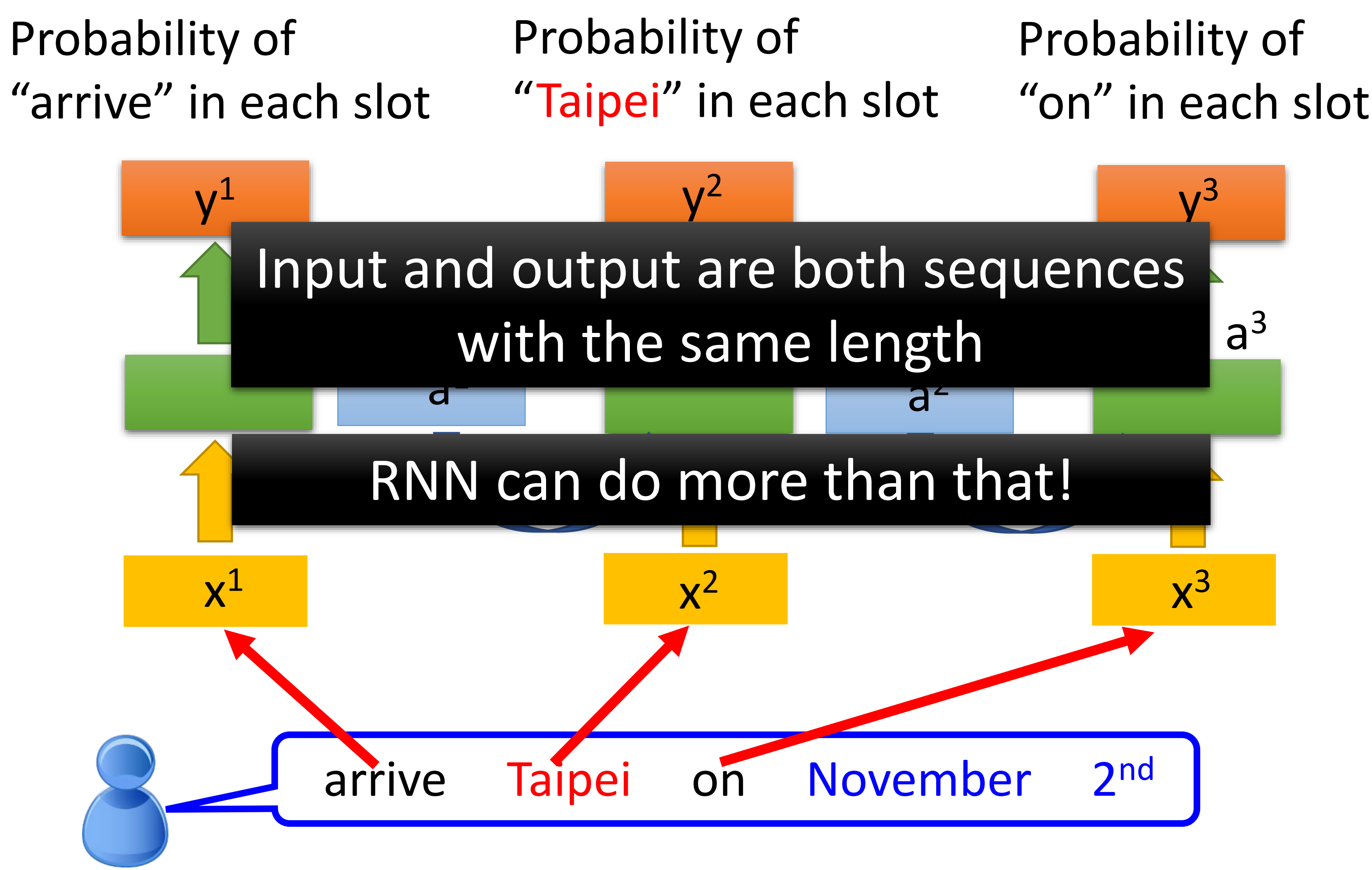


[Tomas Mikolov, ICLR'15]

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 ......

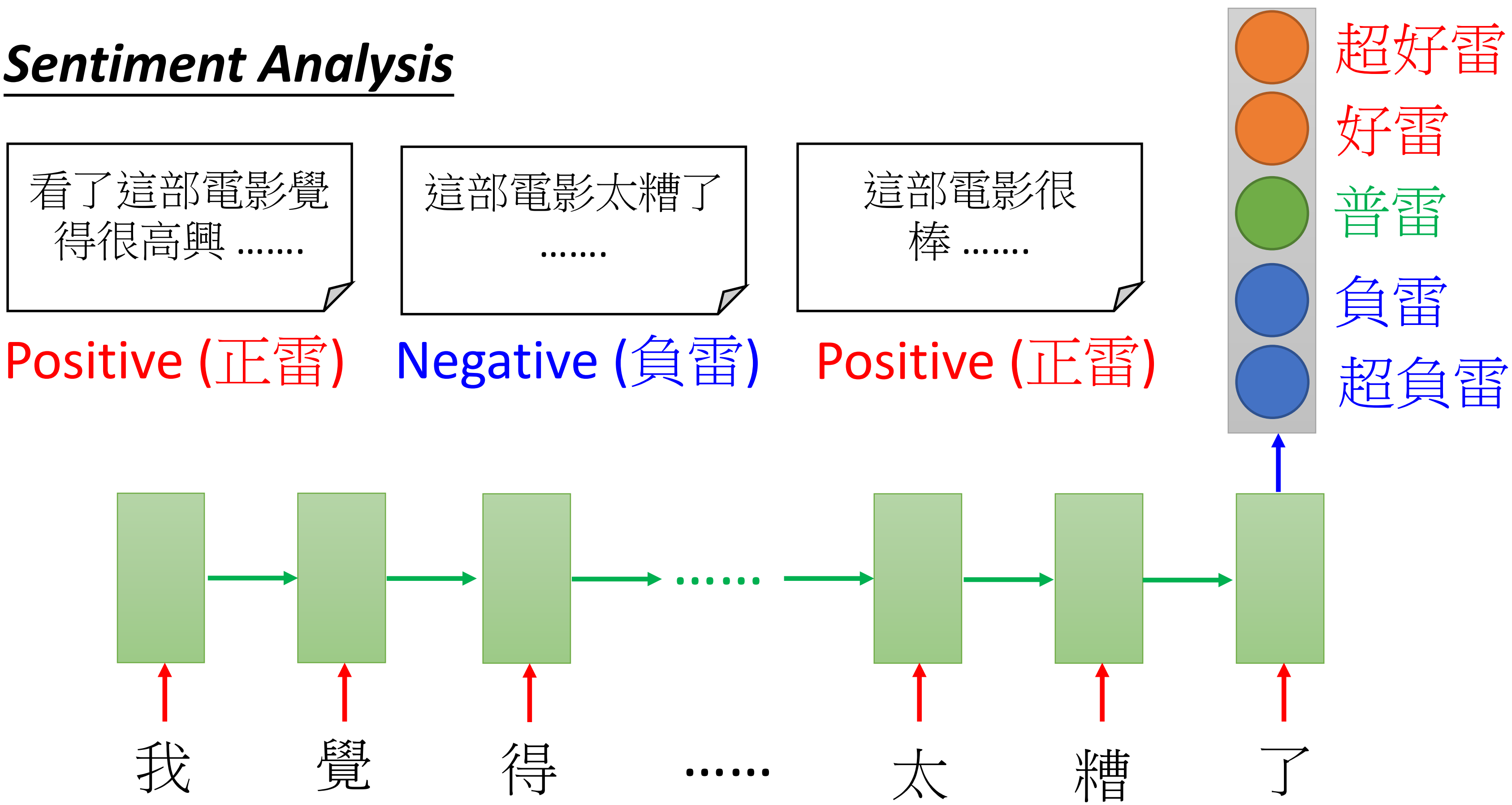


Many to one

Keras Example: https://github.com/fchollet/keras/blob/master/examples/imdb_lstm.py

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

Sentiment Analysis

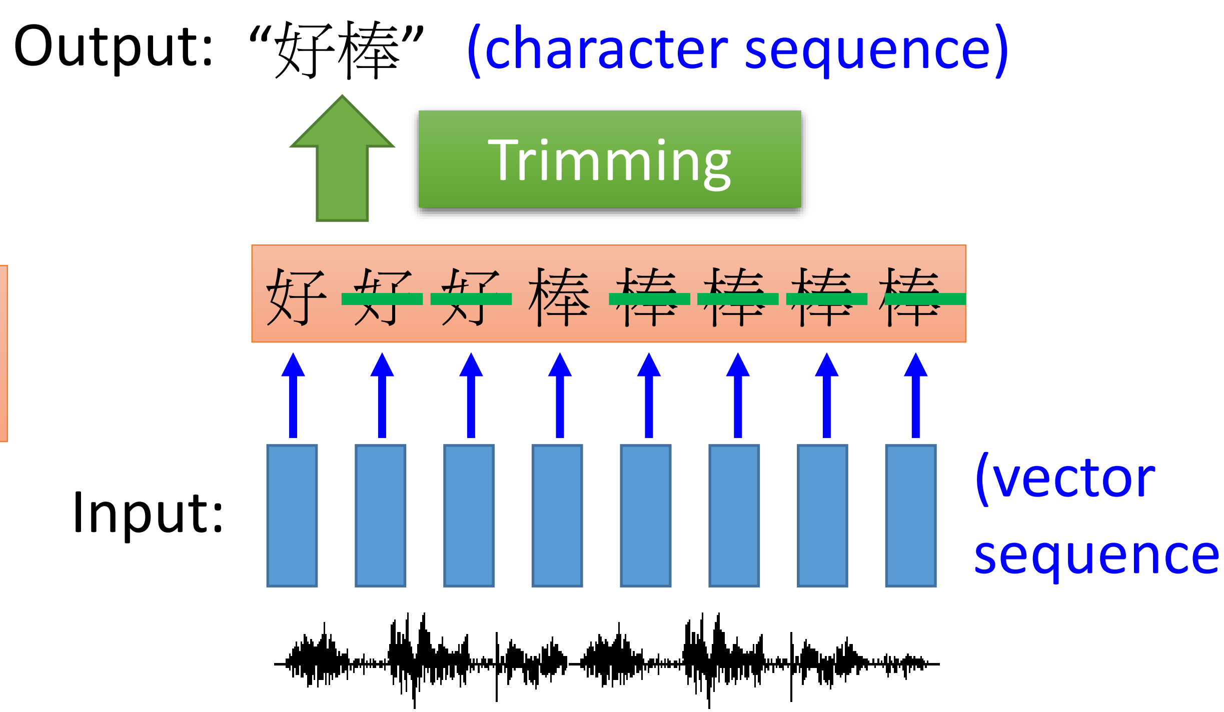


Many to Many (Output is shorter)

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

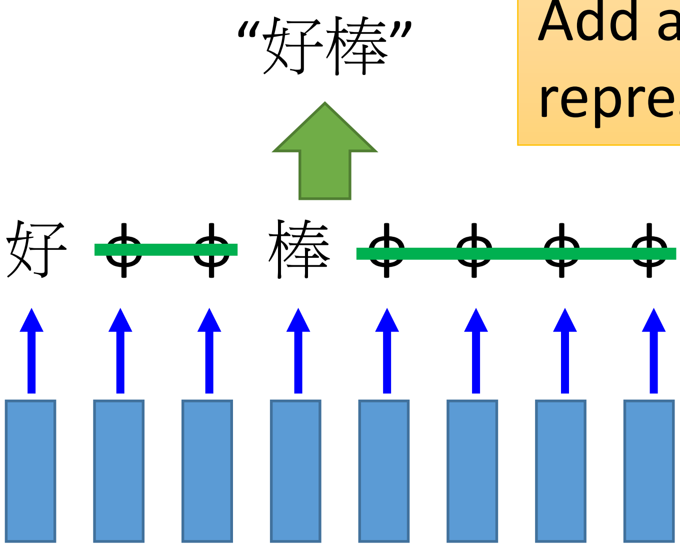
Problem?

Why can't it be "好棒棒"

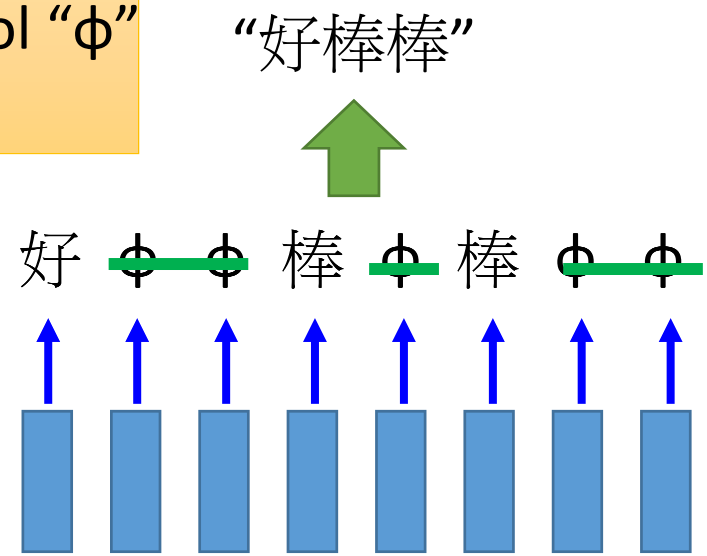


Many to Many (Output is shorter)

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

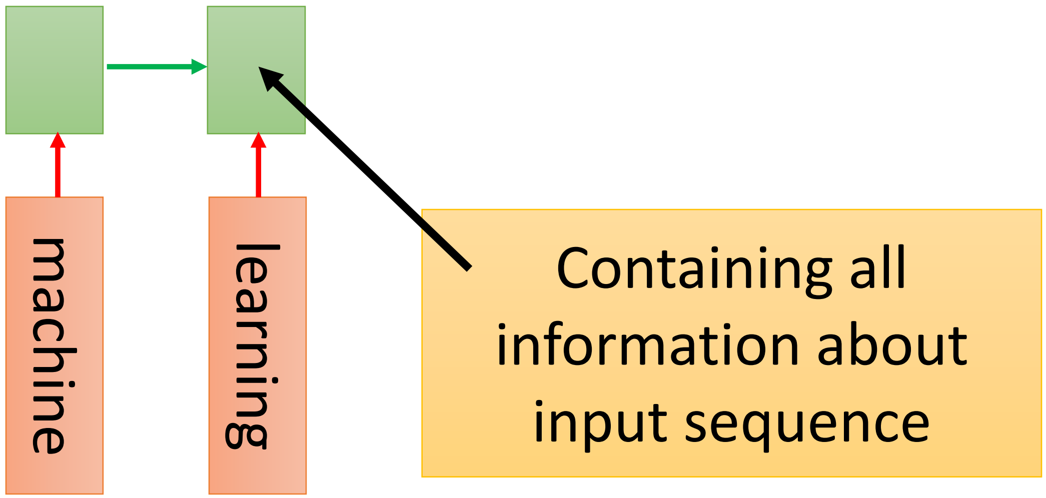


Add an extra symbol “φ” representing “null”



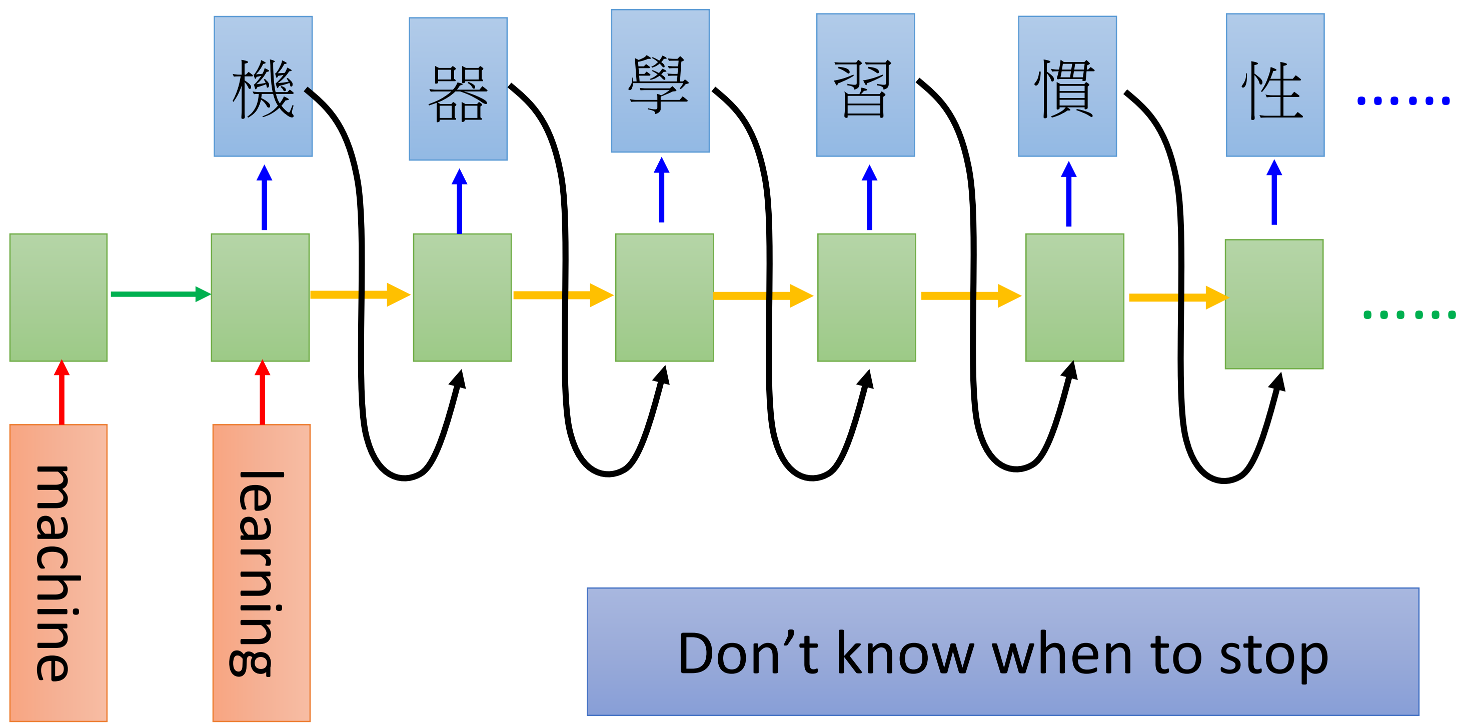
Many to Many (No Limitation)

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

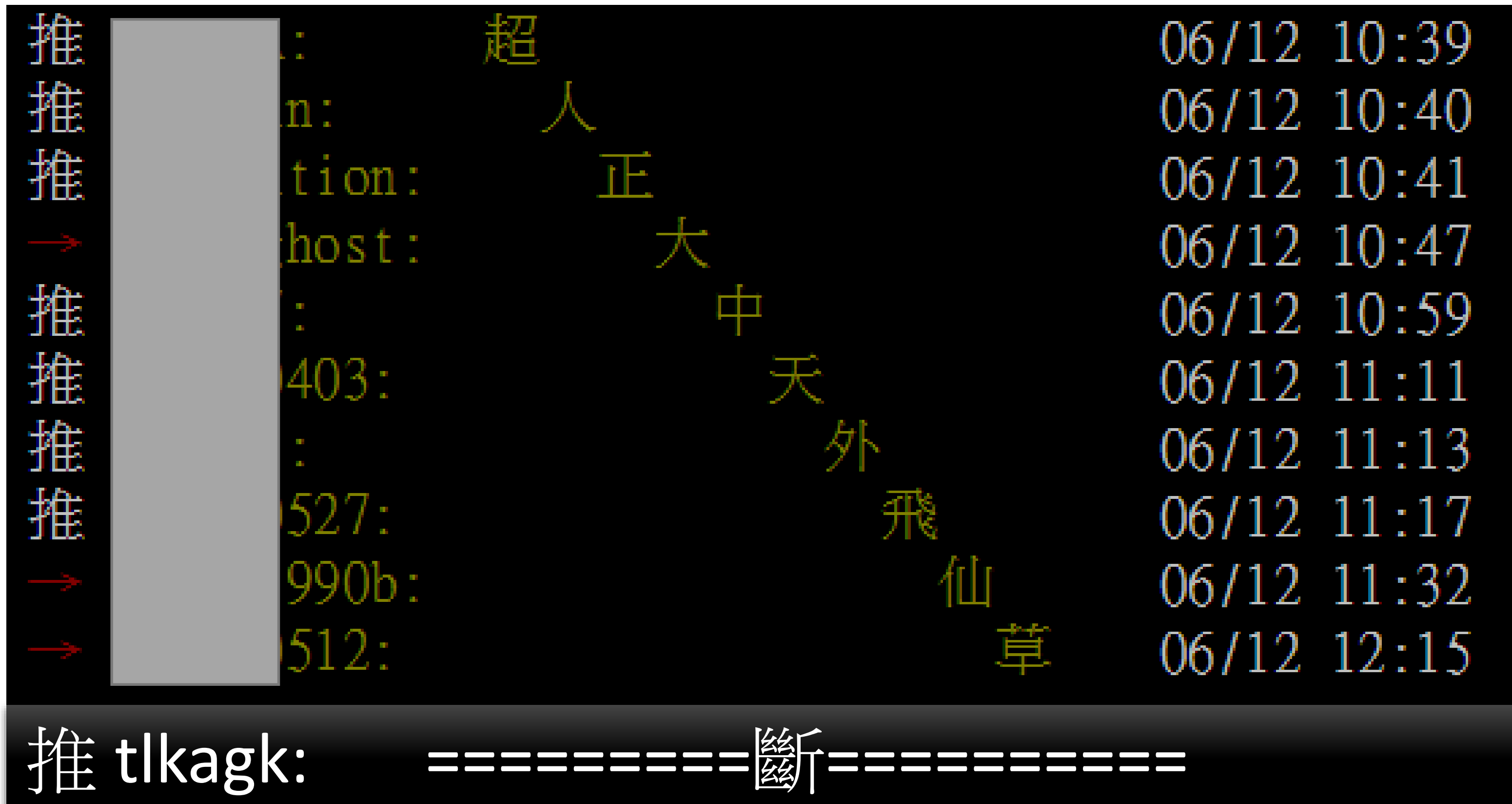


Many to Many (No Limitation)

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



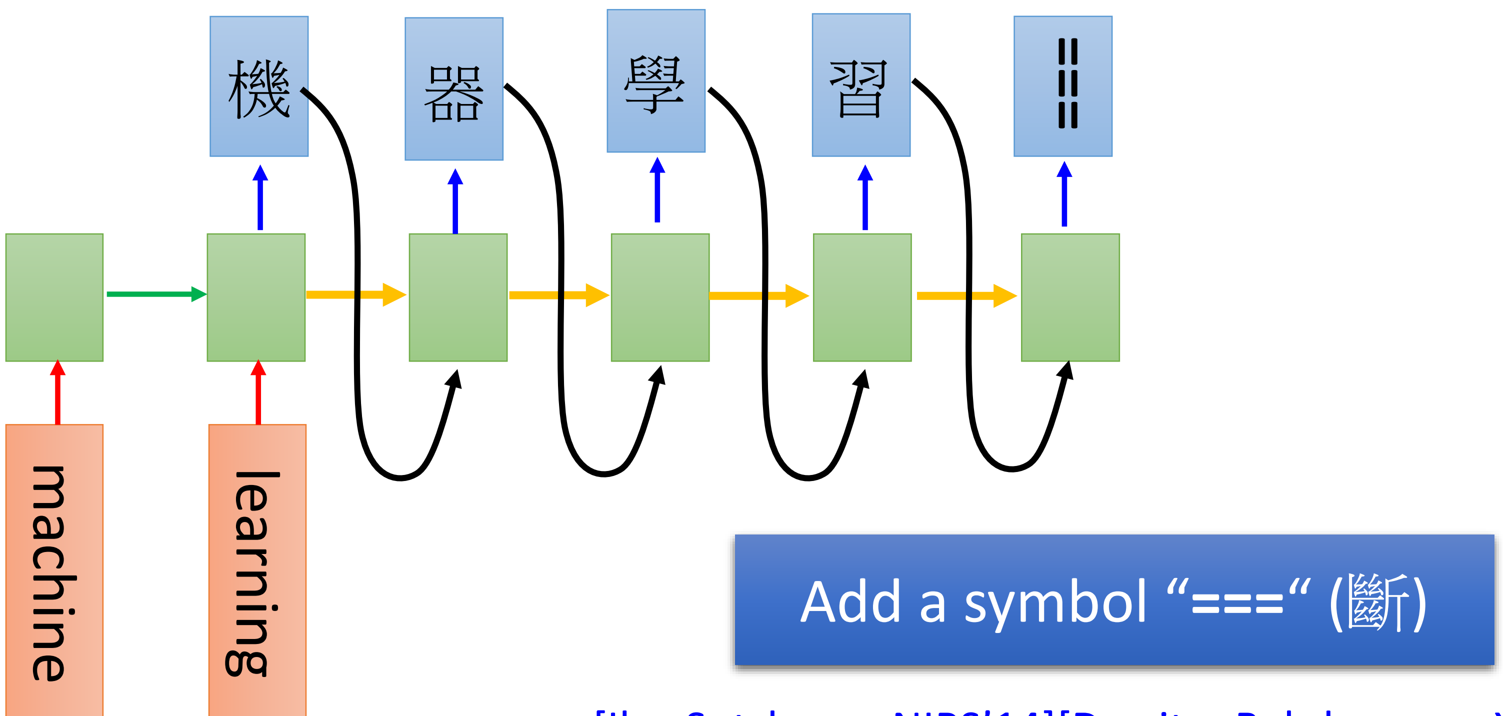
Many to Many (No Limitation)



Ref:http://zh.pttpedia.wikia.com/wiki/%E6%8E%A5%E9%BE%8D%E6%8E%A8%E6%96%87 (鄉民百科)

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)

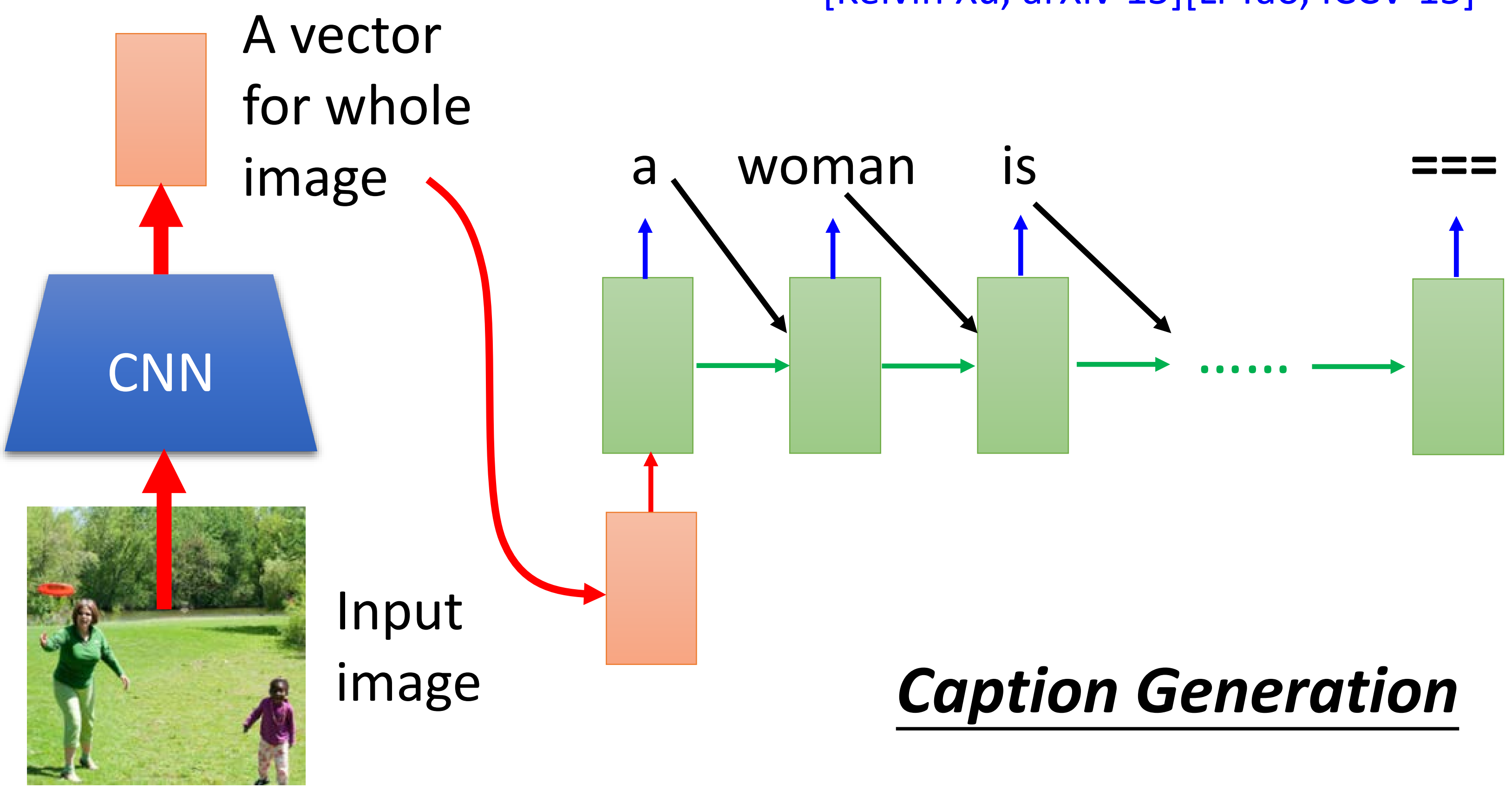


[Ilya Sutskever, NIPS'14][Dzmitry Bahdanau, arXiv'15]

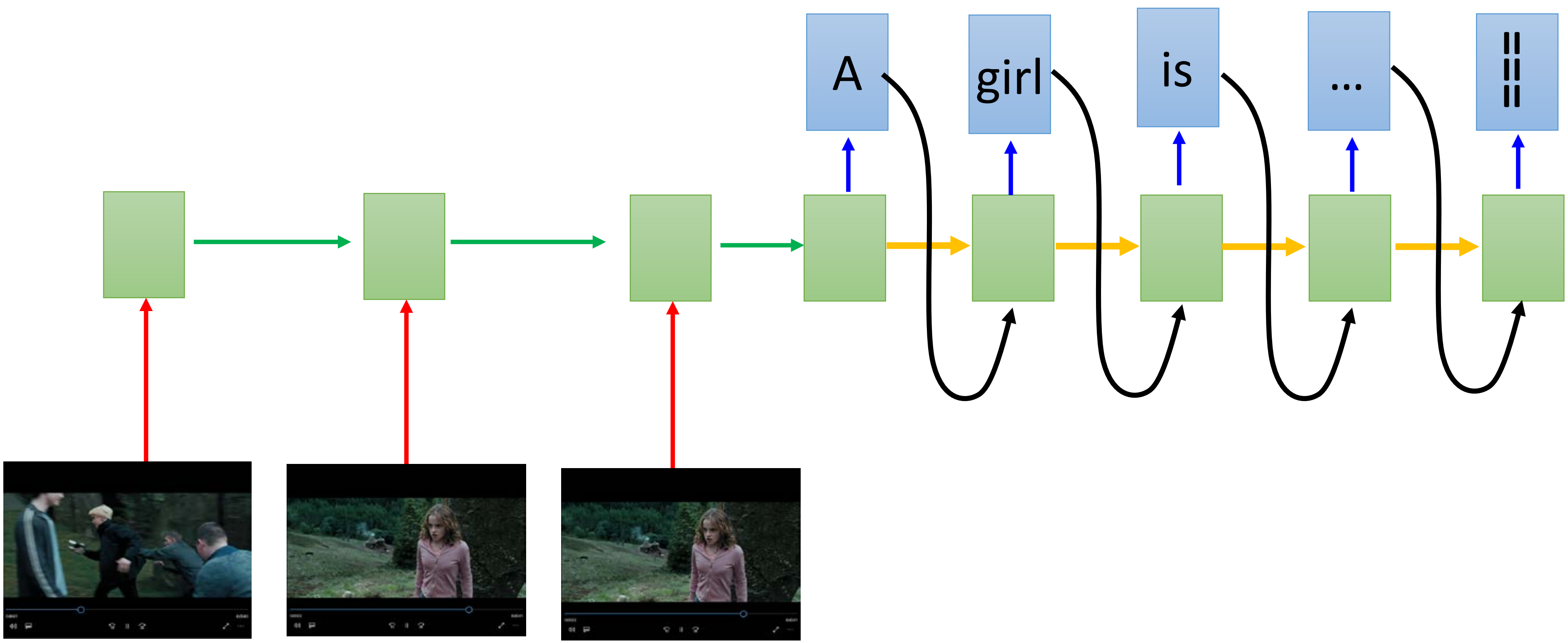
One to Many

- Input an image, but output a sequence of words

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



Video Caption Generation



Video frames

Concluding Remarks

Convolutional Neural Network (CNN)

Recurrent Neural Network (RNN)

Lecture IV:
Next Wave

Outline

- Ultra Deep Network
- Attention Model
- Reinforcement Learning
- Realizing what the World Looks Like
- Understanding the Meaning of Words
- Why Deep?

Ultra Deep Network

Worry about overfitting?

Worry about achieving target first!

This ultra deep network have special structure.

16.4%

AlexNet (2012)

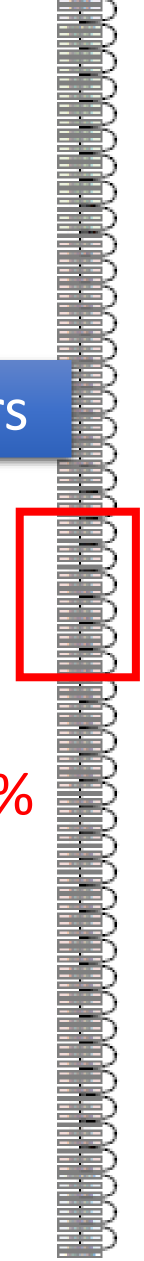
7.3% 

VGG (2014)

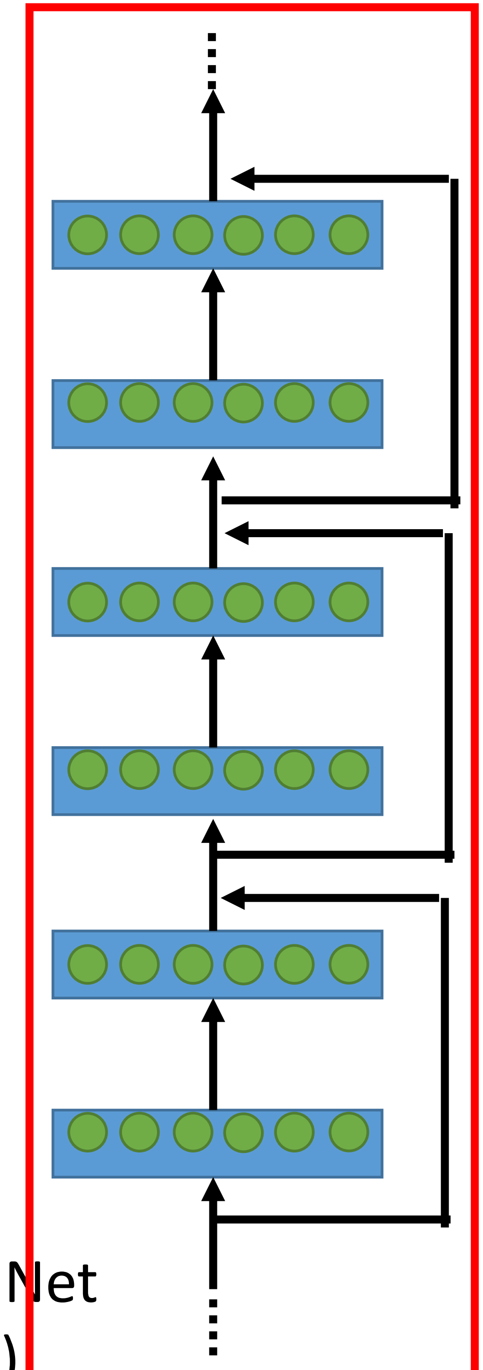
6.7% 

GoogleNet (2014)

152 layers

3.57% 

Residual Net (2015)

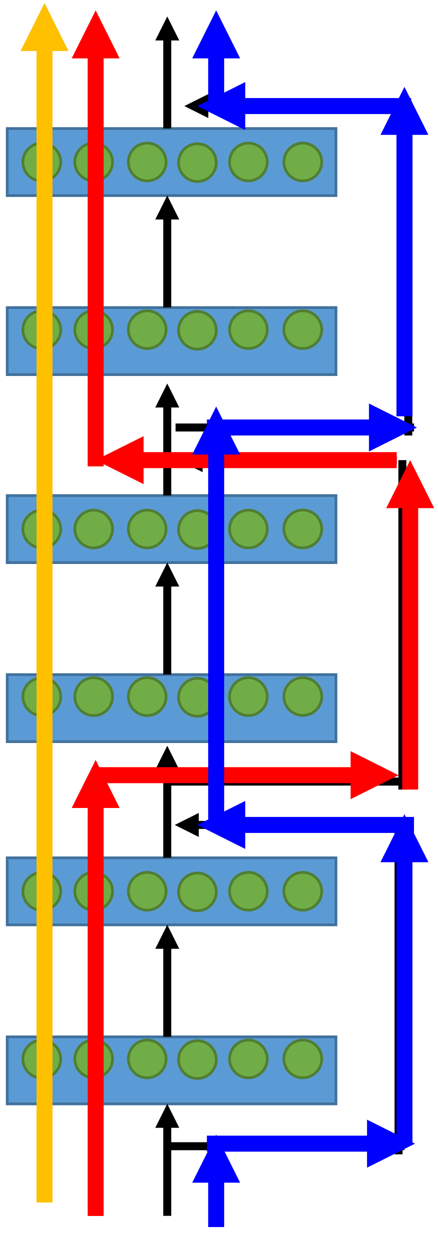


Ultra Deep Network

- 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

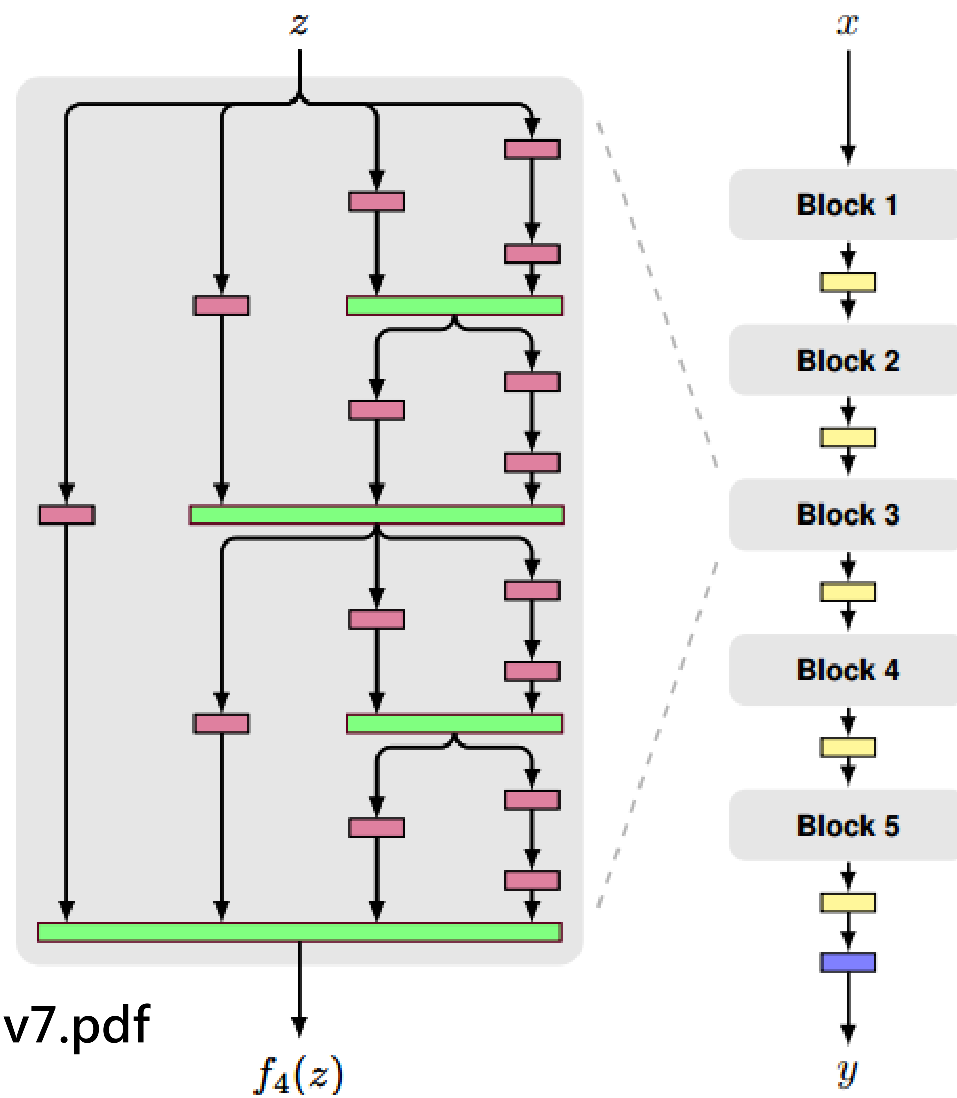
FractalNet: Ultra-Deep Neural Networks without Residuals
https://arxiv.org/abs/1605.07648

Resnet in Resnet

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

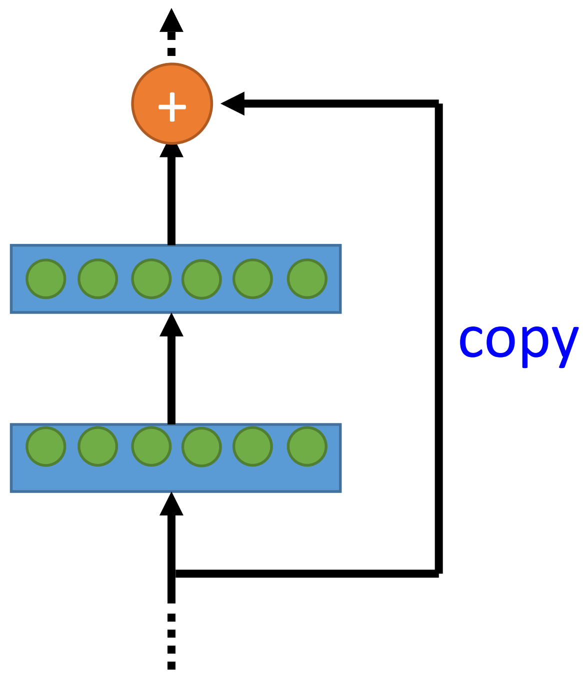
Good Initialization?

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



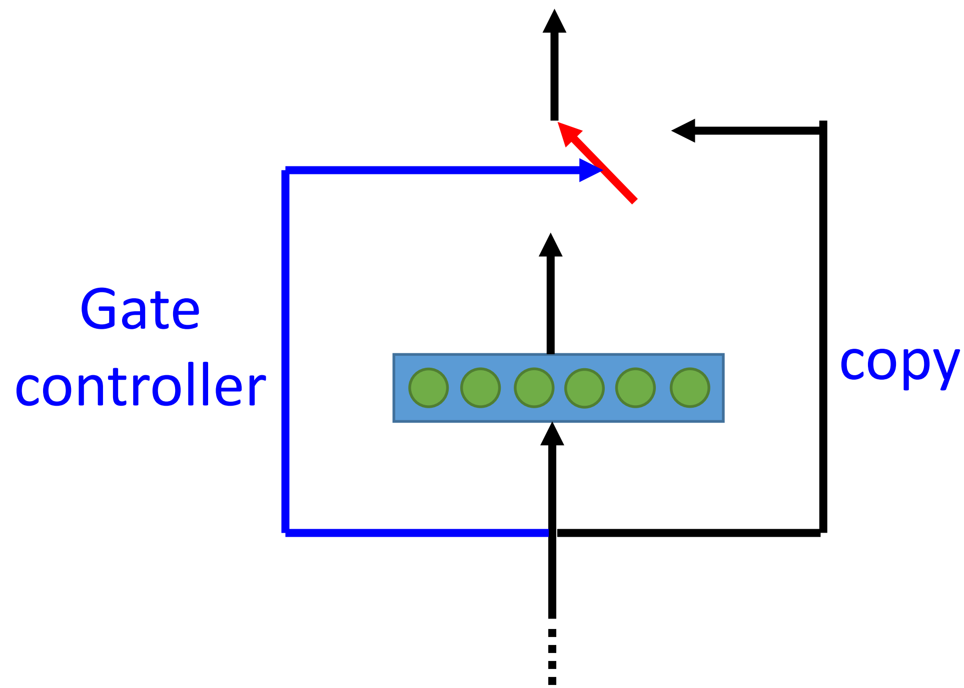
Ultra Deep Network

- **Residual Network**

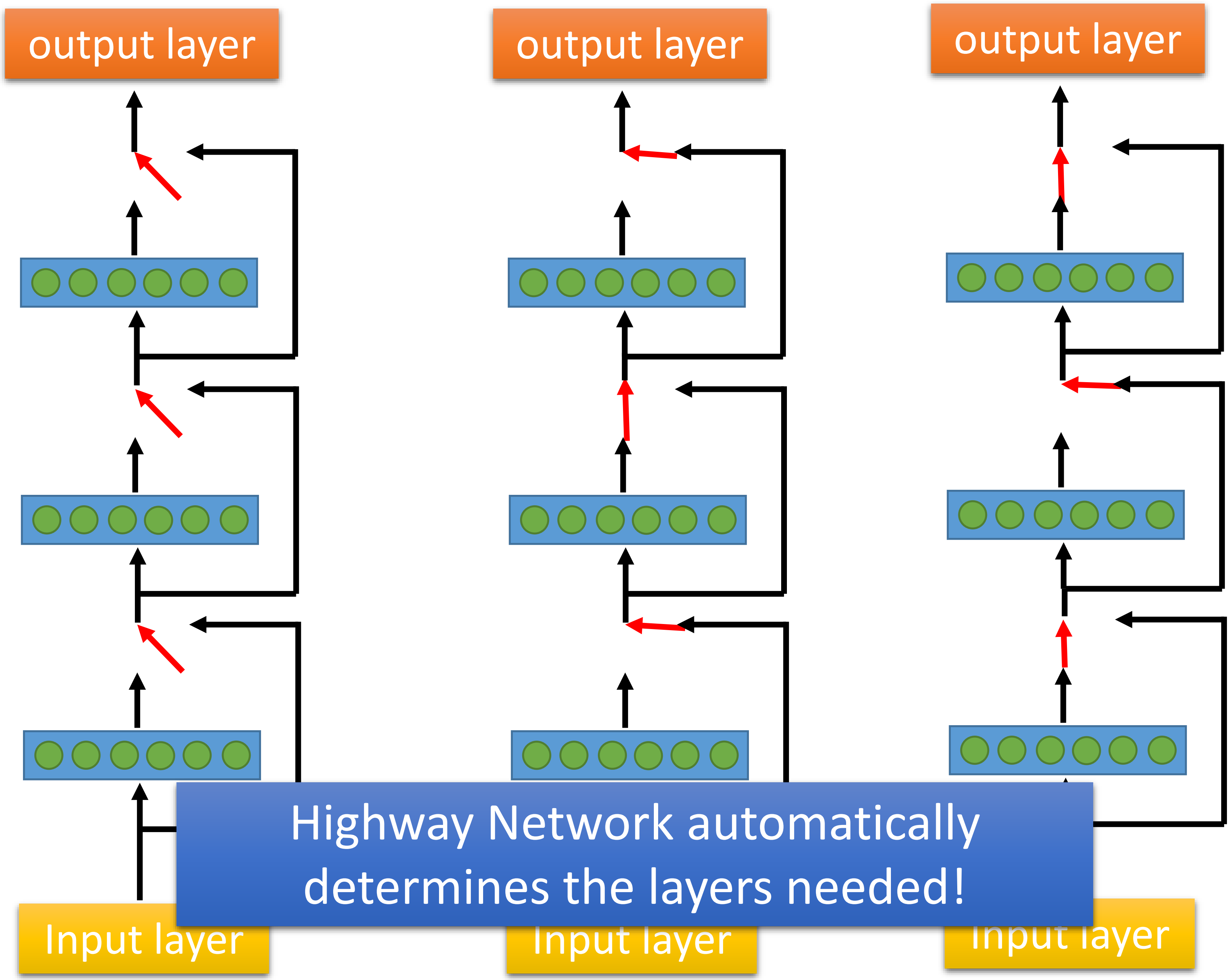


Deep Residual Learning for Image Recognition
http://arxiv.org/abs/1512.03385

- **Highway Network**



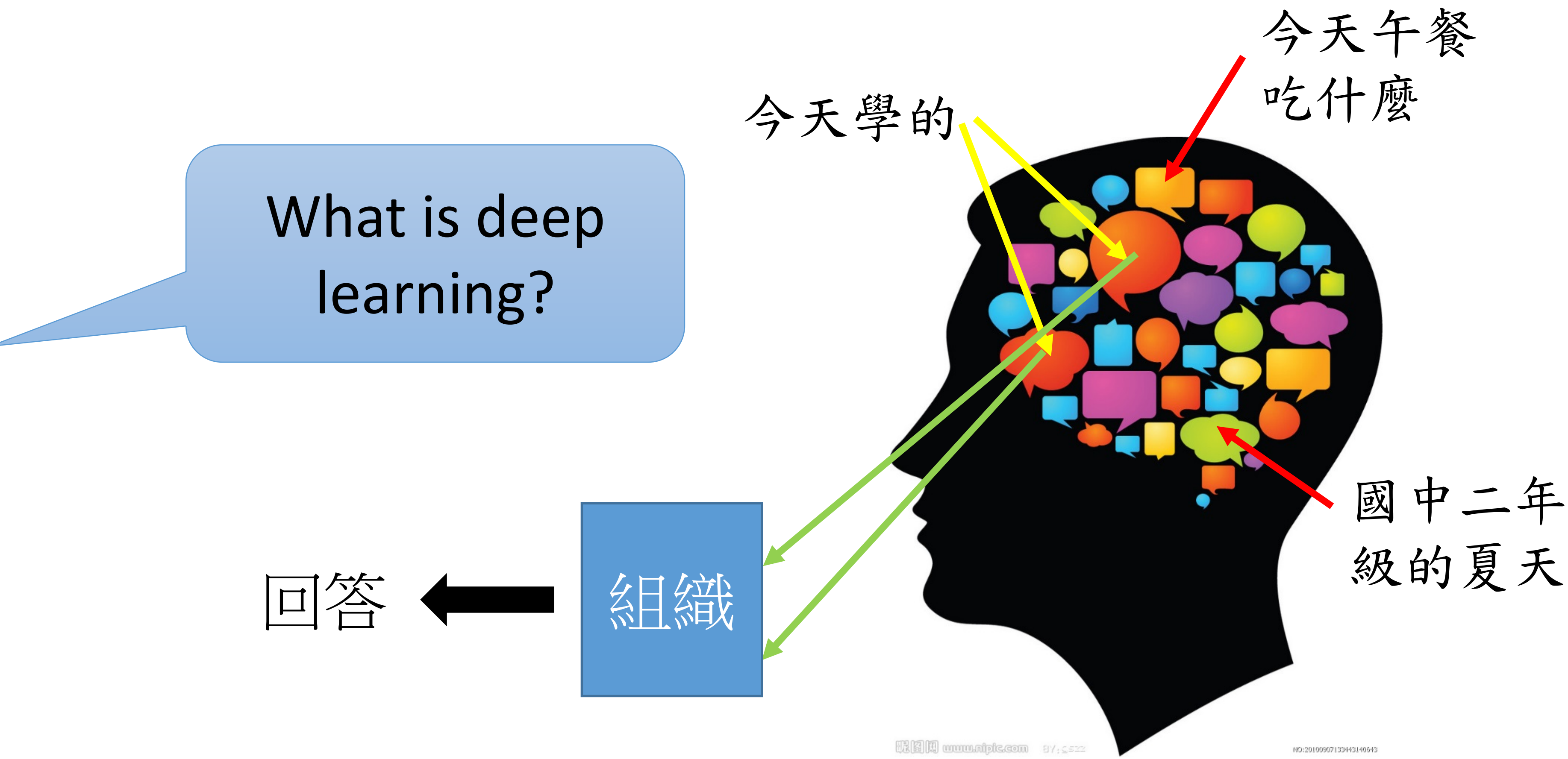
Training Very Deep Networks
https://arxiv.org/pdf/1507.06228v2.pdf


output layer
output layer
output layer
Input layer
Input layer
Input layer
Highway Network automatically determines the layers needed!

Outline

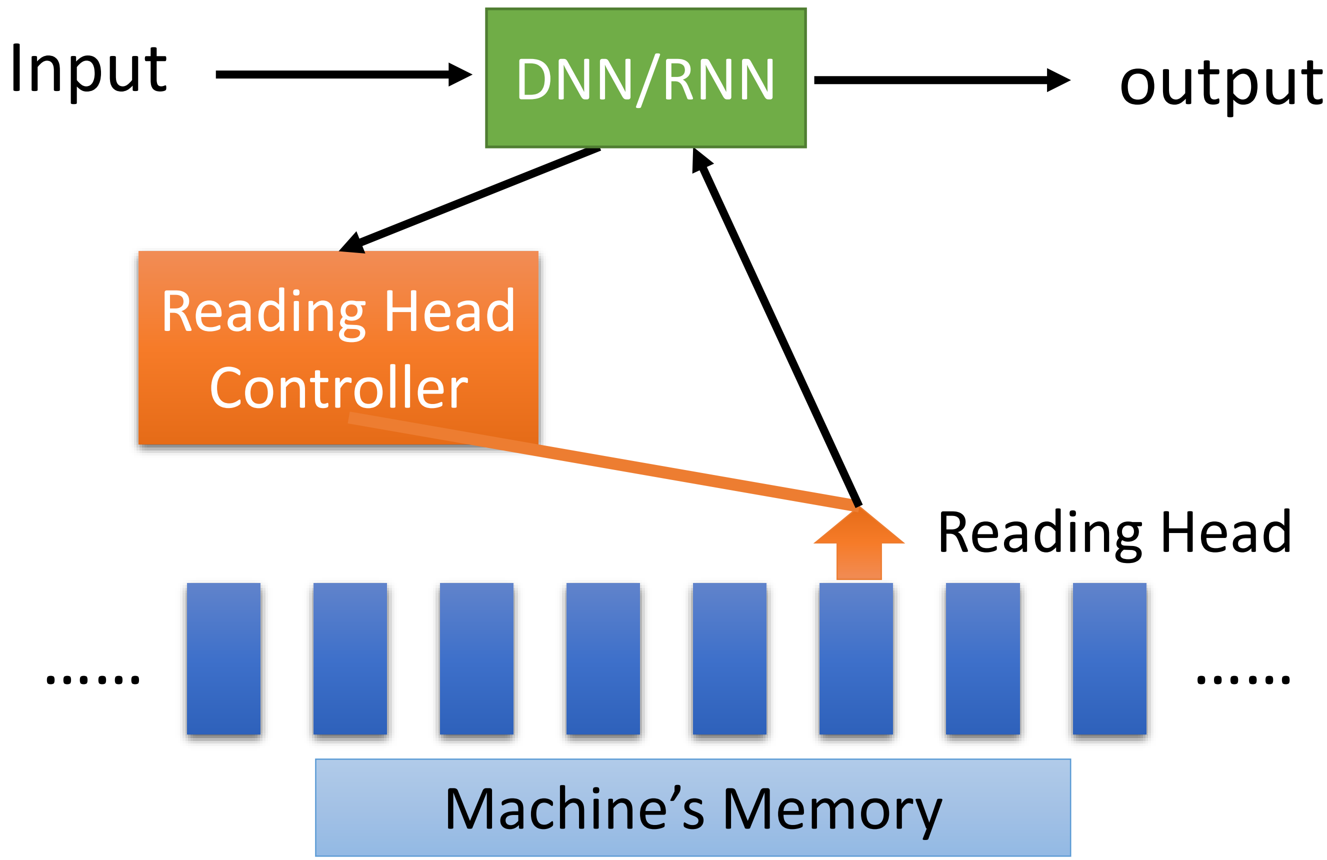
- Ultra Deep Network
- Attention Model
- Reinforcement Learning
- Realizing what the World Looks Like
- Understanding the Meaning of Words
- Why Deep?

Attention-based Model



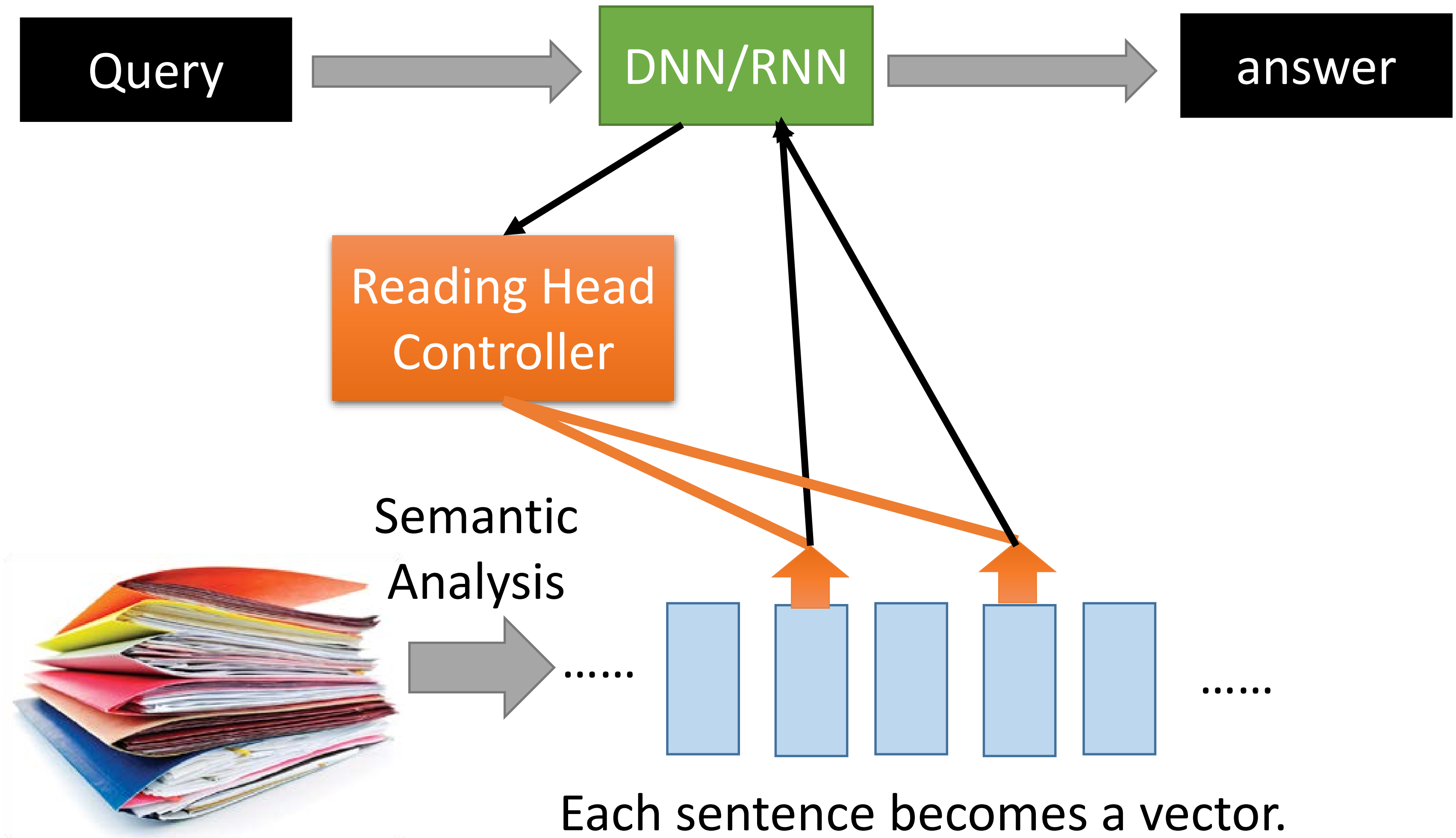
http://henrylo1605.blogspot.tw/2015/05/blog-post_56.html

Attention-based Model



Ref:
http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/Attain%20(v3).ecm.mp4/index.html

Reading Comprehension



Reading Comprehension

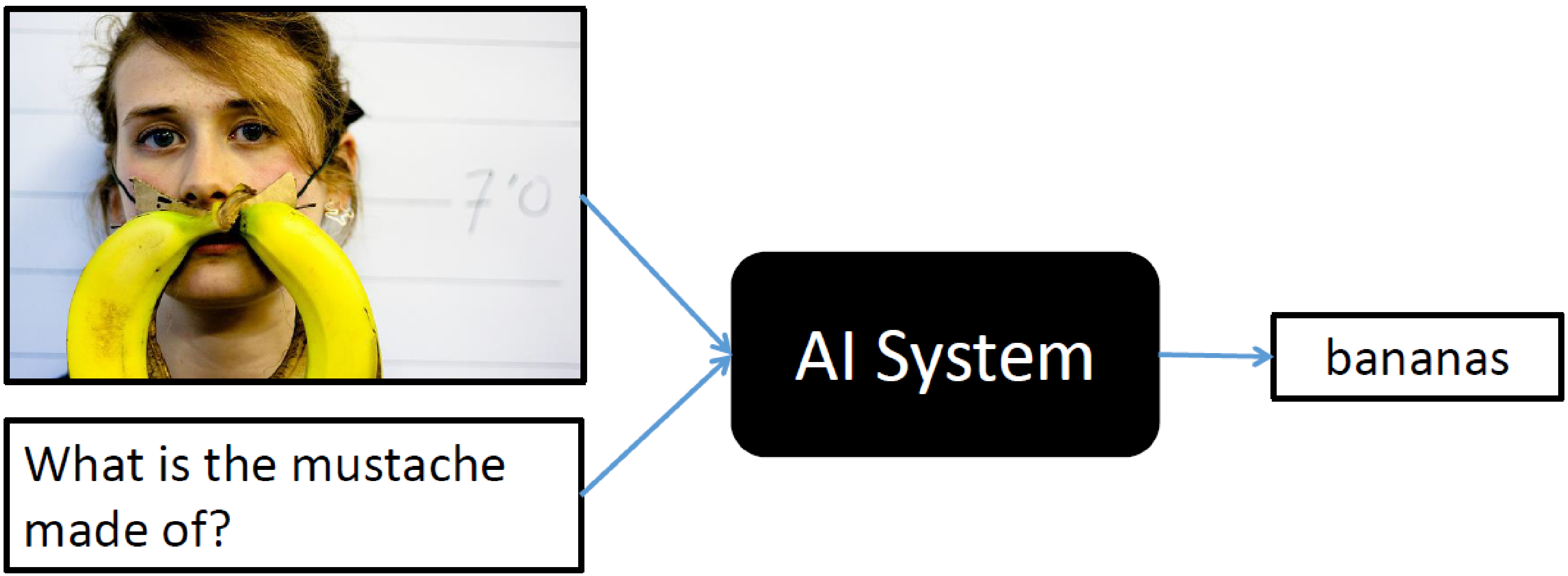
- End-To-End Memory Networks. S. Sukhbaatar, A. Szlam, J. Weston, R. Fergus. NIPS, 2015.

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				

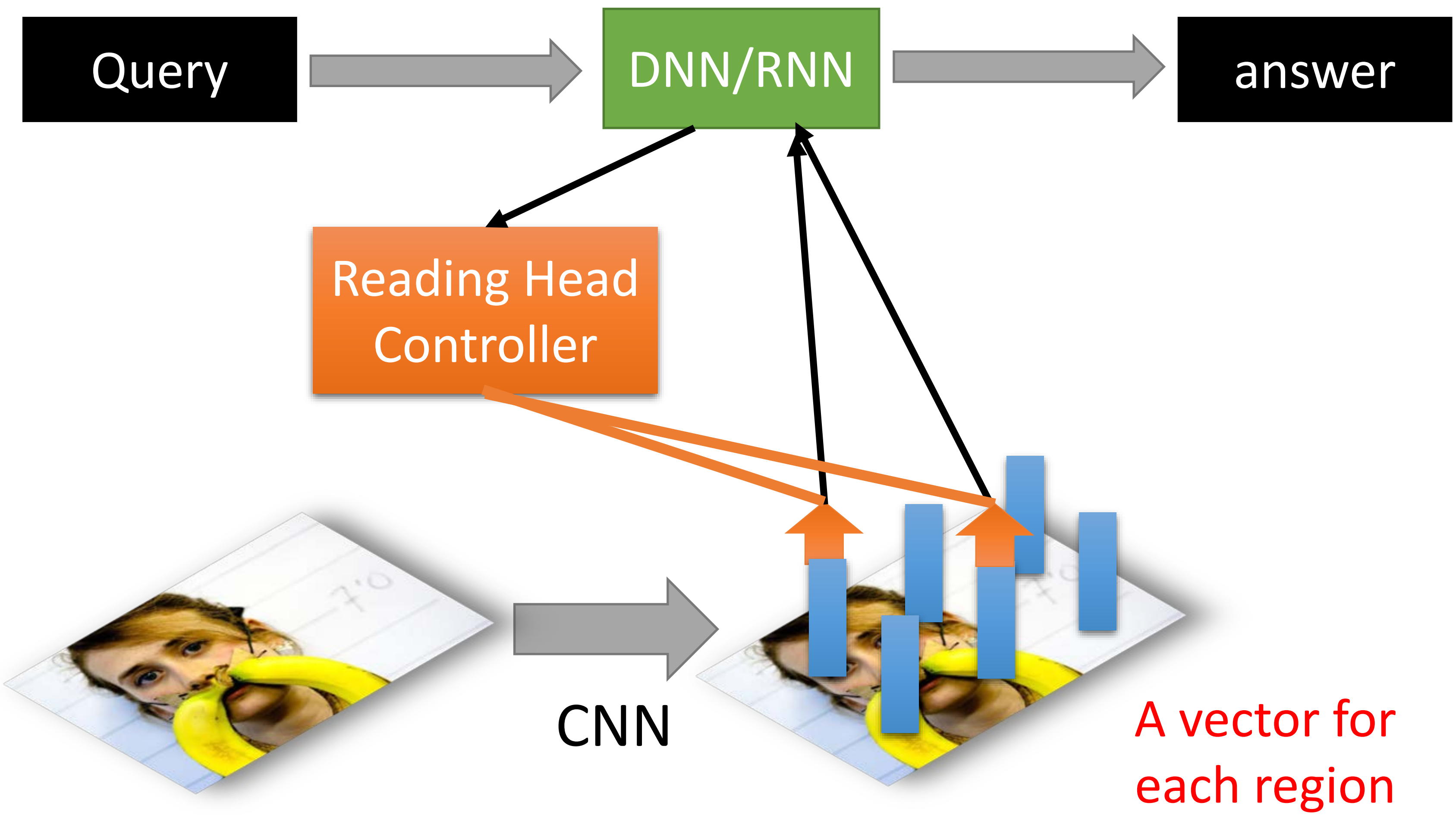
Keras has example:
https://github.com/fchollet/keras/blob/master/examples/babi_memnn.py

Visual Question Answering



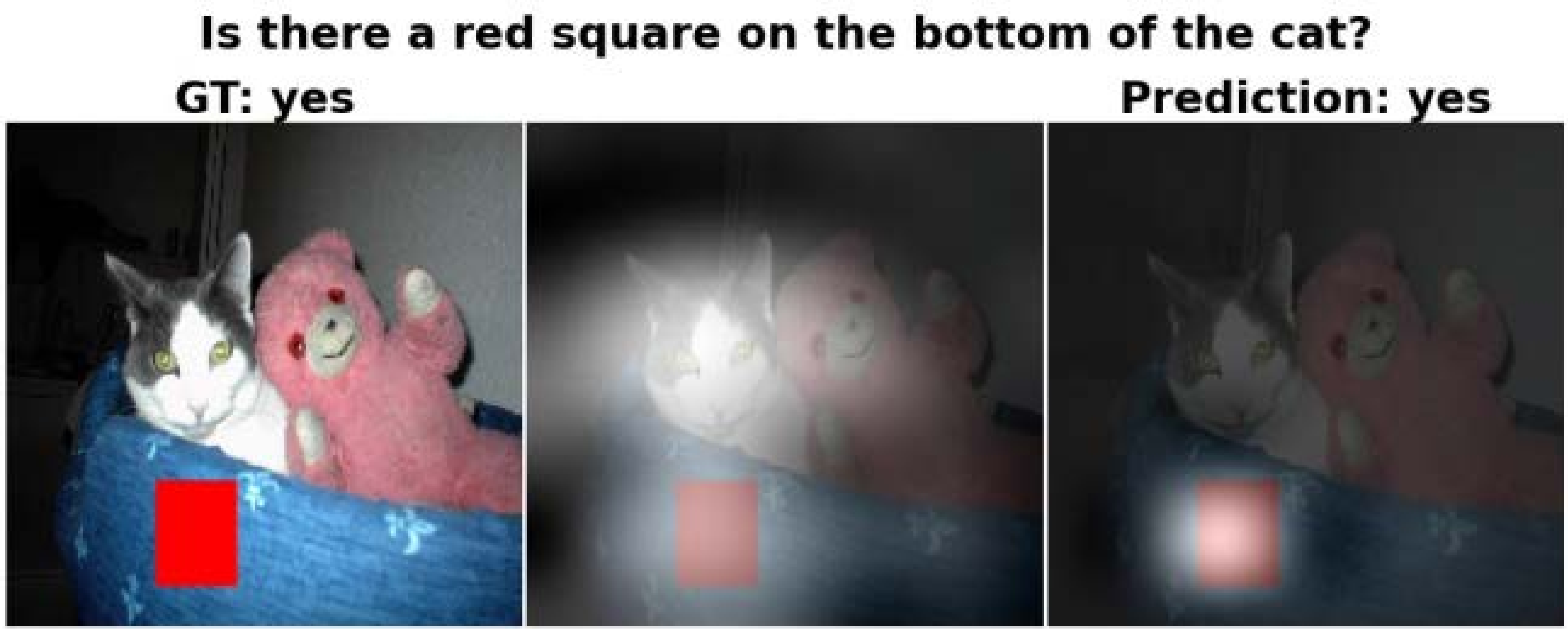
source: http://visualqa.org/

Visual Question Answering



Visual Question Answering

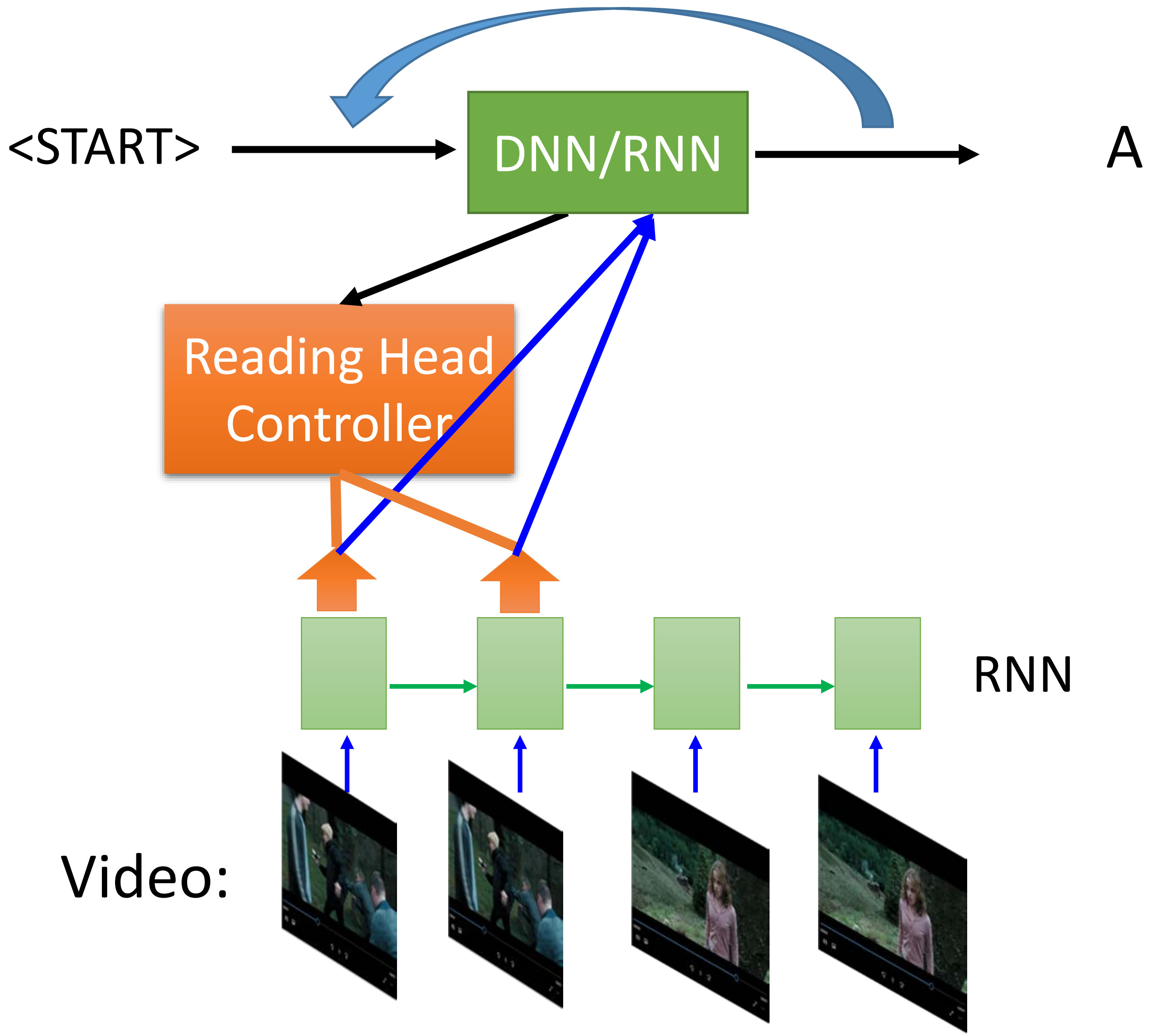
- Huijuan Xu, Kate Saenko. Ask, Attend and Answer: Exploring Question-Guided Spatial Attention for Visual Question Answering. arXiv Pre-Print, 2015



Video Caption Generation

Memory: video frames

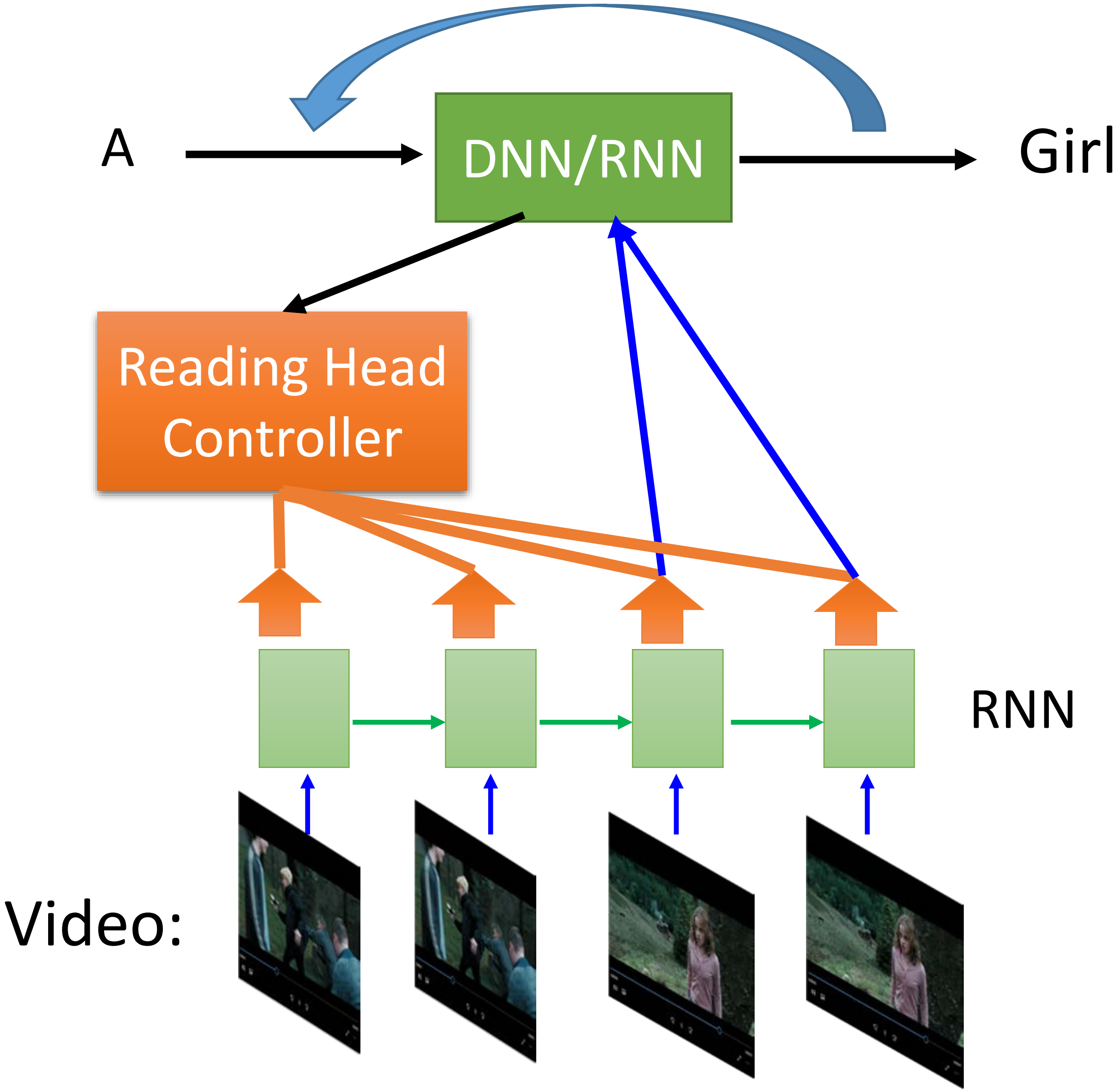
Output: video description



Video Caption Generation

Memory: video frames

Output: video description



Video Caption Generation

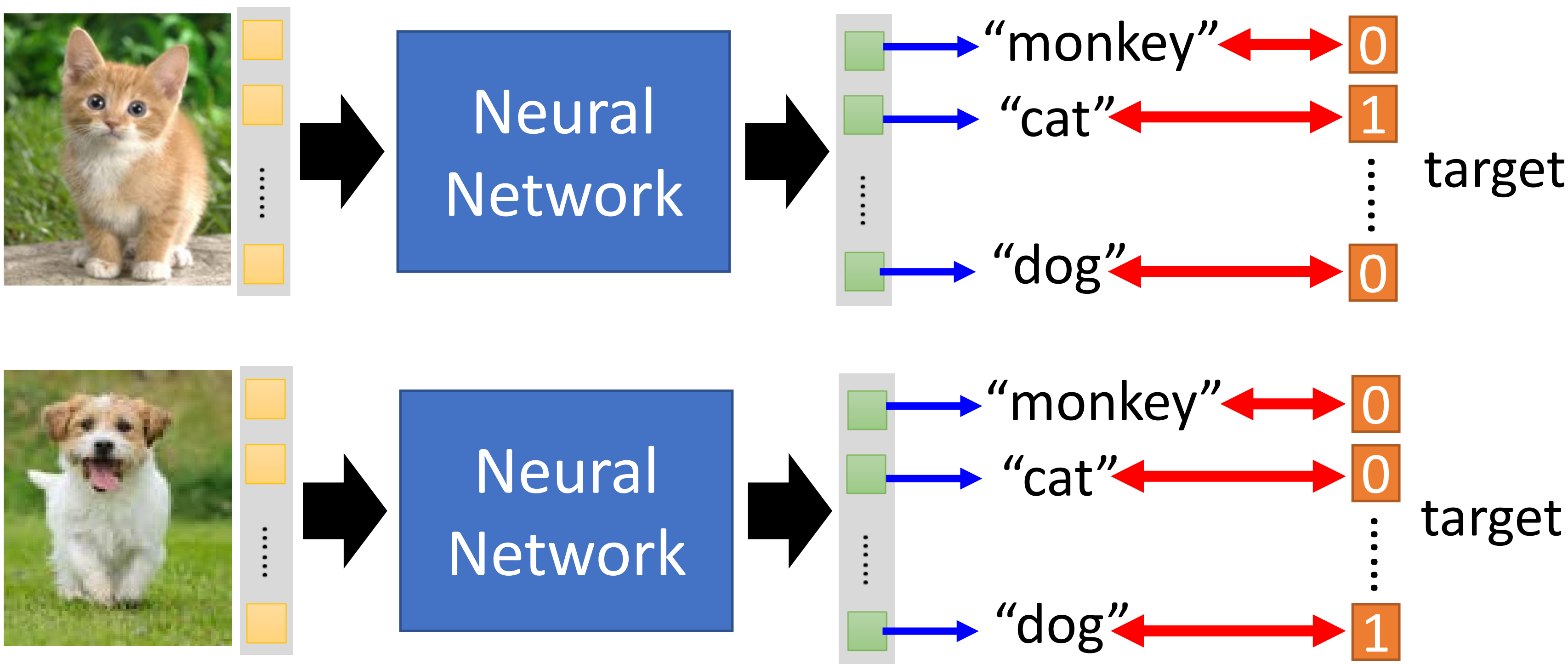
- Demo: 曾柏翔、盧宏宗、吳柏瑜

Outline

- Ultra Deep Network
- Attention Model
- Reinforcement Learning
- Realizing what the World Looks Like
- Understanding the Meaning of Words
- Why Deep?

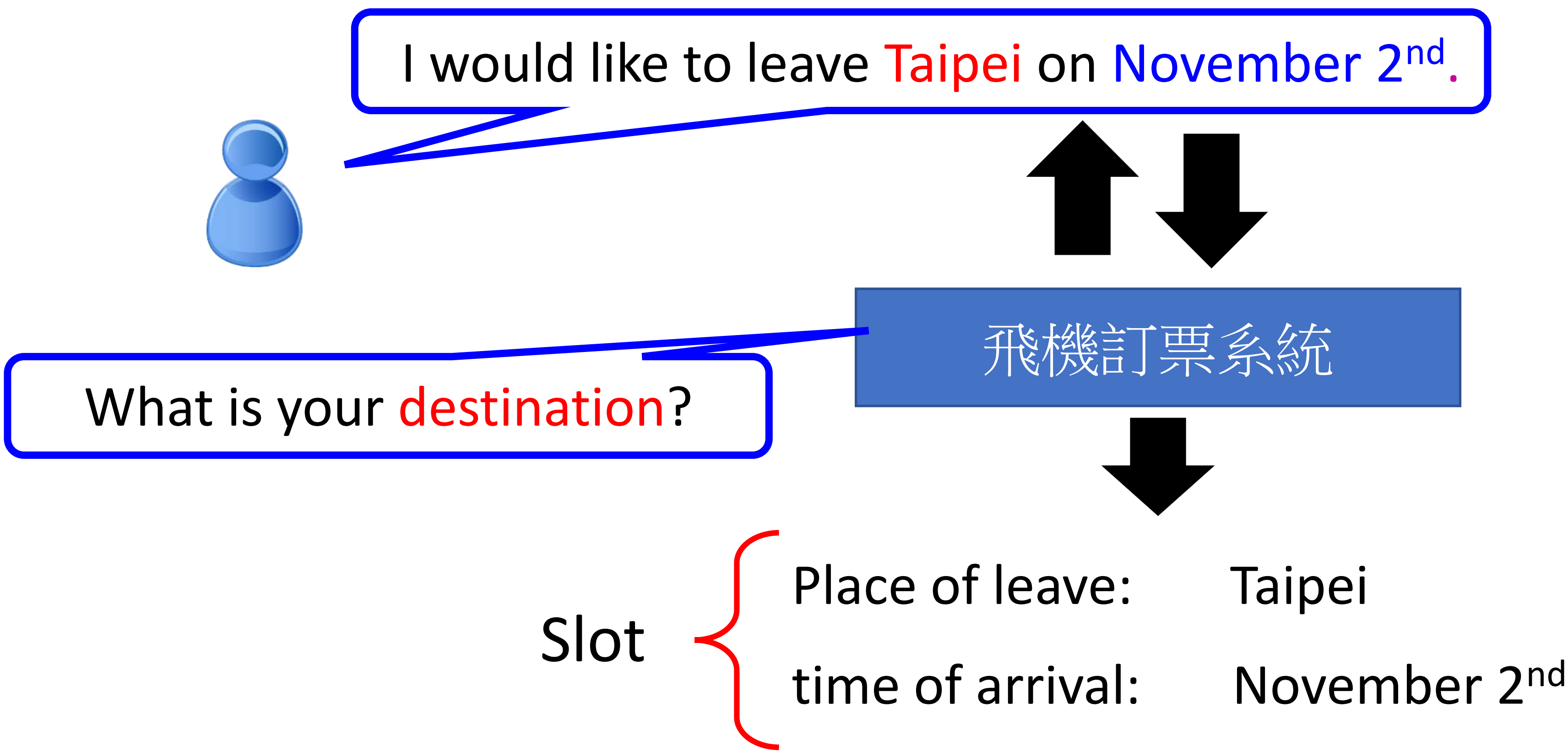
Only Supervised Learning until now ……

- Network is a function. In supervised learning, the input-output pair is given in the training data



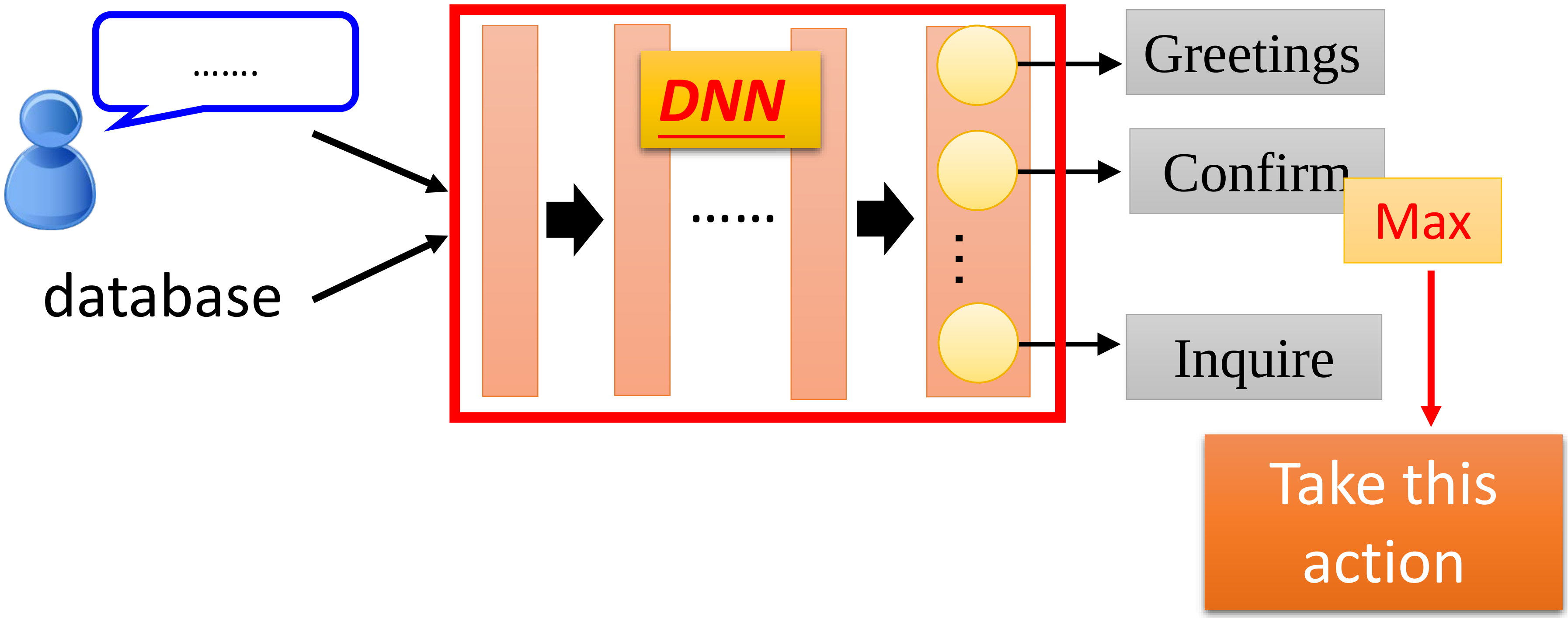
Supervised v.s. Reinforcement

- Example: Dialogue Agent for 訂票系統



Supervised v.s. Reinforcement

- Example: Dialogue Agent for 訂票系統



Supervised v.s. Reinforcement

- Supervised

"Hello" — You have to "greeting"

%$#$%# — You have to "confirm"

- Reinforcement

……. ……. ……

Bad

Hello ☺ ……

Agent Agent

Supervised v.s. Reinforcement

- Playing GO
  - Supervised: 看著棋譜學

下一步:
“5-5”

下一步:
“3-3”

  - Reinforcement Learning

初手天元 → ……下了好幾百手 …… → Win!

Alpha Go is supervised learning + reinforcement learning.

To learn deep reinforcement learning ……

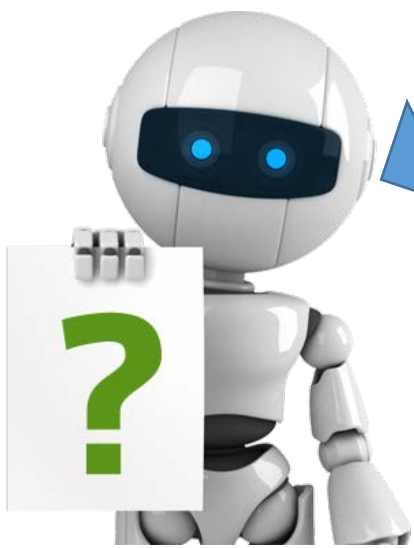
- Lectures of David Silver
  - http://www0.cs.ucl.ac.uk/staff/D.Silver/web/Teaching.html
  - 10 堂課 (1:30 each)
- Deep Reinforcement Learning
  - http://videolectures.net/rldm2015_silver_reinforcement_learning/

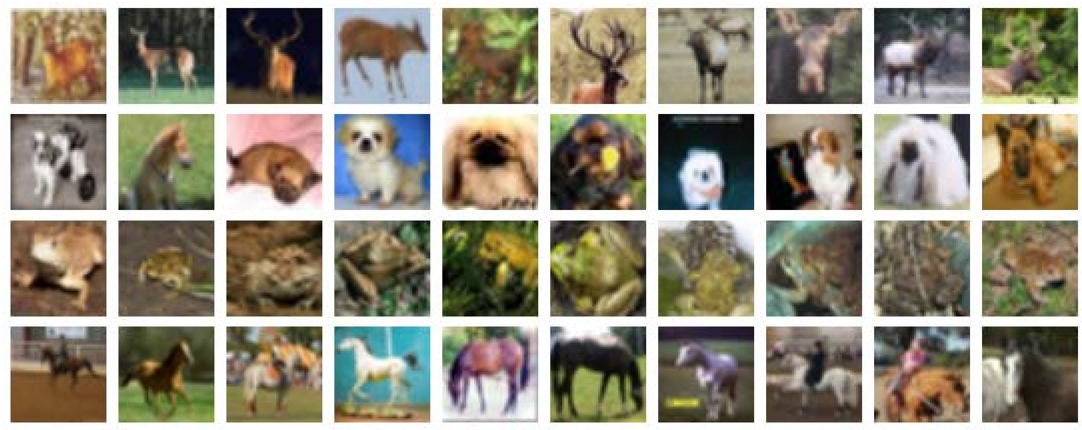
Outline

- Ultra Deep Network
- Attention Model
- Reinforcement Learning
- Realizing what the World Looks Like
- Understanding the Meaning of Words
- Why Deep?

Does machine know what the world look like?

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





Draw something!

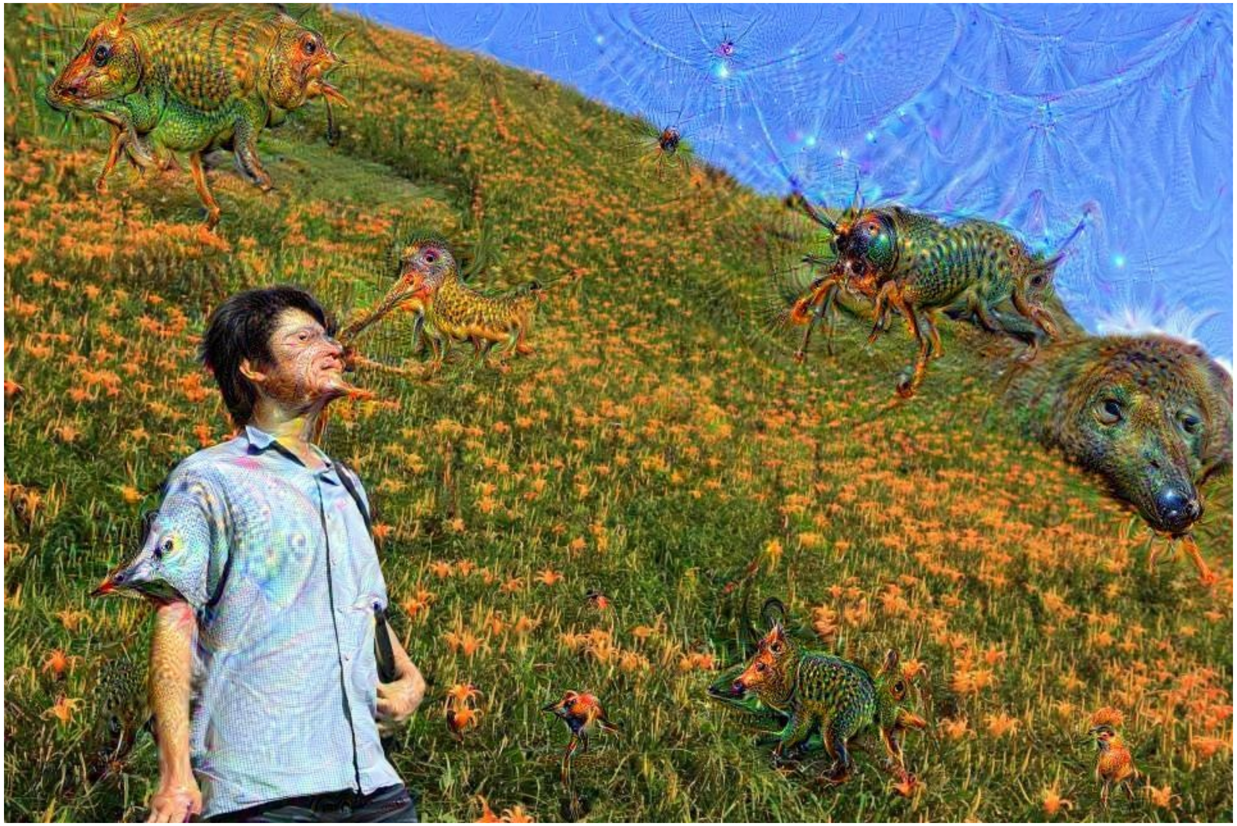
Deep Dream

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

http://deepdreamgenerator.com/

Deep Dream

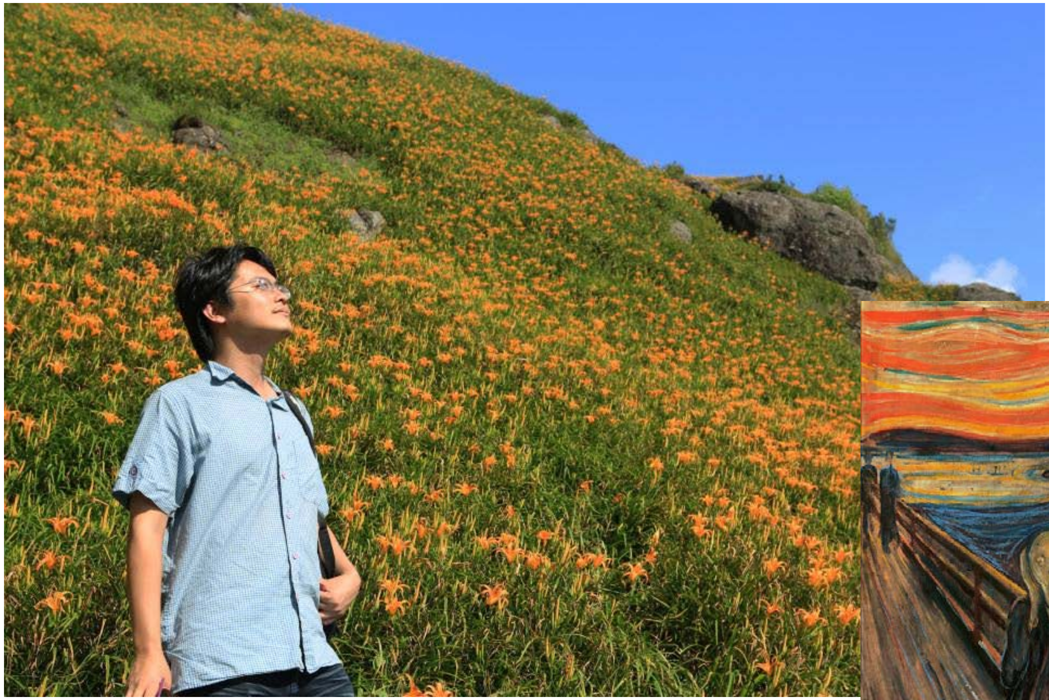
- Given a photo, machine adds what it sees ……

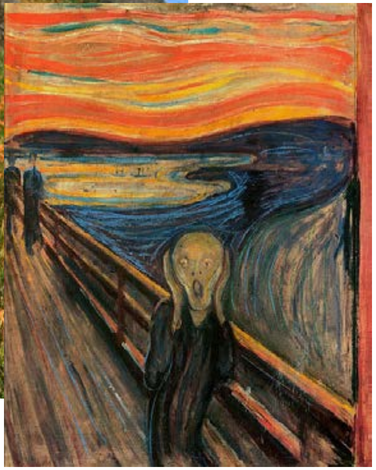


http://deepdreamgenerator.com/

Deep Style

- Given a photo, make its style like famous paintings





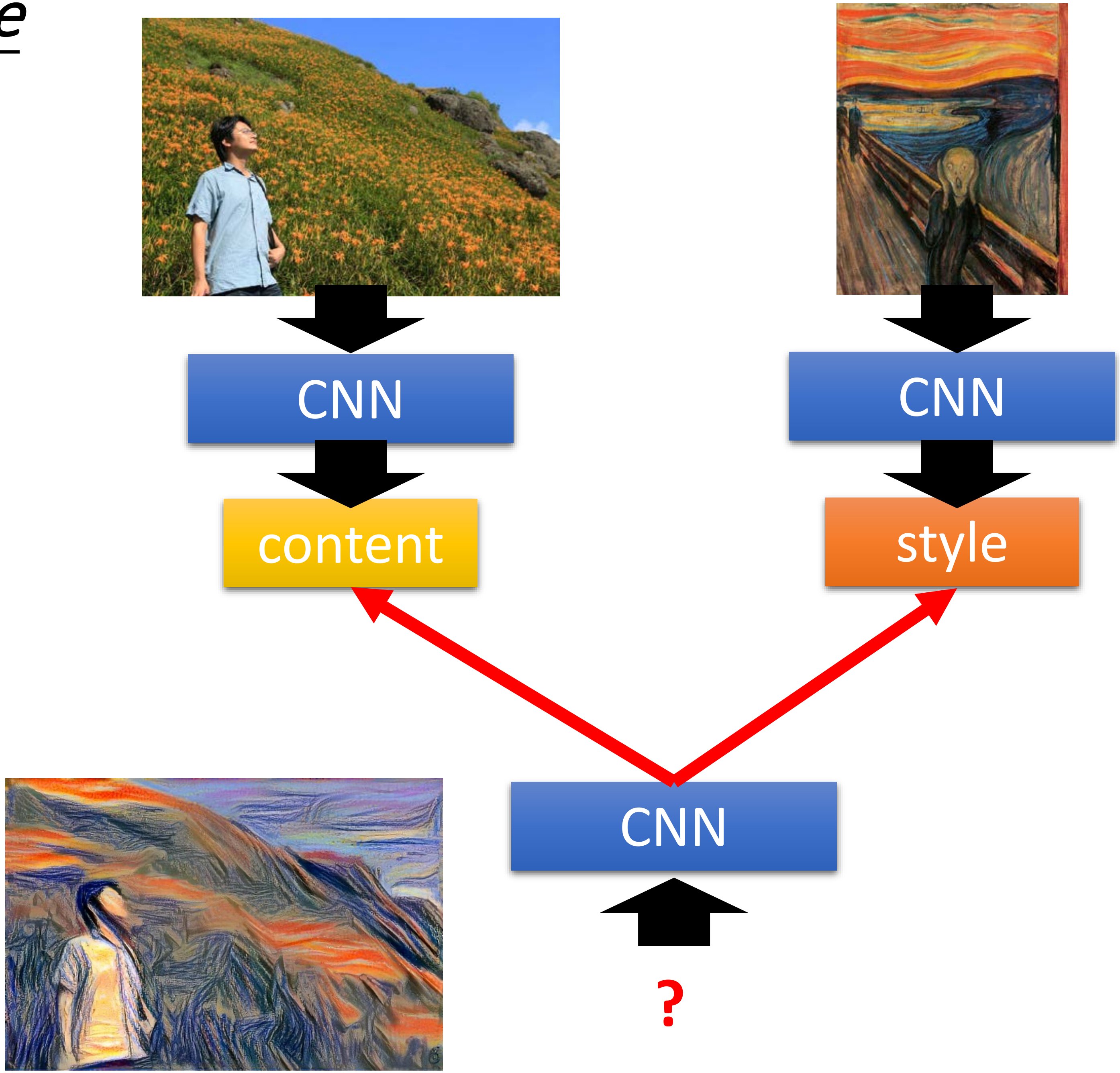
https://dreamscopeapp.com/

Deep Style

- Given a photo, make its style like famous paintings

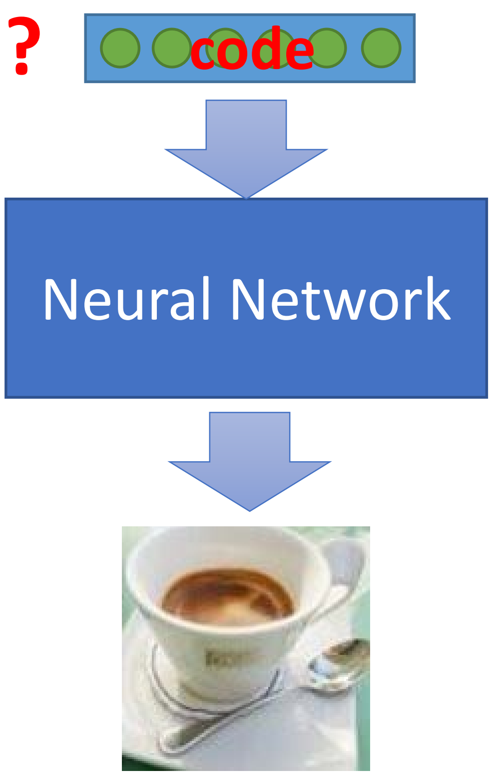
https://dreamscopeapp.com/

Deep Style

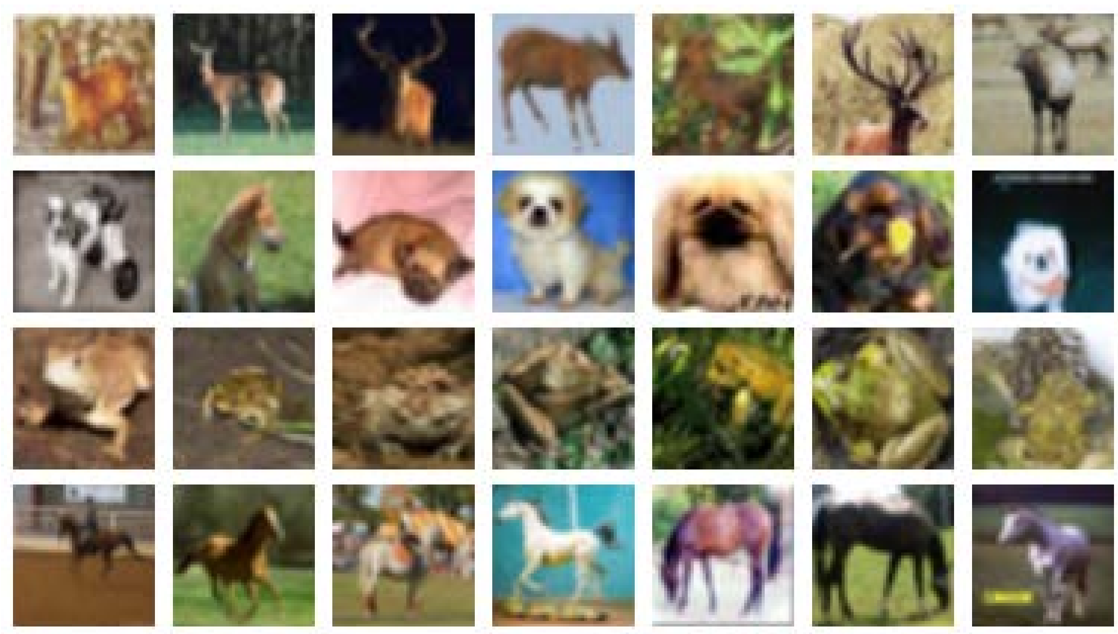


Generating Images (無中生有)

- Training a decoder to generate images is unsupervised

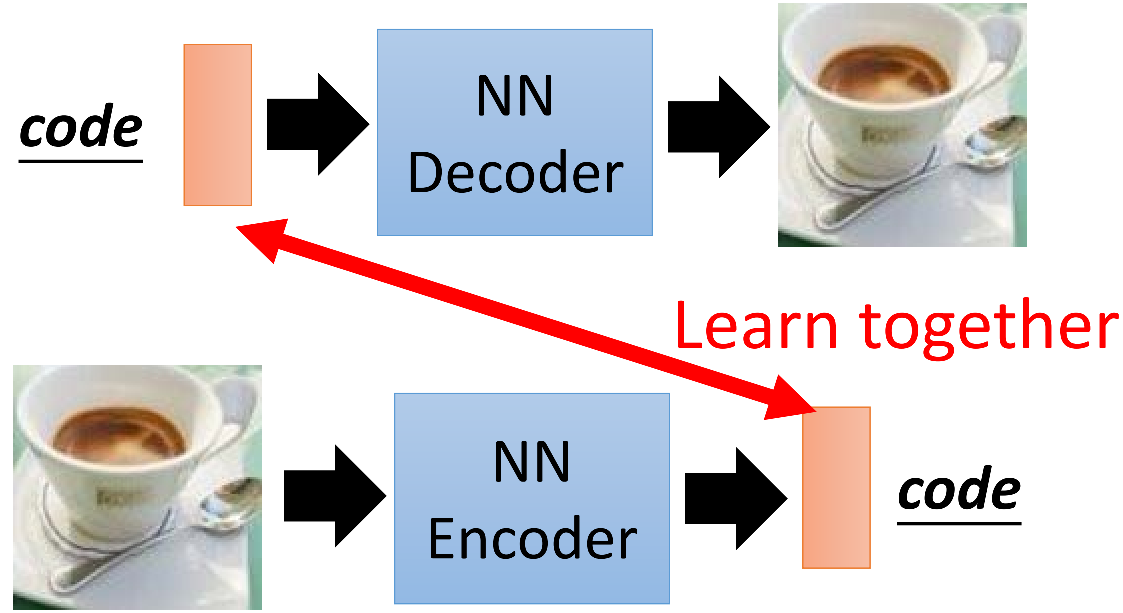


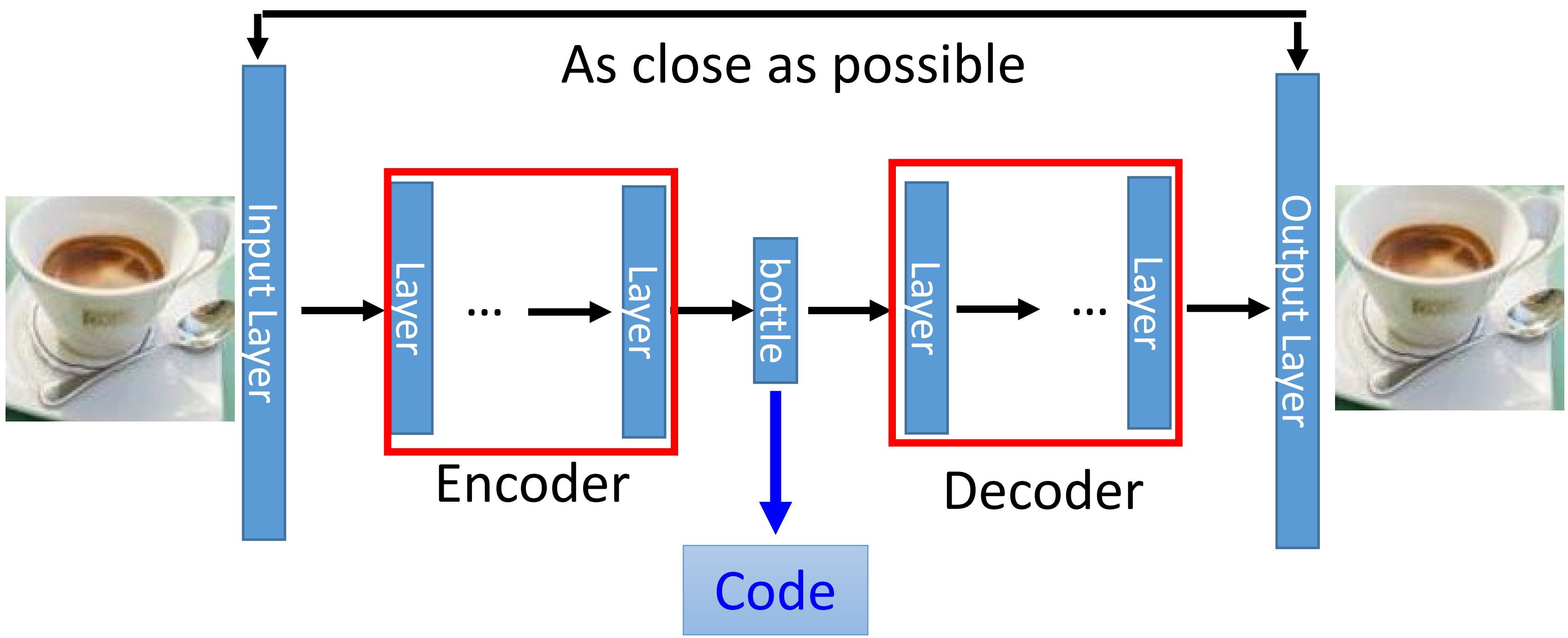
Training data is a lot of images



Auto-encoder

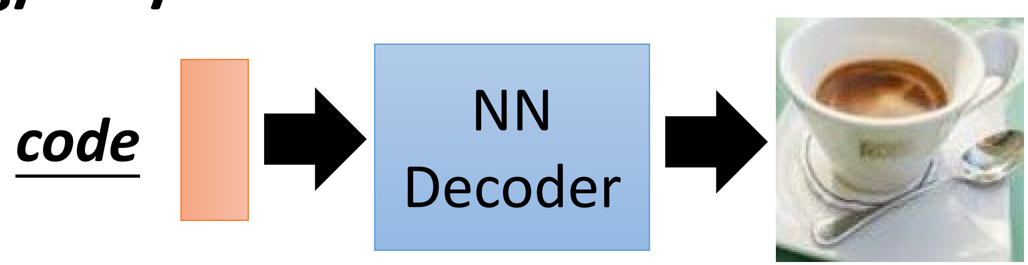
Not state-of-the-art approach



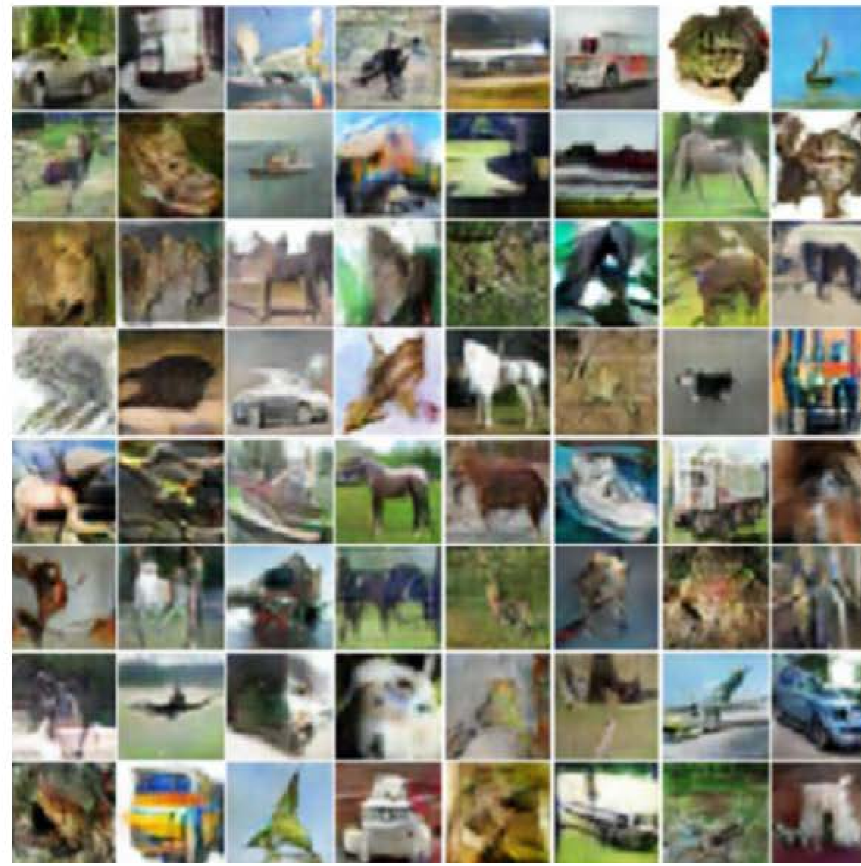


Generating Images

- 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**

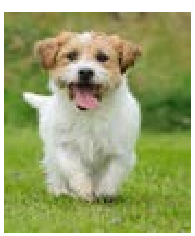


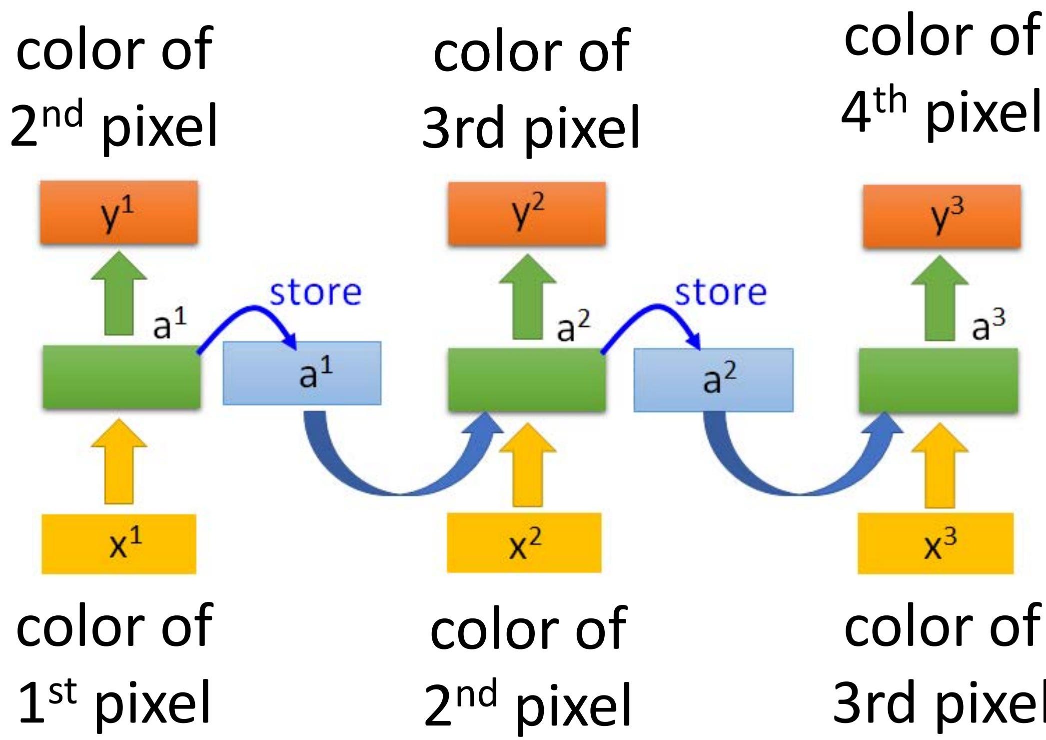
Which one is machine-generated?



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

Generating Images by RNN





Generating Images by RNN

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

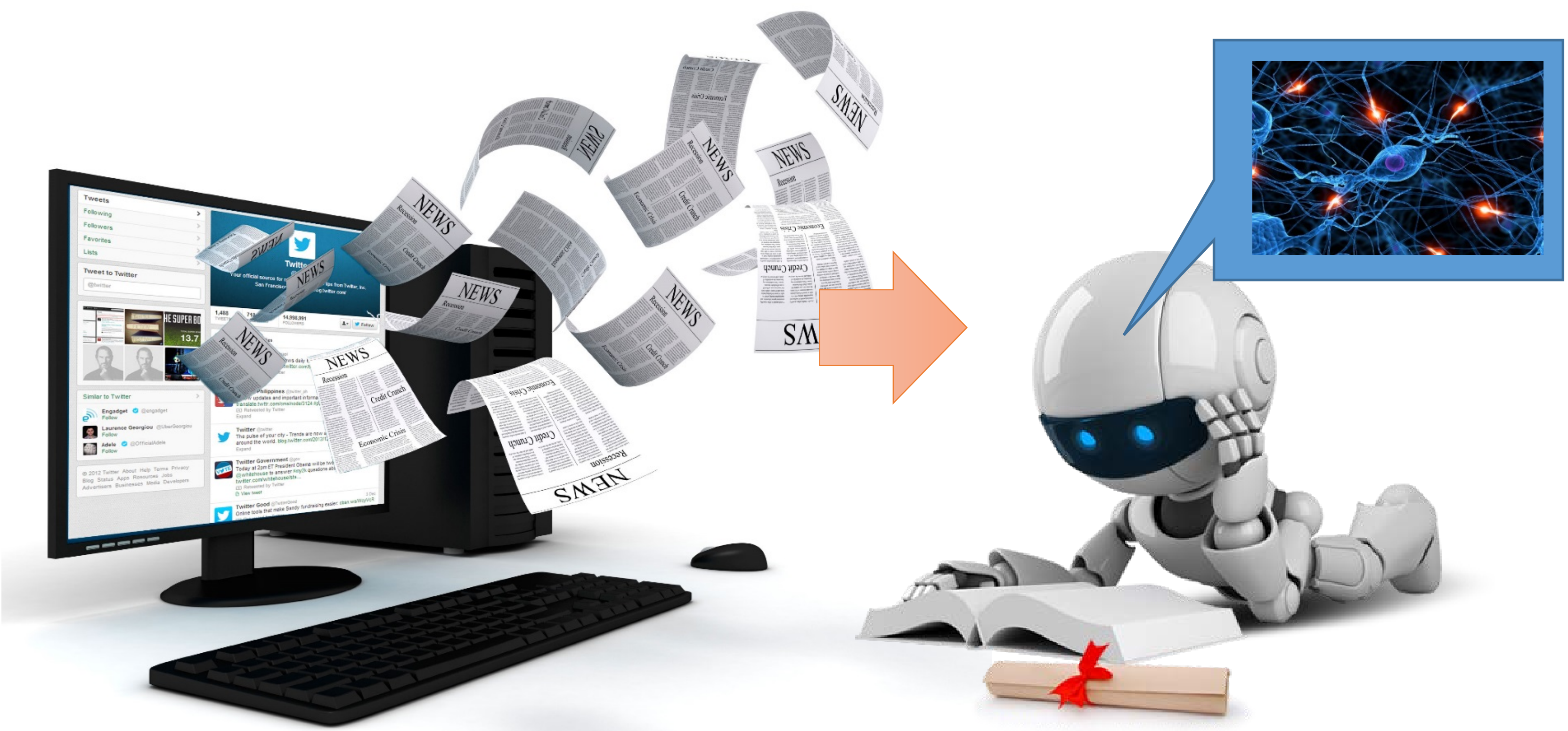
Real World

Outline

- Ultra Deep Network
- Attention Model
- Reinforcement Learning
- Realizing what the World Looks Like
- Understanding the Meaning of Words
- Why Deep?

Machine Reading

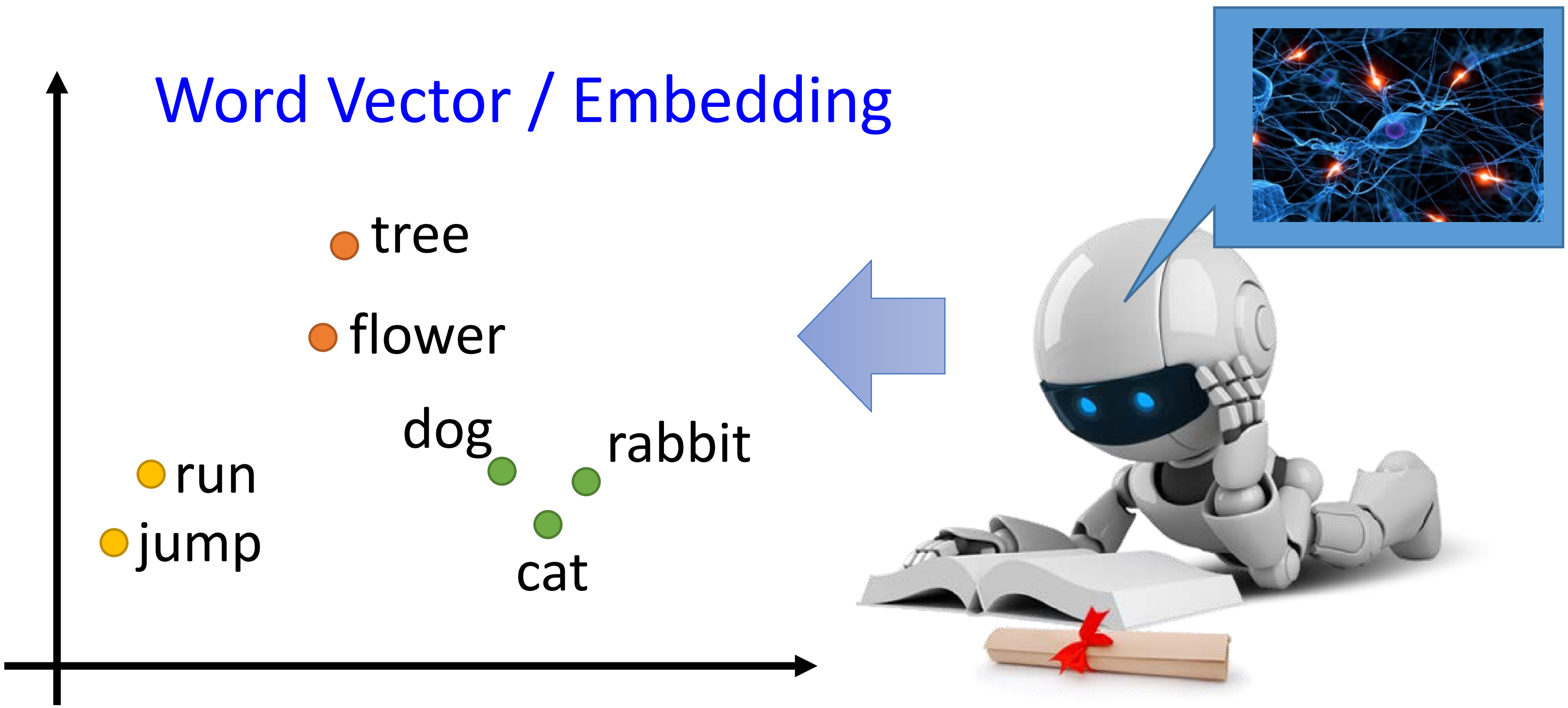
- Machine learn the meaning of words from reading a lot of documents without supervision



http://top-breaking-news.com/

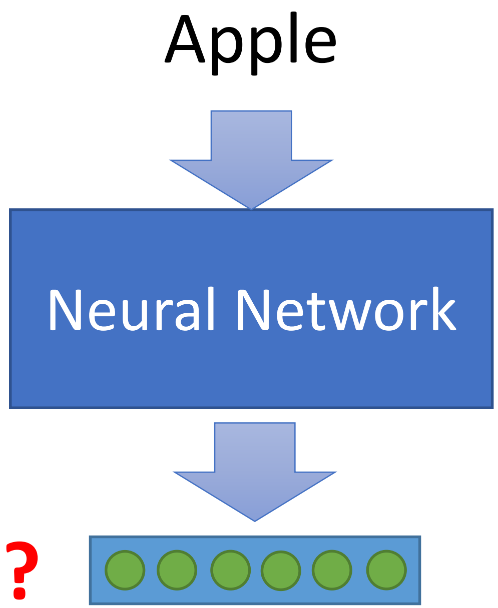
Machine Reading

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

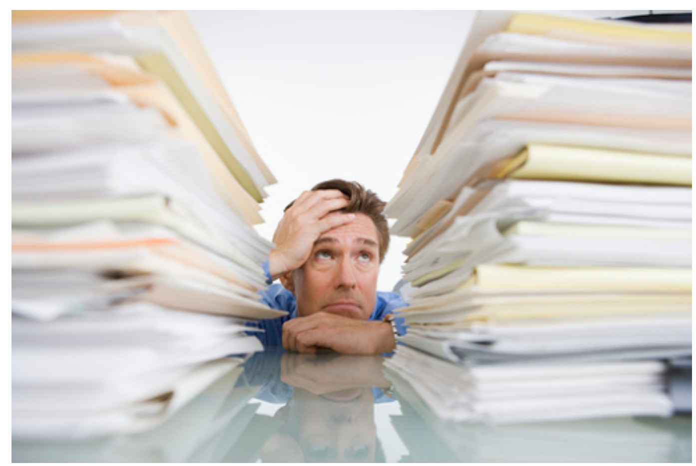


Machine Reading

- Generating Word Vector/Embedding is unsupervised



Training data is a lot of text



https://garavato.files.wordpress.com/2011/11/stacksdocuments.jpg?w=490

Machine Reading

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

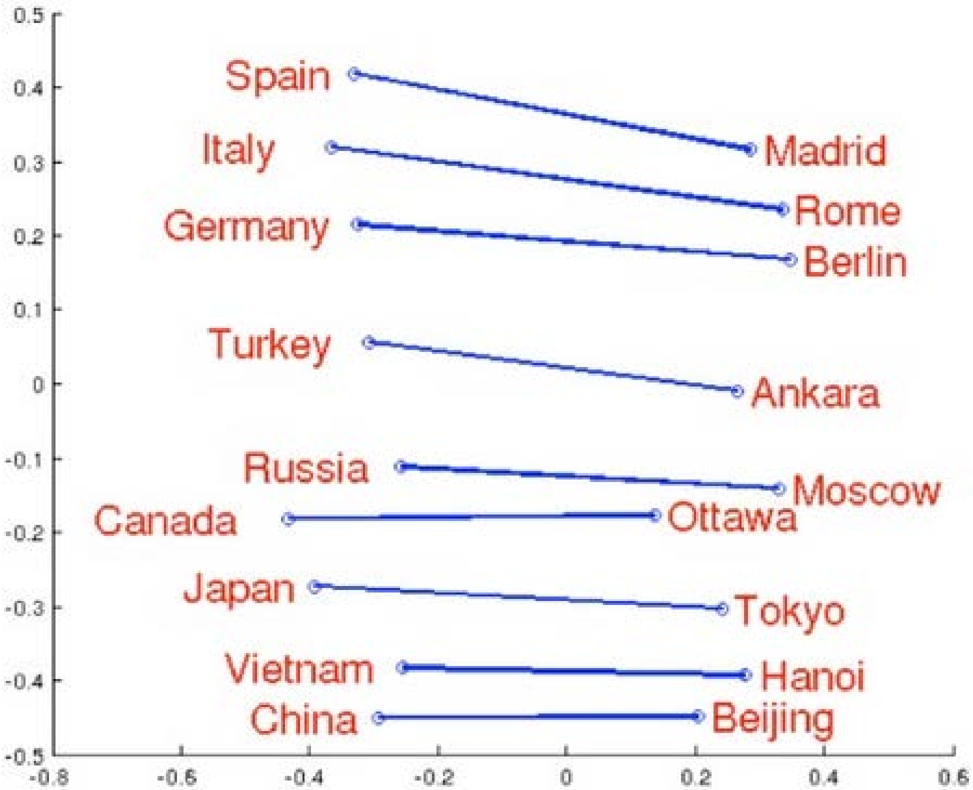
蔡英文、馬英九 are something very similar

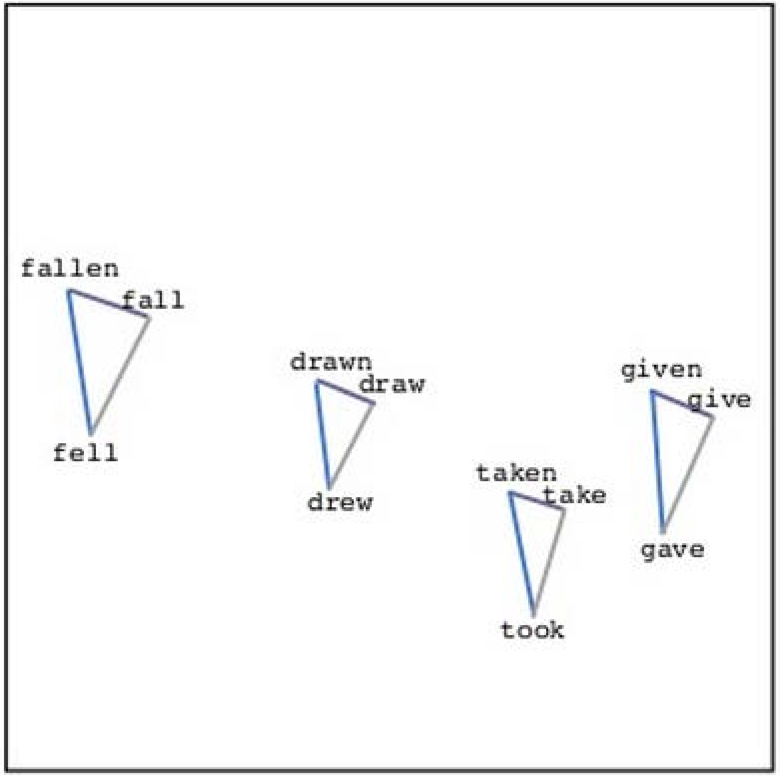
You shall know a word by the company it keeps

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

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

- 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 : ?

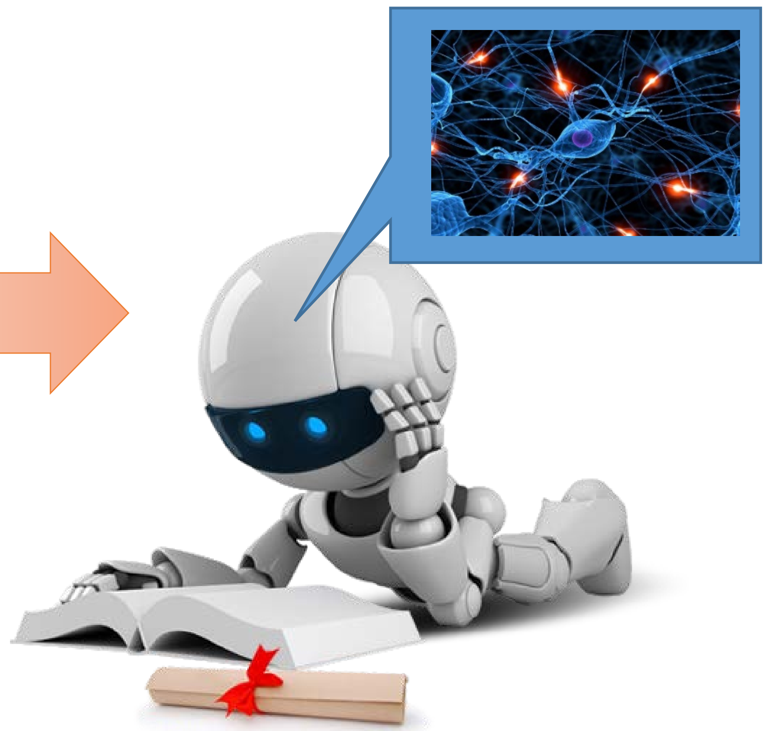
Compute $\underline{V(Berlin) - V(Rome) + V(Italy)}$

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

Machine learns to understand 鄉民用語 via reading the posts on PTT



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

- Ultra Deep Network
- Attention Model
- Reinforcement Learning
- Realizing what the World Looks Like
- Understanding the Meaning of Words
- Why Deep?

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.

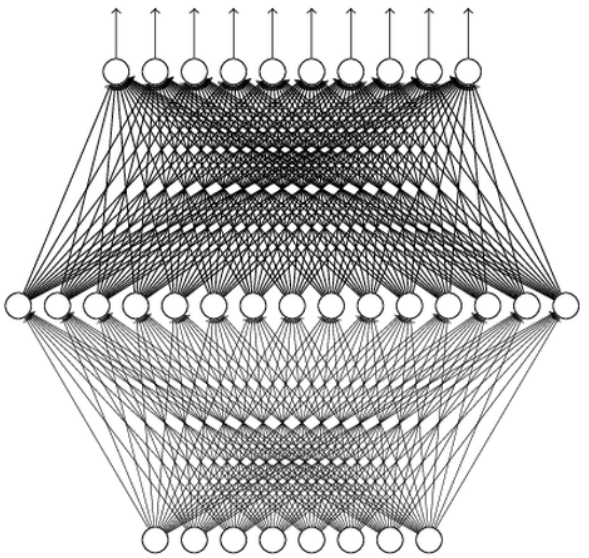
Universality Theorem

Any continuous function f

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

Can be realized by a network with one hidden layer

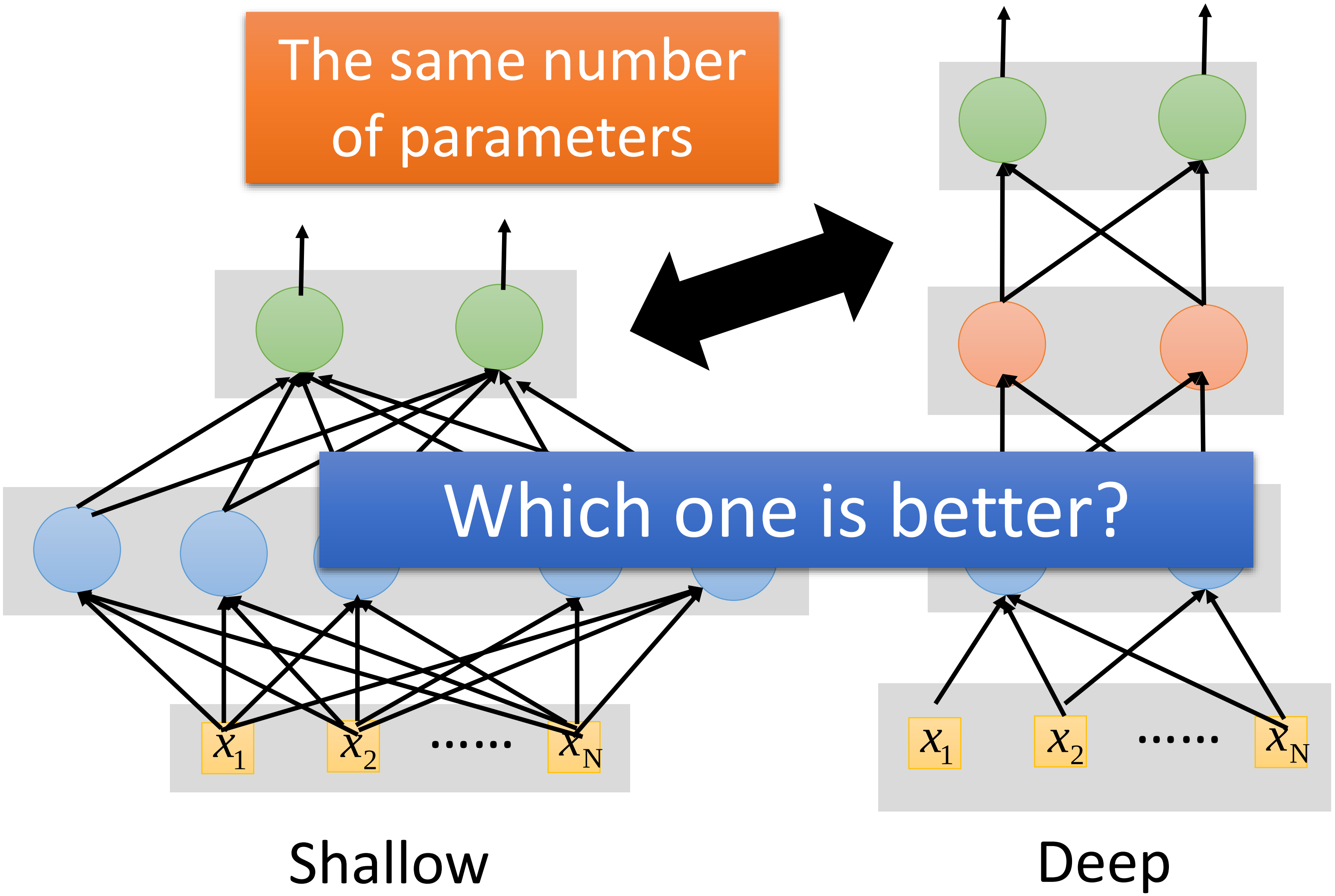
(given **enough** hidden neurons)



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

Why "Deep" neural network not "Fat" neural network?

Fat + Short v.s. Thin + Tall



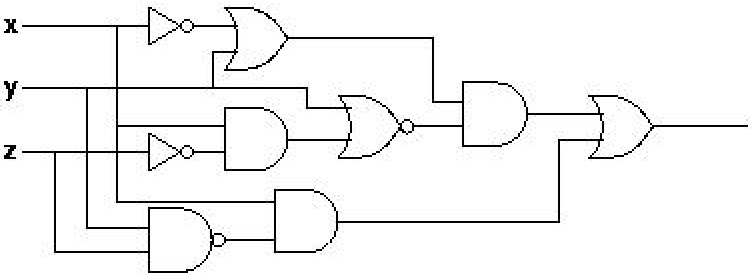
Fat + Short v.s. Thin + Tall

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

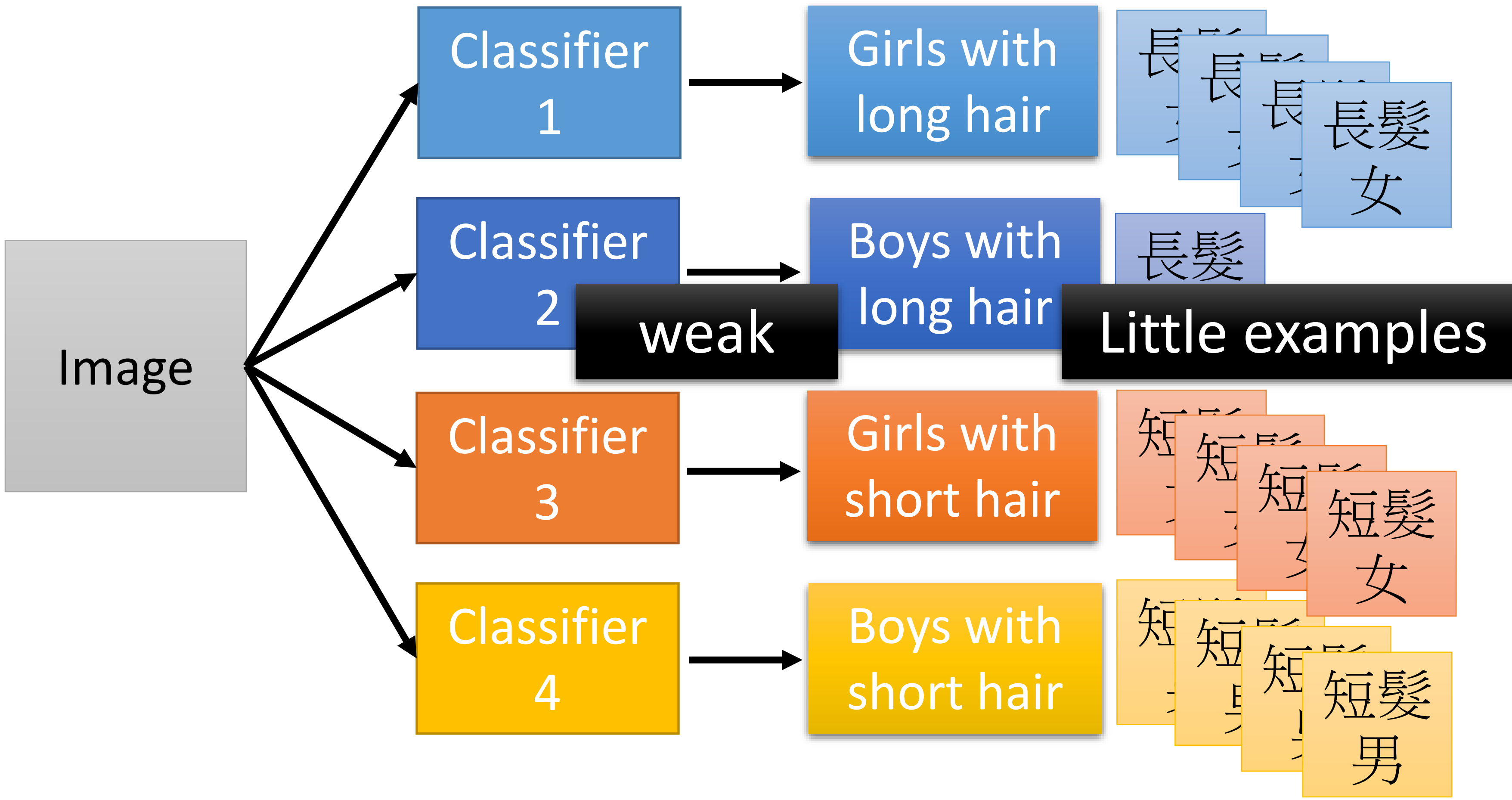
- Neural network consists of **neurons**
- **A hidden layer network** can represent **any continuous function.**
- Using multiple layers of neurons to represent some functions are much simpler

→ less parameters → less data?

This page is for EE background.

Modularization

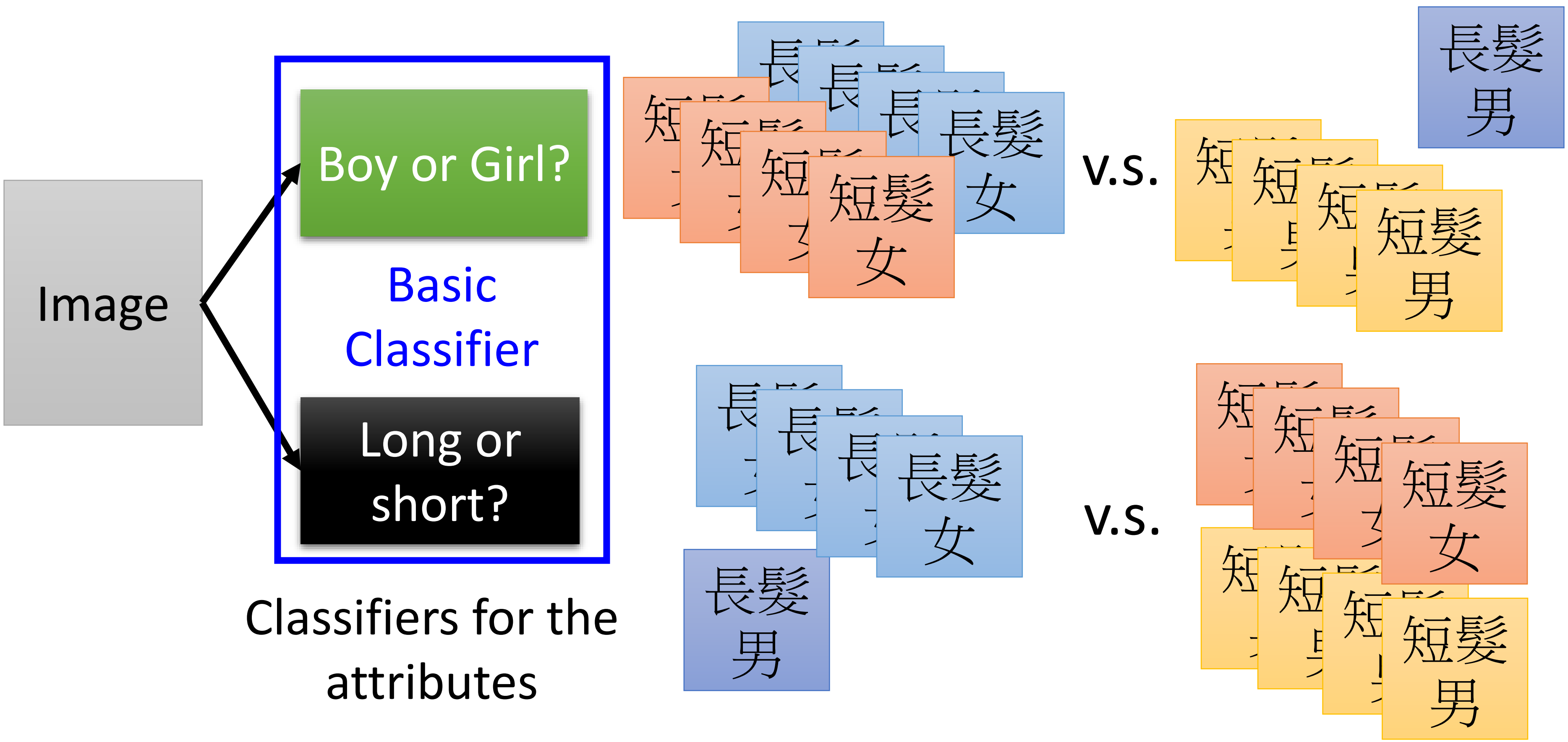
- Deep → Modularization



Modularization

Each basic classifier can have sufficient training examples.

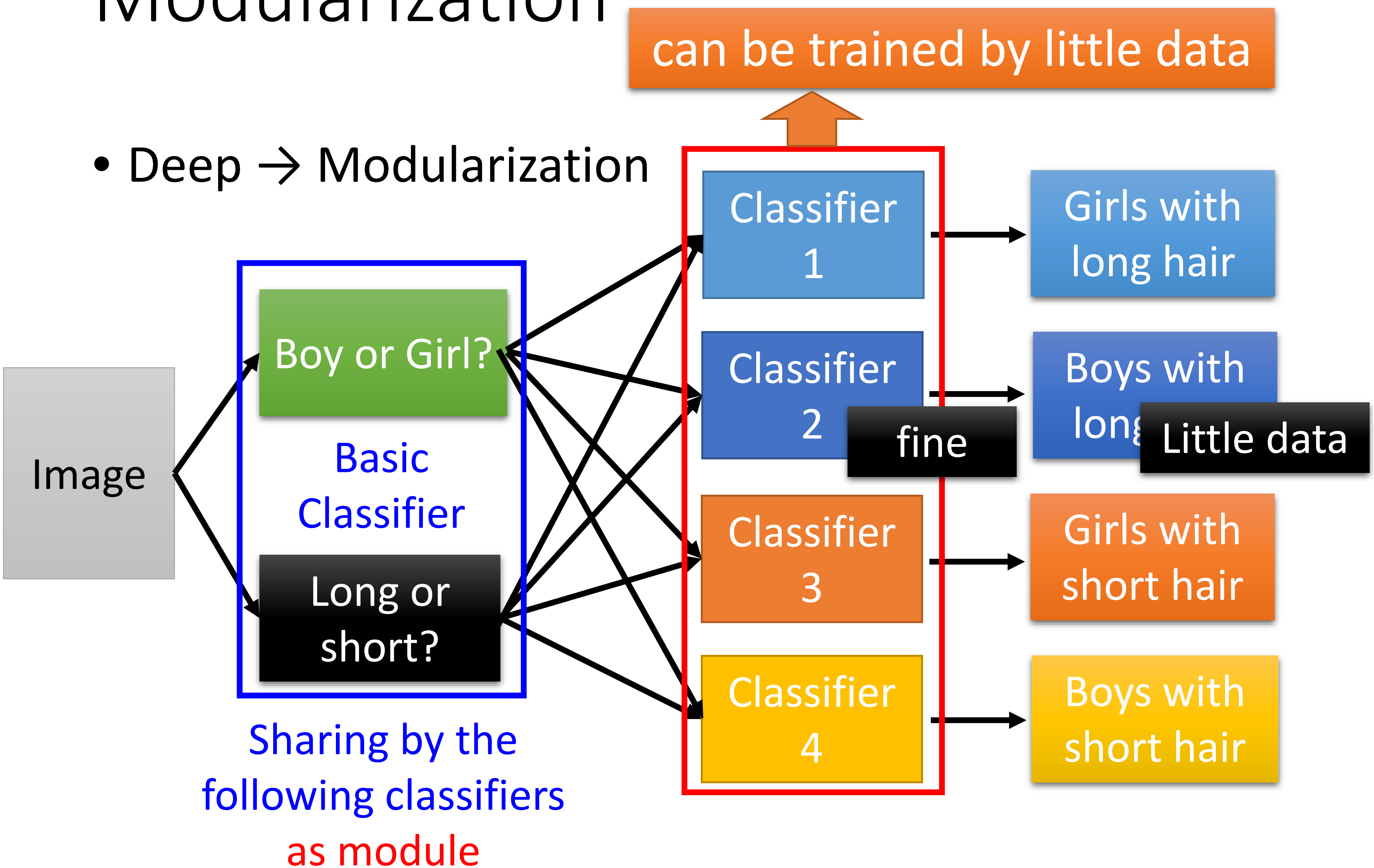
- Deep → Modularization



Classifiers for the attributes

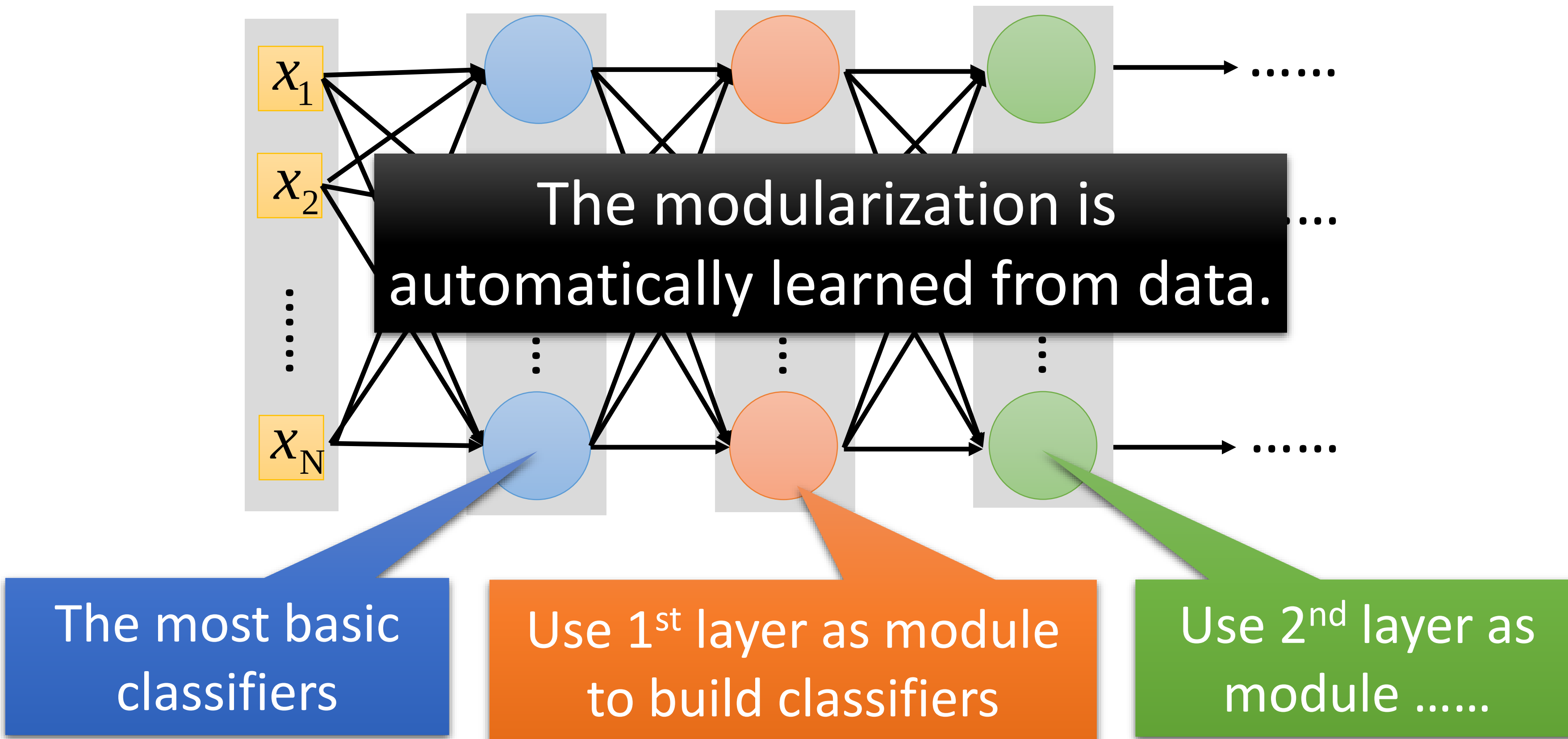
Modularization

- Deep → Modularization



Modularization

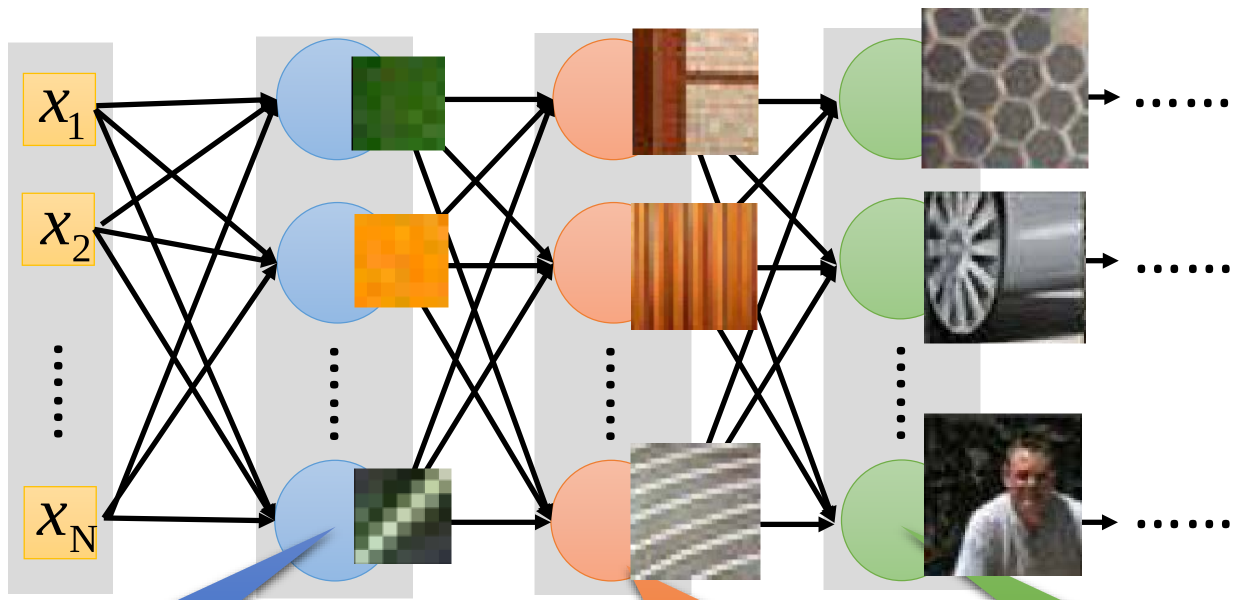
- Deep → Modularization → Less training data?



Modularization

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

- Deep → Modularization

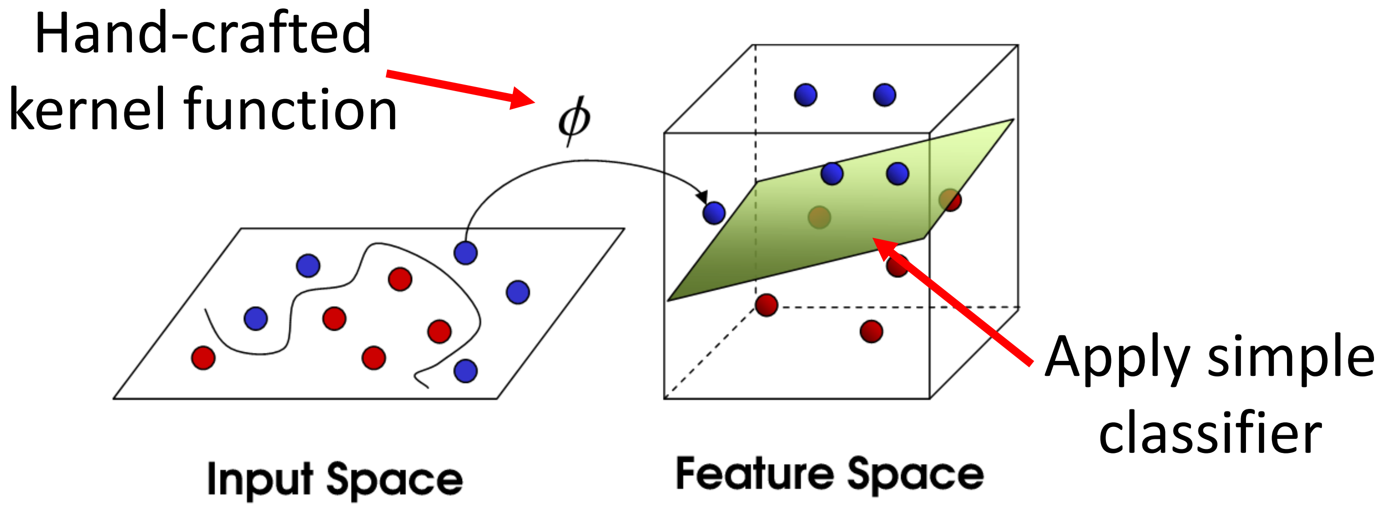


The most basic classifiers

Use 1st layer as module to build classifiers

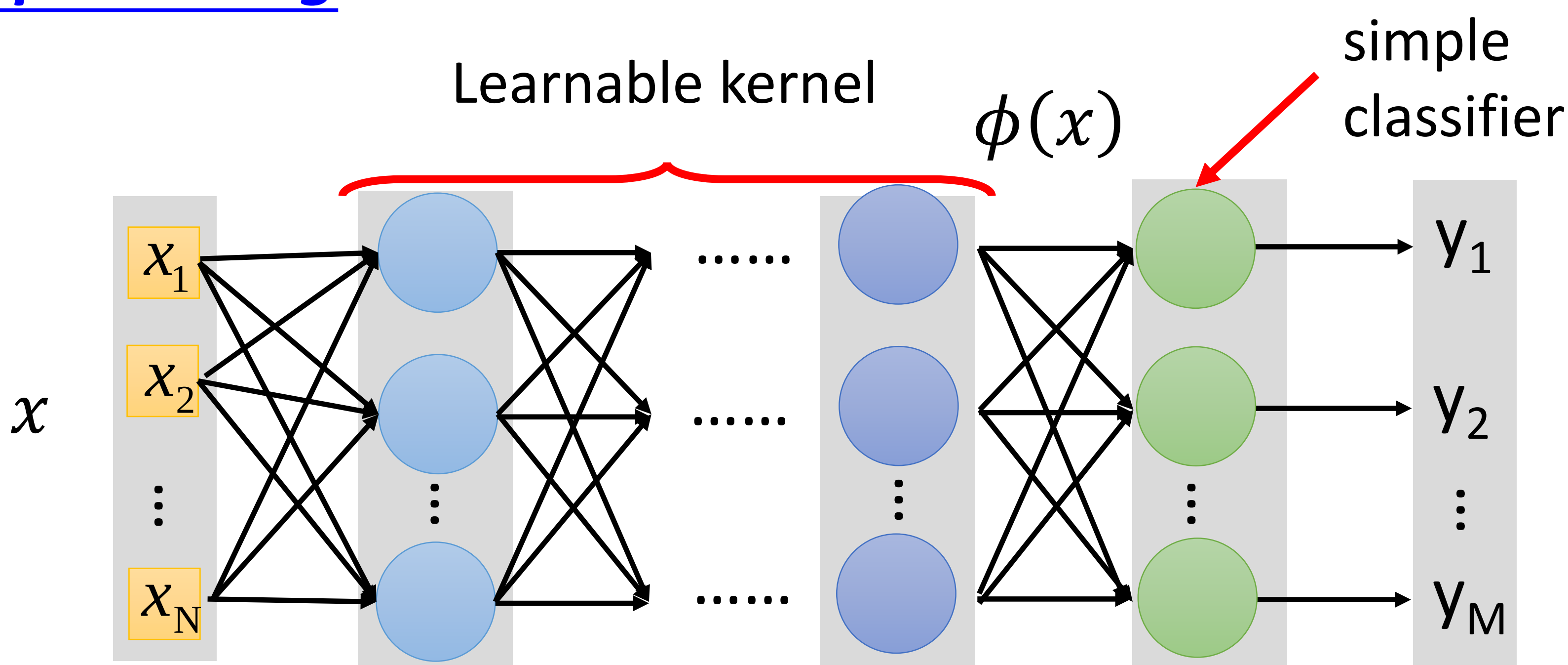
Use 2nd layer as module ……

SVM

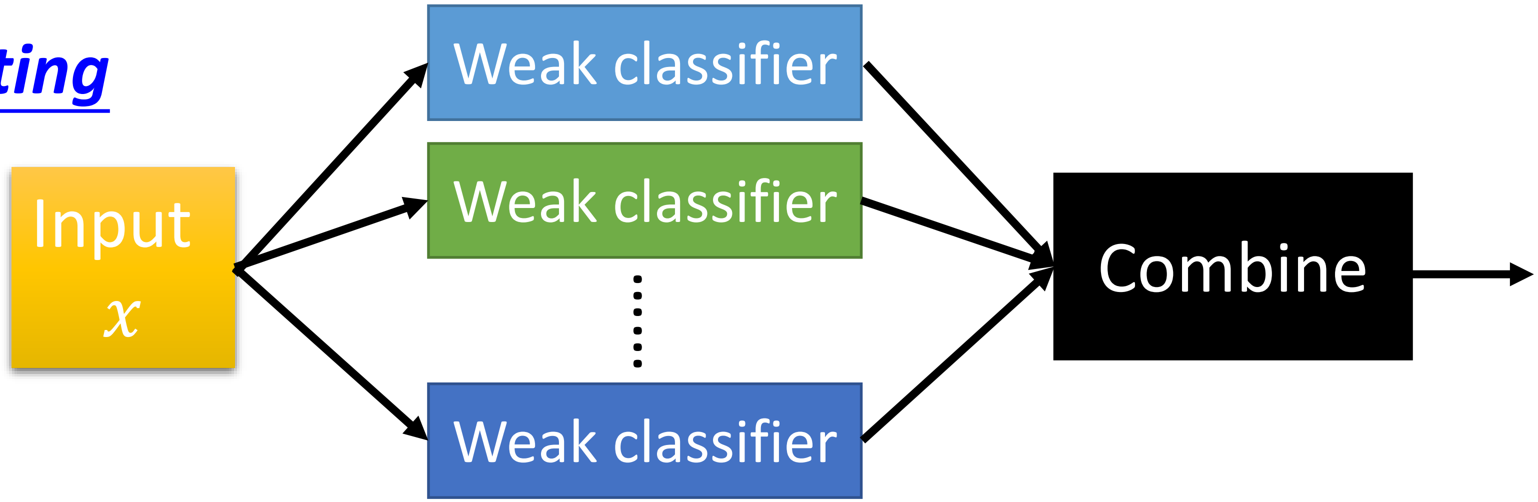


Source of image: http://www.gipsa-lab.grenoble-inp.fr/transfert/seminaire/455_Kadri2013Gipsa-lab.pdf

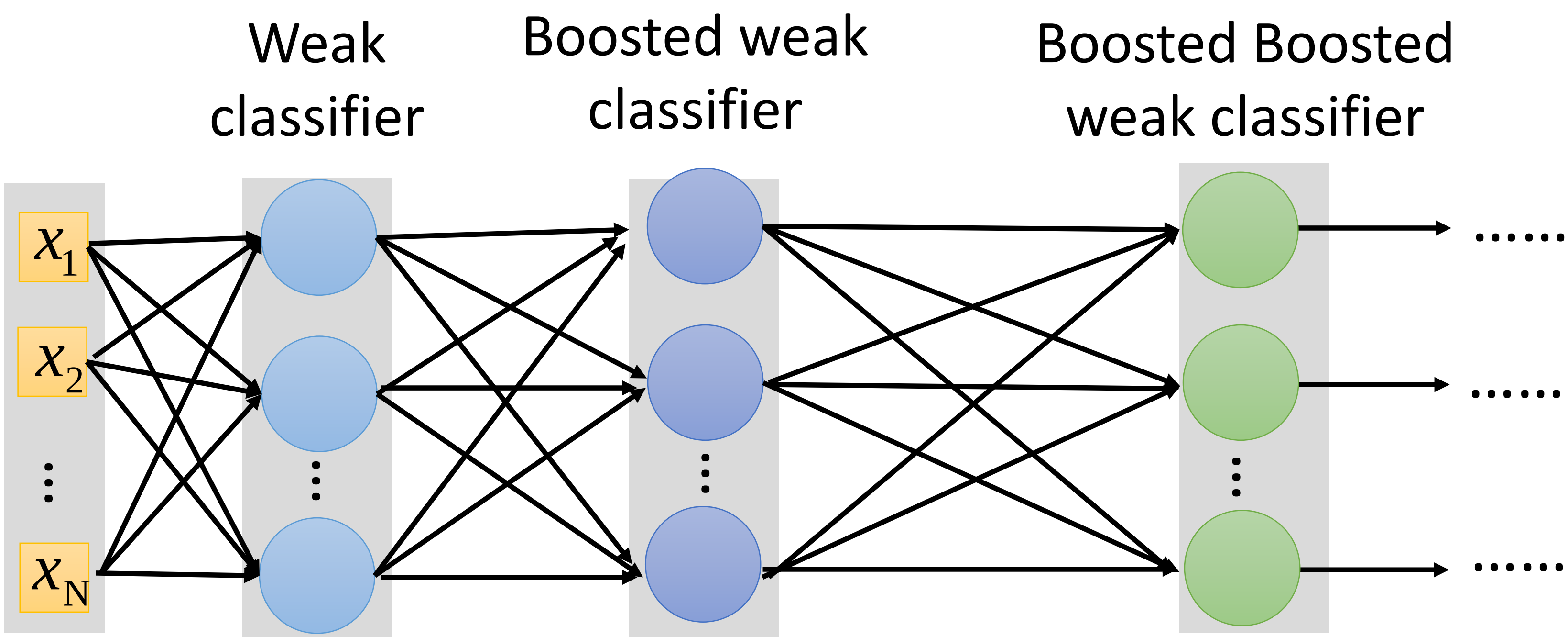
Deep Learning



Boosting

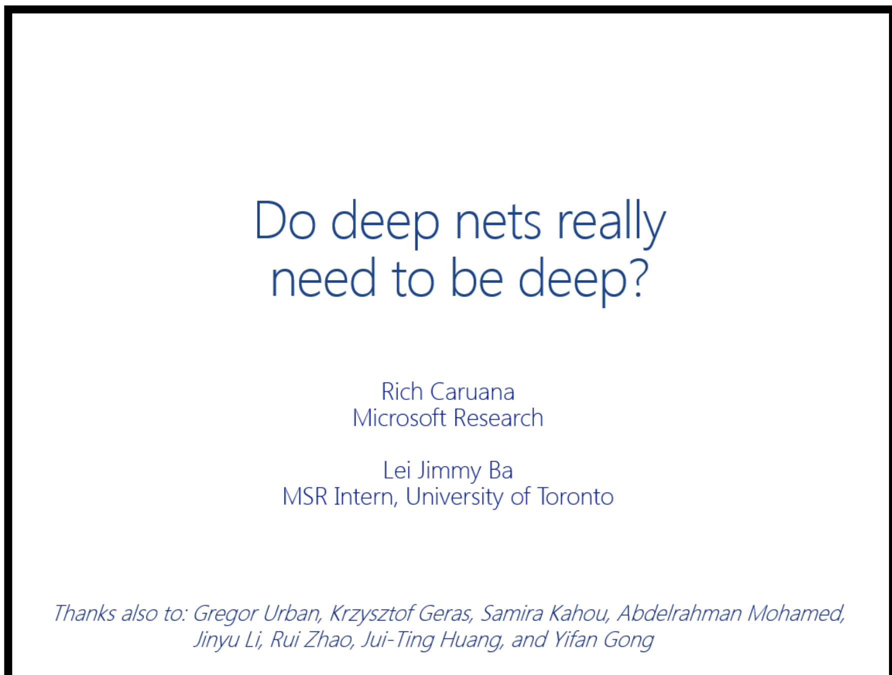


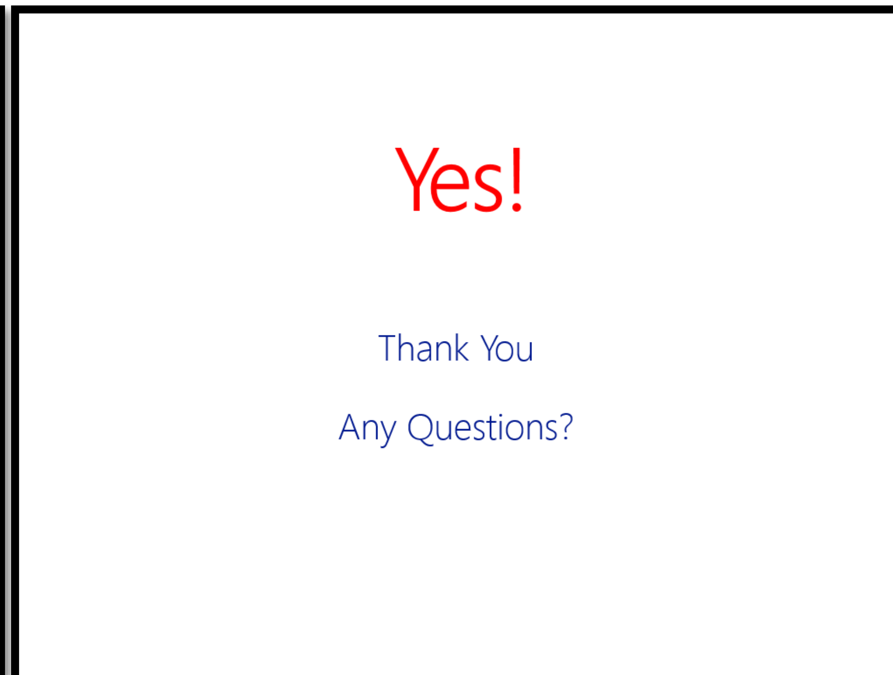
Deep Learning



More Reasons

- Do Deep Nets Really Need To Be Deep? (by Rich Caruana)
- http://research.microsoft.com/apps/video/default.aspx?id=232373&r=1





keynote of Rich Caruana at ASRU 2015

Concluding Remarks

Today's Lecture

Lecture I: Introduction of Deep Learning

Lecture II: Tips for Training Deep Neural Network

Lecture III: Variants of Neural Network

Lecture IV: Next Wave

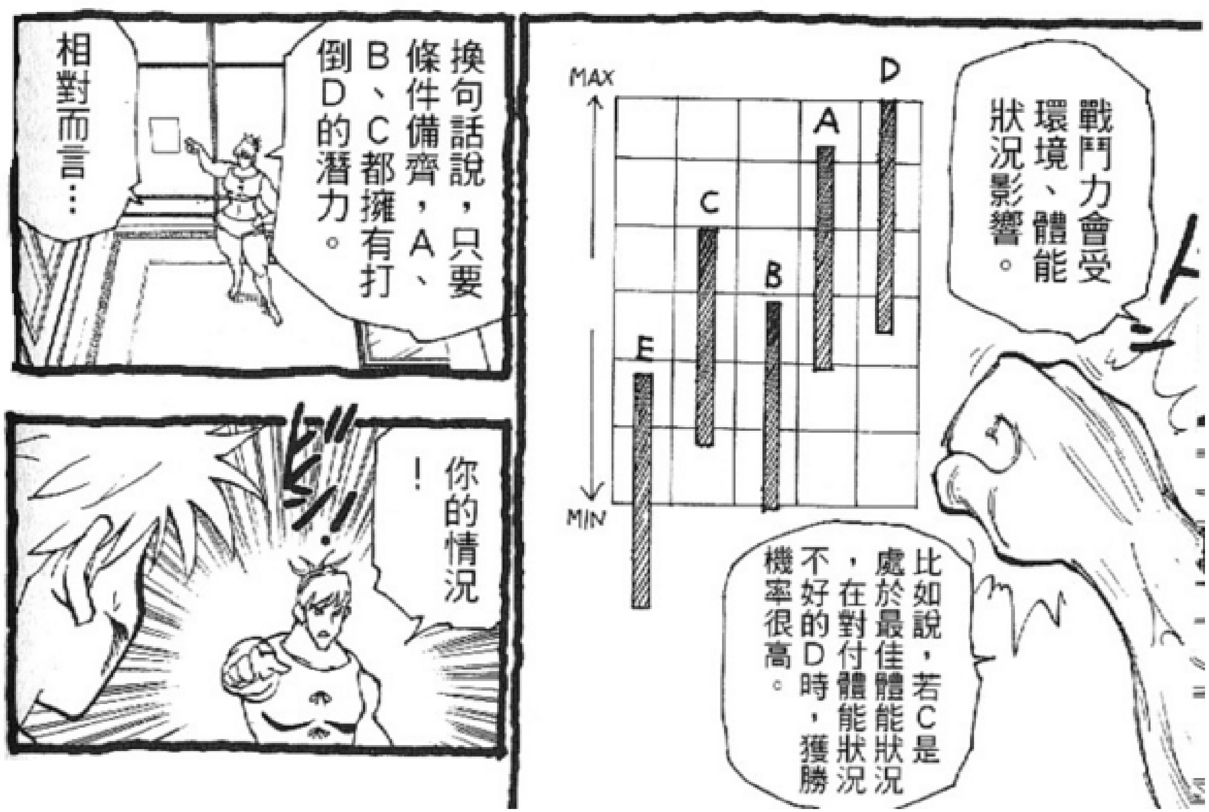
Some Opinions

- Also learn other machine learning methods

不想知道 Deep Learning 以外的方法

Some Opinions

- In some situations, the simpler machine learning methods can be very powerful.



Some Opinions

http://www.baike.com/gwiki/%E5%AF%92%E6%AD%A6%E7%BA%AA%E5%A4%A7%E7%88%86%E5%8F%91

- 寒武纪大爆炸

已經有一些生物滅絕了

Some Opinions

- Deep Learning is still at the phase of “神農嘗百草”
- Lots of questions still do not have answers
- However, probably also easy to enter

http://orchid.shu.edu.tw/upload/article/20110927181605_1_pic.png

如果你想 “深度學習 深度學習”

- “Neural Networks and Deep Learning”
  - written by Michael Nielsen
  - http://neuralnetworksanddeeplearning.com/
- “Deep Learning”
  - Written by Yoshua Bengio, Ian J. Goodfellow and Aaron Courville
  - http://www.iro.umontreal.ca/~bengioy/dlbook/
- Course: Machine learning and having it deep and structured
  - http://speech.ee.ntu.edu.tw/~tlkagk/courses_MLSD15_2.html

給資料科學愛好者

- 台大電機系於台大電信所成立「資料科學與智慧網路組」，開始招收碩、博士生
- 今年秋天開始報名