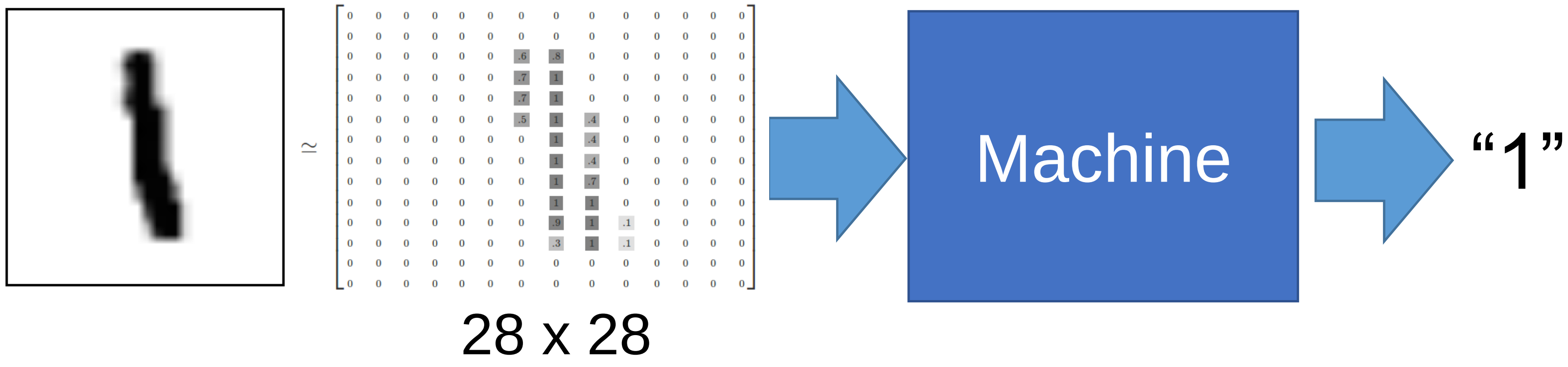
Deep learning
-- Hello world

Junjie Cao @ DLUT

Spring 2018

Example Application

- Handwriting Digit Recognition



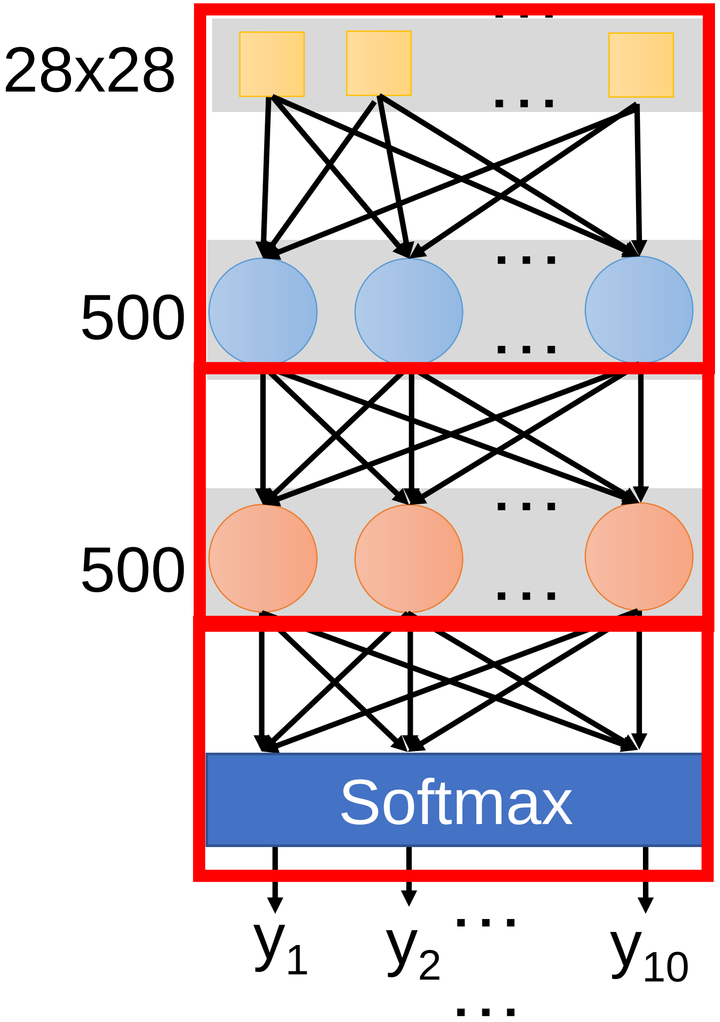
MNIST Data: http://yann.lecun.com/exdb/mnist/

“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




```
model = Sequential()
```

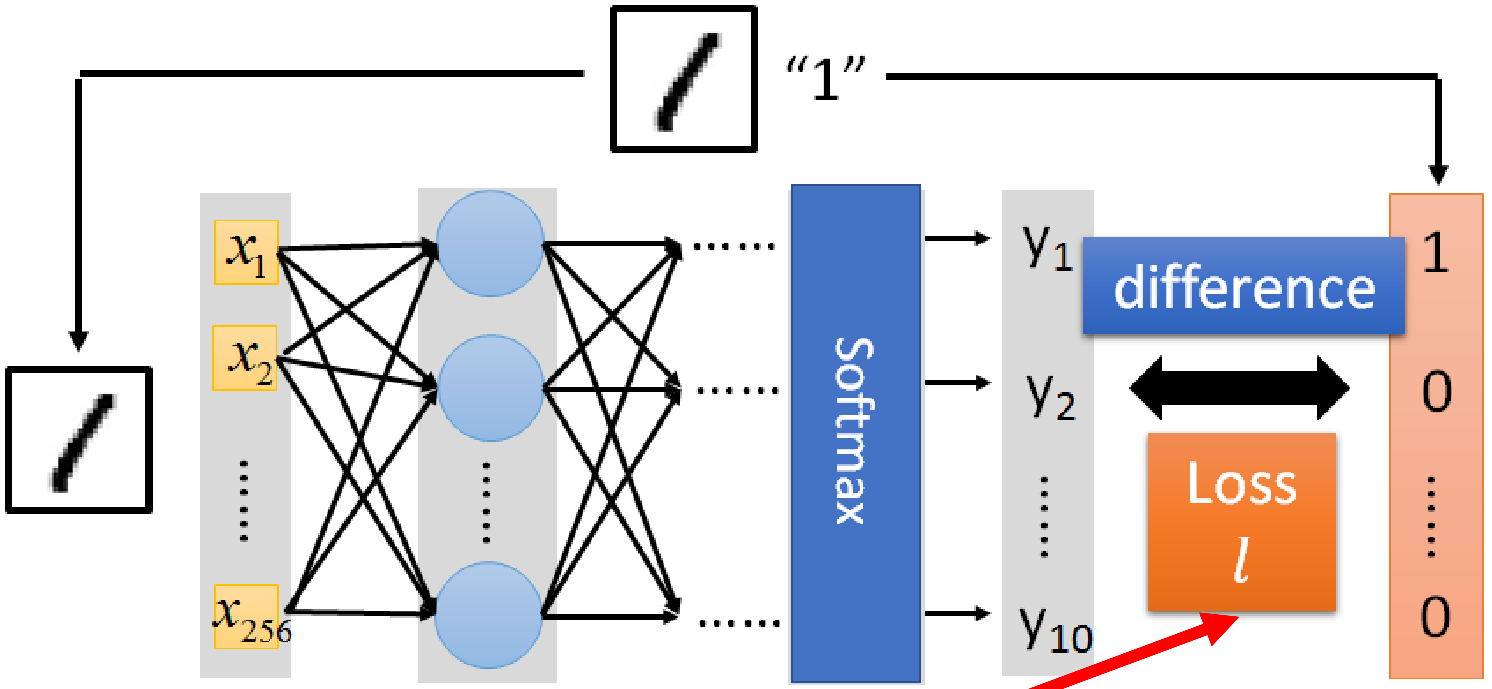
```
model.add( Dense( input_dim=28*28,
                  output_dim=500 ))
model.add( Activation('sigmoid') )
```

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

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

Keras

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




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

Keras

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

Step 3.1: Configuration

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

$$w \leftarrow w - \underbrace{\eta}_{0.1} \partial L / \partial w$$

Step 3.2: Find the optimal network parameters

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

Training data (Images)

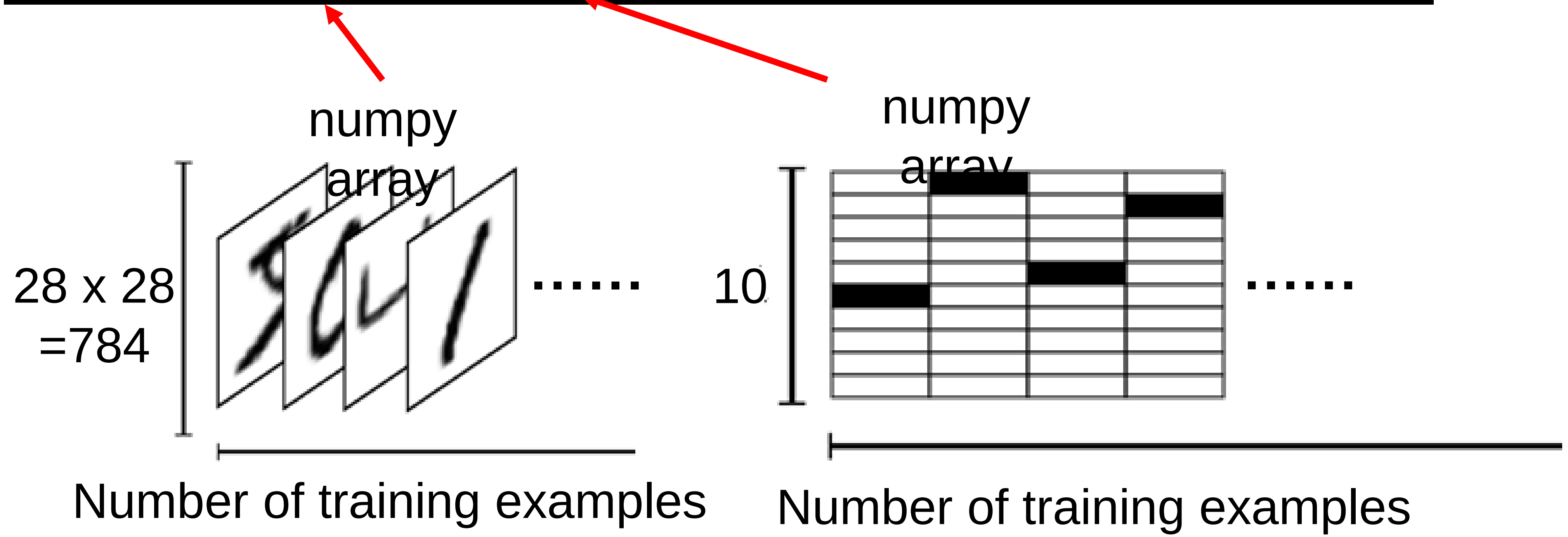
Labels (digits)

Keras

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

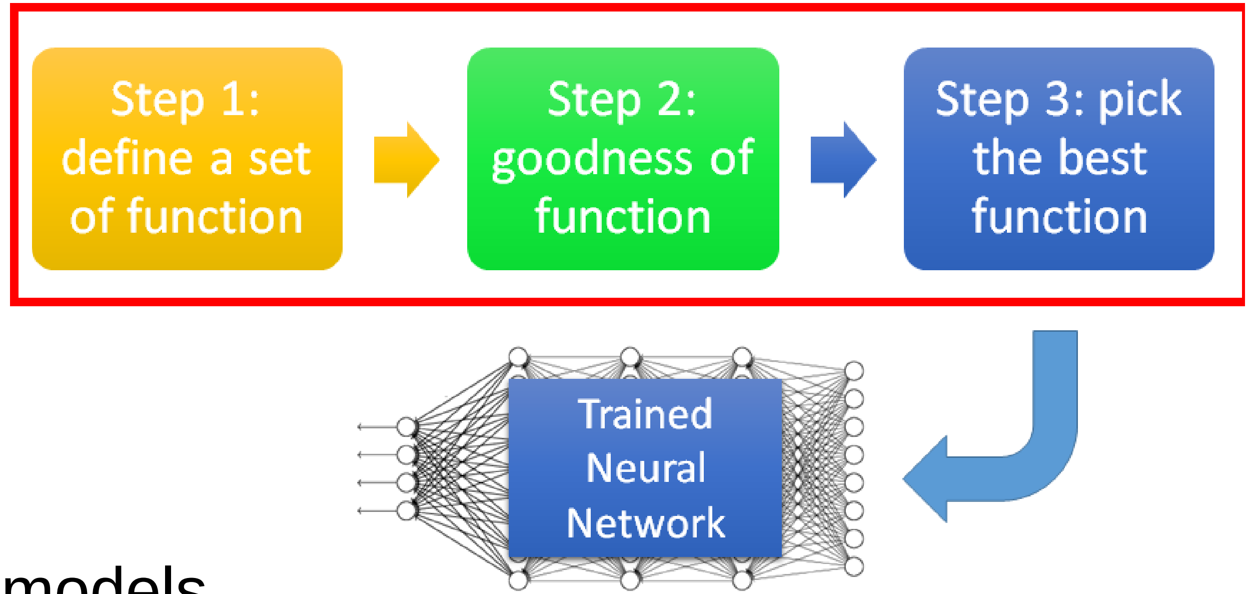
Step 3.2: Find the optimal network parameters

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



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):

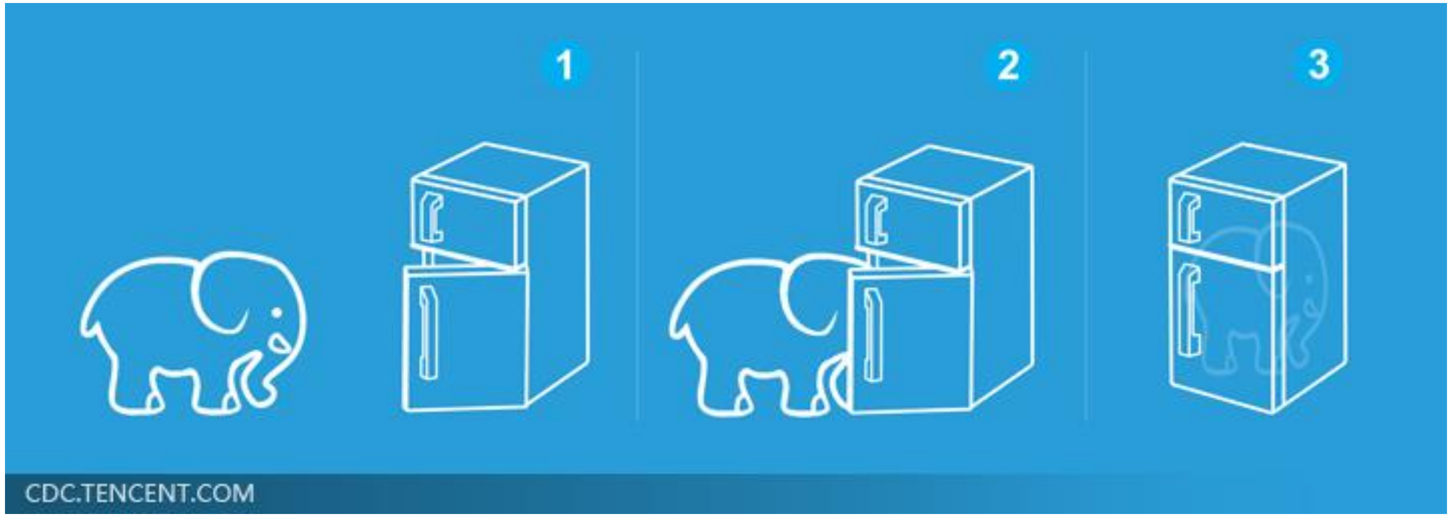
case 1:
```
score = model.evaluate(x_test,y_test)
print('Total loss on Testing Set:', score[0])
print('Accuracy of Testing Set:', score[1])
```

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

Three Steps for Deep Learning

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

Deep Learning is so simple ……



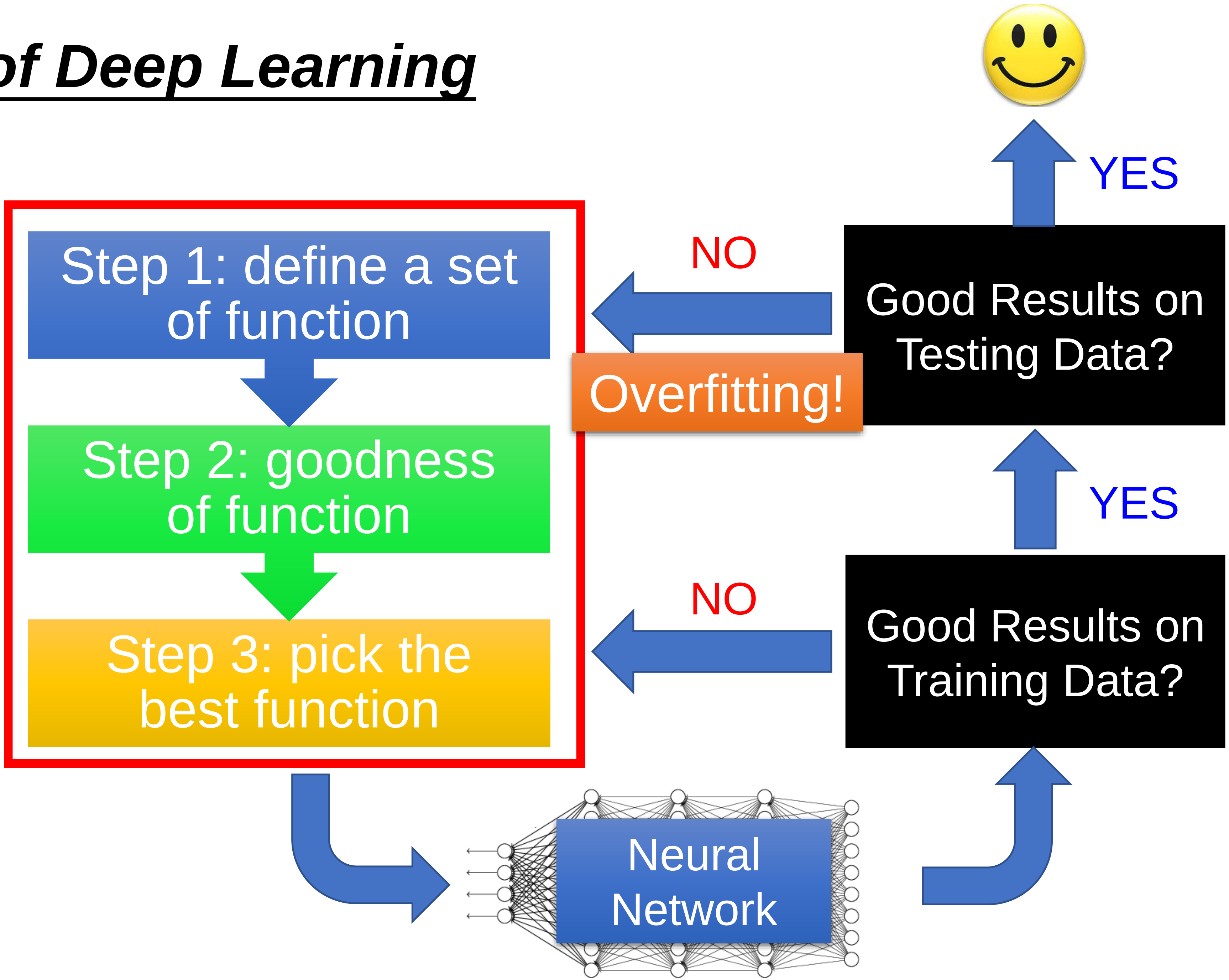
Outline

Introduction of Deep Learning

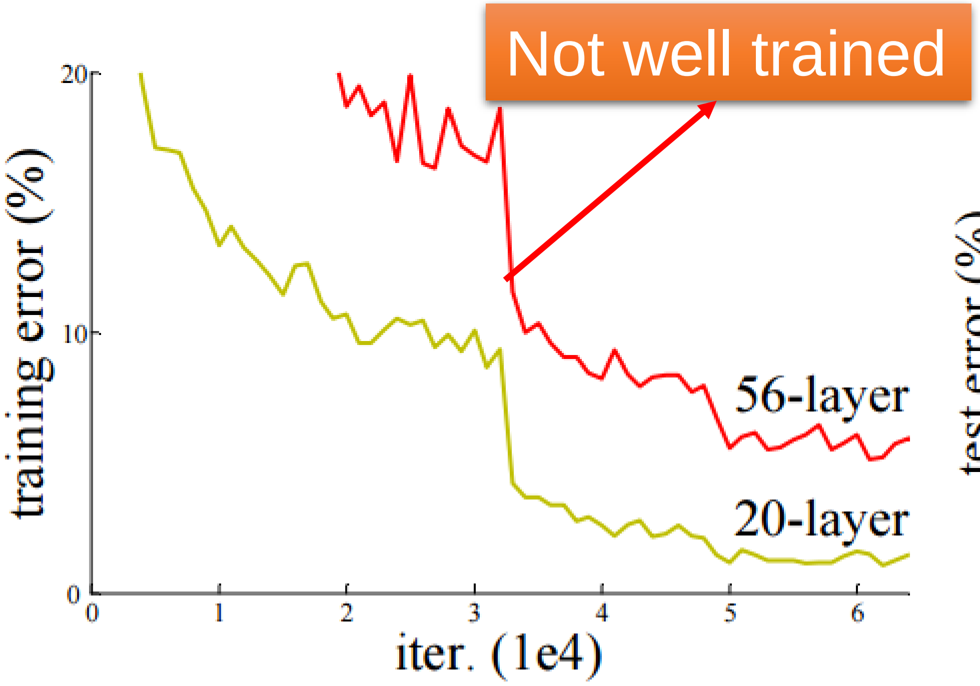
“Hello World” for Deep Learning

Tips for Deep Learning

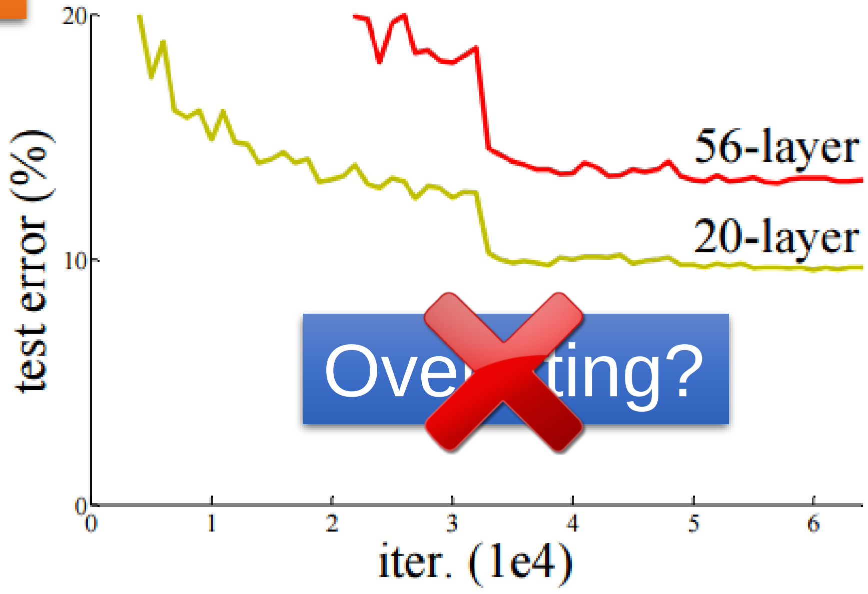
Recipe of Deep Learning



Do not always blame Overfitting



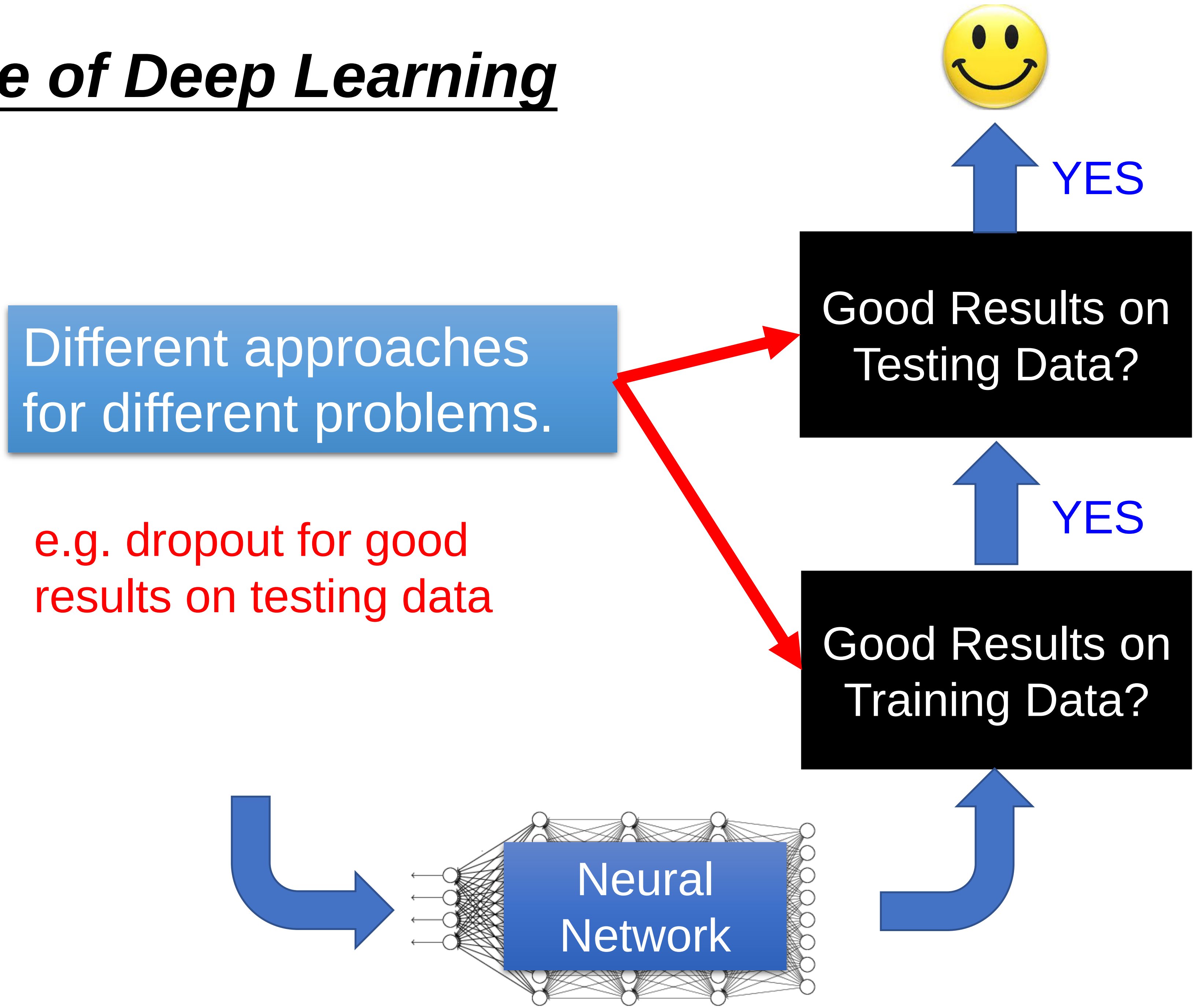
Training Data



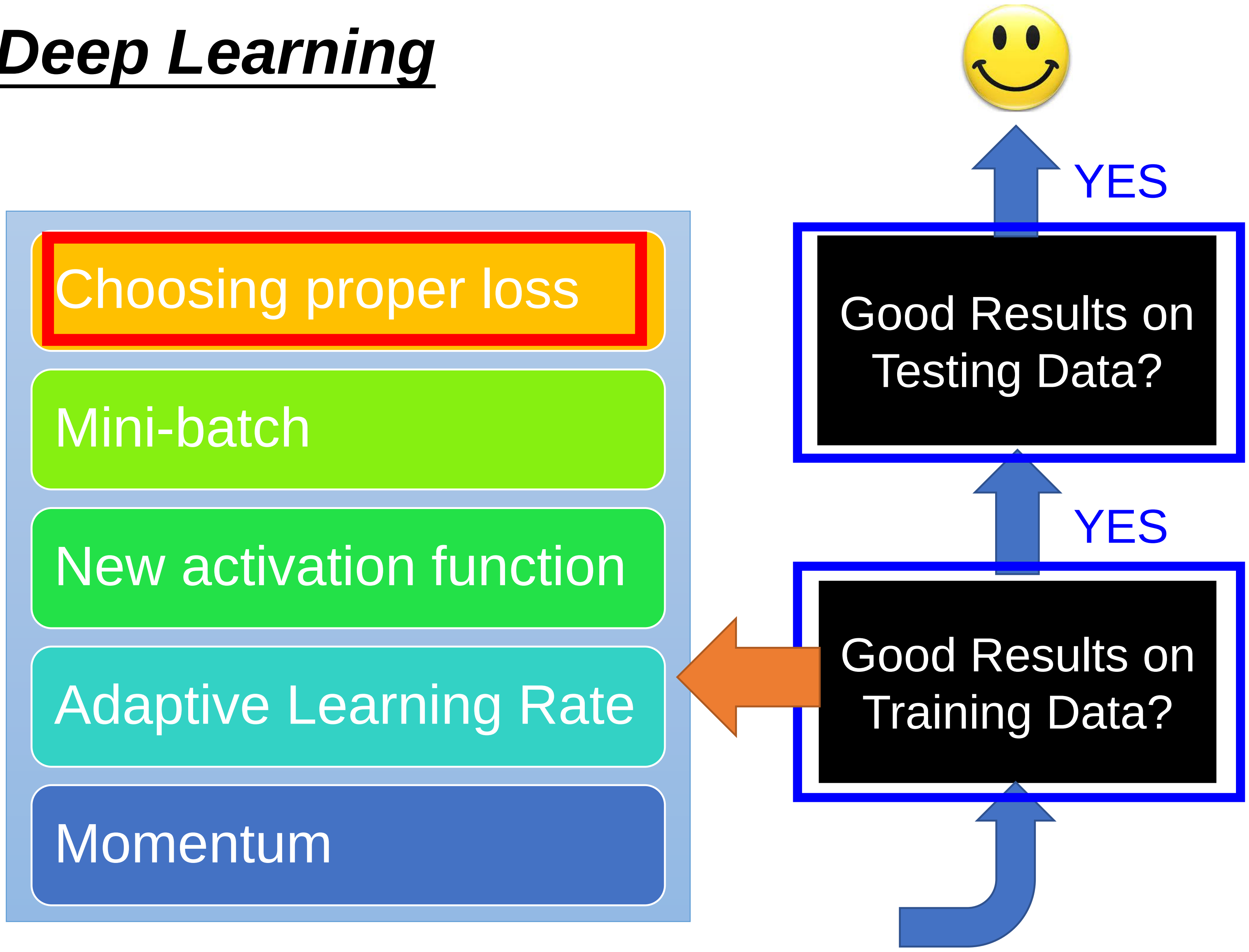
Testing Data

Deep Residual Learning for Image Recognition, 2015

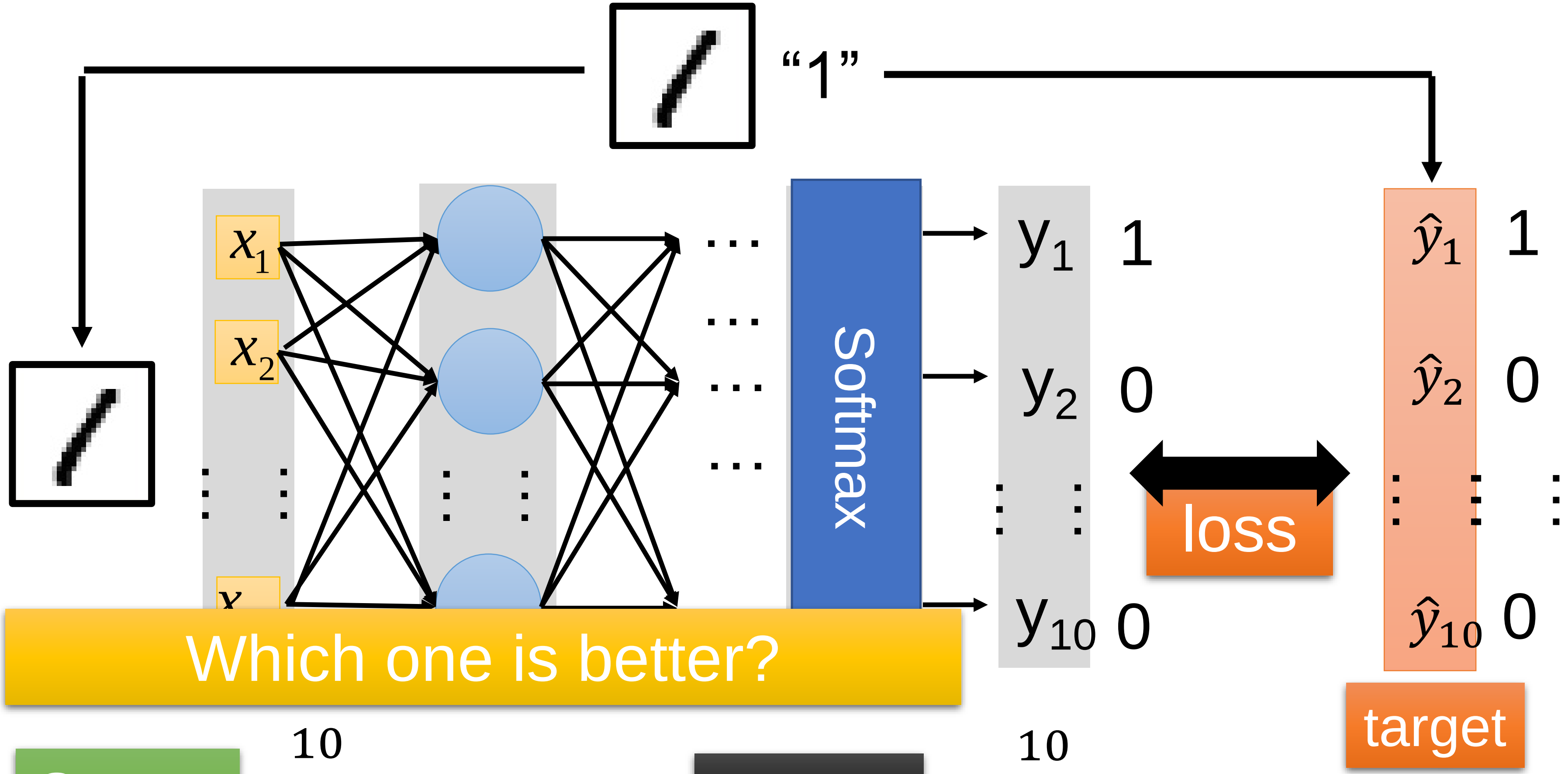
Recipe of Deep Learning



Recipe of Deep Learning



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

Demo

Square Error

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

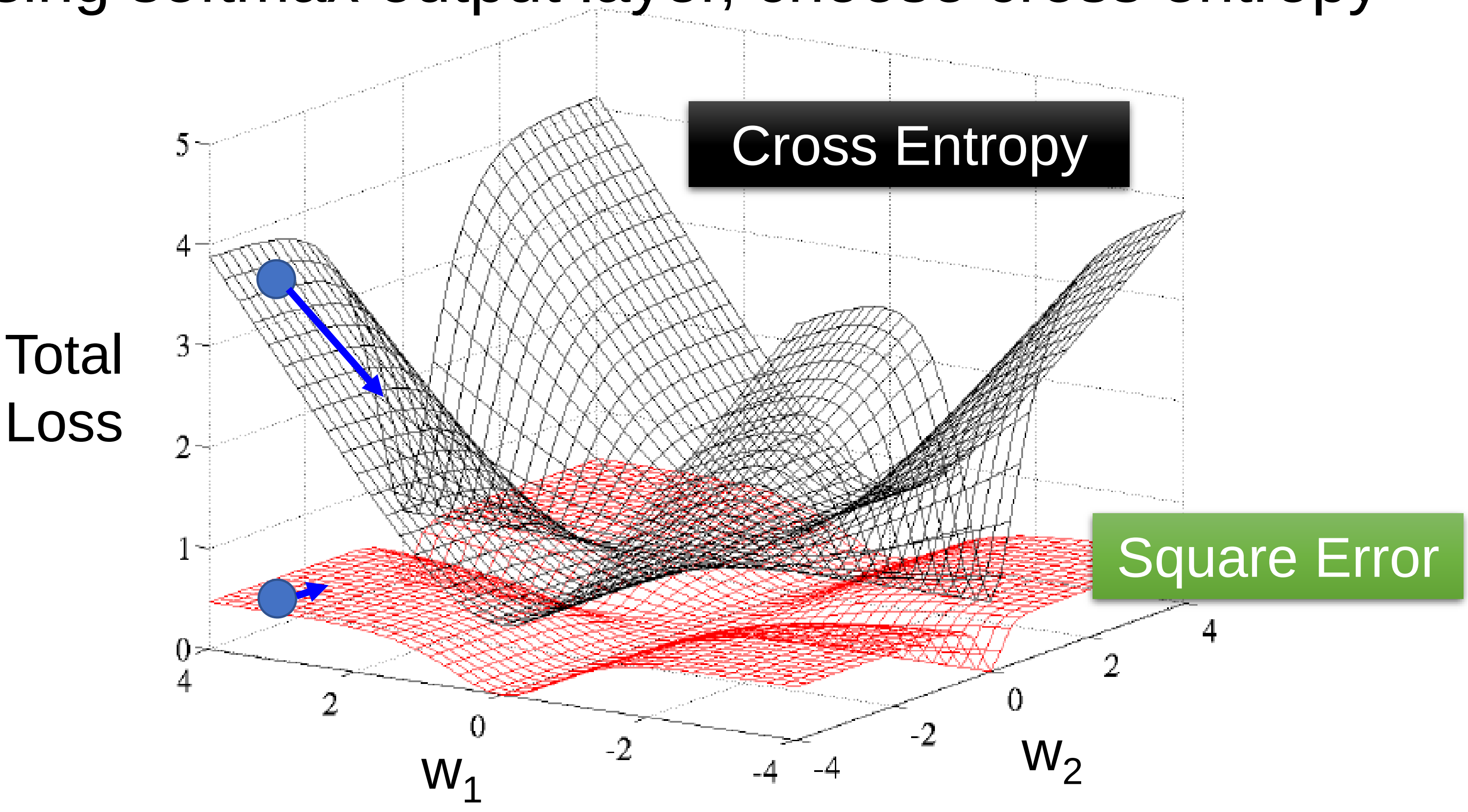
Cross Entropy

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

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

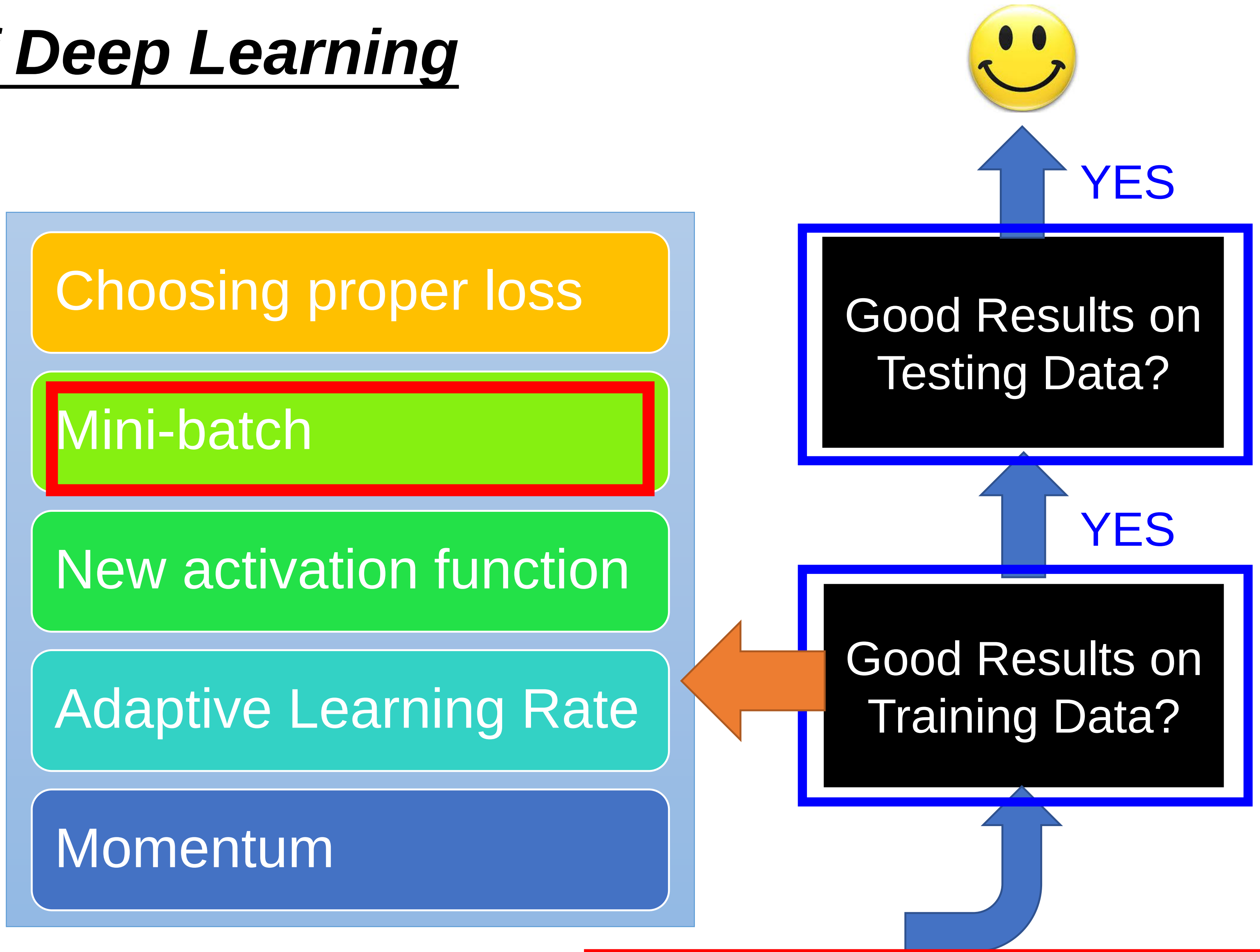
Choosing Proper Loss

When using softmax output layer, choose cross entropy



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

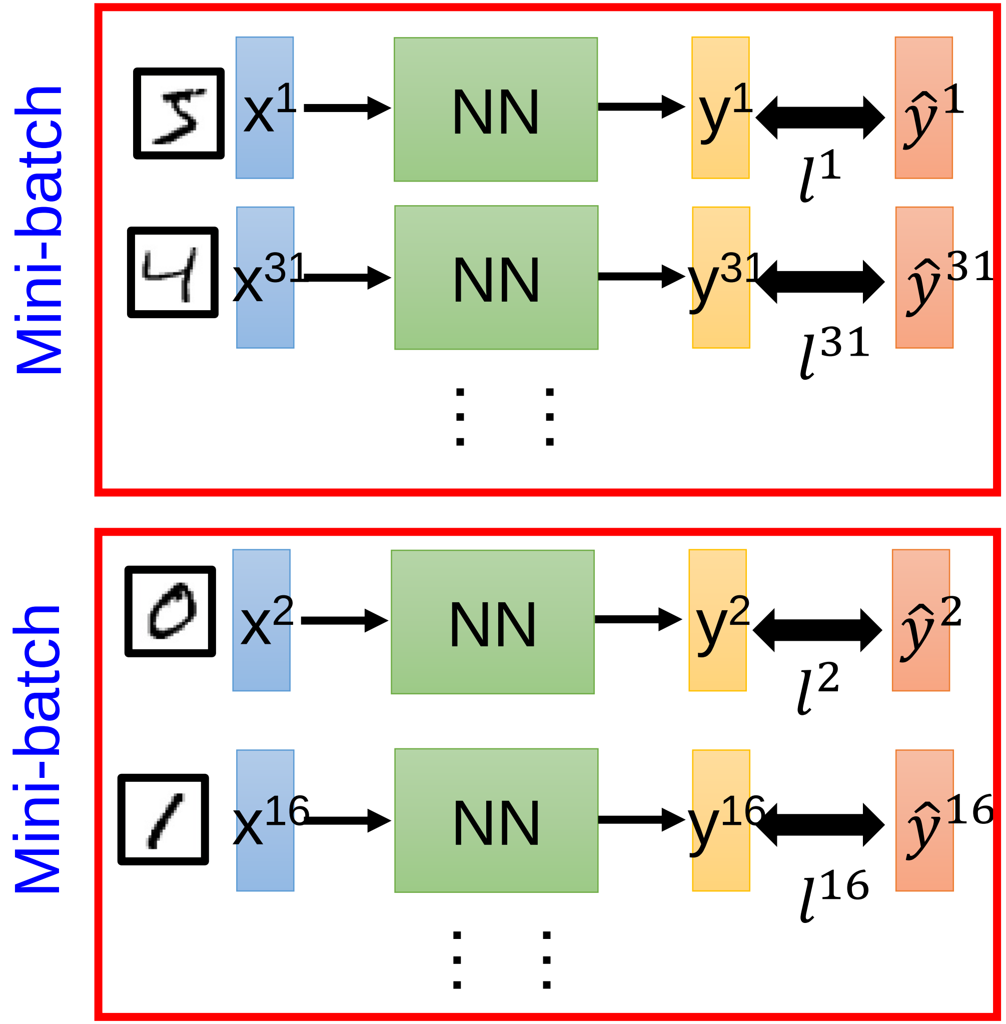
Recipe of Deep Learning




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

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

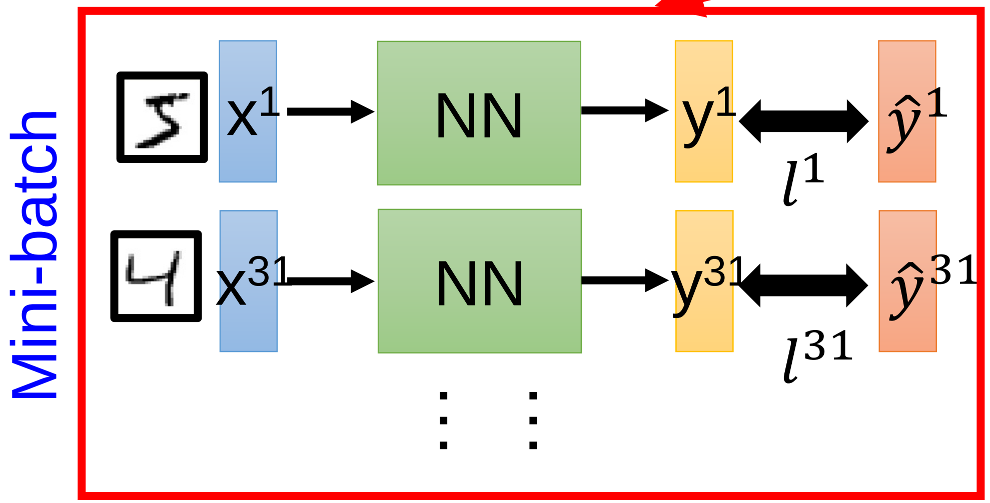
  ⋮
- Until all mini-batches have been picked

one epoch

Repeat the above process

Mini-batch

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



100 examples in a mini-batch

Repeat 20 times

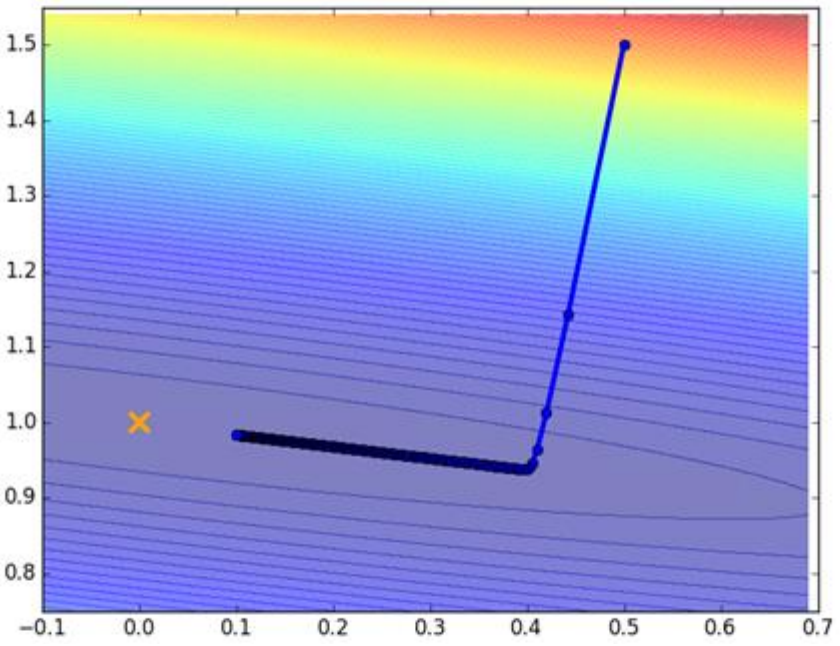
- Pick the 1st batch
  $L' = l^1 + l^{31} + \cdots$
  Update parameters once
- Pick the 2nd batch
  $L'' = l^2 + l^{16} + \cdots$
  Update parameters once

  ⋮
- Until all mini-batches have been picked

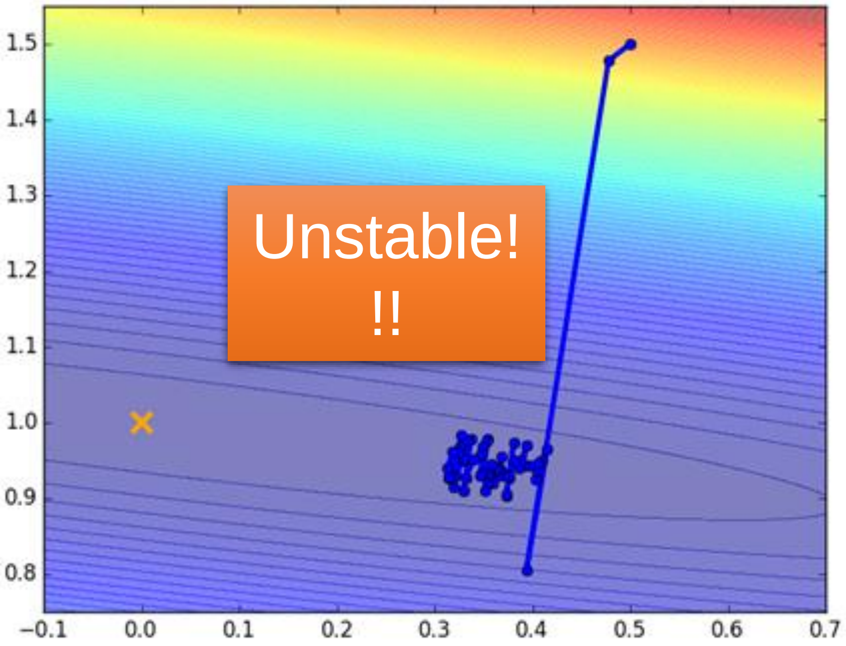
one epoch

Mini-batch

<u>Original Gradient Descent</u>



<u>With Mini-batch</u>



Unstable!!!

The colors represent the total loss.

Mini-batch is Faster

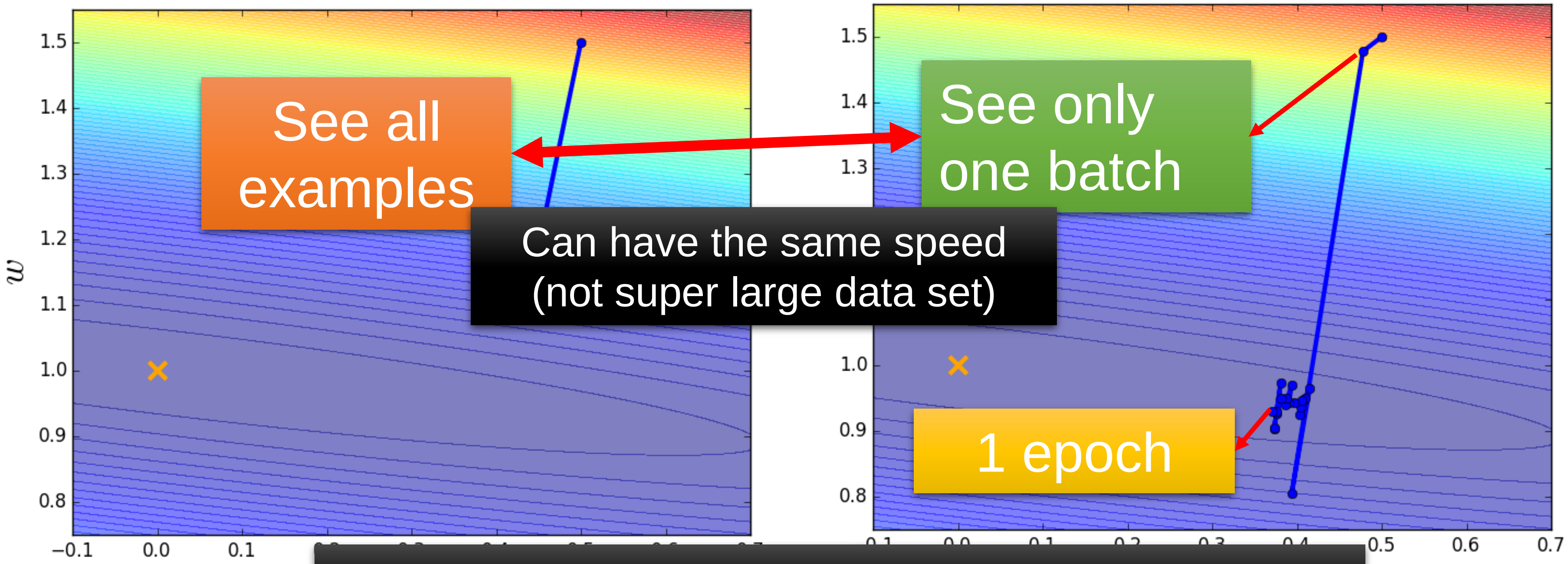
Not always true with parallel computing.

<u>Original Gradient Descent</u>

Update after seeing all examples

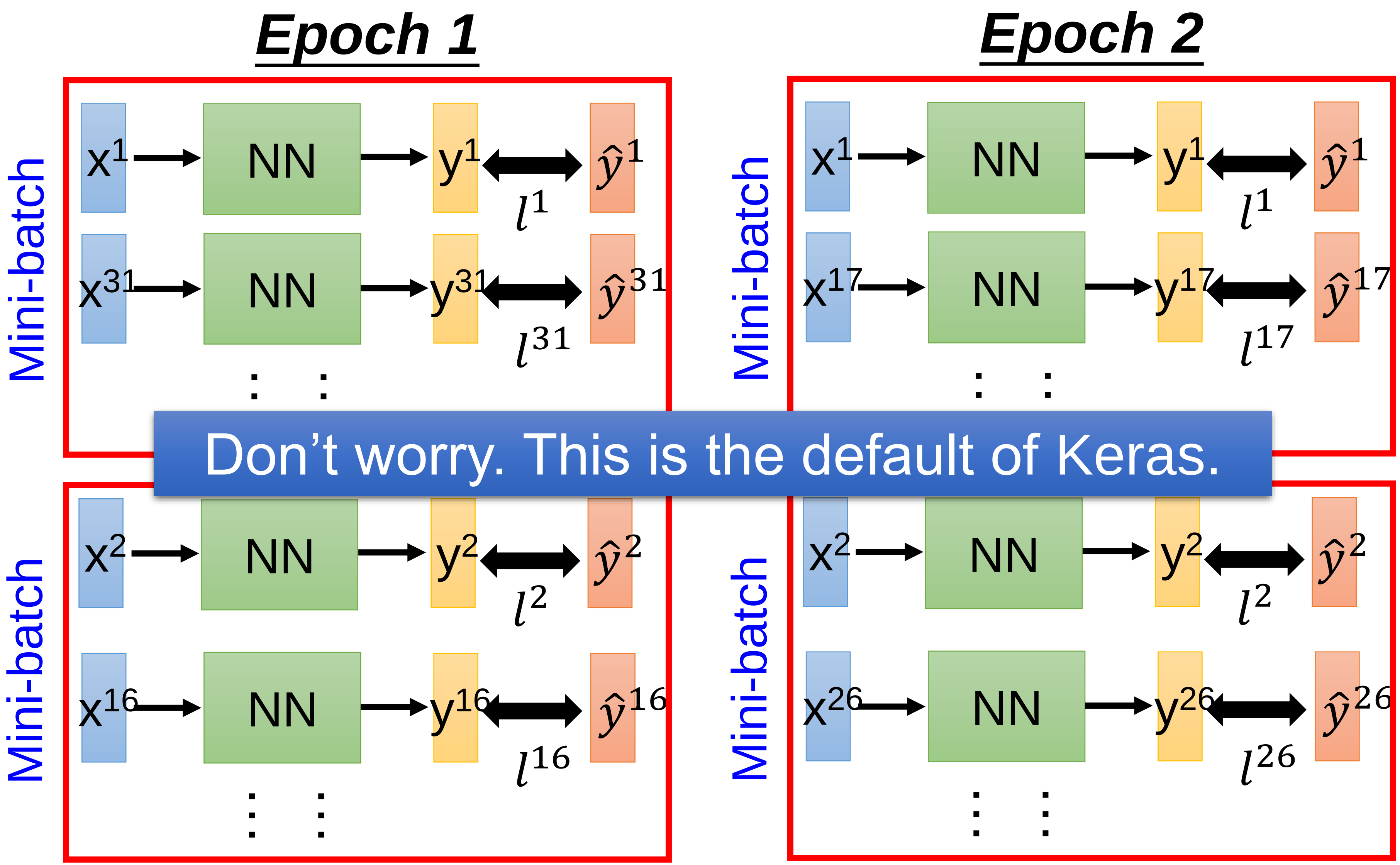
<u>With Mini-batch</u>

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



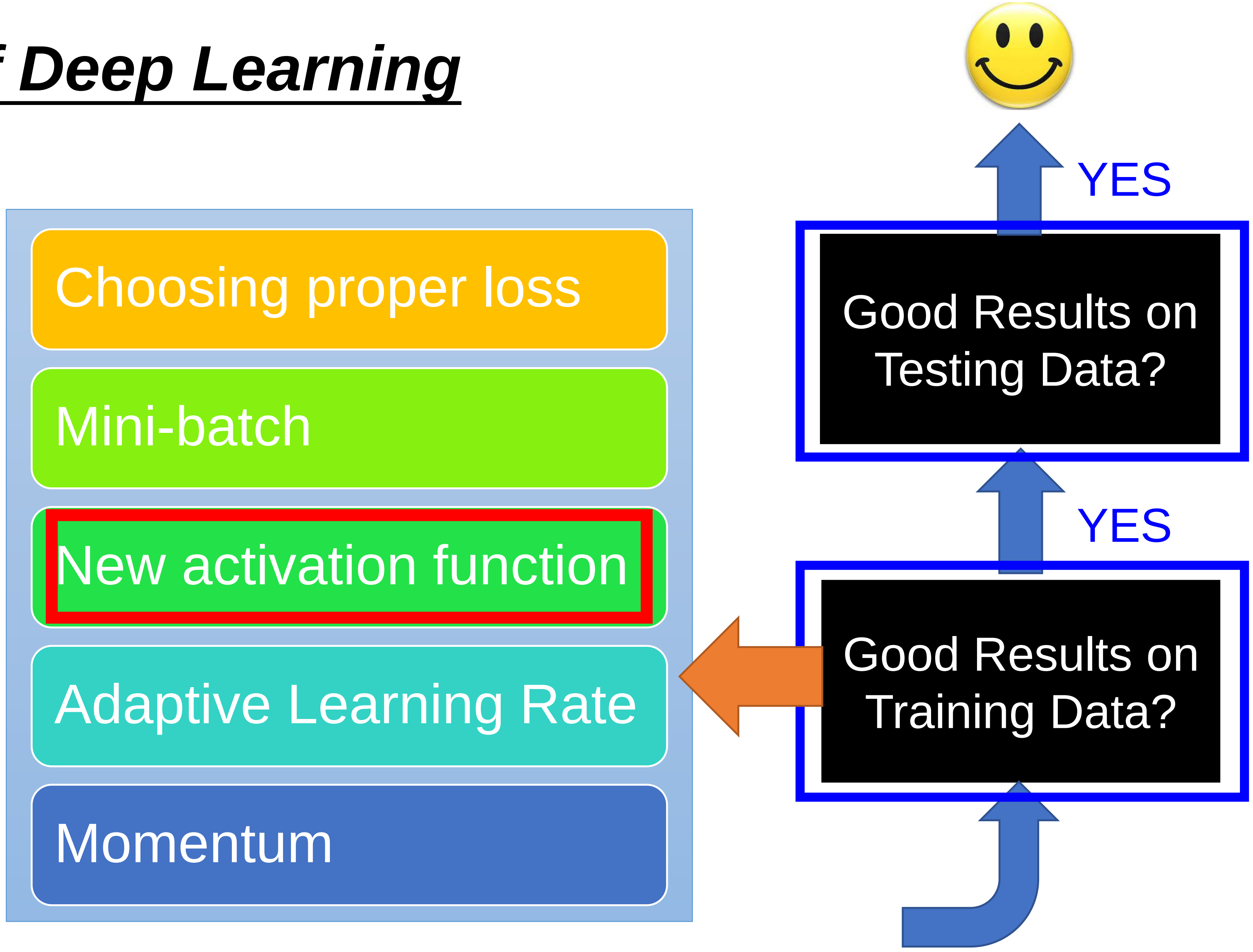
Mini-batch has better performance!

Shuffle the training examples for each epoch

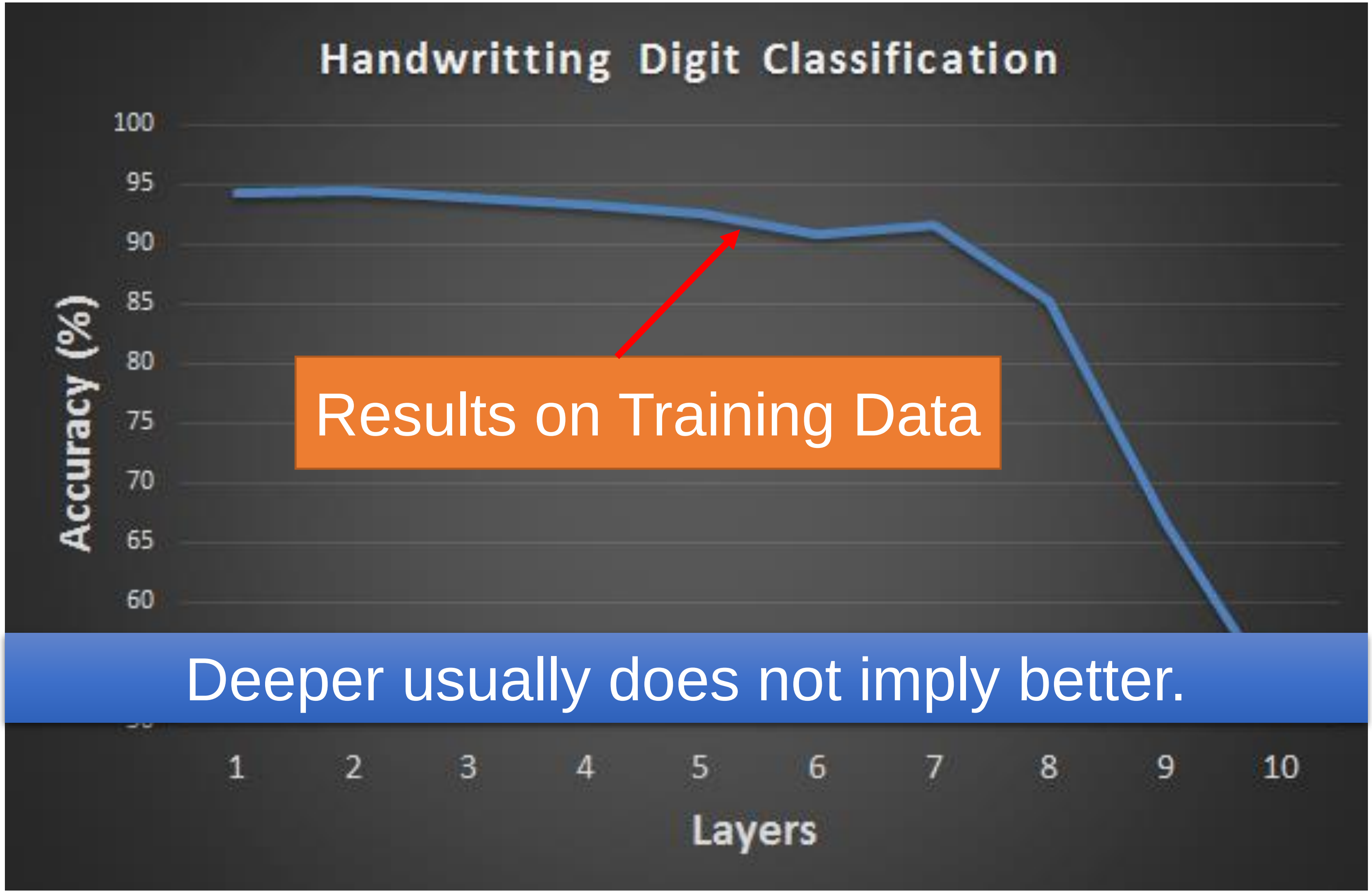


Don't worry. This is the default of Keras.

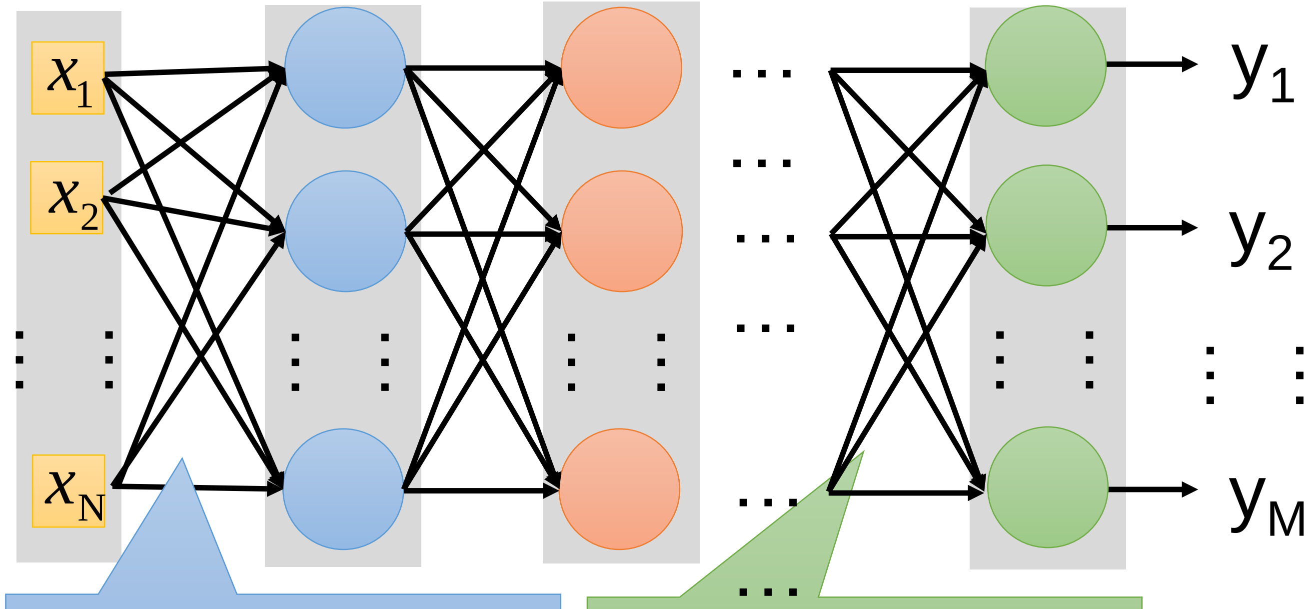
Recipe of Deep Learning



Hard to get the power of Deep …



Vanishing Gradient Problem

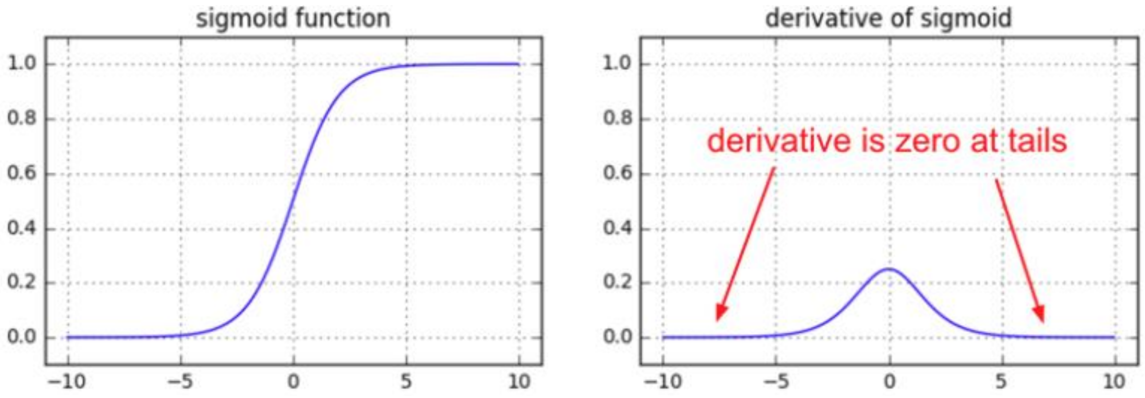


Smaller gradients | Larger gradients

Learn very slow | Learn very fast

Almost random | Already converge

based on random!?



$$y = \frac{1}{1+e^{-x}}$$

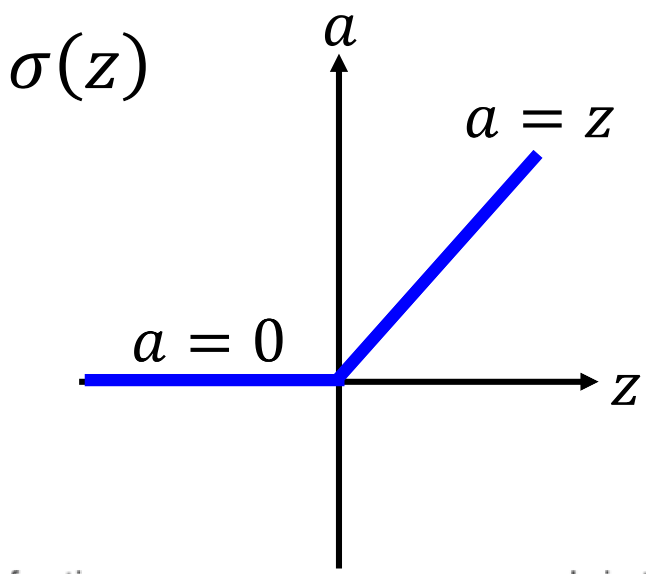
$$\frac{dy}{dx} = -\frac{1}{(1+e^{-x})^2}(-e^{-x}) = \frac{e^{-x}}{(1+e^{-x})^2}$$

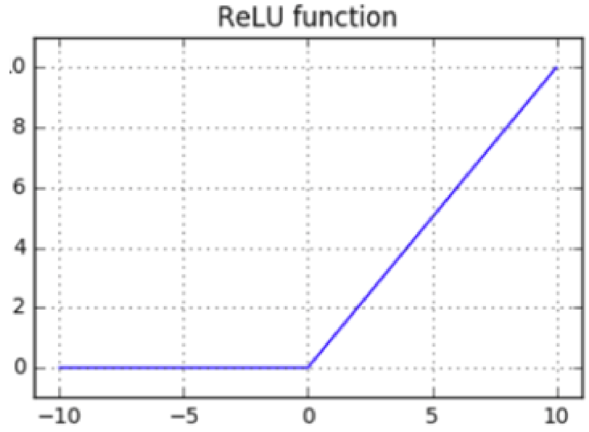
$$= \frac{1}{1+e^{-x}}\left(1 - \frac{1}{1+e^{-x}}\right) = y(1-y)$$

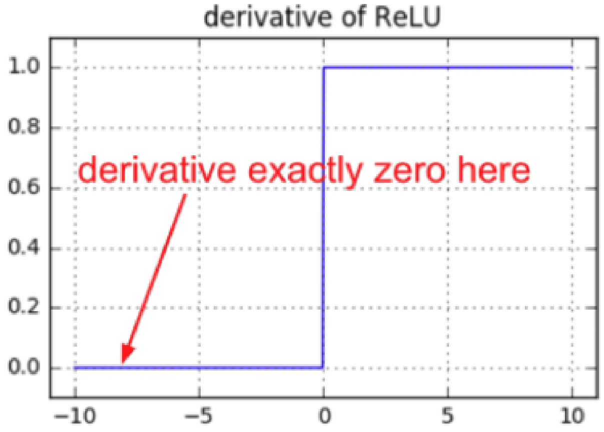
- **Max(dy/dx) = 0.25 => Slow convergence rate (with basic SGD), even vanishing gradient :**
  - **every time the gradient signal flows through a sigmoid gate, its magnitude always diminishes by one quarter (or more)**

ReLU

- Rectified Linear Unit (ReLU)





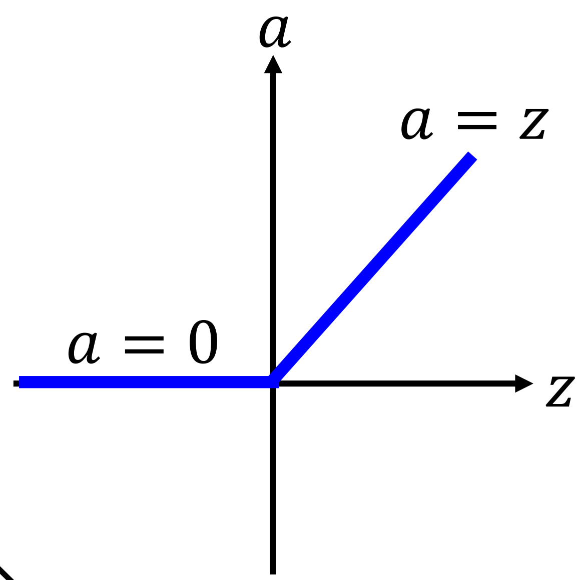


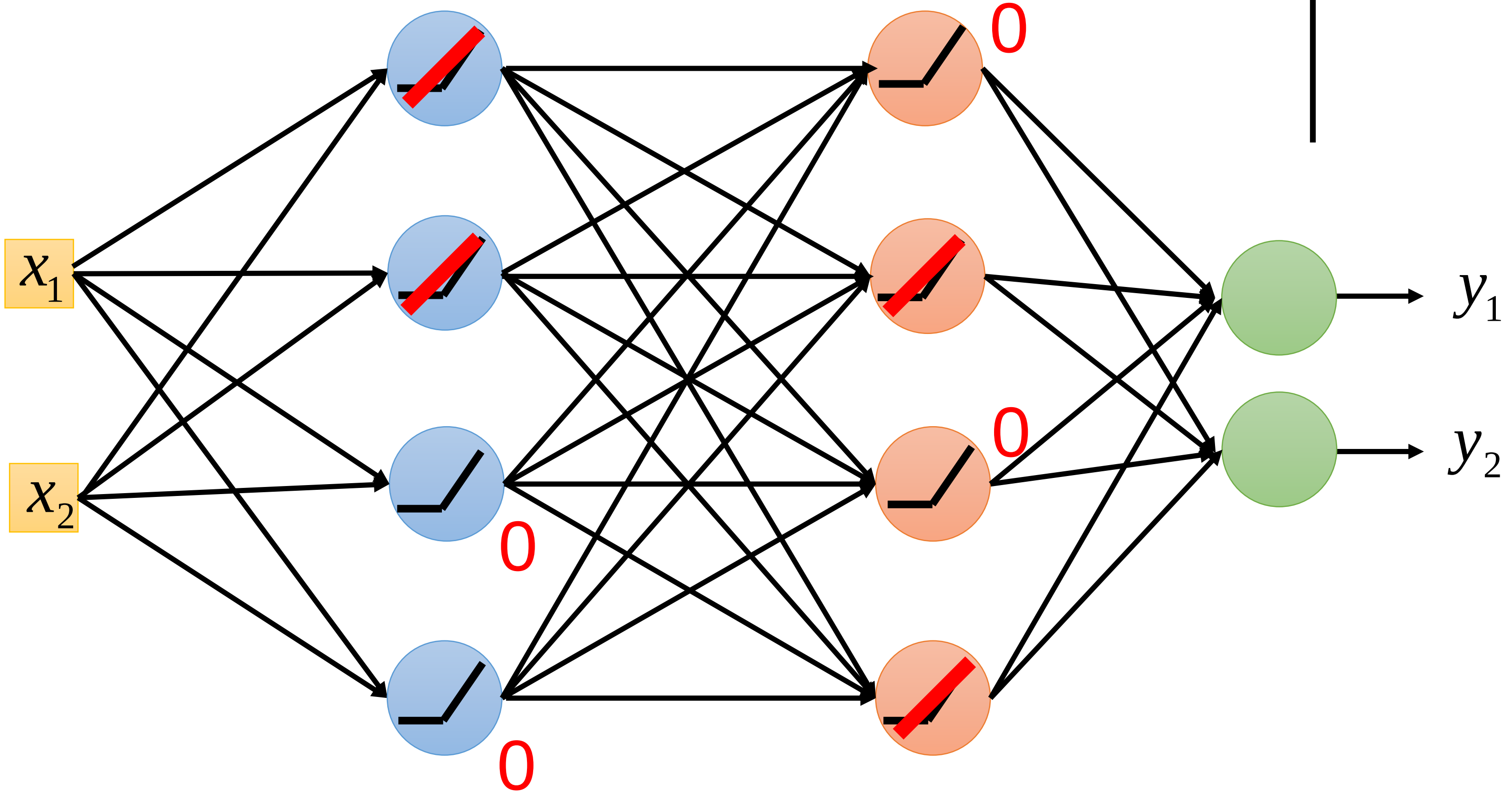
Reason:

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

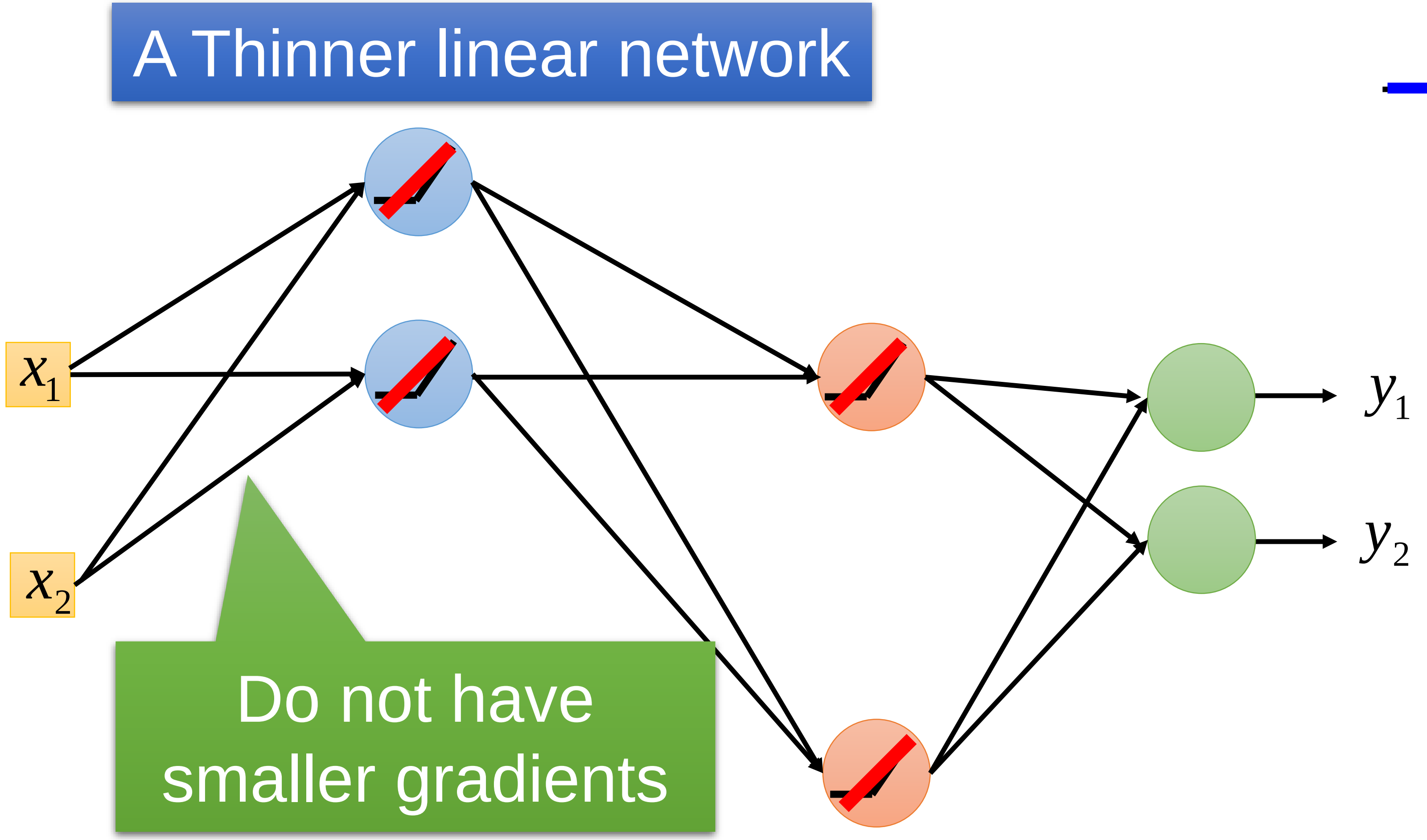
[Xavier Glorot, AISTATS'11] [Andrew L. Maas, ICML'13] [Kaiming He, arXiv'15]

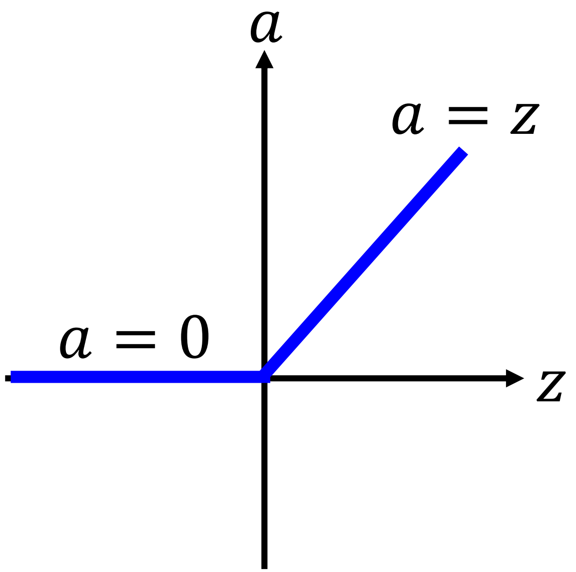
ReLU





ReLU

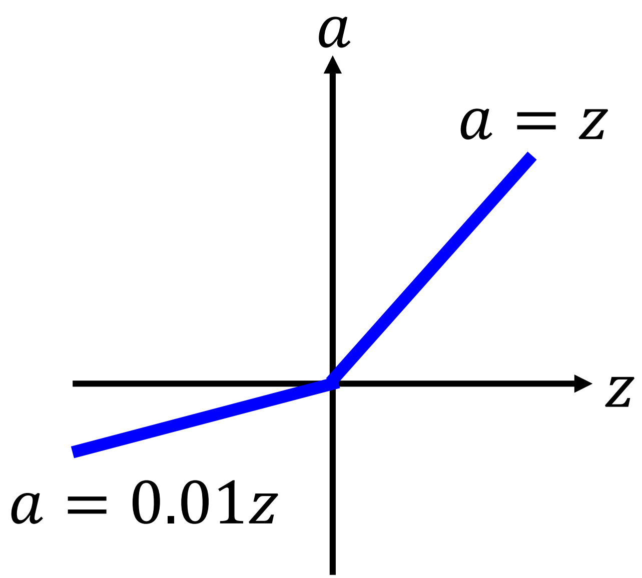




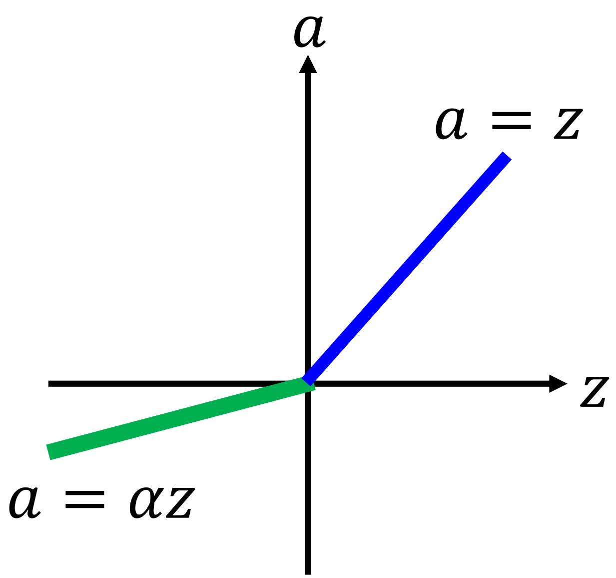
But => Dying ReLUs

ReLU - variant

$Leaky\ ReLU$

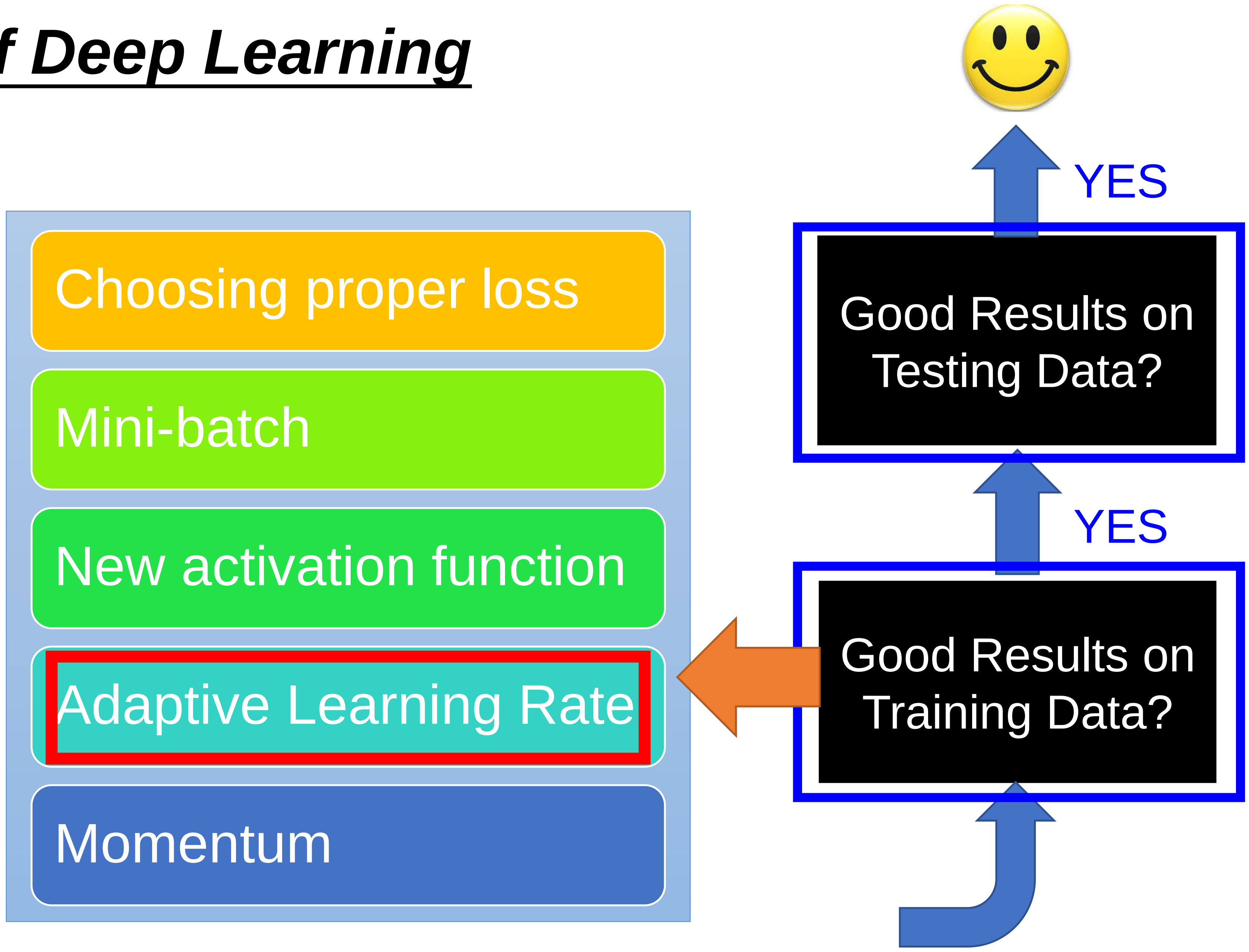


$Parametric\ ReLU$



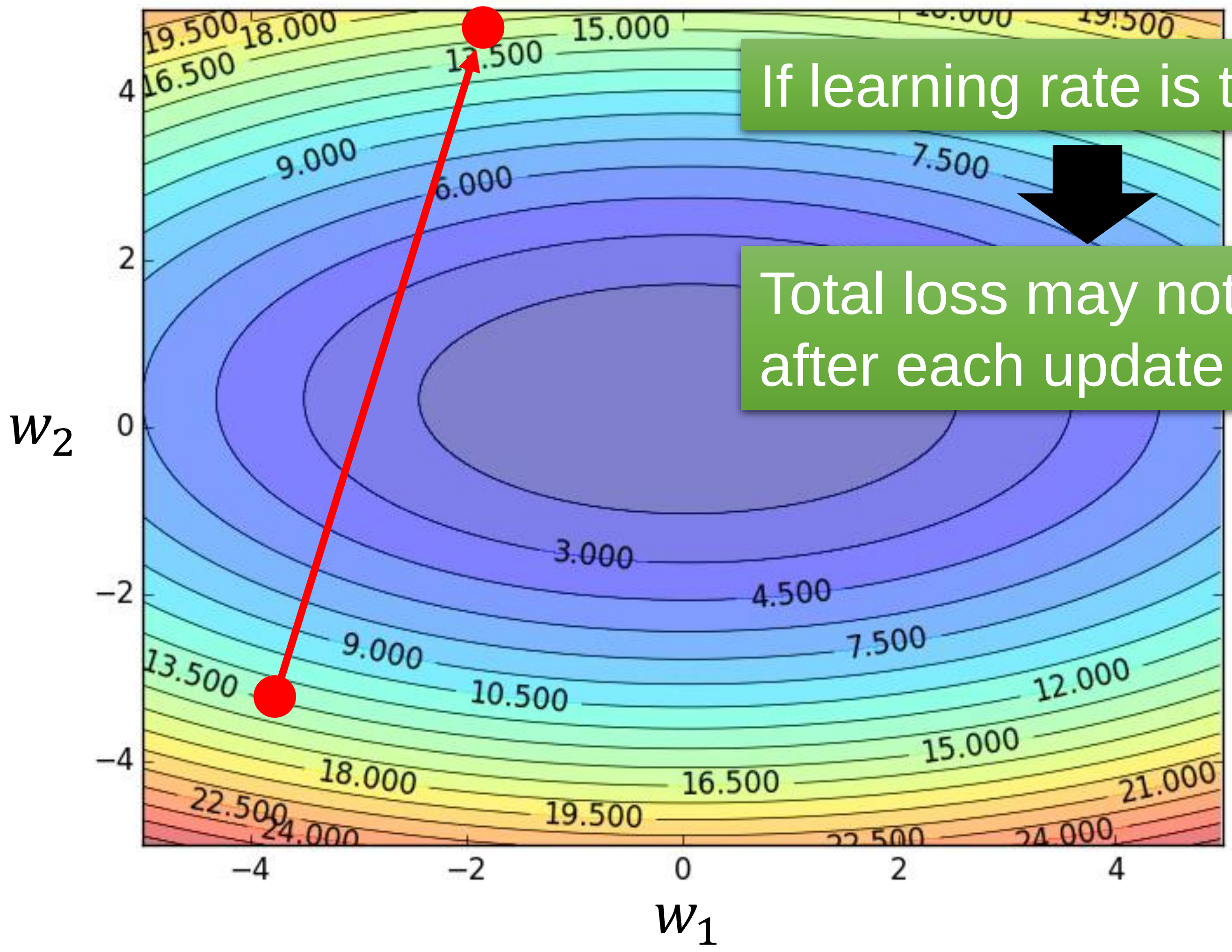
α also learned by gradient descent

Recipe of Deep Learning



Learning Rates

Set the learning rate η carefully

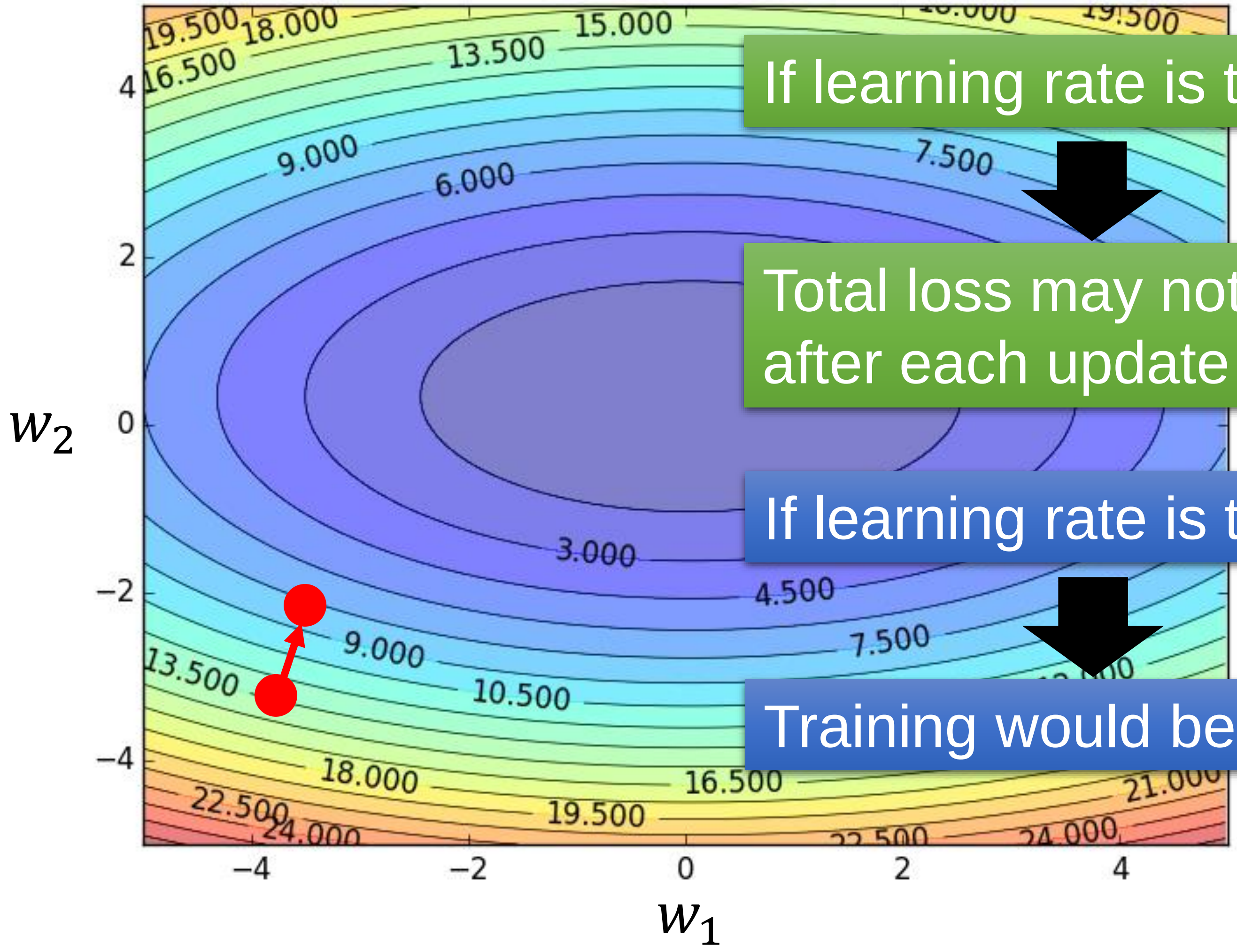


If learning rate is too large

Total loss may not decrease after each update

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

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

	g^0
w_1	0.1

Learning rate:

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

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

	g^0
w_2	20.0

Learning rate:

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

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

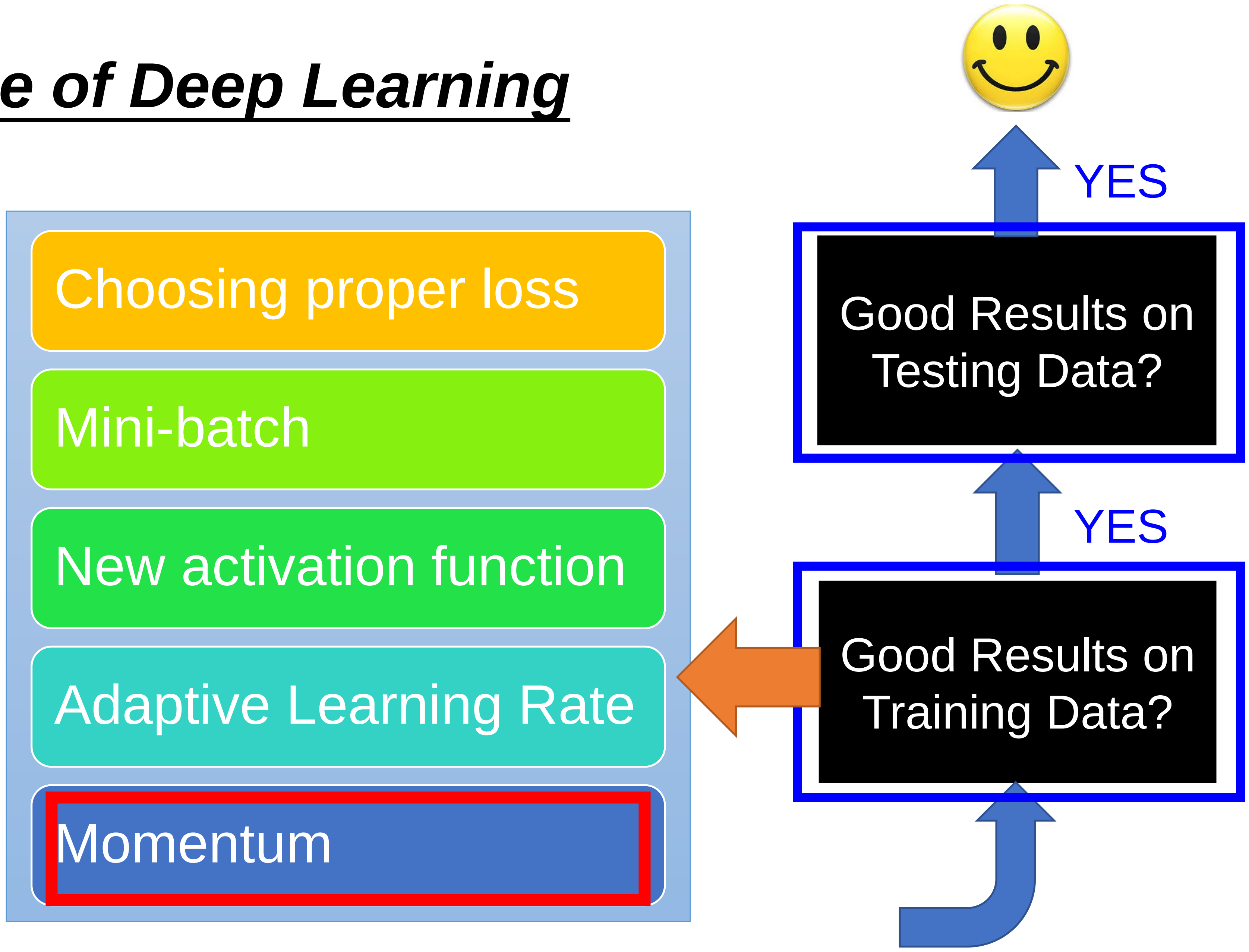
Observation: 1. Learning rate is smaller and smaller for all parameters

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

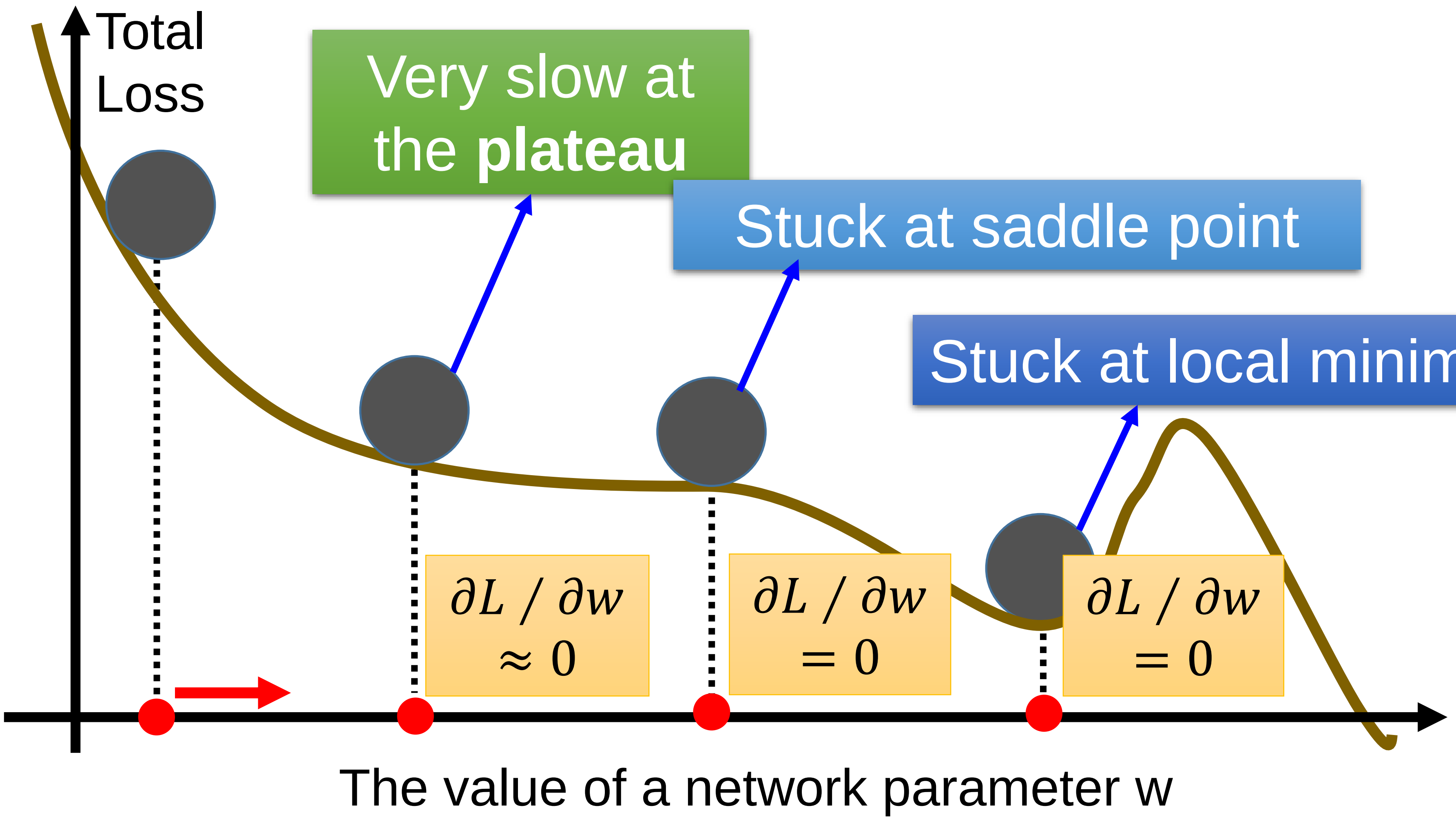
Not the whole story ……

- 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

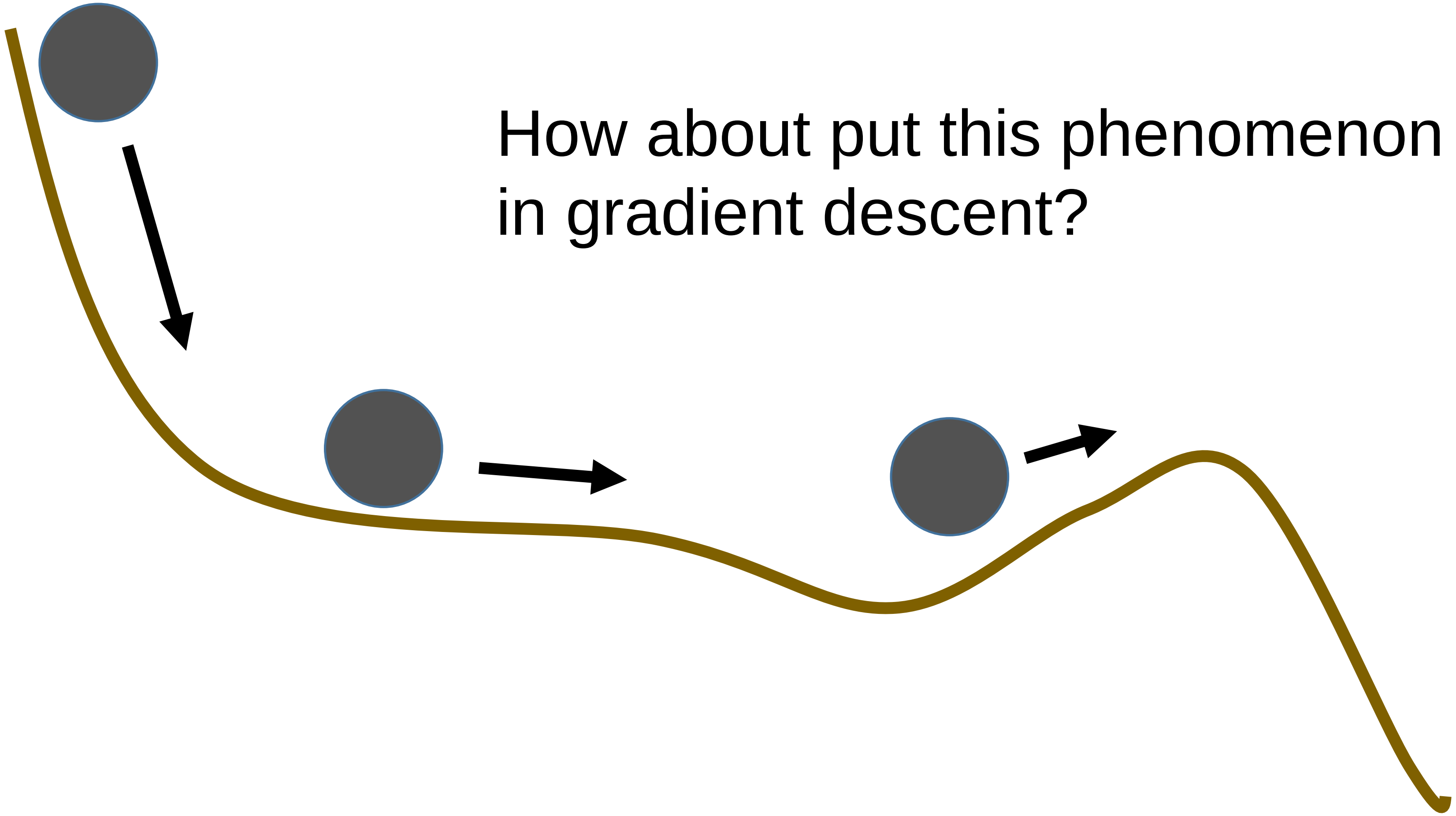


Hard to find optimal network parameters



In physical world ……

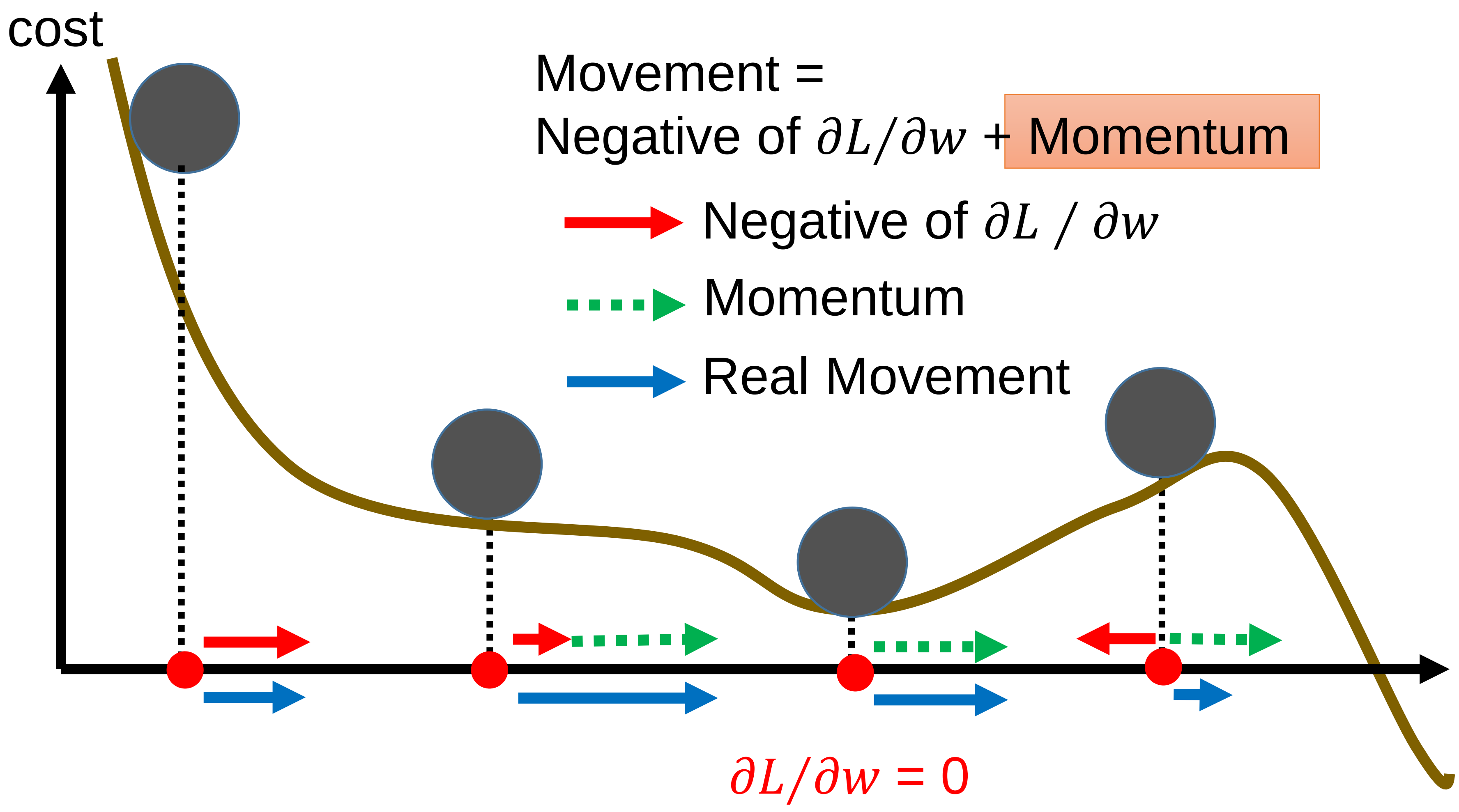
- Momentum



How about put this phenomenon in gradient descent?

Momentum

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



Adam

RMSProp (Advanced Adagrad) + Momentum

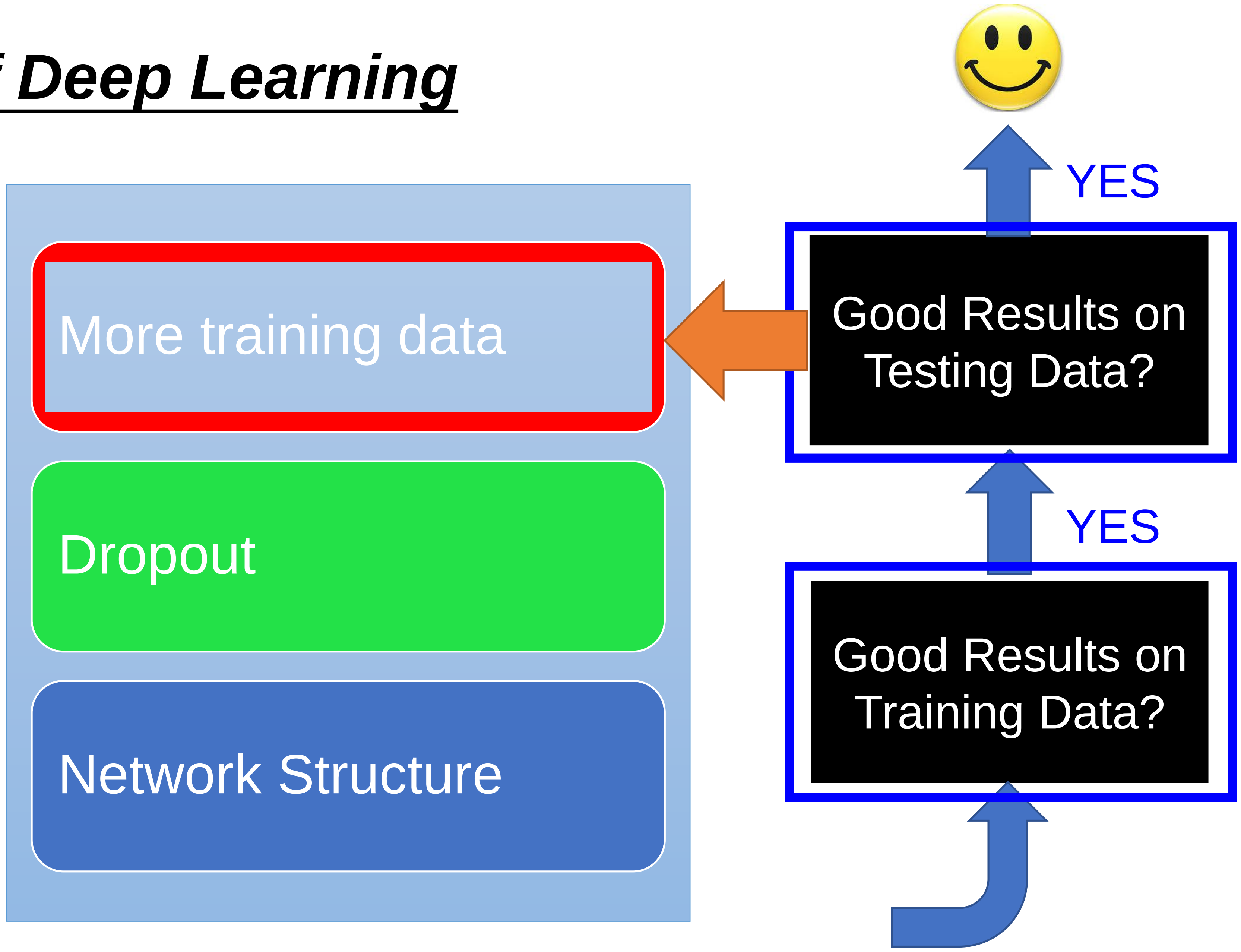
```
model.compile(loss='categorical_crossentropy',
              optimizer=SGD(lr=0.1),
              metrics=['accuracy'])
```

```
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)

Recipe of Deep Learning

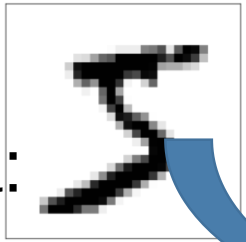


Panacea for Overfitting

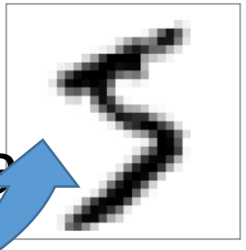
- Have more training data
- ***Create*** more training data (?)

Handwriting recognition:

Original Training Data:



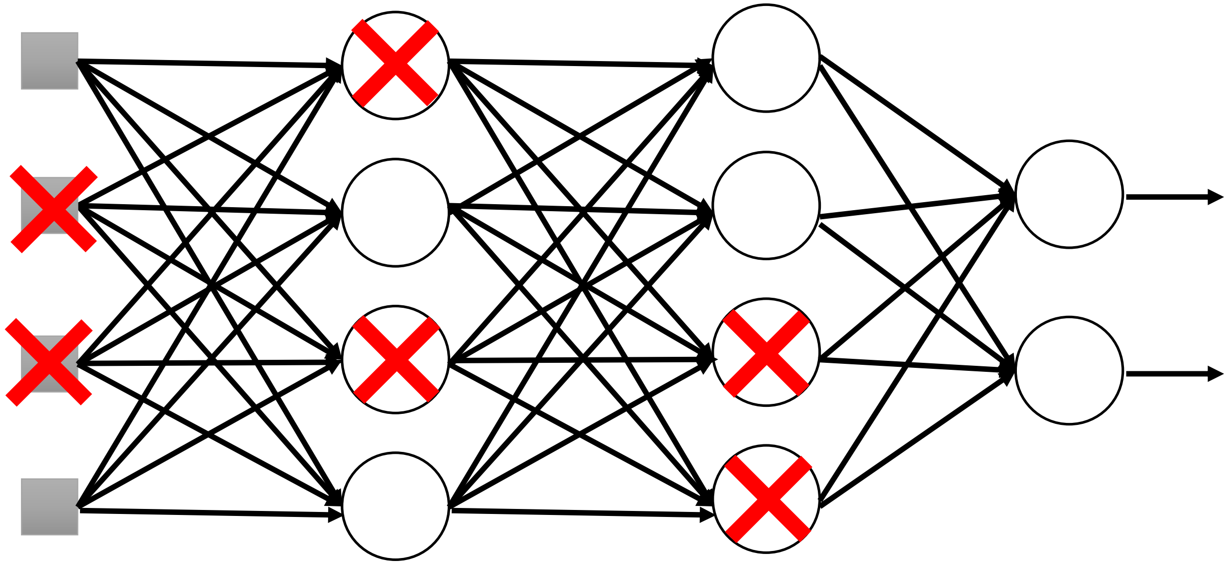
Created Training Data:



Shift 15°

Dropout

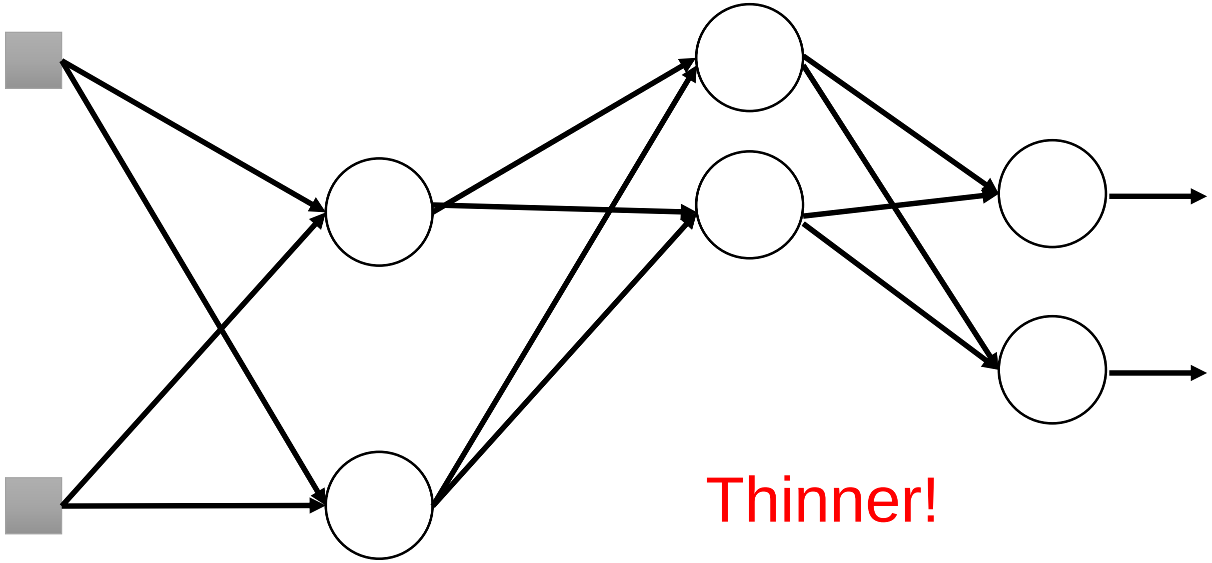
Training:



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

Dropout

Training:

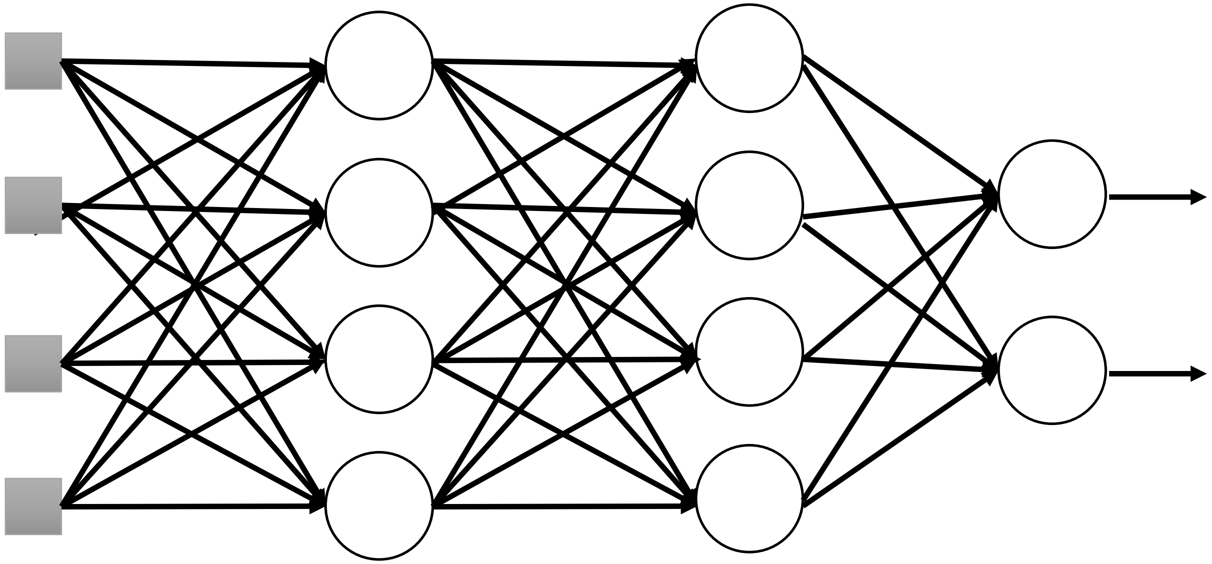


- **Each time before updating the parameters**
  - Each neuron has p% to dropout
    - **The structure of the network is changed.**
  - Using the new network for training

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

Testing

No dropout
(拿下重物後就變很強)

Training

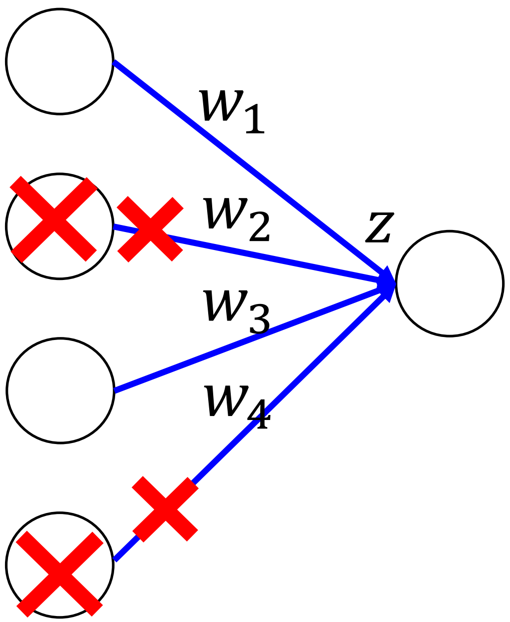
Dropout (腳上綁重物)

Dropout - Intuitive Reason

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

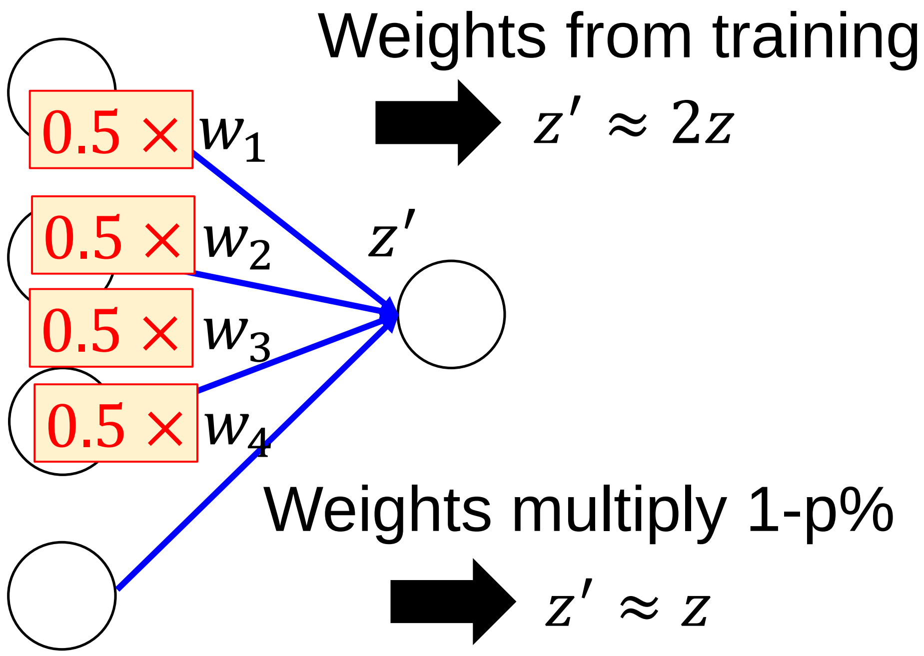
Training of Dropout

Assume dropout rate is 50%

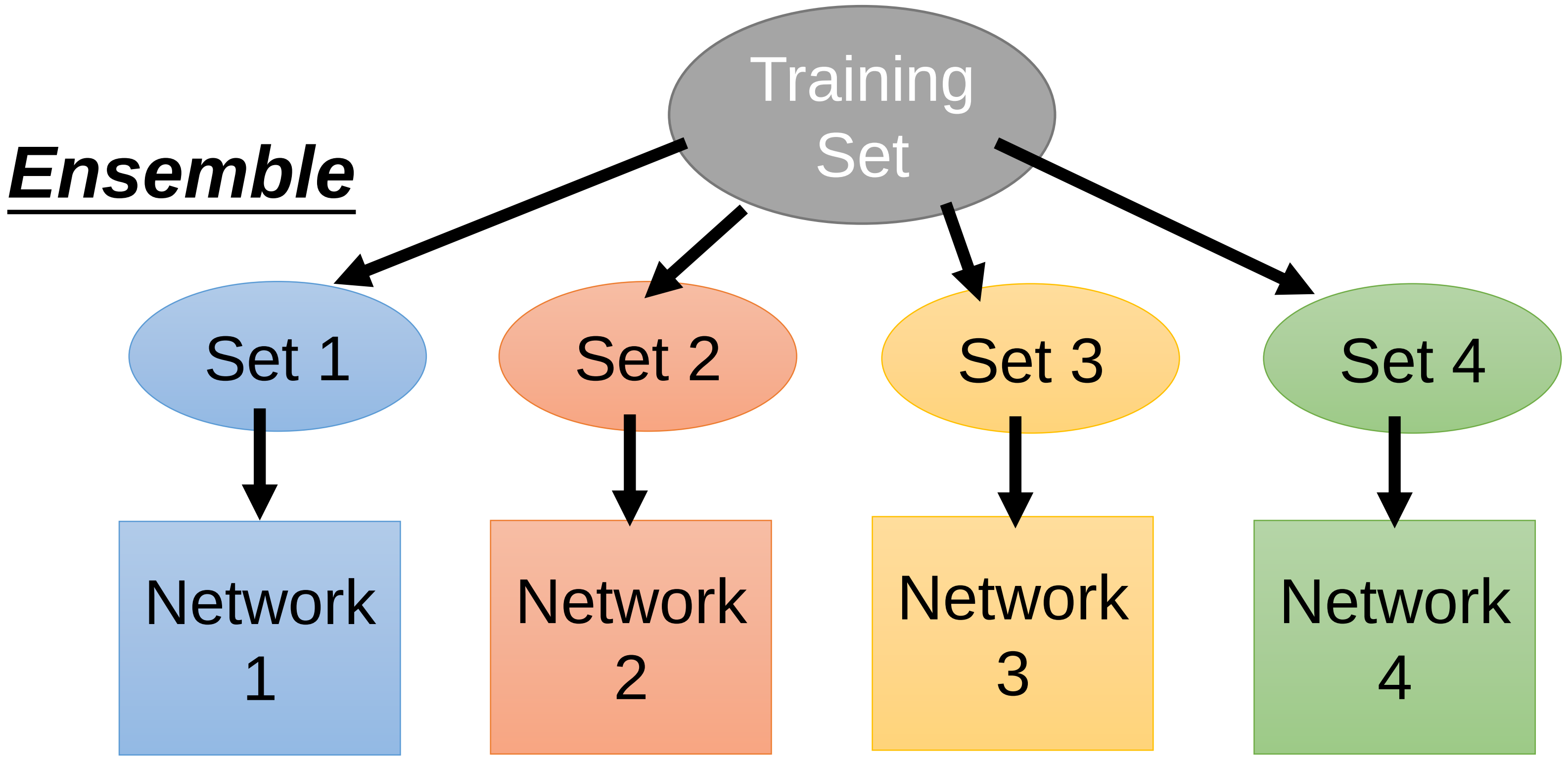


Testing of Dropout

No dropout



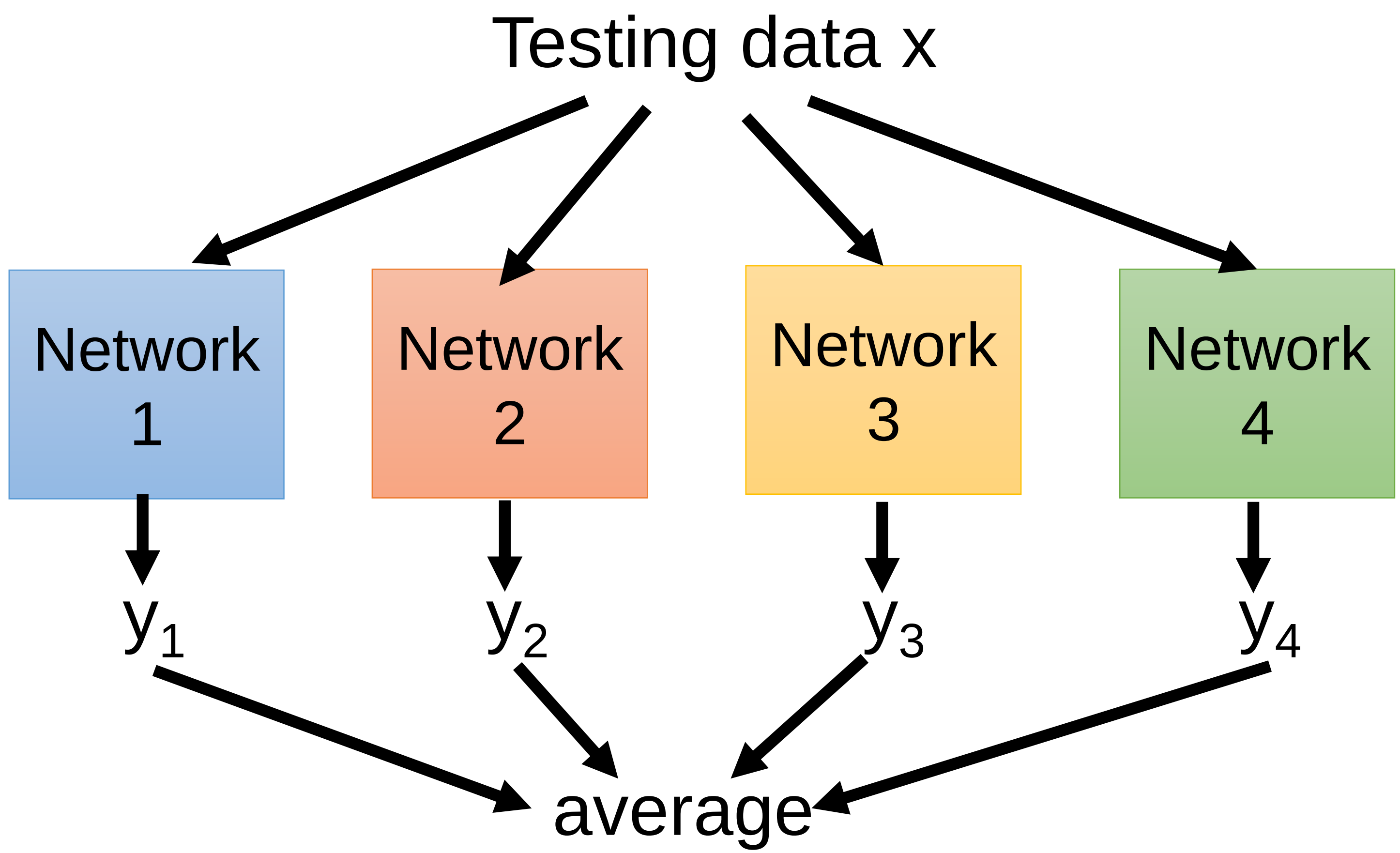
Dropout is a kind of ensemble.



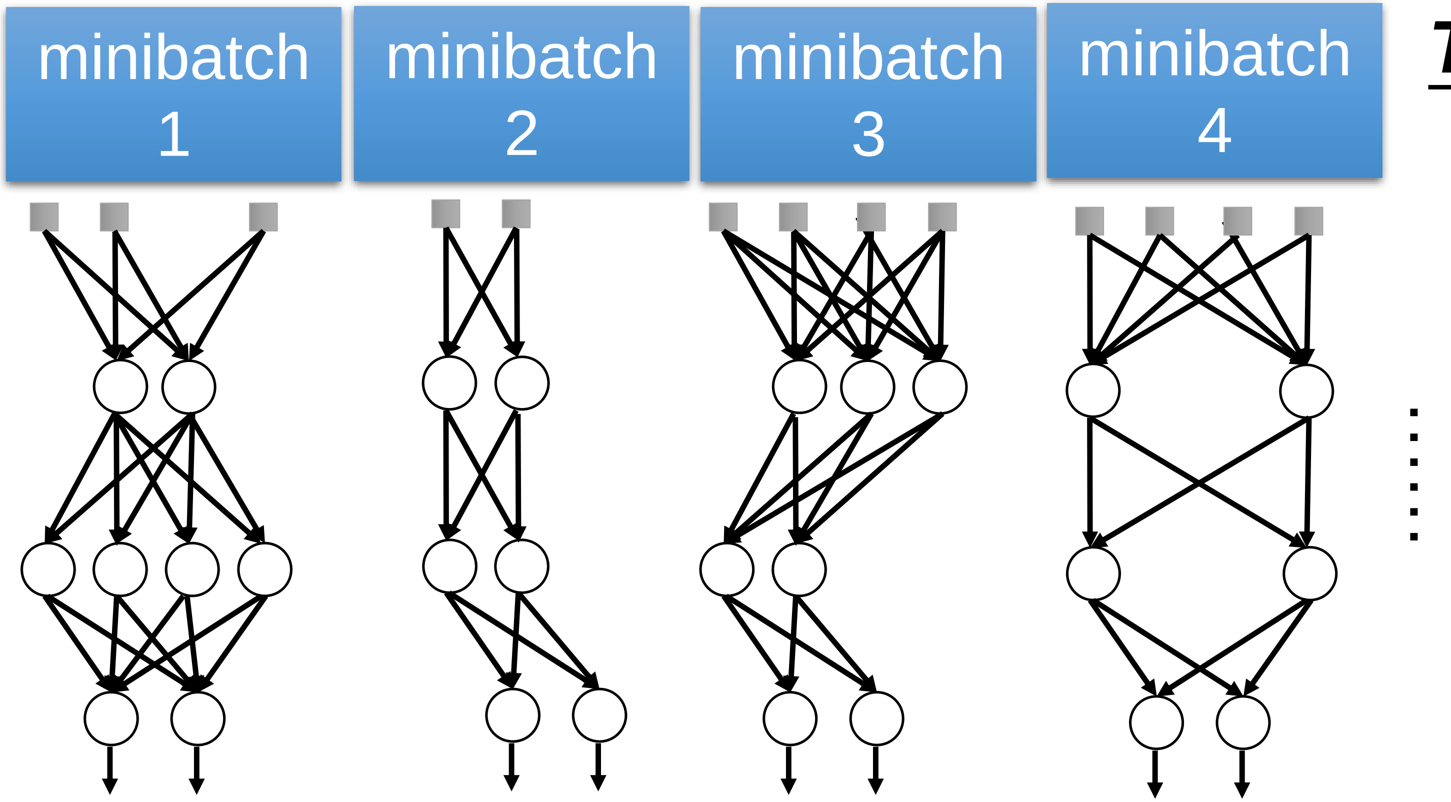
Train a bunch of networks with different structures

Dropout is a kind of ensemble.

Ensemble



Dropout is a kind of ensemble.



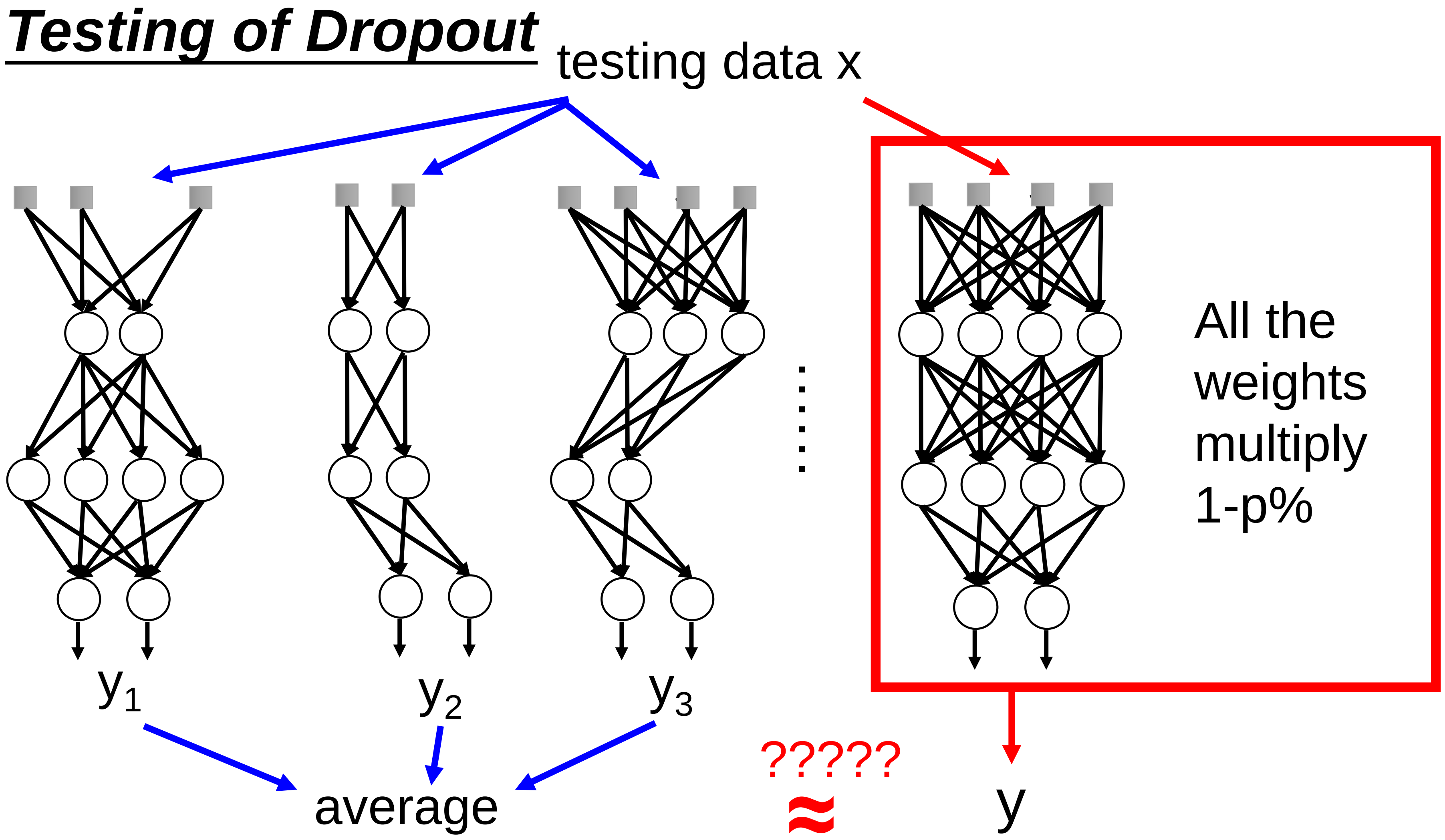
<u>Training of Dropout</u>

M neurons

2^M possible networks

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

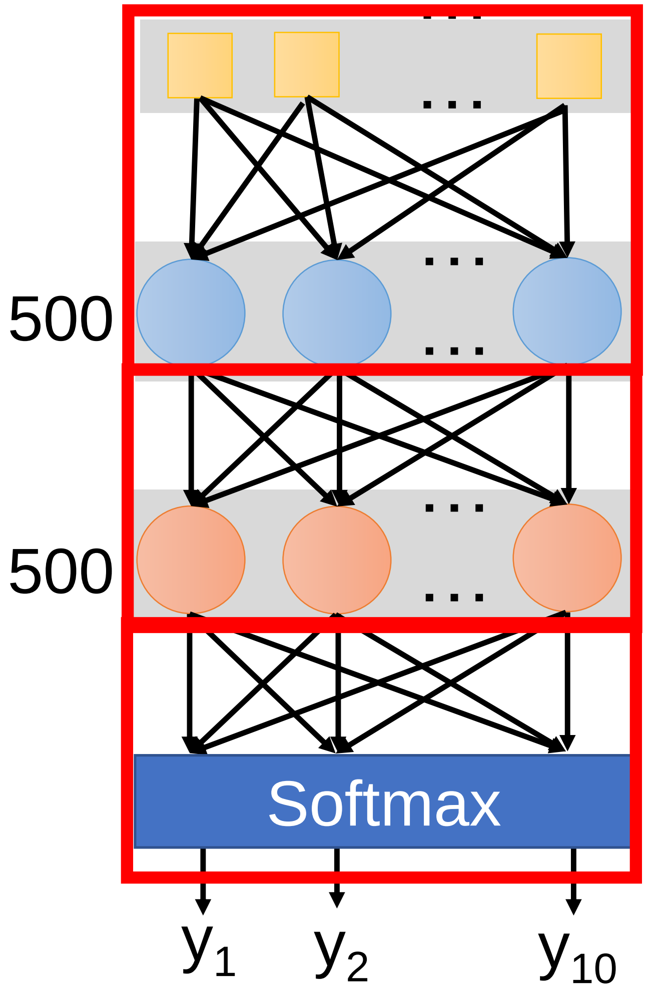
Dropout is a kind of ensemble.



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') )
```

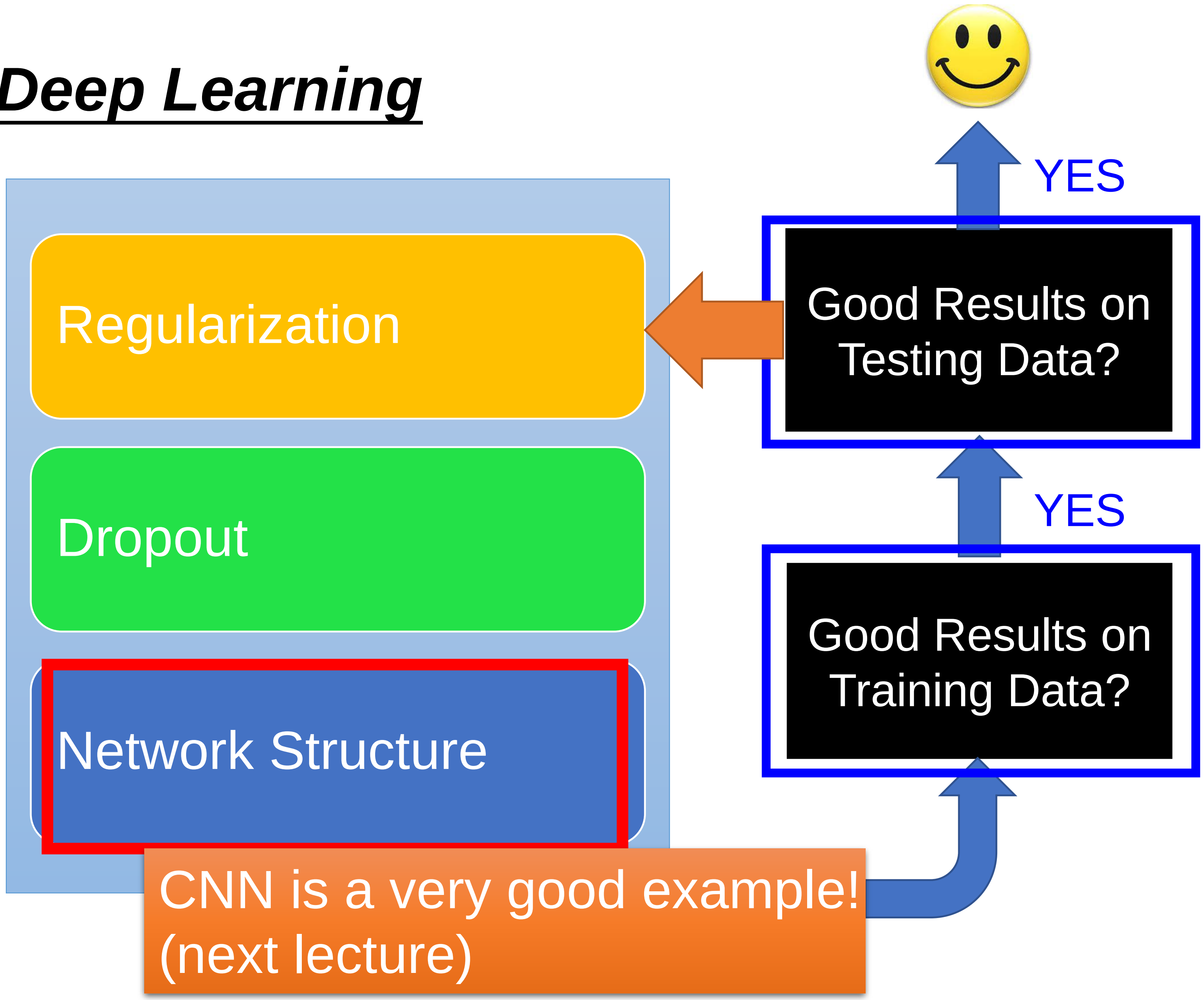
model.add(dropout(0.8))

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

…

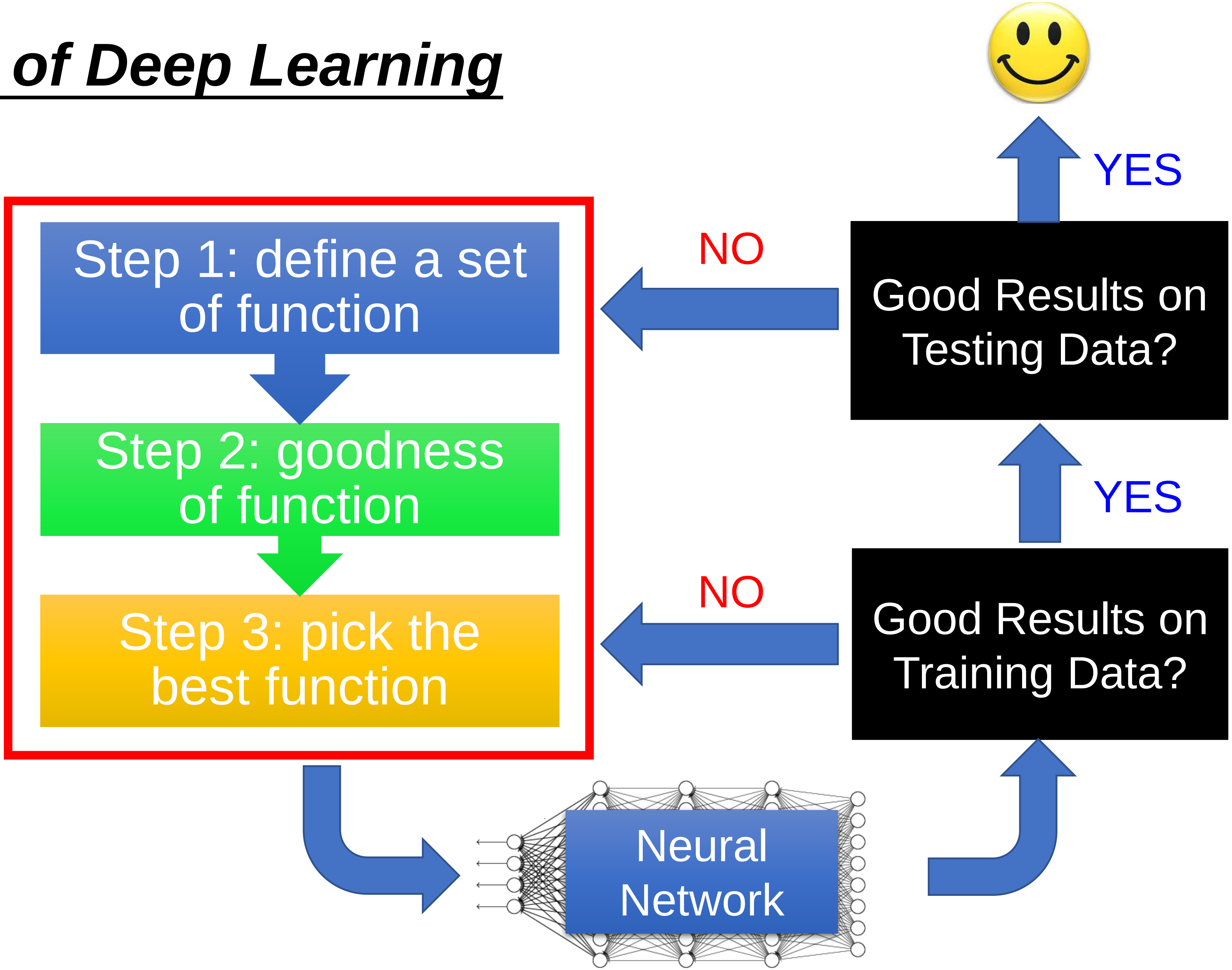
…

Recipe of Deep Learning



Concluding Remarks

Recipe of Deep Learning



Thanks