DEEP NEURAL NETWORKS

Narges Norouzi

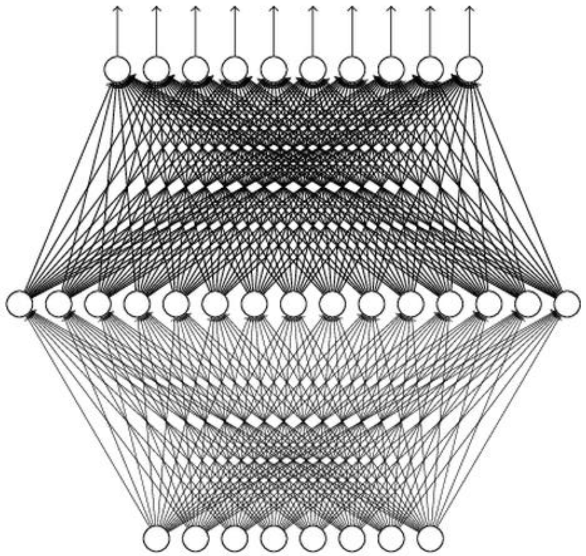
WHY DEEP? UNIVERSALITY THEOREM

Any continuous function f

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

Can be realized by a network with one hidden layer

(given **enough** hidden neurons)



Very cool visual proof:
http://neuralnetworksanddeeplearning.com/chap4.html

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

Why Deep? Analogy

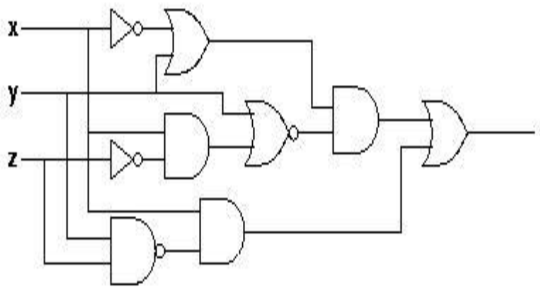
Logic circuits

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.**

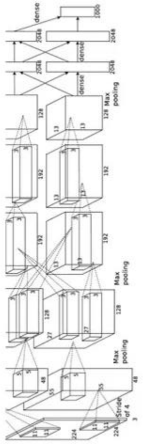
Using multiple layers of neurons to represent some functions are much simpler

→ less parameters → less data?

Deep = Many hidden layers

8 layers

16.4%



AlexNet (2012)

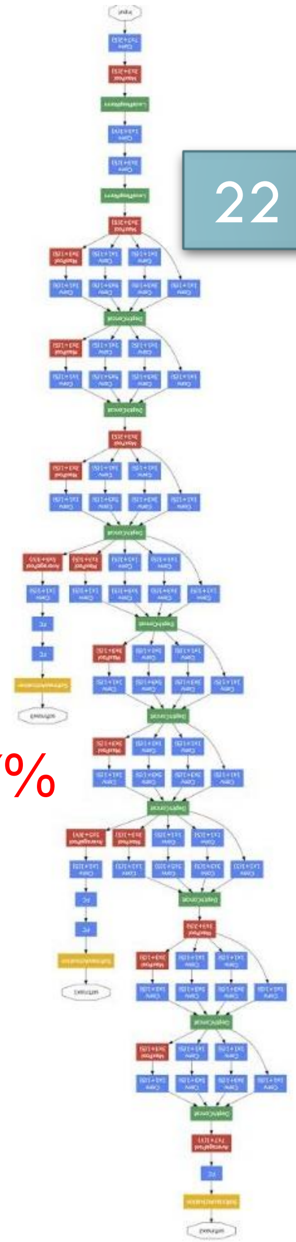
7.3%

19 layers

VGG (2014)

22 layers

6.7%

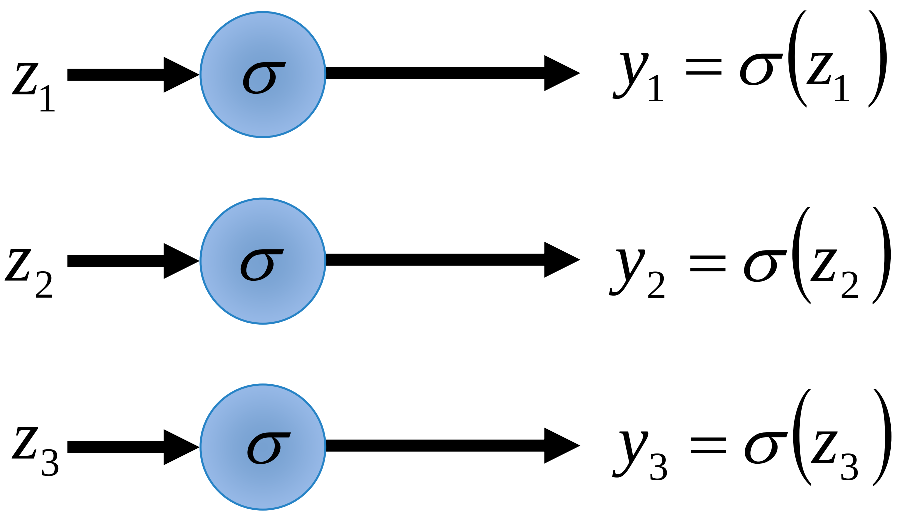


GoogleNet (2014)

OUTPUT LAYER

- Softmax layer as the output layer

<u>Ordinary Layer</u>



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

May not be easy to interpret

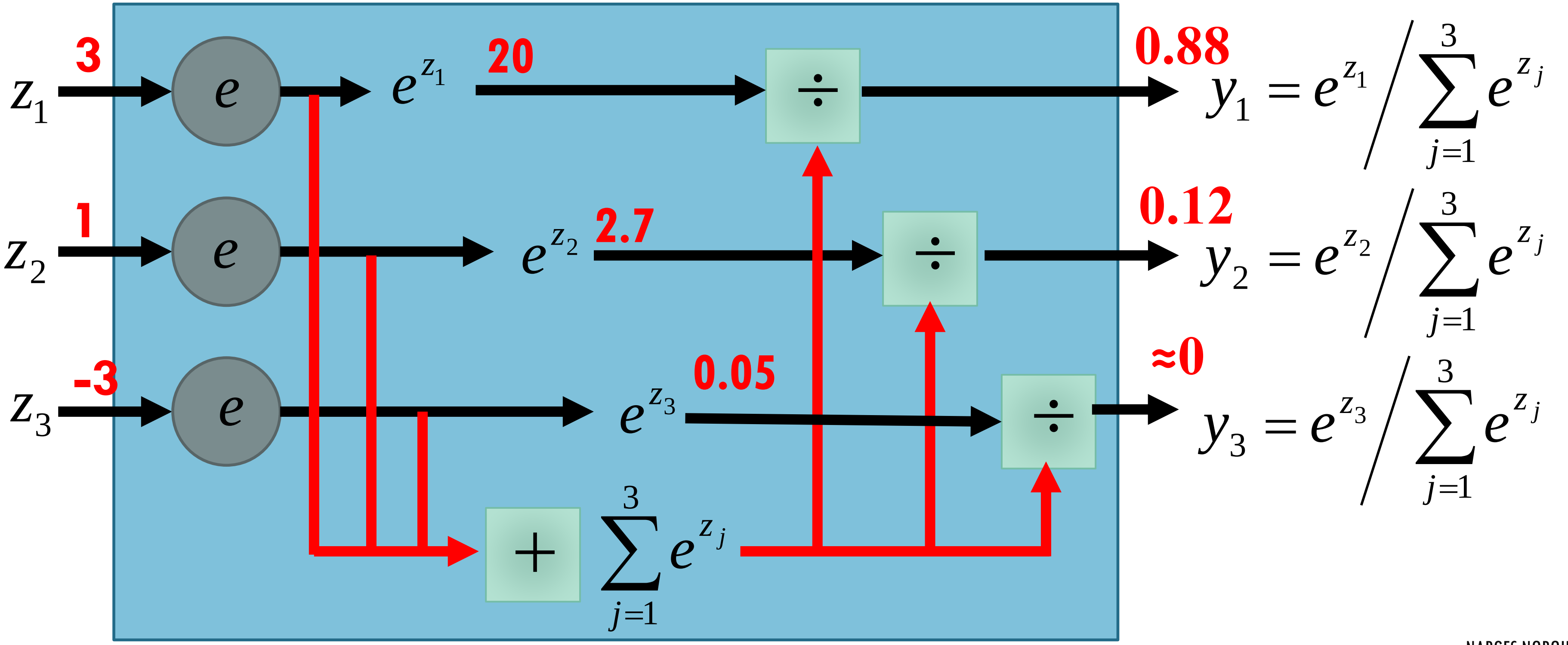
OUTPUT LAYER

Probability:

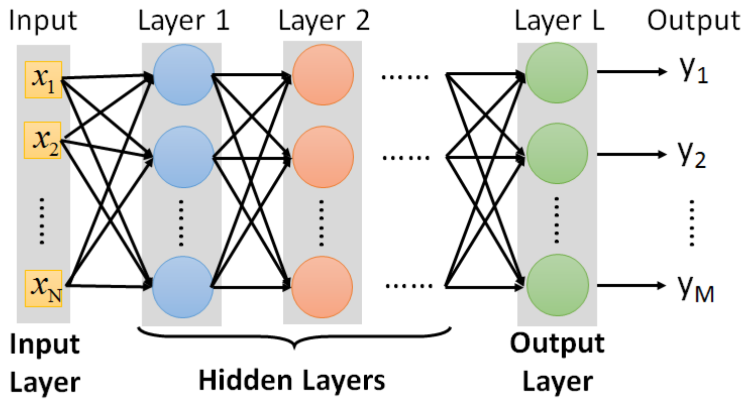
- $1 > y_i > 0$
- $\sum_i y_i = 1$

- Softmax layer as the output layer

Softmax Layer



FAQ



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

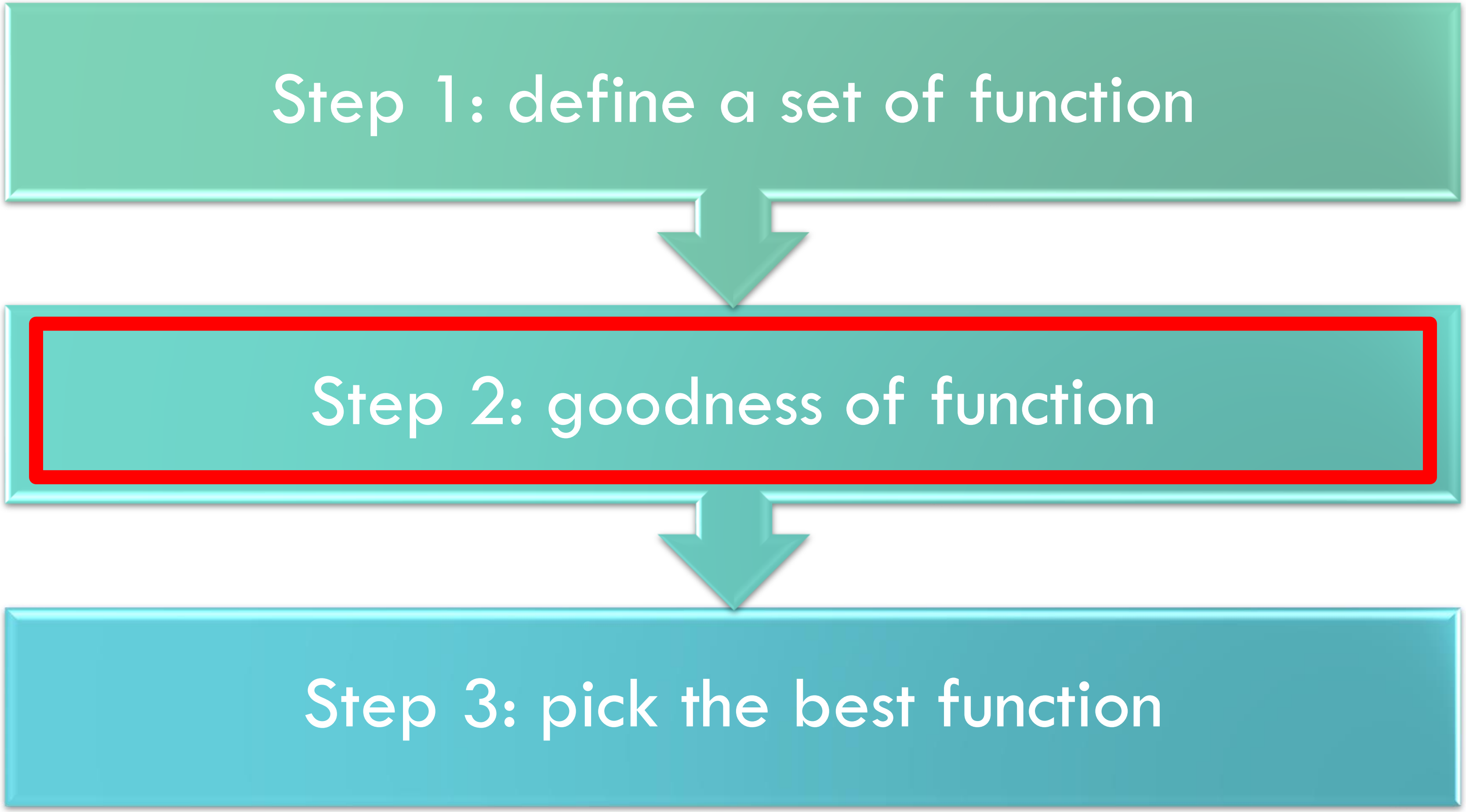
Trial and Error + Intuition

- Q: Can we design the network structure?

Convolutional Neural Network (CNN)

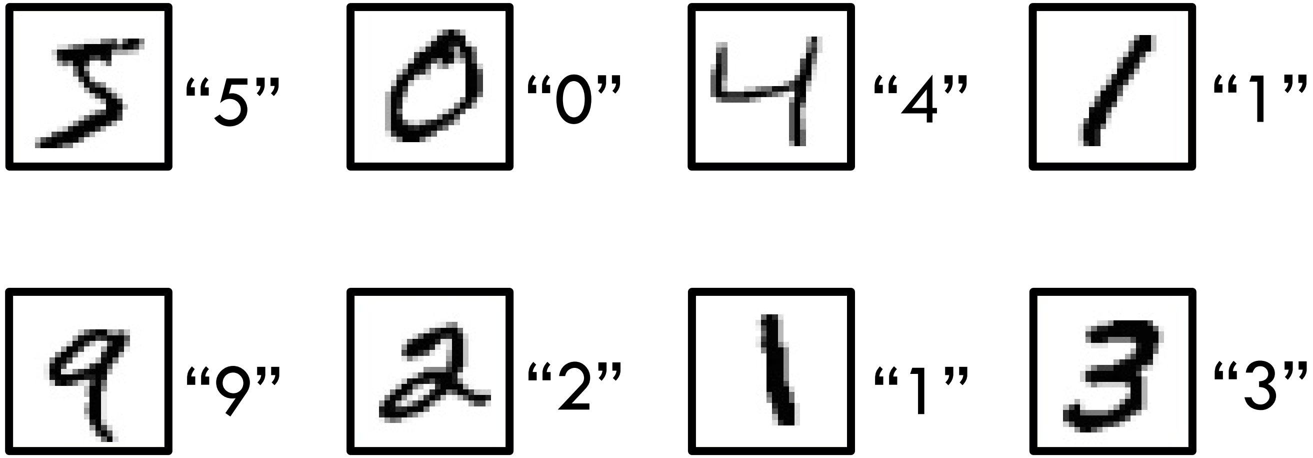
- Q: Can the structure be automatically determined?
  - Yes, but not widely studied yet.

THREE STEPS FOR DEEP LEARNING



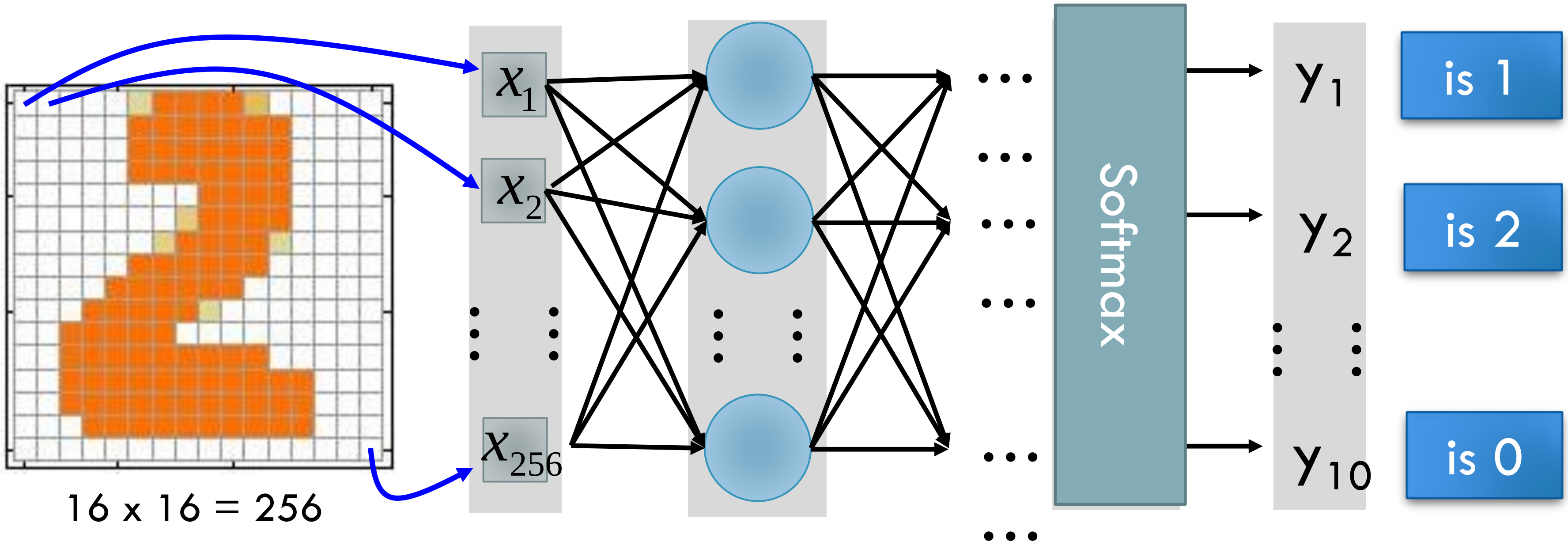
TRAINING DATA

- Preparing training data: images and their labels



The learning target is defined on the training data.

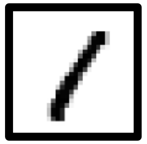
LEARNING TARGET



Ink → 1

No ink → 0

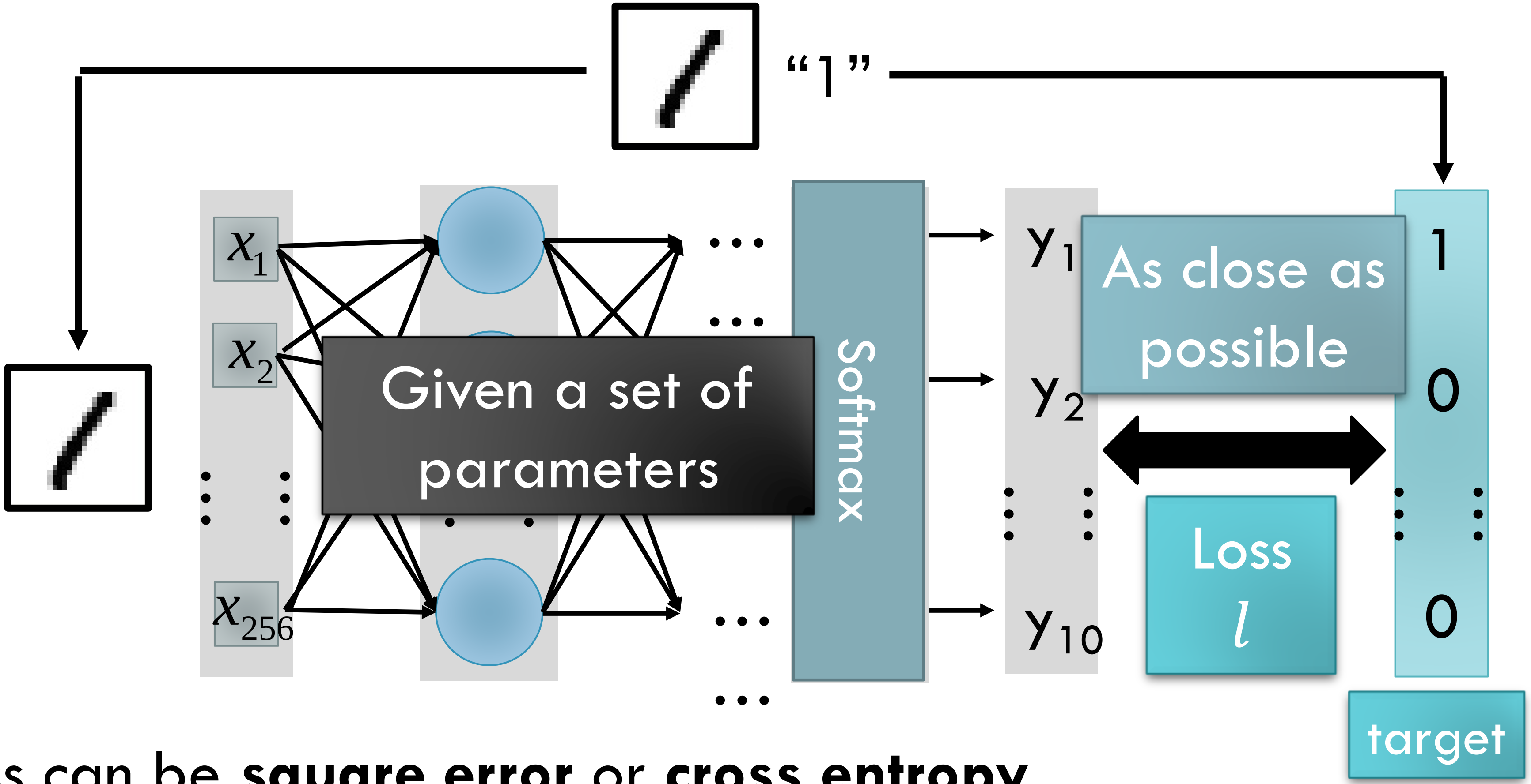
The learning target is ……

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 for the model to work well on average for all training examples.



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

Cross Entropy Function

LIKELIHOOD?

- Let's think about h_i as estimated probability of outcome i and y_i as it's true label (or probability).
- Say, We have "red" and "blue" balls. In the training set, we were given a sample of [R][B][R][B][B]. The $y_R = \frac{2}{5}, y_B = \frac{3}{5}$ to be able to properly model this specific training set.
  - Bad estimation for the model: $h_R = \frac{4}{5}, h_B = \frac{1}{5}$.

$$Likelihood_{bad} = \left(\frac{4}{5}\right)^2 \left(\frac{1}{5}\right)^3$$

  - Good estimation for the model: $h_R = \frac{2.5}{5}, h_B = \frac{2.5}{5}$.

$$Likelihood_{good} = \left(\frac{2.5}{5}\right)^2 \left(\frac{2.5}{5}\right)^3$$

LIKELIHOOD?

$$Likelihood = \prod_i h_i^{num(y_i)} , \sum h_i = 1$$

- For only two classes: $y_i \in \{0, 1\}$

$$likelihood = \prod_i h_i^{num(y_i)} = h_0^{num(y_i=0)} \times h_1^{num(y_i=1)}, h_0 + h_1 = 1$$

- We need to maximize the likelihood function to have a model that very well represents our training set.
- Let's take the logarithm of the likelihood function:

$$\log(likelihood) = \log\left(h_0^{num(y_i=0)} \times h_1^{num(y_i=1)}\right)$$
$$= num(y_i = 0) \times \log h_0 + num(y_i = 1) \times \log h_1$$
$$= num(y_i = 0) \times \log(1 - h_1) + num(y_i = 1) \times \log h_1$$

NEGATIVE OF LOG LIKELIHOOD

$$\log(likelihood) = num(y_i = 0) \times \log(1 - h_1) + num(y_i = 1) \times \log h_1$$
$$= \sum_i y_i \log h_i + (1 - y_i) \log(1 - h_i)$$

- Since the likelihood function is a monotonic function, the maximum point of the likelihood function is at the minimum point of the negative of logarithm of the likelihood function.
- The negative of the logarithm of likelihood function divided by n is:

$$-\frac{1}{n} H(y, h) = -\frac{1}{n} \sum_i y_i \log h_i + (1 - y_i) \log(1 - h_i)$$

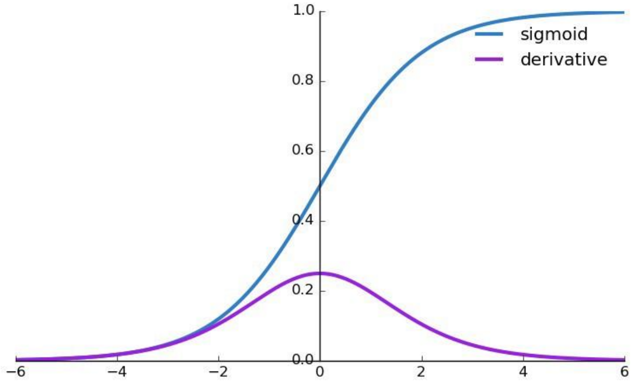
- Why we are taking log:
  - Taking derivative from a summation is easier than a multiplication
  - Log tends to compress a dynamic range

DIFFERENT LOSS FUNCTIONS

MSE or SSE

Sigmoid activation with MSE cost function gives you the following partial derivative:

$$\frac{\partial C}{\partial \theta} \approx a \times \sigma'(z)$$



Binary Cross entropy

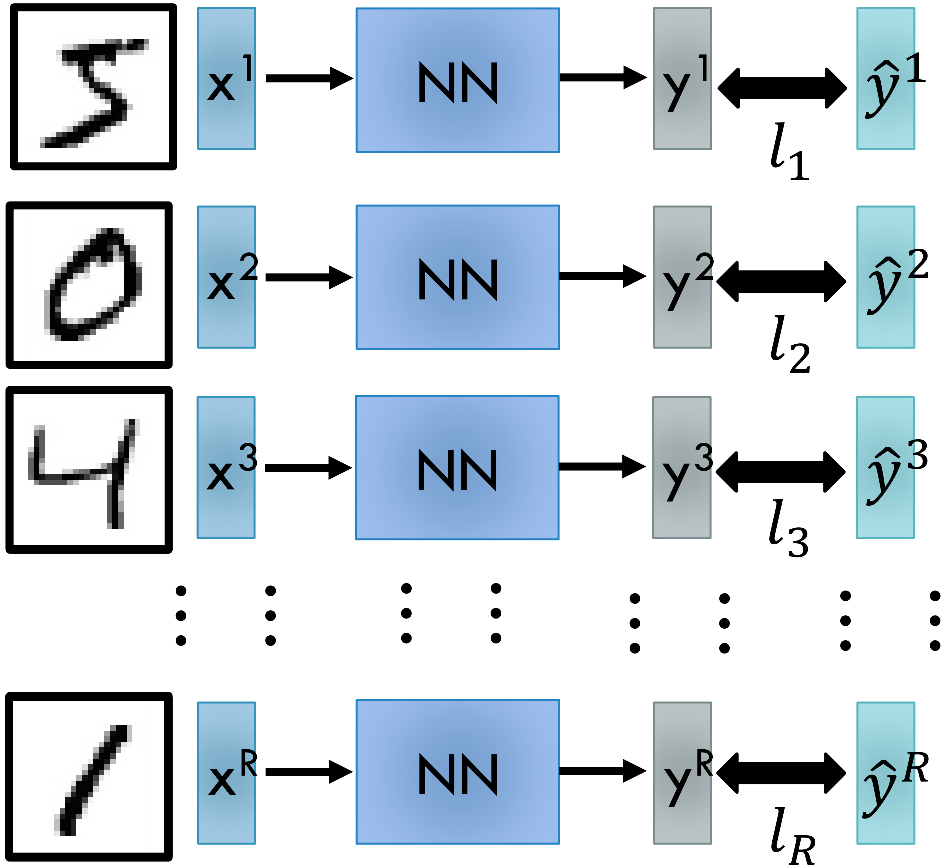
$$D(H,Y) = -\frac{1}{n}\sum_i Y_i \log H_i + (1 - Y_i)\log(1 - H_i)$$

$$\frac{\partial C}{\partial \theta} \approx \sum_j x_j\left(\sigma\left(z_j\right) - y_j\right)$$

Good reading: https://rohanvarma.me/Loss-Functions/

TOTAL LOSS

For all training data …



Total Loss:

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

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

THREE STEPS FOR DEEP LEARNING

Step 1: define a set of function

Step 2: goodness of function

Step 3: pick the best function

HOW TO PICK THE BEST FUNCTION

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

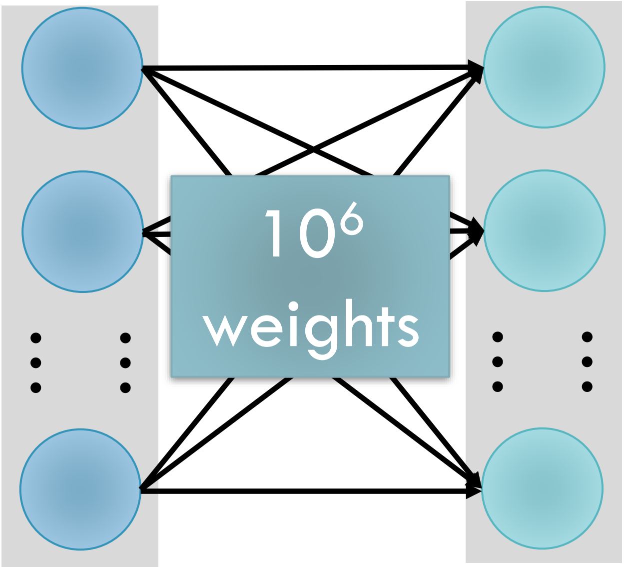
Enumerate all possible values

Network p me

$\theta = \{\theta_1 \quad \theta_3, \cdots\}$

Millio of pa meters

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



1000 neurons

1000 neurons

GRADIENT DESCENT

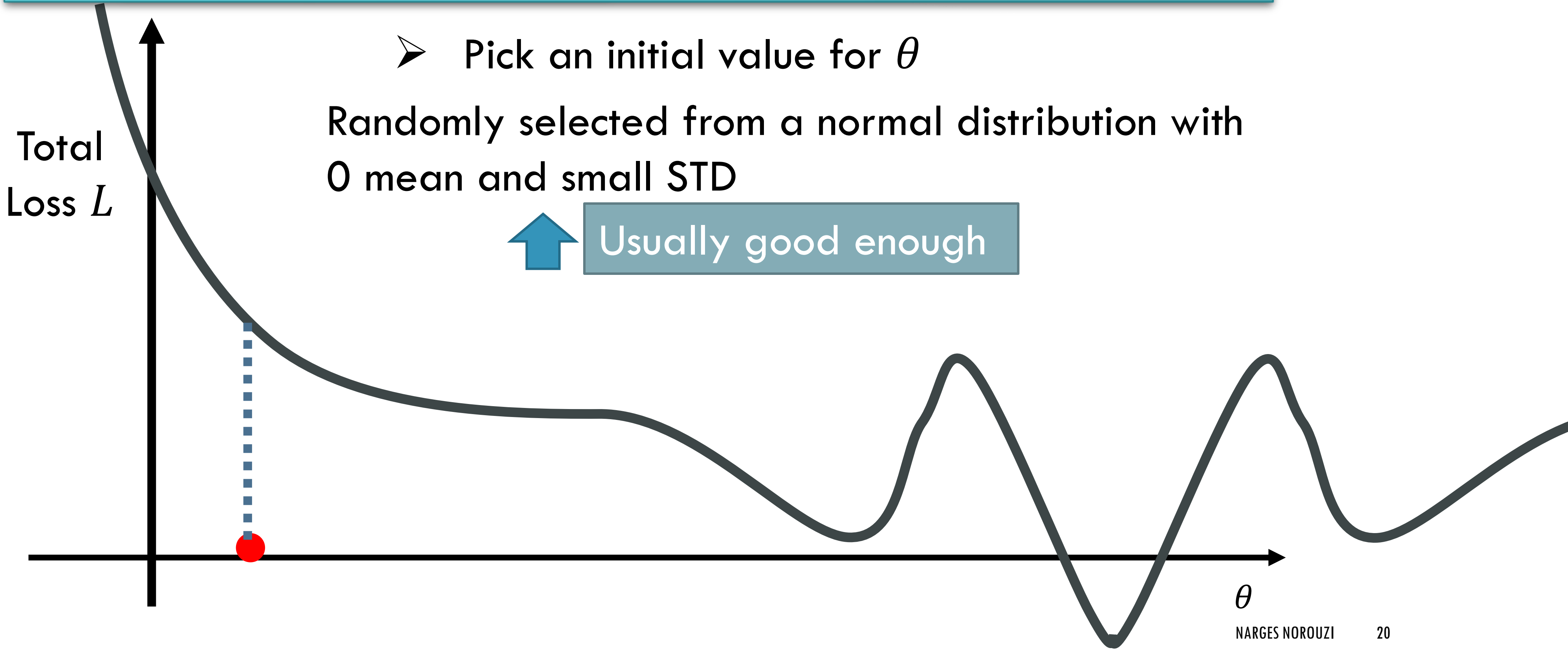
Network parameters $\theta = \{\theta_1, \theta_2, \cdots, b_1, b_2, \cdots\}$

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

- Pick an initial value for θ

Randomly selected from a normal distribution with 0 mean and small STD

Usually good enough



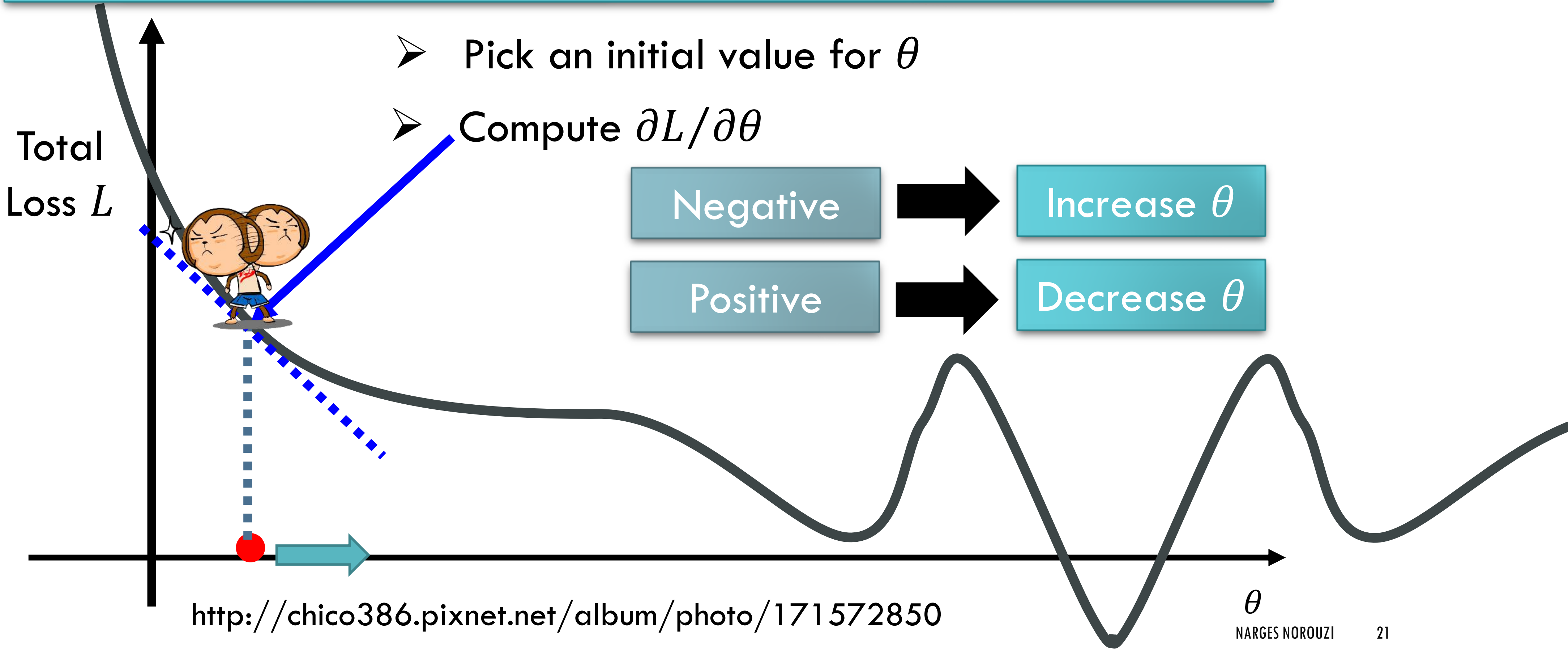
GRADIENT DESCENT

Network parameters $\theta = \{\theta_1, \theta_2, \cdots\}$

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

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

Negative	→	Increase θ
Positive	→	Decrease θ



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

GRADIENT DESCENT

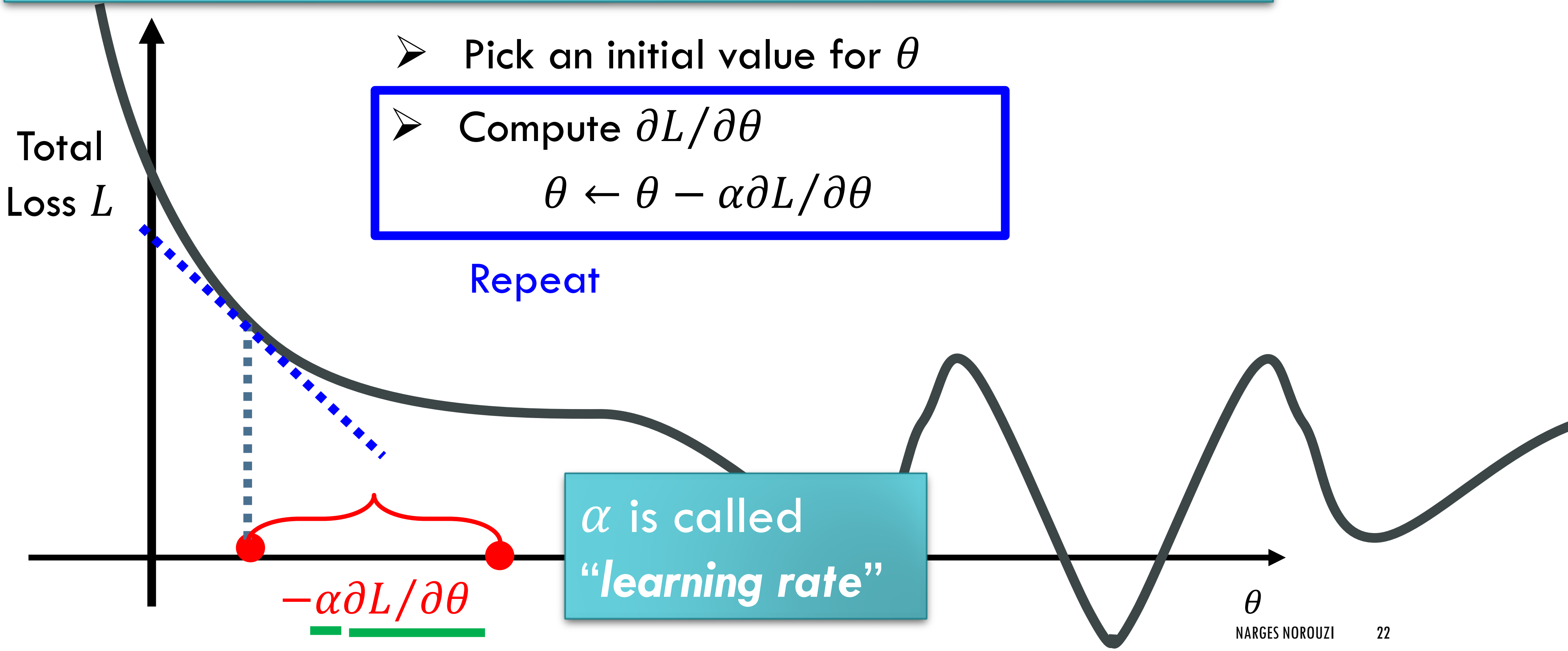
Network parameters $\theta = \{\theta_1, \theta_2, \cdots\}$

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

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

$$\theta \leftarrow \theta - \alpha \partial L / \partial \theta$$

Repeat



α is called
"learning rate"

GRADIENT DESCENT

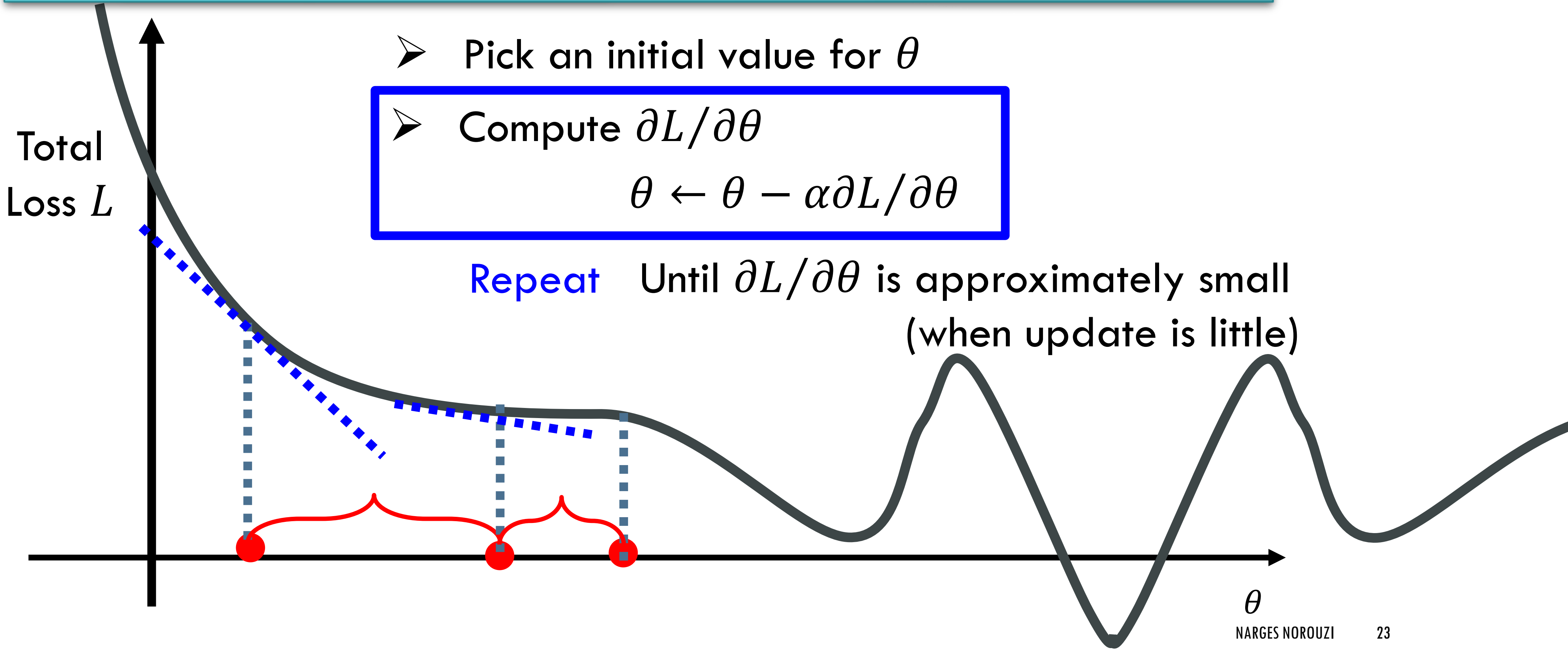
Network parameters $\theta = \{\theta_1, \theta_2, \cdots\}$

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

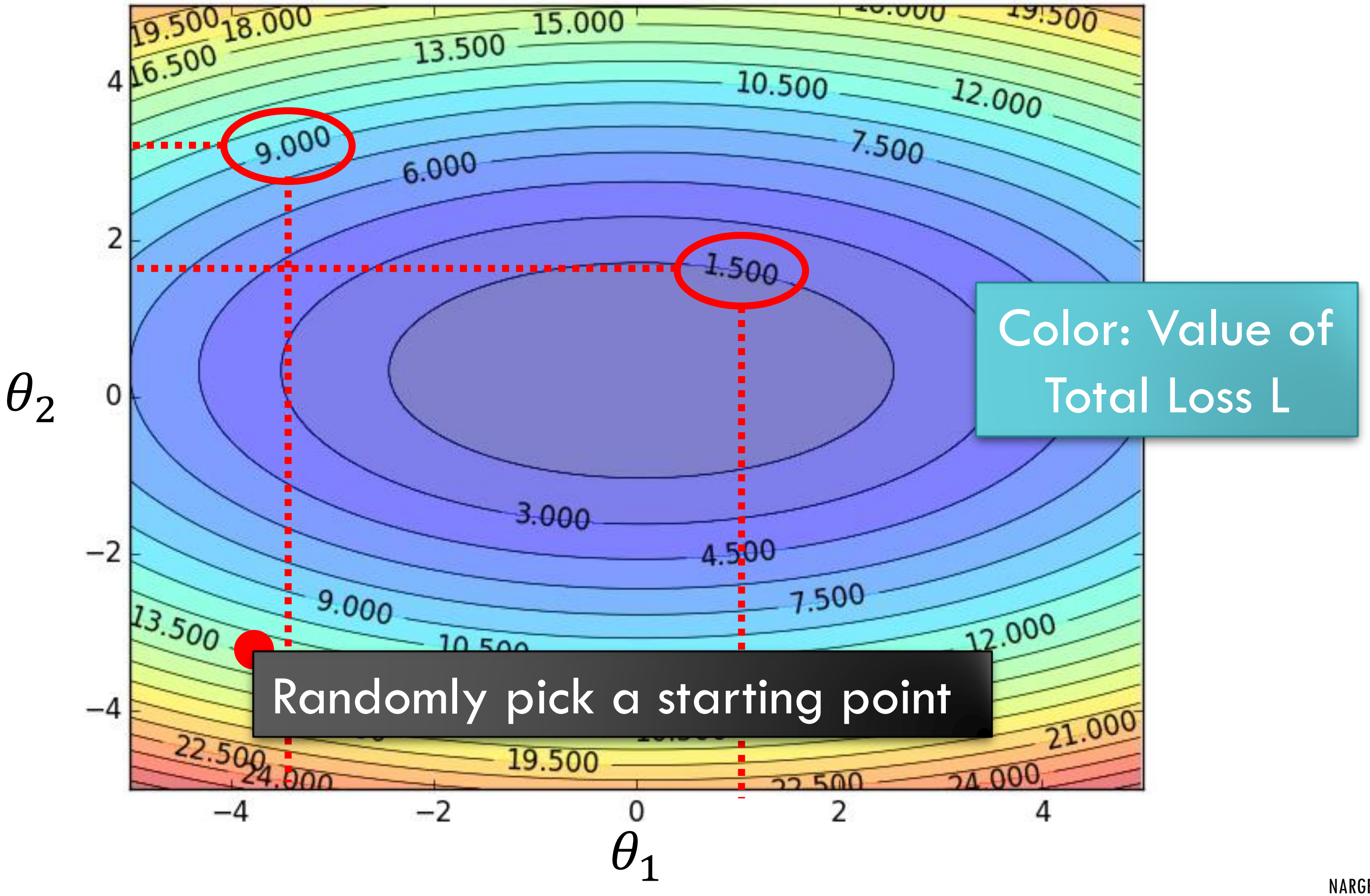
- Pick an initial value for θ

$$\boxed{\begin{array}{l} \text{➢ Compute } \partial L / \partial \theta \\ \qquad \theta \leftarrow \theta - \alpha \partial L / \partial \theta \end{array}}$$

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

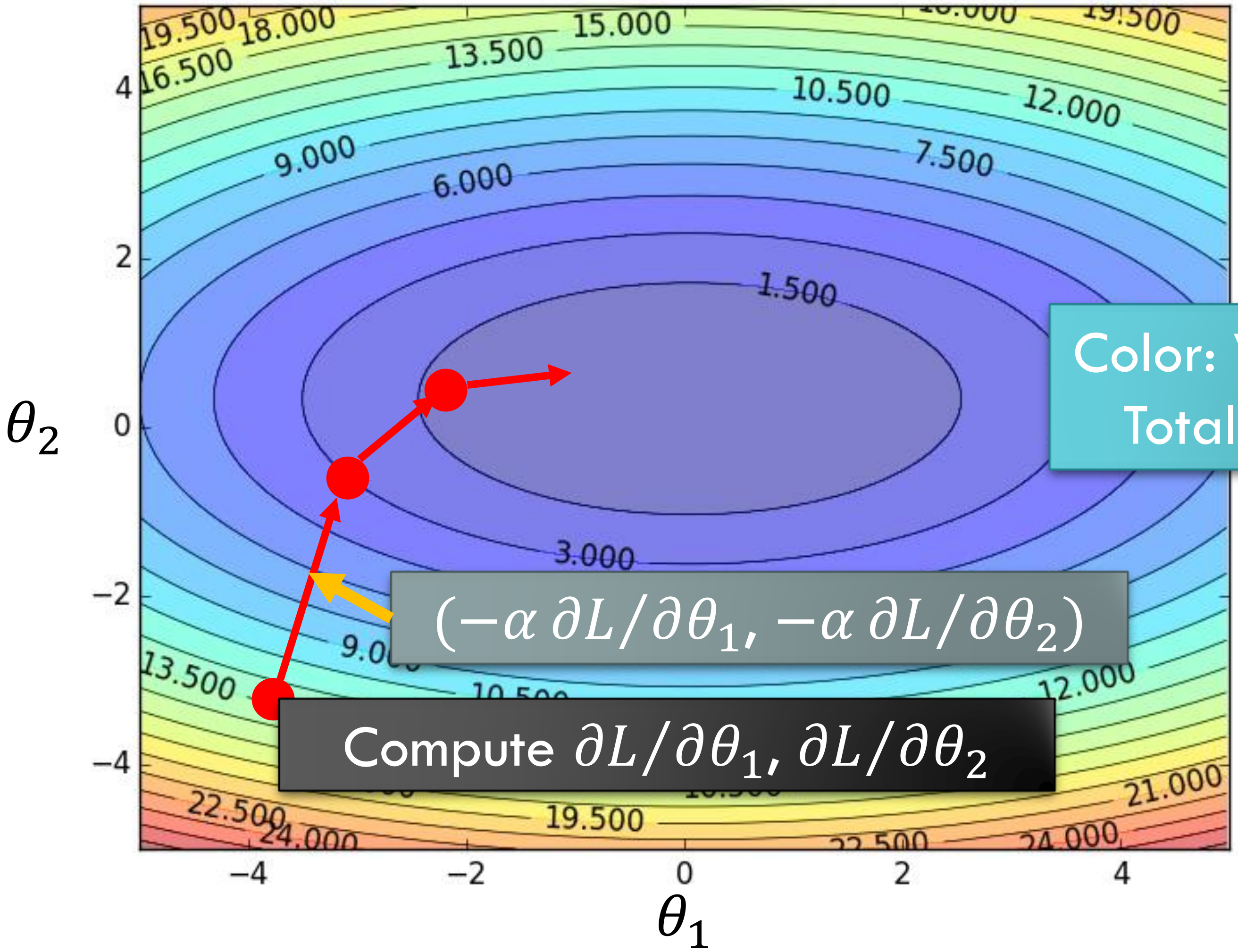


GRADIENT DESCENT



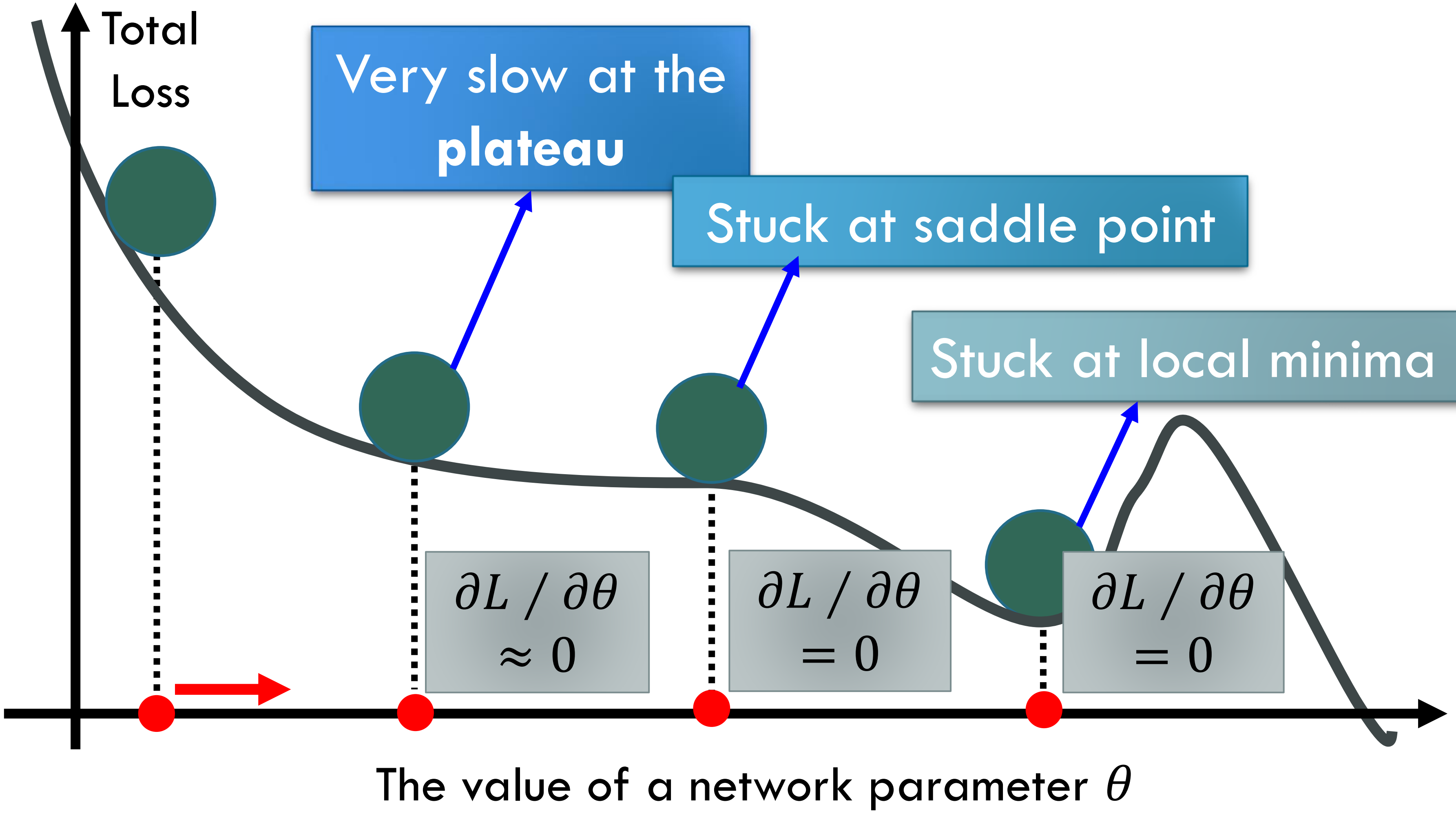
GRADIENT DESCENT

Hopfully, we would reach a minima …..



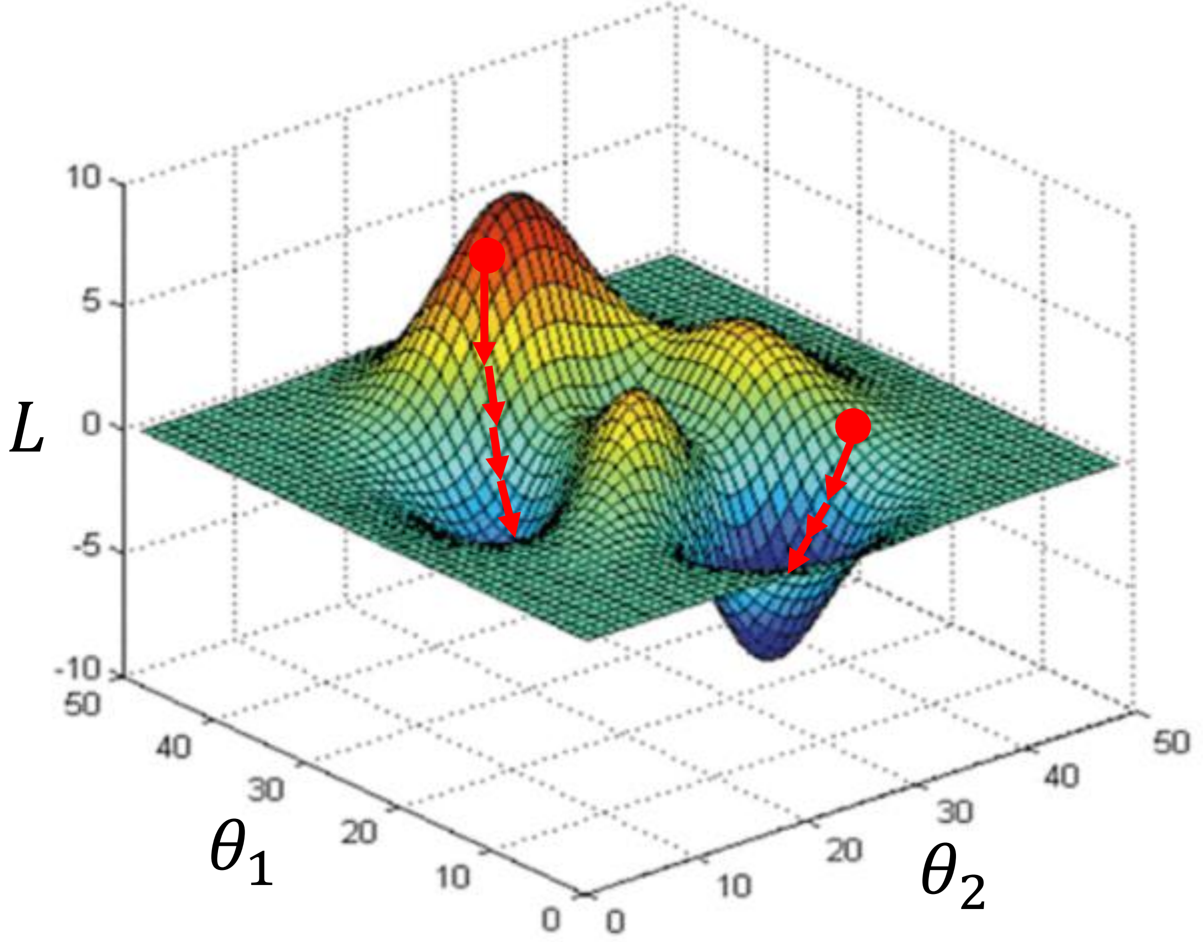
Color: Value of Total Loss L

LOCAL MINIMA



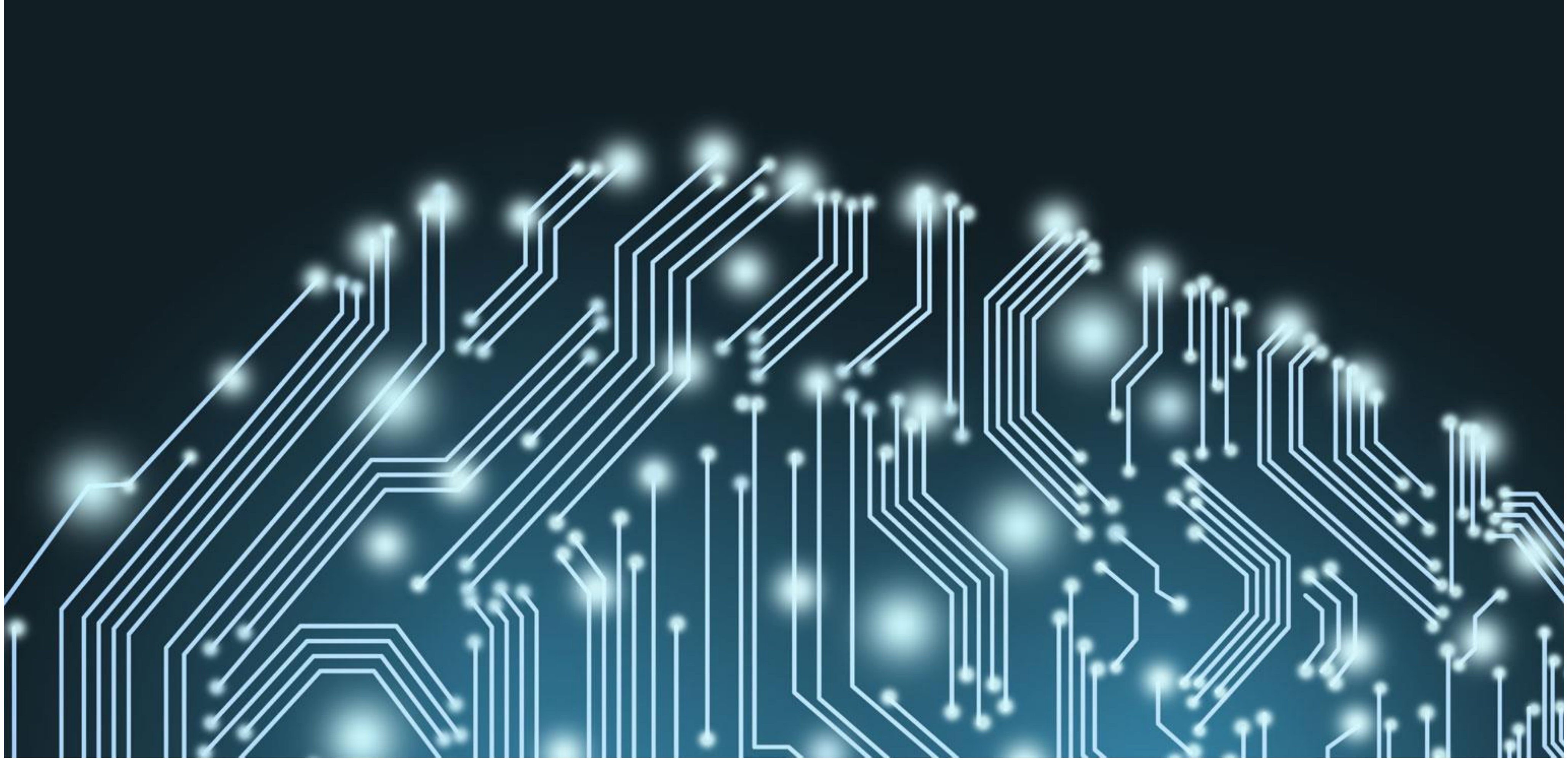
LOCAL MINIMA

- Gradient descent never guarantee global minima



Different initial point

Reach different minima, so different results



BACKPROPAGATION VIDEOS

OUTLINE

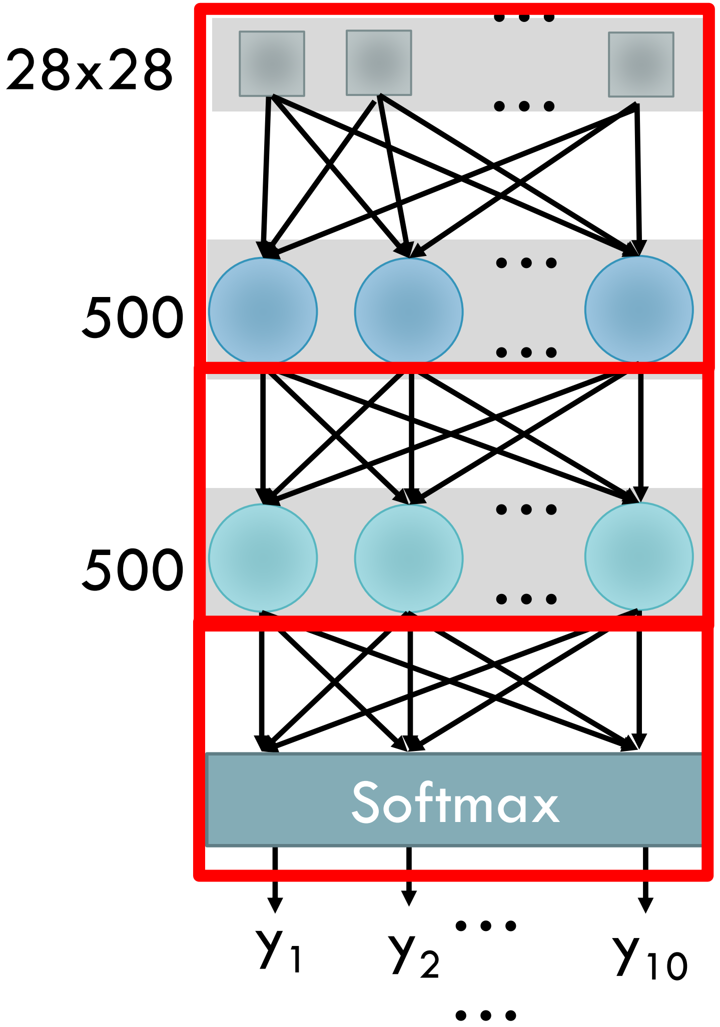
Introduction of Deep Learning

"Hello World" for Deep Learning

Tips for Deep Learning

KERAS

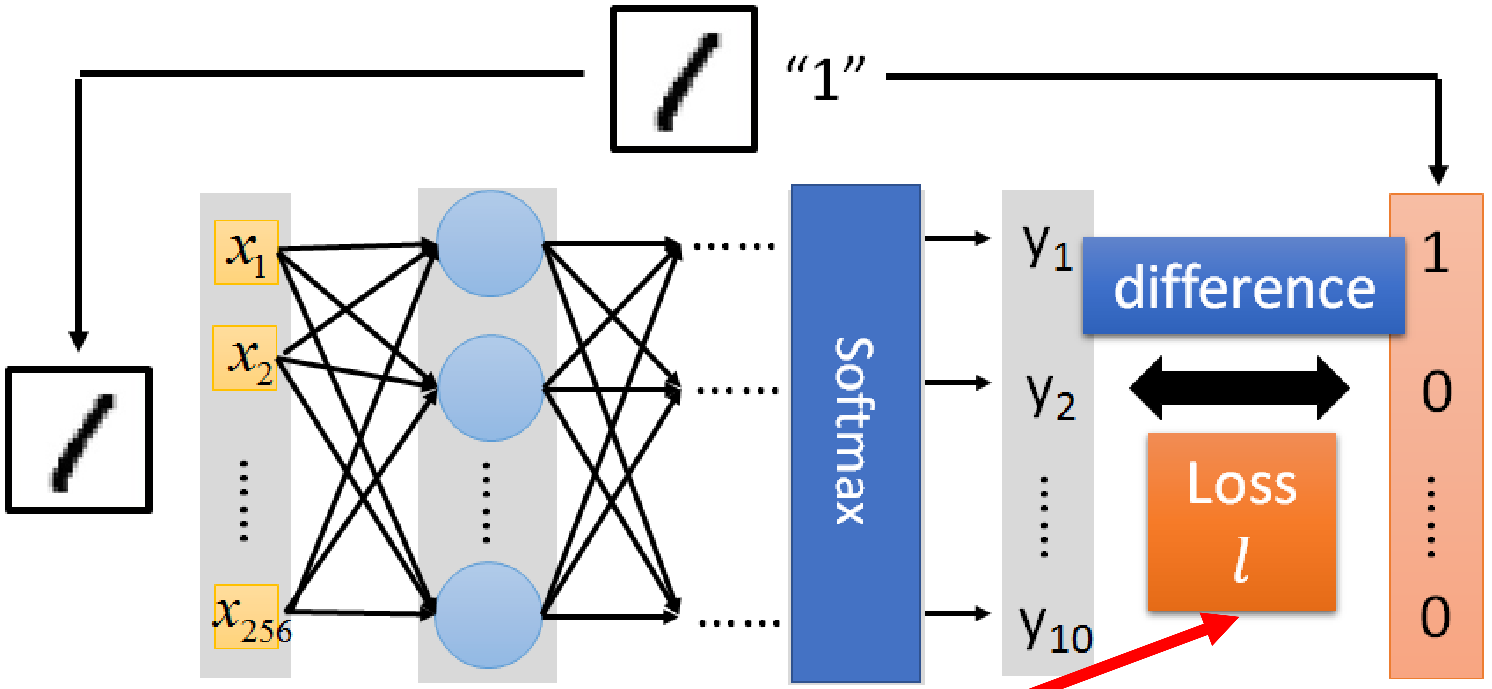
Step 1: define a set of function → Step 2: goodness of function → Step 3: pick the best function




```
model = Sequential()

model.add(Dense(units=500,
                input_dim=28*28,
                activation='sigmoid'))

model.add(Dense(units=500,
                activation='sigmoid'))

model.add(Dense(units=10,
                activation='softmax'))
```

KERAS

Step 1: define a set of function → **Step 2: goodness of function** → Step 3: pick the best function




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

KERAS

Step 1: define a set of function → Step 2: goodness of function → **Step 3: pick the best function**

Step 3.1: Configuration

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

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

0.1

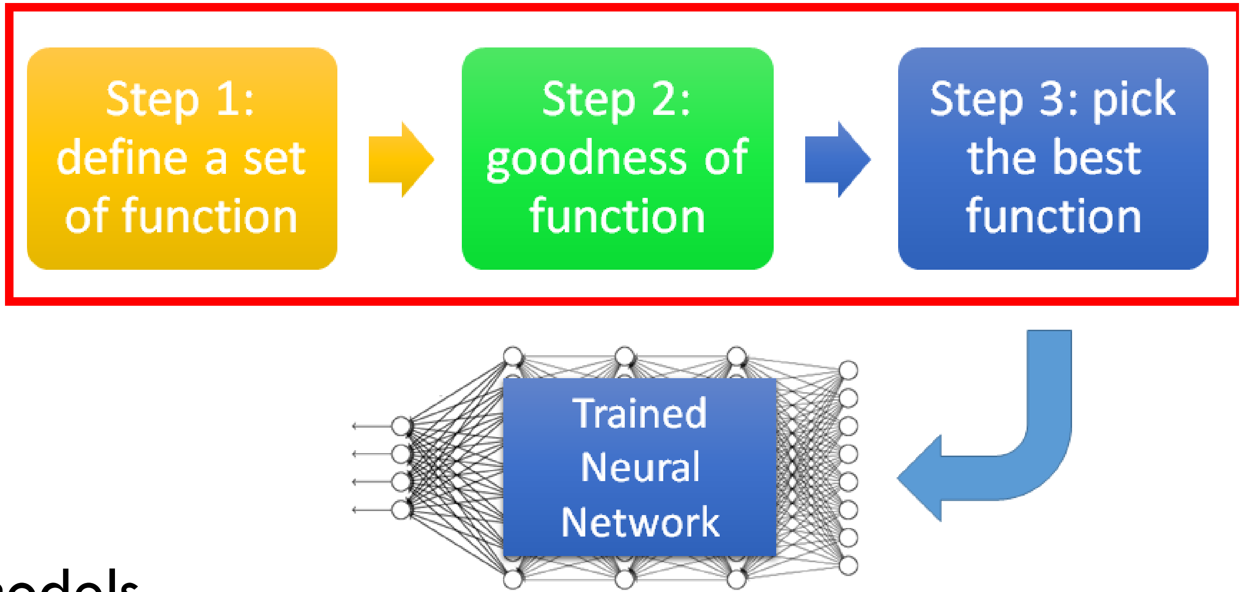
Step 3.2: Find the optimal network parameters

```
model.fit(x_train, y_train, epochs = 200, batch_size = 100)
```

Training data (Images)

Labels (digits)

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 the Testing set:', score[0])
print('Accuracy of Testing set:', score[1])
```

case 2:

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

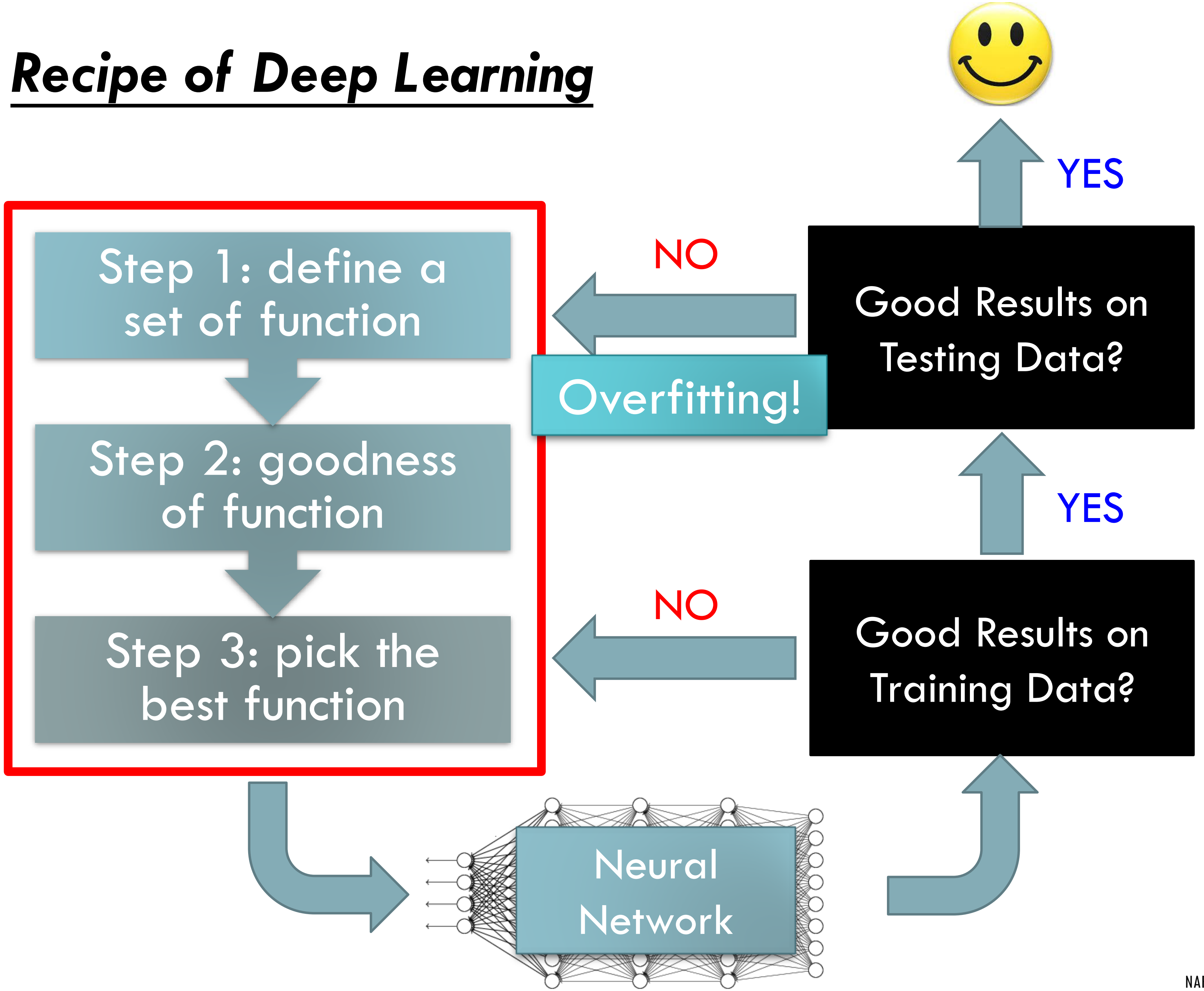
OUTLINE

Introduction of Deep Learning

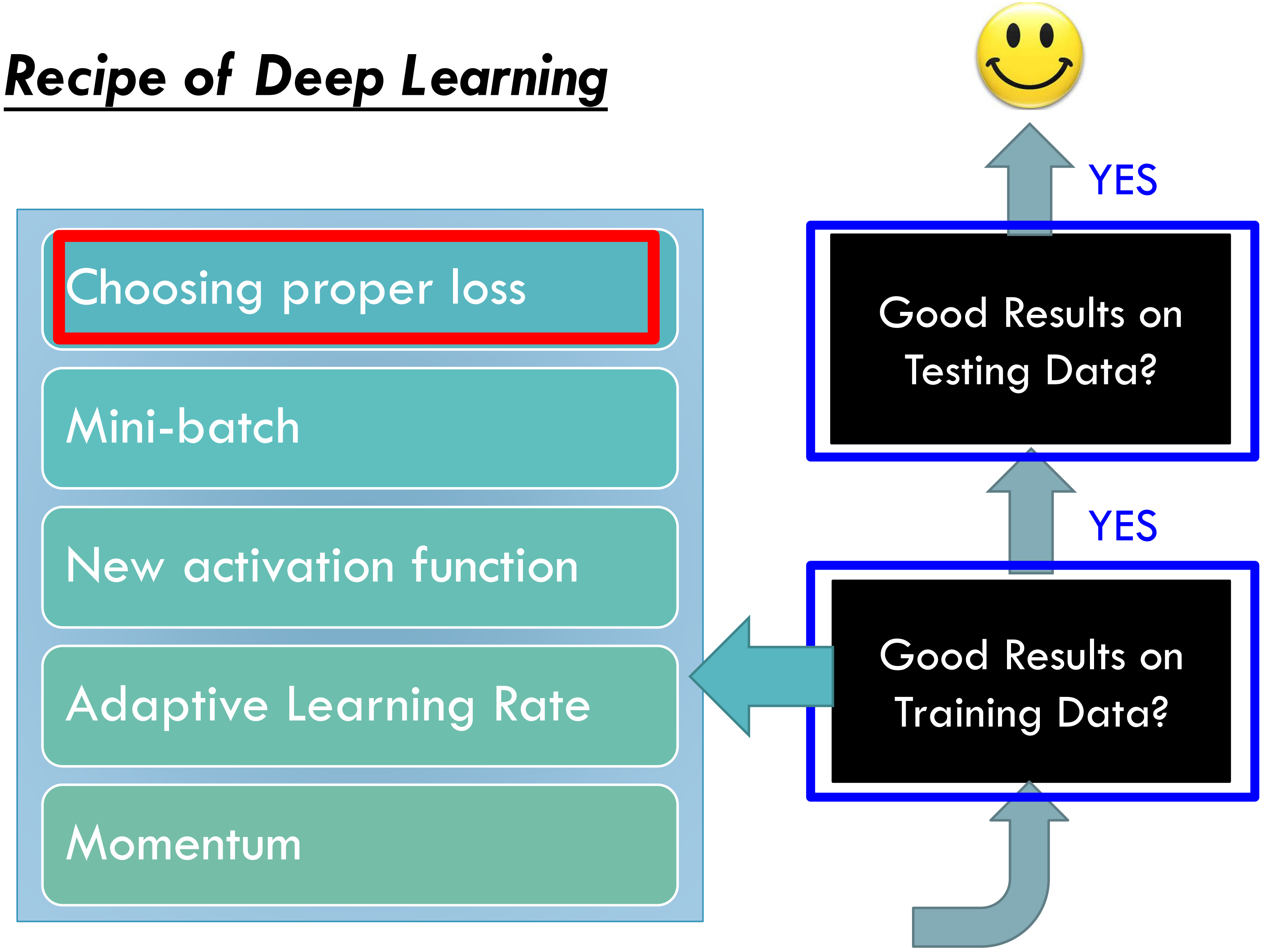
“Hello World” for Deep Learning

Tips for Deep Learning

Recipe of Deep Learning



Recipe of Deep Learning



DEMO

Square Error

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

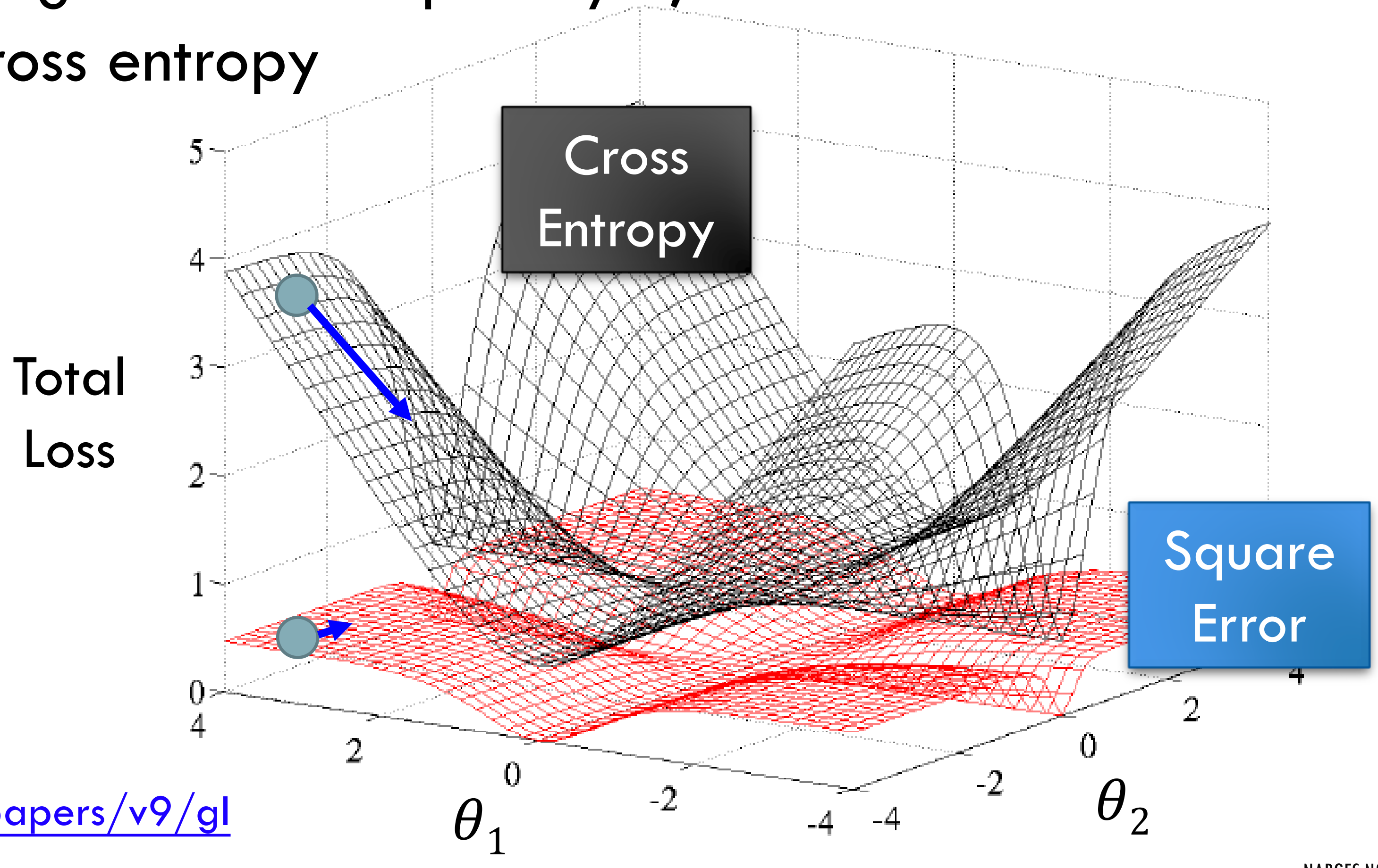
Binary Cross Entropy

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

Several alternatives: https://keras.io/losses/

CHOOSING PROPER LOSS

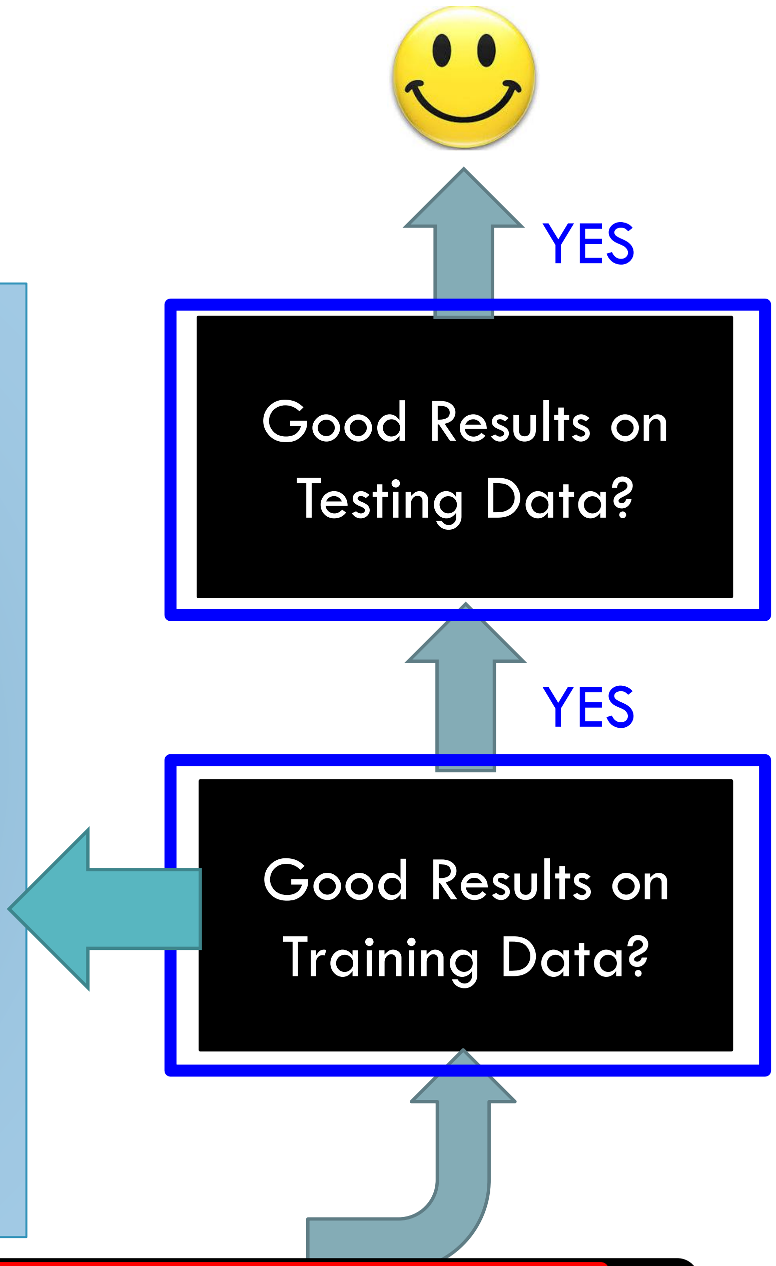
When using softmax output layer, choose cross entropy



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

Recipe of Deep Learning

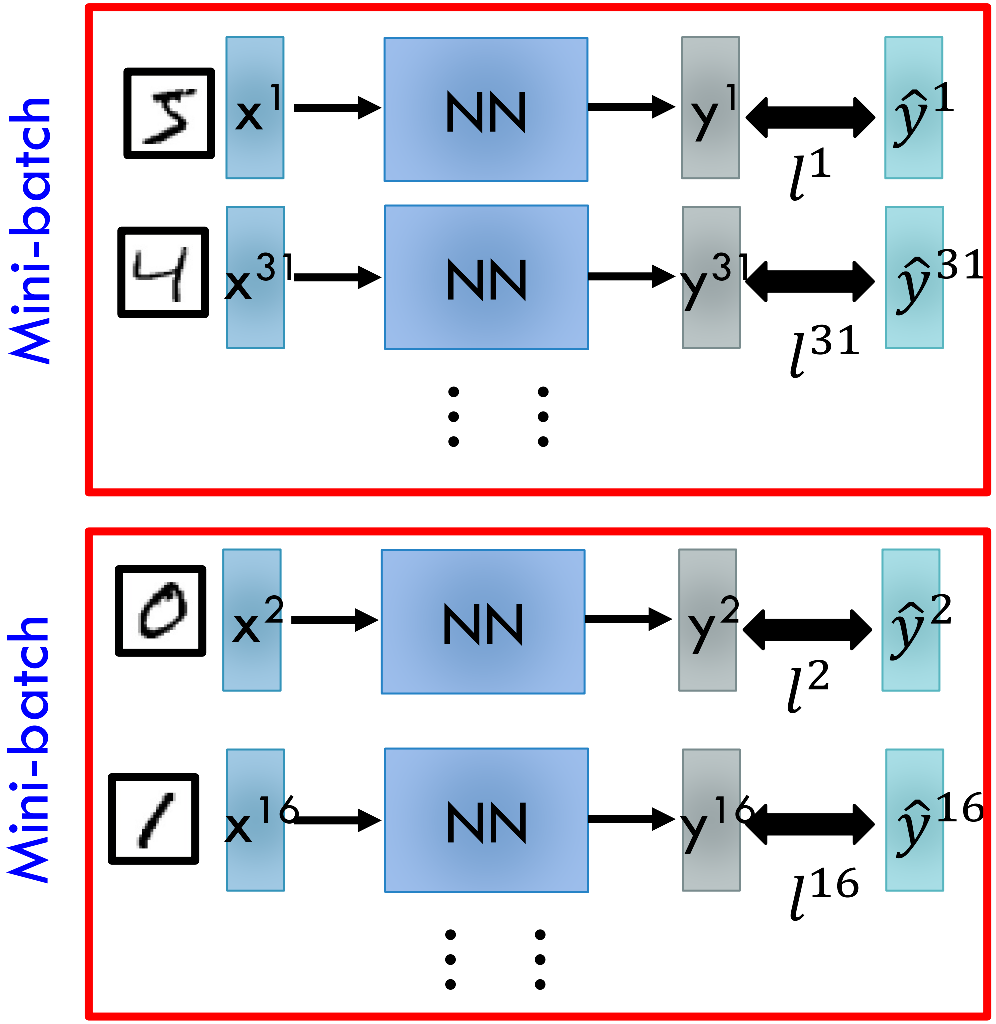
- Choosing proper loss
- Mini-batch
- New activation function
- Adaptive Learning Rate
- Momentum




```
model.fit(x_train, y_train, epochs = 200, batch_size = 100)
```

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

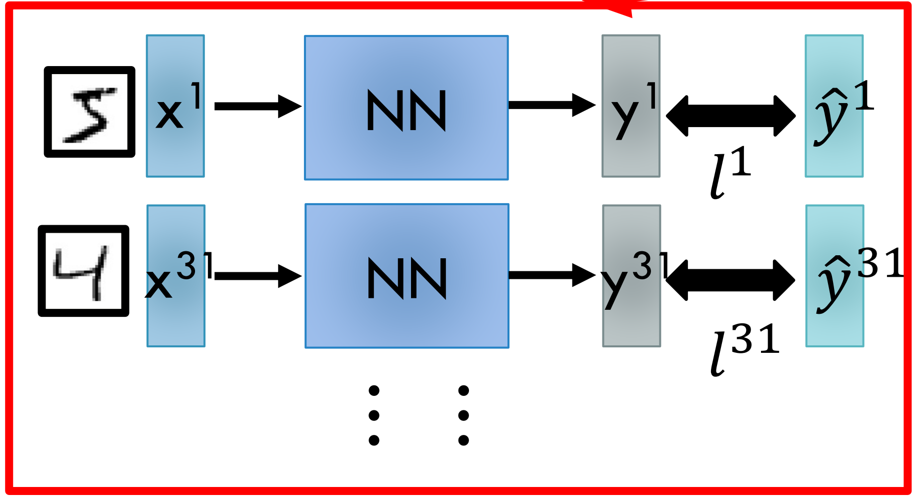
one epoch

Repeat the above process

MINI-BATCH

```
model.fit(x_train, y_train, epochs = 200, batch_size = 100)
```

Mini-batch



100 examples in a mini-batch

Repeat 200 times

- 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

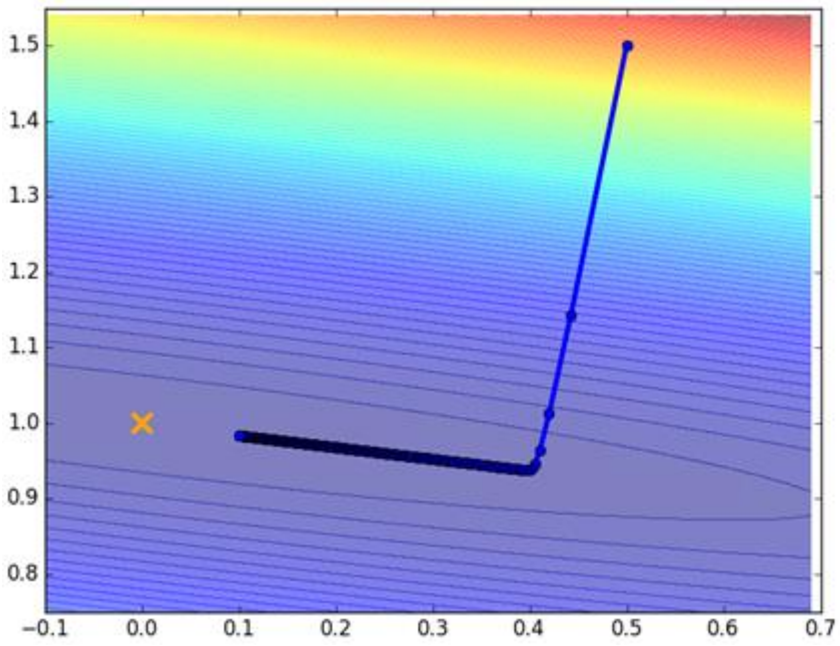
  ⋮

- Until all mini-batches have been picked

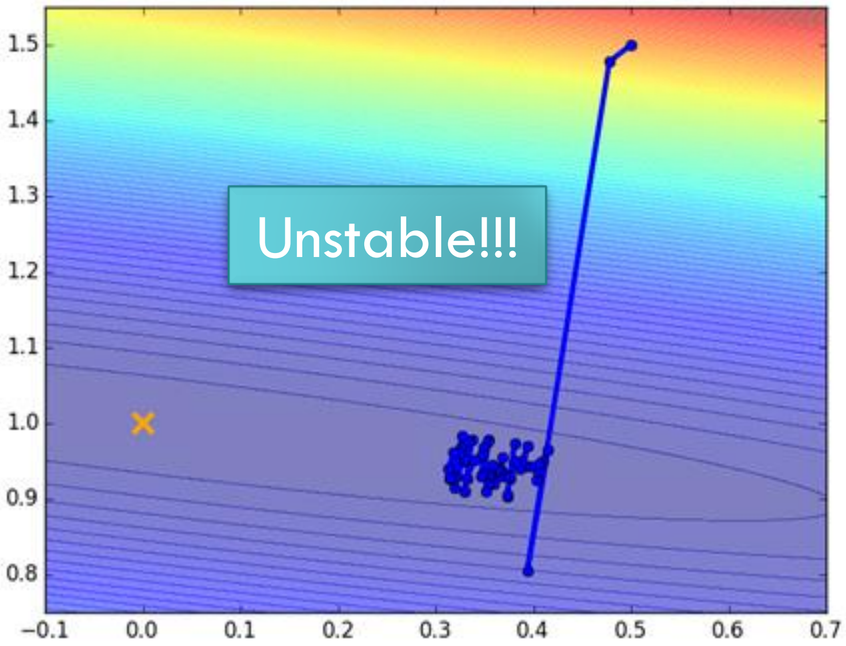
one epoch

MINI-BATCH

Original Gradient Descent



With Mini-batch

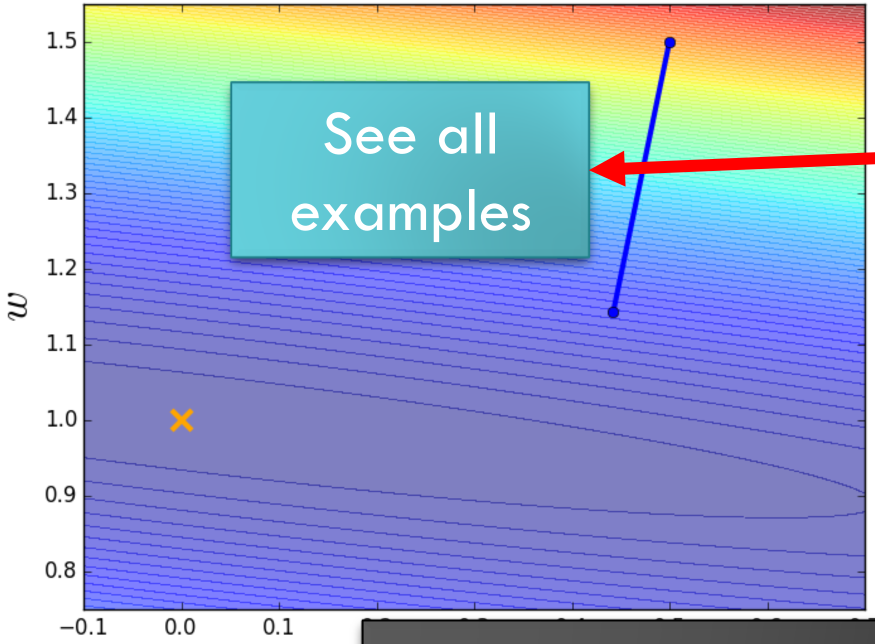


The colors represent the total loss.

MINI-BATCH IS FASTER

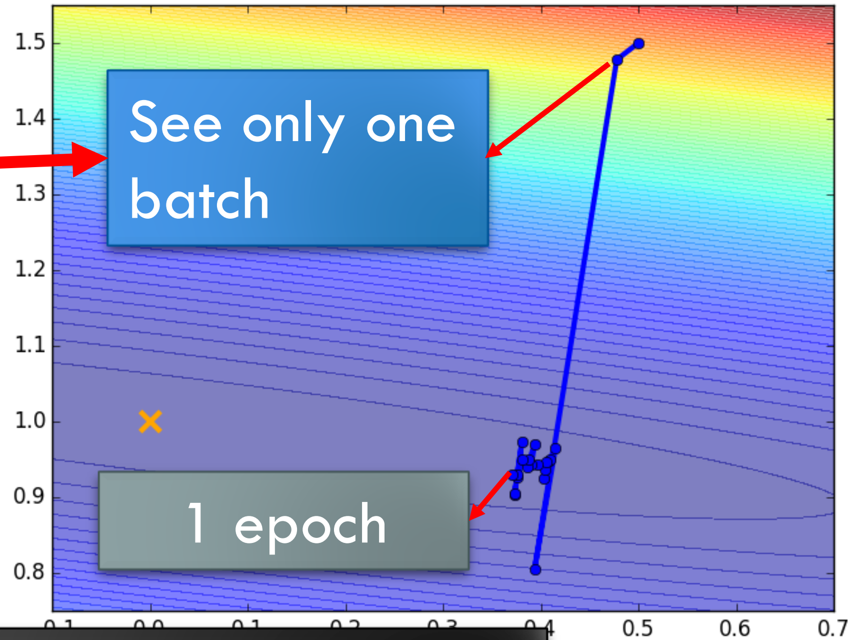
Original Gradient Descent

Update after seeing all examples



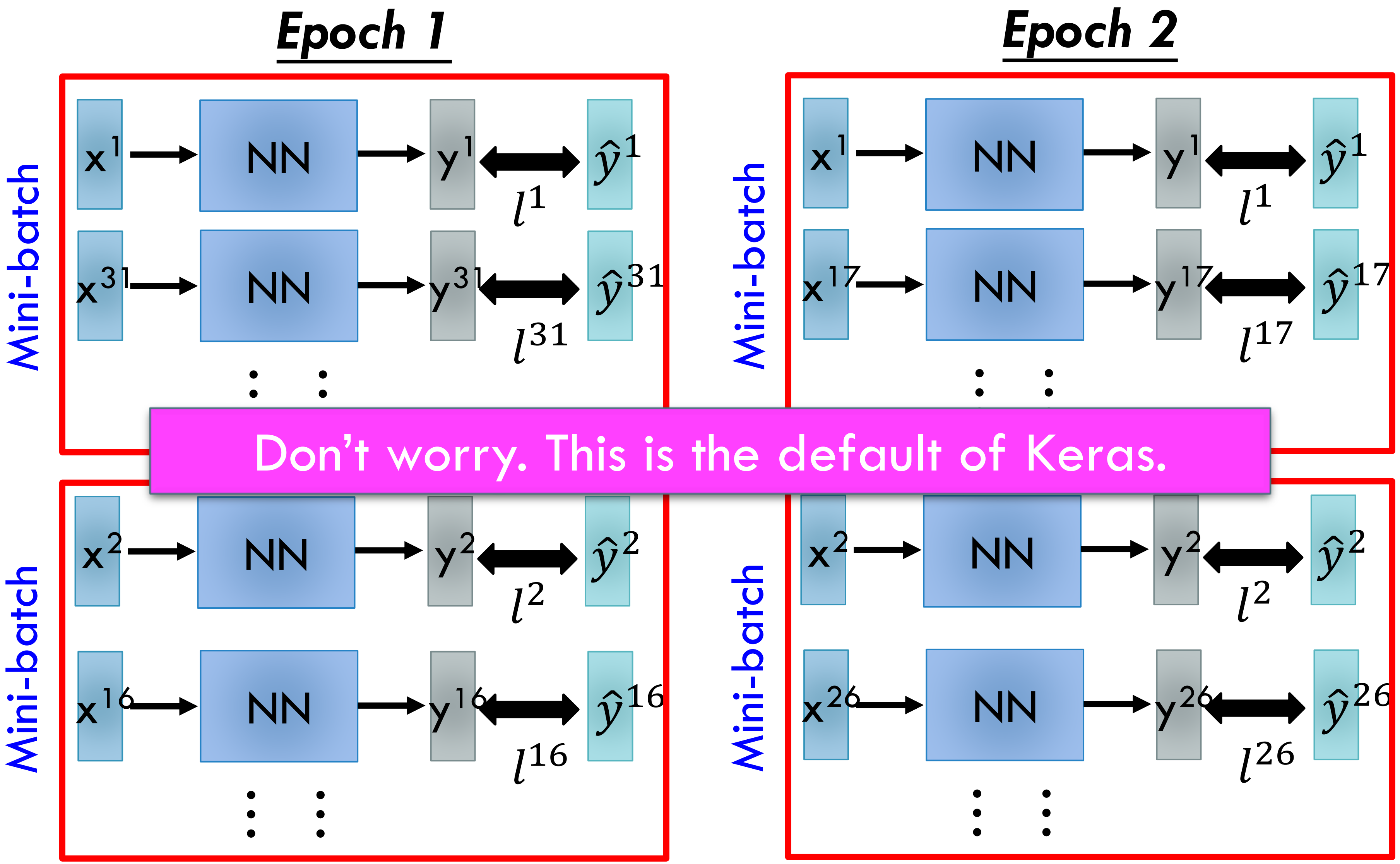
With Mini-batch

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



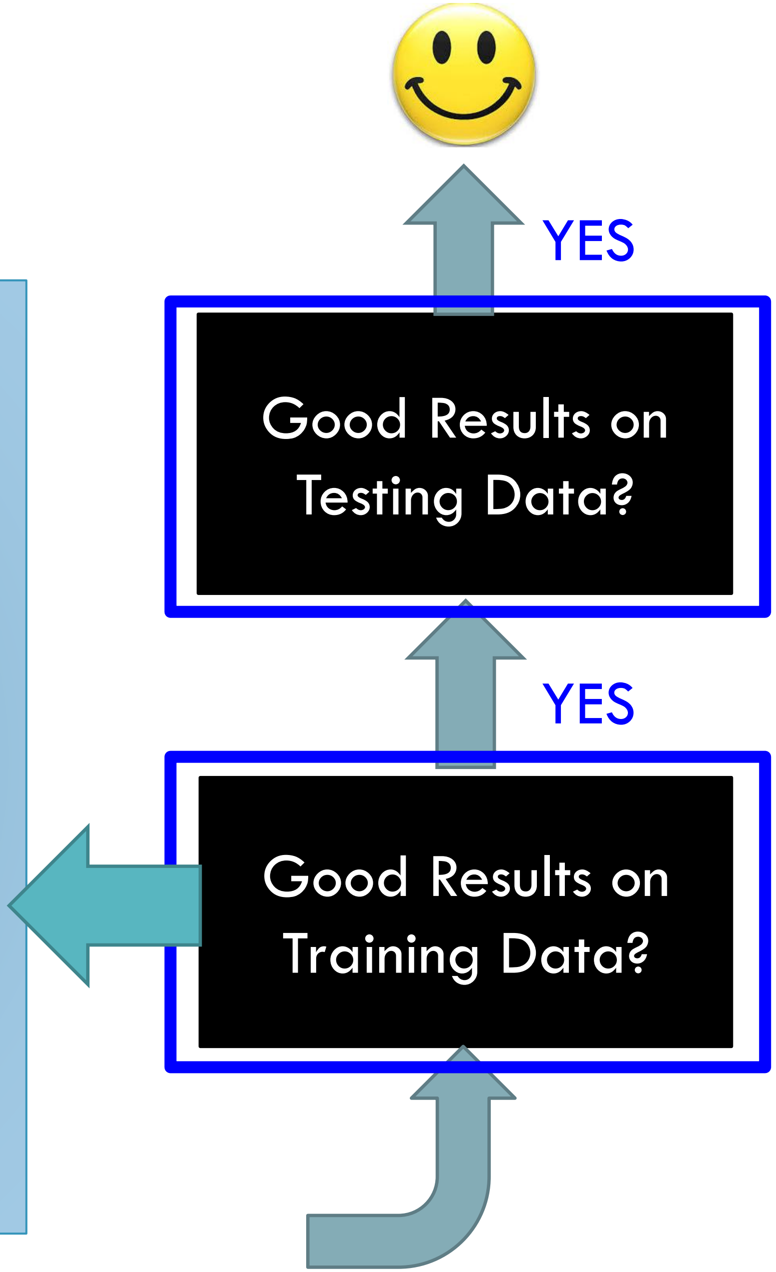
Mini-batch has better performance!

Shuffle the training examples for each epoch

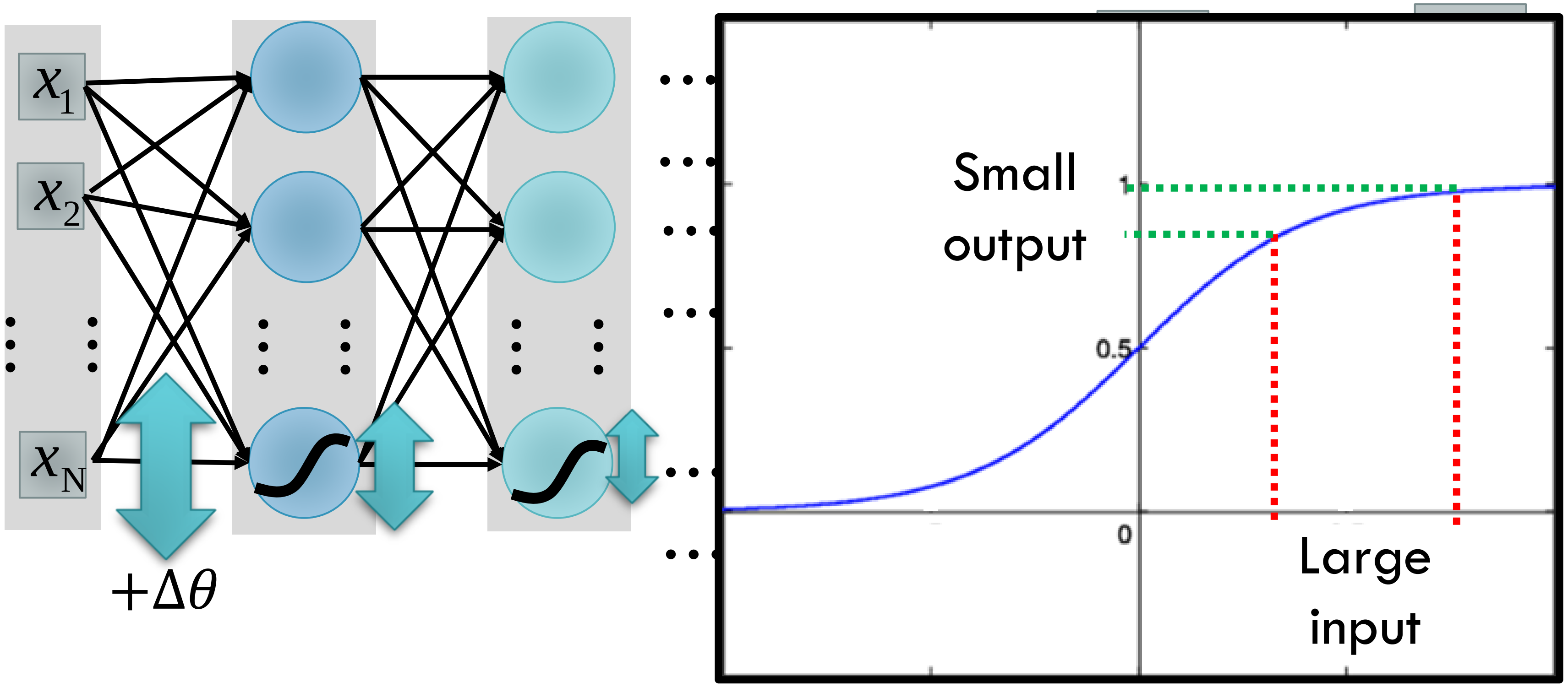


Recipe of Deep Learning

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

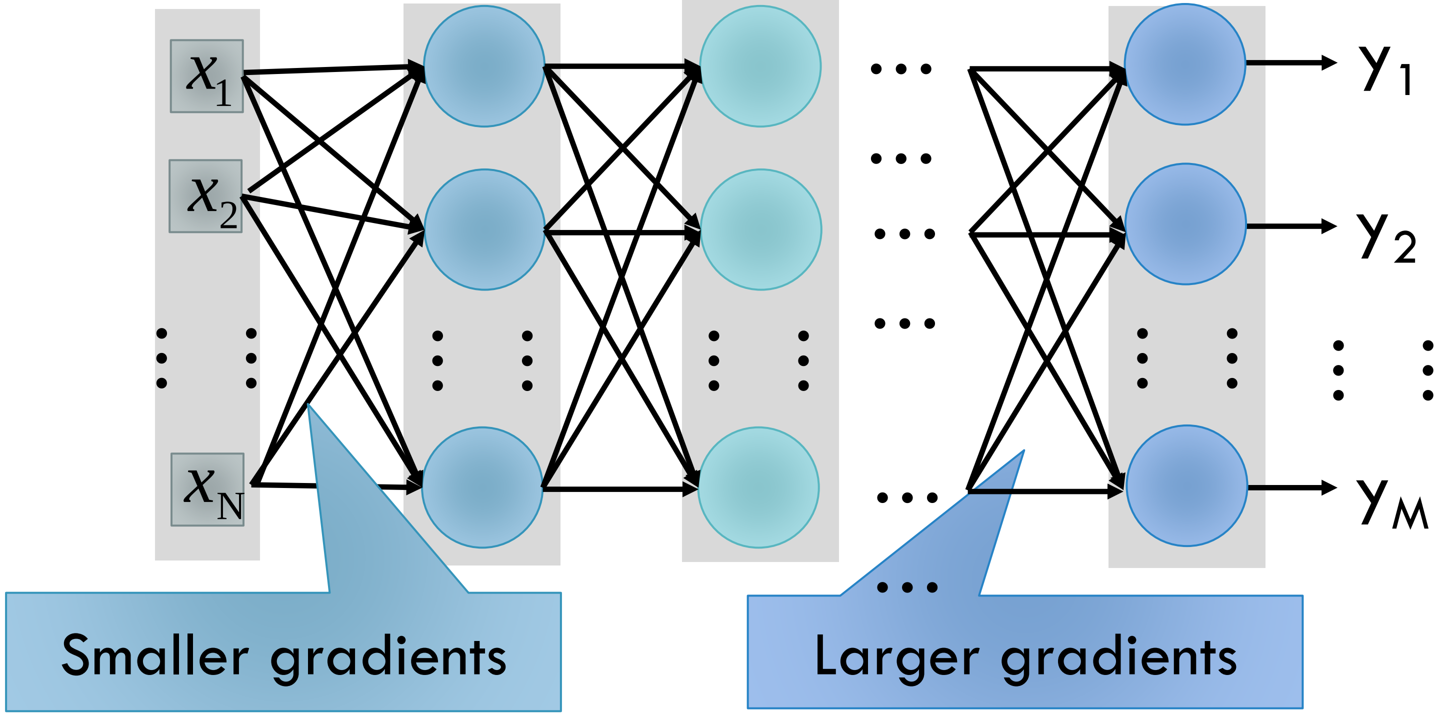


VANISHING GRADIENT PROBLEM



Intuitive way to compute the derivatives … $\frac{\partial l}{\partial \theta} =? \frac{\Delta l}{\Delta \theta}$

VANISHING GRADIENT PROBLEM



Learn very slow

Almost random

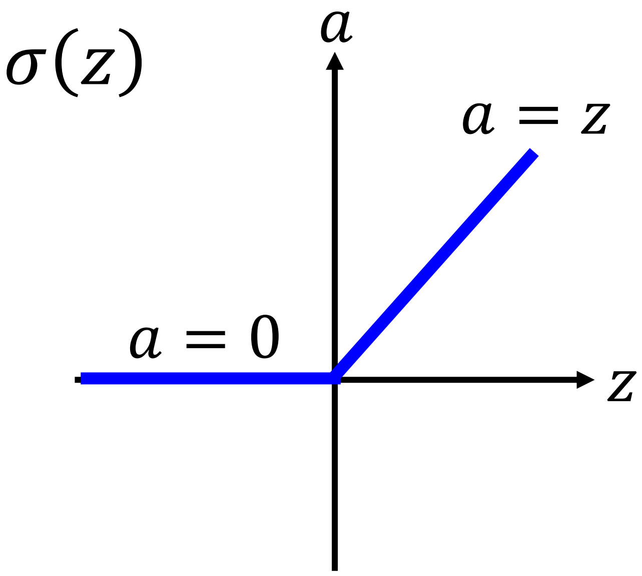
Learn very fast

Already converge

based on random!?

RELU

- Rectified Linear Unit (ReLU)

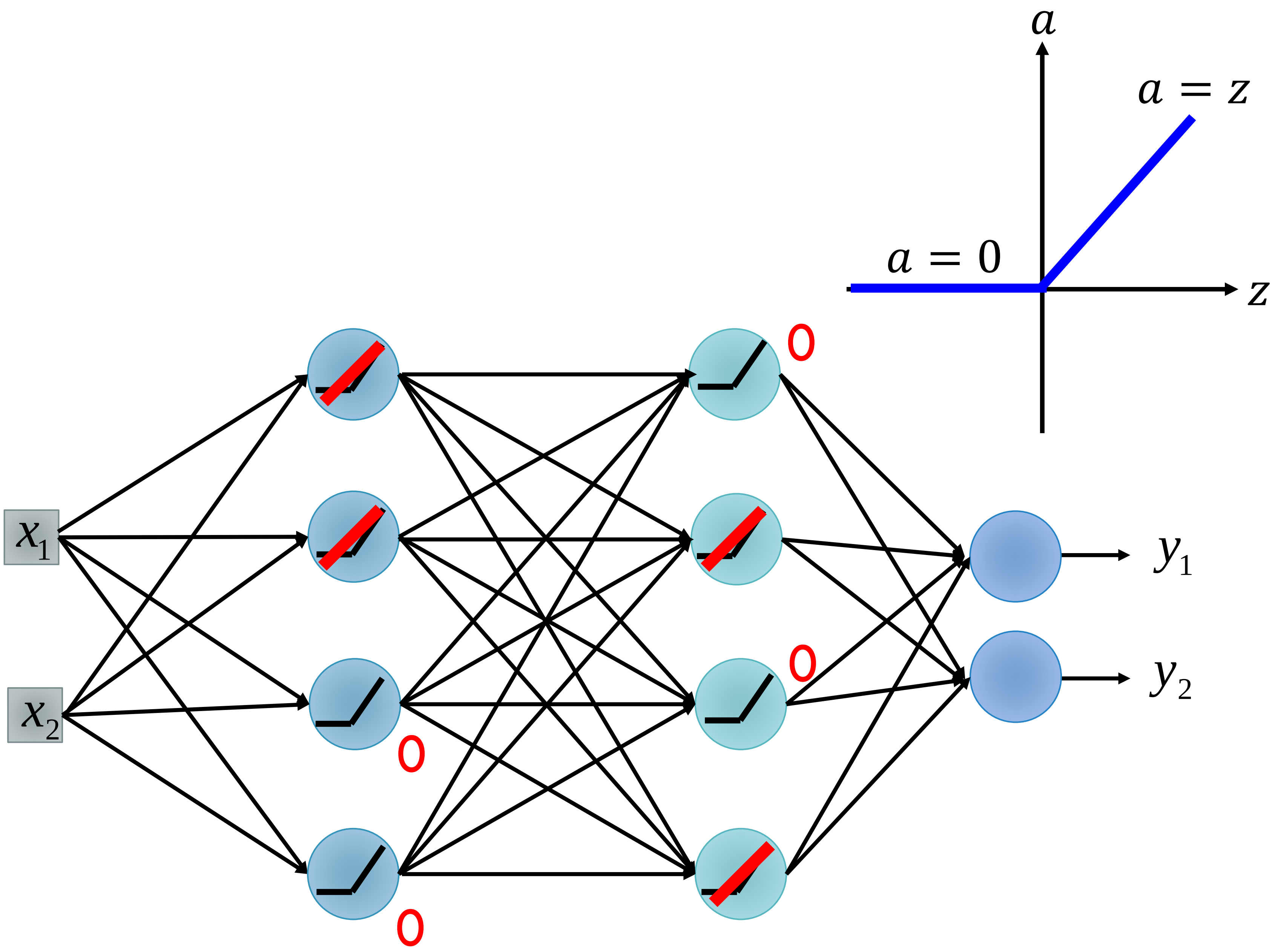


[Xavier Glorot, AISTATS'11]
[Andrew L. Maas, ICML'13]
[Kaiming He, arXiv'15]

Reason:

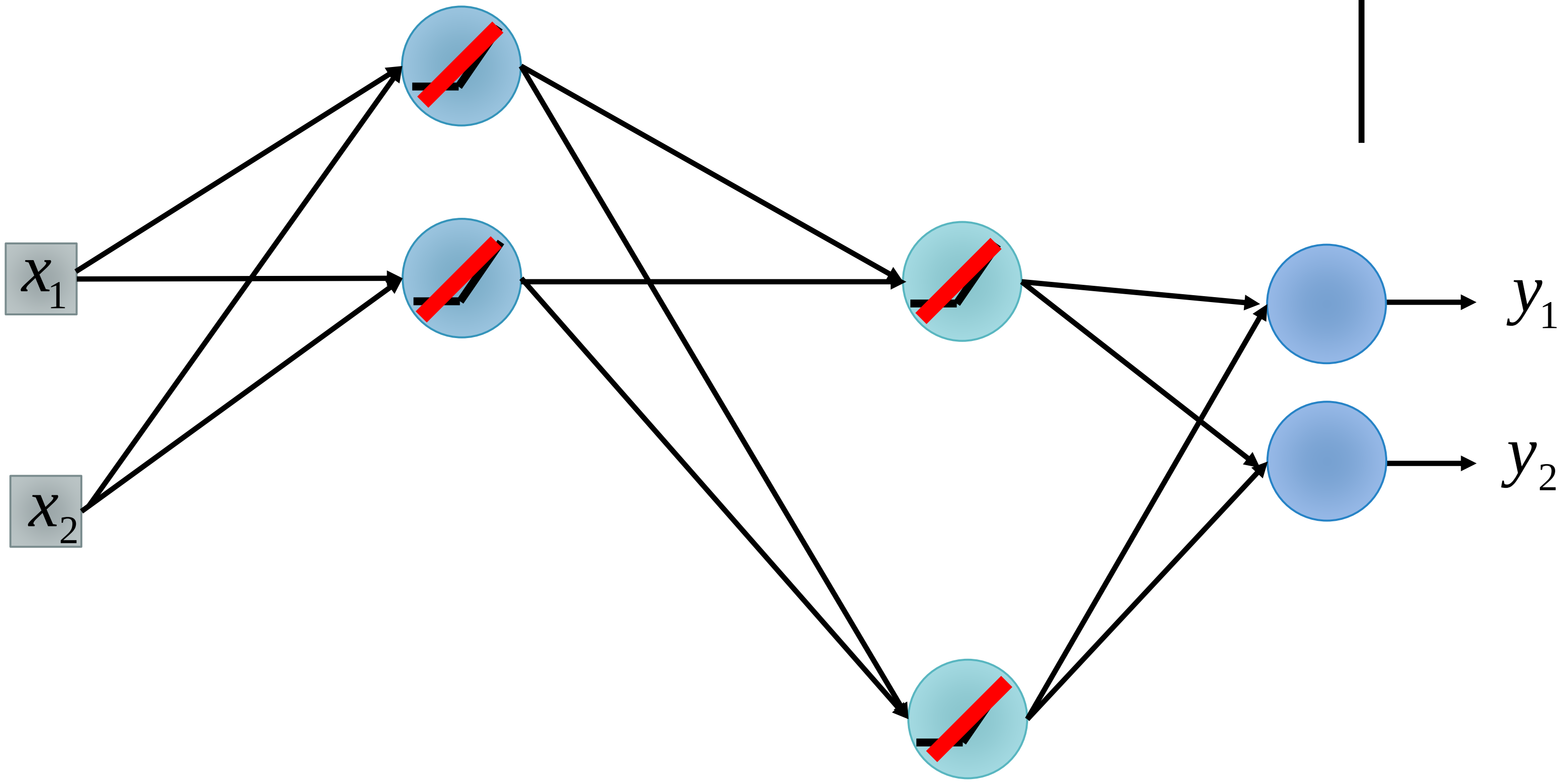
1. Fast to compute
2. Biological reason
3. Vanishing gradient problem

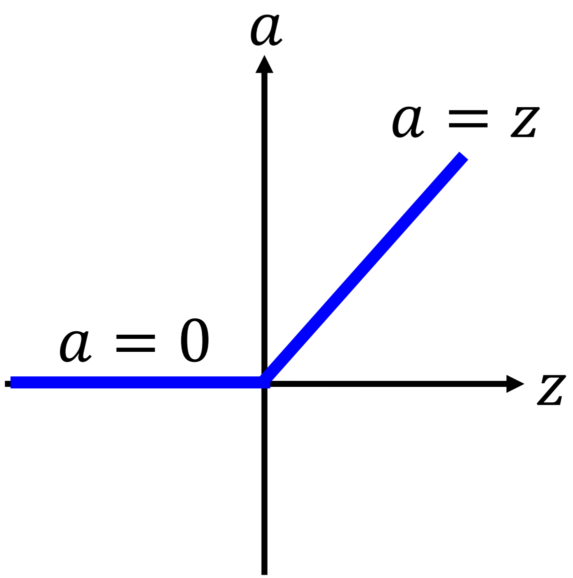
RELU



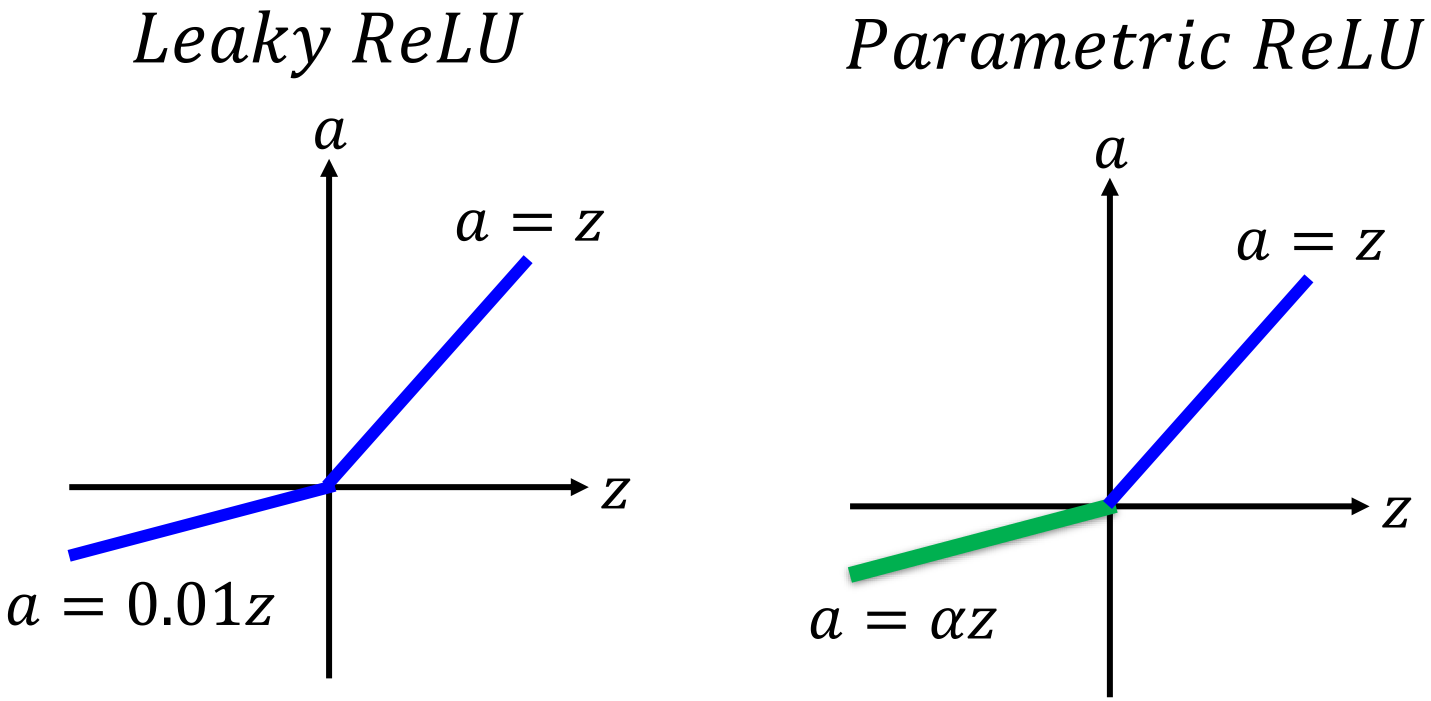
RELU

A Thinner linear network



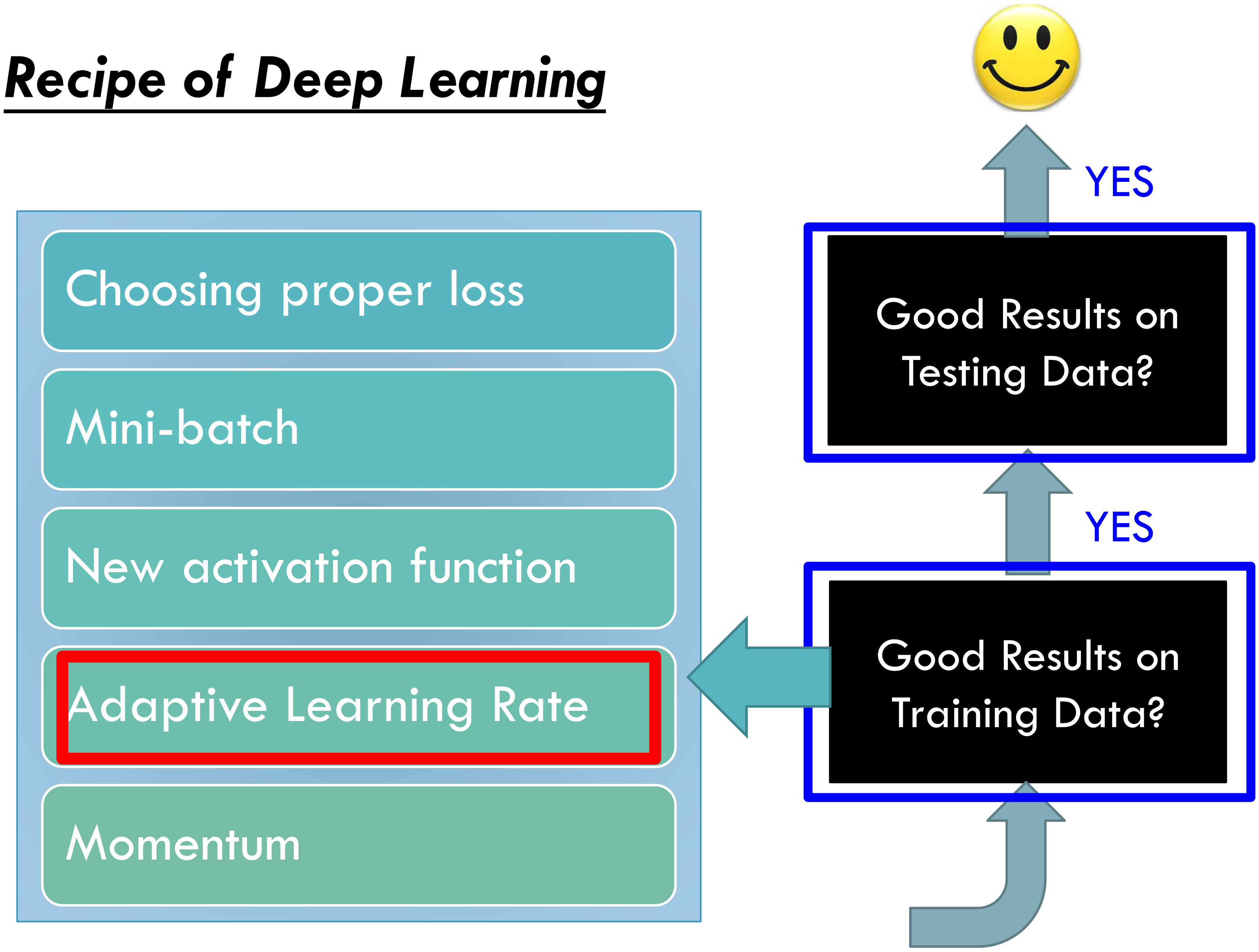


RELU - VARIANT

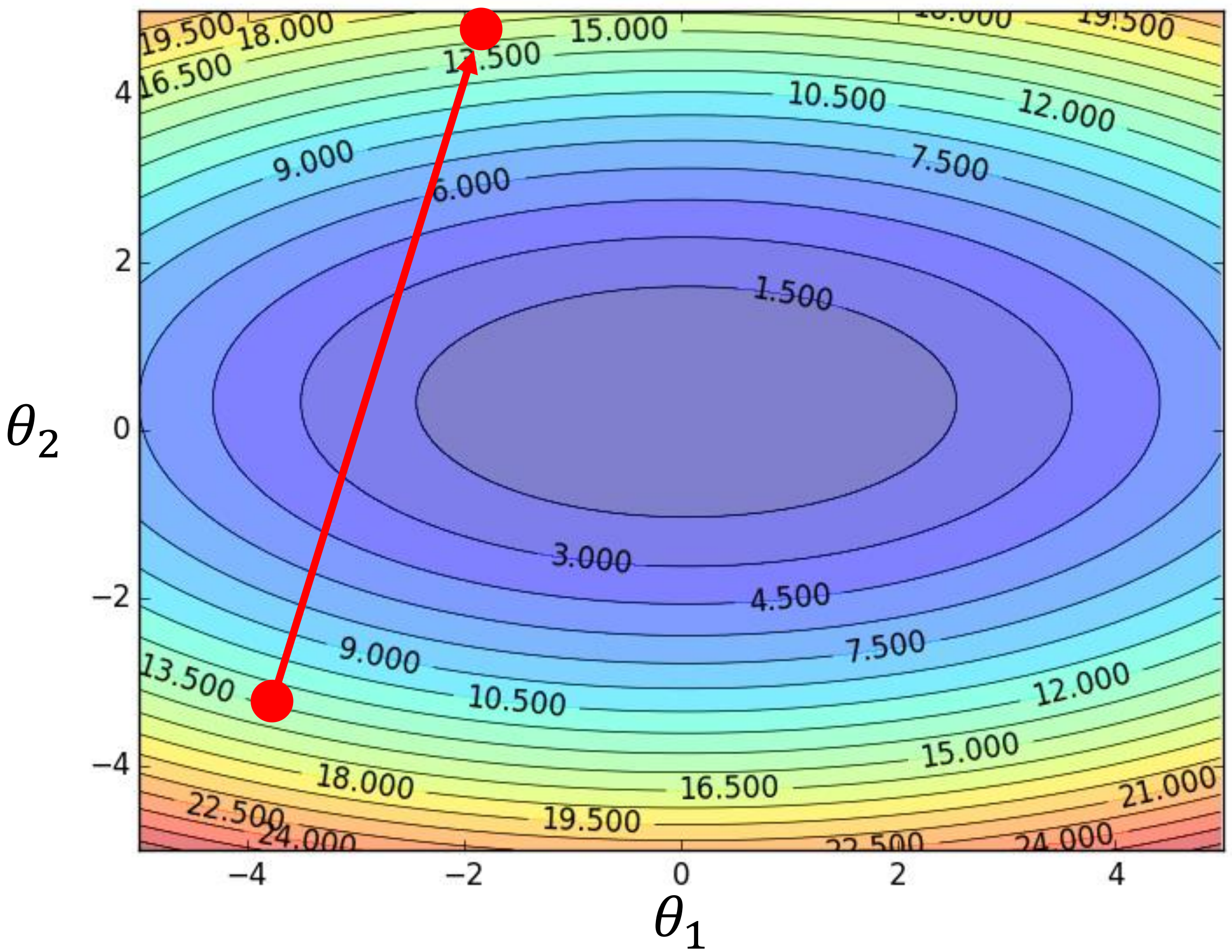


α also learned by gradient descent

Recipe of Deep Learning



LEARNING RATES

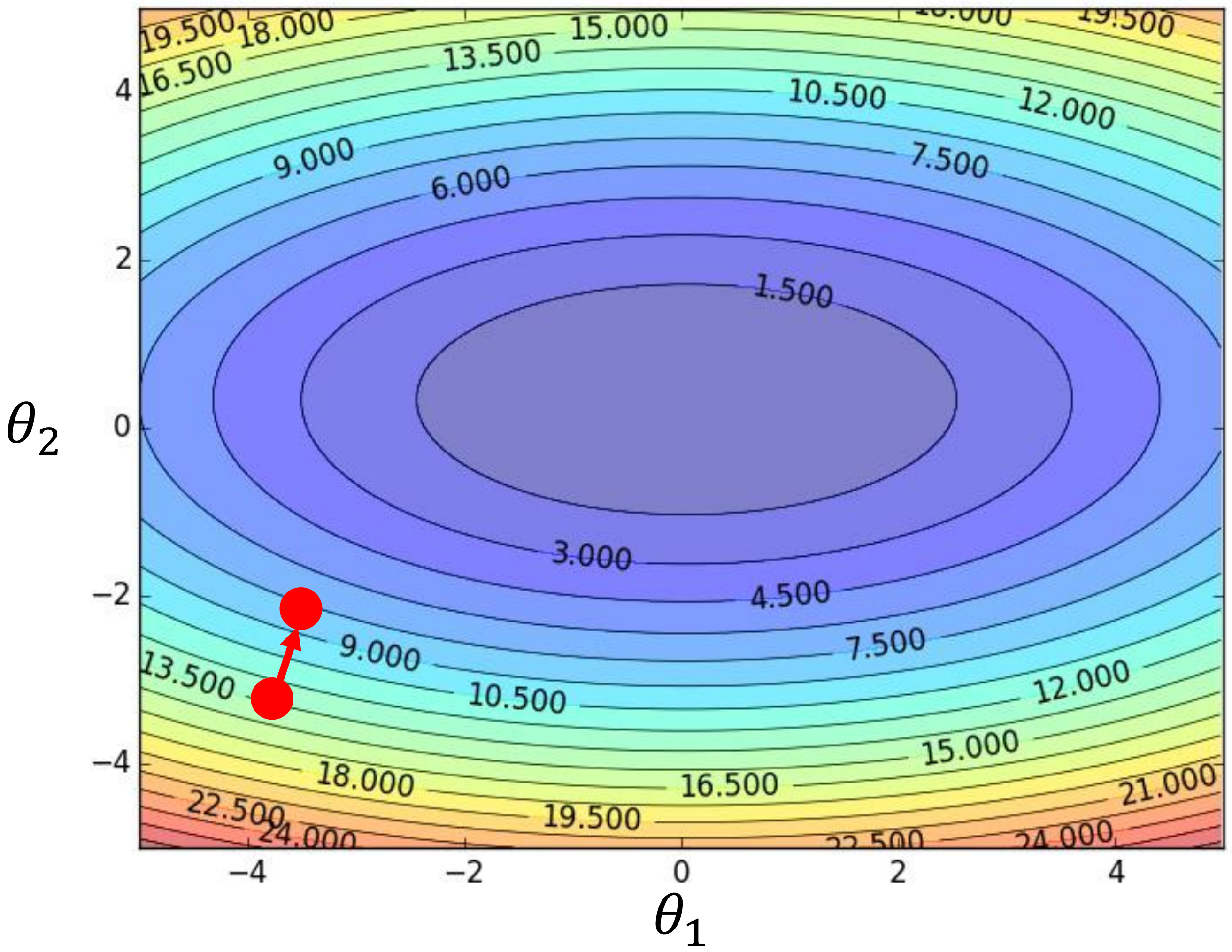


Set the learning rate α carefully

If learning rate is too large

Total loss may not decrease after each update

LEARNING RATES



Set the learning rate α 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. $\frac{1}{t}$ decay: $\alpha^t = \frac{\alpha}{\sqrt{t+1}}$
- Learning rate cannot be one-size-fits-all
  - Giving different parameters different learning rates

ADAGRAD

Original: $\theta \leftarrow \theta - \alpha \partial L \,/\, \partial\theta$

Adagrad: $\theta \leftarrow \theta - \boxed{\alpha_\theta} \partial L \,/\, \partial\theta$

Parameter-dependent learning rate

$$\alpha_\theta = \frac{\alpha}{\sqrt{\boxed{\sum_{i=0}^{t} (g^i)^2}}}$$

constant

g^i is $\partial L \,/\, \partial\theta$ obtained at the i^{th} update

Summation of the square of the previous derivatives

ADAGRAD

$$\alpha_\theta = \boxed{\frac{\alpha}{\sqrt{\sum_{i=0}^{t}(g^i)^2}}}$$

	g^0
θ_1	0.1

	g^0
θ_2	20.0

Learning rate:

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

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

Learning rate:

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

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

Observation:

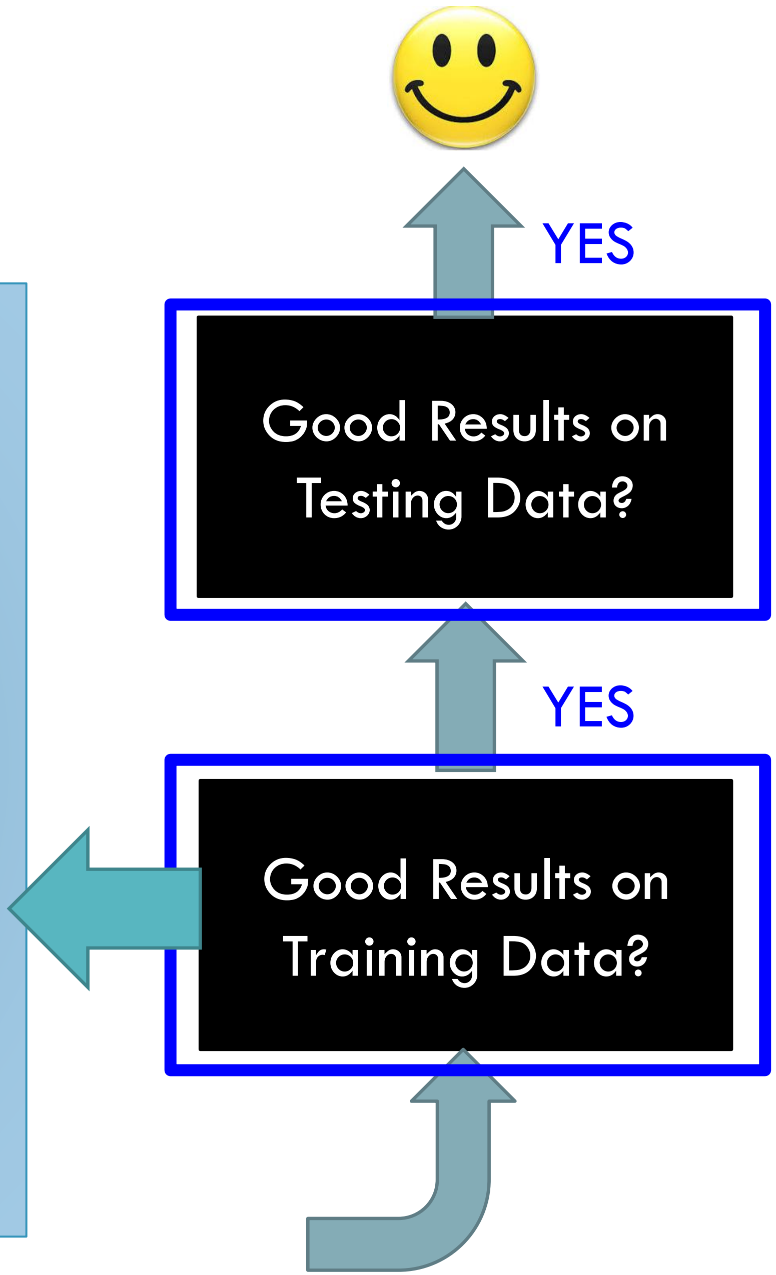
1. Learning rate is smaller and smaller for all parameters
2. Smaller derivatives, larger learning rate, and vice versa

Very useful tutorial on an overview of gradient descent optimization algorithms

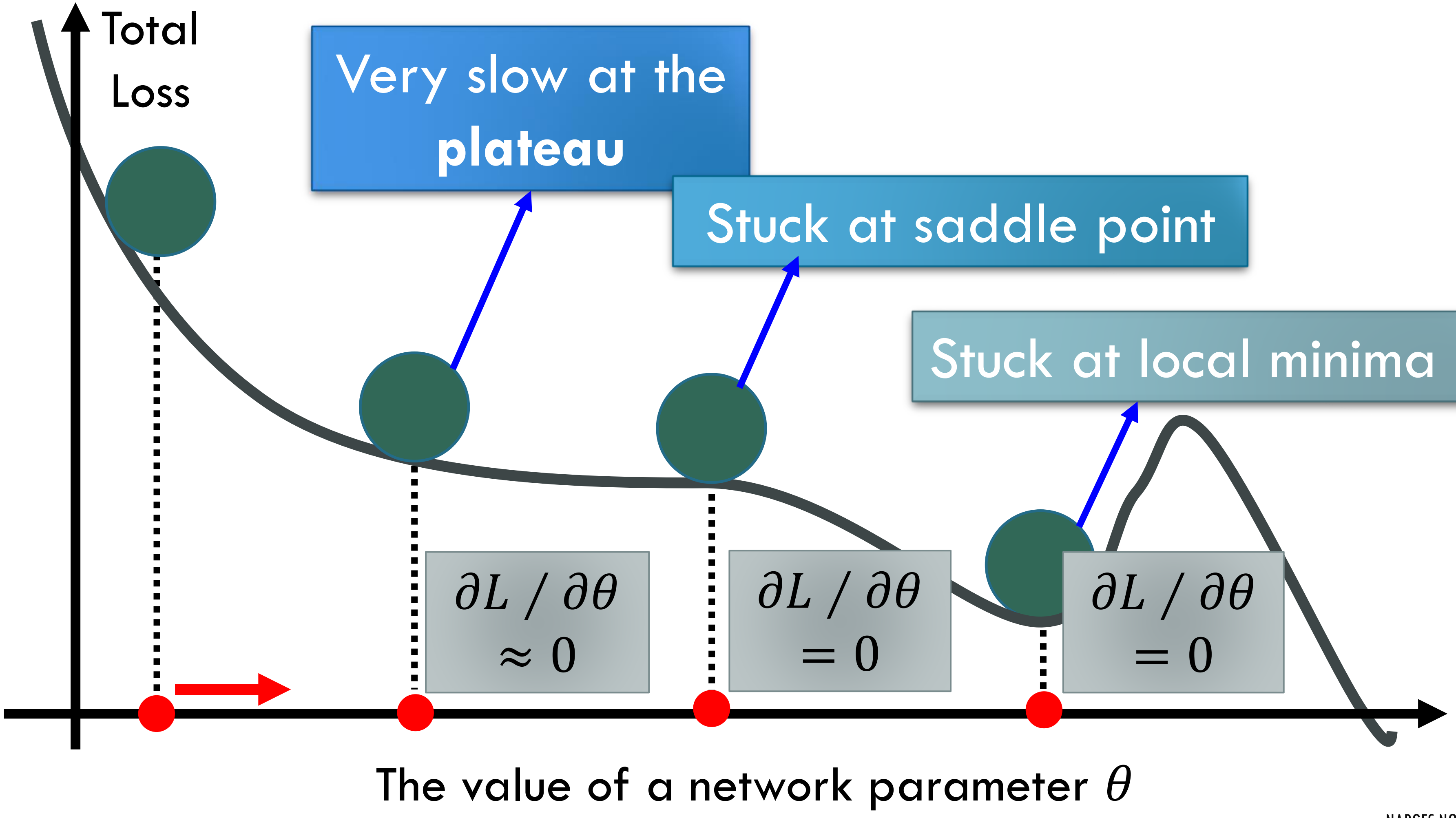
https://ruder.io/optimizing-gradient-descent/

Recipe of Deep Learning

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

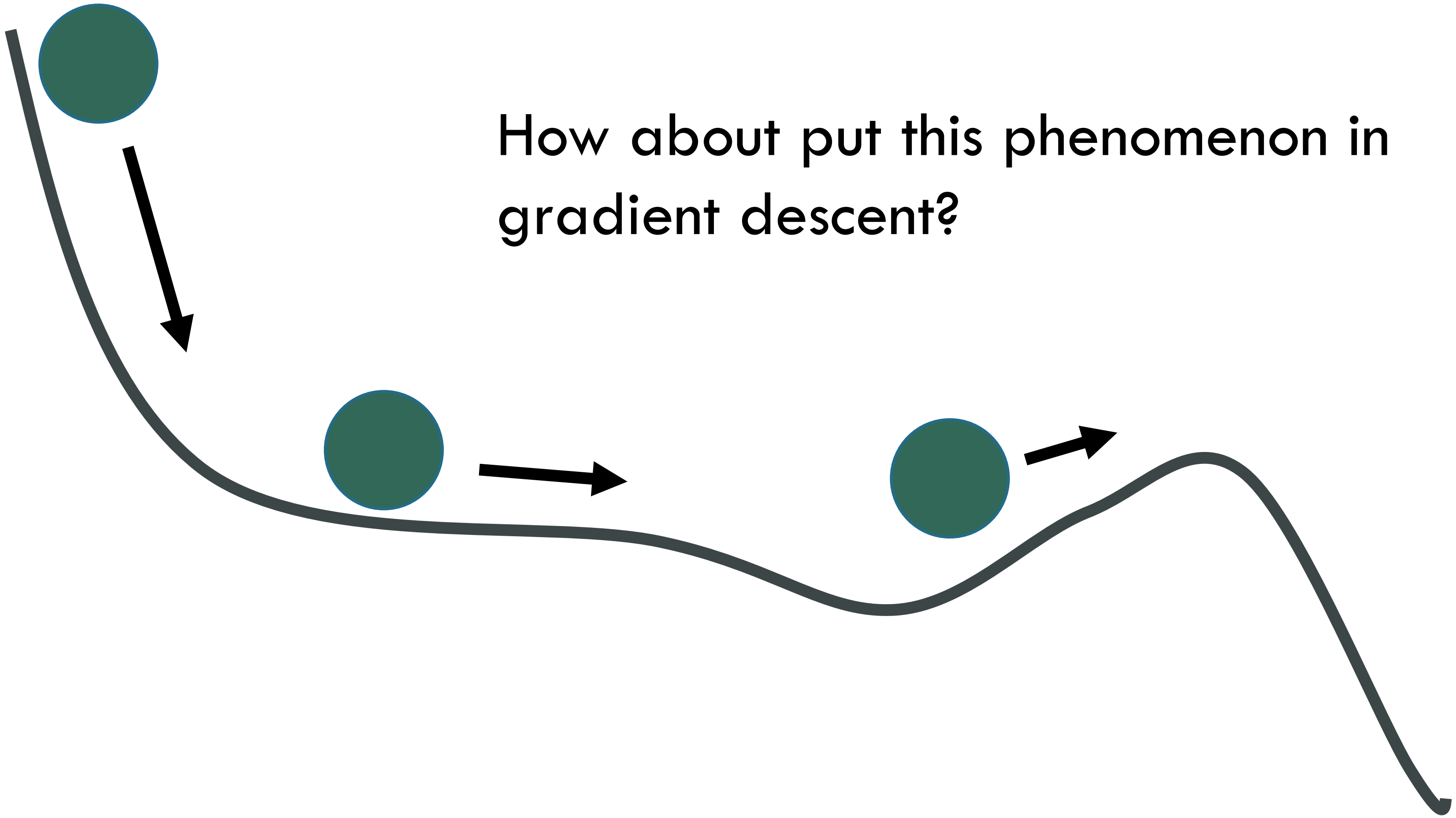


HARD TO FIND
OPTIMAL NETWORK PARAMETERS



IN PHYSICAL WORLD ……

- Momentum



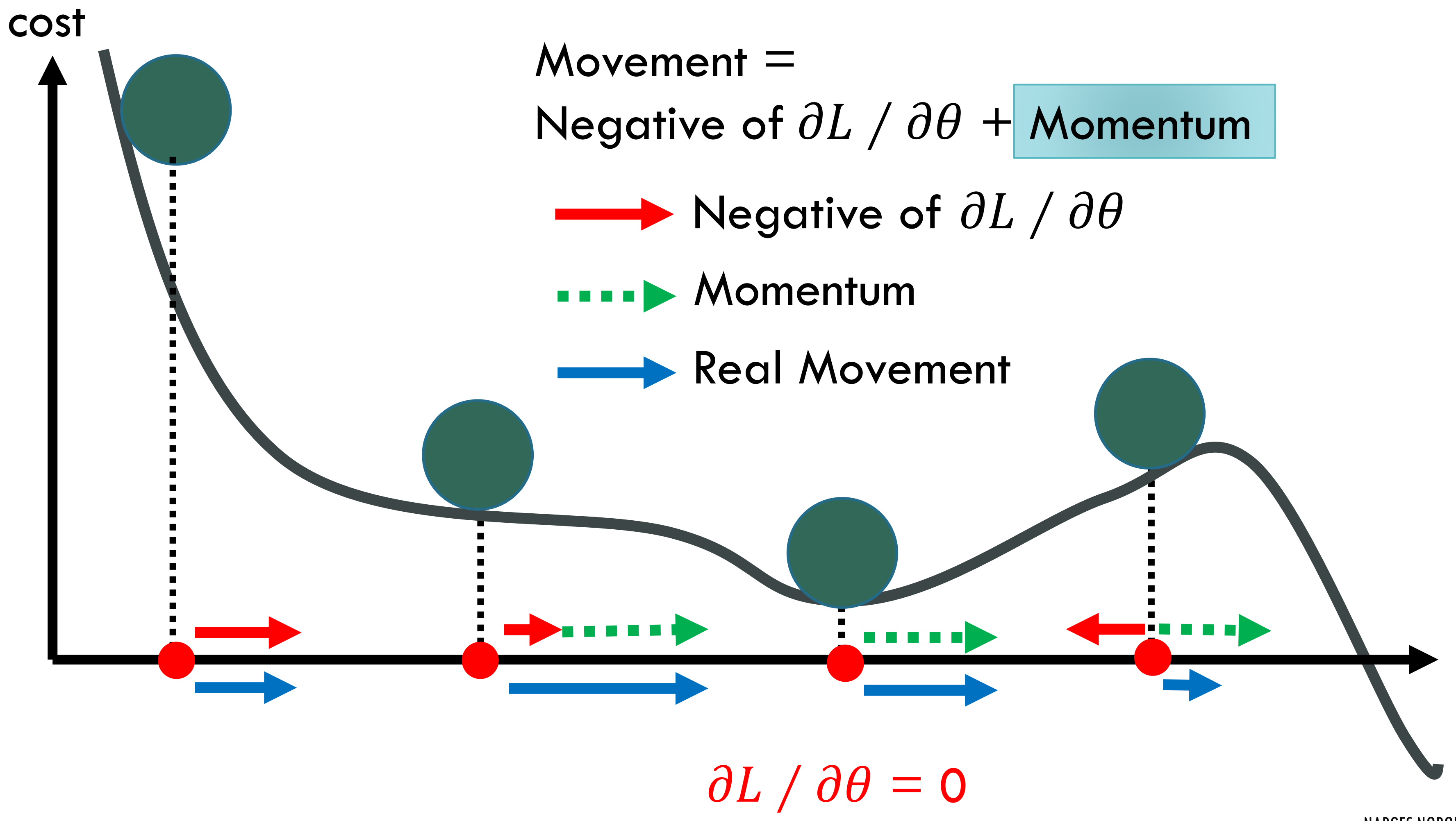
How about put this phenomenon in gradient descent?

MOMENTUM

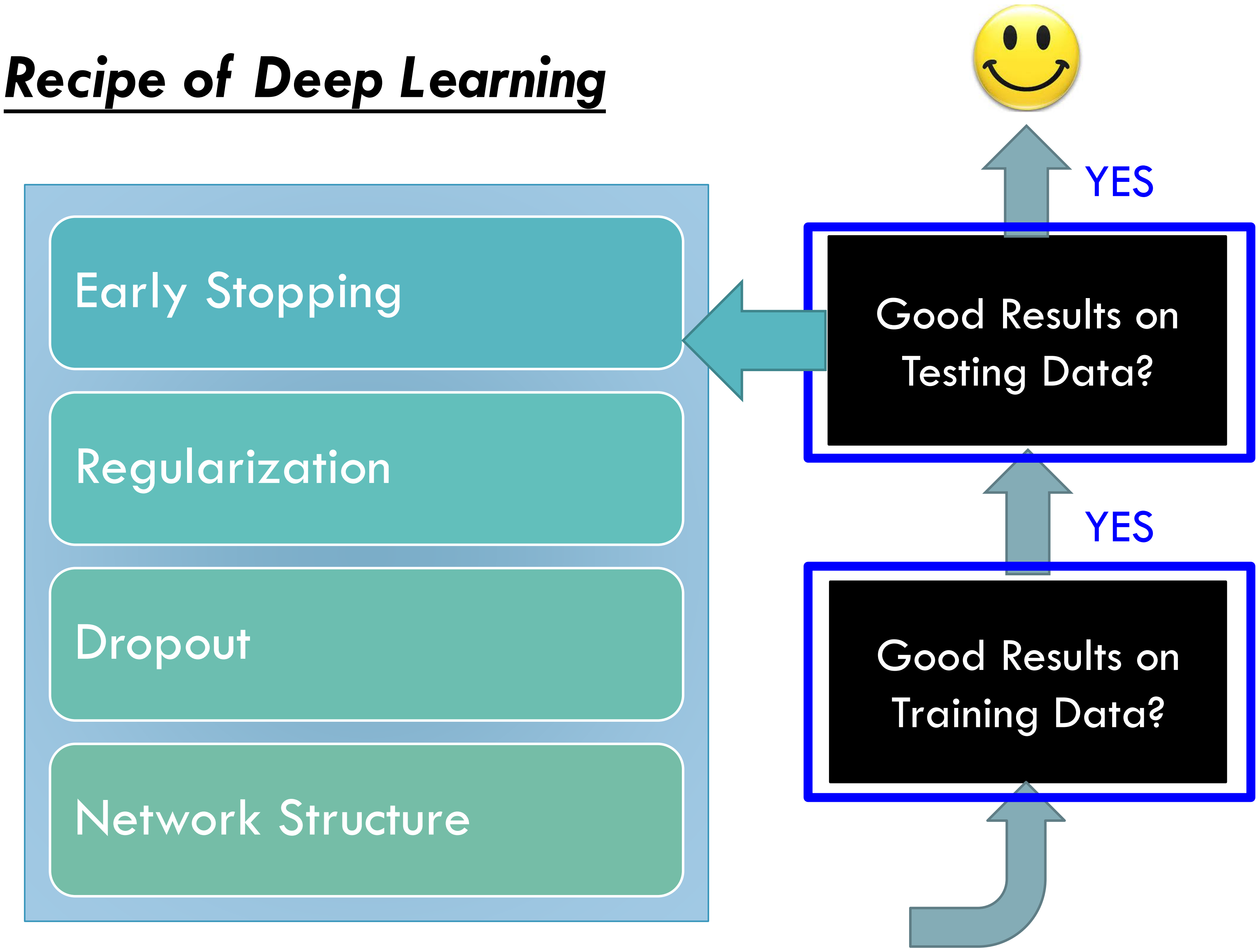
Still not guarantee reaching global minima, but give some hope ……

Movement =
Negative of $\partial L / \partial \theta$ + $\boxed{\text{Momentum}}$

- Negative of $\partial L / \partial \theta$
- Momentum
- Real Movement



Recipe of Deep Learning

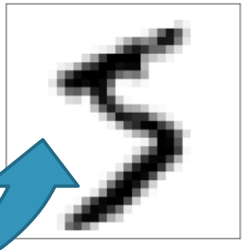


AVOIDING OVERFITTING

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

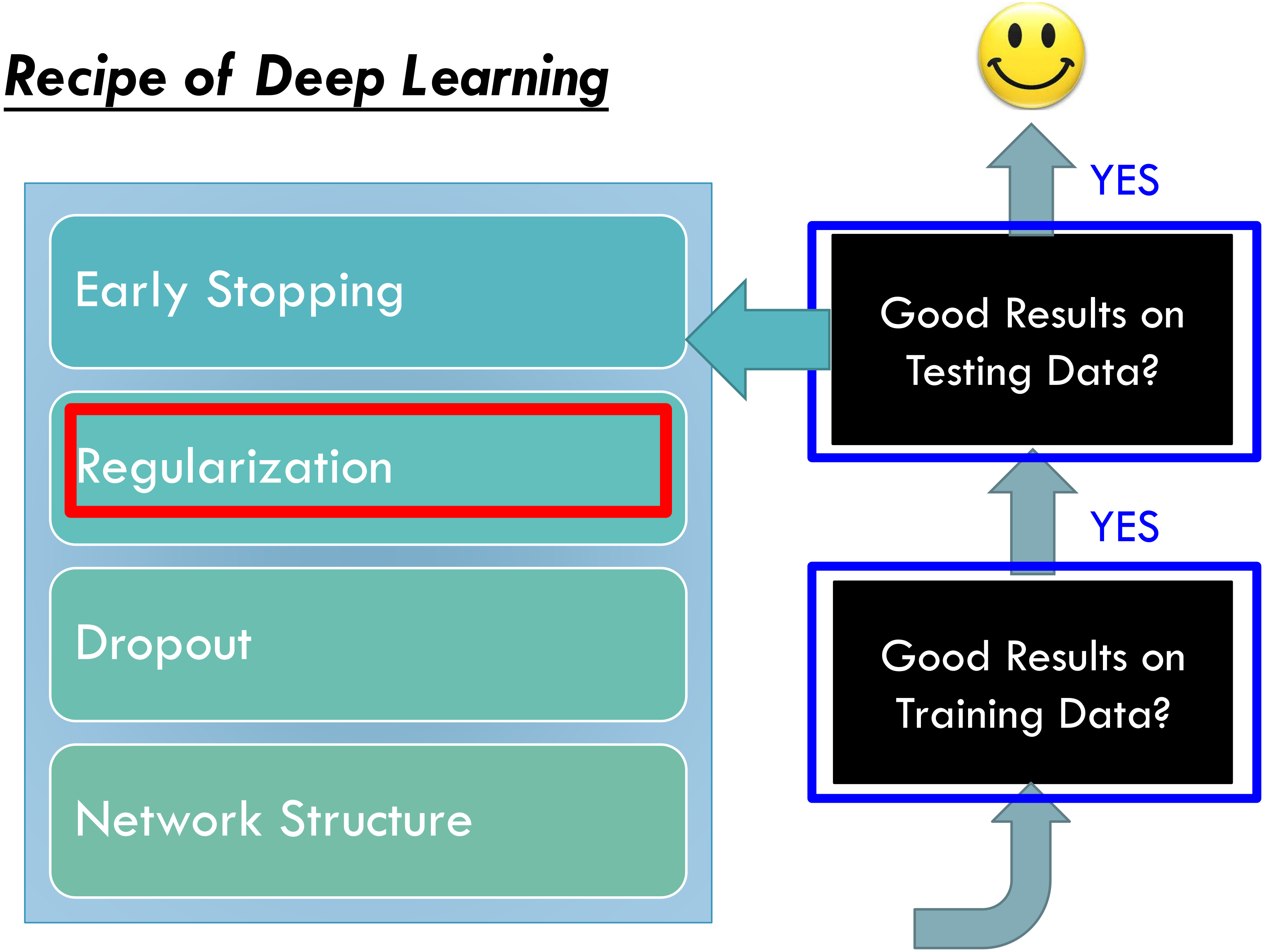
Handwriting recognition:

Original Training Data: 

Created Training Data: 

Shift 15 °

Recipe of Deep Learning



OVERFITTING REVISED: REGULARIZATION

- A regularizer is an additional criteria to the loss function to make sure that we don't overfit
- It's called a regularizer since it tries to keep the parameters more normal/regular
- It is a bias on the model forces the learning to prefer certain types of weights over others

$$TrainLoss(\theta) = \frac{1}{|D_{train}|} \sum_{(x,y)\in D_{train}} Loss(x, y, \theta)$$

$$\min_{\theta\in\mathbb{R}^d} TrainLoss(\theta) + \lambda\, regularizer(\theta)$$

COMMON REGULARIZERS

- Sum of the weights

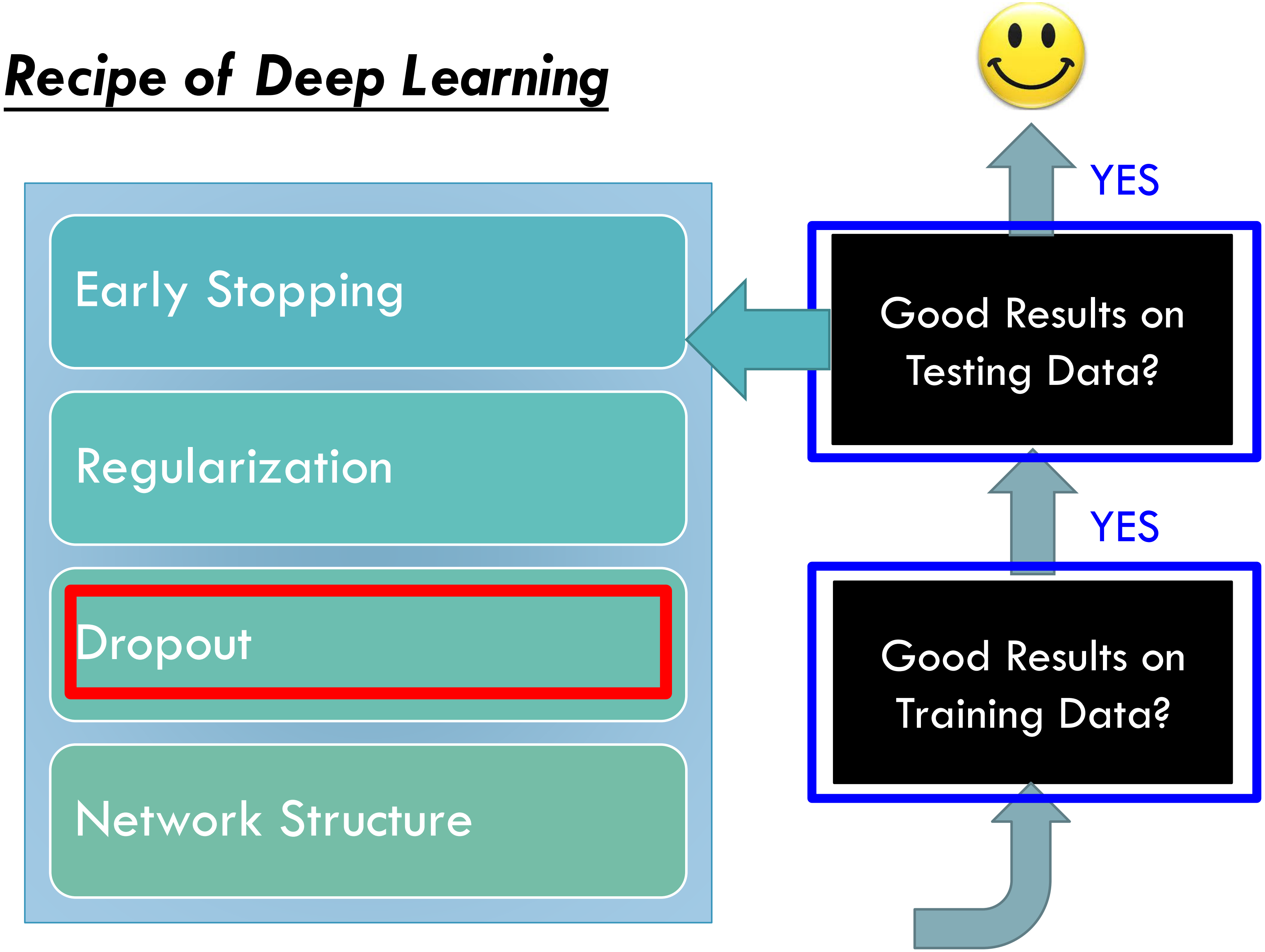
$$r(\theta) = \sum_{w_j} |\theta_j|$$

- Sum of the squared weights

$$r(\theta) = \sqrt{\sum_{\theta_j} \left|\theta_j\right|^2}$$

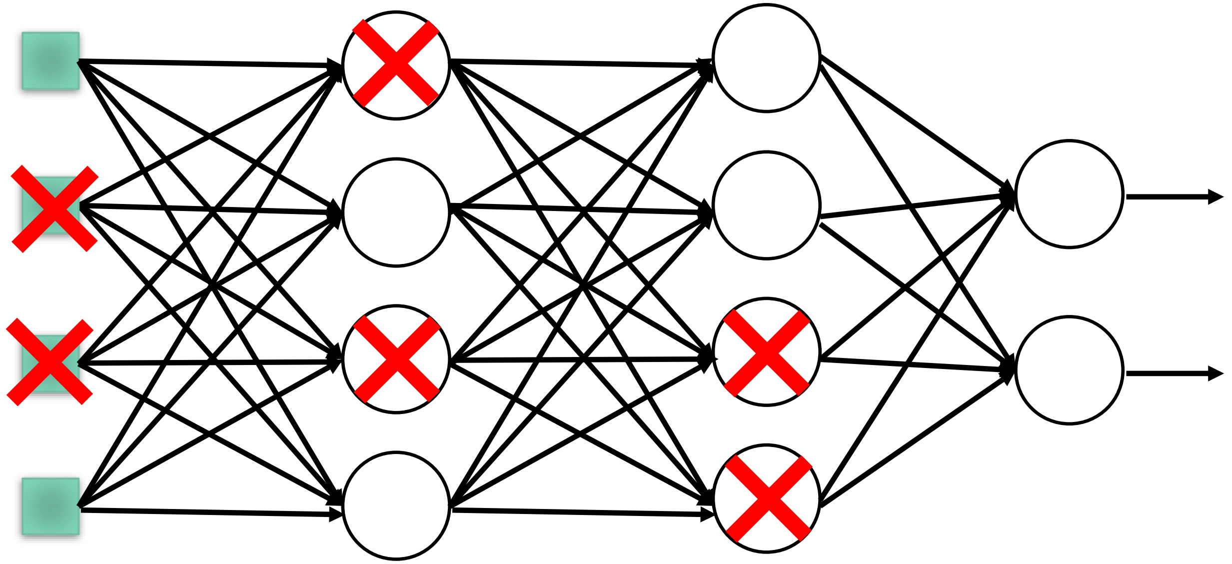
Squared weights penalizes large values more.

Recipe of Deep Learning



DROPOUT

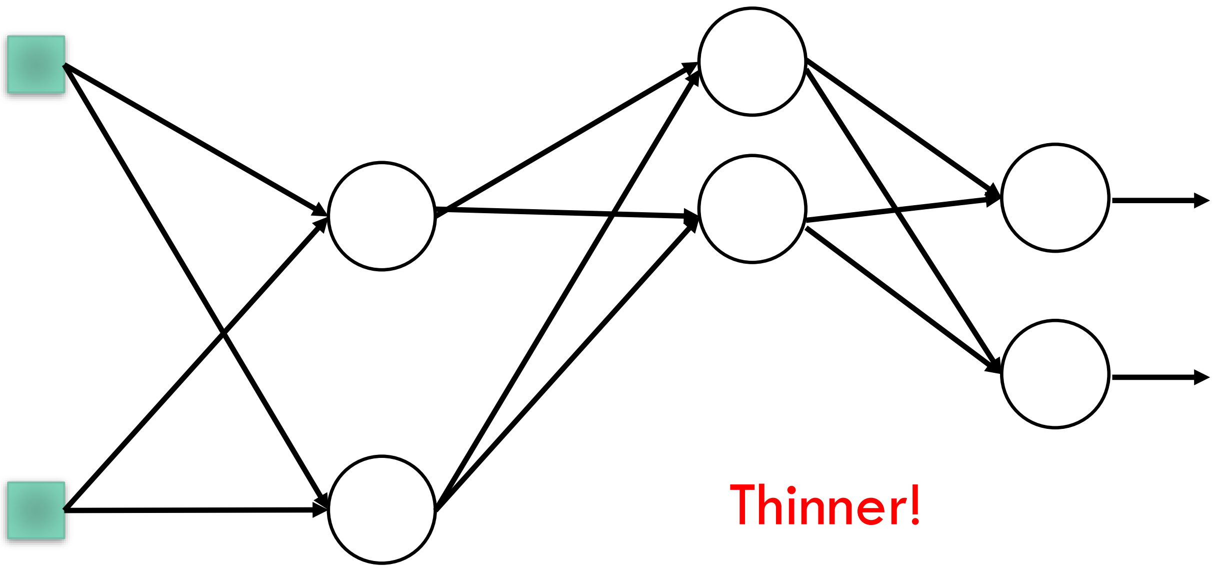
Training:



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

DROPOUT

Training:

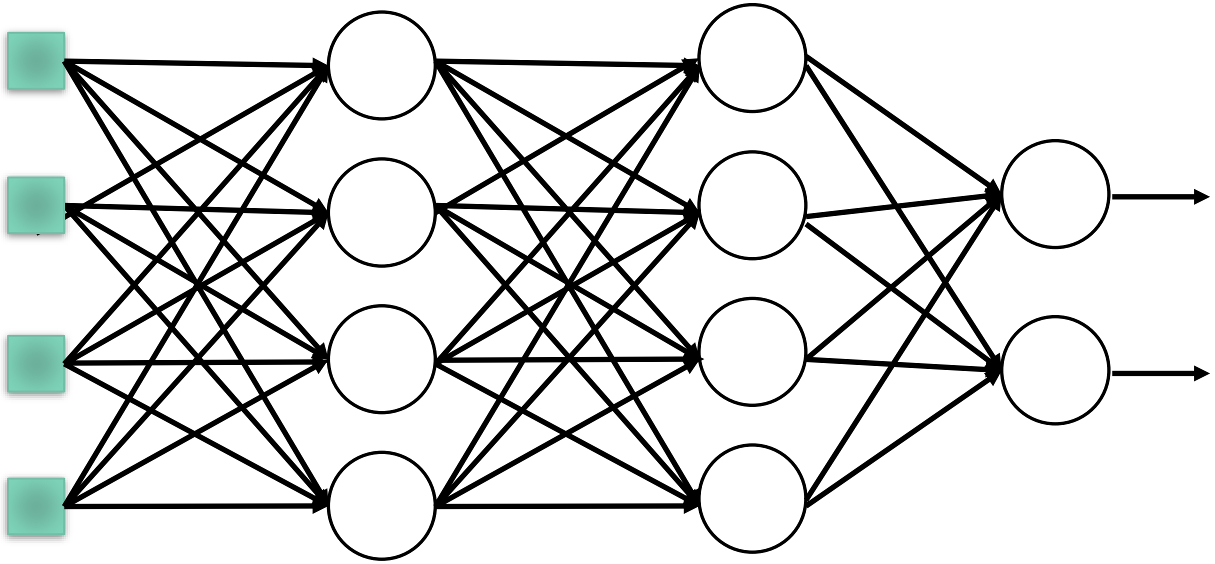


- **Each time before updating the parameters**
  - Each neuron has $p\%$ to dropout
    - **The structure of the network is changed.**
  - Using the new network for training

For each mini-batch, we resample the dropout neurons

DROPOUT

Testing:



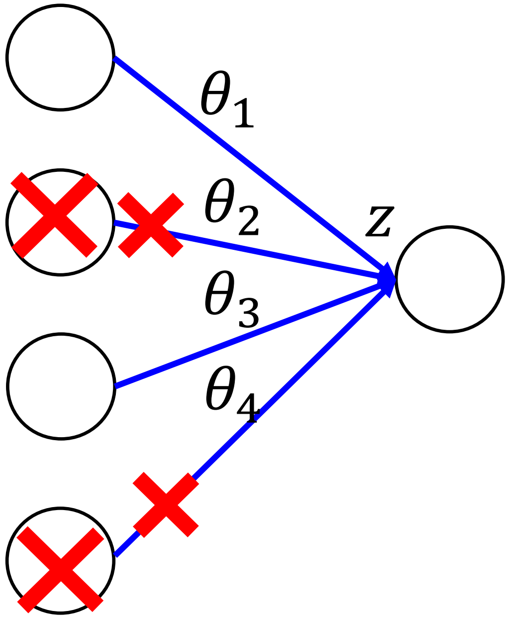
- **No dropout**
  - If the dropout rate at training is $p\%$, all the weights times $1 - p\%$
  - Assume that the dropout rate is 50%. If a weight $\theta = 1$ by training, set $\theta = 0.5$ for testing.

DROPOUT - INTUITIVE REASON

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

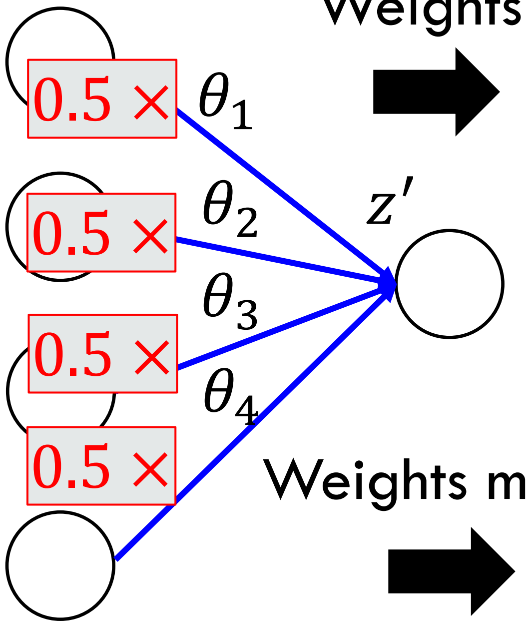
Training of Dropout

Assume dropout rate is 50%



Testing of Dropout

No dropout



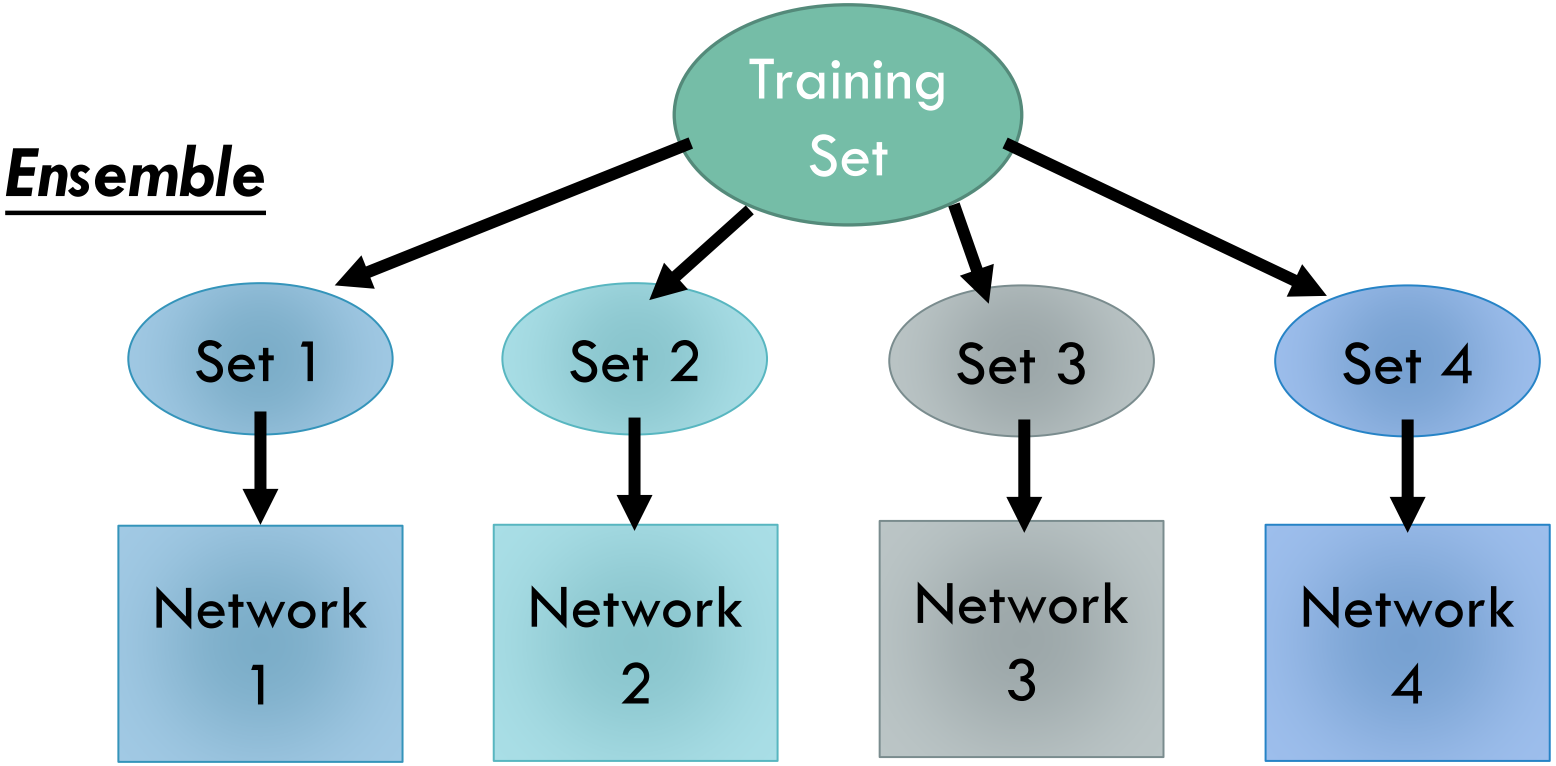
Weights from training

$\Rightarrow z' \approx 2z$

Weights multiply $1-p\%$

$\Rightarrow z' \approx z$

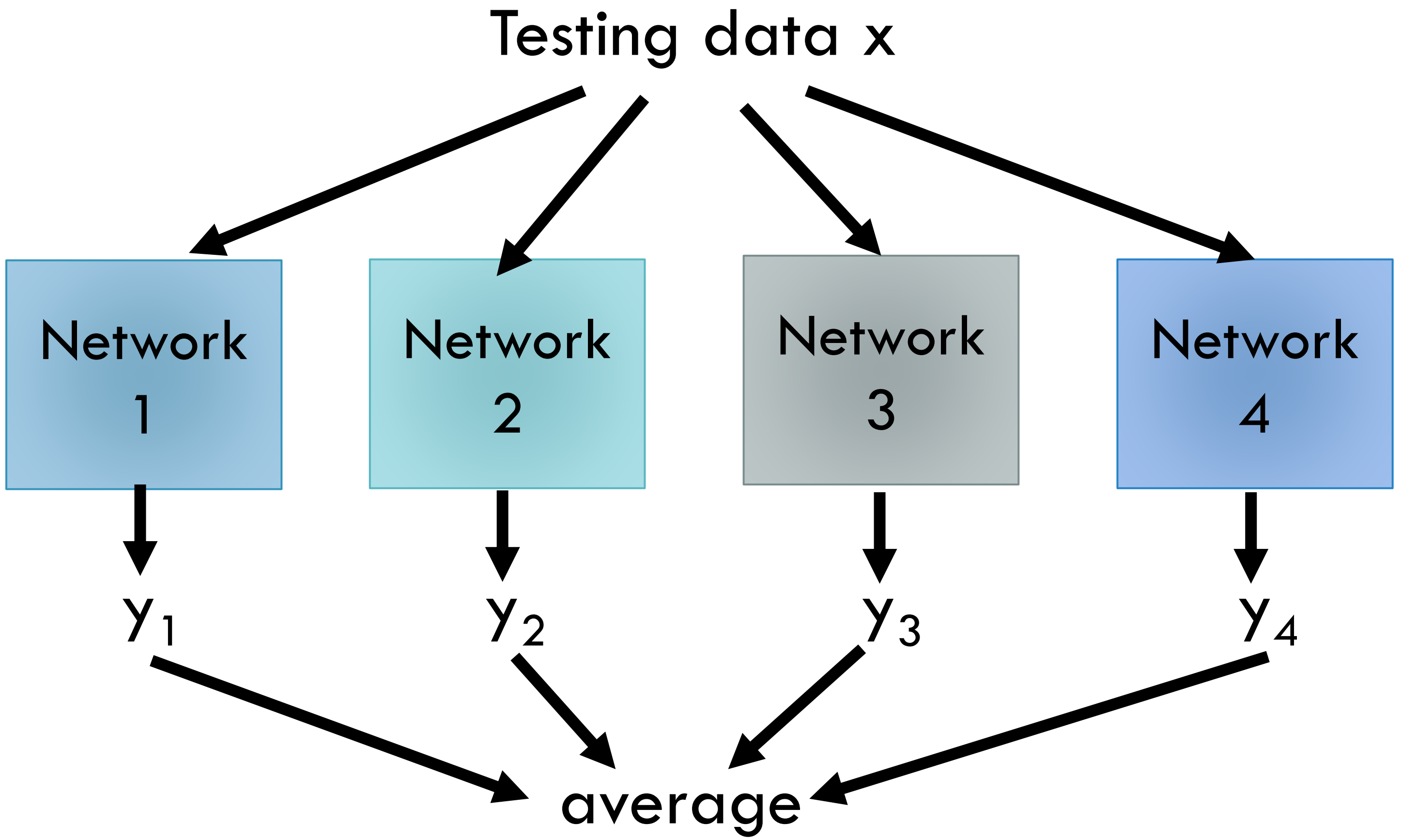
DROPOUT IS A KIND OF ENSEMBLE.



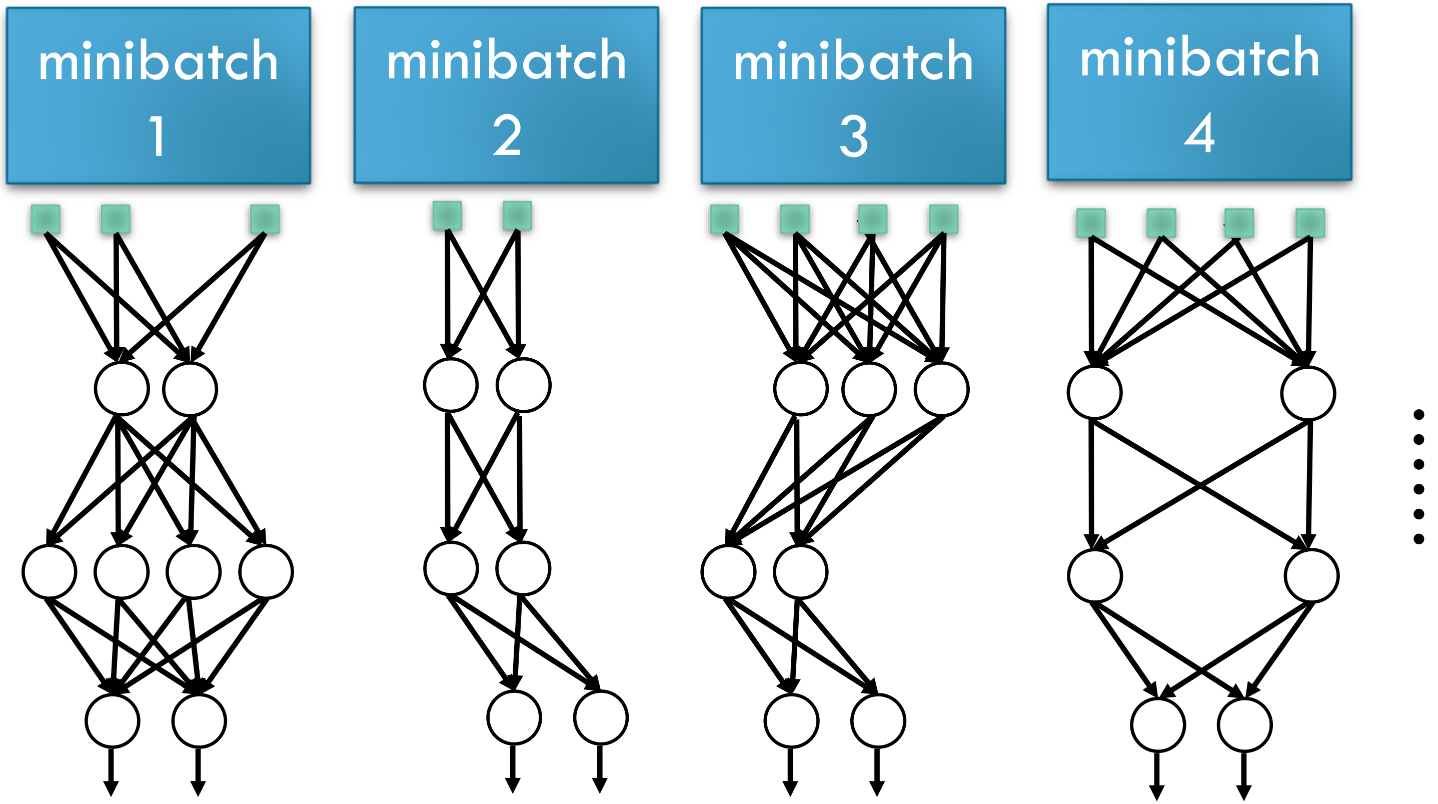
Train a bunch of networks with different structures

DROPOUT IS A KIND OF ENSEMBLE.

Ensemble



DROPOUT IS A KIND OF ENSEMBLE.



Training of Dropout

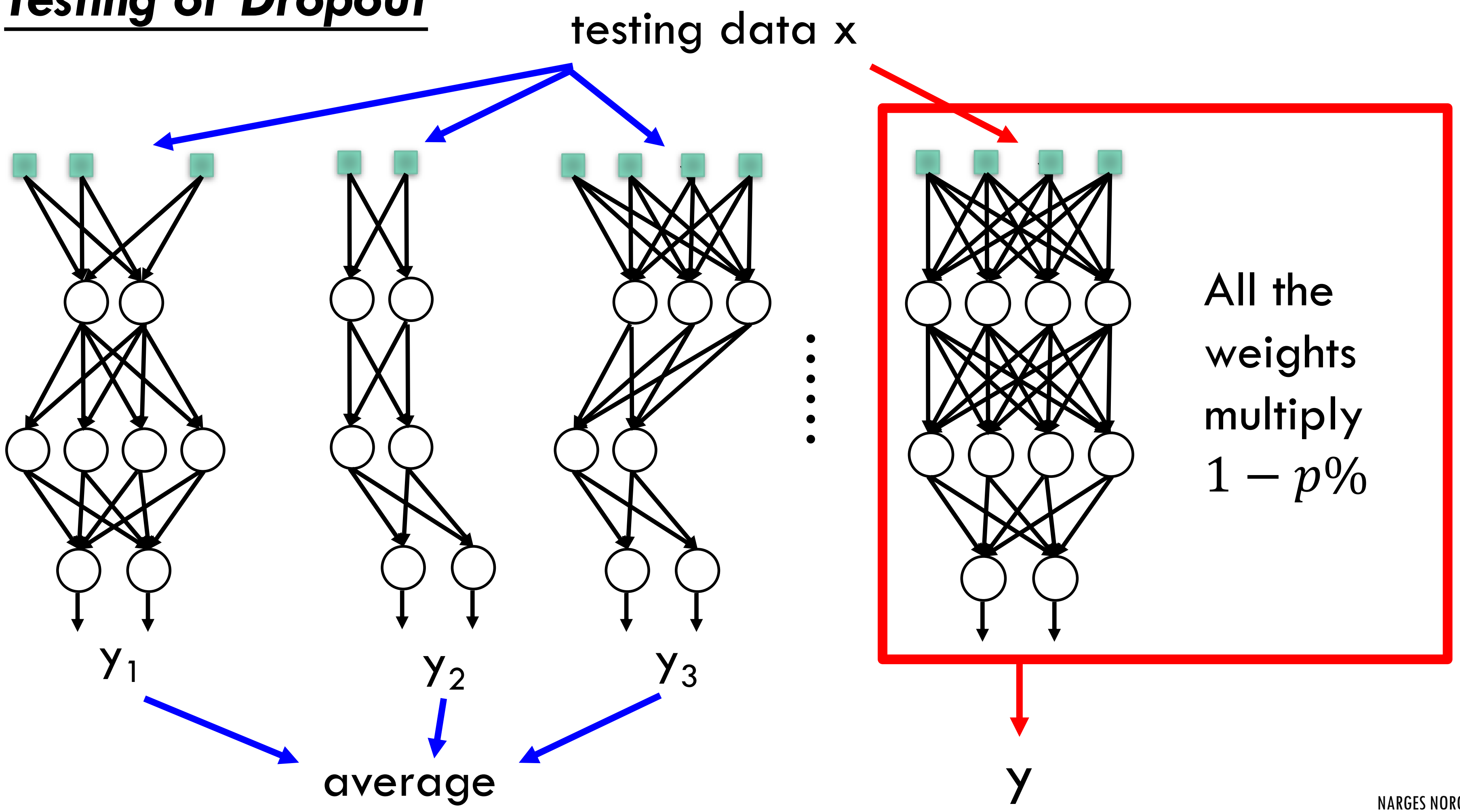
M neurons

2^M possible networks

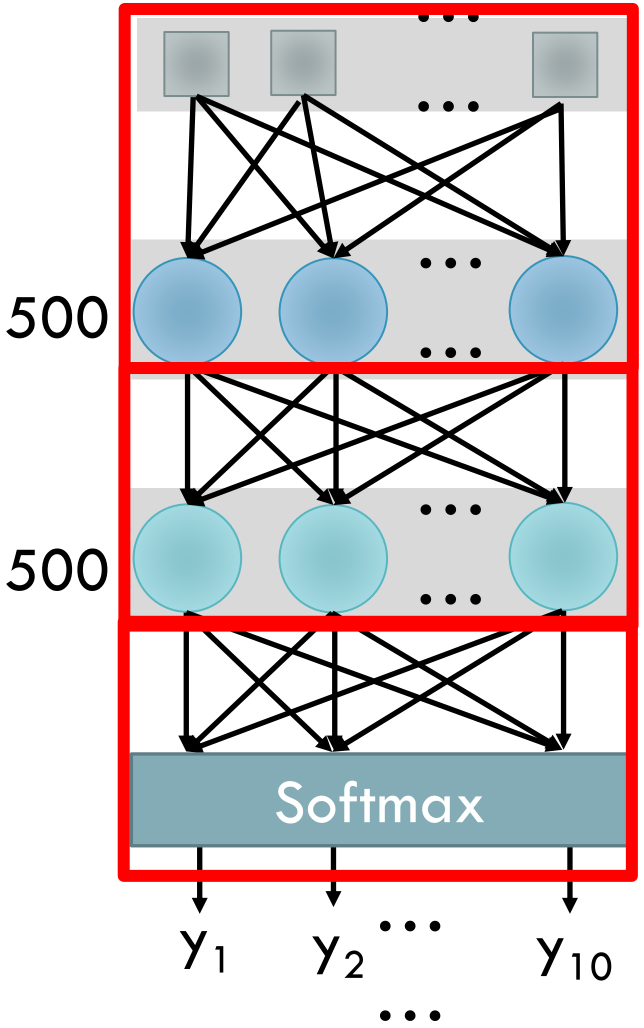
- ➢Using one mini-batch to train one network
- ➢Some parameters in the network are shared

DROPOUT IS A KIND OF ENSEMBLE.

Testing of Dropout



DEMO




```
model = Sequential()
```

```
model.add(Dense(units=500,
                input_dim=28*28,
                activation='sigmoid'))
```

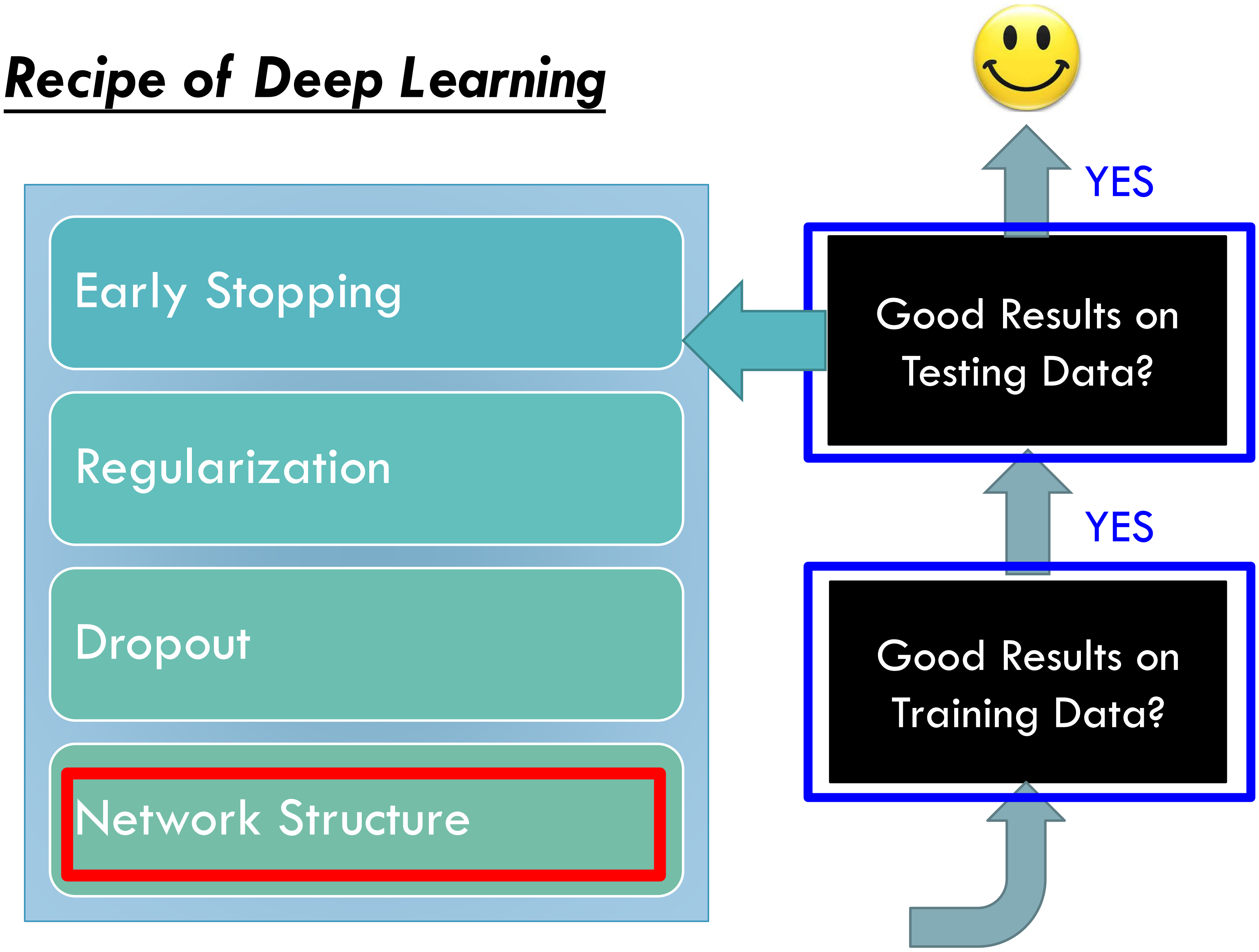
```
model.add(Dropout(0.8))
```

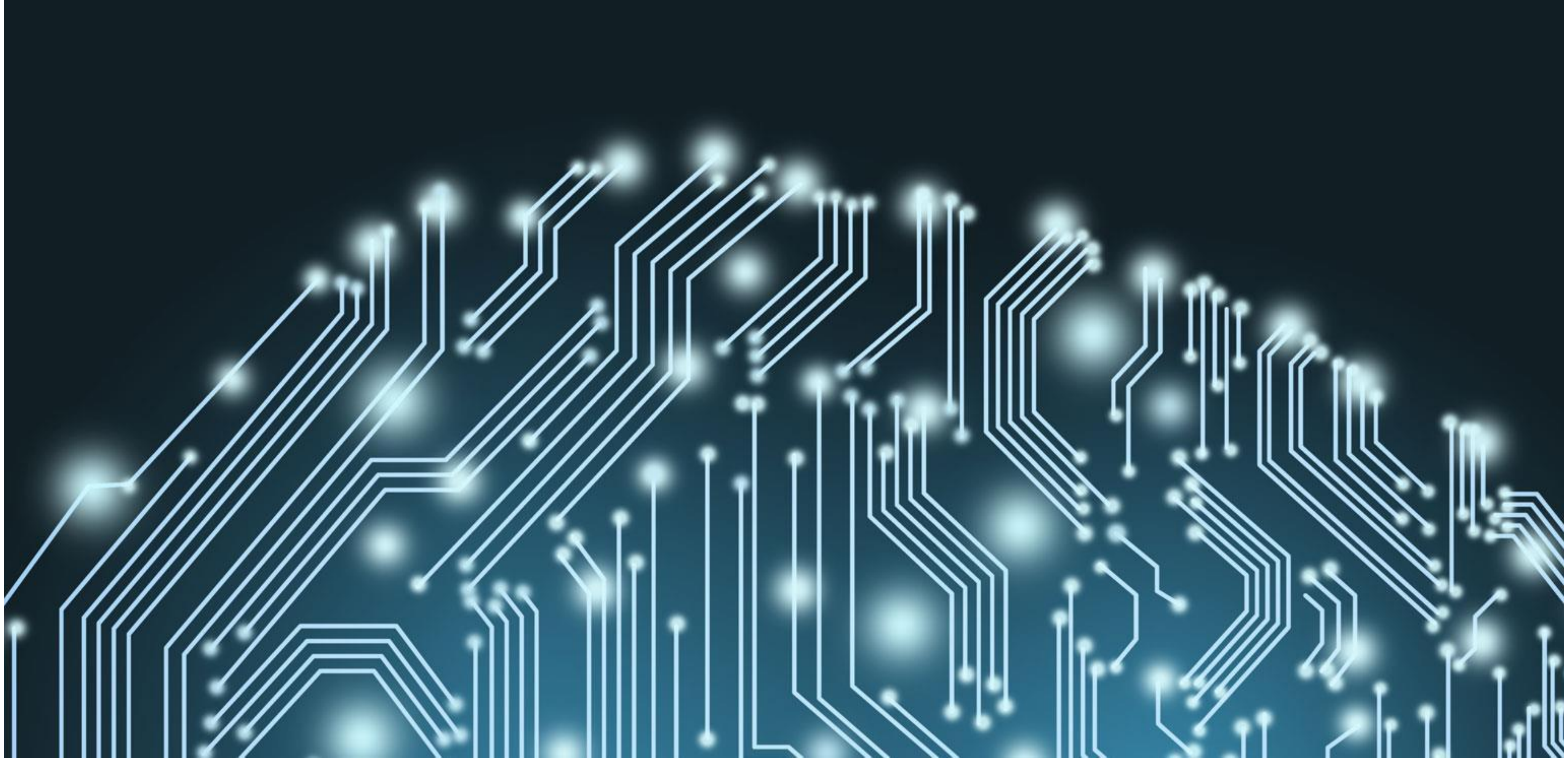
```
model.add(Dense(units=500,
                activation='sigmoid'))
```

```
model.add(Dropout(0.6))
```

```
model.add(Dense(units=10,
                activation='softmax'))
```

Recipe of Deep Learning





CNN ARCHITECTURE IS COMING NEXT...