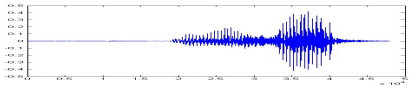
Introduction of Neural Networks

Machine Learning
≈ Looking for a Function

- Speech Recognition

$f($  $)=$ “How are you”

- Image Recognition

$f($  $)=$ “Cat”

- Playing Go

$f($  $)=$ “5-5” (next move)

- Dialogue System

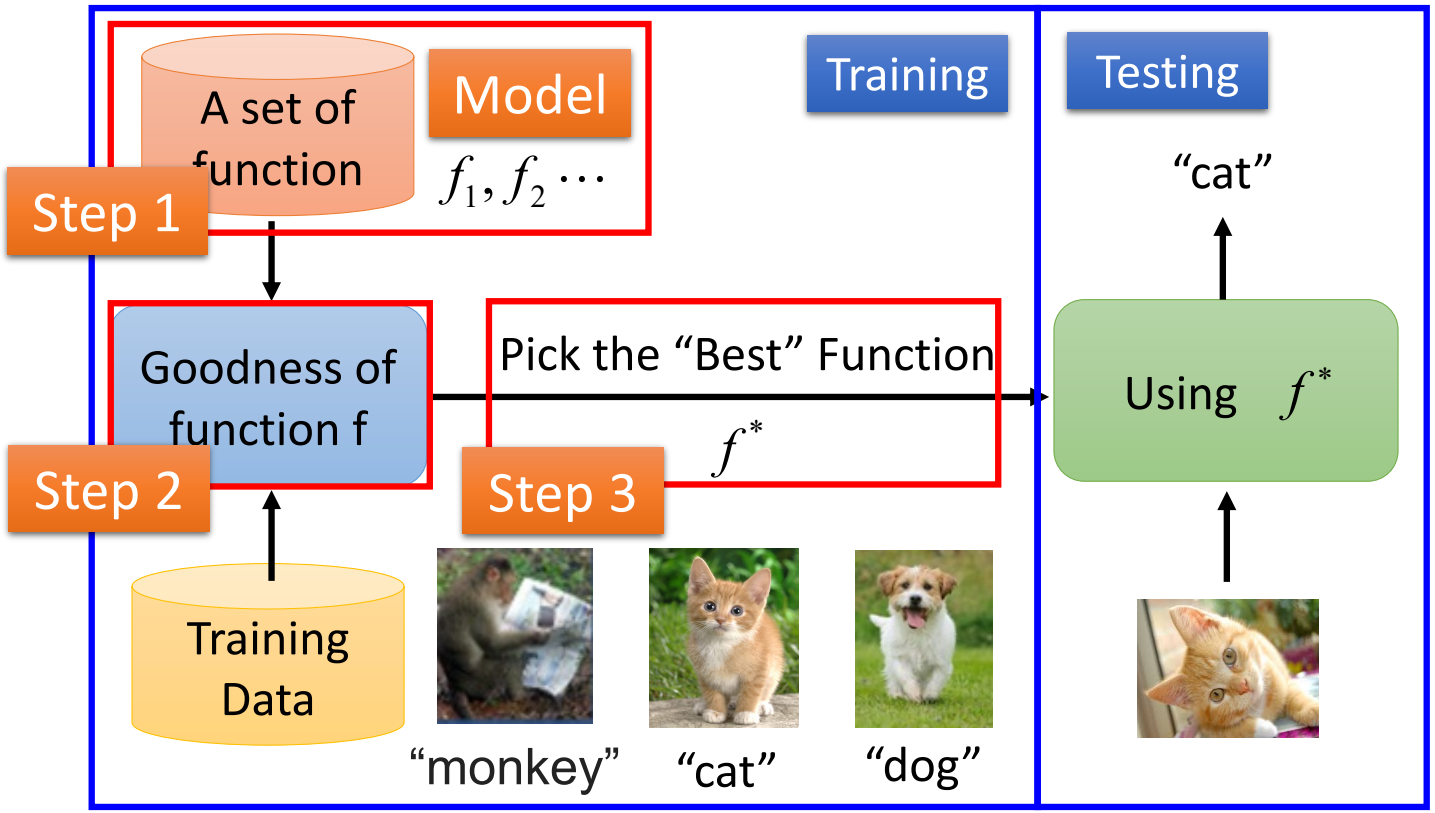
$f($ “Hi” $)=$ “Hello”

(what the user said) (system response)

Framework

Image Recognition:

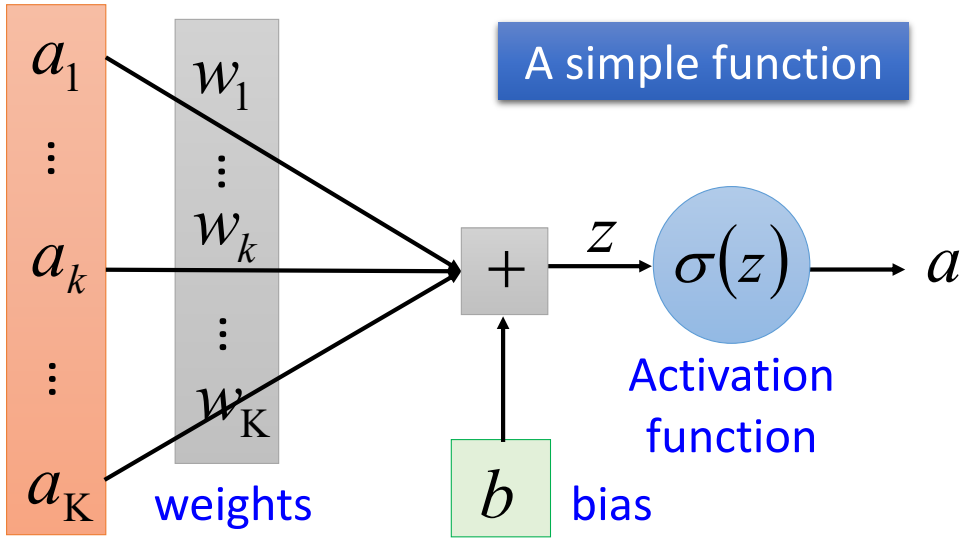
$f($  $) =$ "cat"



Neural Network

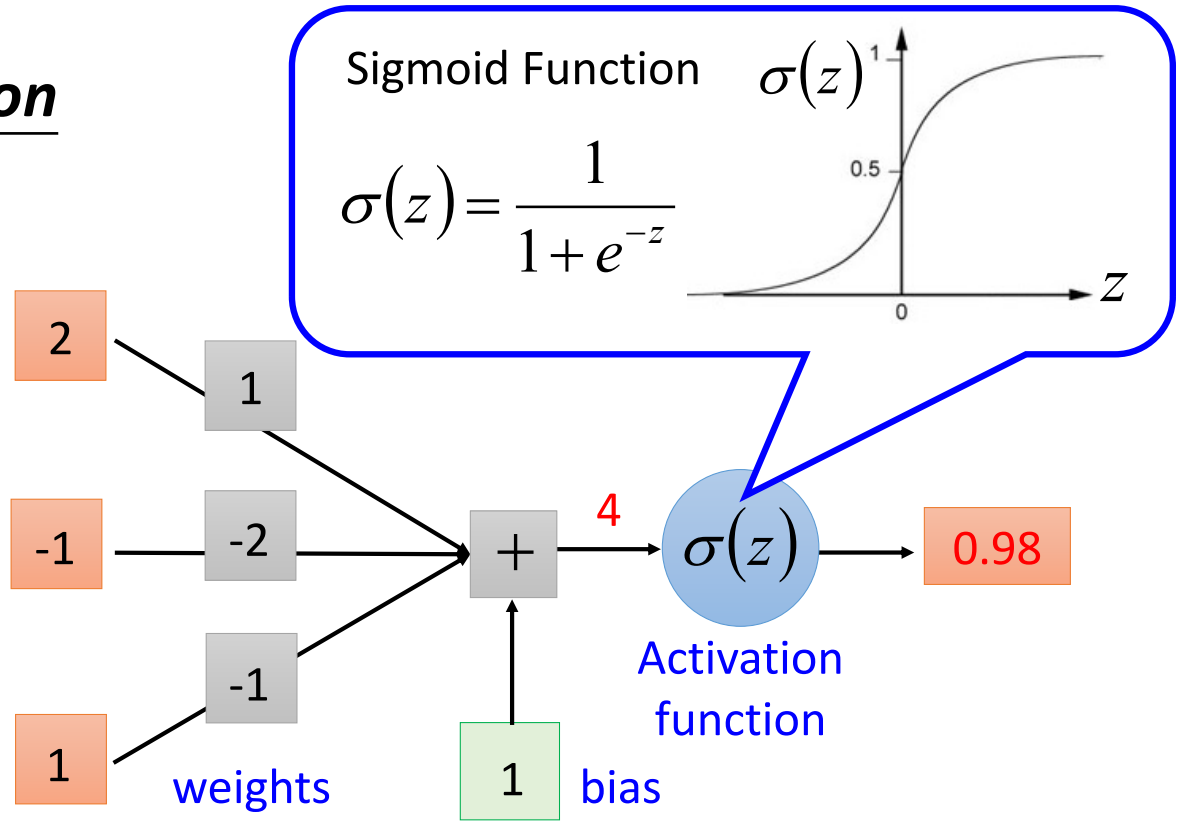
Neuron

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



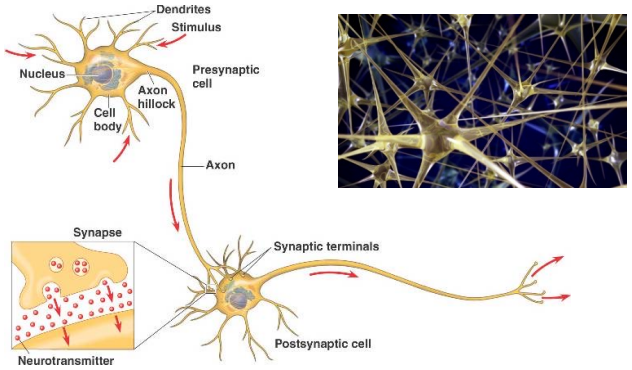
Neural Network

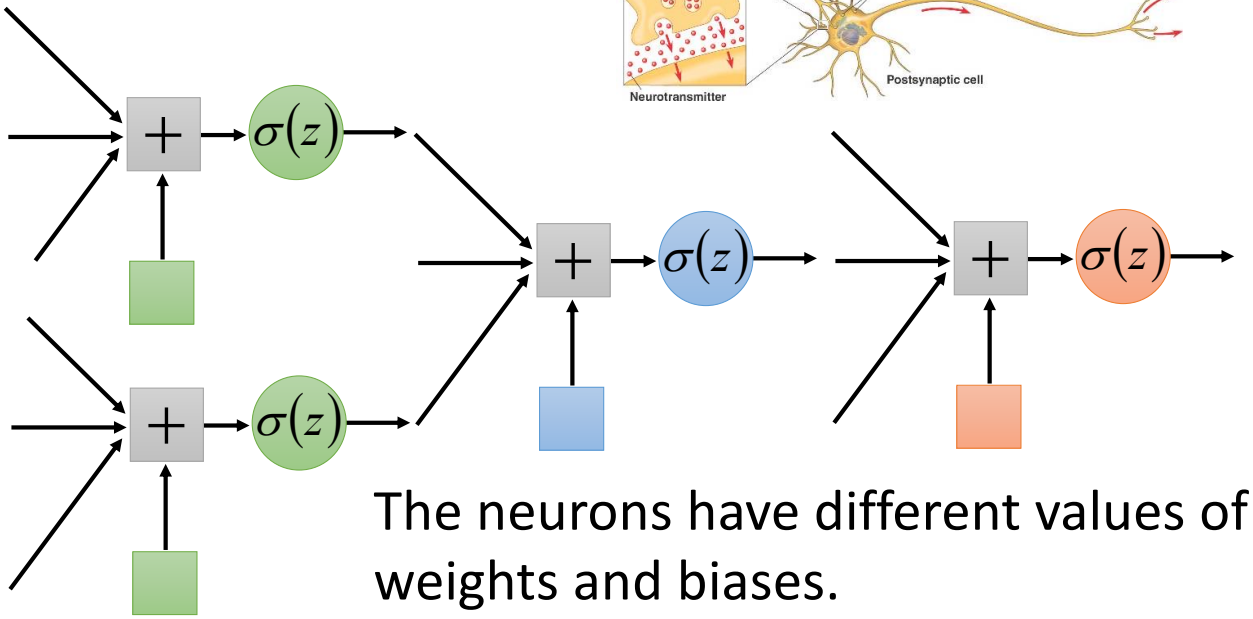
Neuron



Neural Network

Different connections lead to different network structures

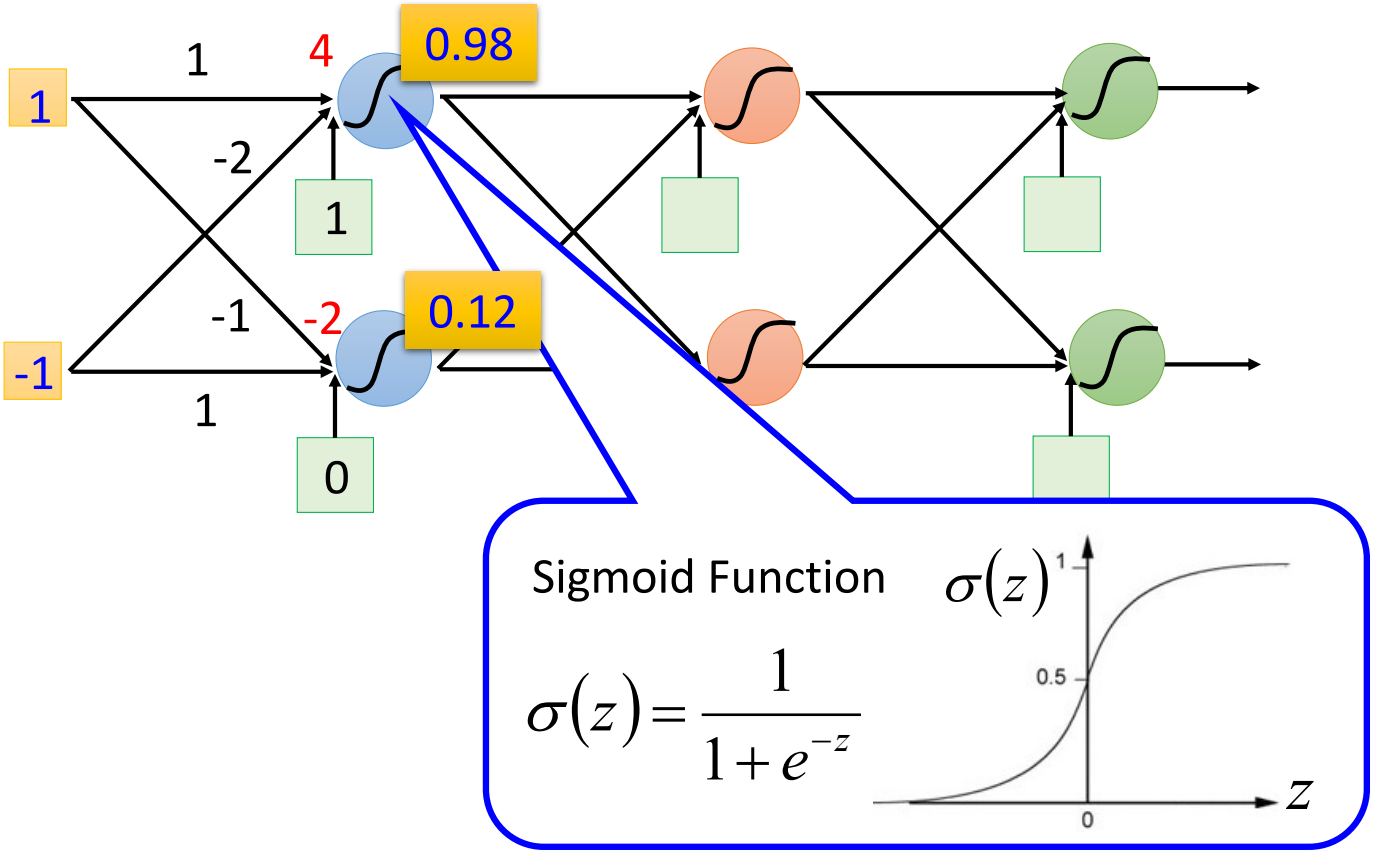




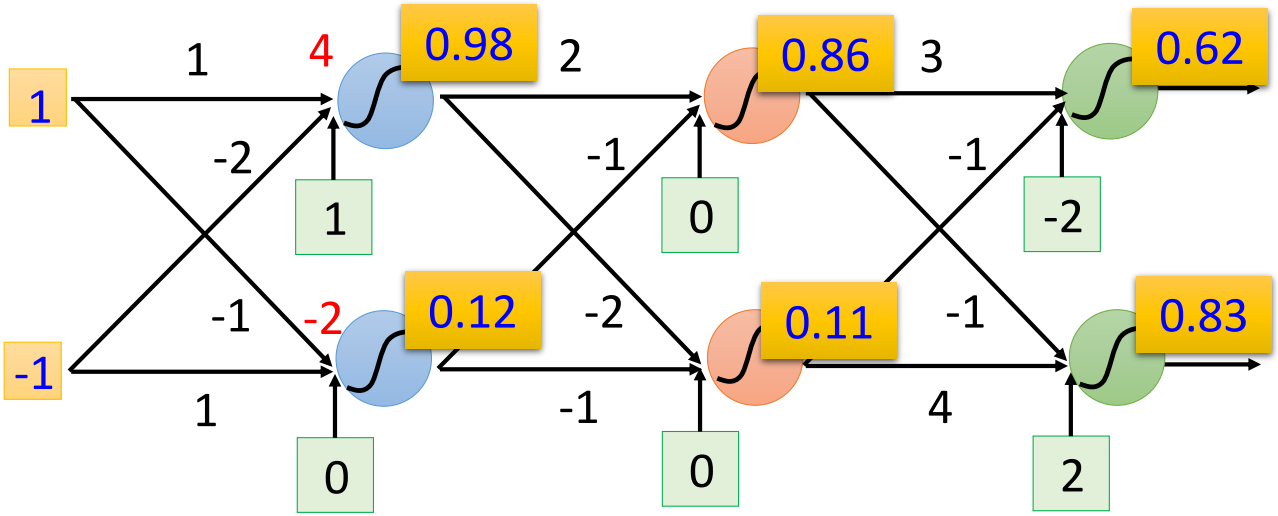
The neurons have different values of weights and biases.

Weights and biases are network parameters θ

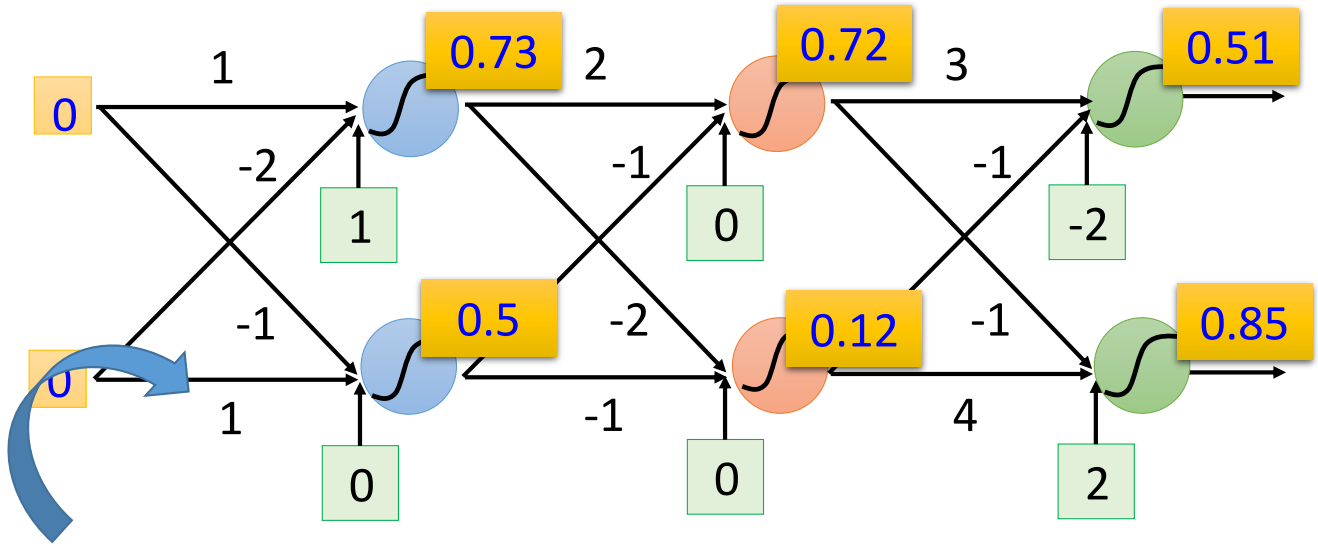
Fully Connect Feedforward Network



Fully Connect Feedforward Network



Fully Connect Feedforward Network



This is a function.

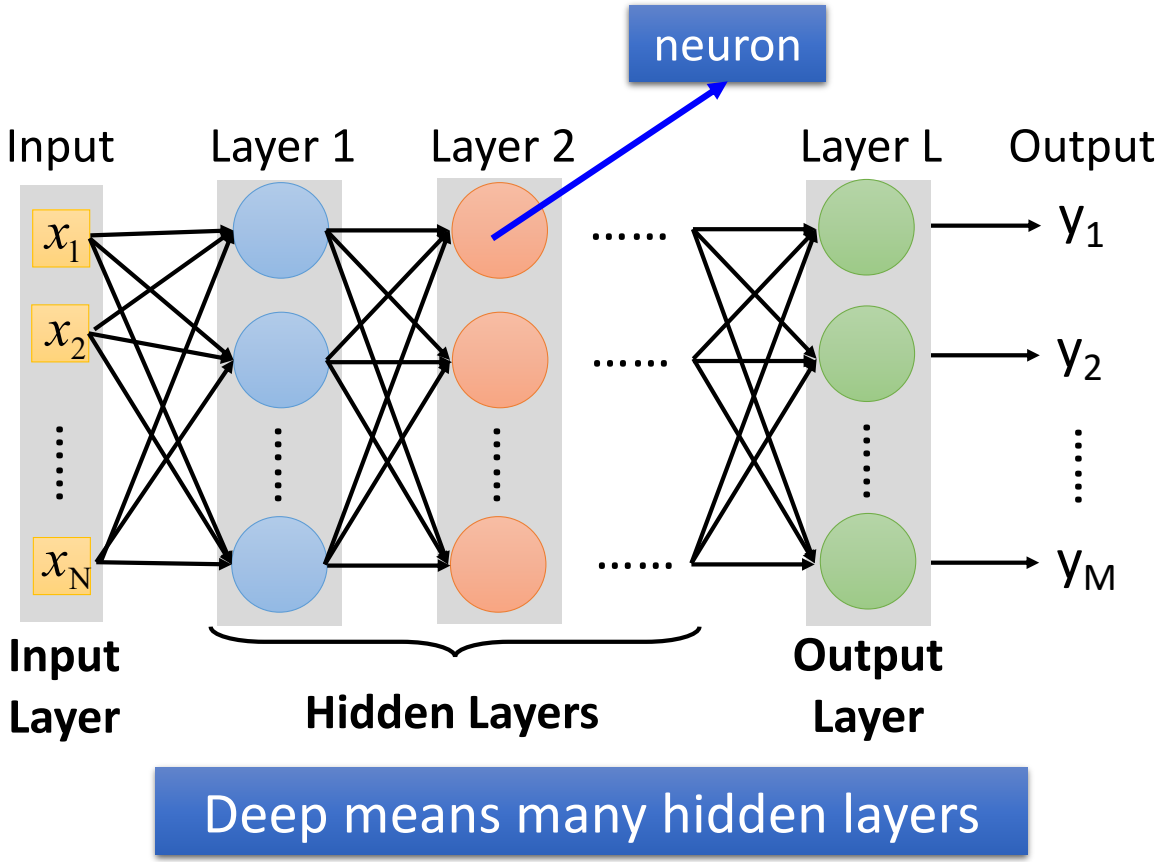
Input vector, output vector

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

Given parameters θ, define a function

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

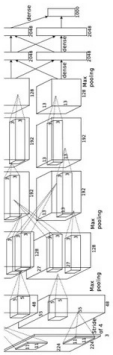
Fully Connect Feedforward Network



Deep means many hidden layers

Deep = Many hidden layers

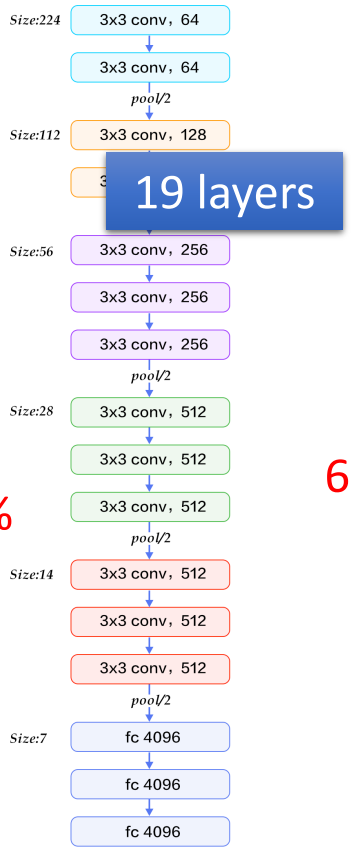
8 layers



16.4%

AlexNet (2012)

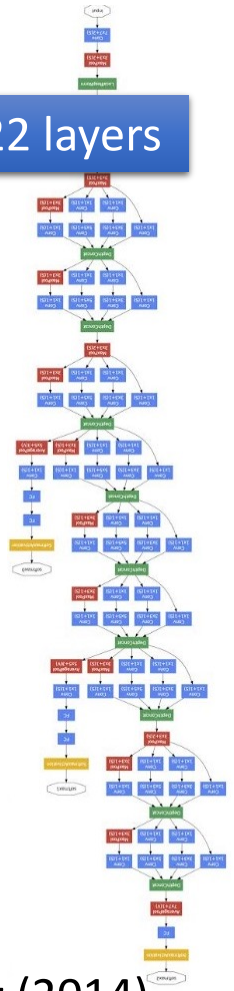
19 layers



7.3%

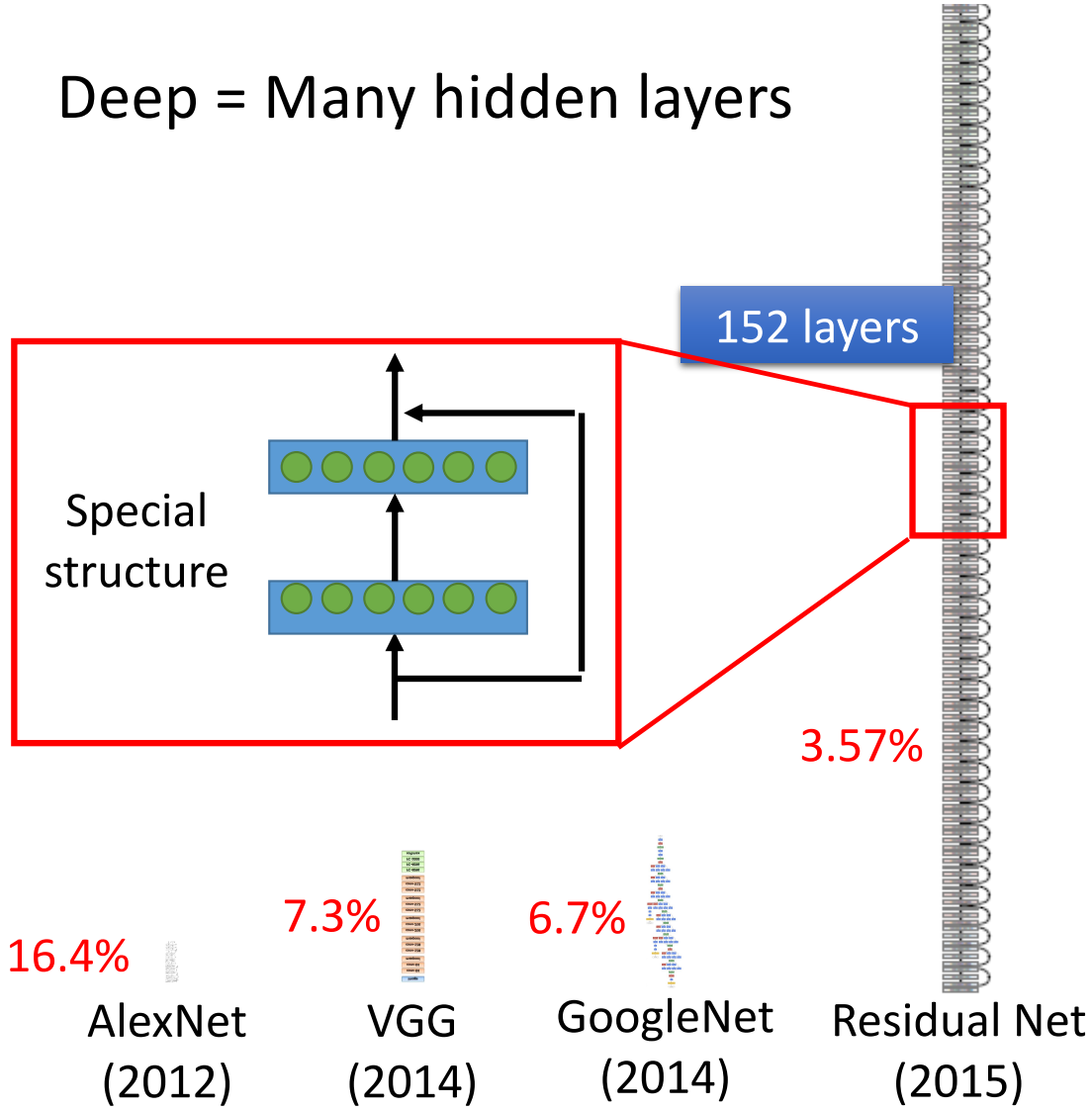
VGG (2014)

22 layers



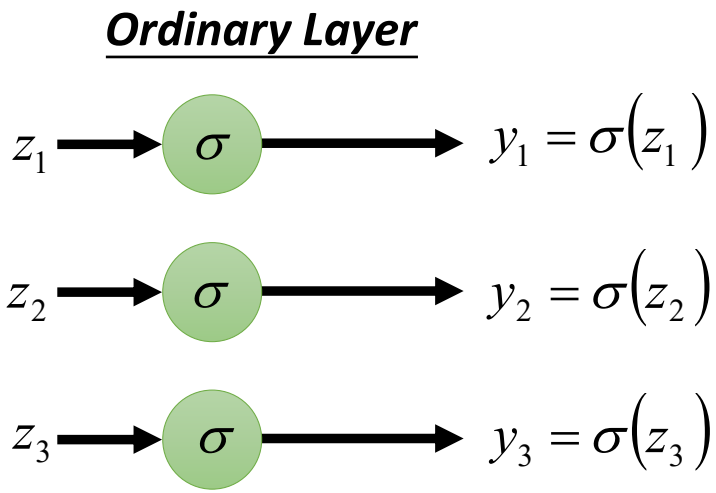
6.7%

GoogleNet (2014)


Deep = Many hidden layers
152 layers
Special structure
3.57%
7.3%
6.7%
16.4%
AlexNet (2012)
VGG (2014)
GoogleNet (2014)
Residual Net (2015)

Output Layer

- Softmax layer as the output layer



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

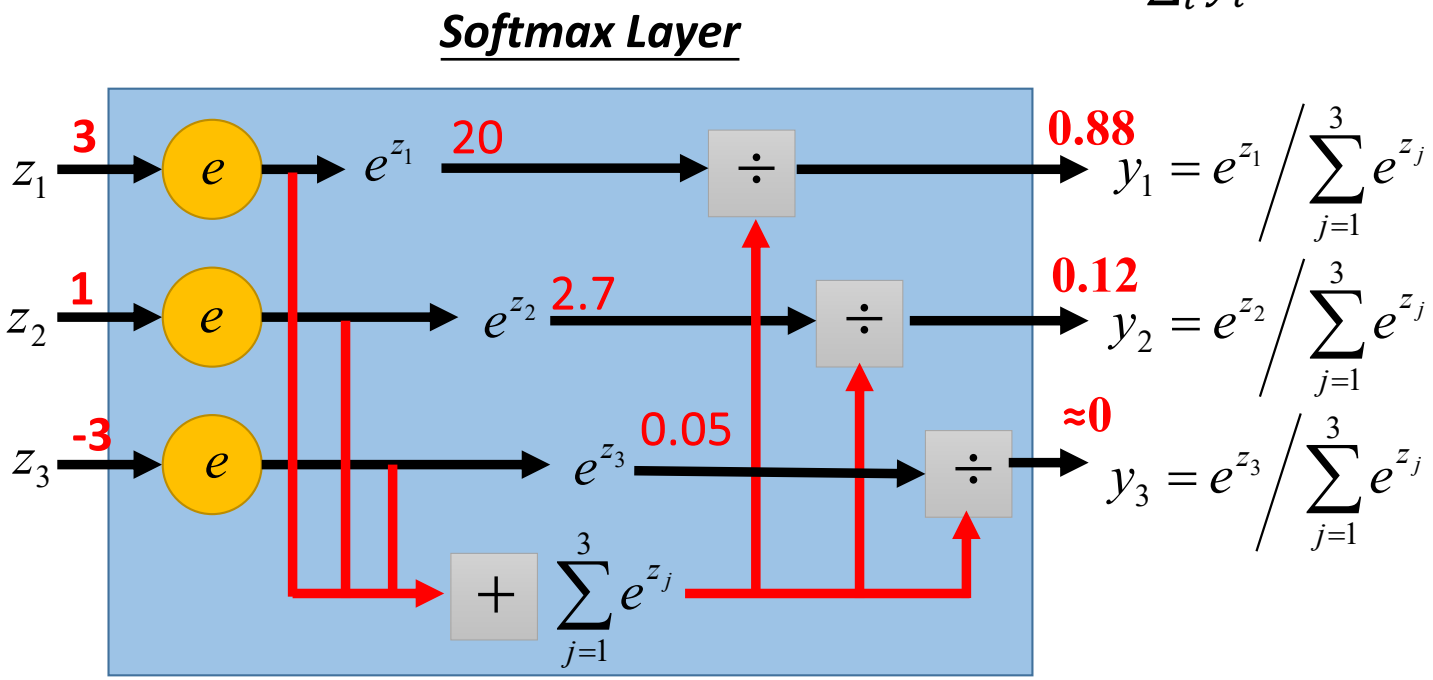
May not be easy to interpret

Output Layer

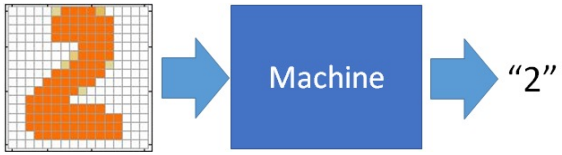
- Softmax layer as the output layer

Probability:

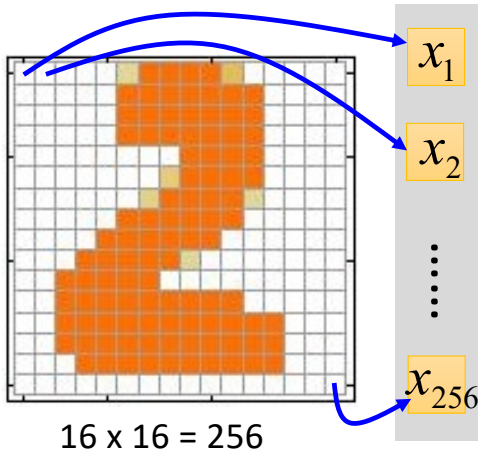
- $1 > y_i > 0$
- $\sum_i y_i = 1$



Example Application



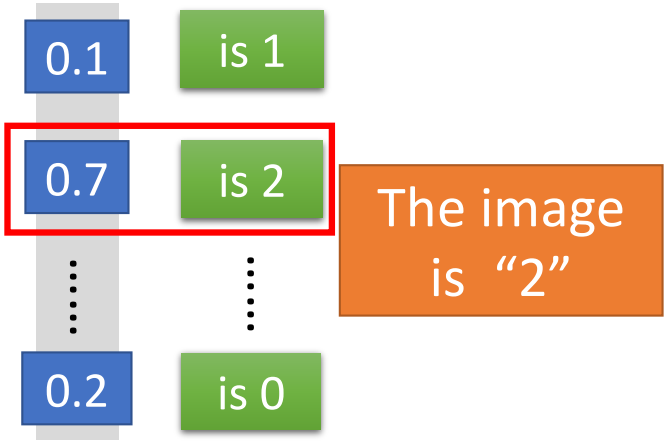
Input



Ink → 1

No ink → 0

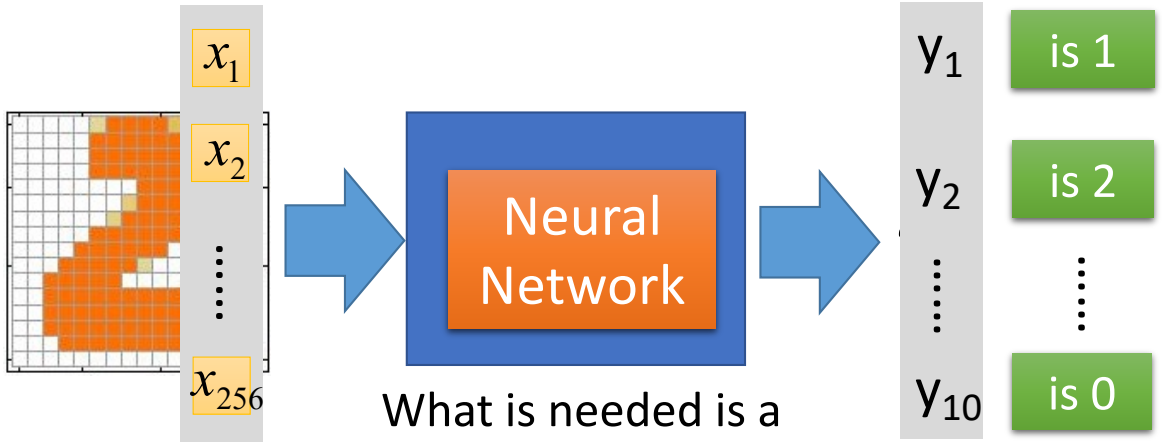
Output



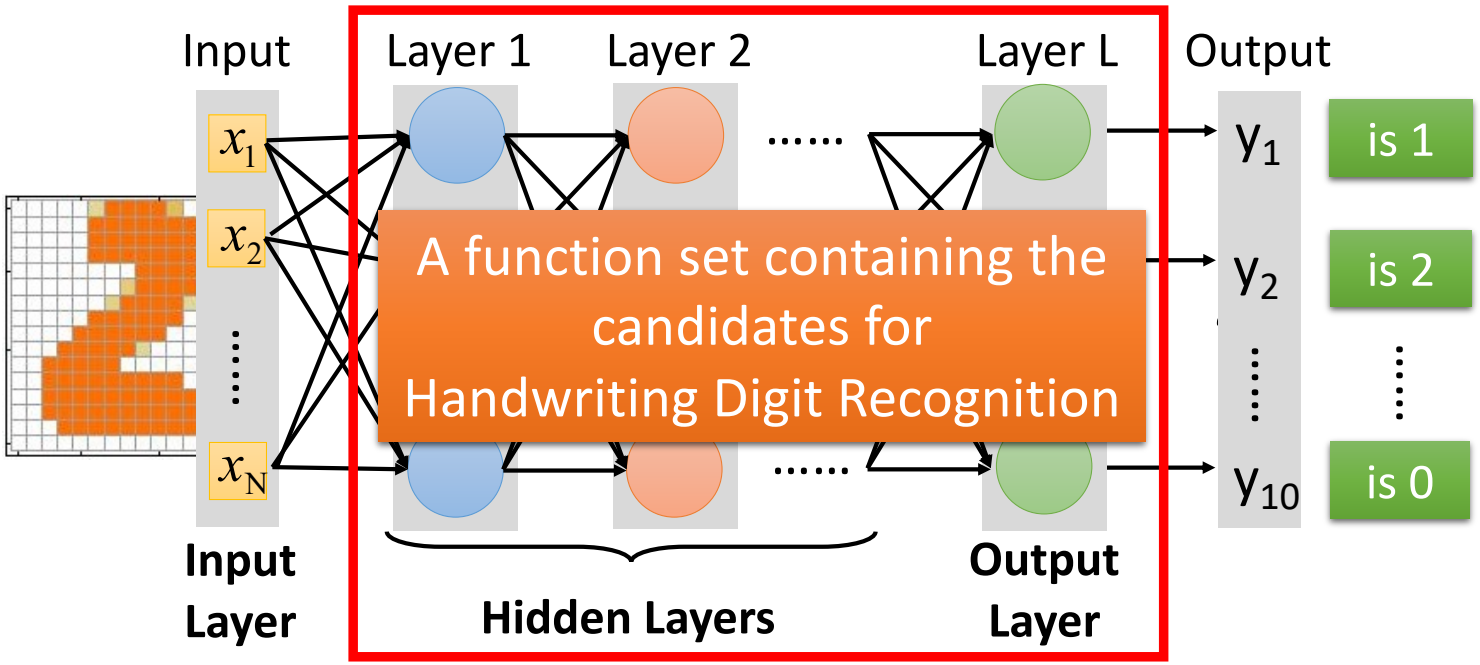
Each dimension represents the confidence of a digit.

Example Application

- Handwriting Digit Recognition

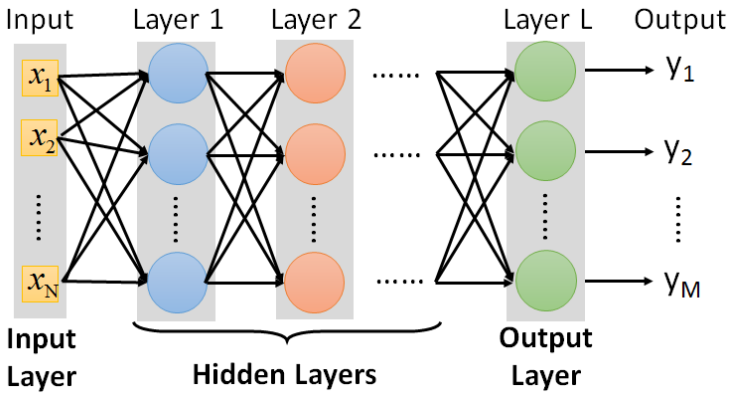


Example Application



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

FAQ



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

  Trial and Error + Intuition

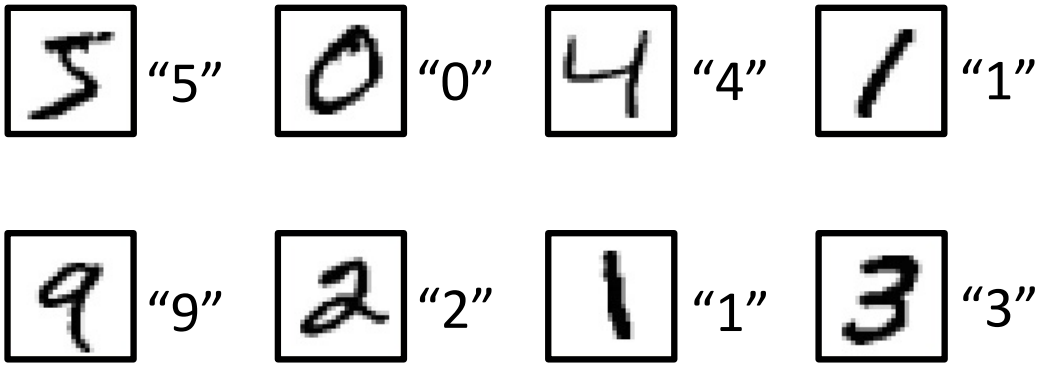
- Q: Can we design the network structure?

  Convolutional Neural Network (CNN)

- Q: Can the structure be automatically determined?
  - Yes, but not widely studied yet.

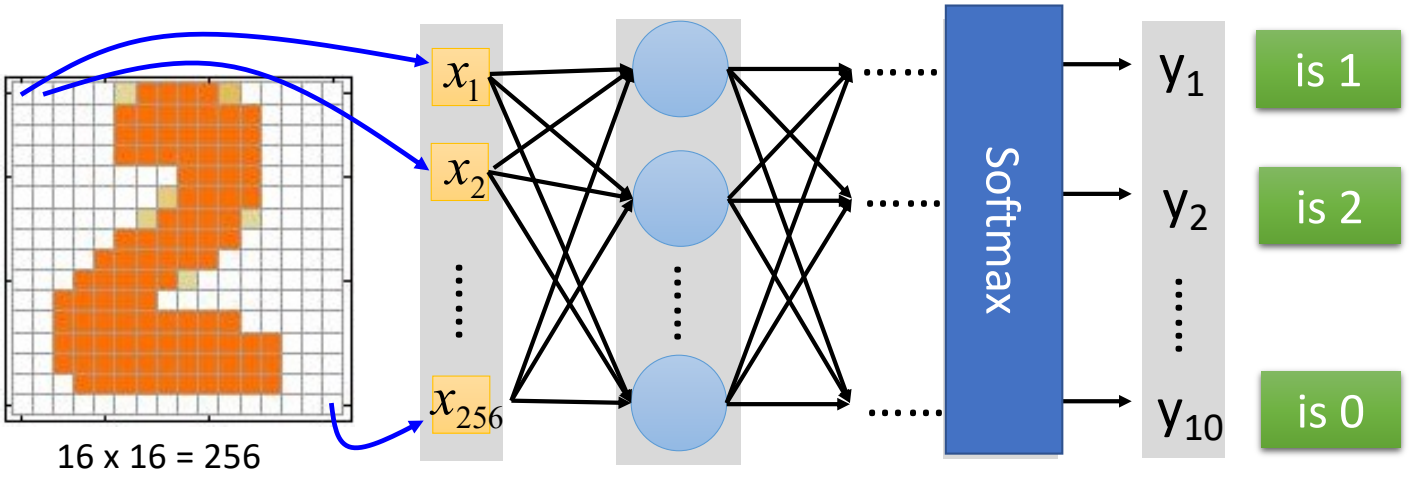
Training Data

- Preparing training data: images and their labels



The learning target is defined on the training data.

Learning Target



16 x 16 = 256

Ink → 1

No ink → 0

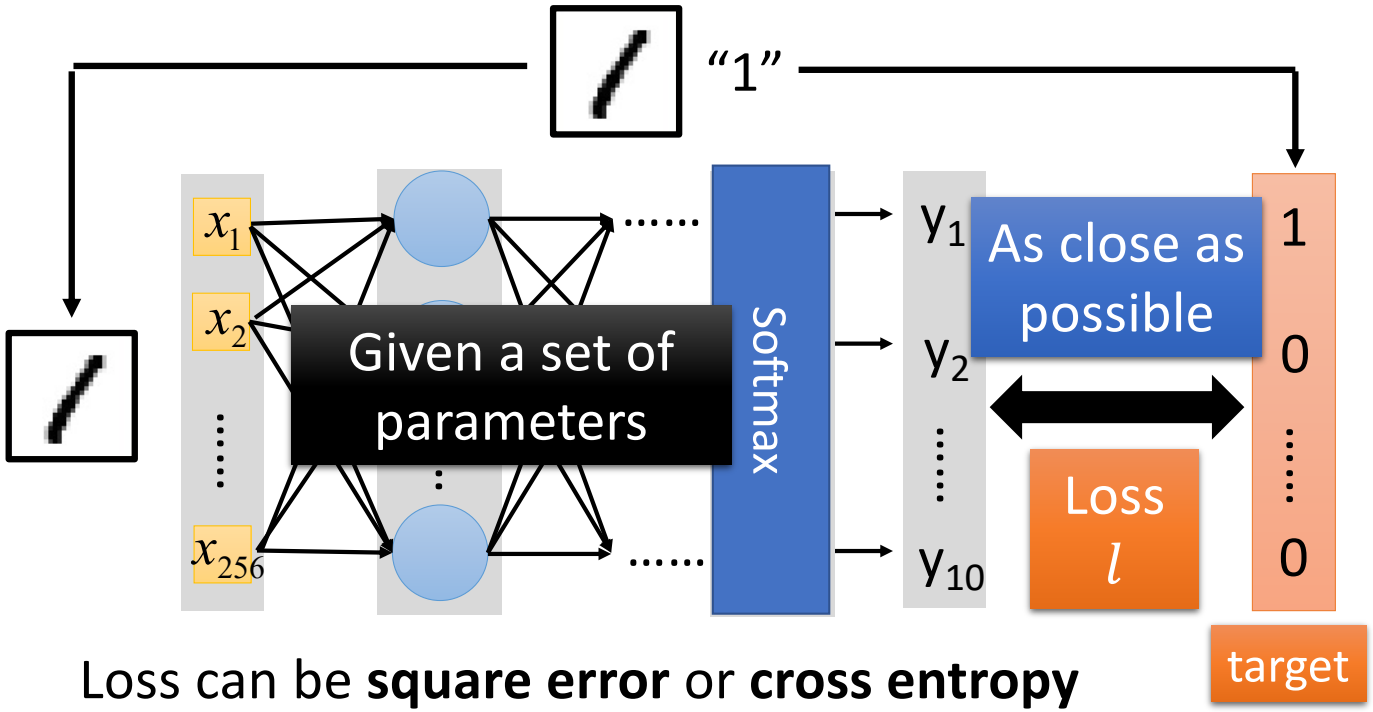
The learning target is ……

Input:  → y_1 has the maximum value

Input:  → y_2 has the maximum value

Loss

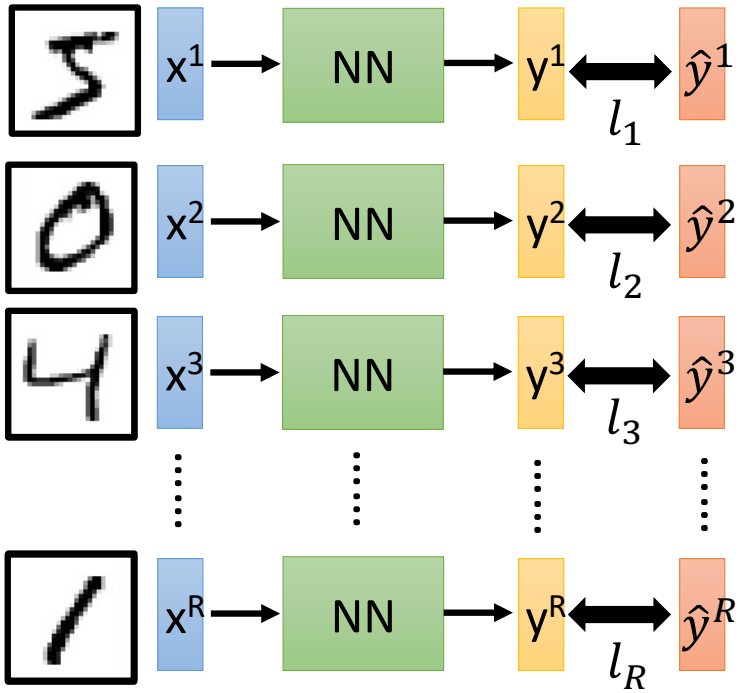
A good function should make the loss of all examples as small as possible.



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 minimizes total loss L

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

How to pick the best function

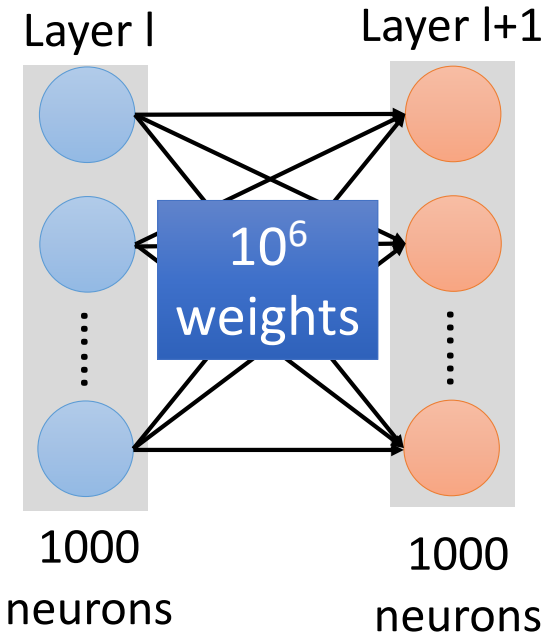
Find ***network parameters*** θ^* that minimize total loss L

Enumerate all possible values

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

Millions of parameters

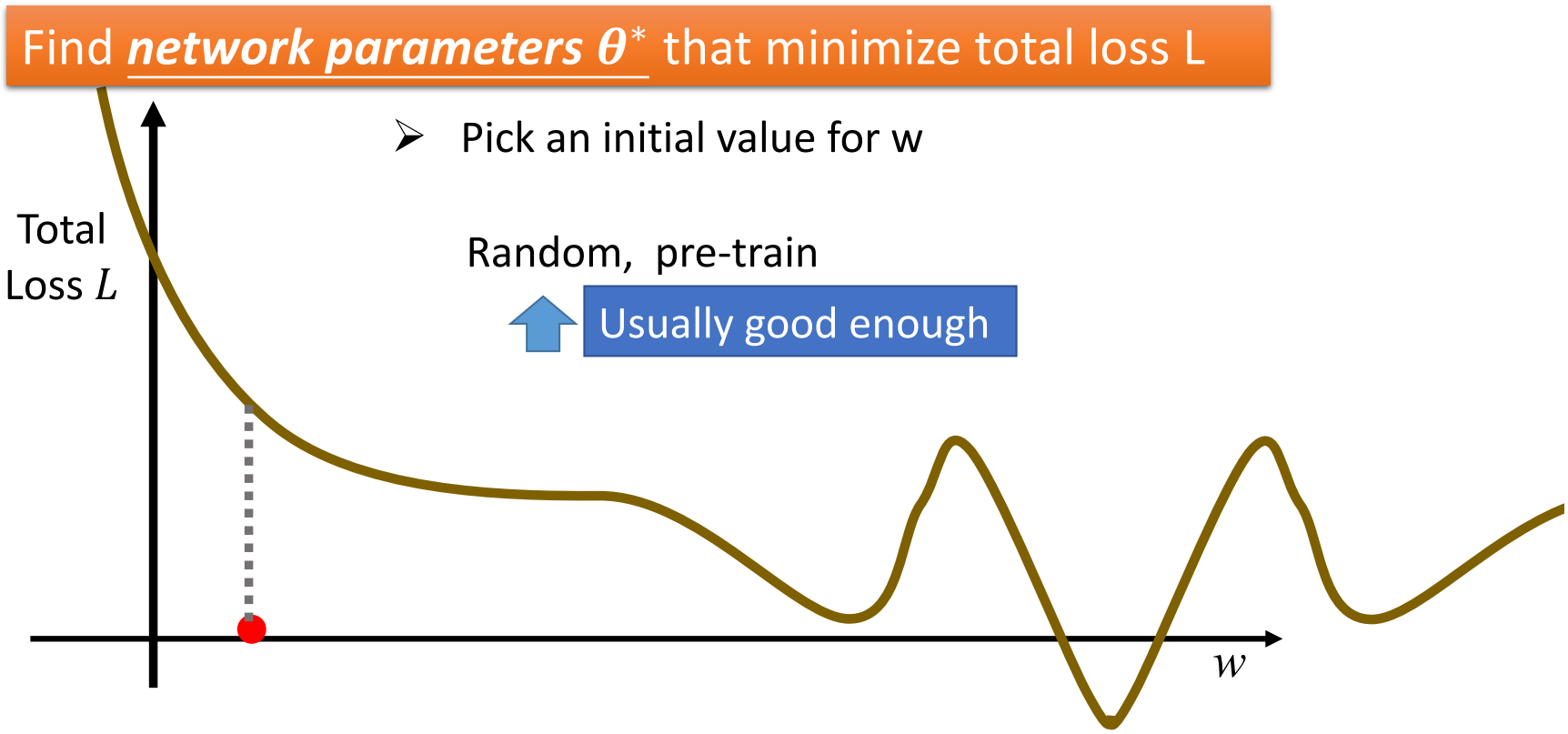
E.g. speech recognition: 8 layers and 1000 neurons each layer



Gradient Descent

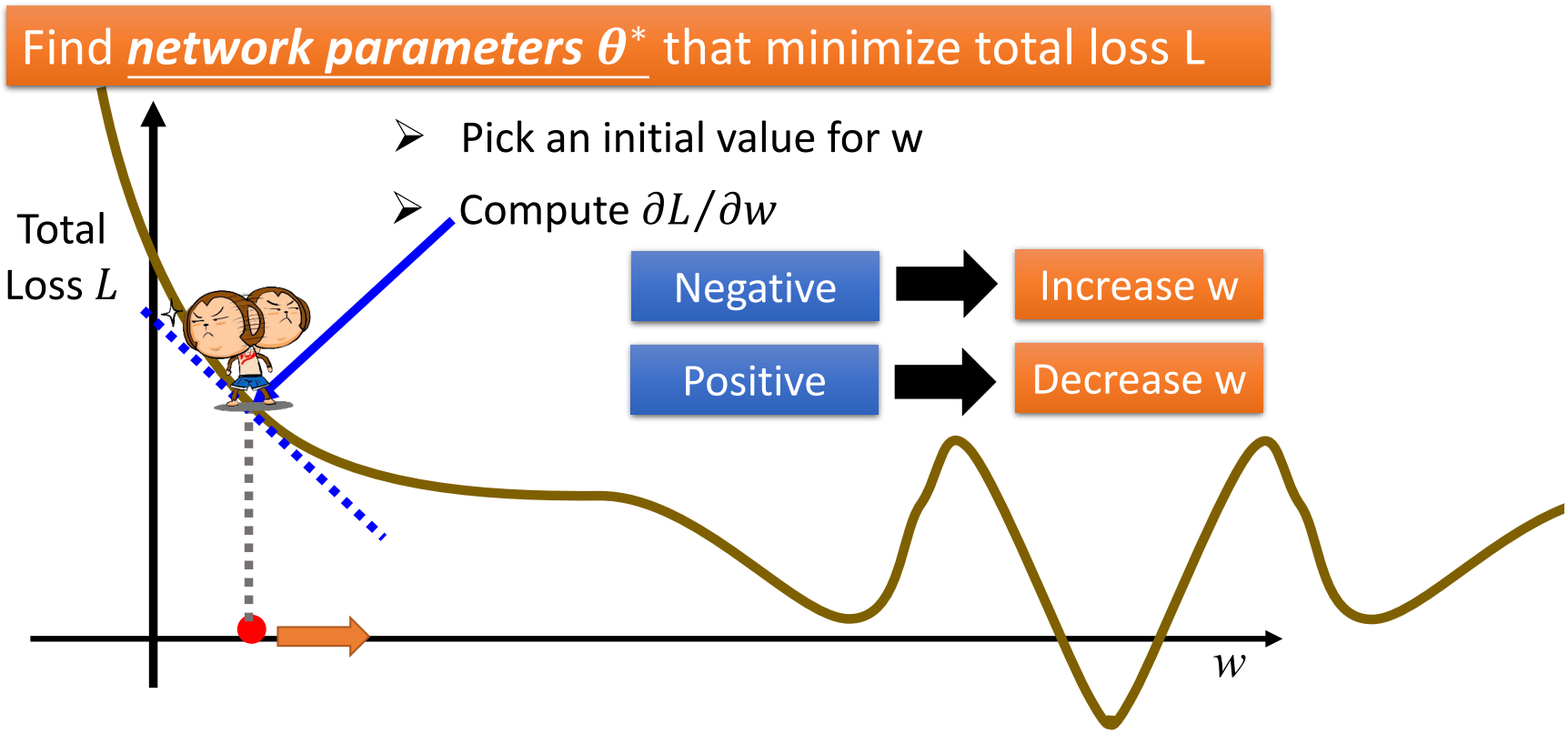
Network parameters $\theta = \{w_1, w_2, \cdots, b_1, b_2, \cdots\}$

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



Gradient Descent

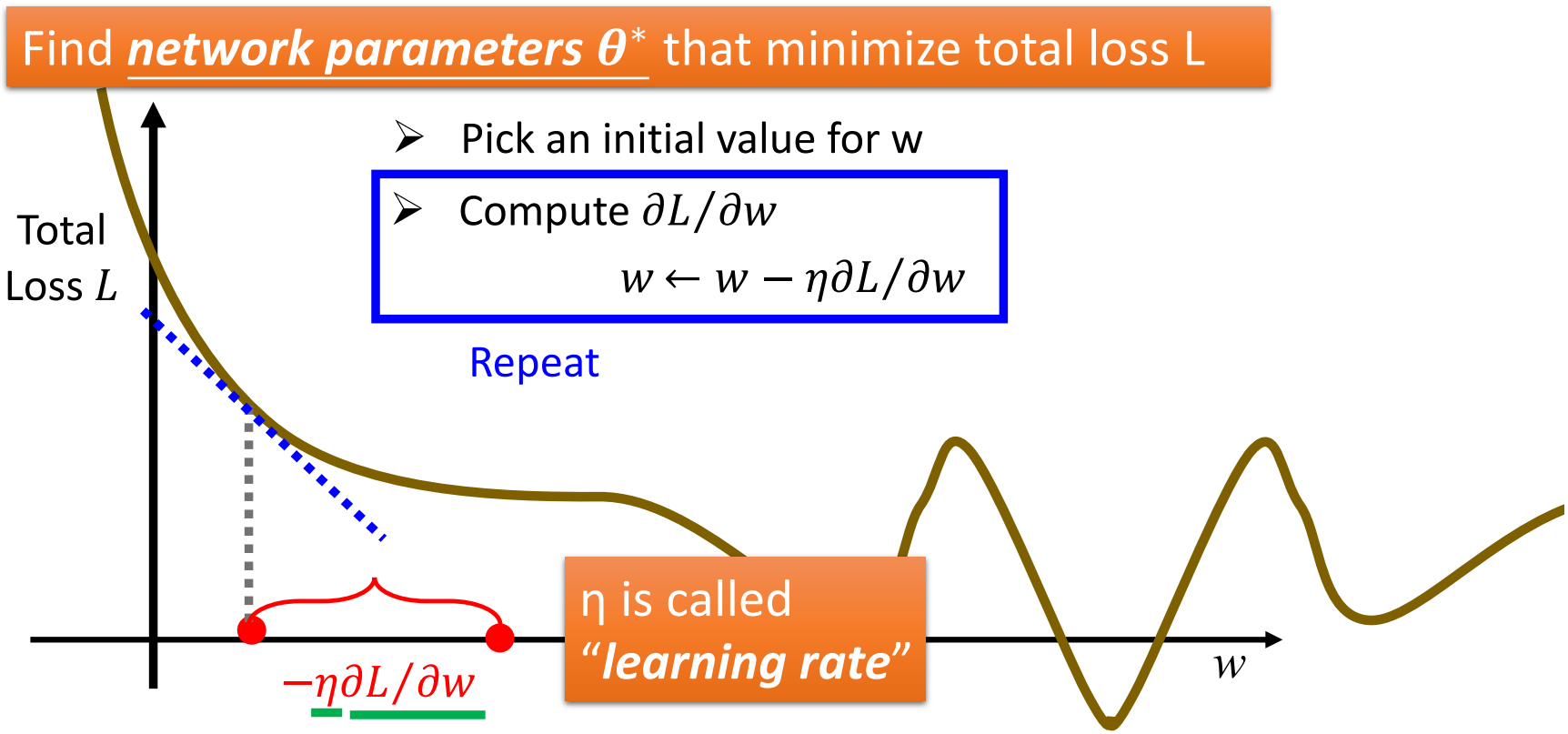
Network parameters $\theta = \{w_1, w_2, \cdots, b_1, b_2, \cdots\}$



Gradient Descent

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

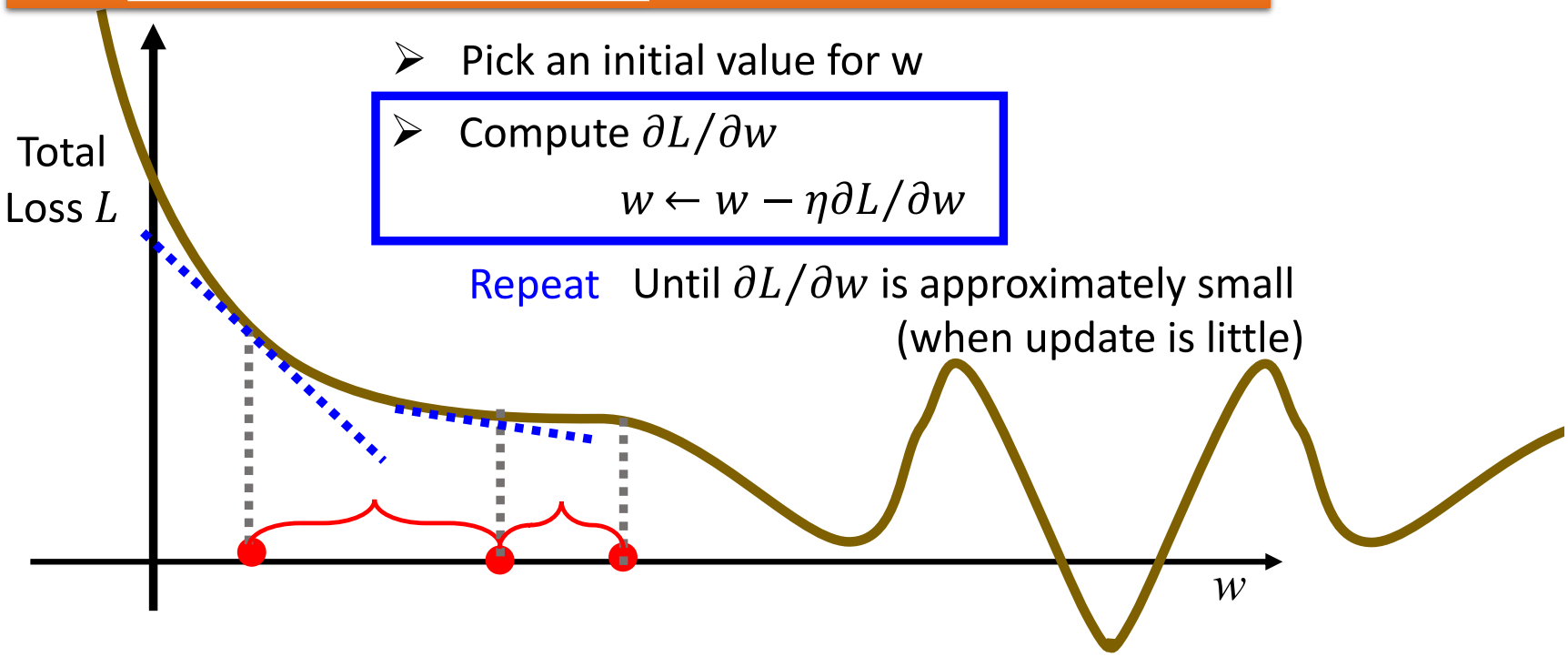
Find ***network parameters*** θ^* that minimize total loss L



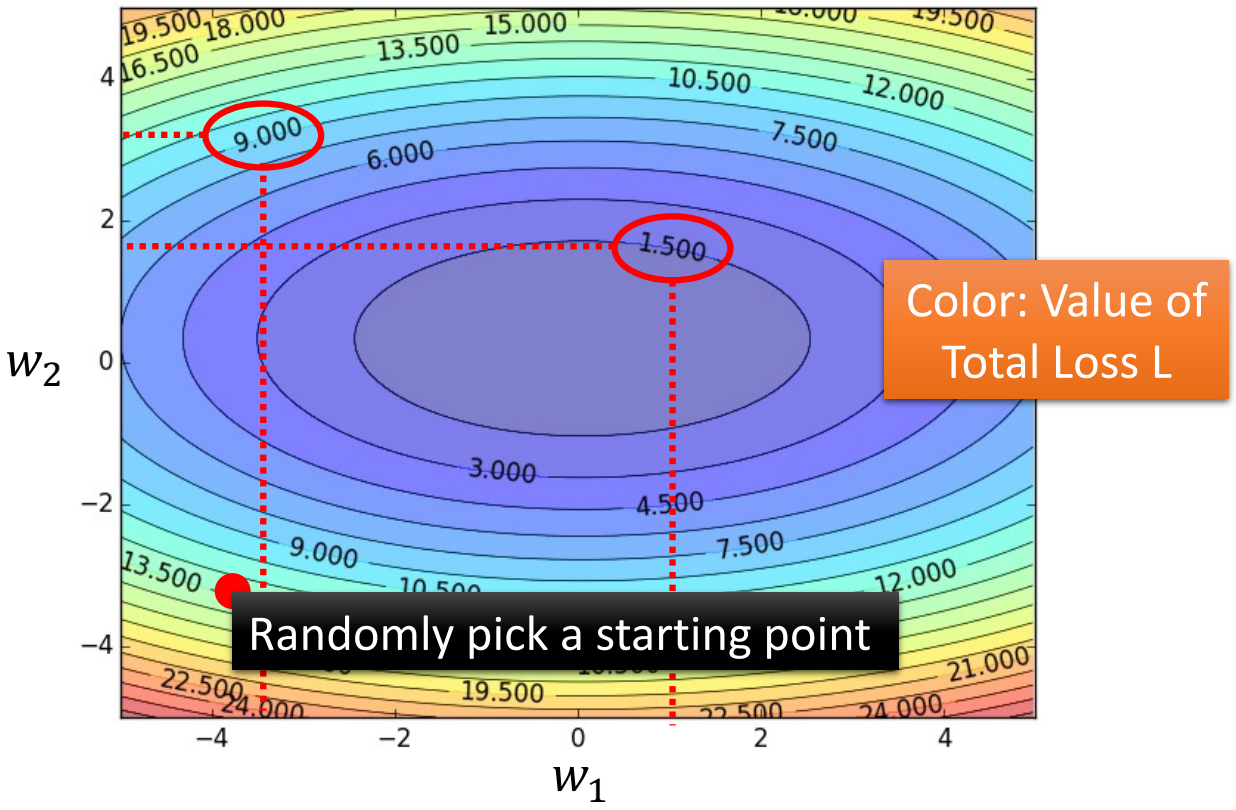
Gradient Descent

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

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

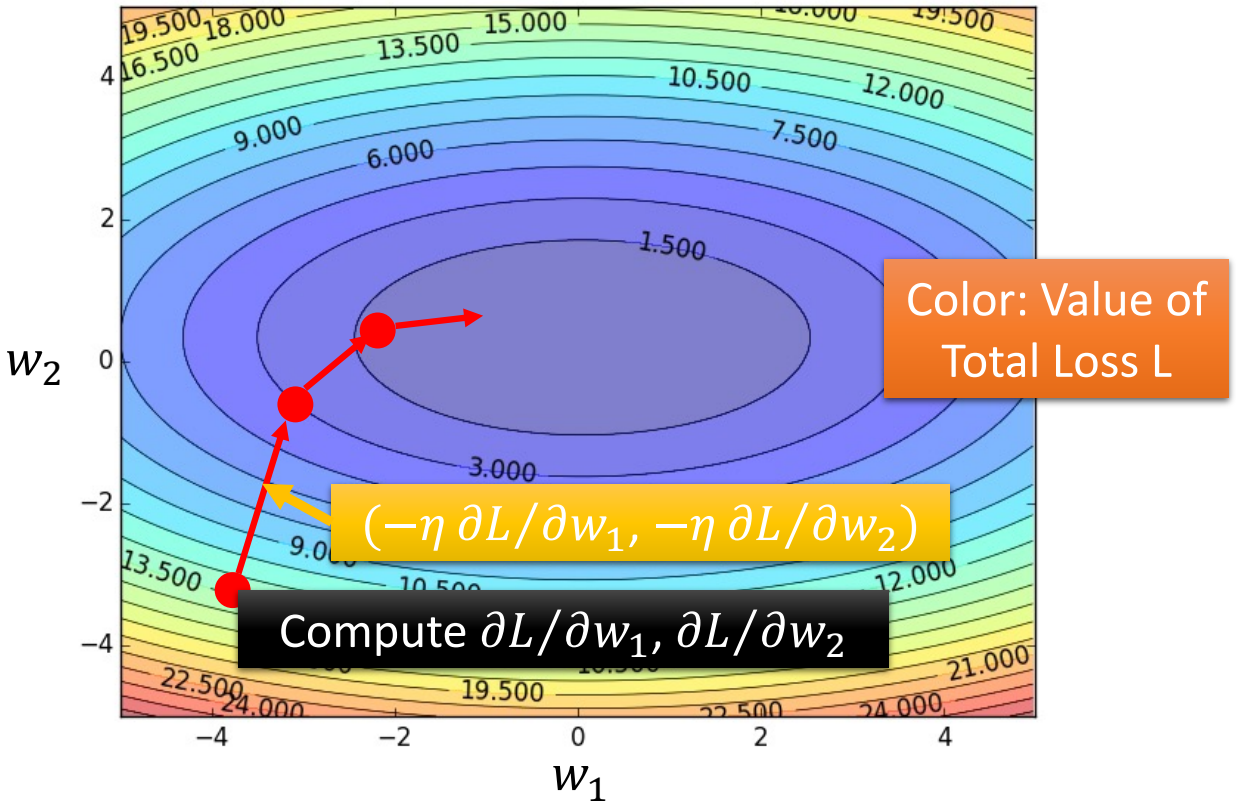


Gradient Descent

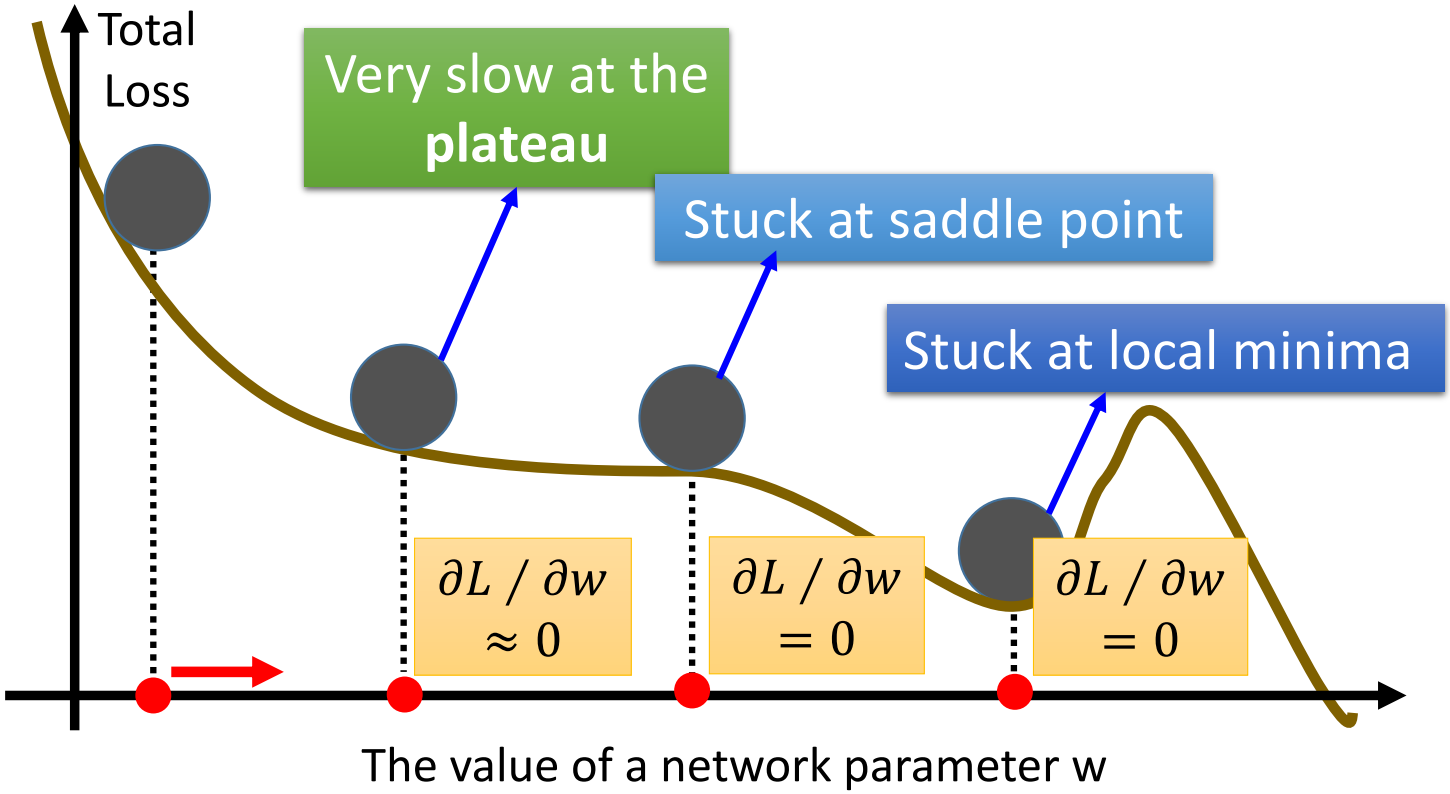


Gradient Descent

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

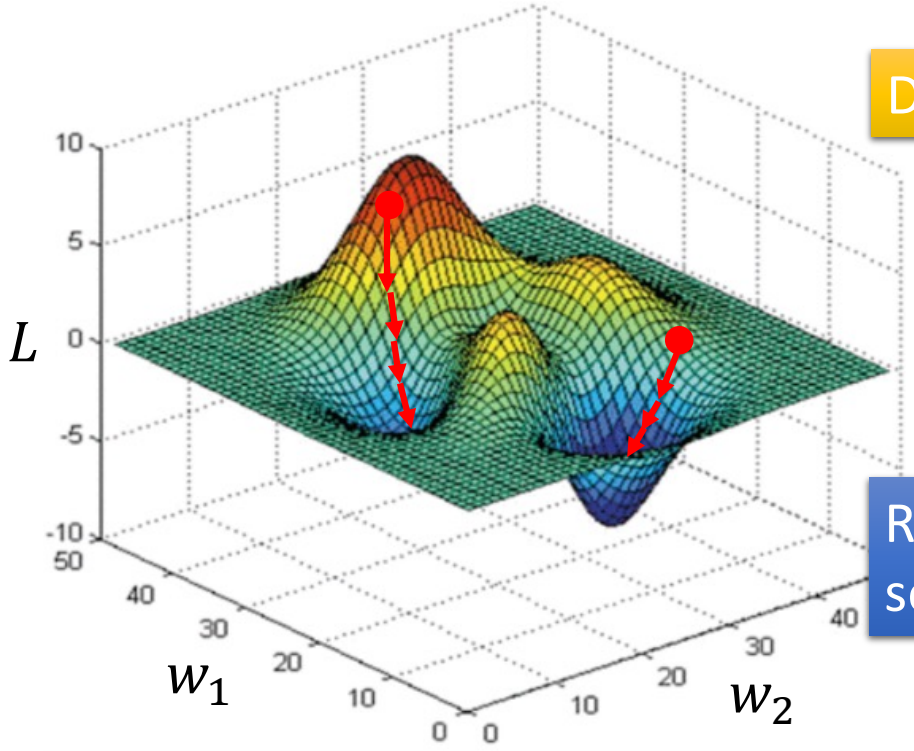


Local Minima



Local Minima

- Gradient descent never guarantee global minima



Different initial point

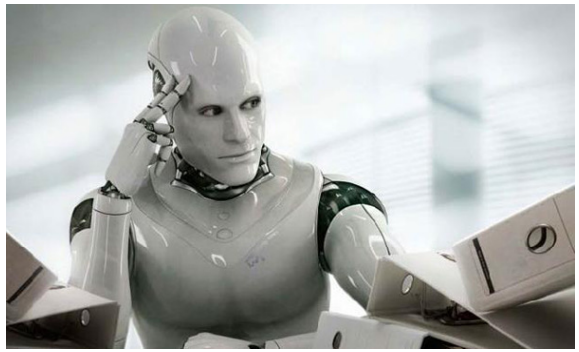
Reach different minima, so different results

Gradient Descent

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

➡ Even alpha go using this approach.

People image ……

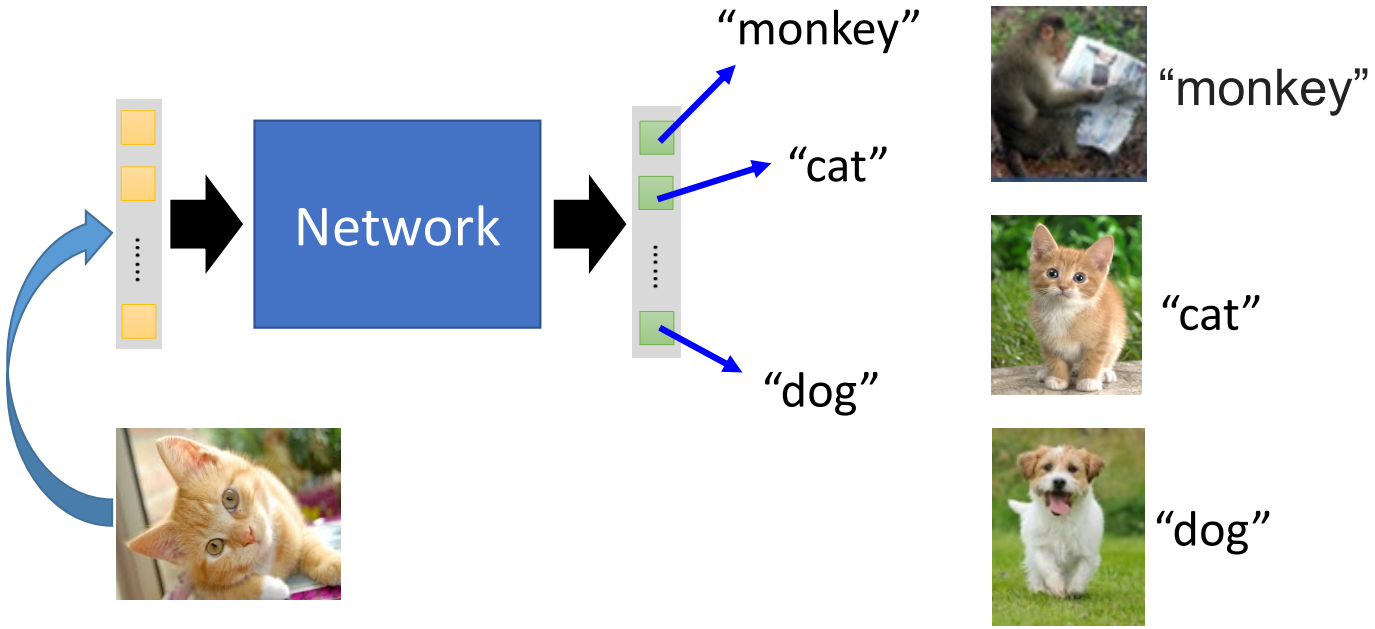


Actually …..

I hope you are not too disappointed

For example, you can do …….

- Image Recognition



For example, you can do …….

Backpropagation

- Backpropagation: an efficient way to compute $\partial L / \partial w$ in neural network

Decision Trees vs Neural Networks

Decision Trees and Tree ensembles

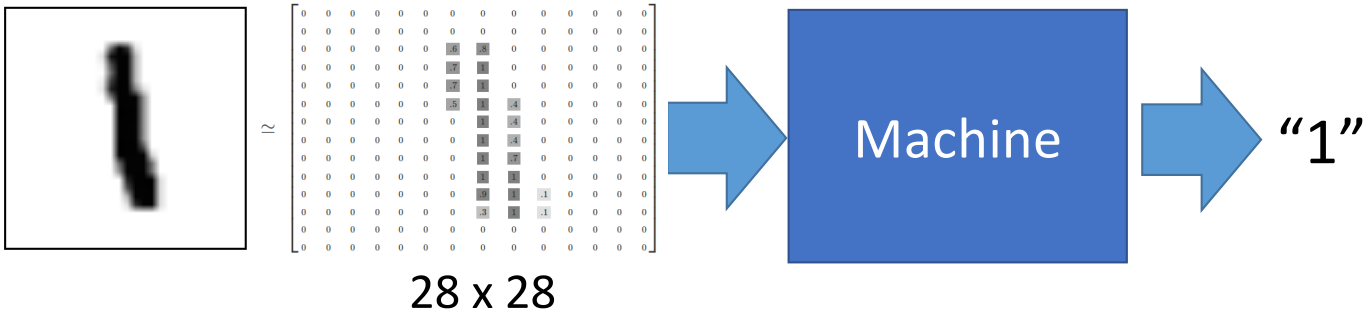
- Works well on tabular (structured) data
- Not recommended for unstructured data (images, audio, text)
- Fast
- Small decision trees may be human interpretable

Neural Networks

- Works well on all types of data, including tabular (structured) and unstructured data
- May be slower than a decision tree
- Works with transfer learning
- When building a system of multiple models working together, it might be easier to string together multiple neural networks

Example Application

- Handwriting Digit Recognition

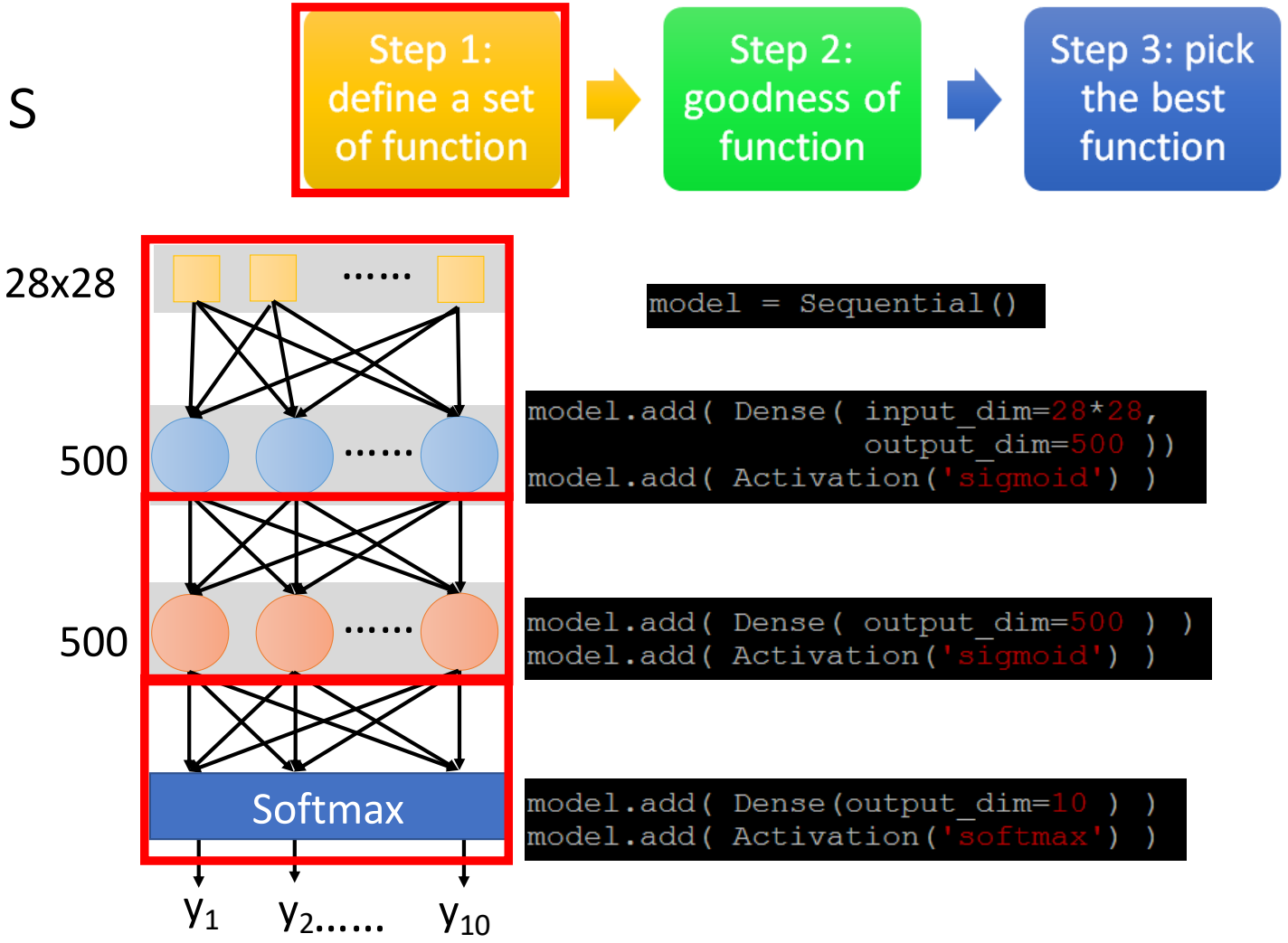


MNIST Data: http://yann.lecun.com/exdb/mnist/

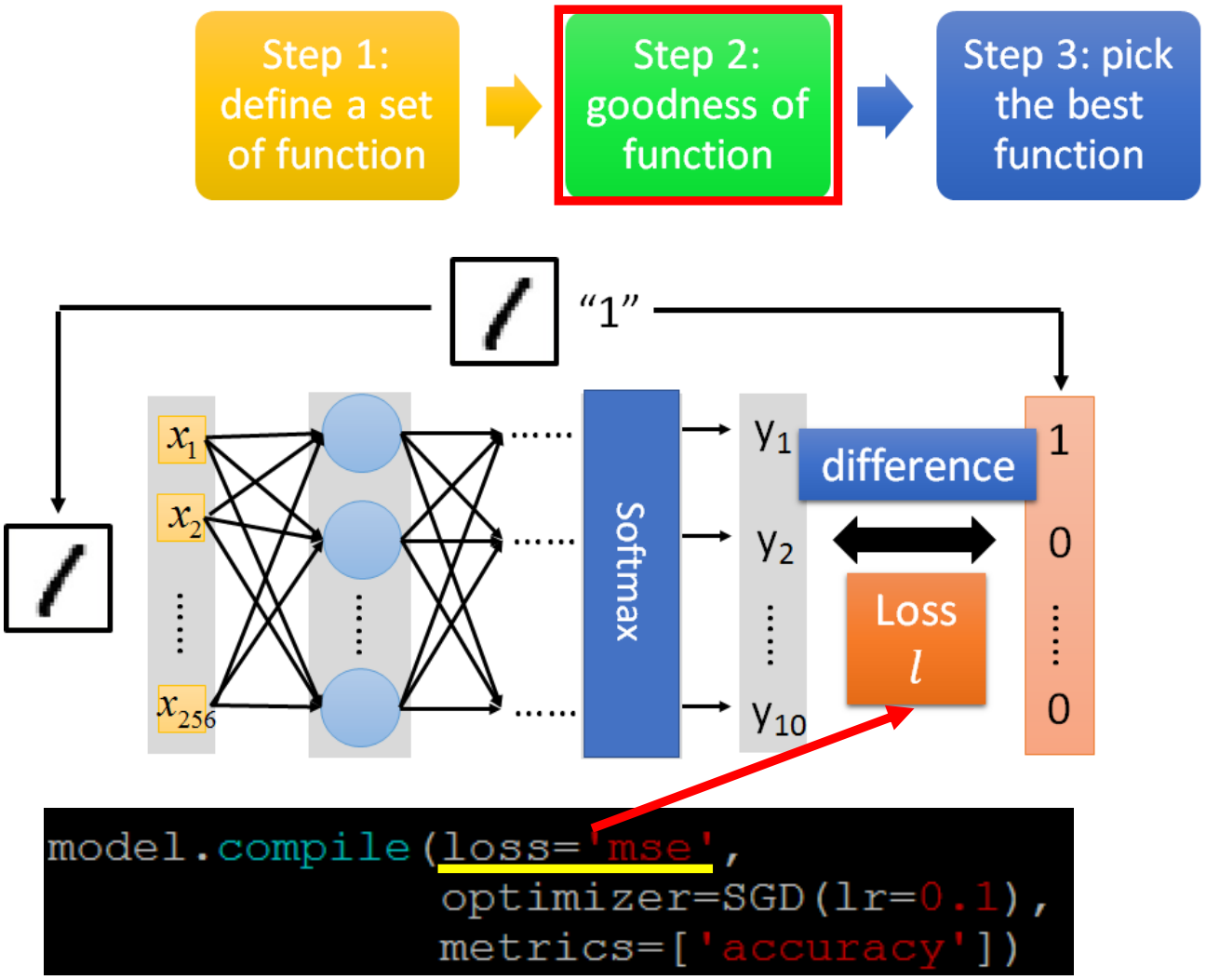
“Hello world” for deep learning

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

Keras


Step 1: define a set of function
Step 2: goodness of function
Step 3: pick the best function
28x28
500
500
Softmax
y1
y2.....
y10
model = Sequential()
model.add(Dense(input_dim=28*28, output_dim=500))
model.add(Activation('sigmoid'))
model.add(Dense(output_dim=500))
model.add(Activation('sigmoid'))
model.add(Dense(output_dim=10))
model.add(Activation('softmax'))

Keras


Step 1: define a set of function
Step 2: goodness of function
Step 3: pick the best function
"1"
x_1
x_2
x_{256}
Softmax
y_1
y_2
y_{10}
difference
Loss l
1
0
0
model.compile(loss='mse',
optimizer=SGD(lr=0.1),
metrics=['accuracy'])

Keras

Step 3.1: Configuration

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

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

0.1

Step 3.2: Find the optimal network parameters

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

Training data (Images)

Labels (digits)

Keras

Step 3.2: Find the optimal network parameters

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

numpy array

28 x 28 =784

.....

Number of training examples

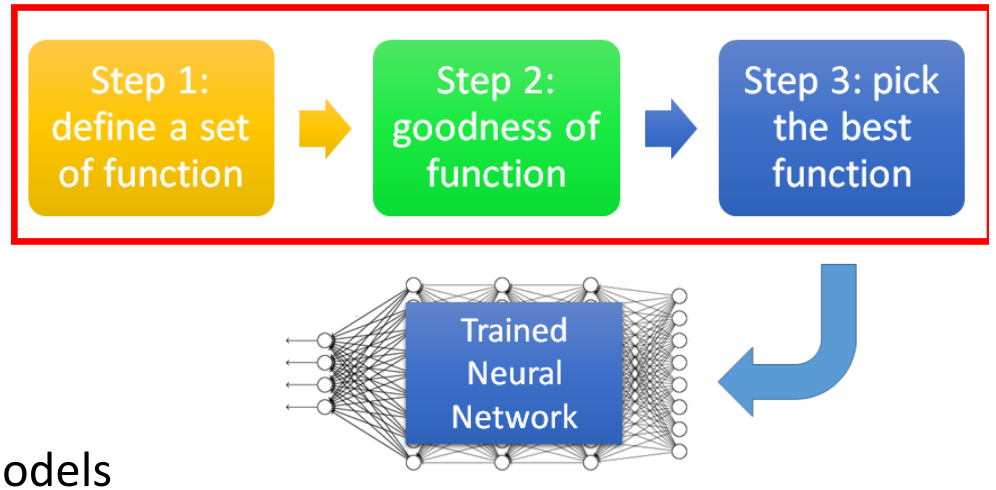
numpy array

10

.....

Number of training examples

Keras



Save and load models

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

How to use the neural network (testing):

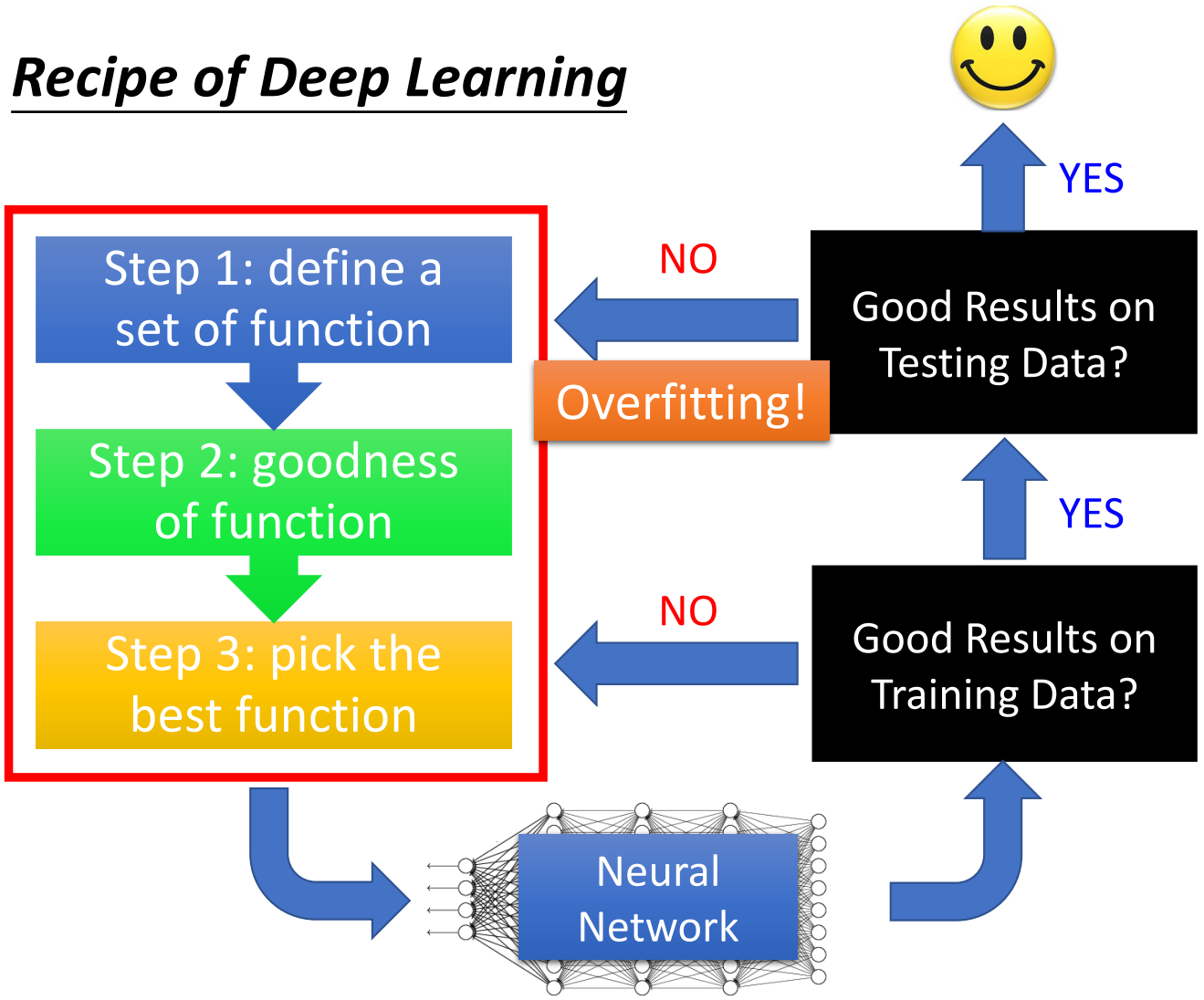
case 1:

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

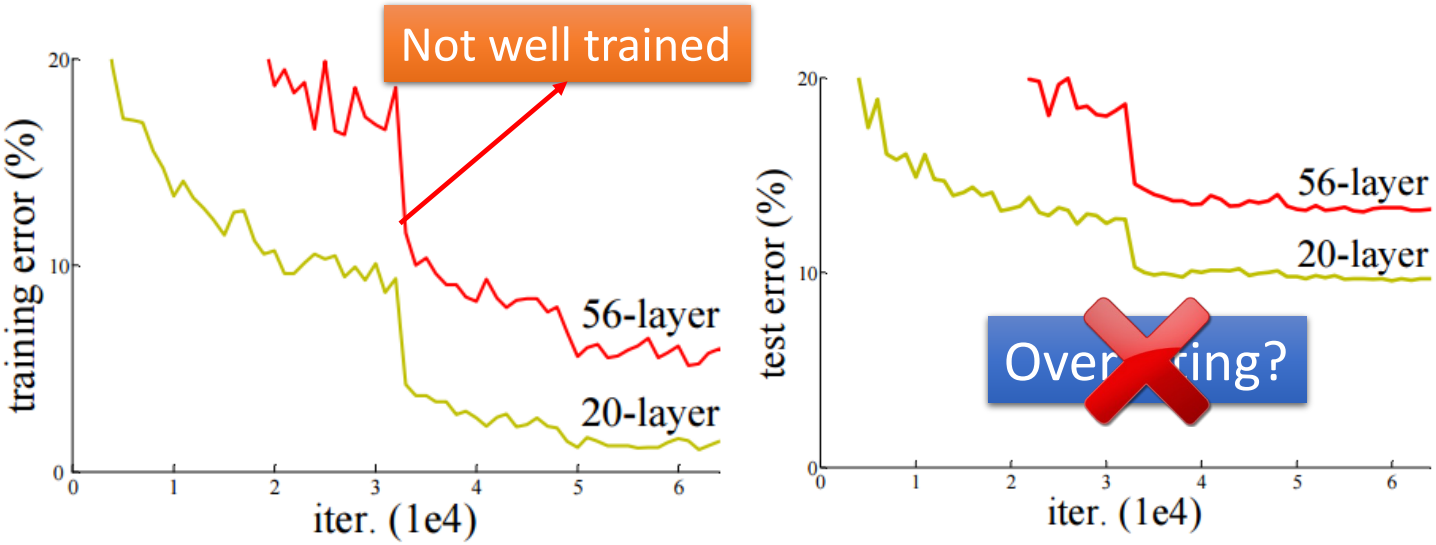
case 2:

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

Recipe of Deep Learning



Do not always blame Overfitting

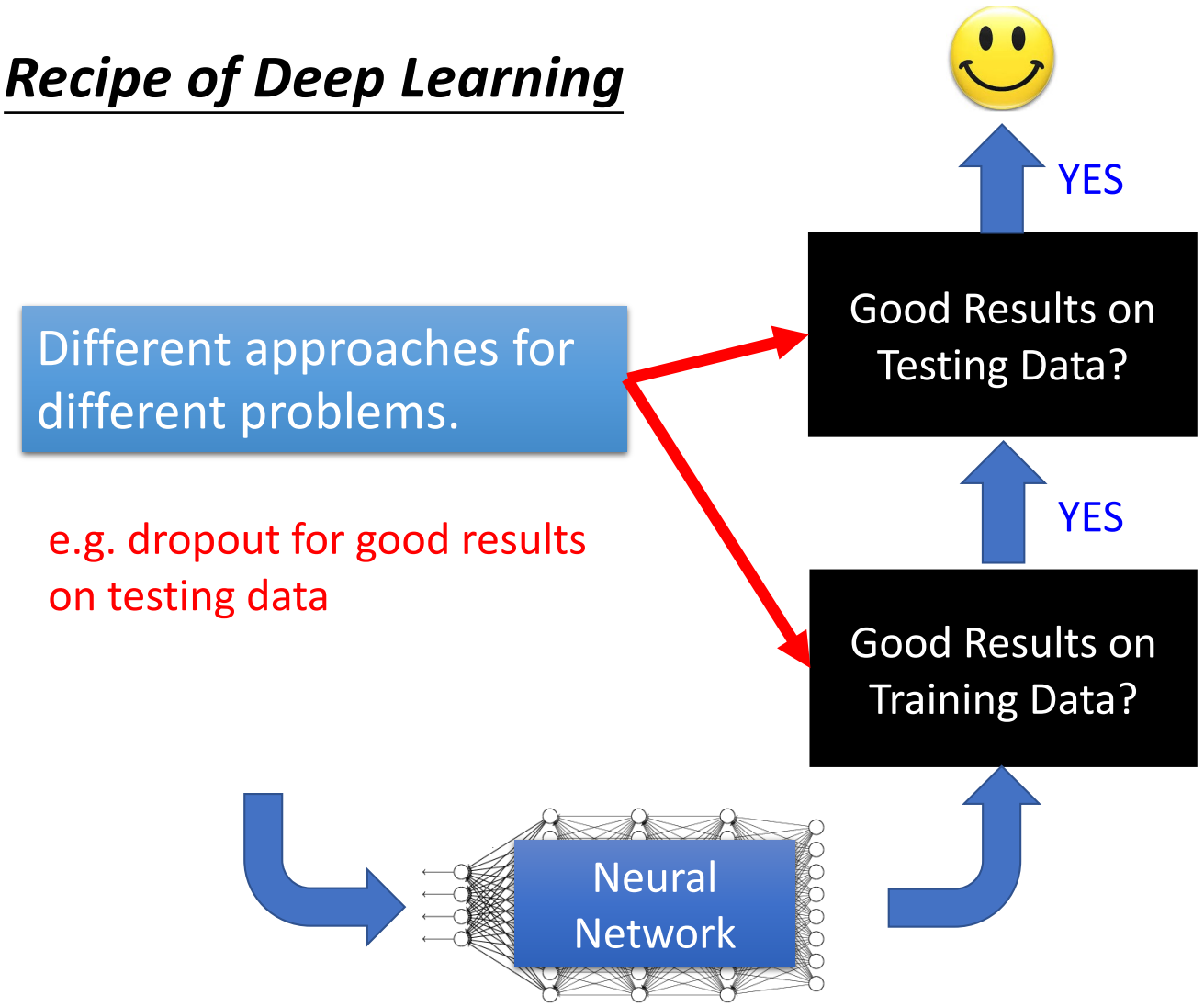


Training Data

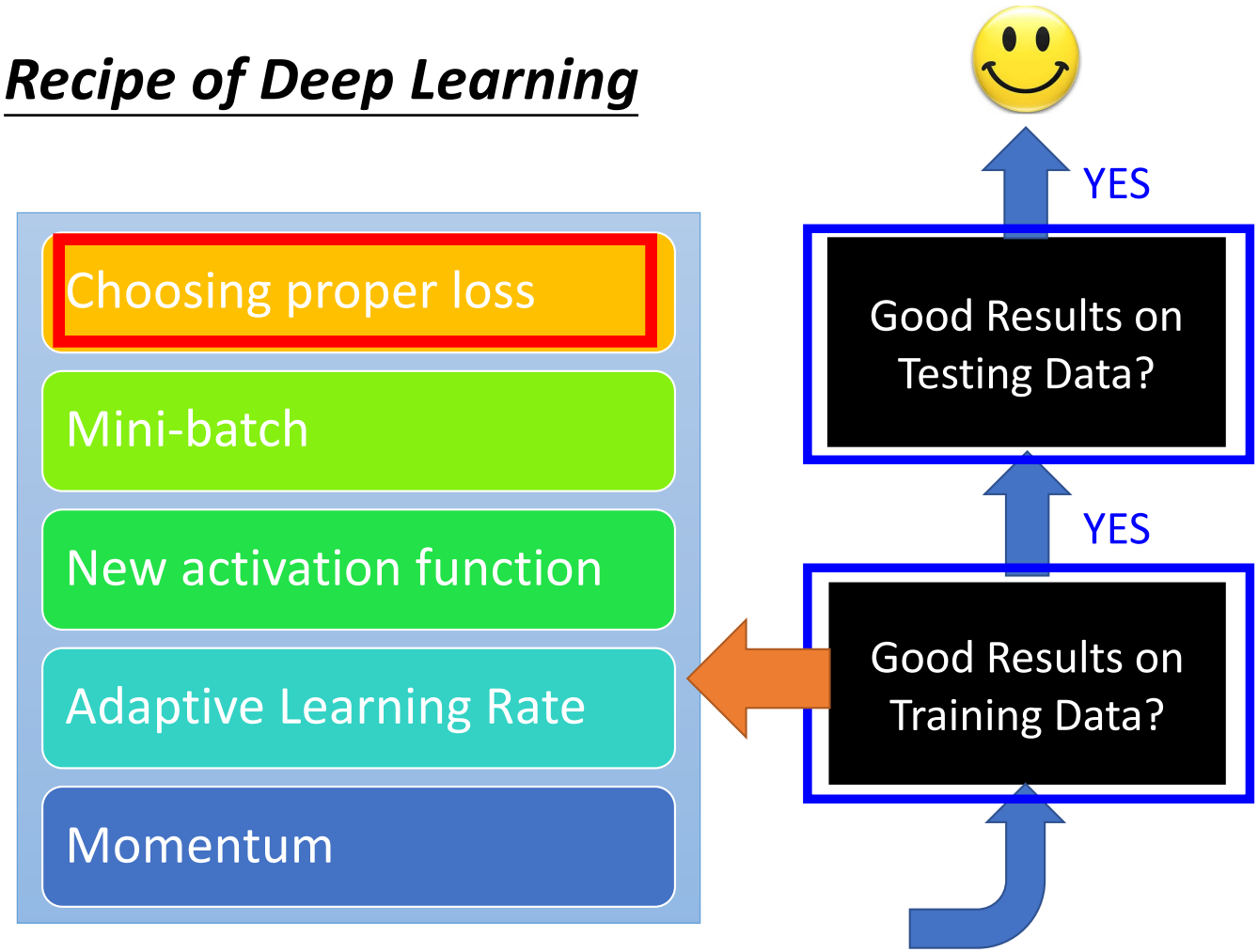
Testing Data

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

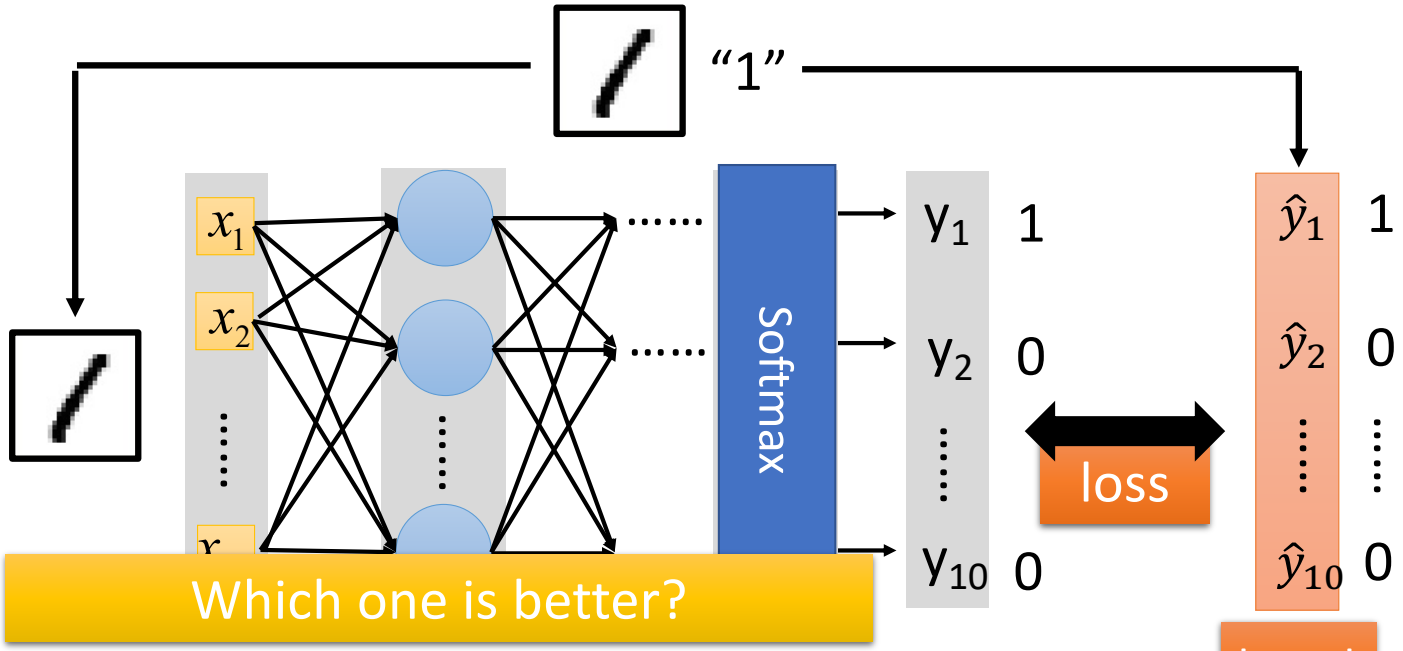
Recipe of Deep Learning



Recipe of Deep Learning



Choosing Proper Loss



Demo

Square Error

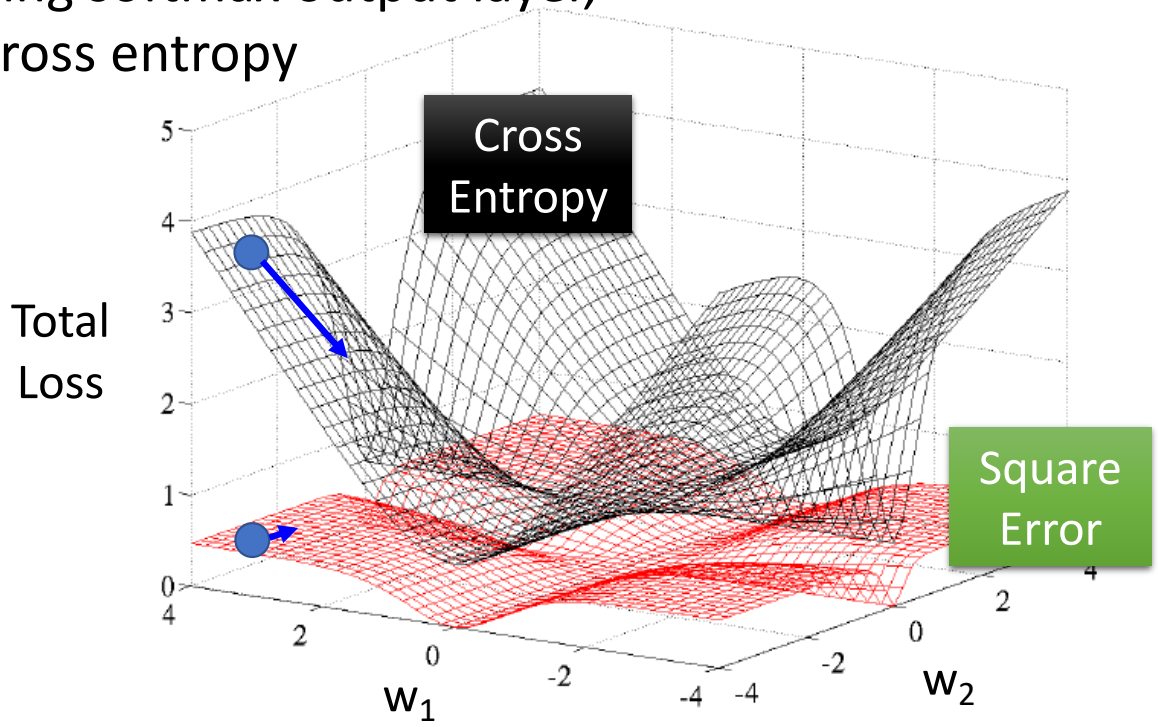
```
model.compile(loss='mse',
              optimizer=SGD(lr=0.1),
              metrics=['accuracy'])
```

Cross Entropy

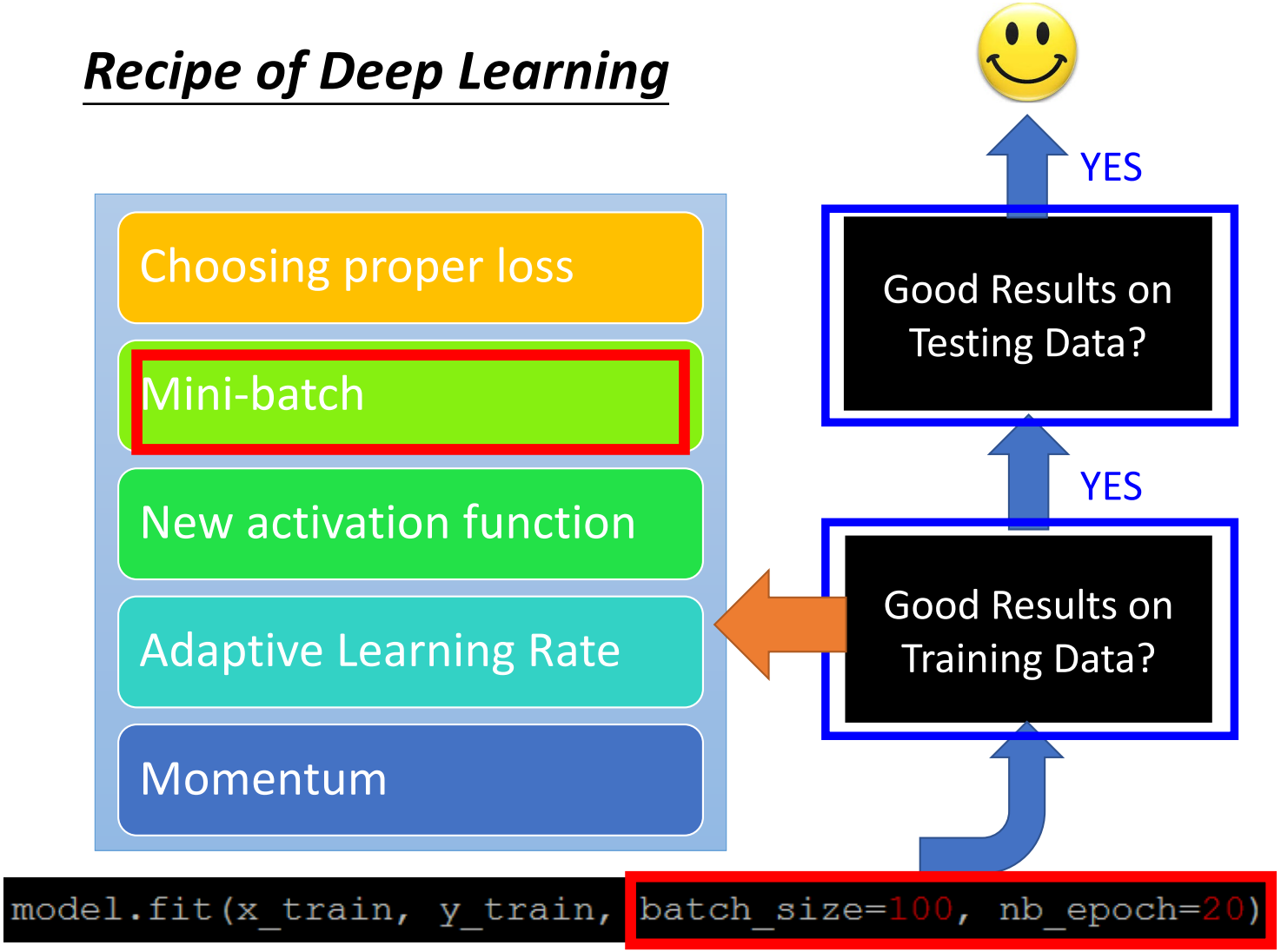
```
model.compile(loss='categorical_crossentropy',
              optimizer=SGD(lr=0.1),
              metrics=['accuracy'])
```

Choosing Proper Loss

When using softmax output layer, choose cross entropy

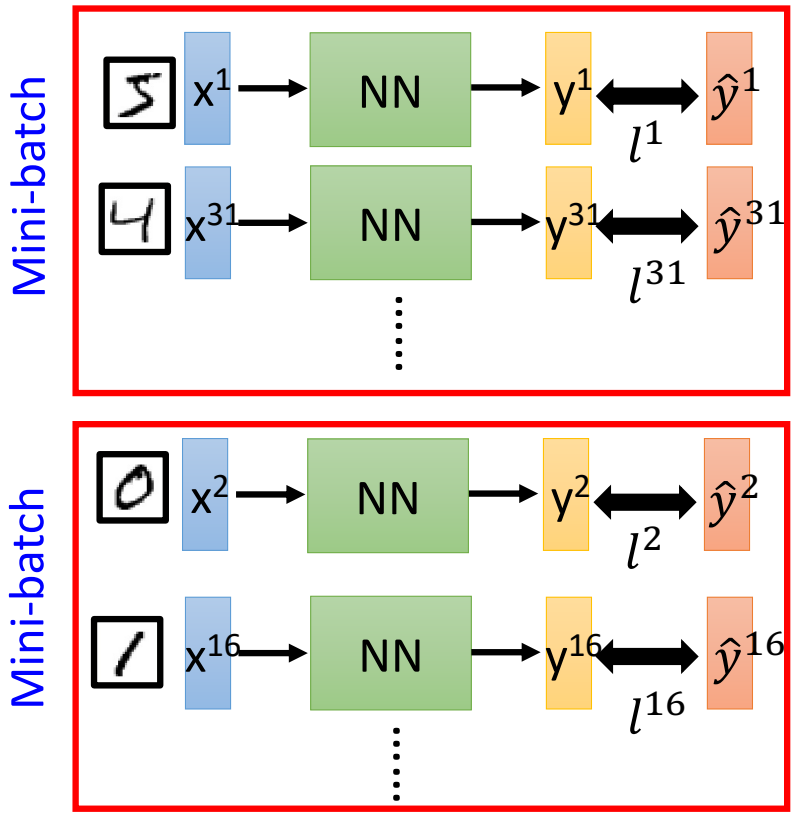


Recipe of Deep Learning



Mini-batch

We do not really minimize total loss!



- 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

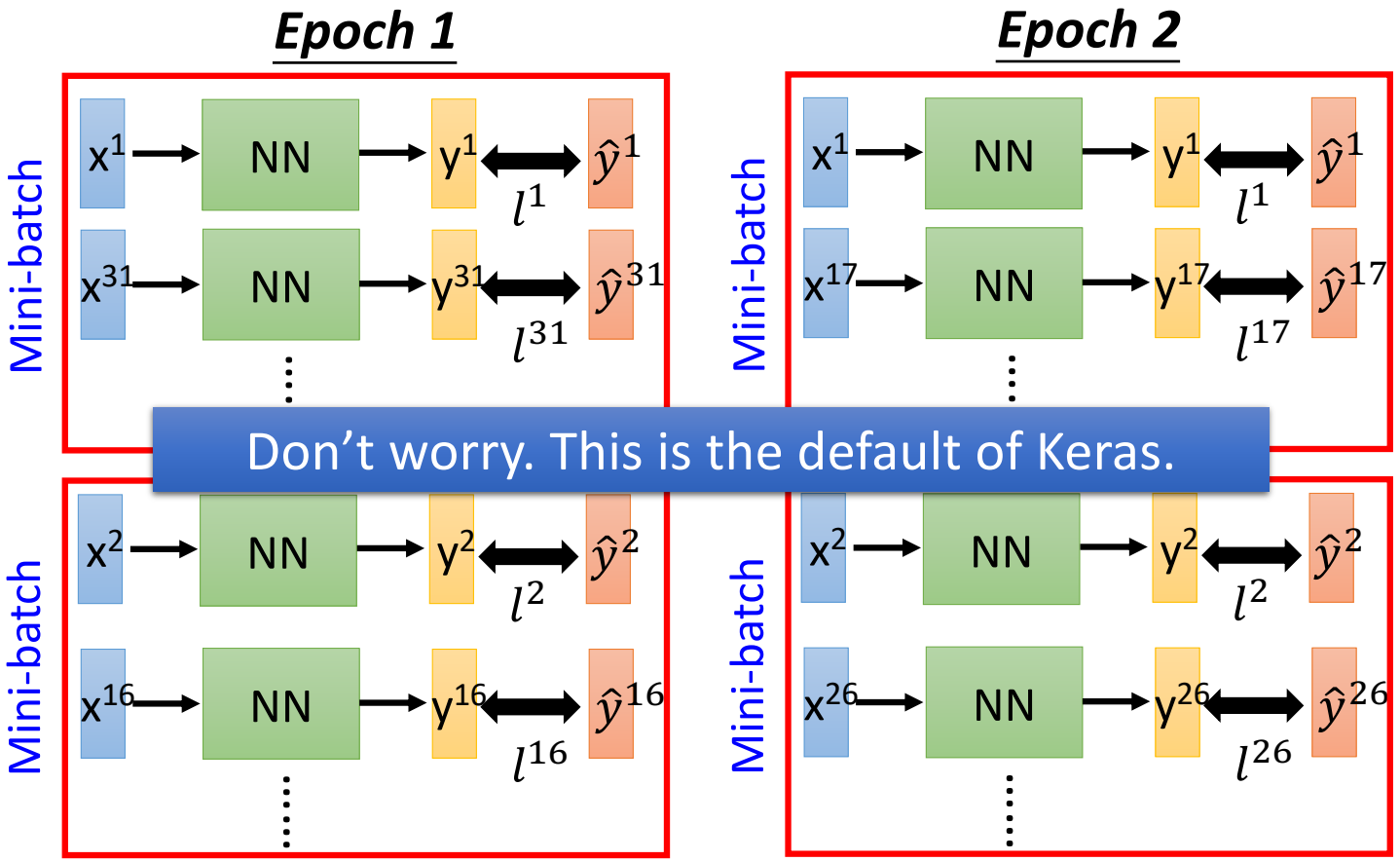
  ⋮

- Until all mini-batches have been picked

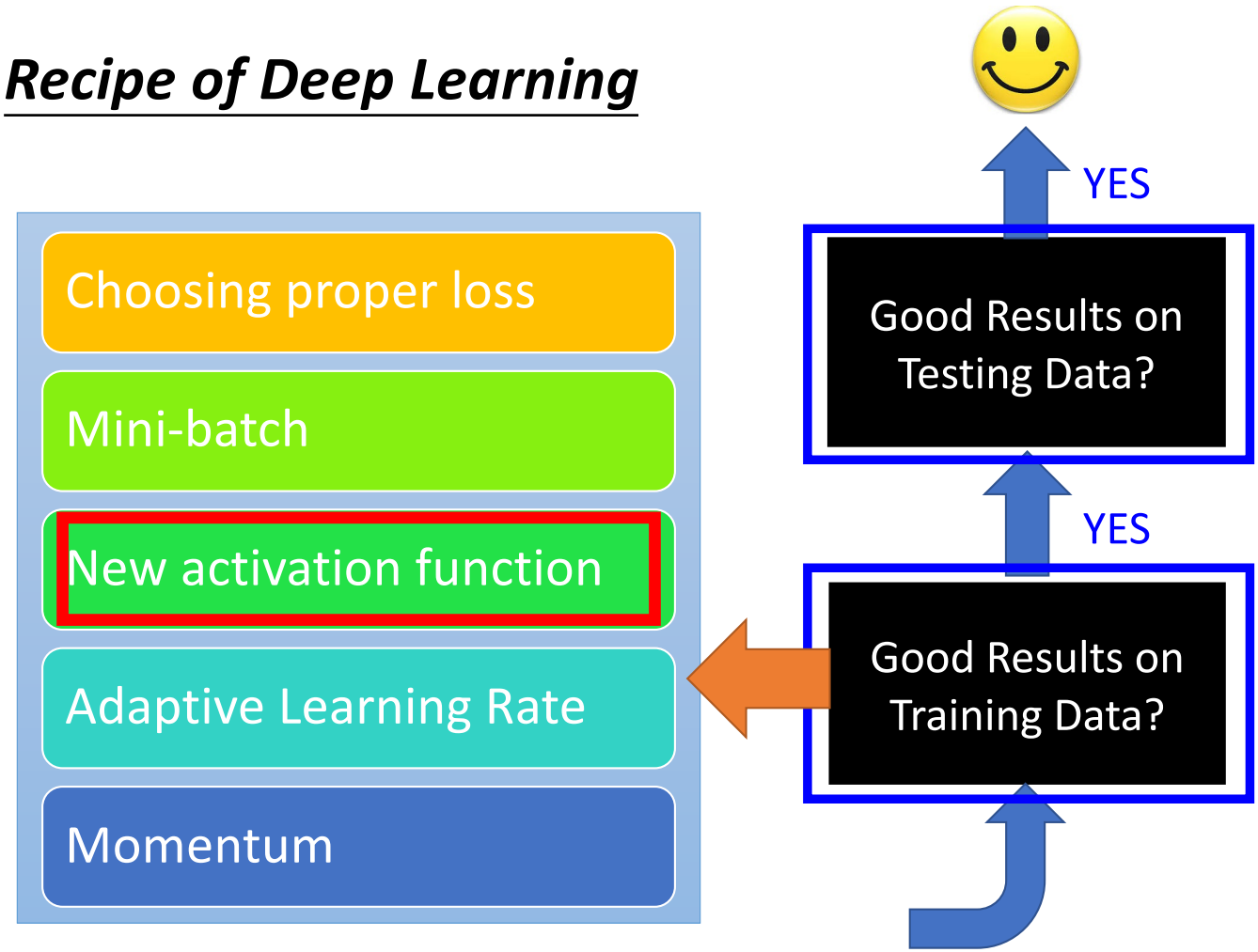
one epoch

Repeat the above process

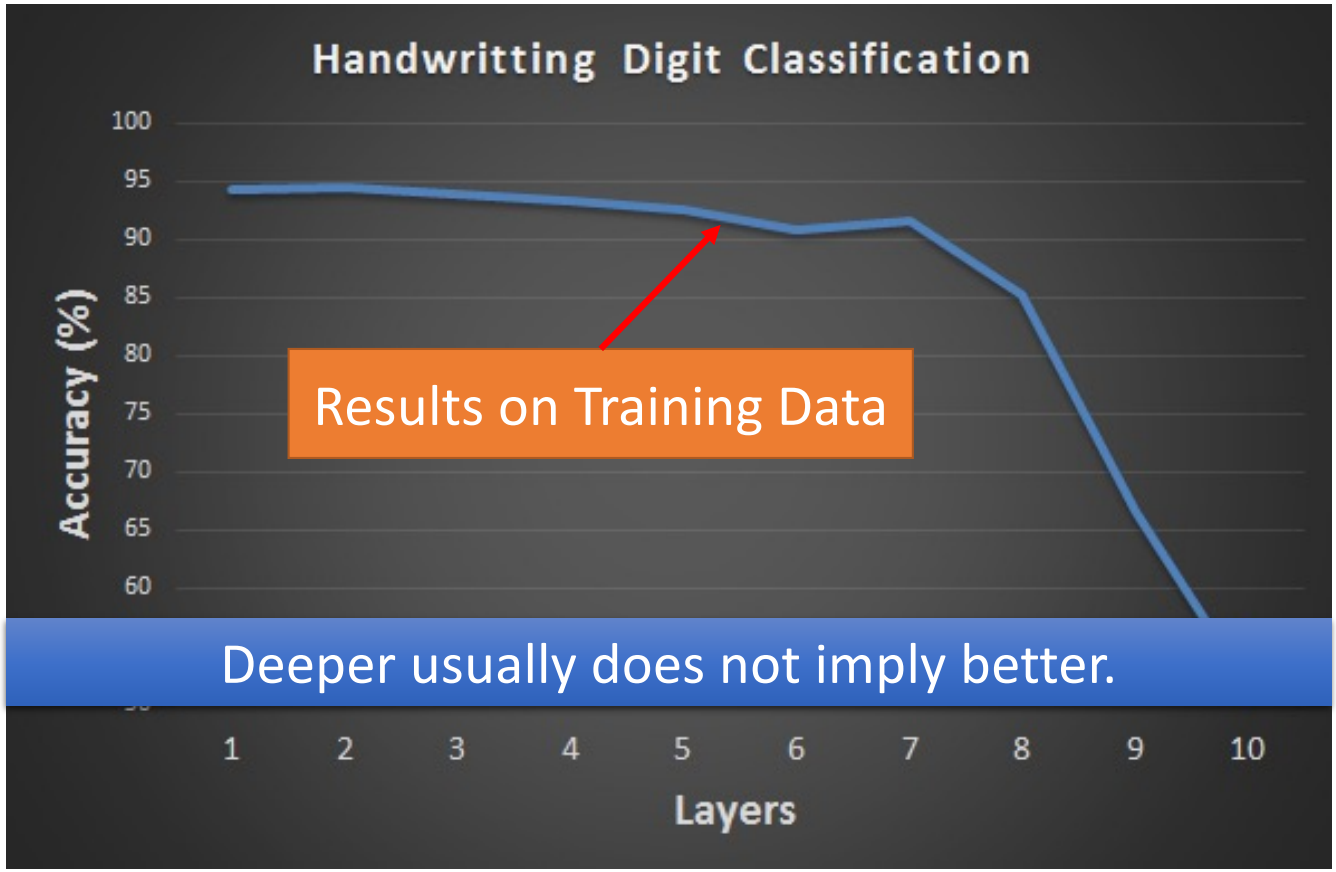
Shuffle the training examples for each epoch



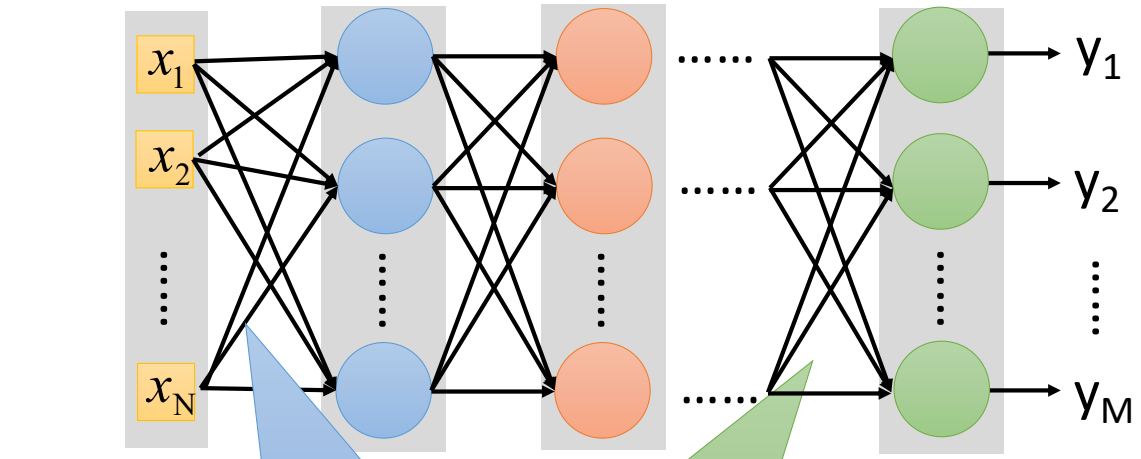
Recipe of Deep Learning



Hard to get the power of Deep …



Vanishing Gradient Problem



Smaller gradients

Learn very slow

Almost random

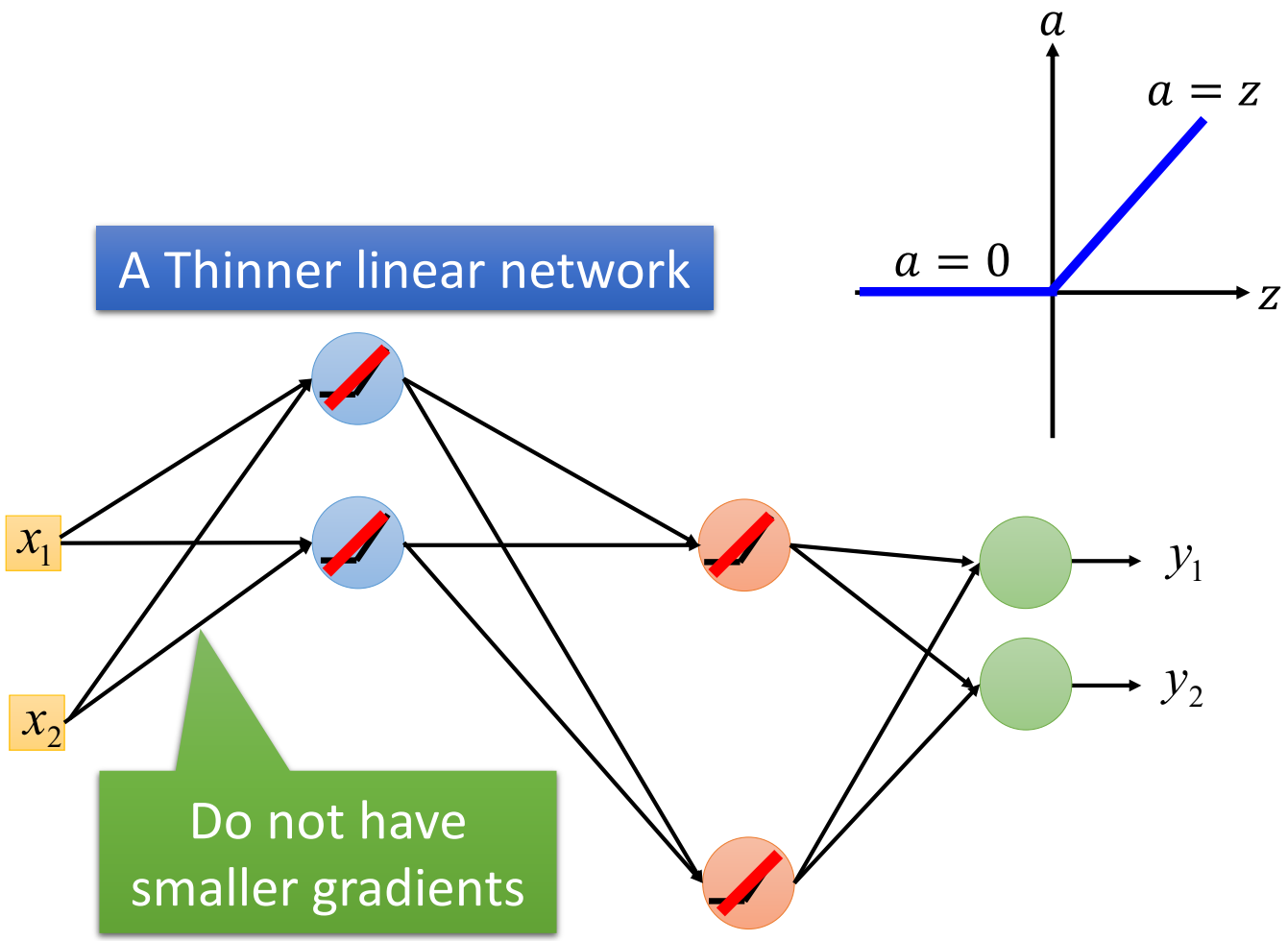
Larger gradients

Learn very fast

Already converge

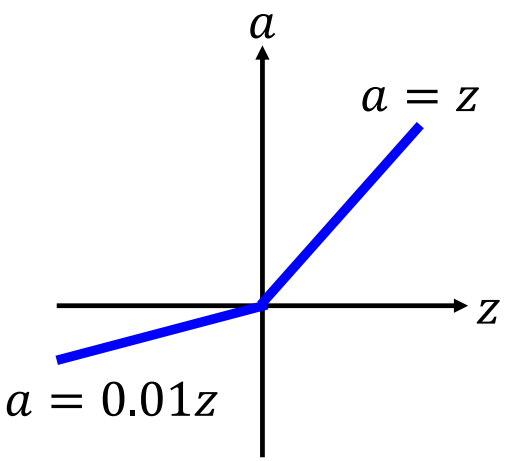
based on random!?

ReLU

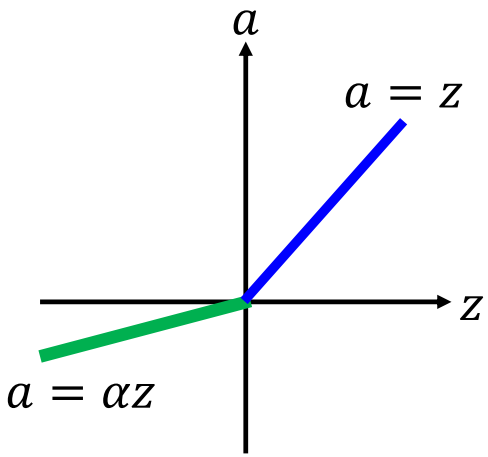

A Thinner linear network
a
$a = z$
$a = 0$
z
x_1
x_2
y_1
y_2
Do not have smaller gradients

ReLU - variant

Leaky ReLU



Parametric ReLU

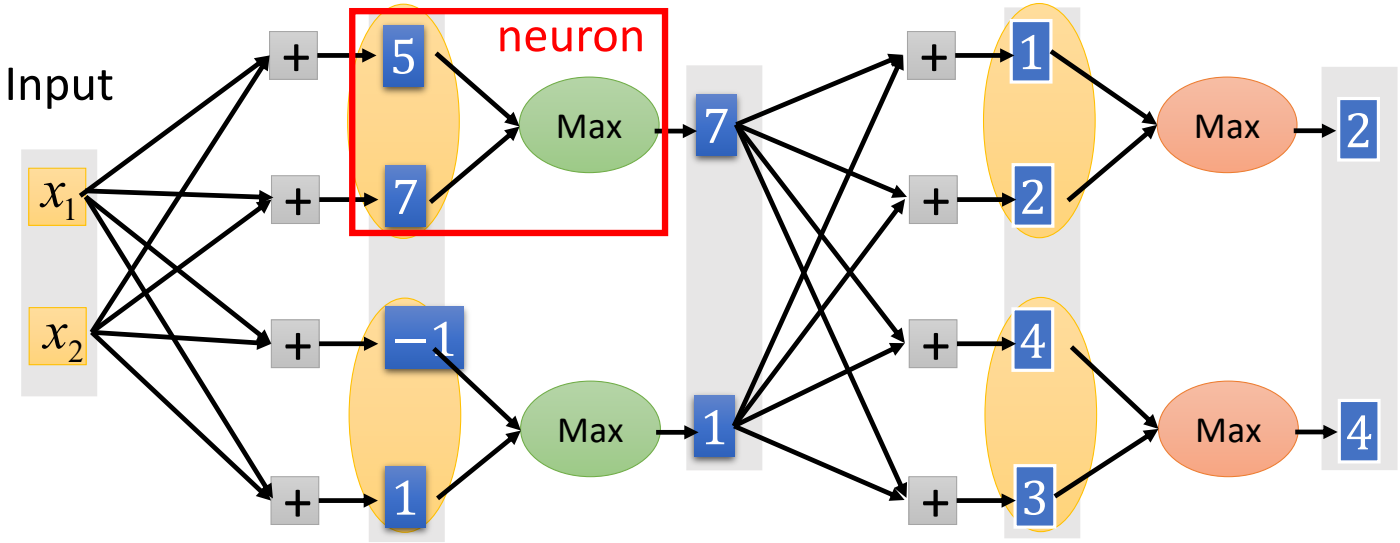


α also learned by gradient descent

Maxout

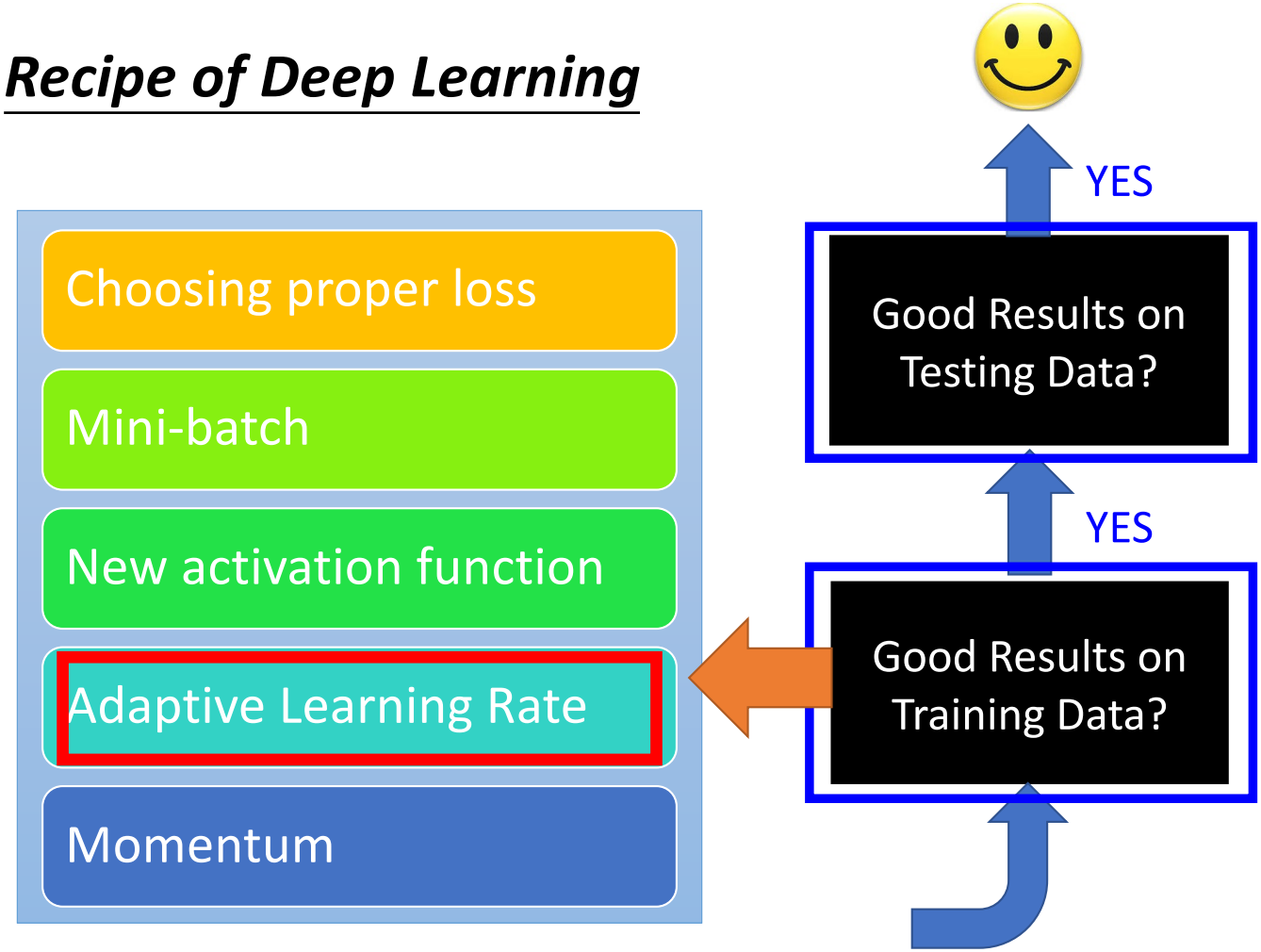
ReLU is a special cases of Maxout

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



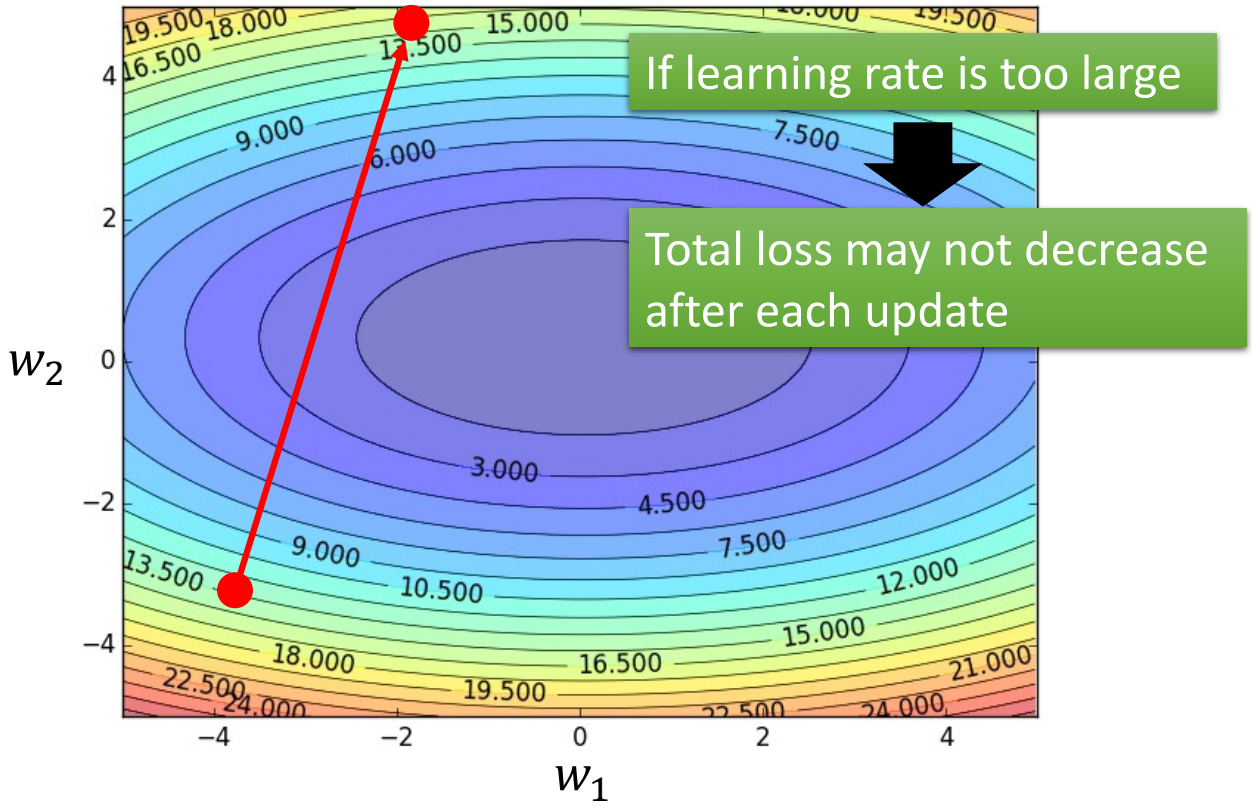
You can have more than 2 elements in a group.

Recipe of Deep Learning



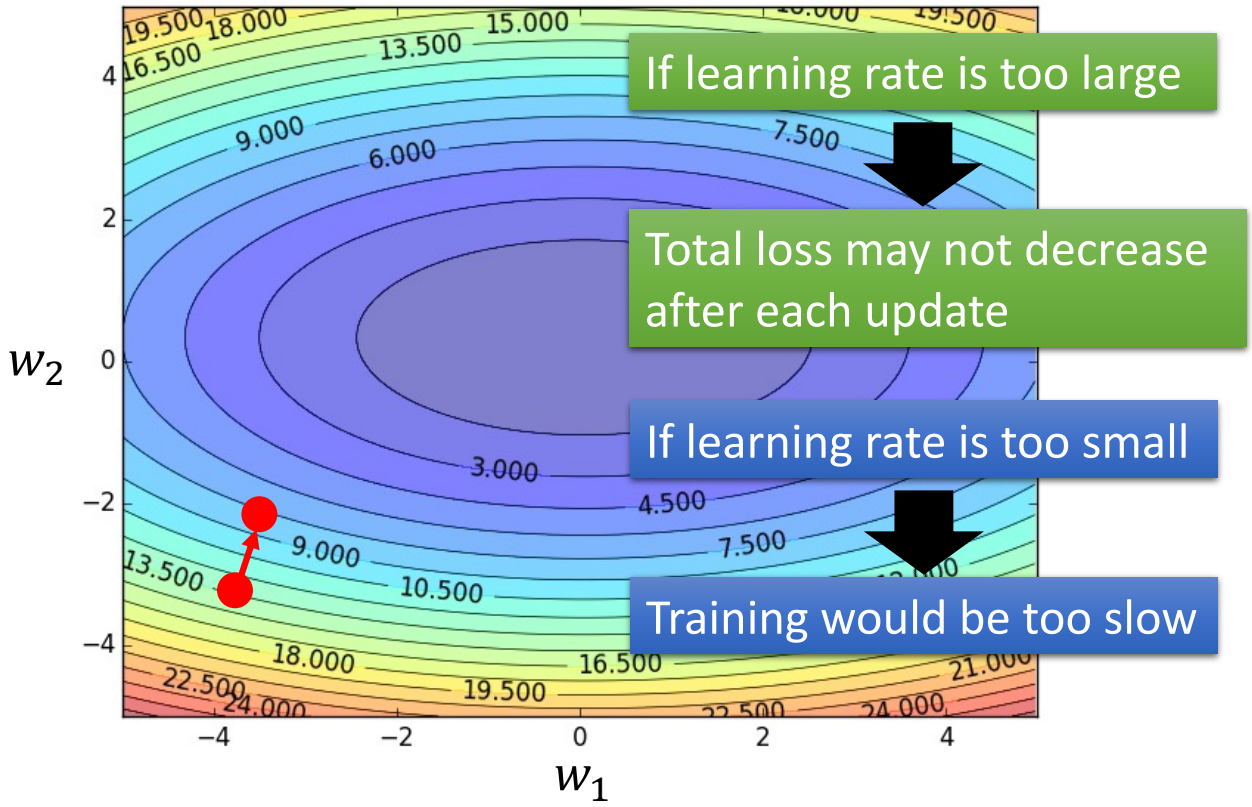
Learning Rates

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

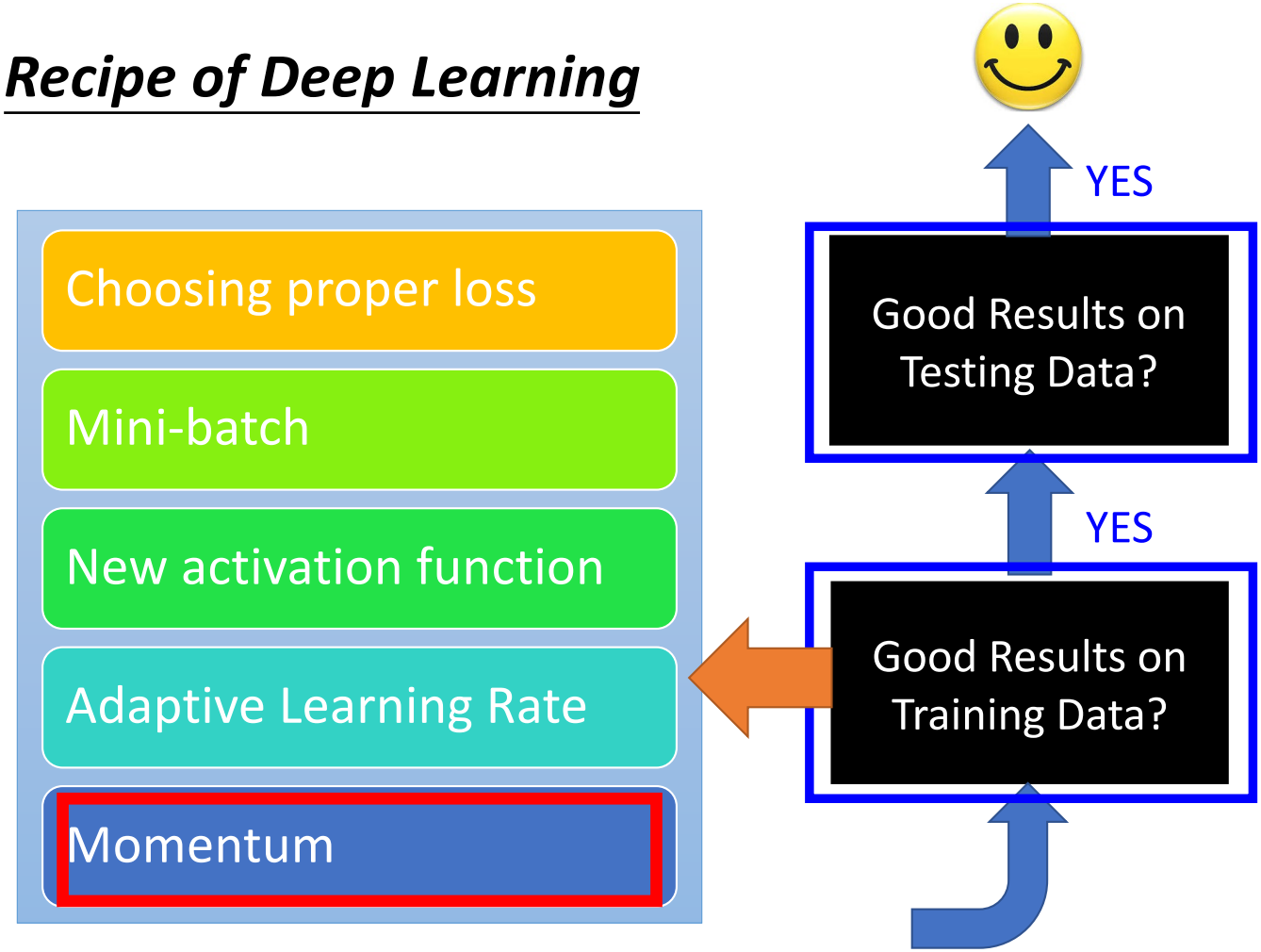
Learning Rates

- 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

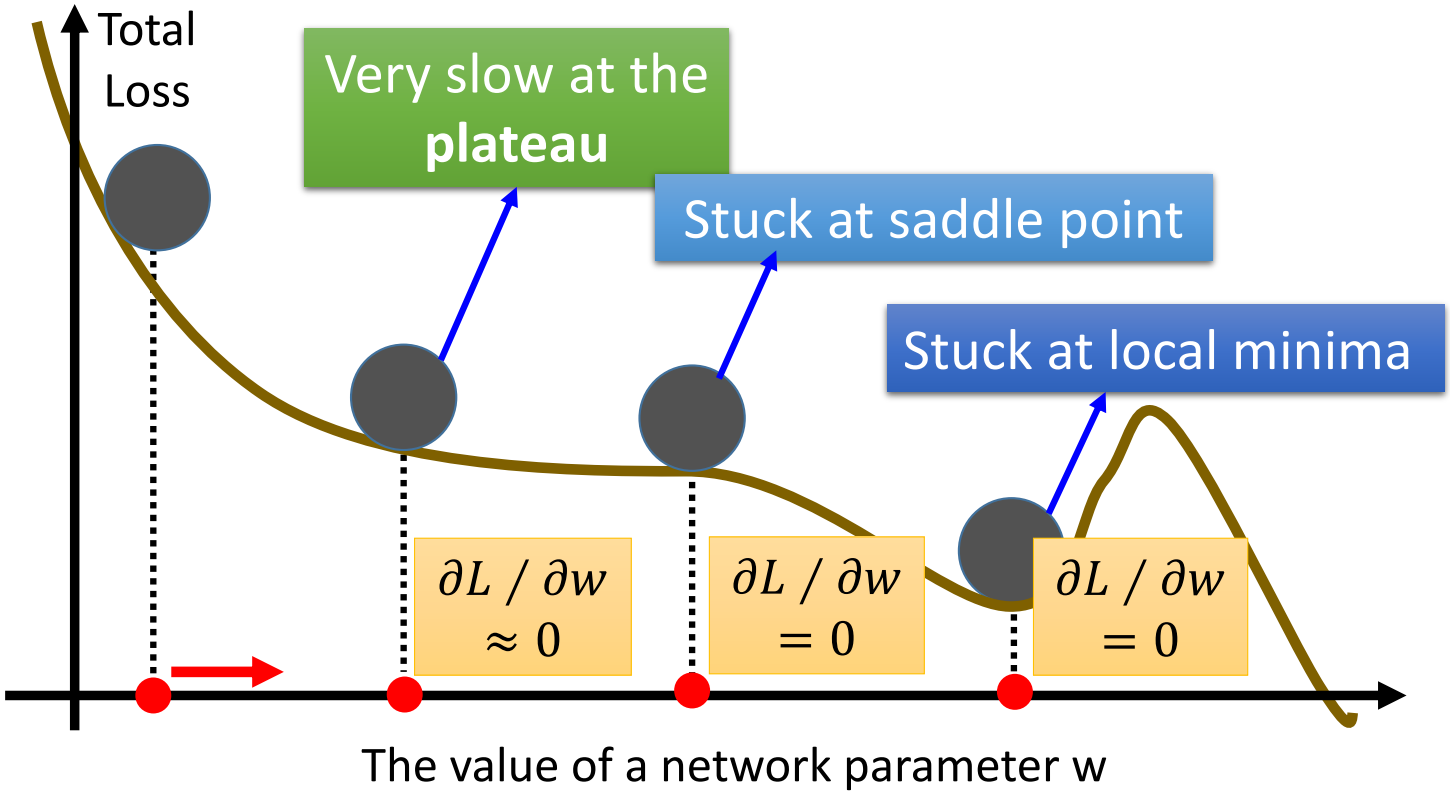
Not the whole story ……

- Adagrad [John Duchi, JMLR'11]
- RMSprop
- 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

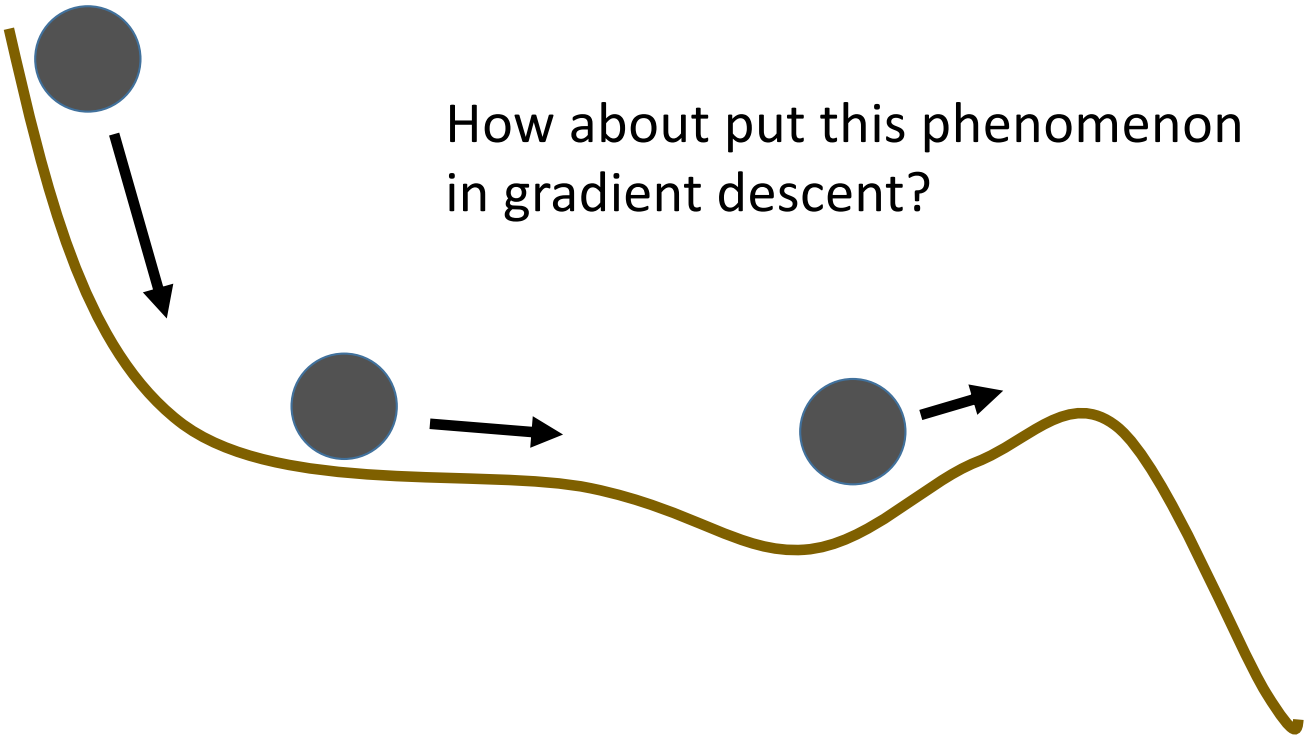


Hard to find optimal network parameters



In physical world ……

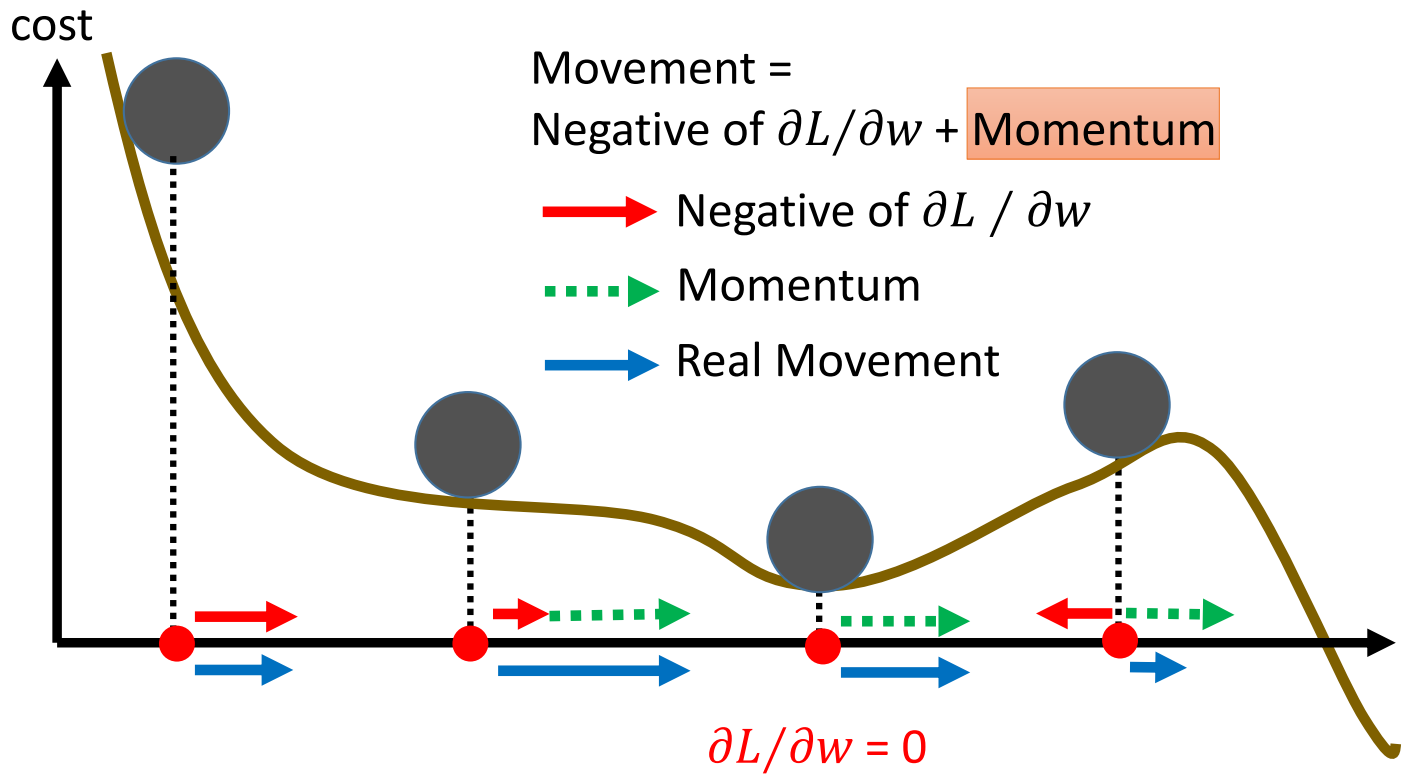
- Momentum



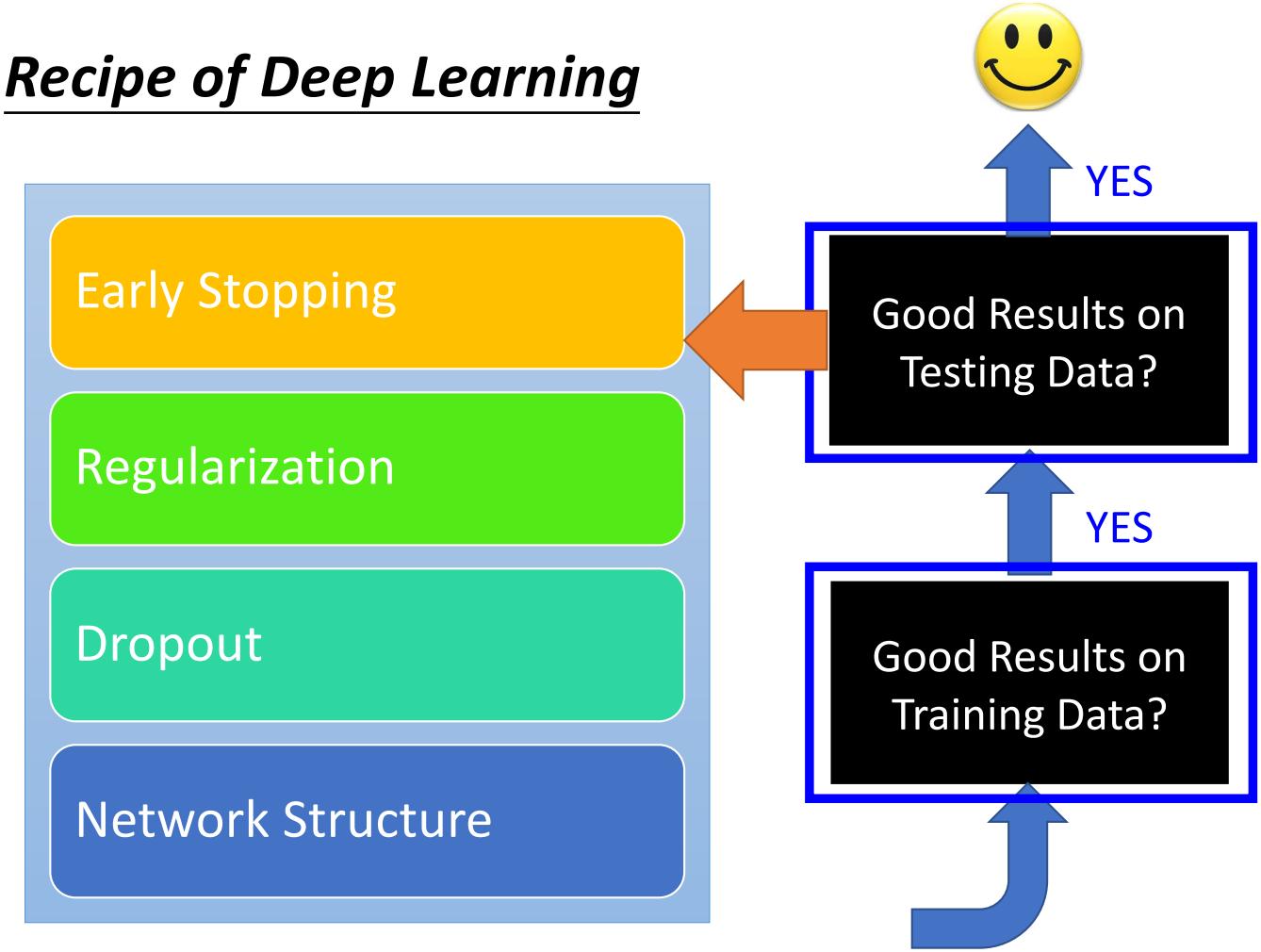
How about put this phenomenon in gradient descent?

Momentum

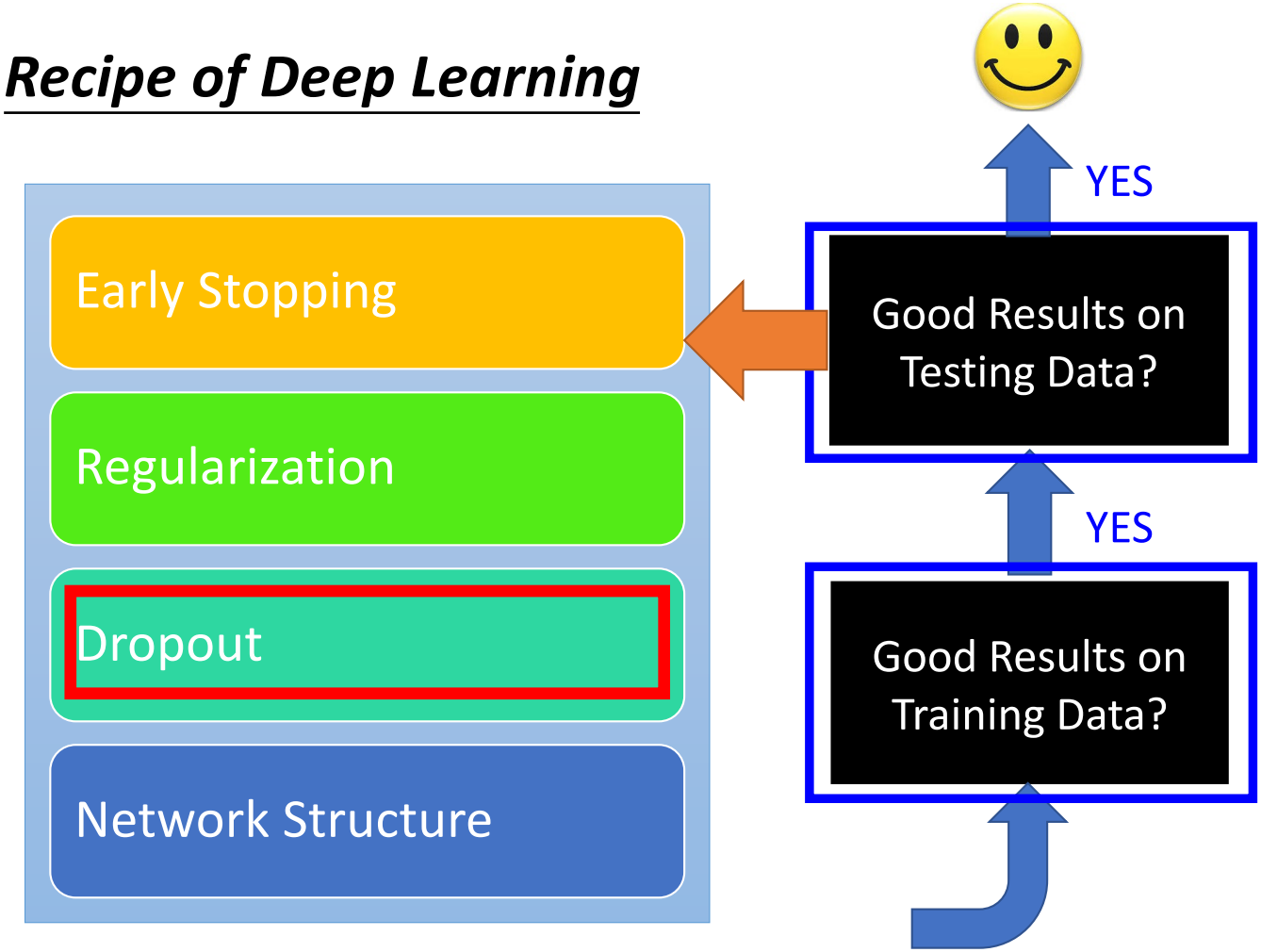
Still not guarantee reaching global minima, but give some hope ……



Recipe of Deep Learning

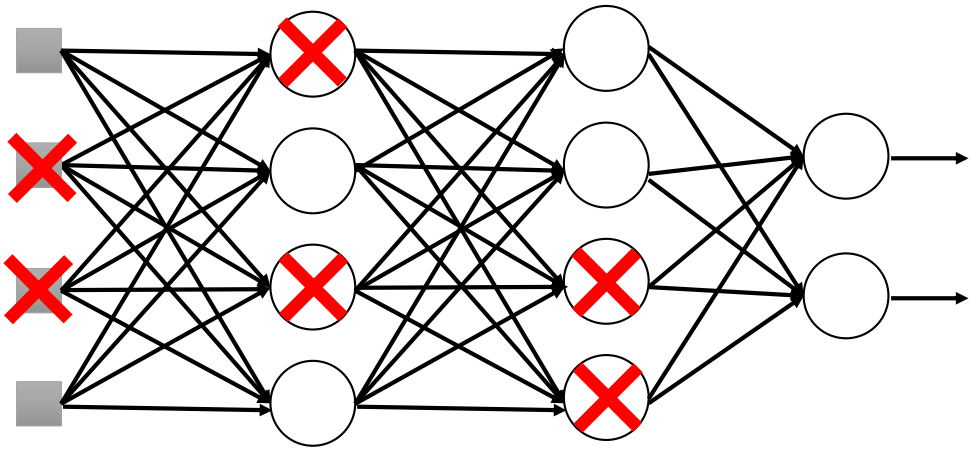


Recipe of Deep Learning

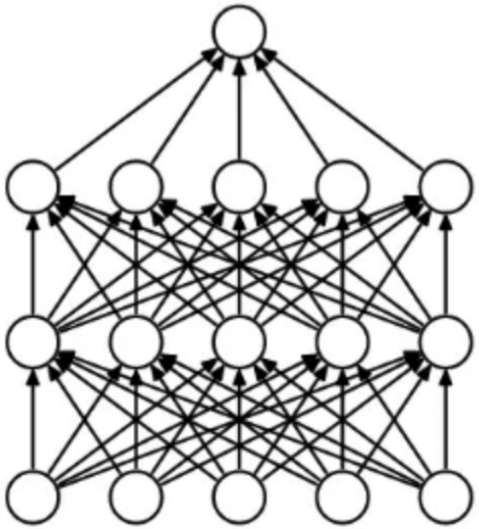


Dropout

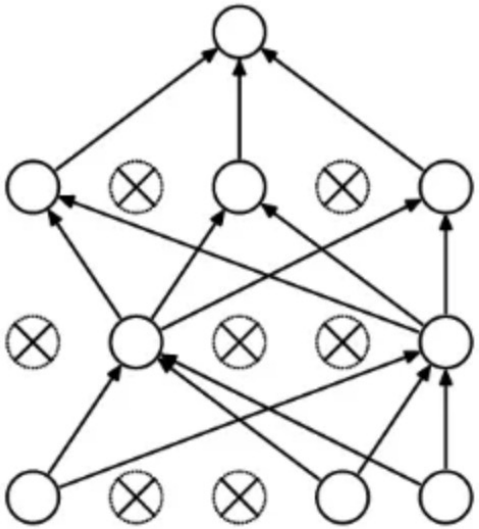
Training:



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



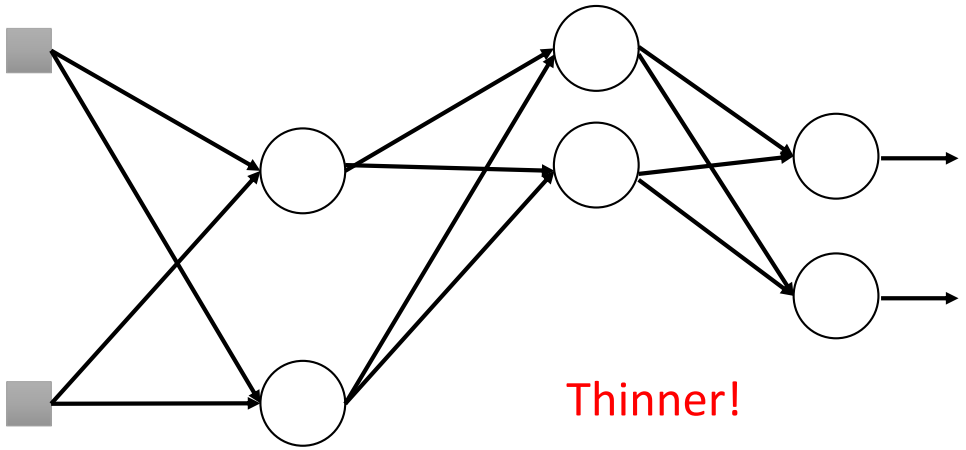
(a) Standard Neural Net



(b) After applying dropout.

Dropout

Training:

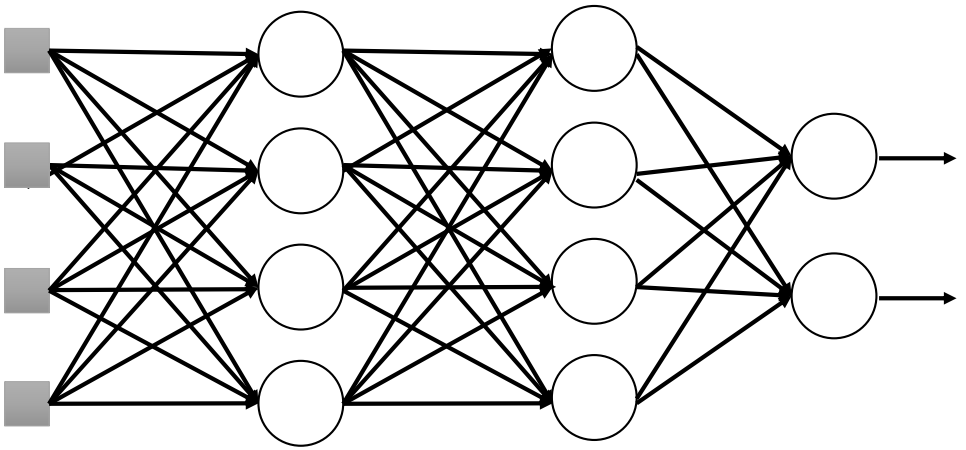


- **Each time before updating the parameters**
  - Each neuron has p% to dropout
    - **The structure of the network is changed.**
  - Using the new network for training

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

Testing

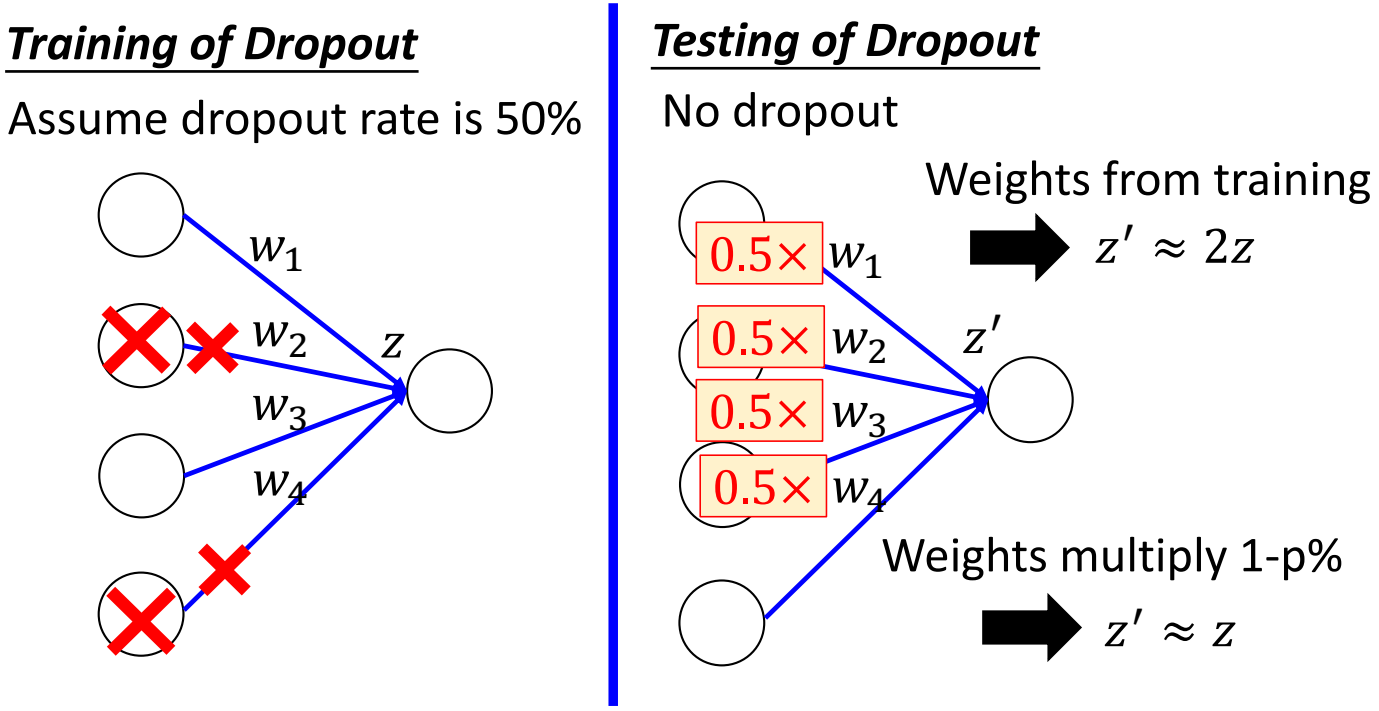
No dropout

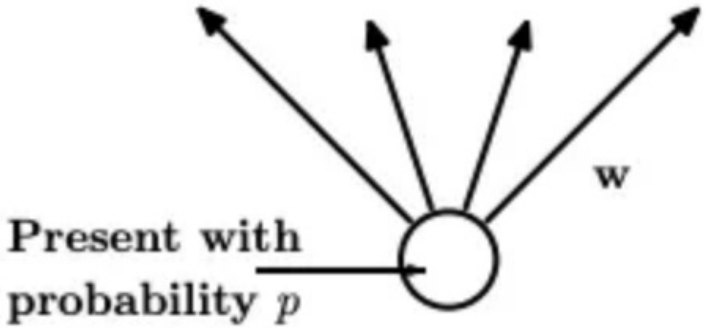
Training

Dropout

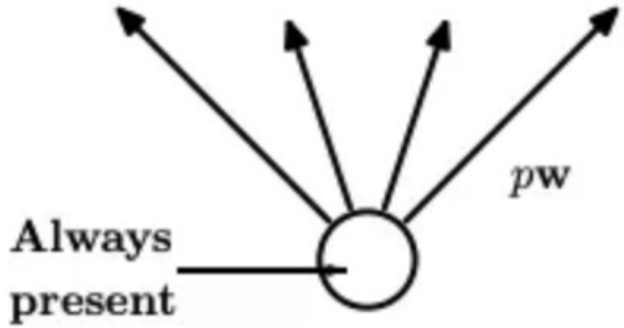
Dropout - Intuitive Reason

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



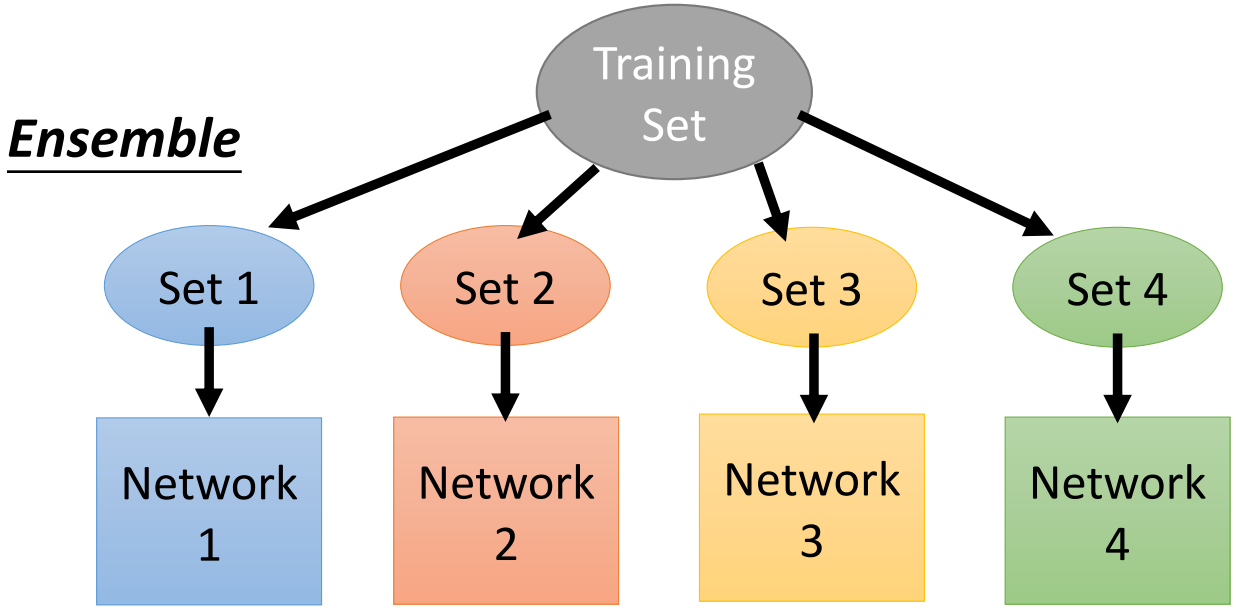


(a) At training time



(b) At test time

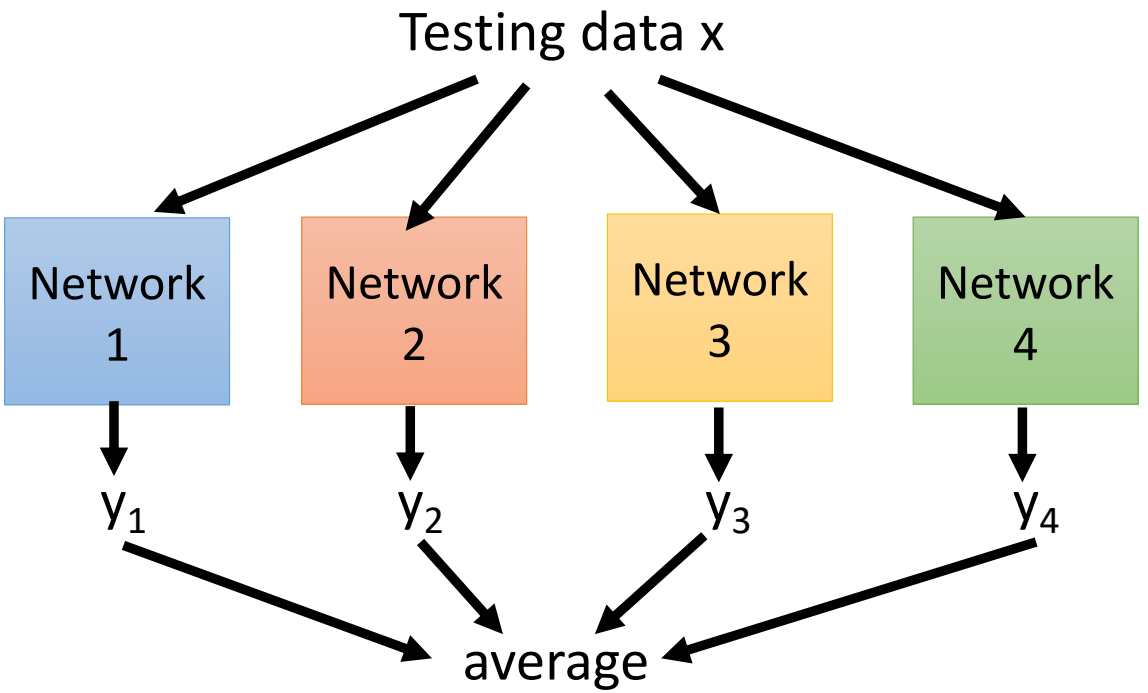
Dropout is a kind of ensemble.



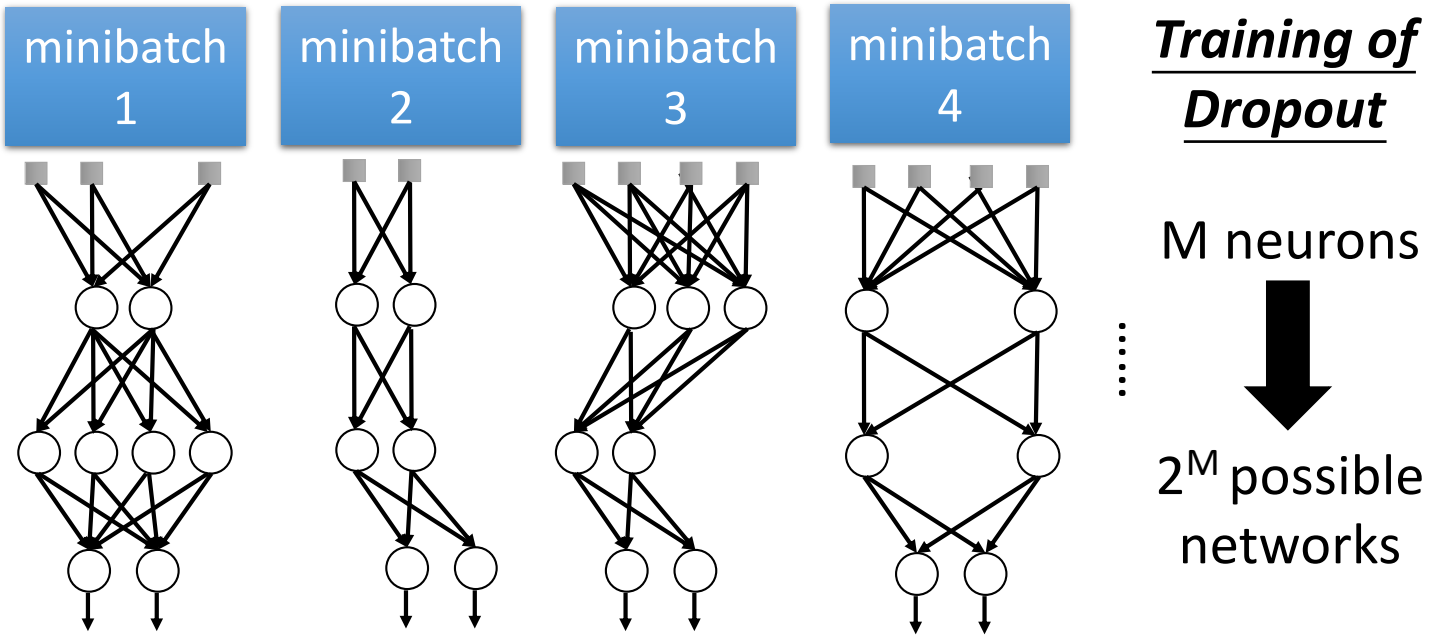
Train a bunch of networks with different structures

Dropout is a kind of ensemble.

Ensemble

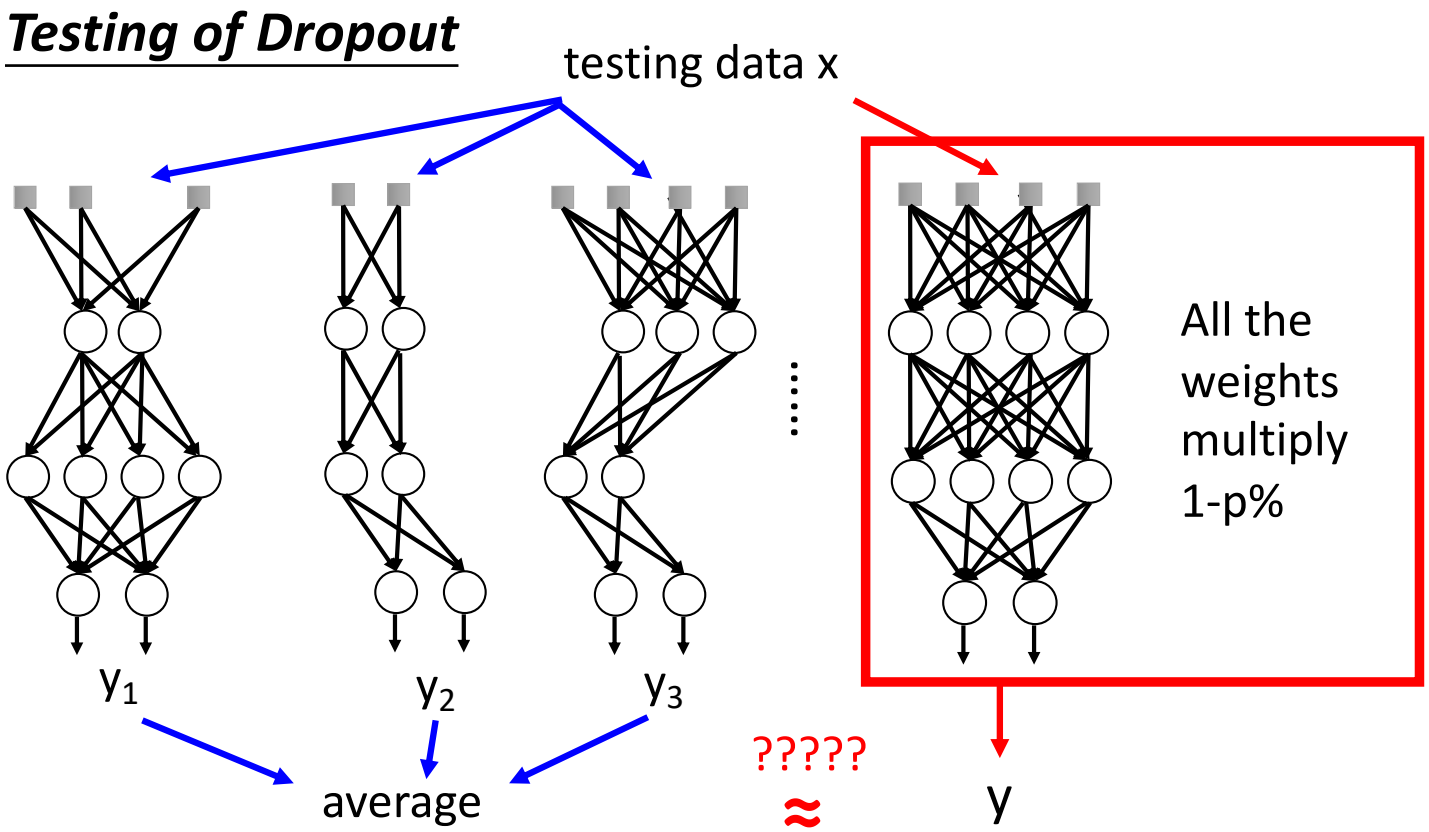


Dropout is a kind of ensemble.



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

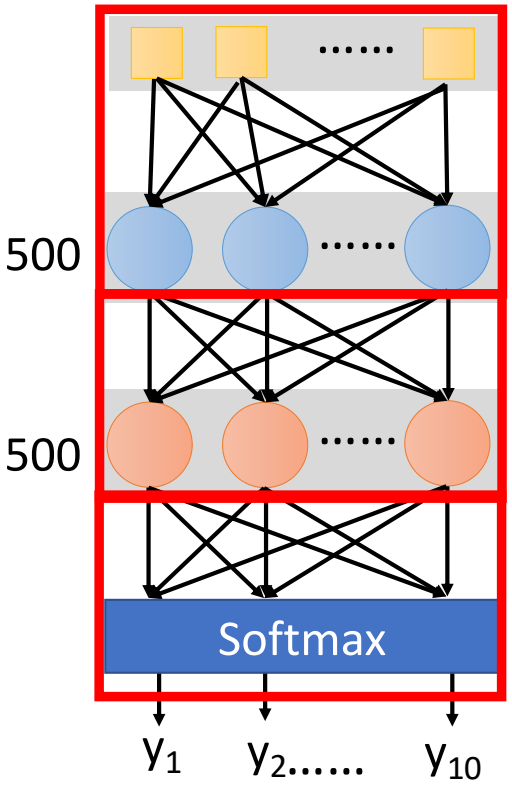
Dropout is a kind of ensemble.



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

Demo




```
model = Sequential()
```

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

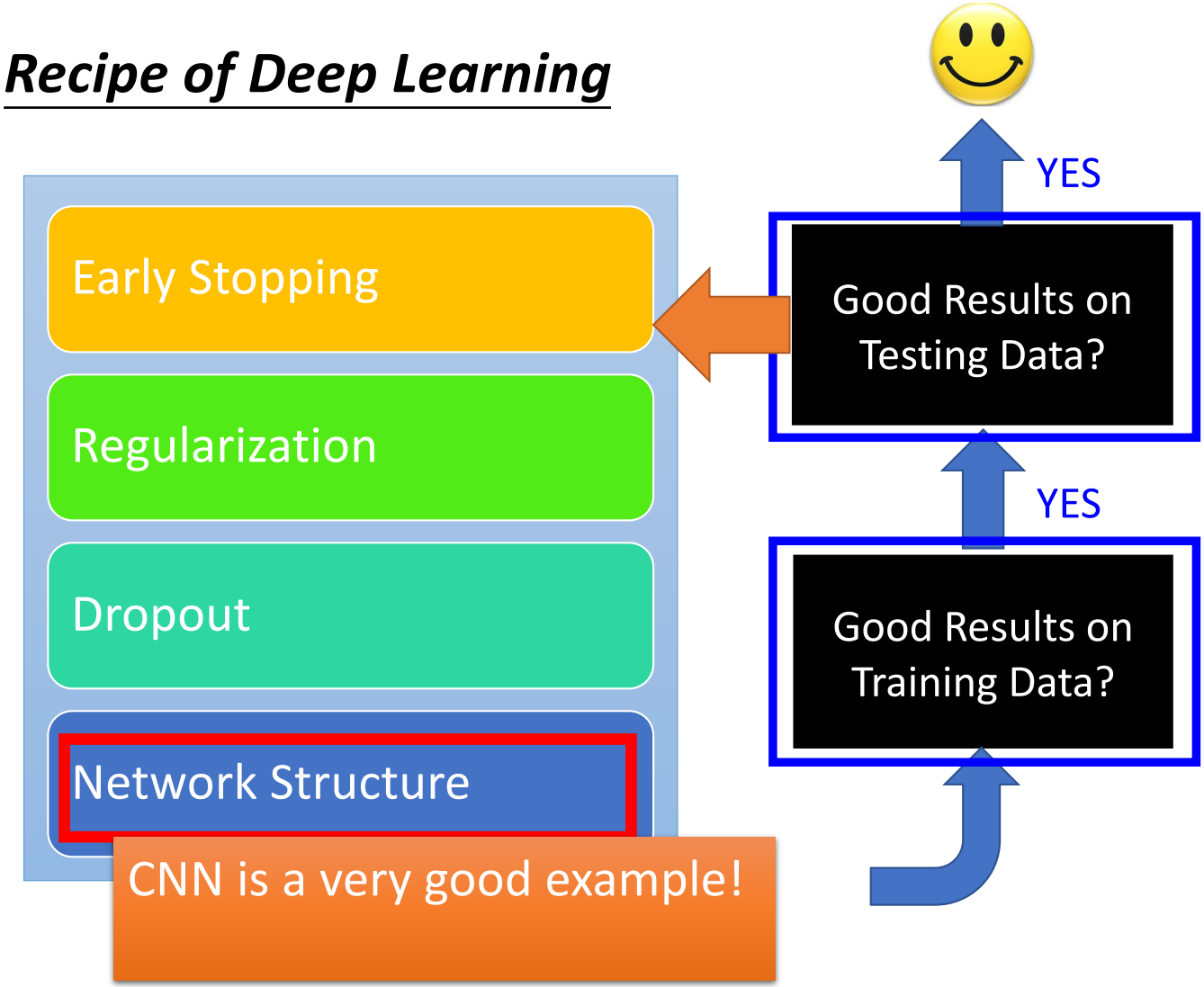
model.add(dropout(0.8))

```
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') )
```

Recipe of Deep Learning



Concluding Remarks

Recipe of Deep Learning