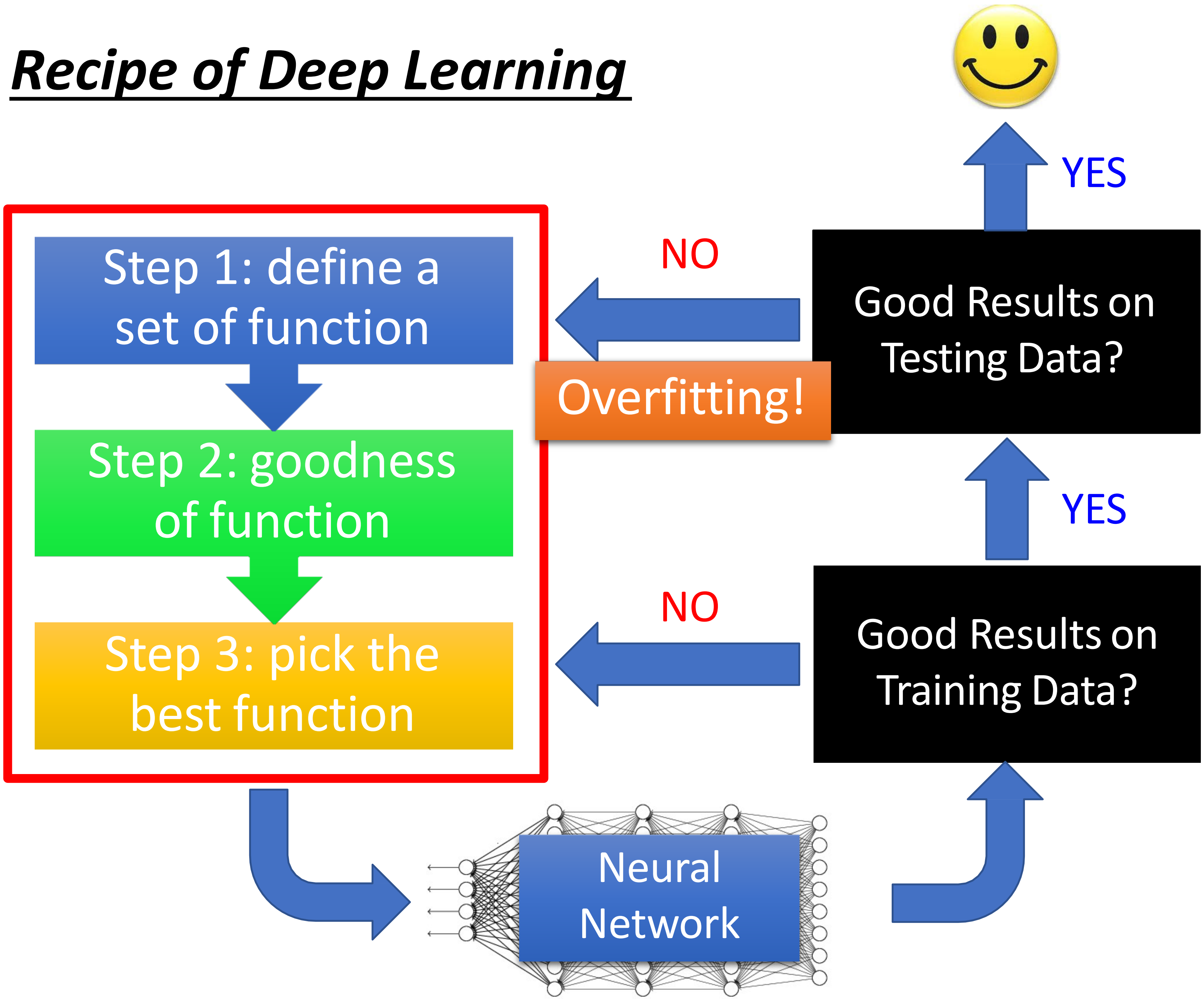
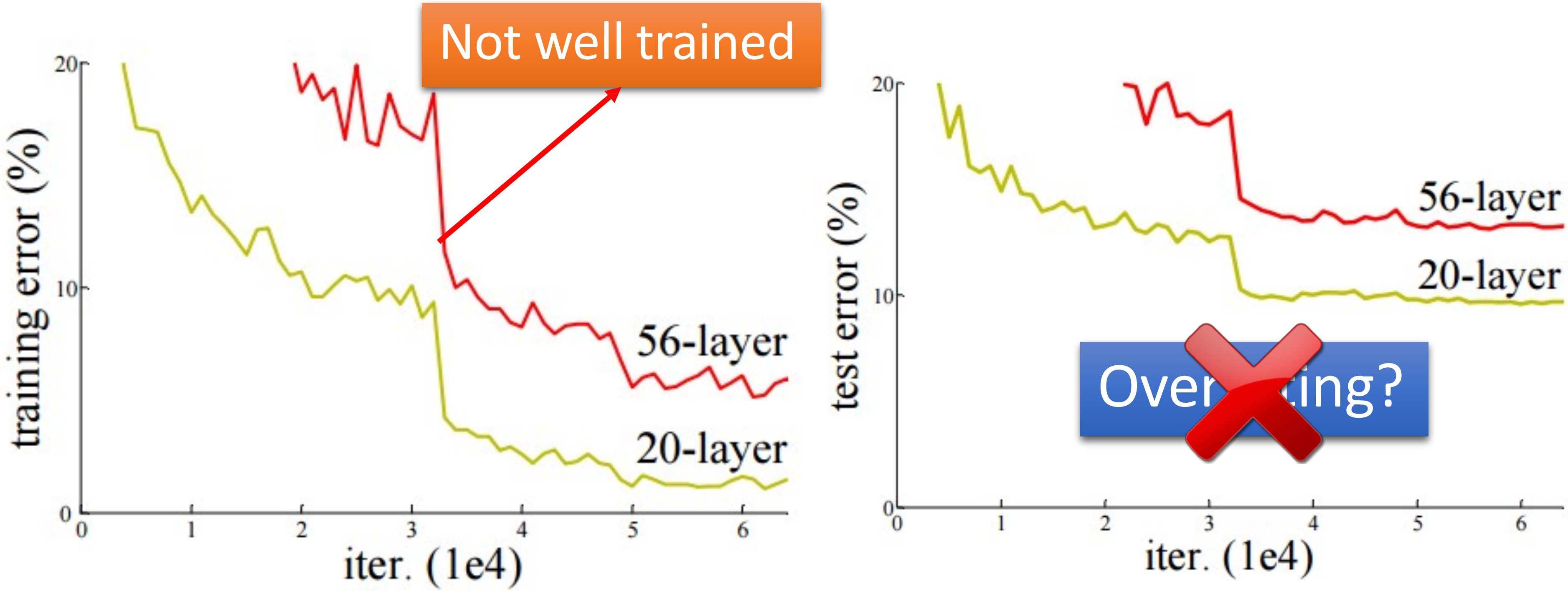
Tips for Training DNN

Slides are provided by Prof. Hung-yi Lee

Recipe of Deep Learning



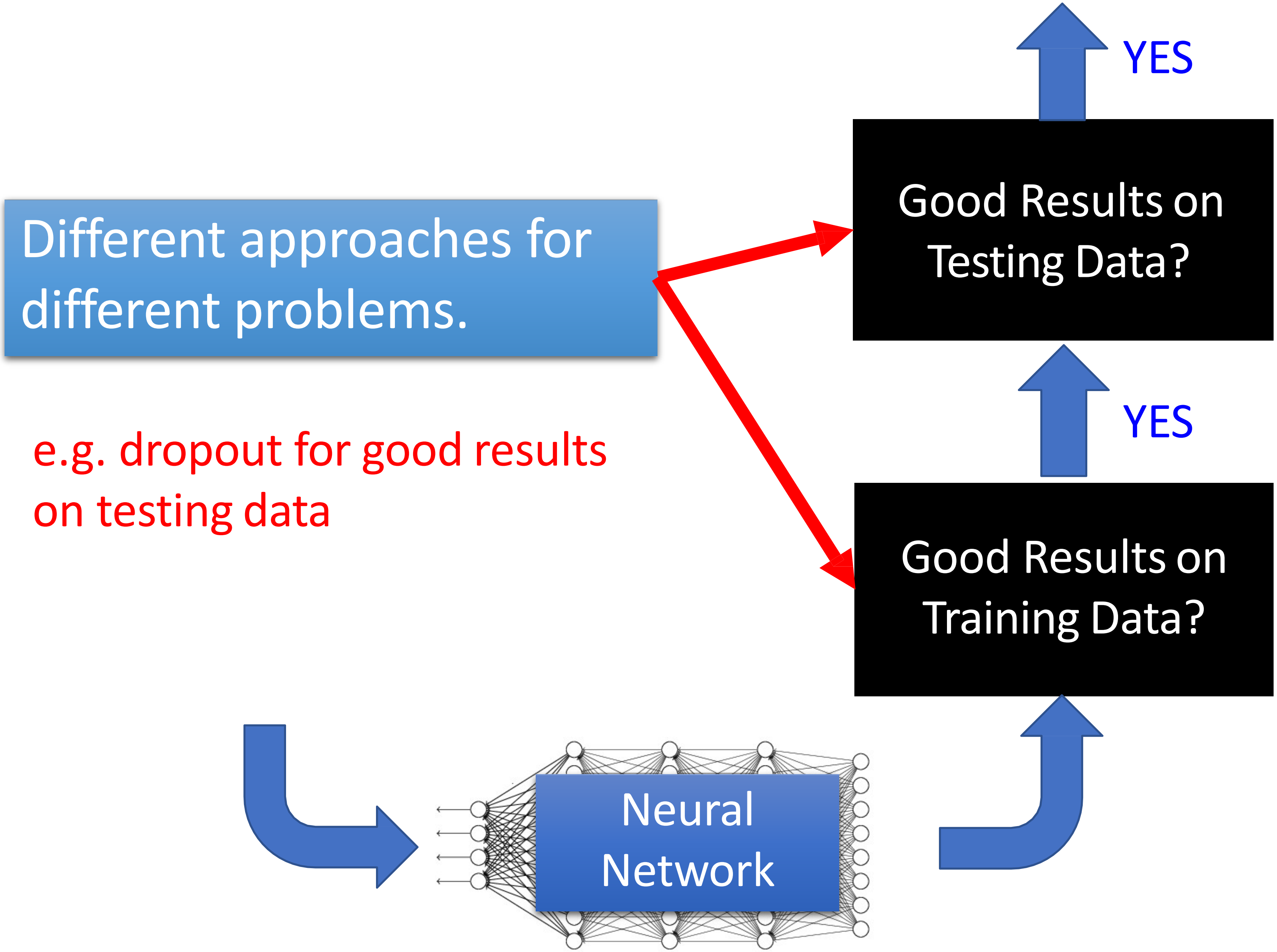
Do not always blame Overfitting



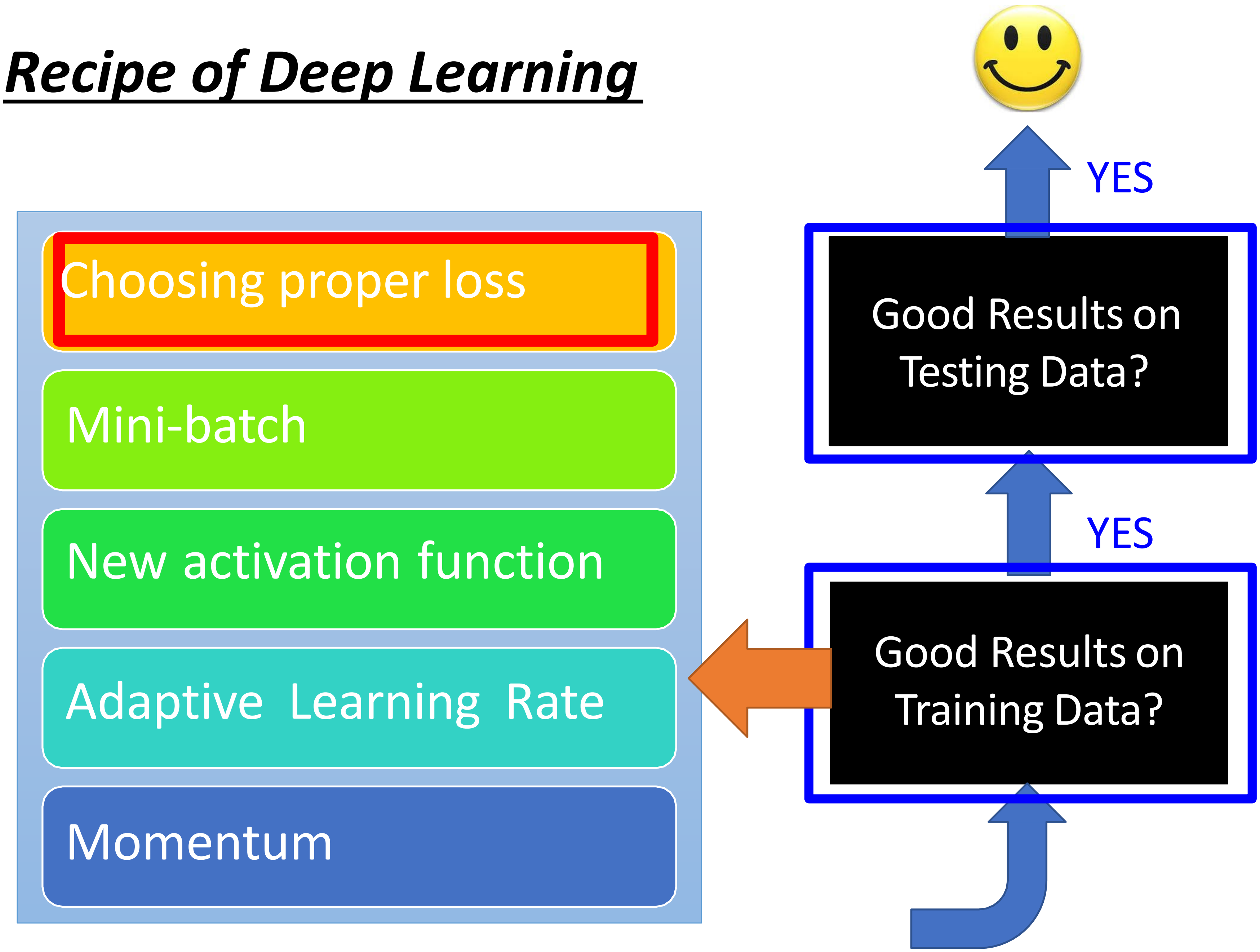
Training Data　　Testing Data

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

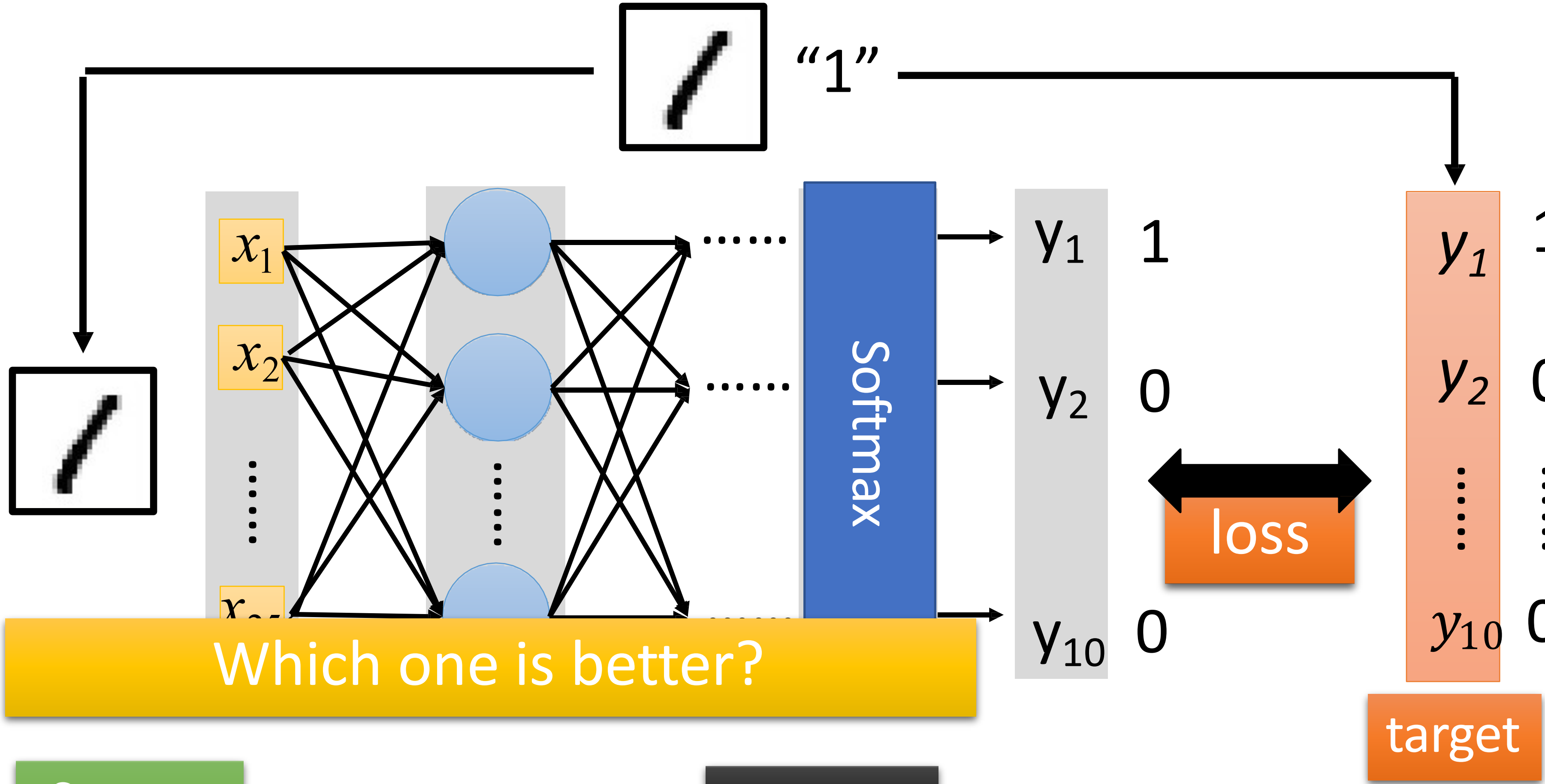
Recipe of Deep Learning



Recipe of Deep Learning



Choosing Proper Loss



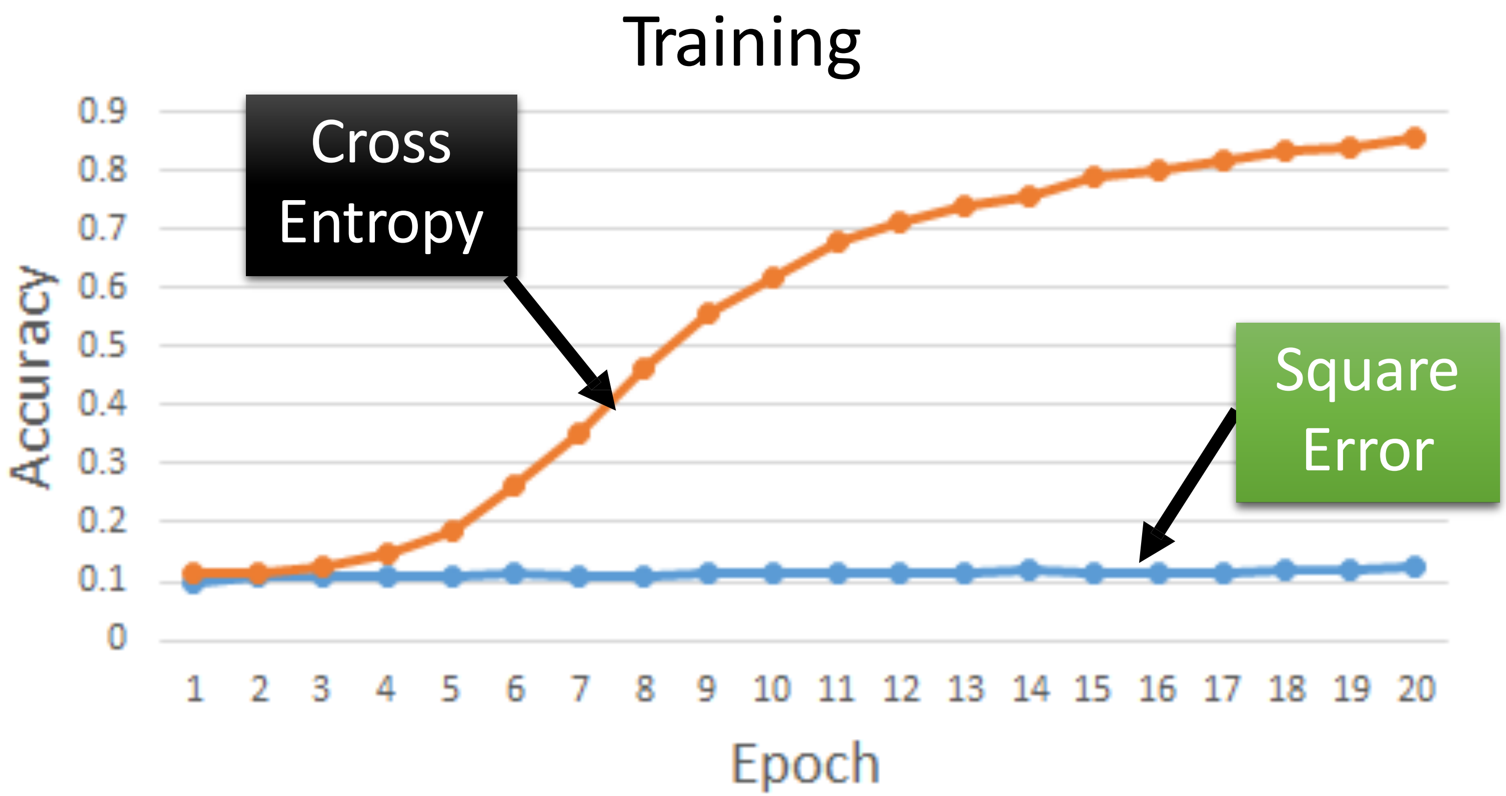
Square Error $\sum_{i=1}^{10}(y_i - \hat{y}_i)^2$

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

Let’s try it

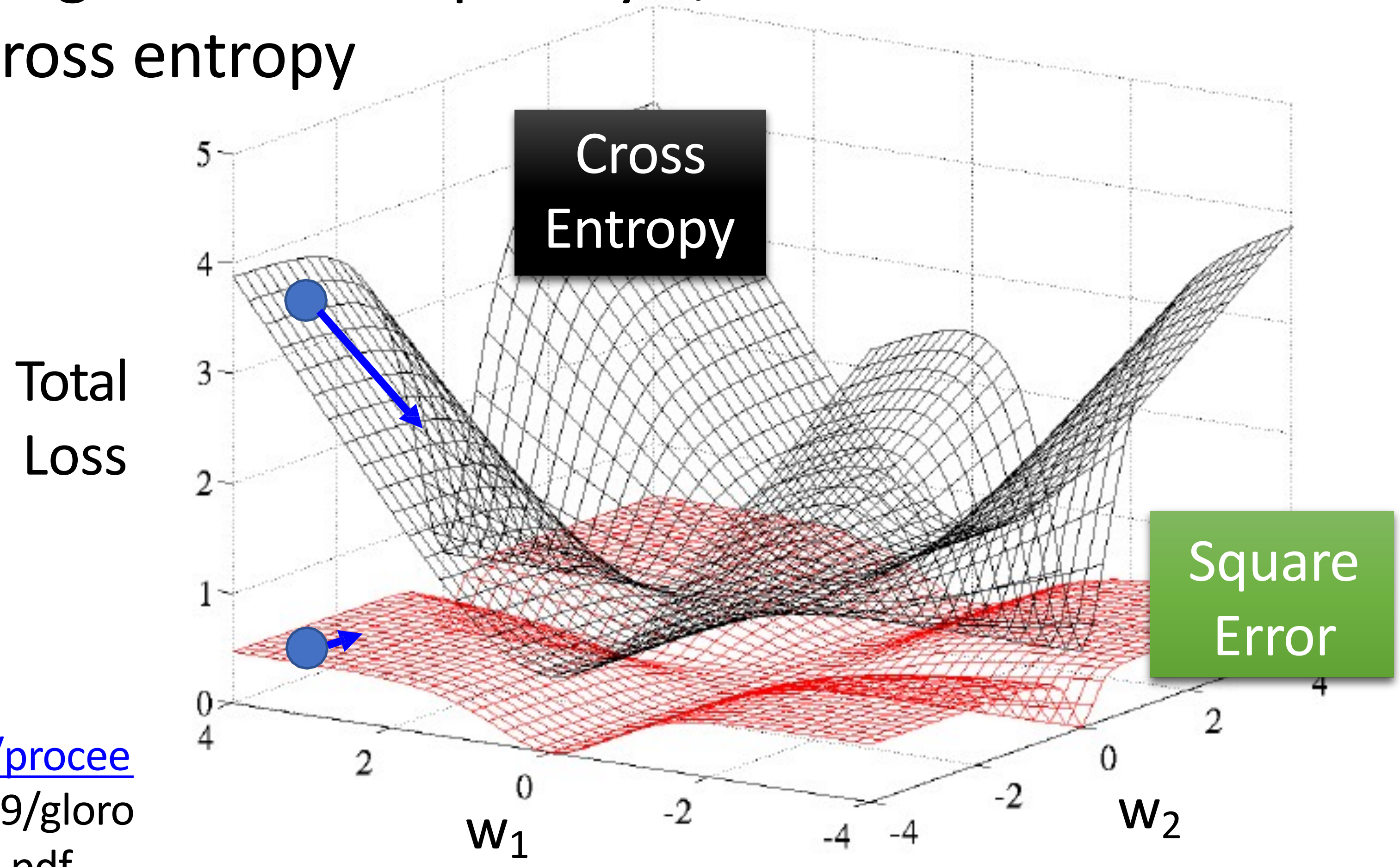
Testing:

	Accuracy
Square Error	0.11
Cross Entropy	0.84



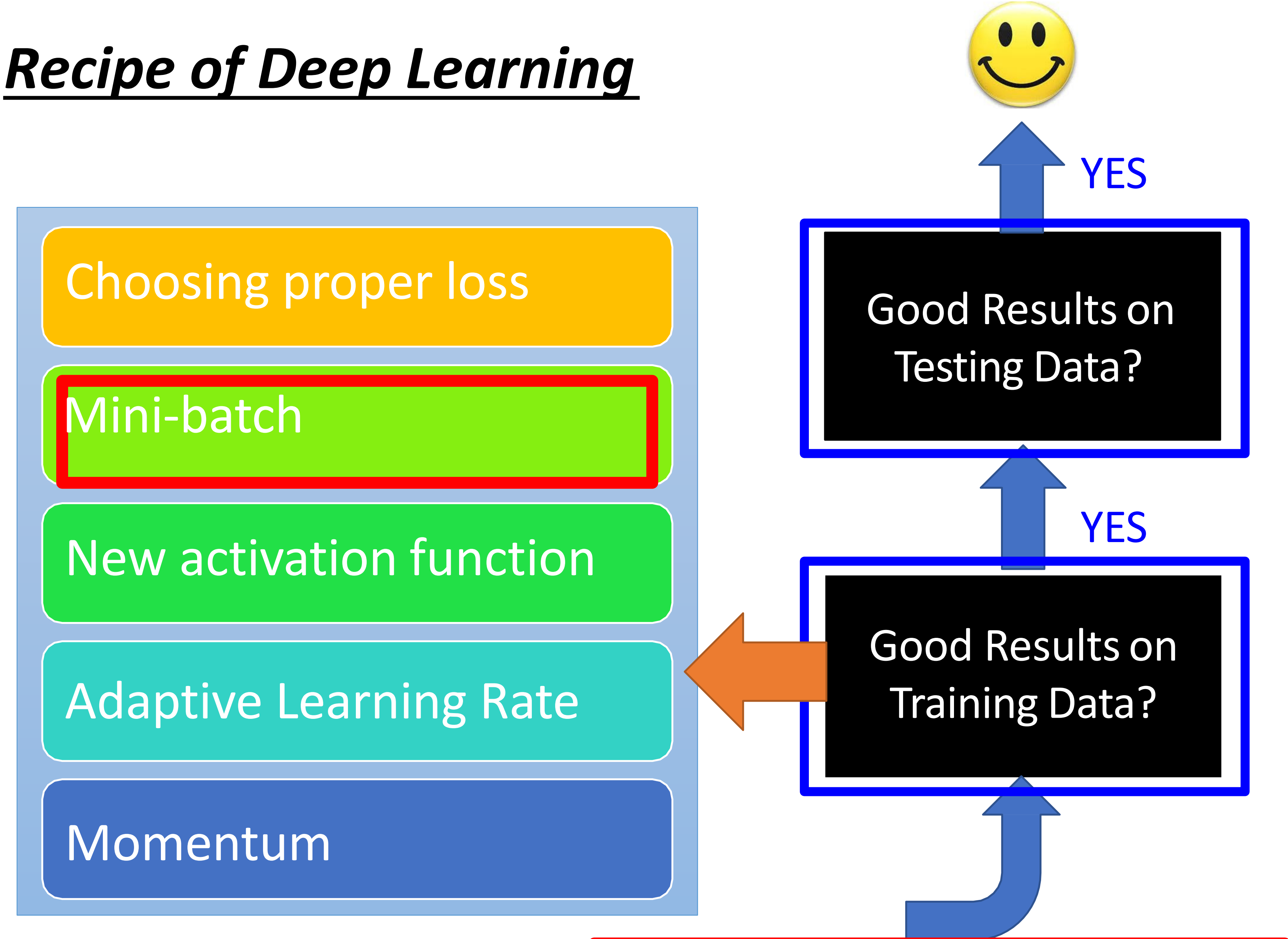
Choosing Proper Loss

When using softmax output layer,
choose cross entropy



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

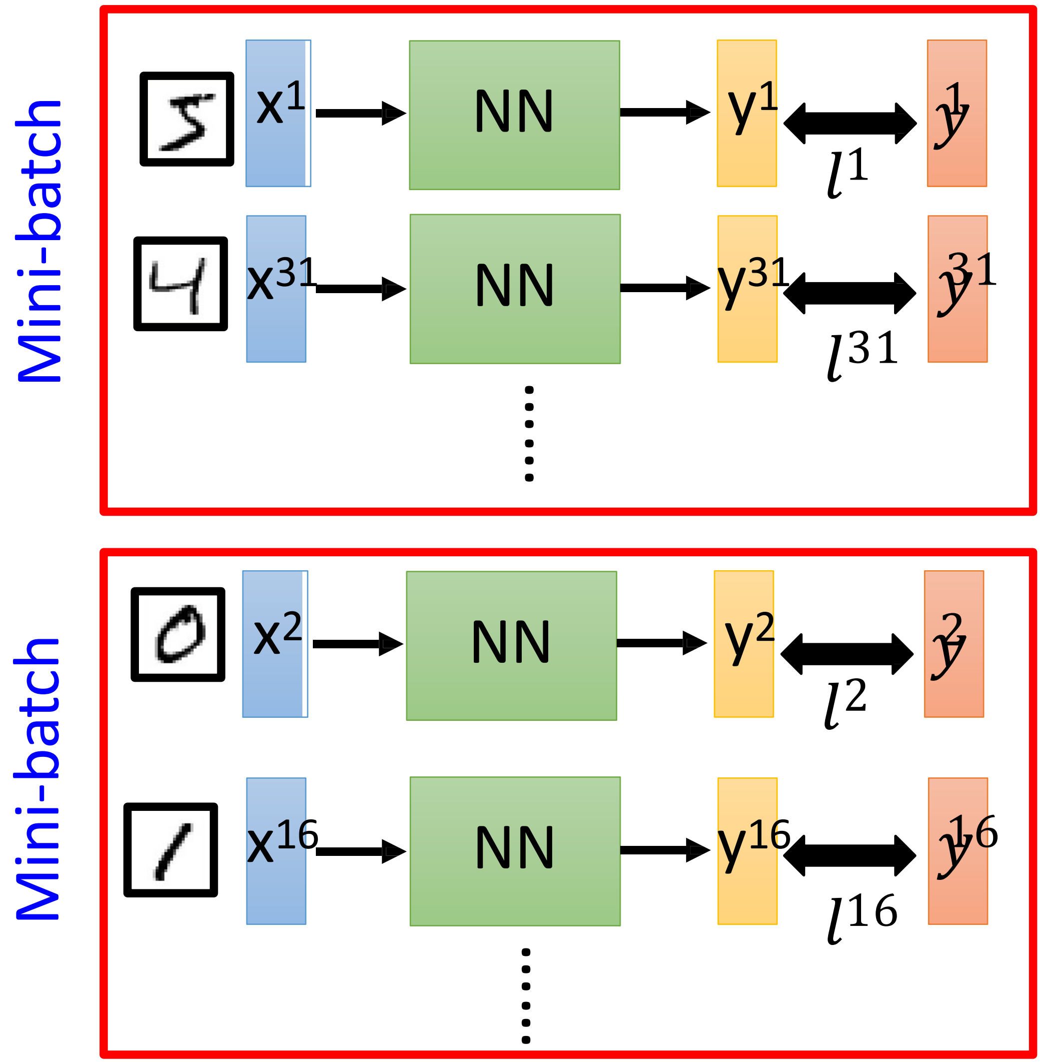
Recipe of Deep Learning




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

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

  ⋮

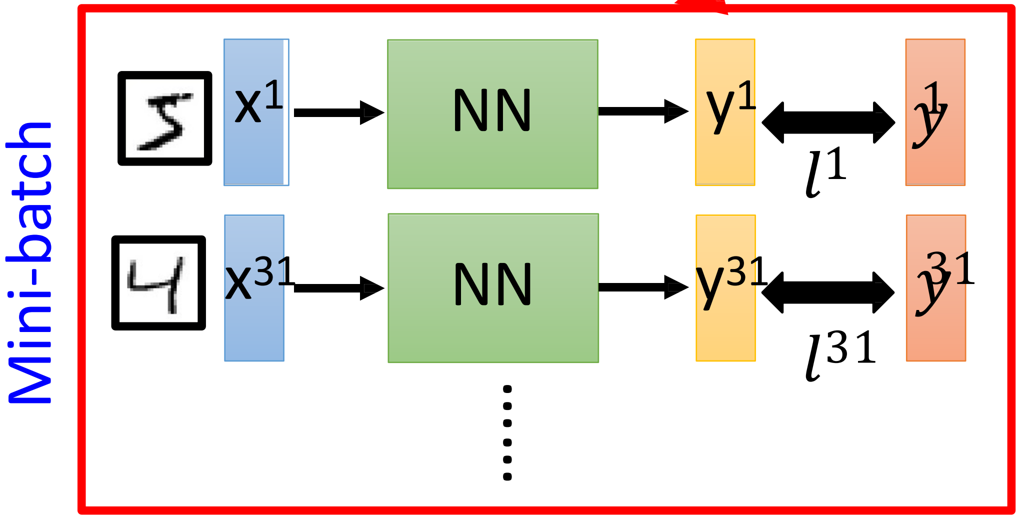
- Until all mini-batches have been picked

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

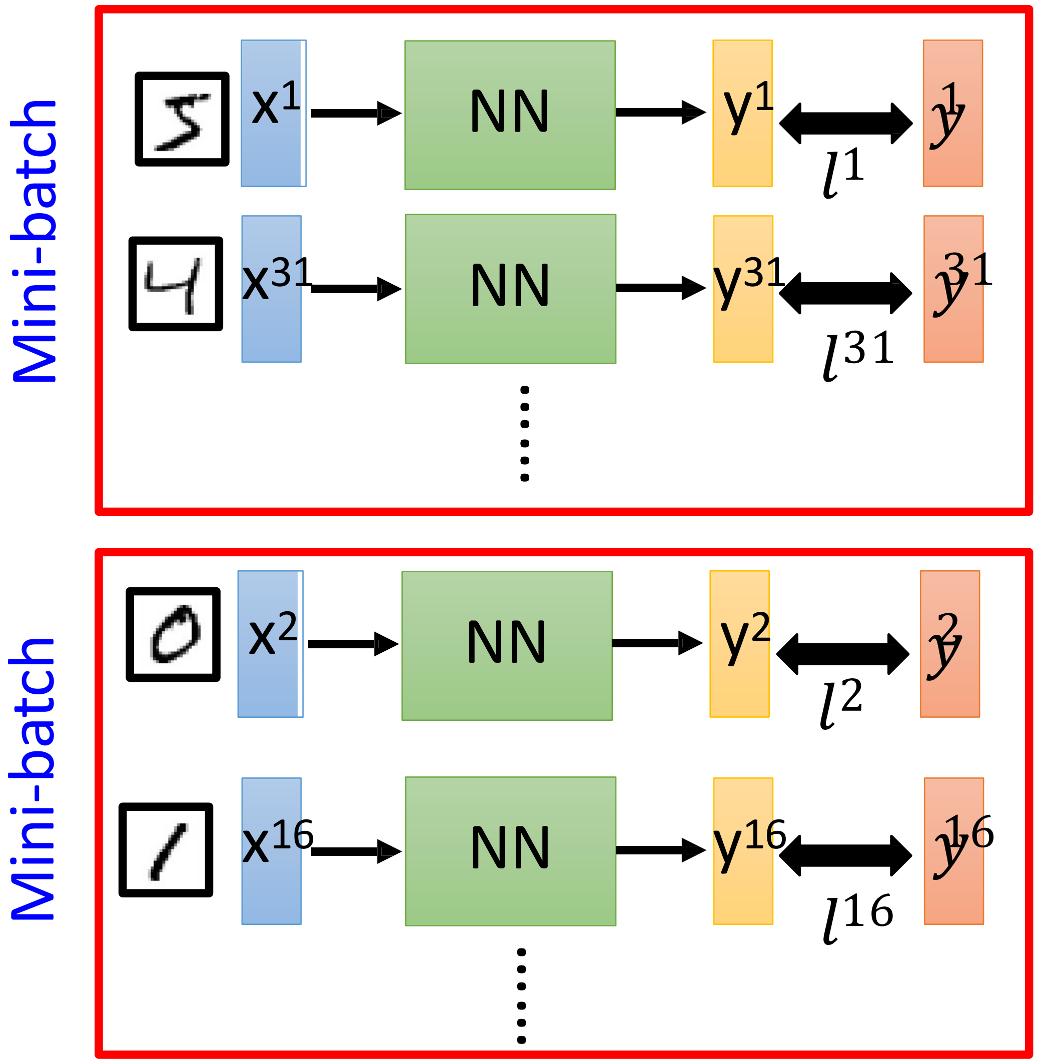
- Pick the 1st batch
  $L' = l^1 + l^{31} + \cdots$
  Update parameters once
- Pick the 2nd batch
  $L'' = l^2 + l^{16} + \cdots$
  Update parameters once
  ⋮
- Until all mini-batches have been picked

one epoch

Repeat 20 times

We do not really minimize total loss!

Mini-batch

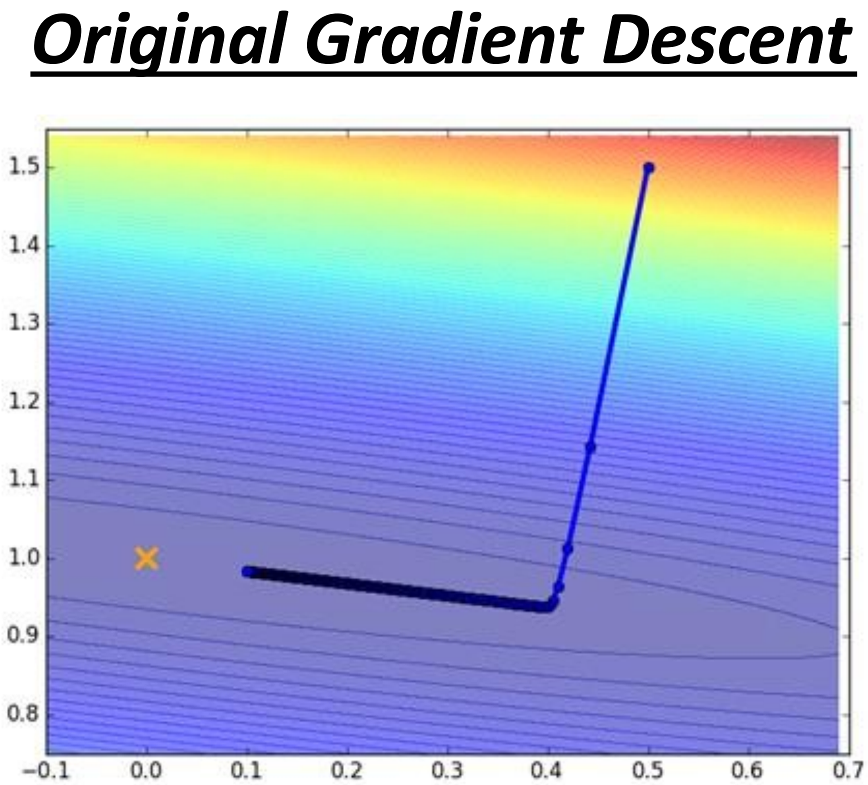


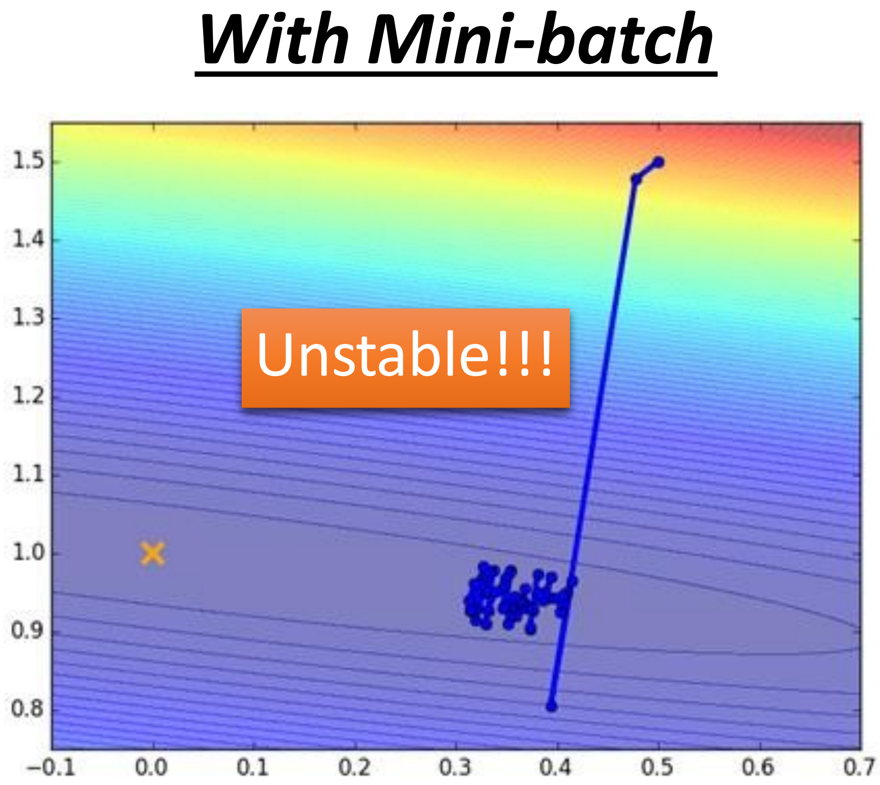
- Randomly initialize network parameters
- Pick the 1st batch
  $L' = l^1 + l^{31} + \cdots$
  Update parameters once
- Pick the 2nd batch
  $L'' = l^2 + l^{16} + \cdots$
  Update parameters once

  ⋮

L is different each time when we update parameters!

Mini-batch





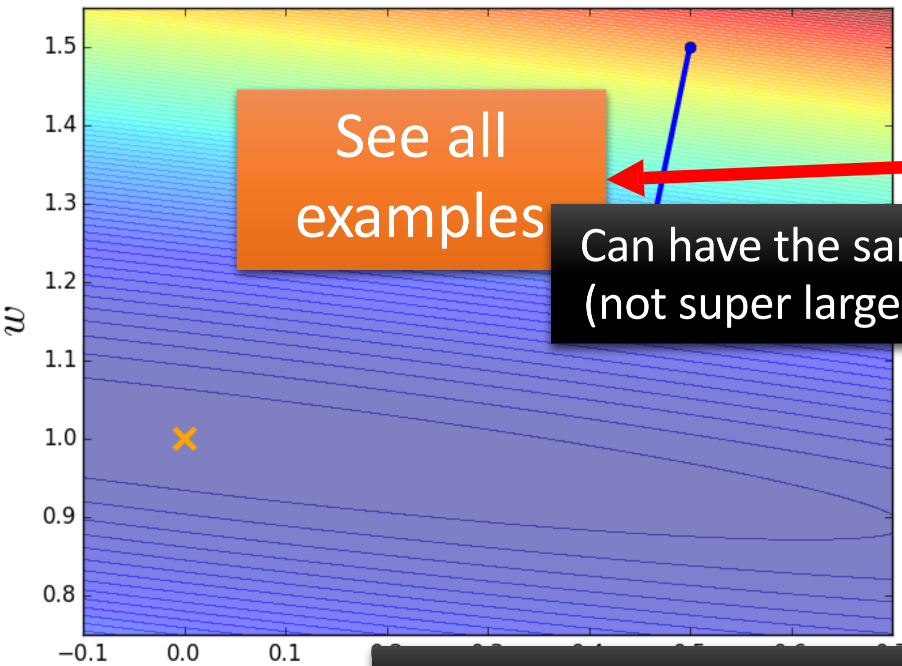
The colors represent the total loss.

Mini-batch is Faster

Not always true with parallel computing.

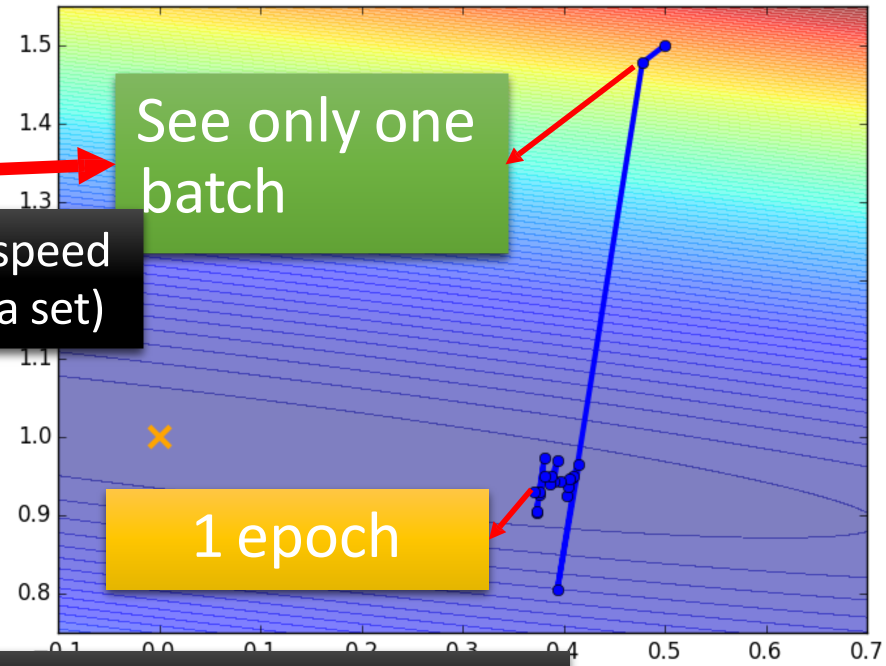
Original Gradient Descent

Update after seeing all examples



With Mini-batch

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



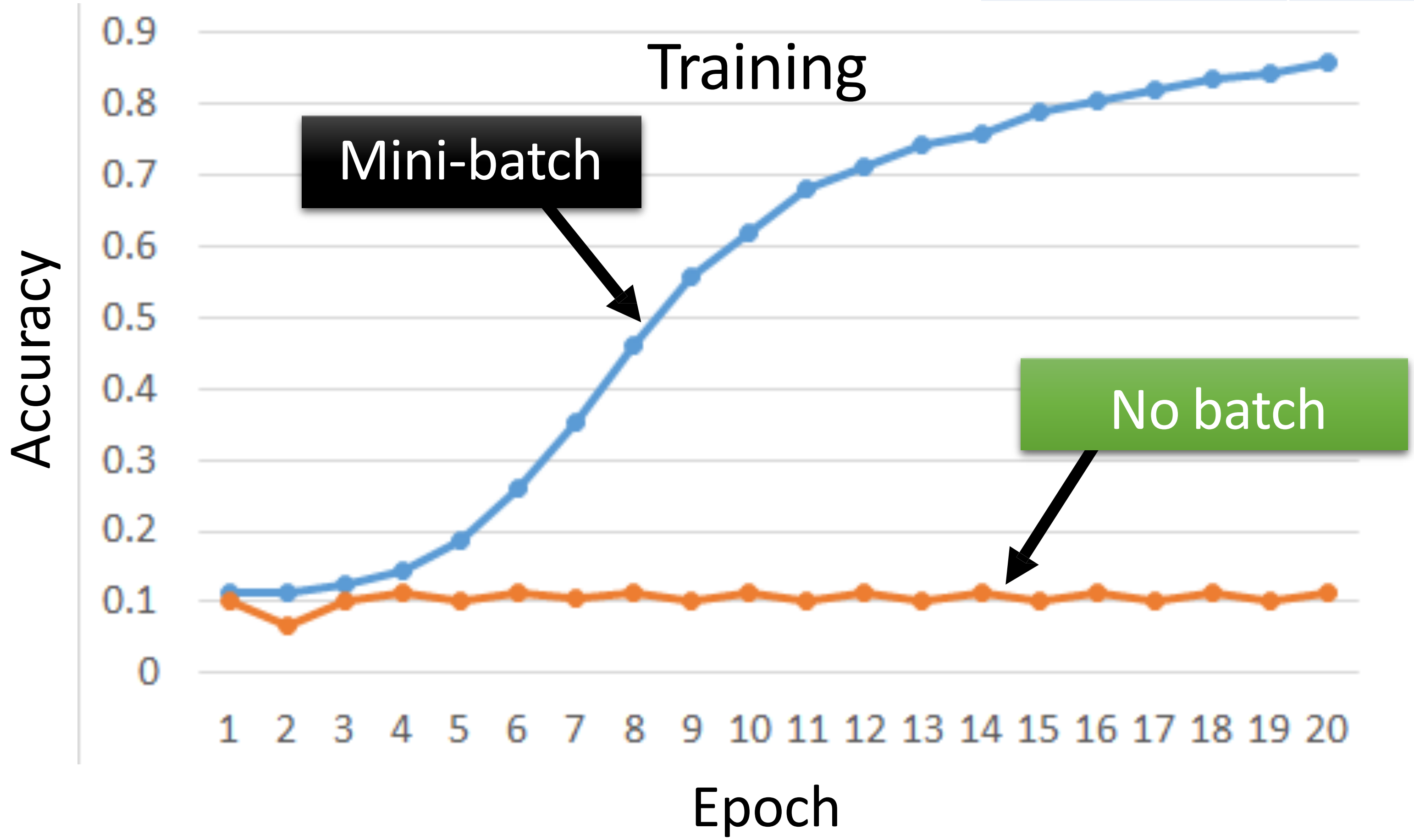
Can have the same speed (not super large data set)

Mini-batch has better performance!

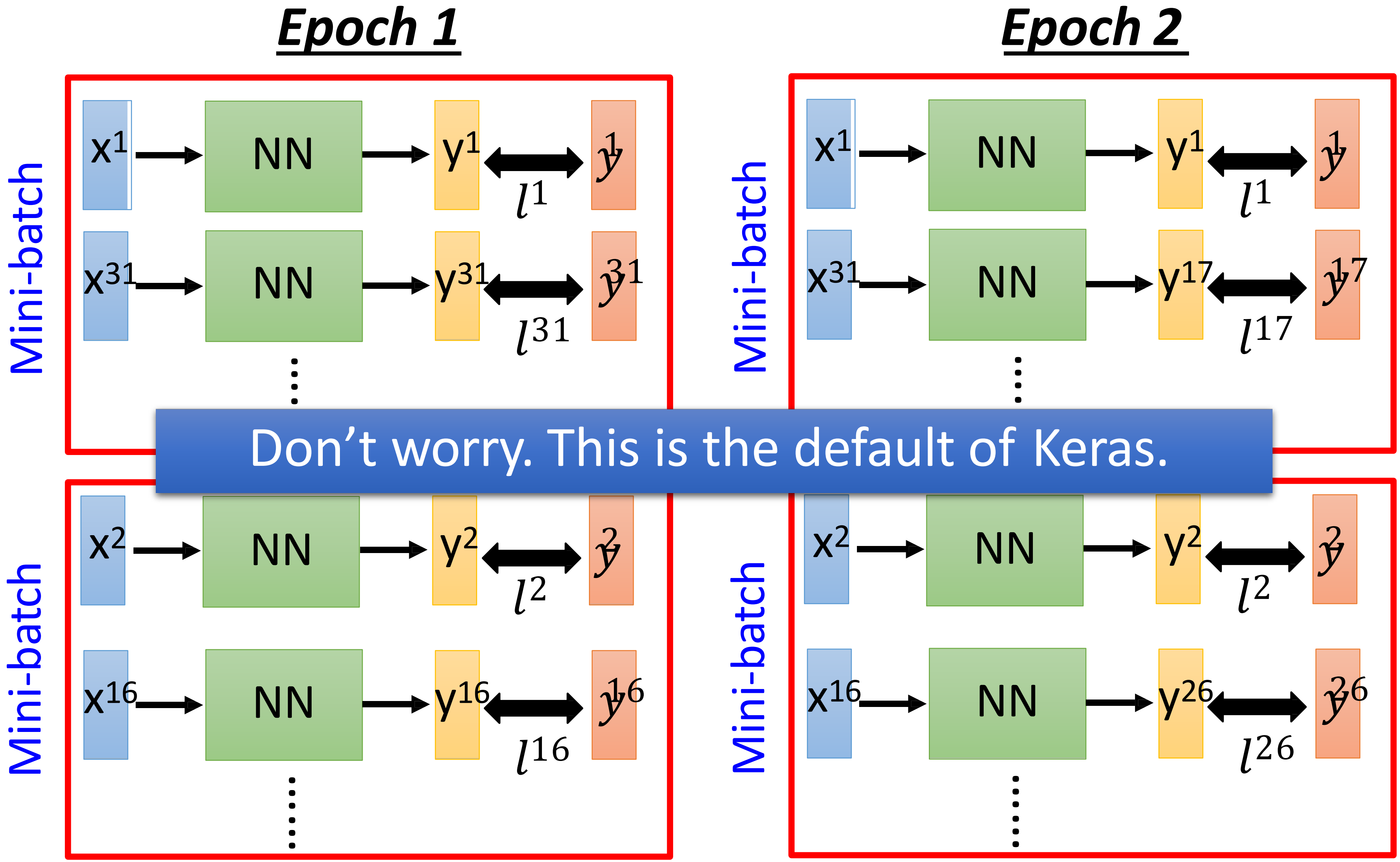
Mini-batch is Better!

Testing:

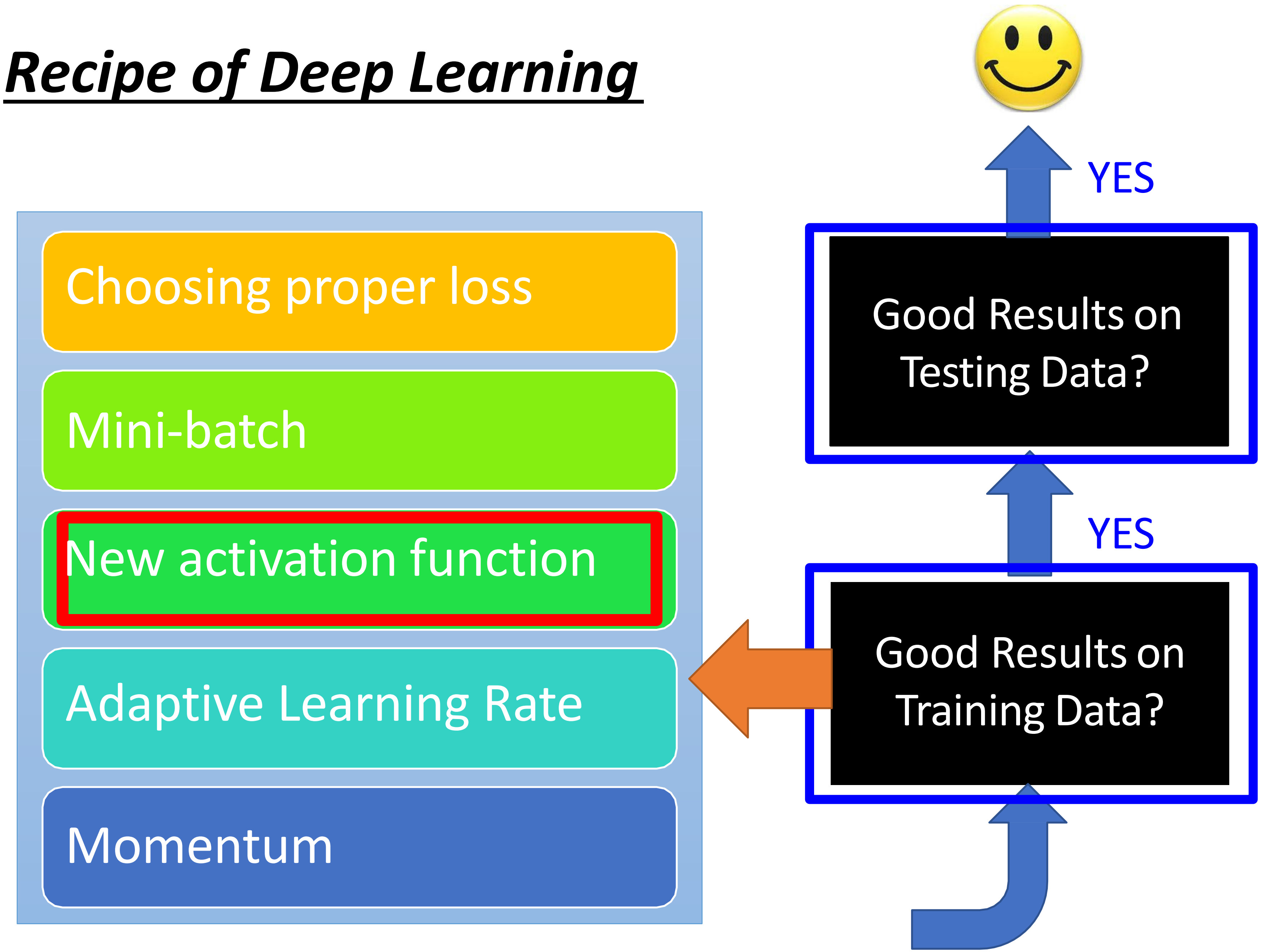
	Accuracy
Mini-batch	0.84
No batch	0.12



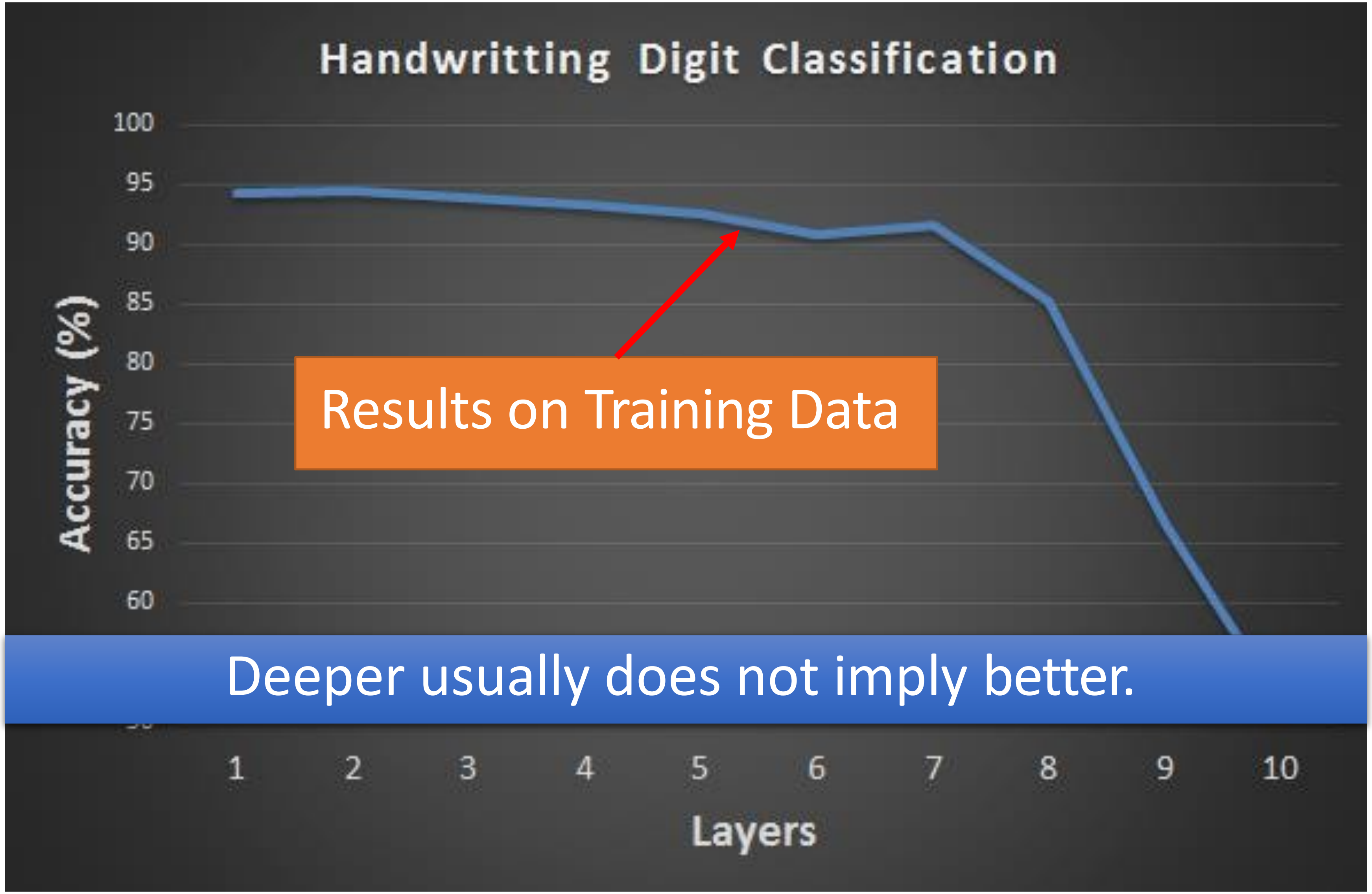
Shuffle the training examples for each epoch



Recipe of Deep Learning



Hard to get the power of Deep …

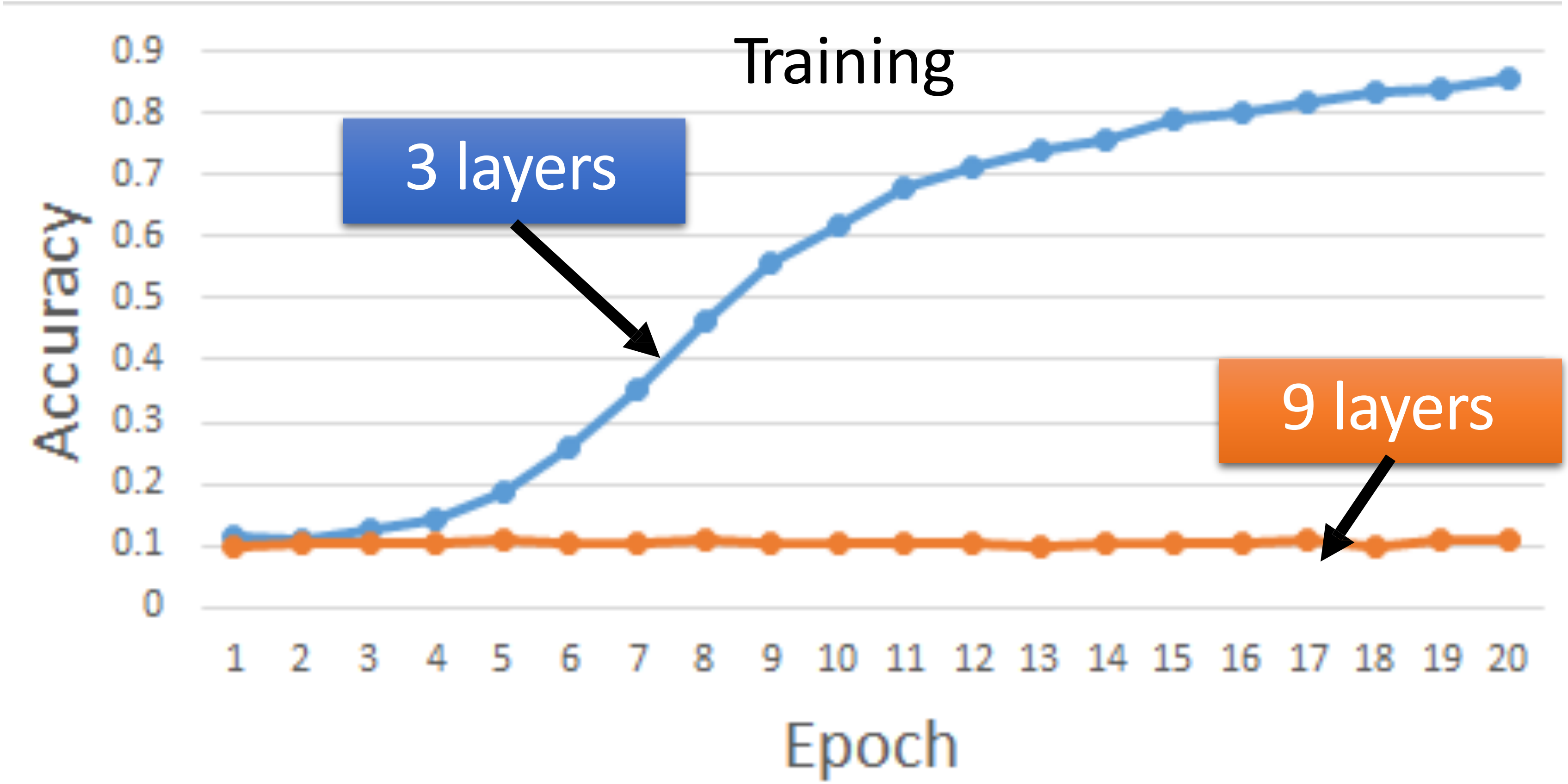


Deeper usually does not imply better.

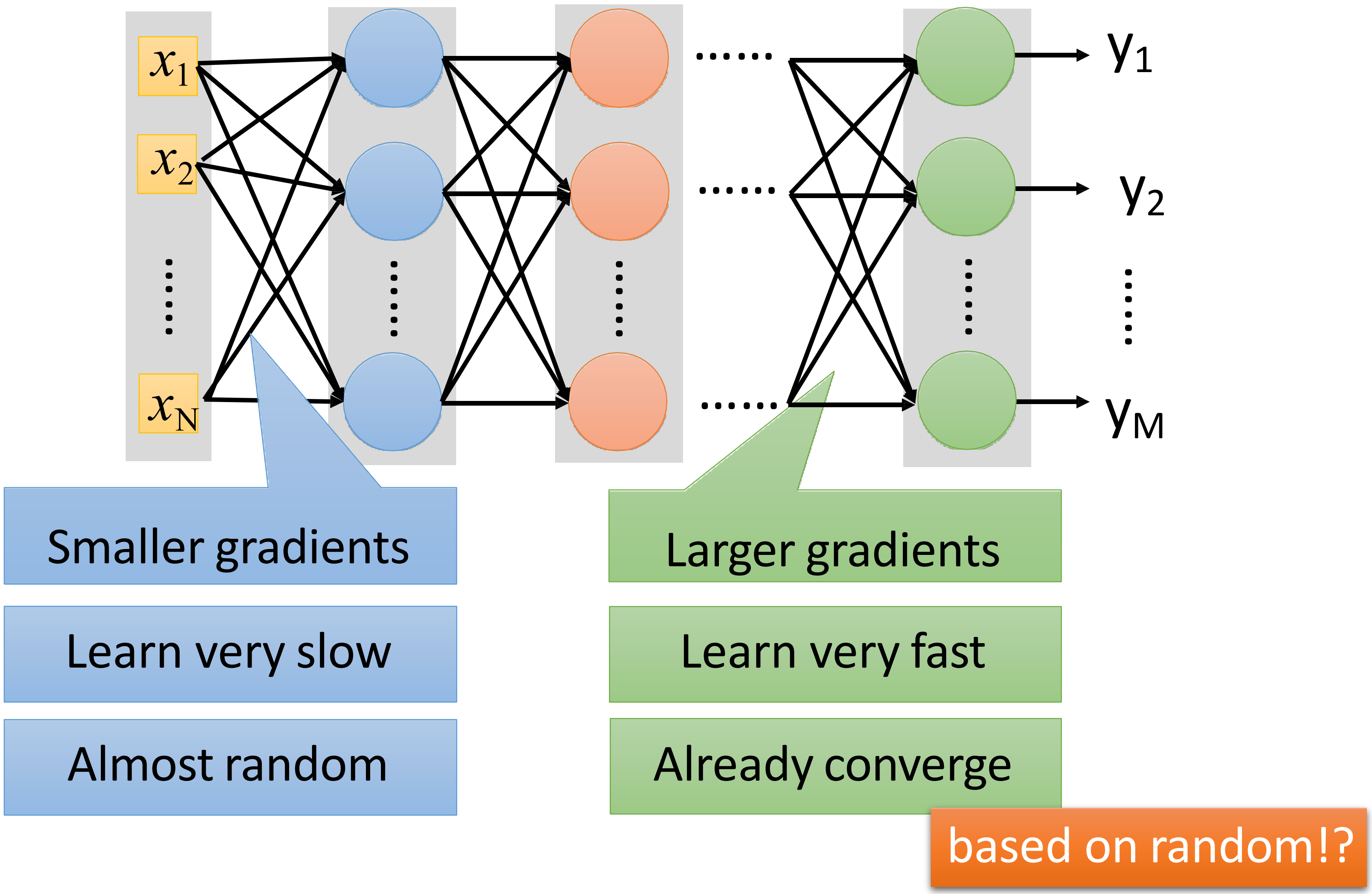
Let's try it

Testing:

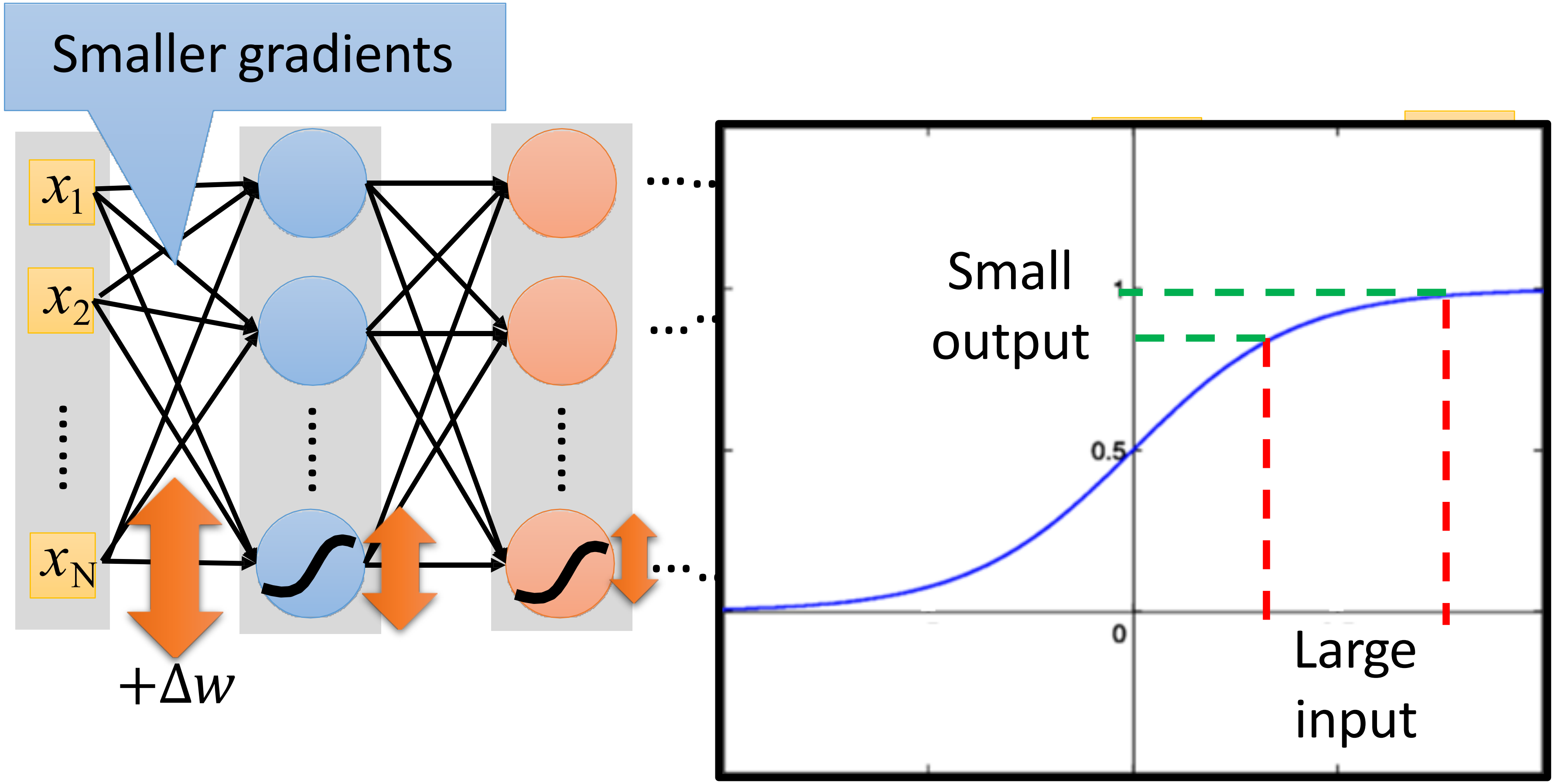
	Accuracy
3 layers	0.84
9 layers	0.11



Vanishing Gradient Problem



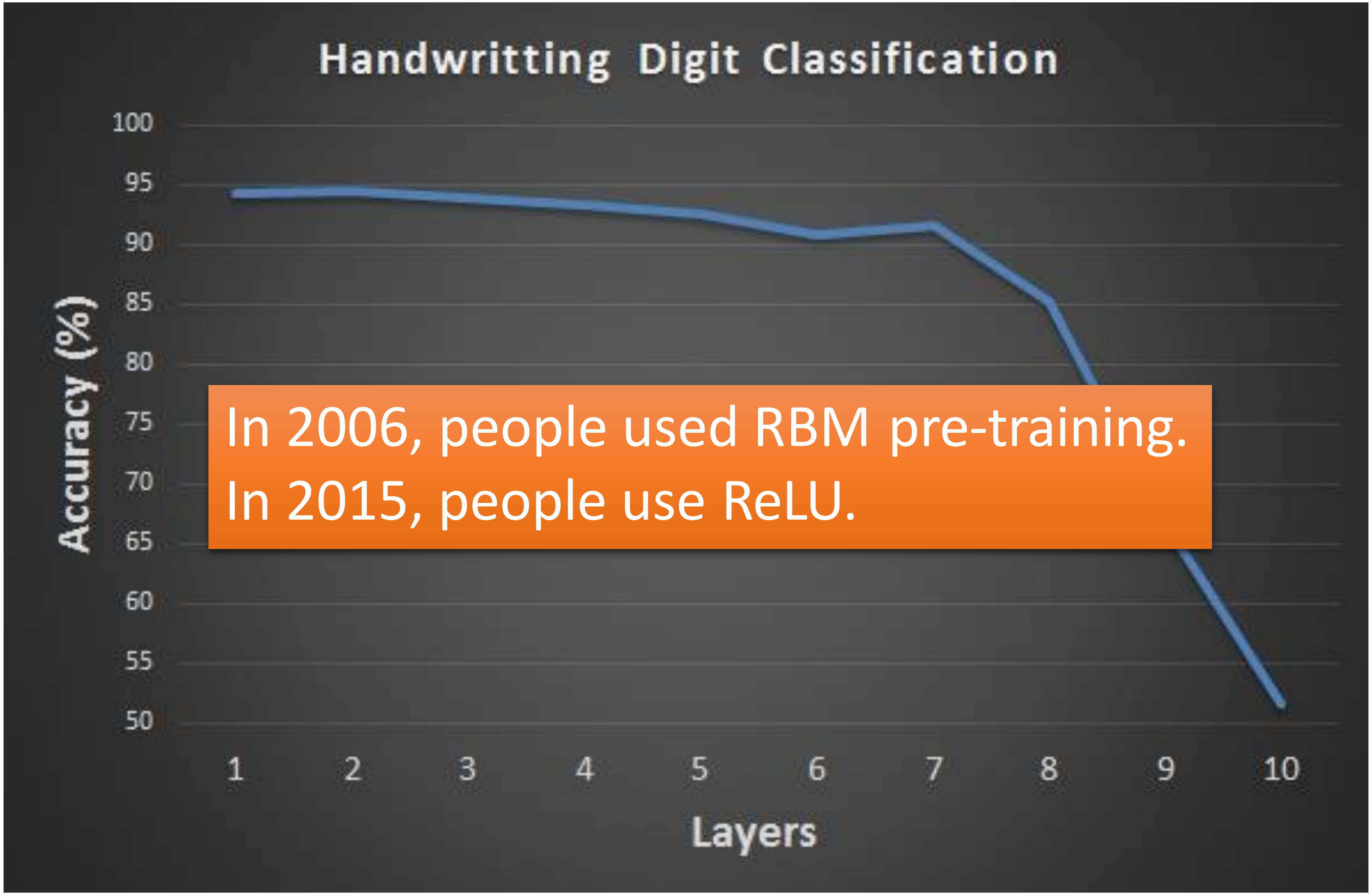
Vanishing Gradient Problem



Intuitive way to compute the derivatives …

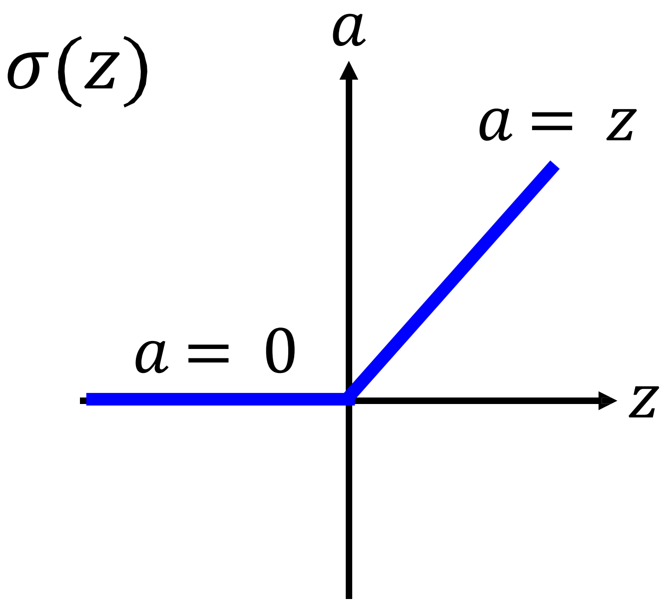
$$\frac{\partial l}{\partial w} =? \frac{\Delta l}{\Delta w}$$

Hard to get the power of Deep …



ReLU

- Rectified Linear Unit (ReLU)

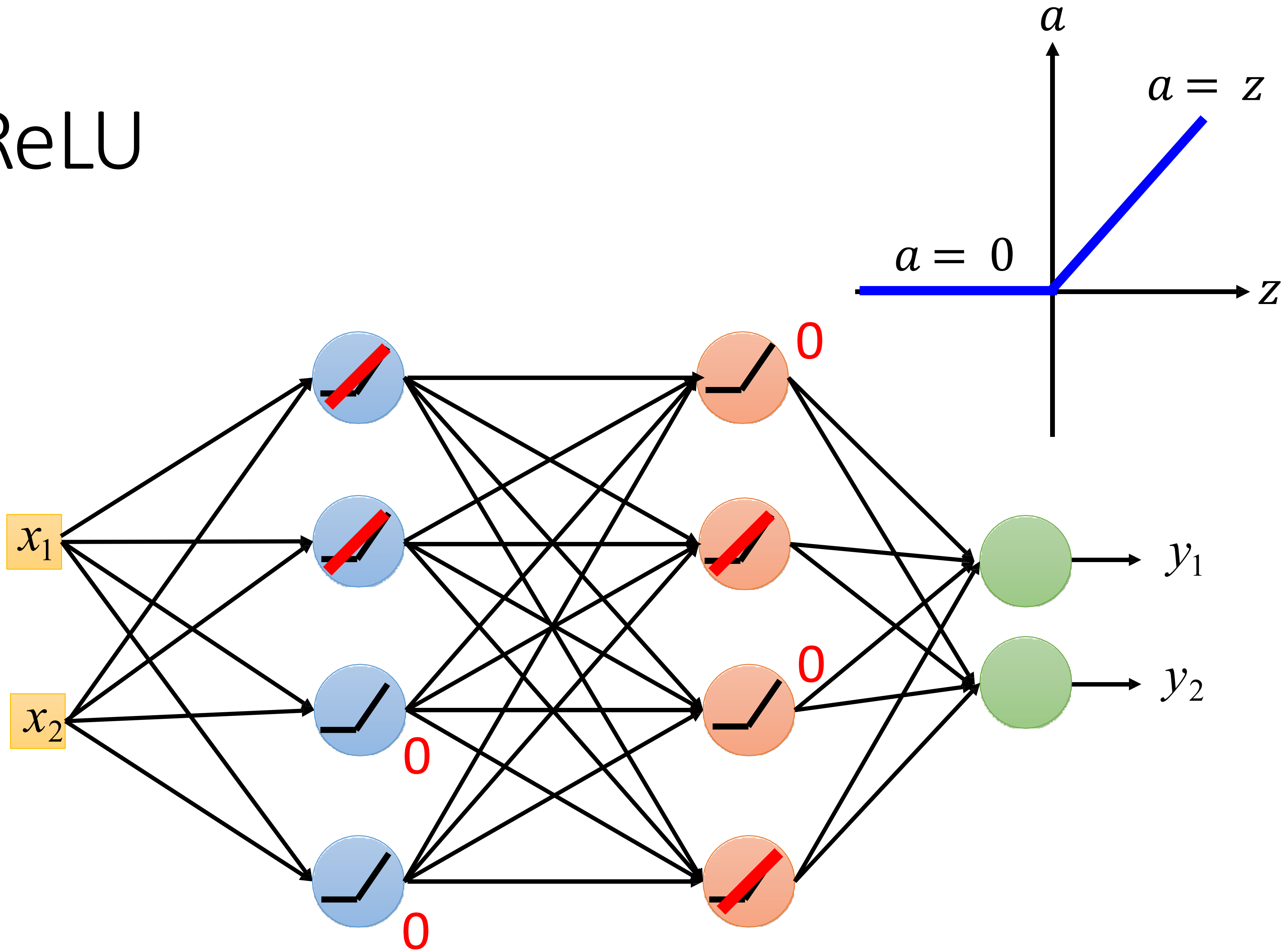


[Xavier Glorot, AISTATS'11]
[Andrew L. Maas, ICML'13]
[Kaiming He, arXiv'15]

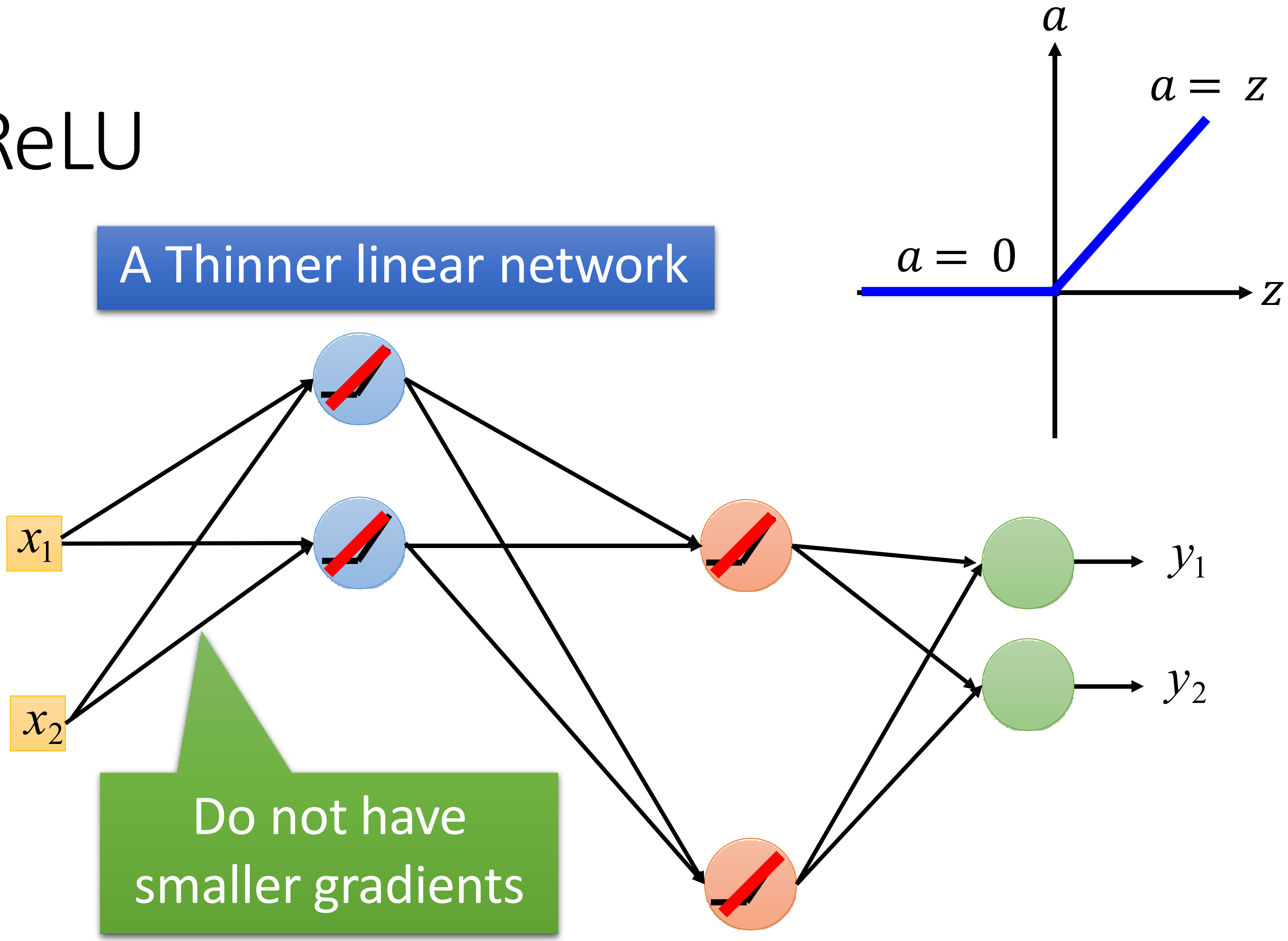
Reason:

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

ReLU



ReLU



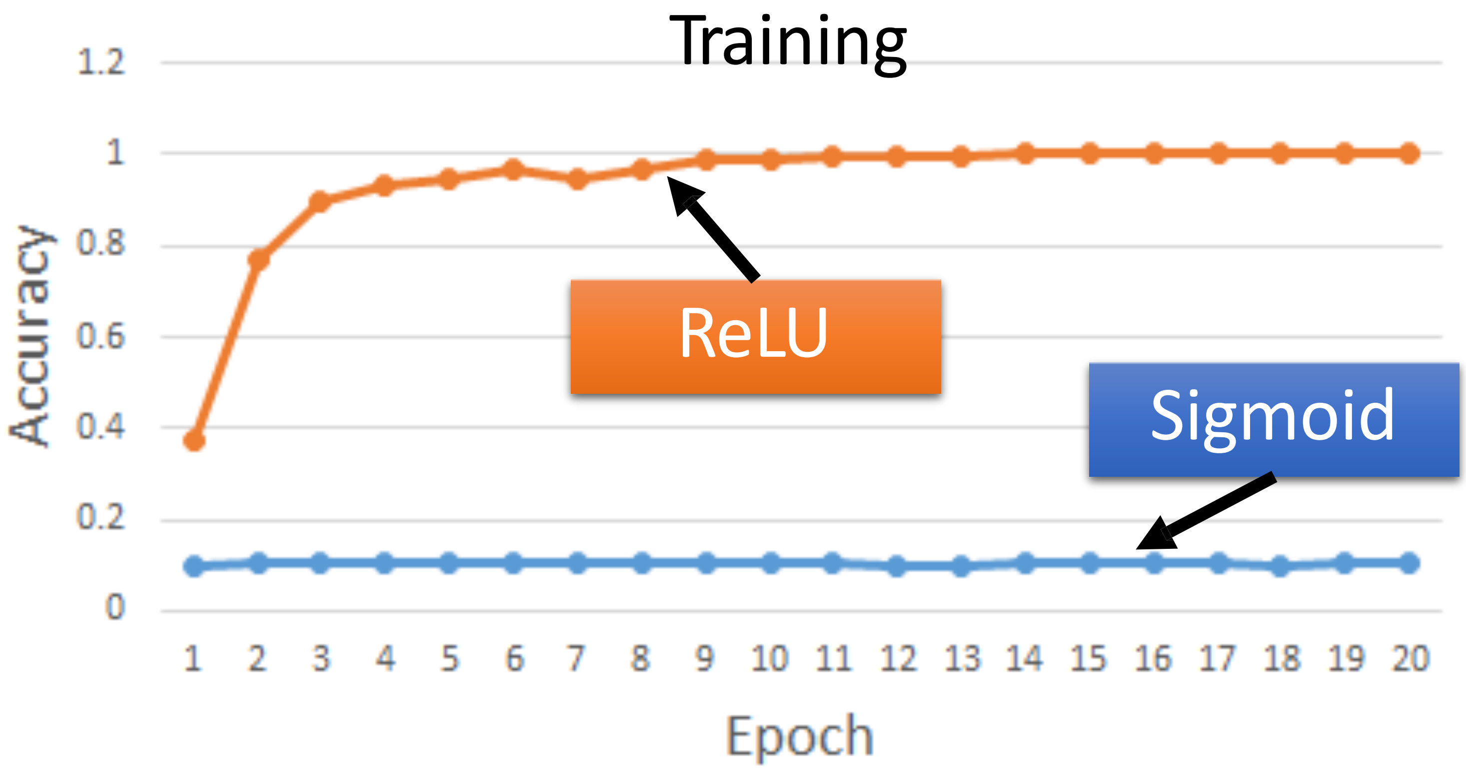
A Thinner linear network
a
$a = z$
$a = 0$
z
x_1
x_2
y_1
y_2
Do not have smaller gradients

Let's try it

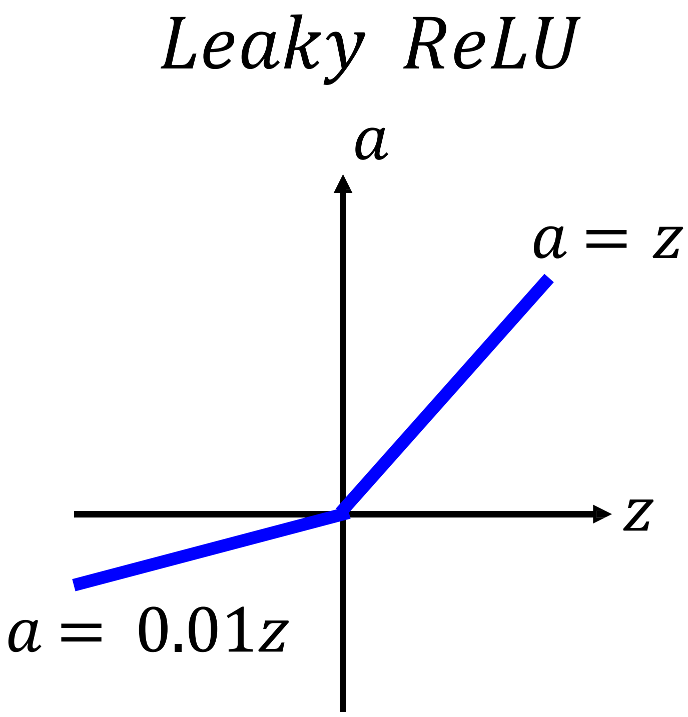
Testing:

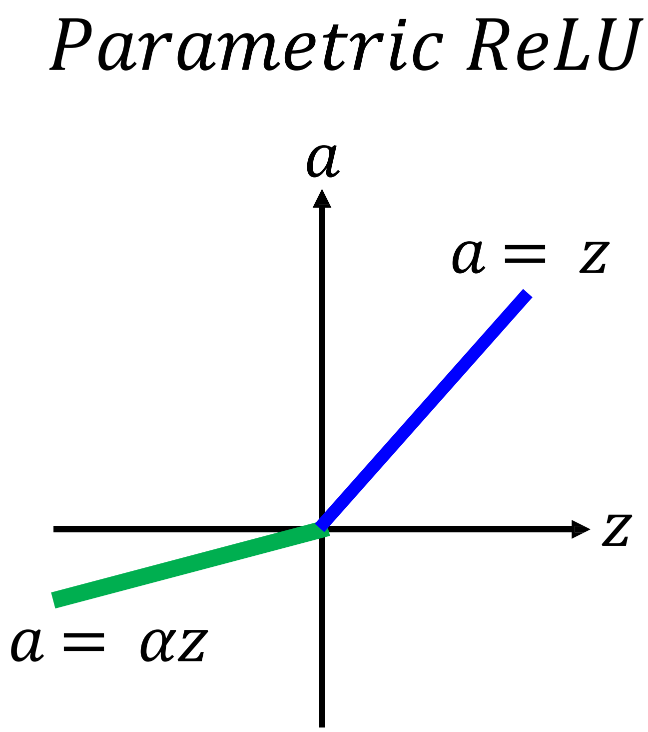
9 layers	Accuracy
Sigmoid	0.11
ReLU	0.96

- 9 layers



ReLU - variant



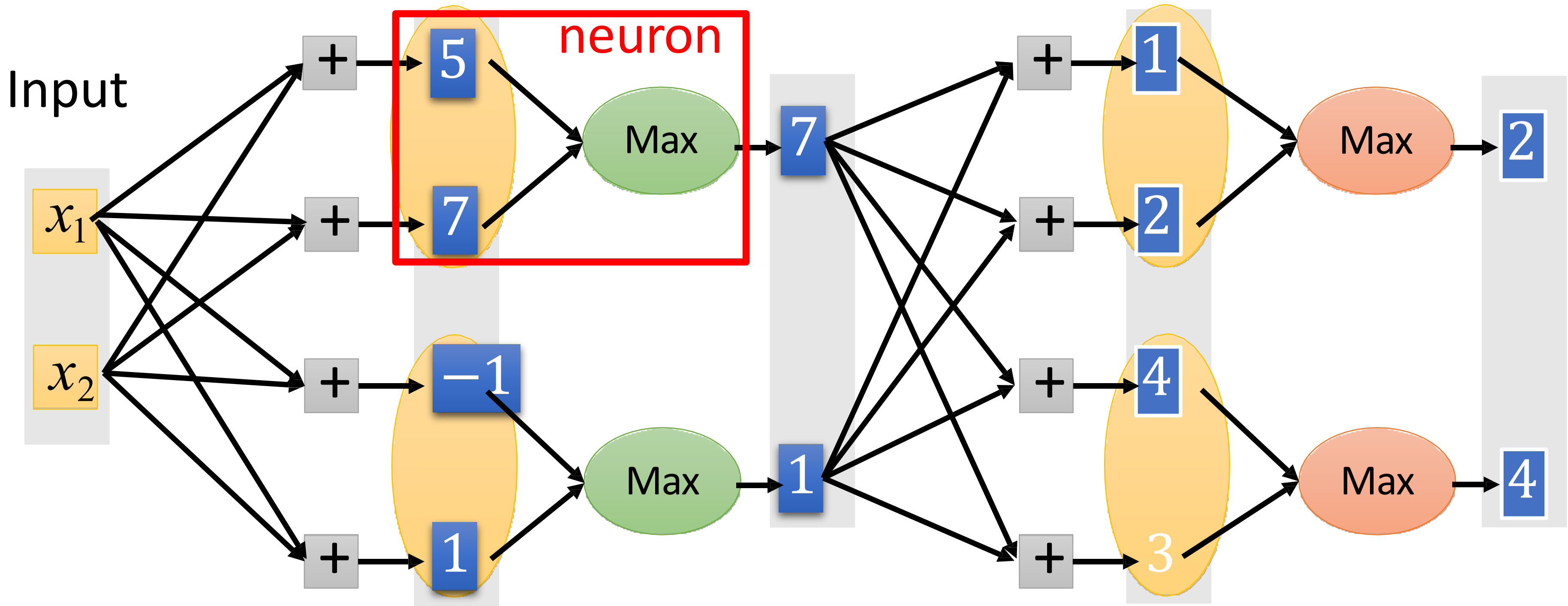


α also learned by gradient descent

Maxout

ReLU is a special case of Maxout

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



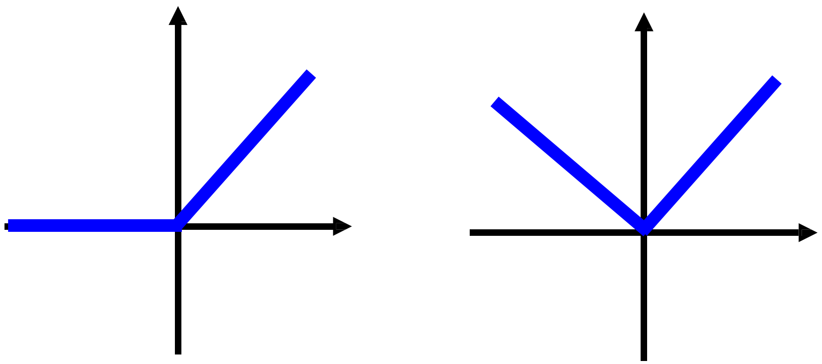
You can have more than 2 elements in a group.

Maxout

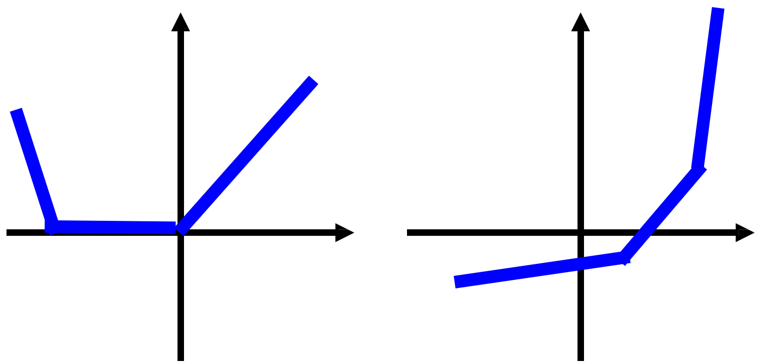
ReLU is a special cases of Maxout

- 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

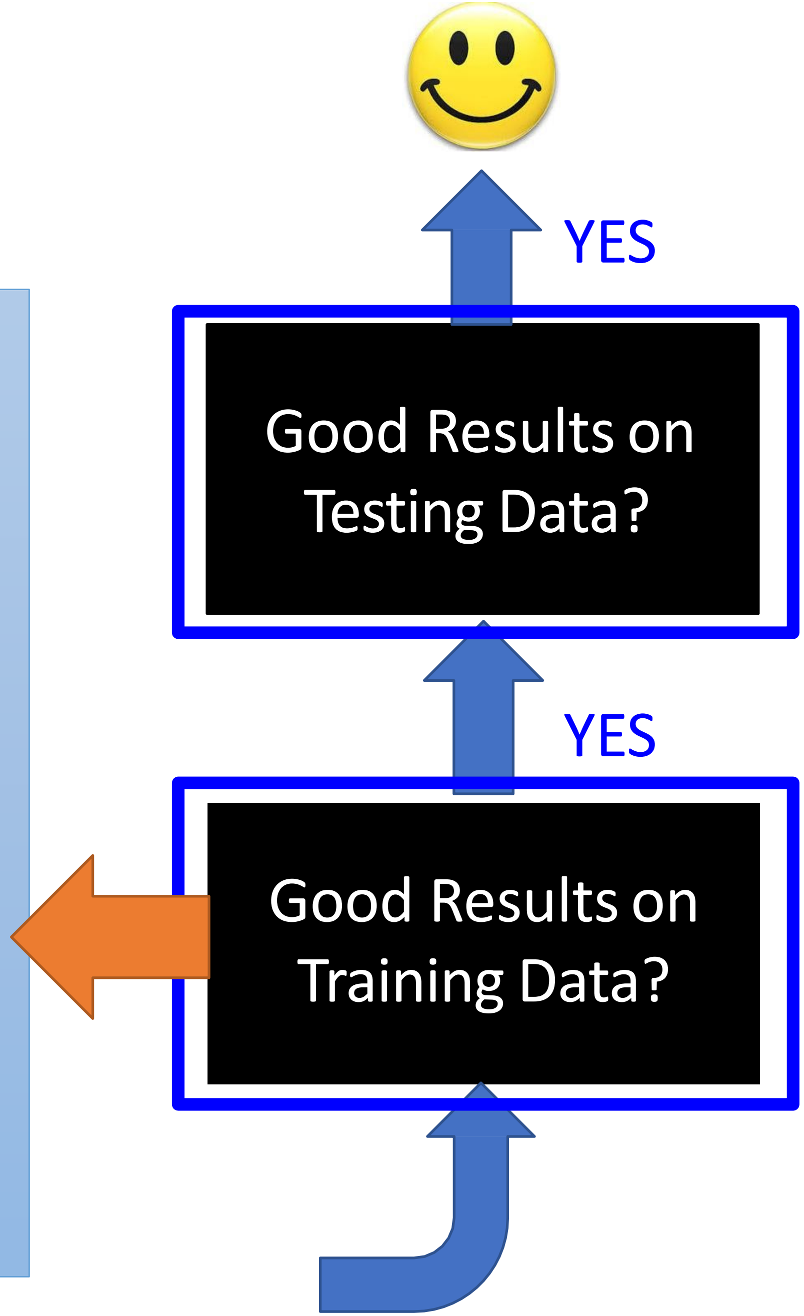


3 elements in a group



Recipe of Deep Learning

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

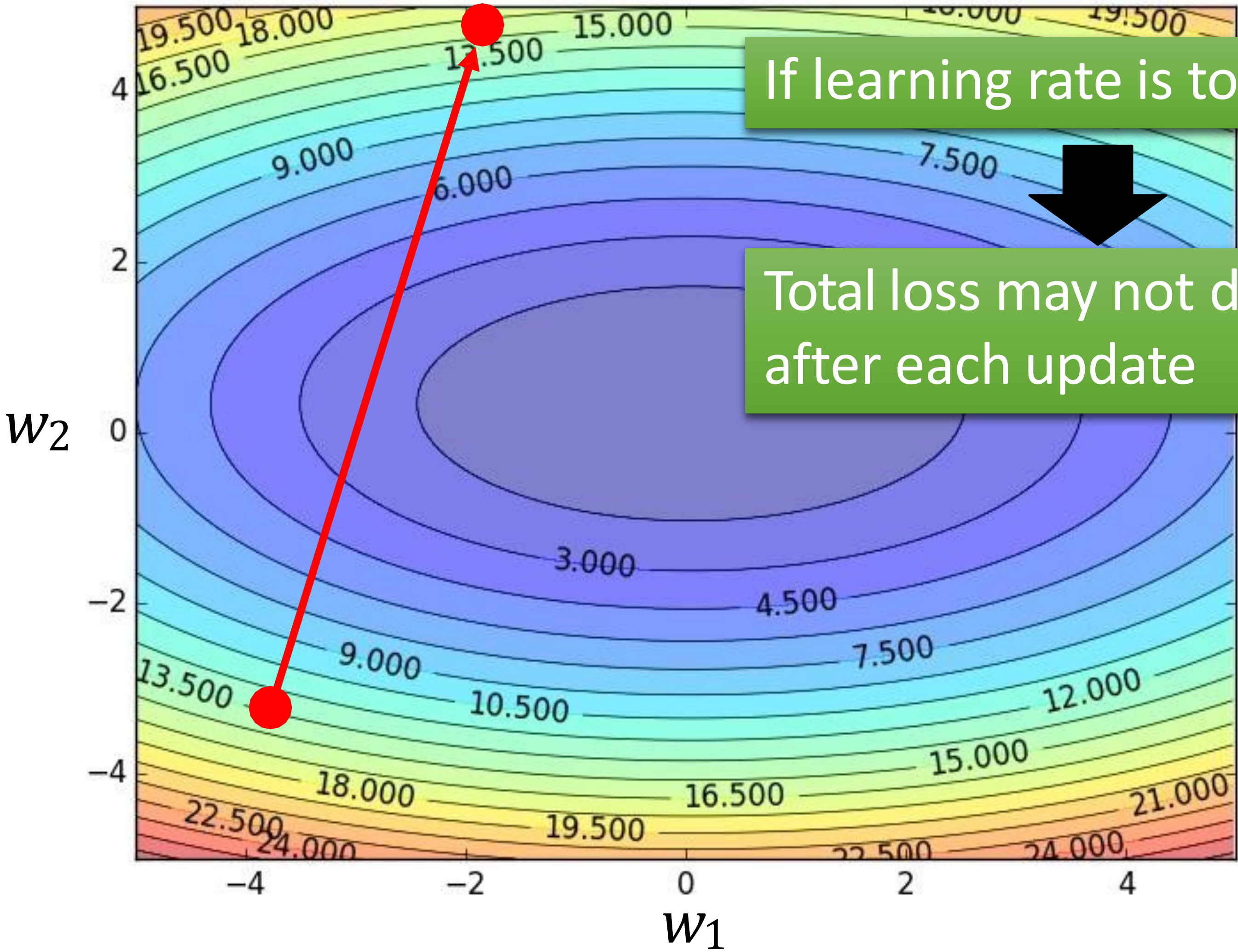


Learning Rates

Set the learning rate η carefully

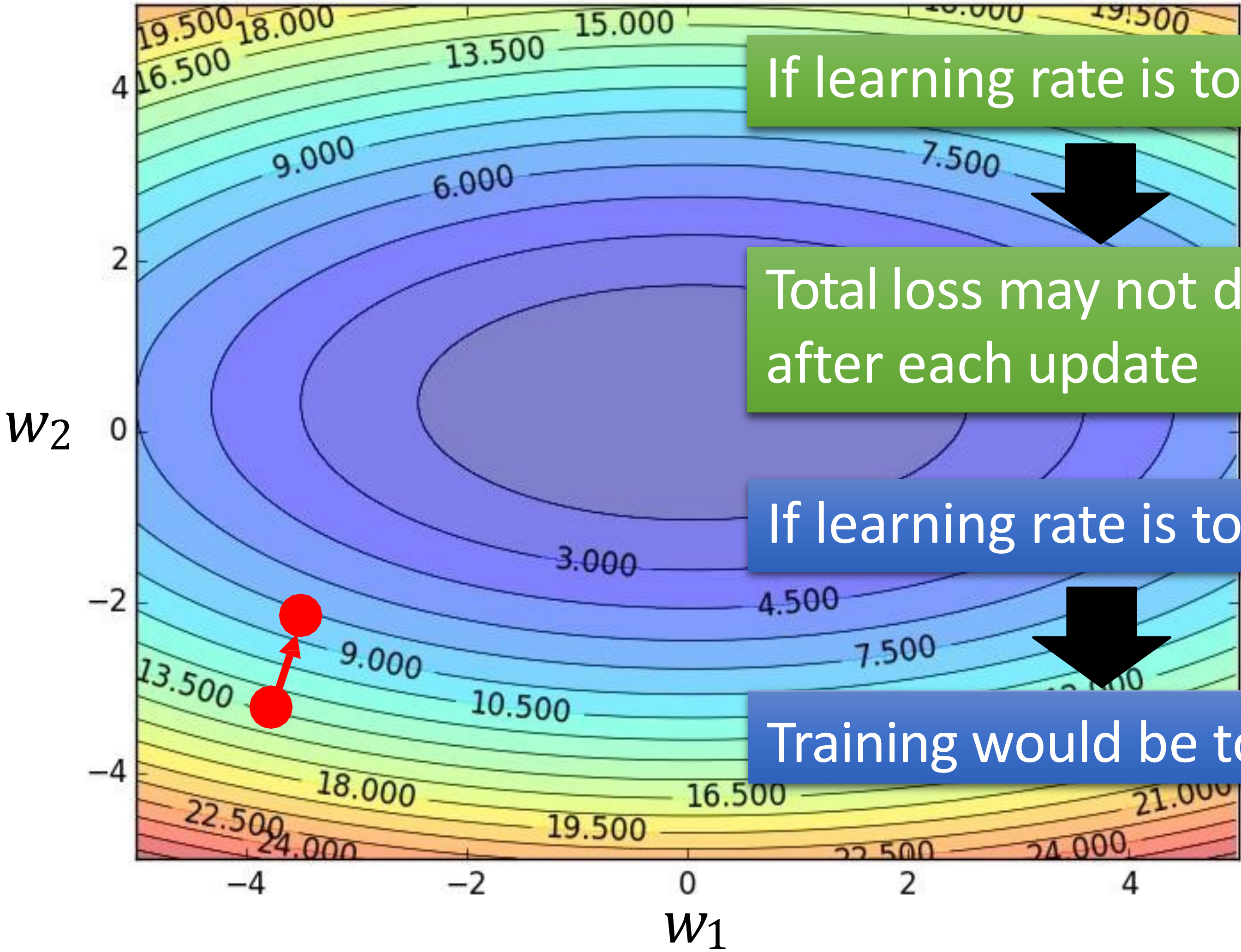
If learning rate is too large

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

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

constant

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

Summation of the square of the previous derivatives

Adagrad

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

w_1

$\mathbf{g^0}$
0.1

w_2

$\mathbf{g^0}$
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:

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

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

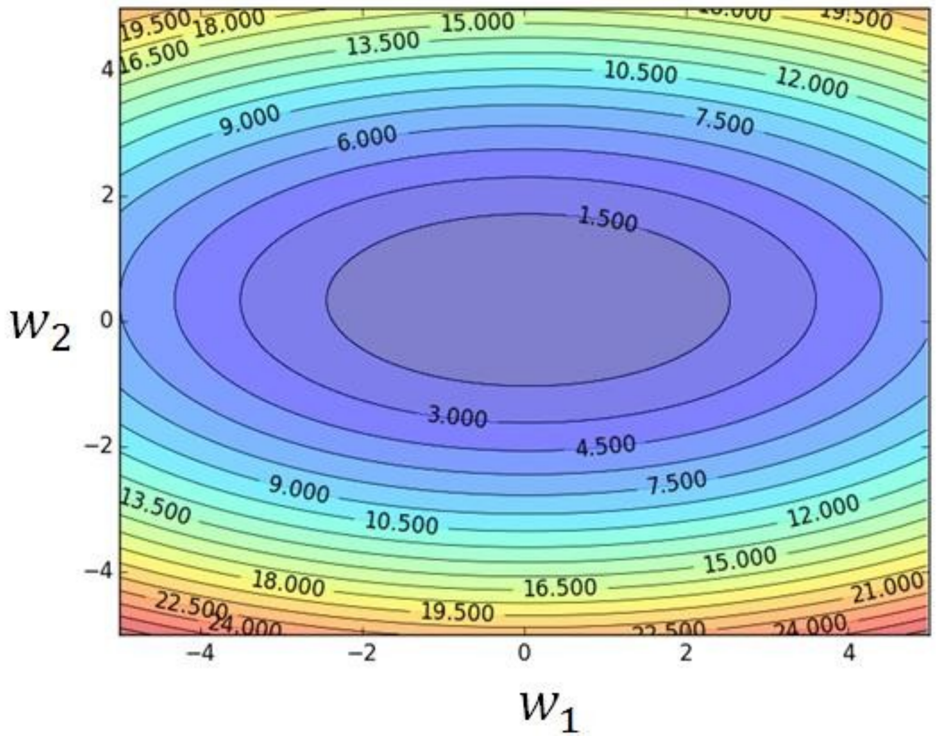
Observation: 1.Learning rate is smaller and smaller for all parameters

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

Why?

Larger derivatives

Smaller Learning Rate



Smaller Derivatives

Larger Learning Rate

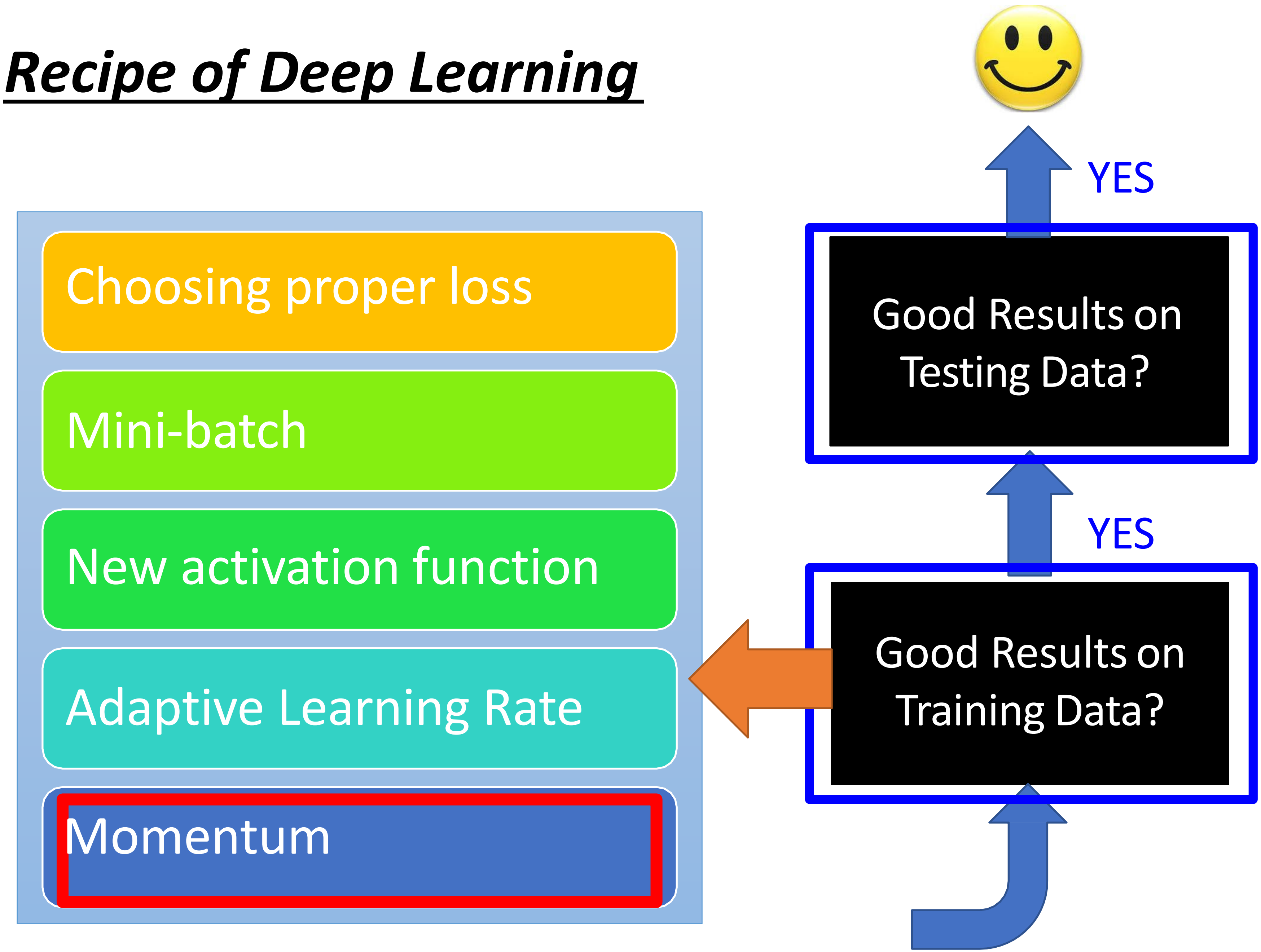
2. Smaller derivatives, larger learning rate, and vice versa

Why?

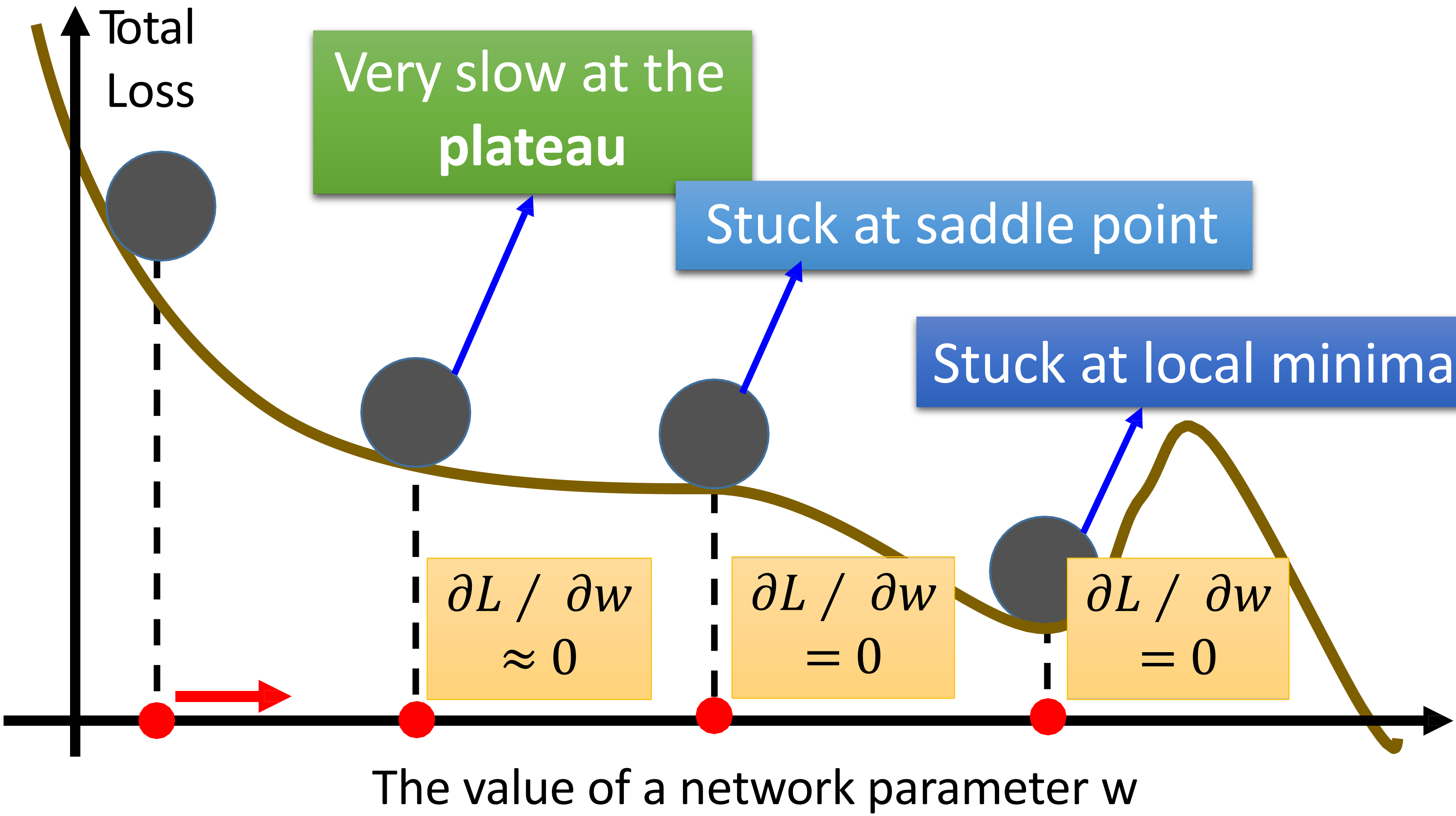
Not the whole story ……

- 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

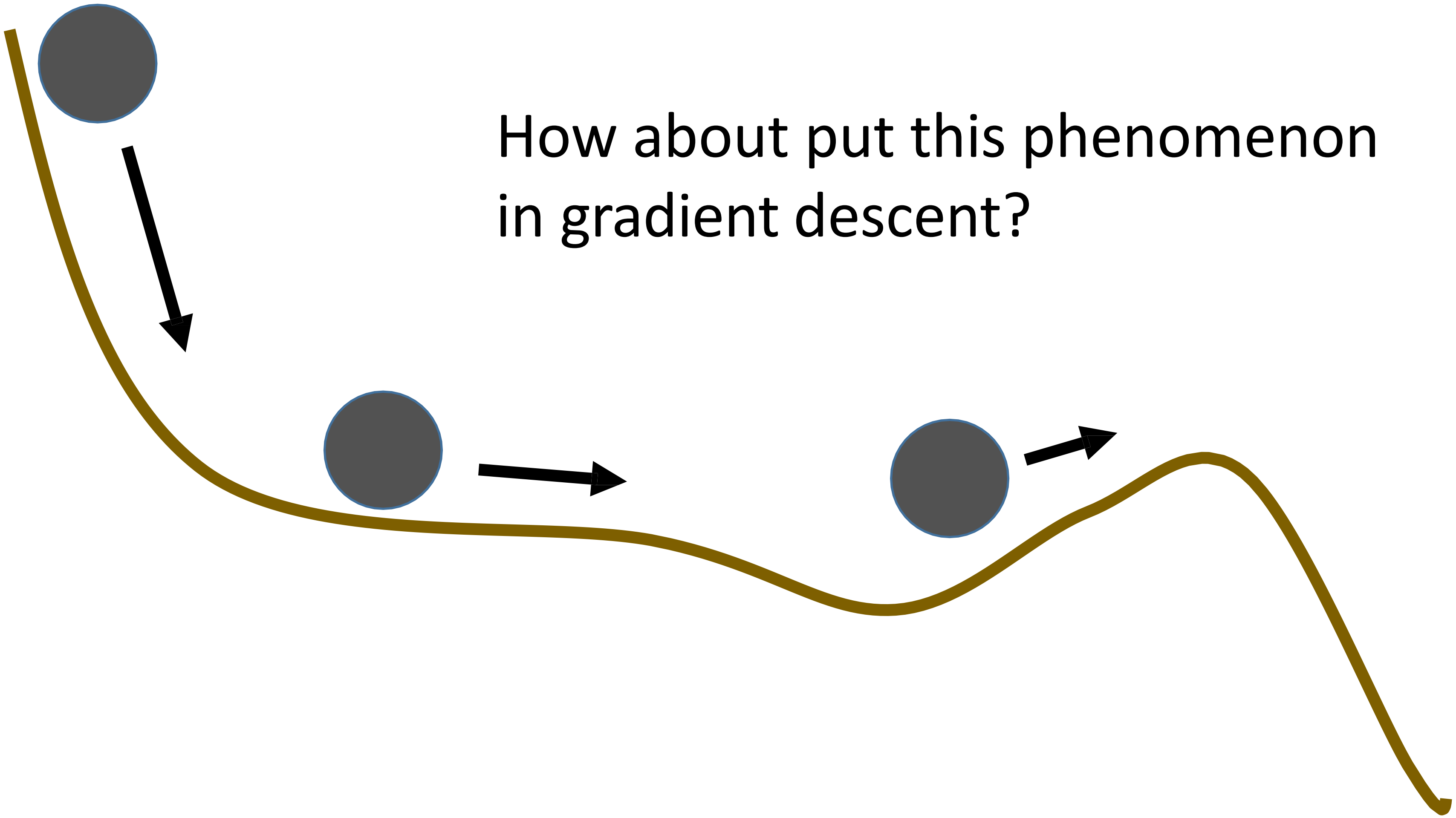


Hard to find optimal network parameters



In physical world ……

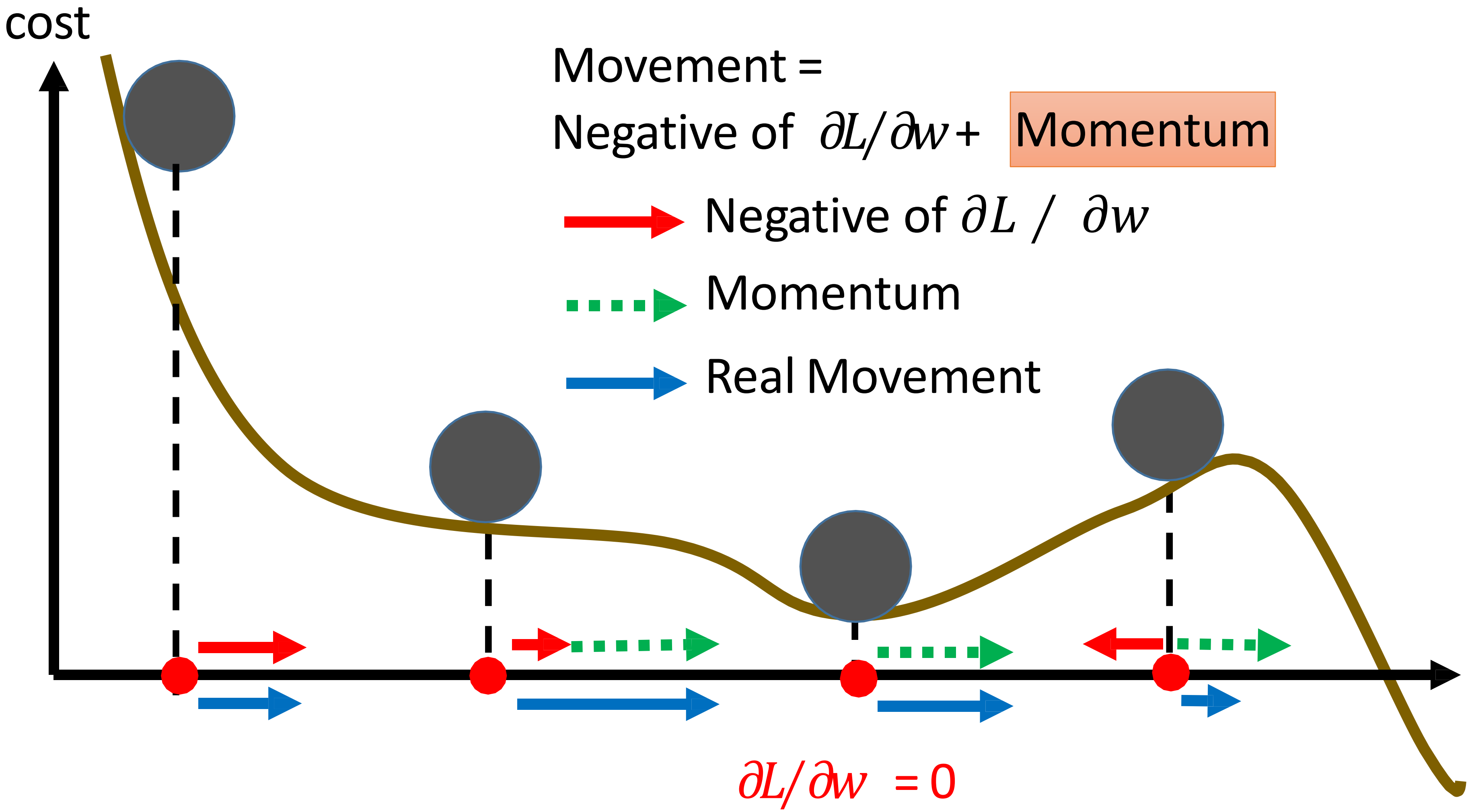
- Momentum



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.

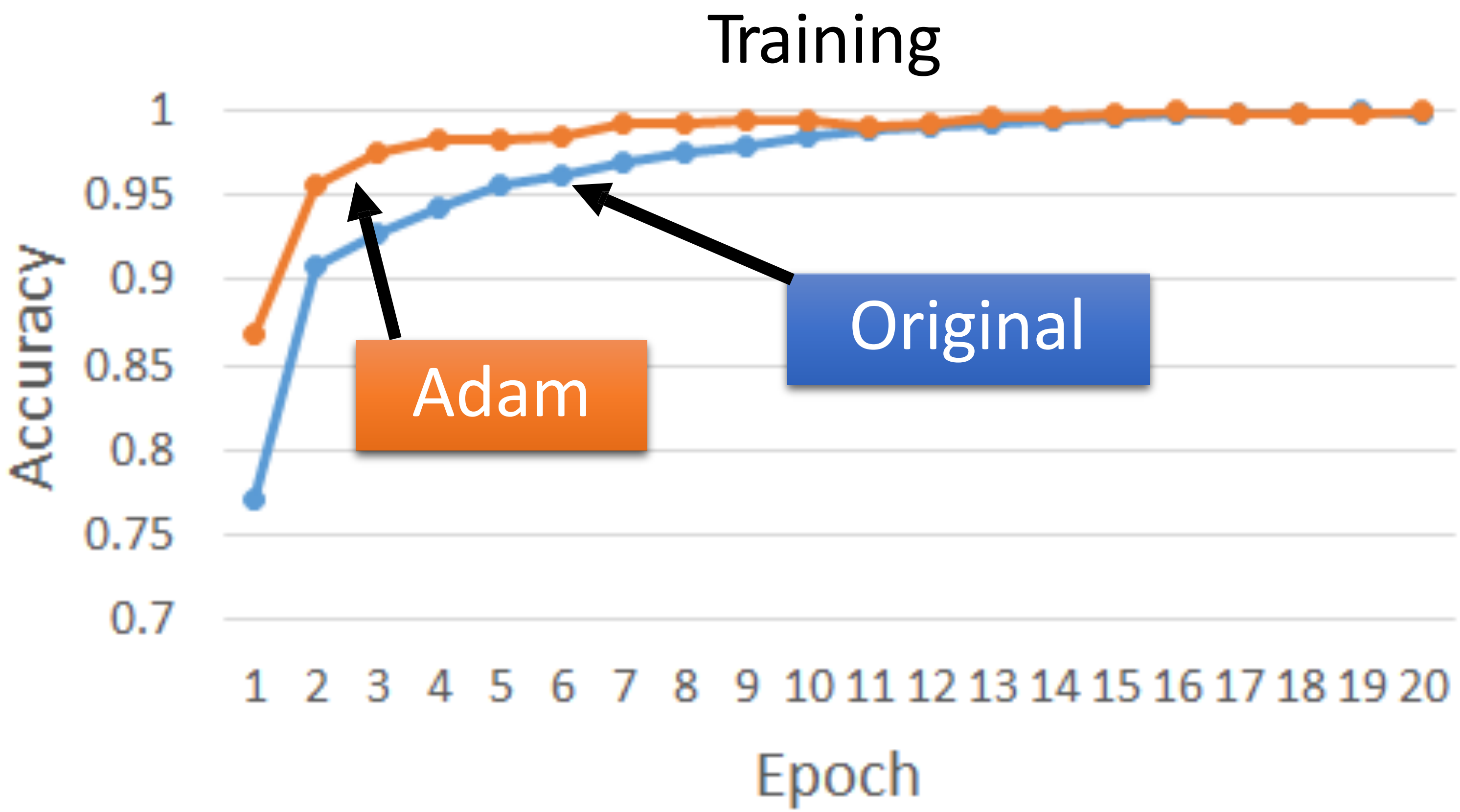
Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 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

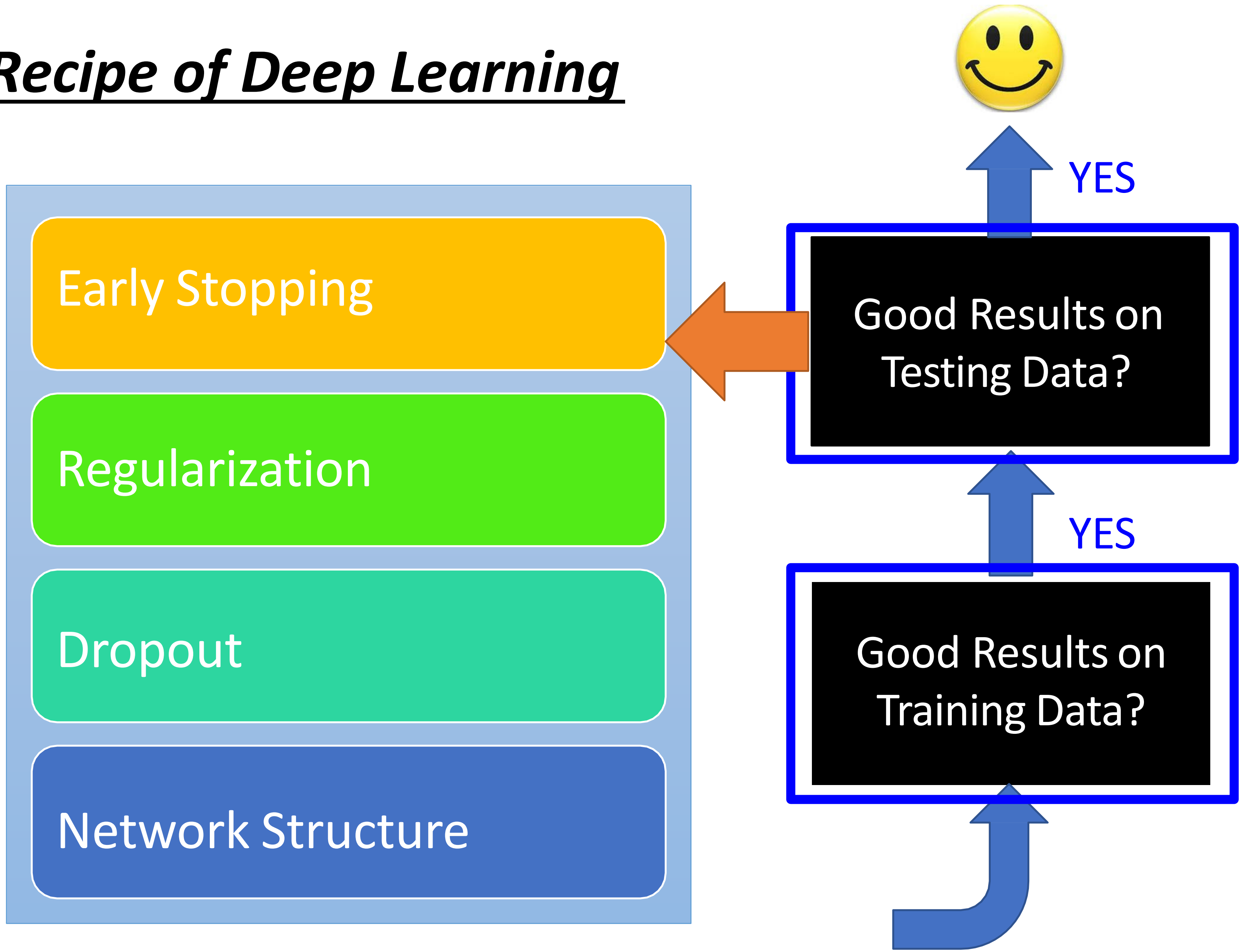
Testing:

	Accuracy
Original	0.96
Adam	0.97

- ReLU, 3 layer



Recipe of Deep Learning



Why Overfitting?

- Training data and testing data can be different.

Learning target is defined by the training data.

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

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:

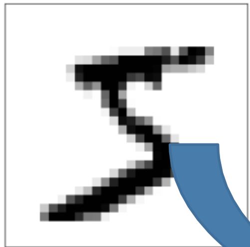
	Accuracy
Clean	0.97
Noisy	0.50

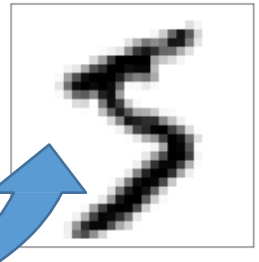
Training is not influenced.

Panacea for Overfitting

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

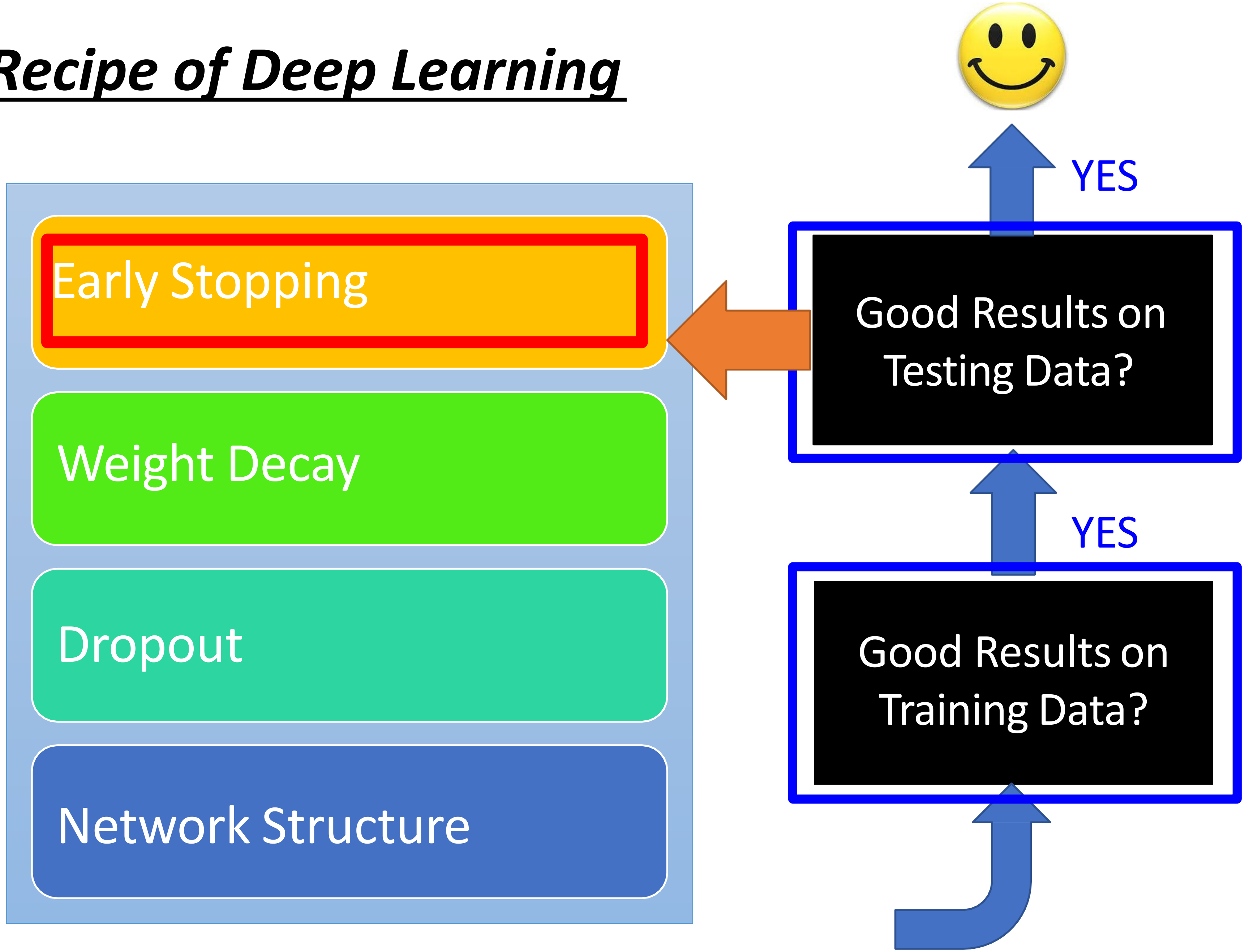
Handwriting recognition:

Original Training Data: 

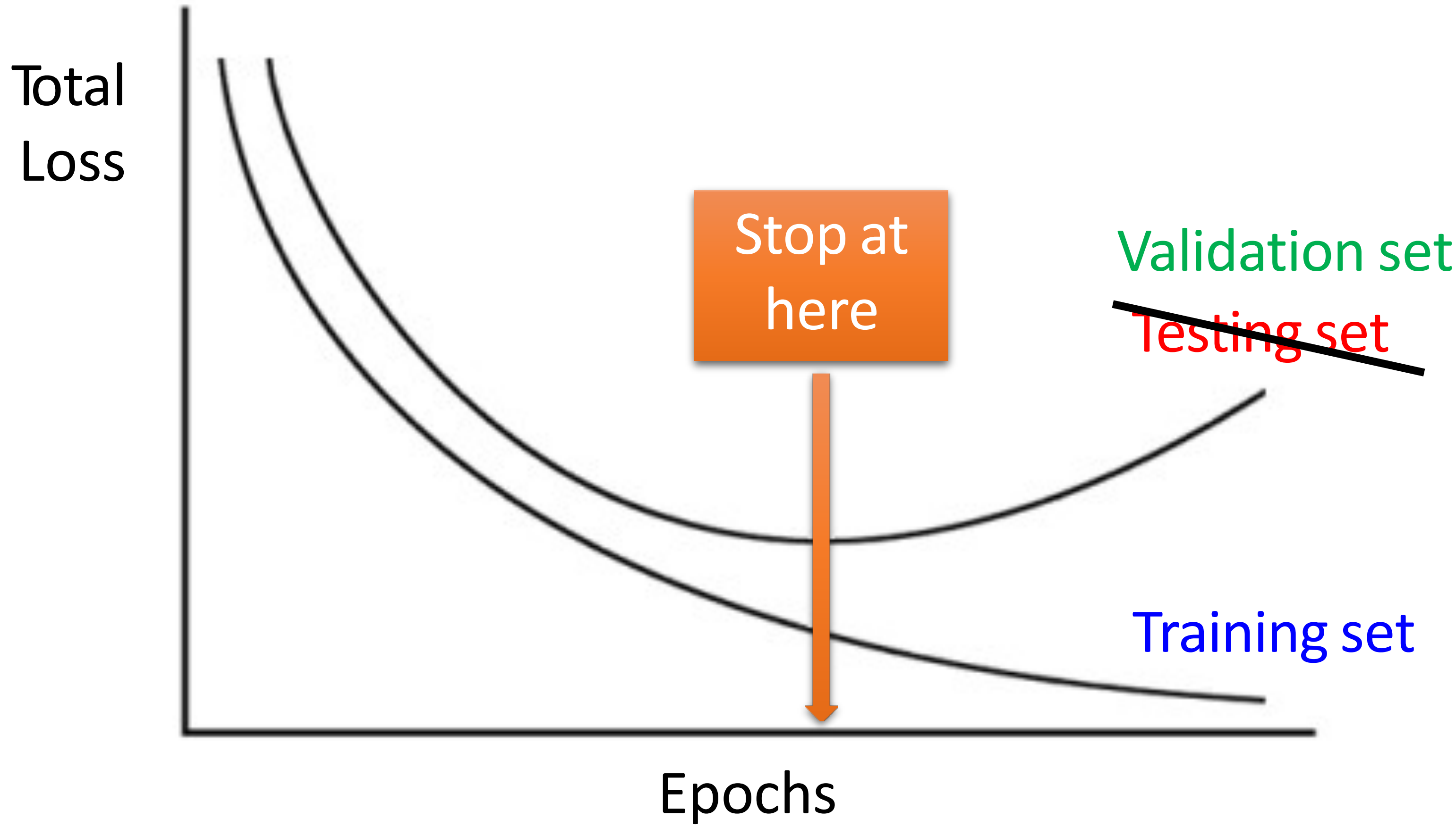
Created Training Data: 

Shift 15 °

Recipe of Deep Learning

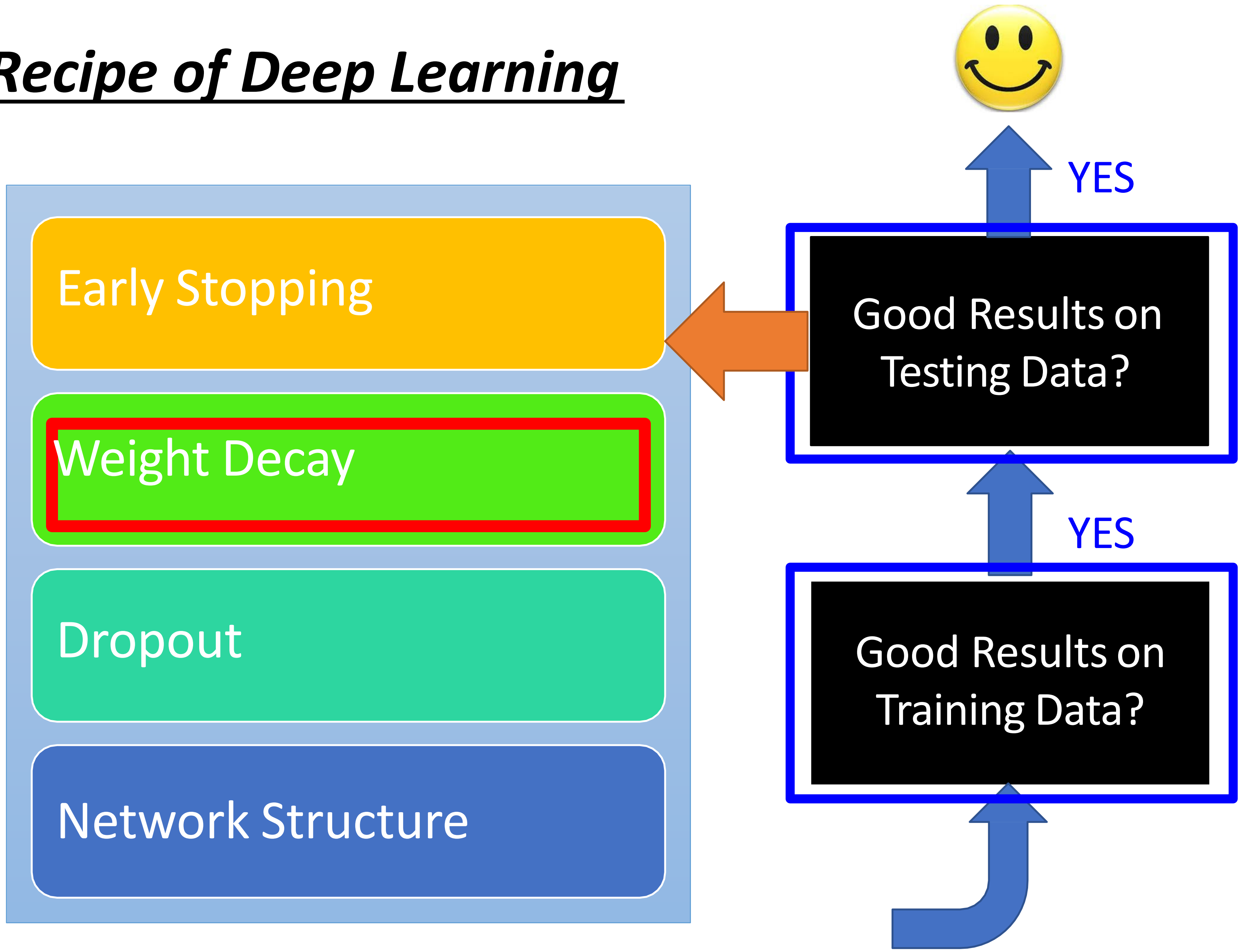


Early Stopping



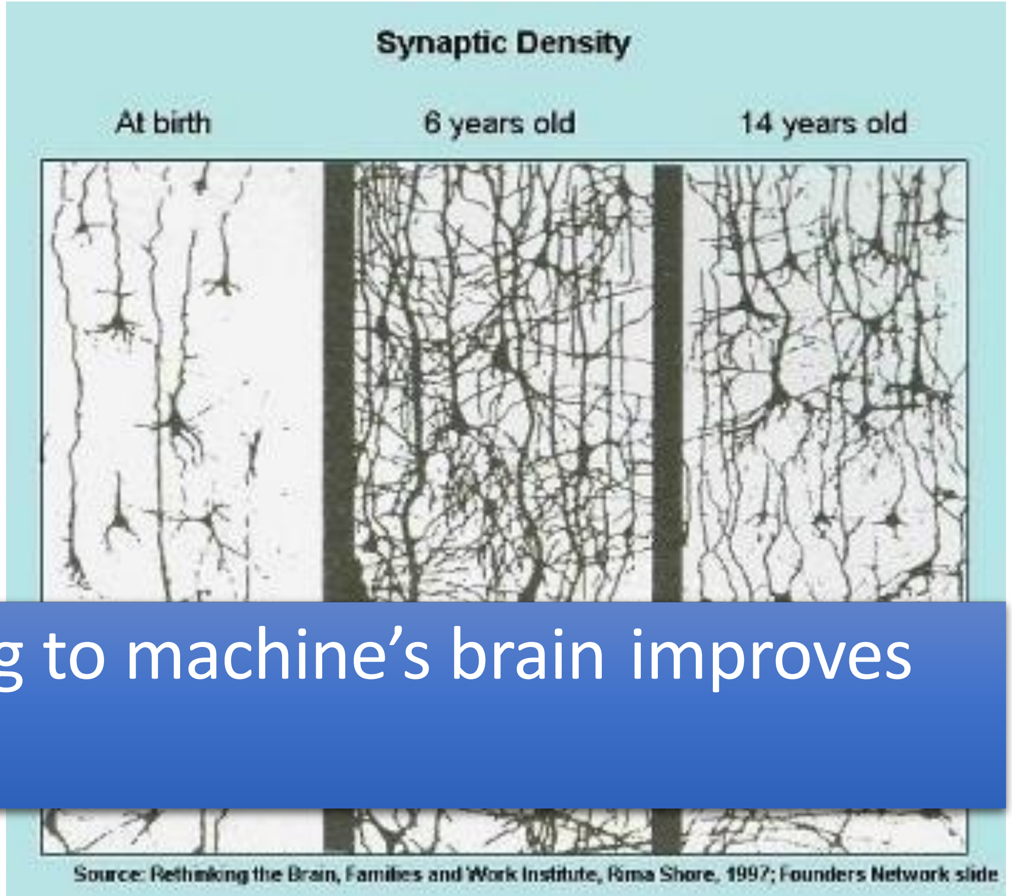
Keras: http://keras.io/getting-started/faq/#how-can-i-interrupt-training-when-the-validation-loss-isnt-decreasing-anymore

Recipe of Deep Learning



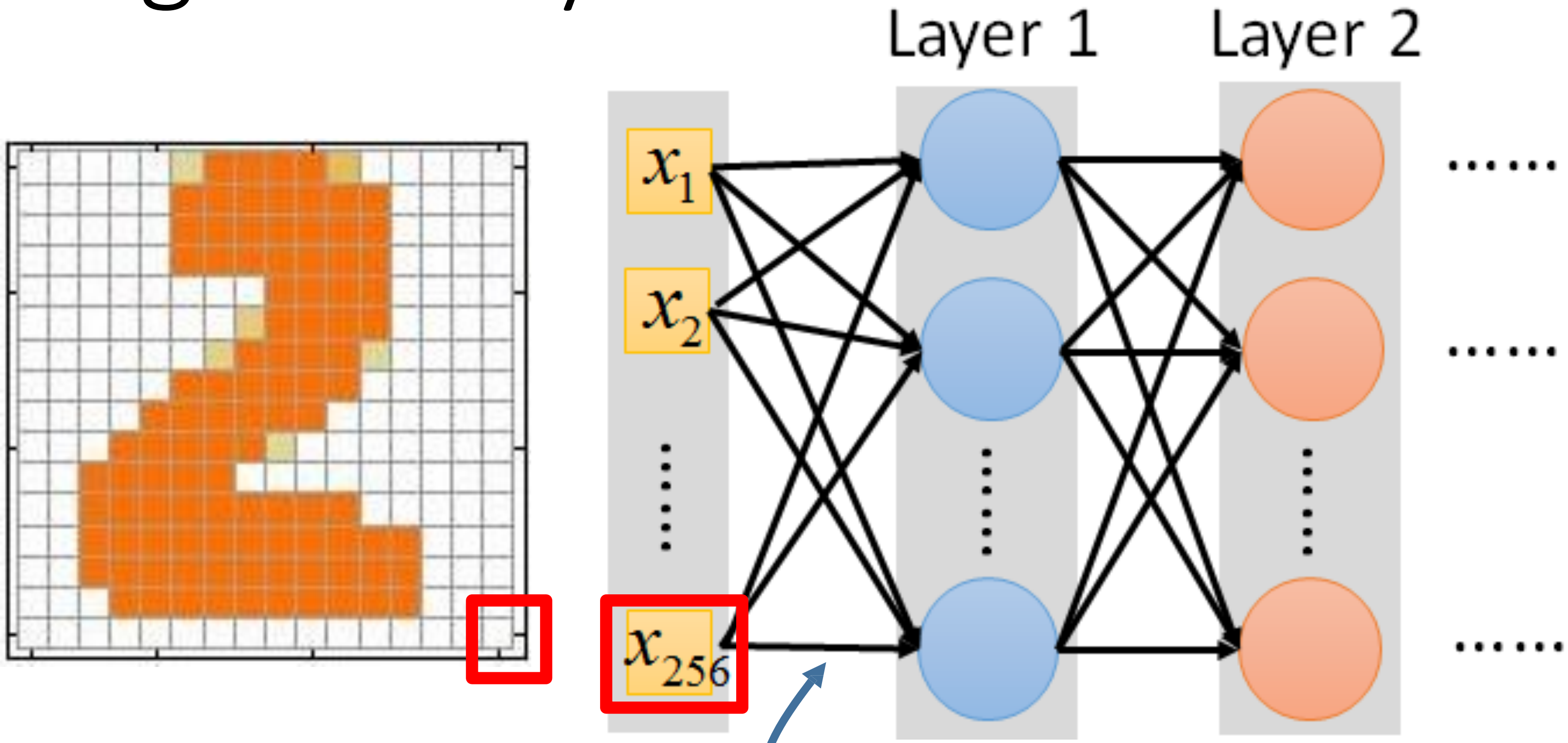
Weight Decay

- Our brain prunes out the useless link between neurons.



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

Weight Decay



Useless

Close to zero (萎縮了)

Weight decay is one kind of regularization

Weight Decay

- Implementation

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

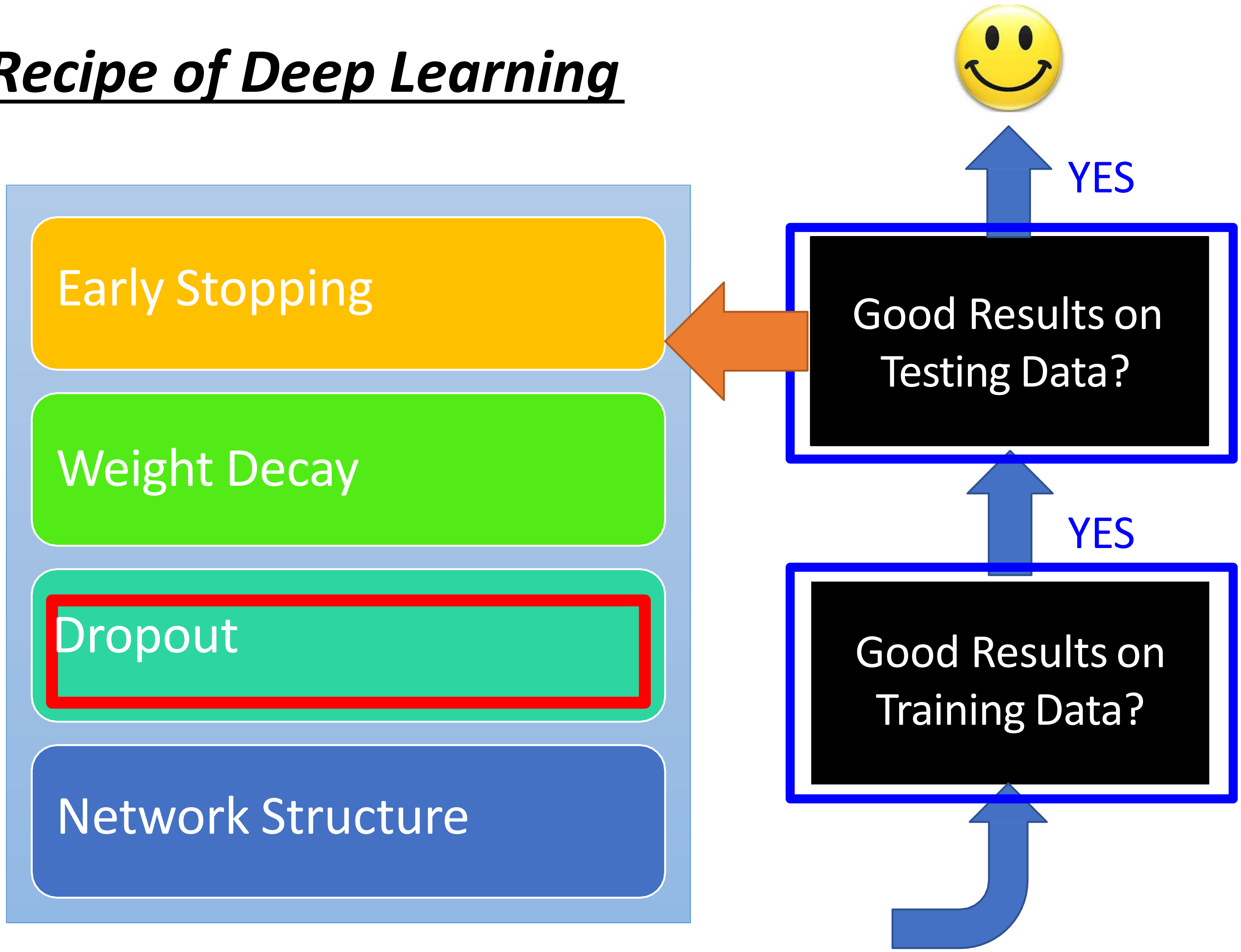
$\lambda = 0.01$

Weight Decay: $w \leftarrow \underline{\boxed{(0.99)}} w - \eta \frac{\partial L}{\partial w}$

Smaller and smaller

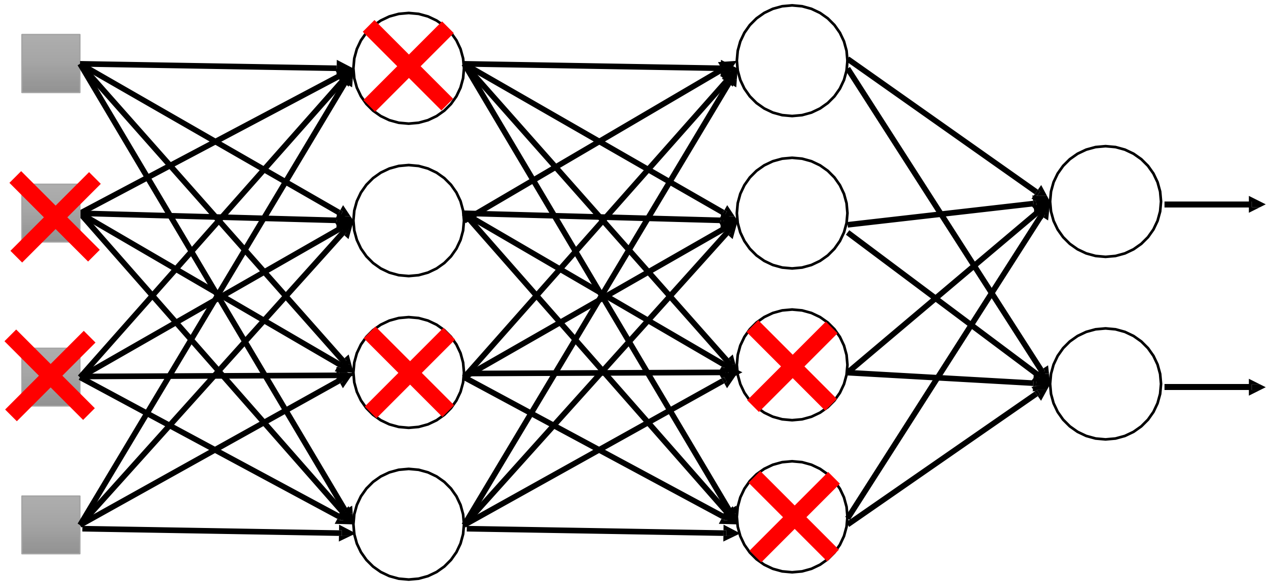
Keras: http://keras.io/regularizers/

Recipe of Deep Learning



Dropout

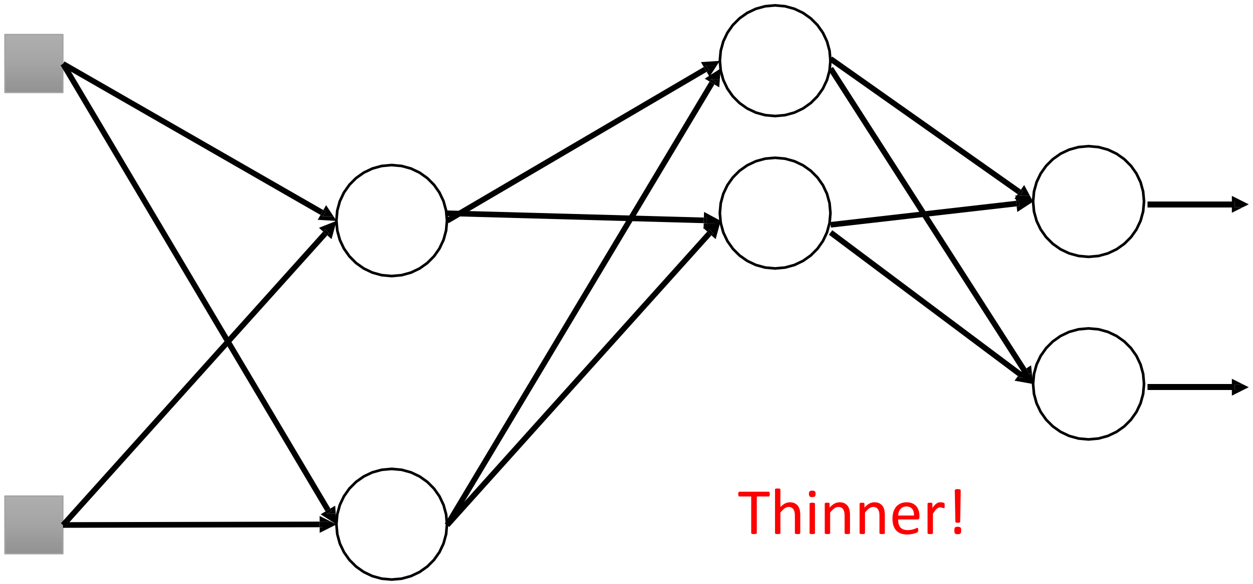
Training:



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

Dropout

Training:



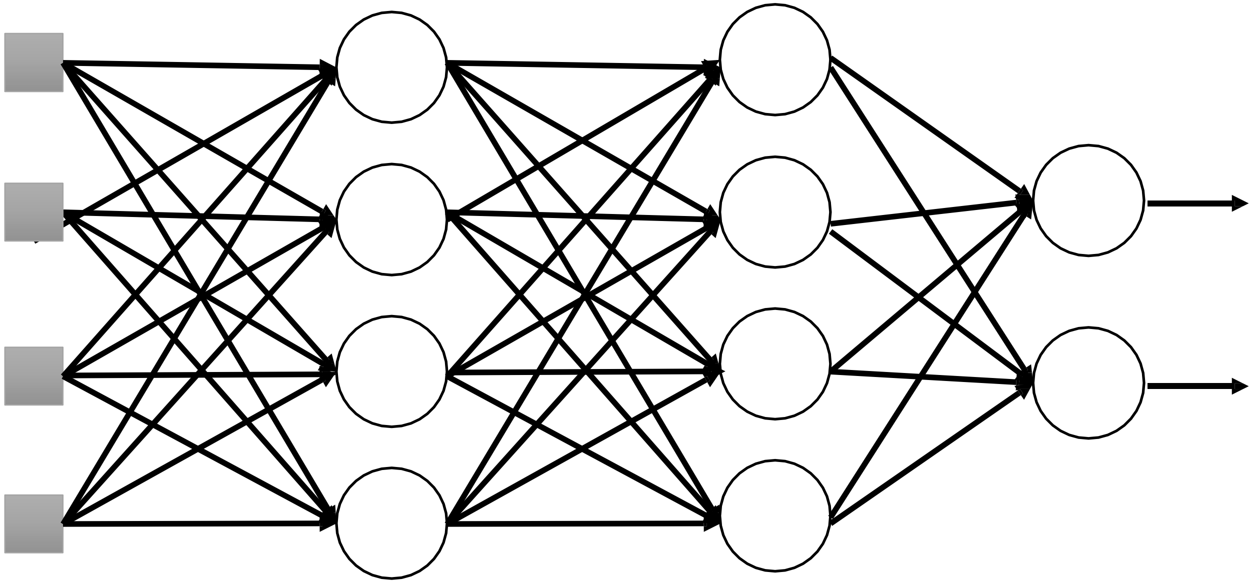
Thinner!

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

For each mini-batch, we resample the dropout neurons

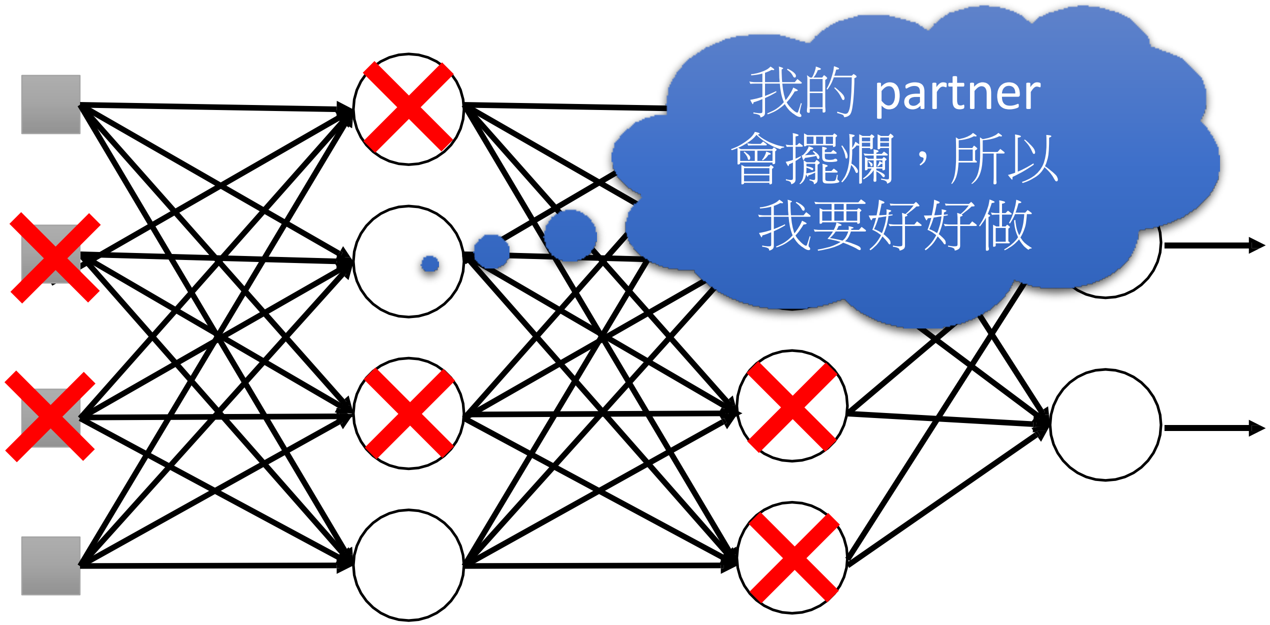
Dropout

Testing:



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

Dropout - Intuitive Reason



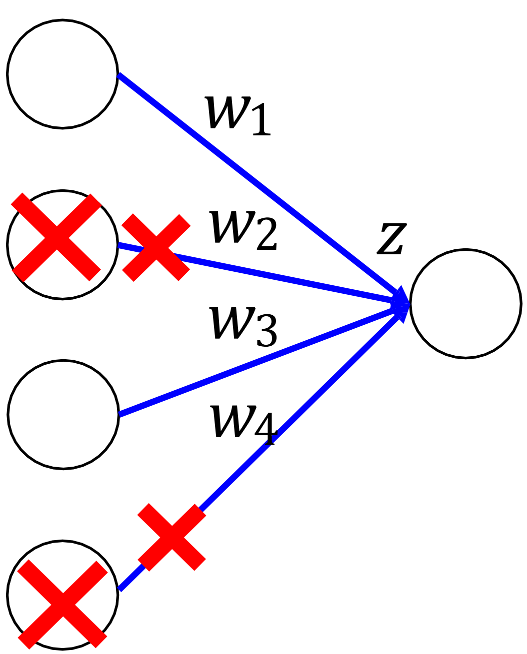
- When teams up, if everyone expect the partner will do the work, nothing will be done finally.
- However, if you know your partner will dropout, you will do better.
- When testing, no one dropout actually, so obtaining good results eventually.

Dropout - Intuitive Reason

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

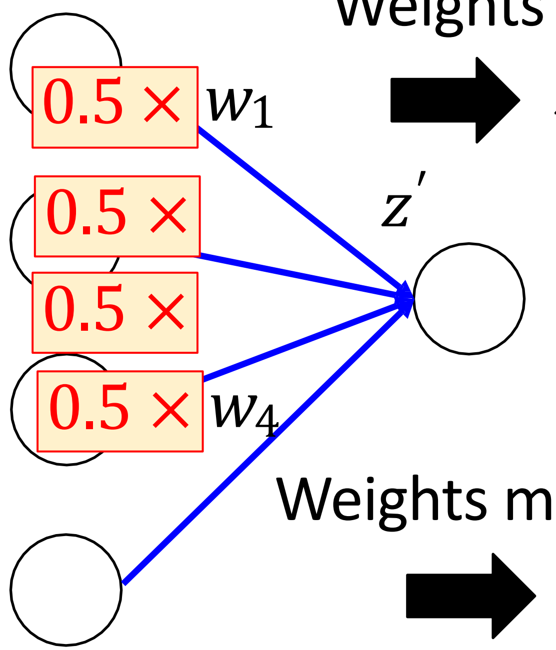
Training of Dropout

Assume dropout rate is 50%



Testing of Dropout

No dropout



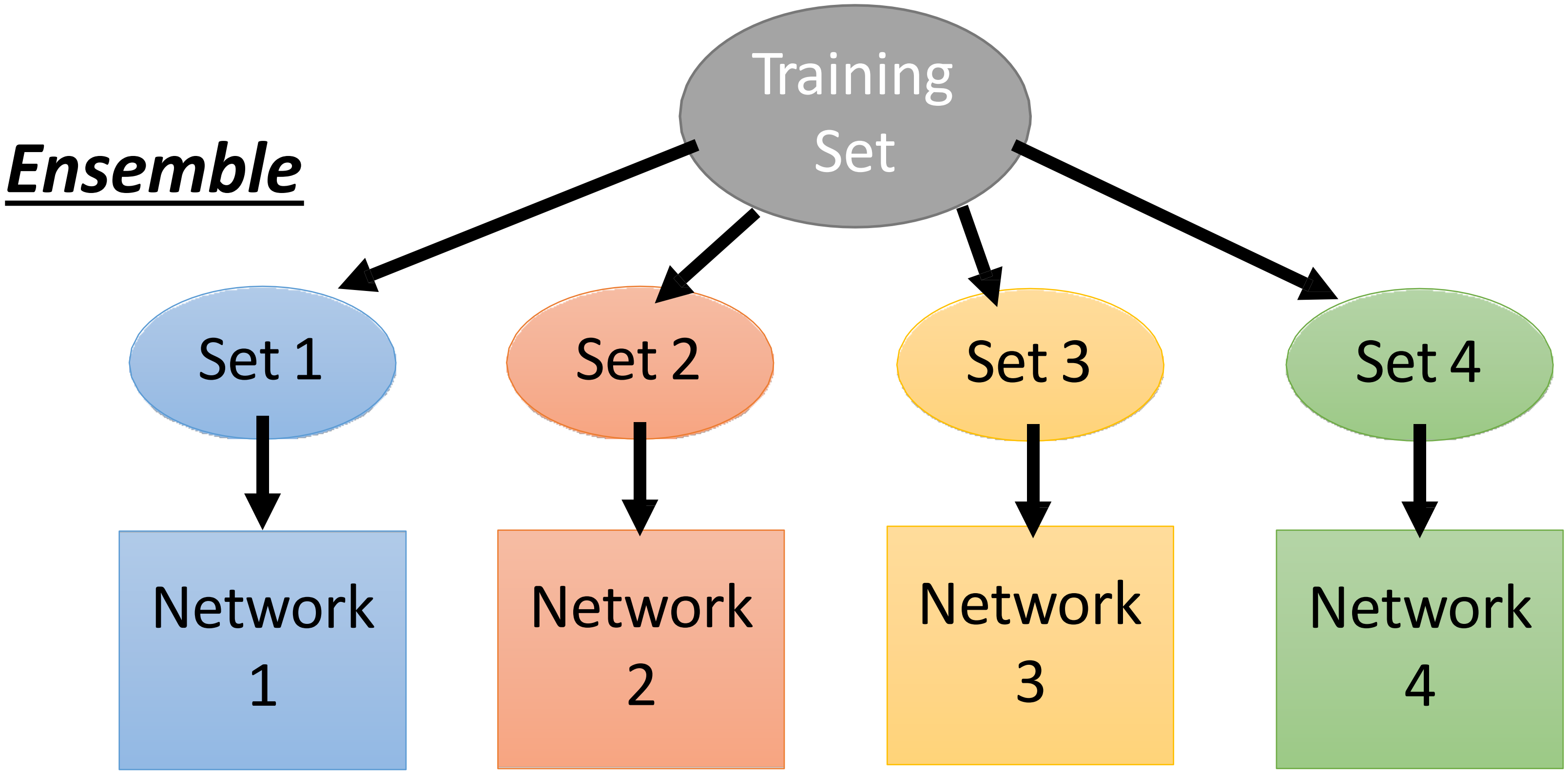
Weights from training

$\Rightarrow z' \approx 2z$

Weights multiply (1-p)%

$\Rightarrow z' \approx z$

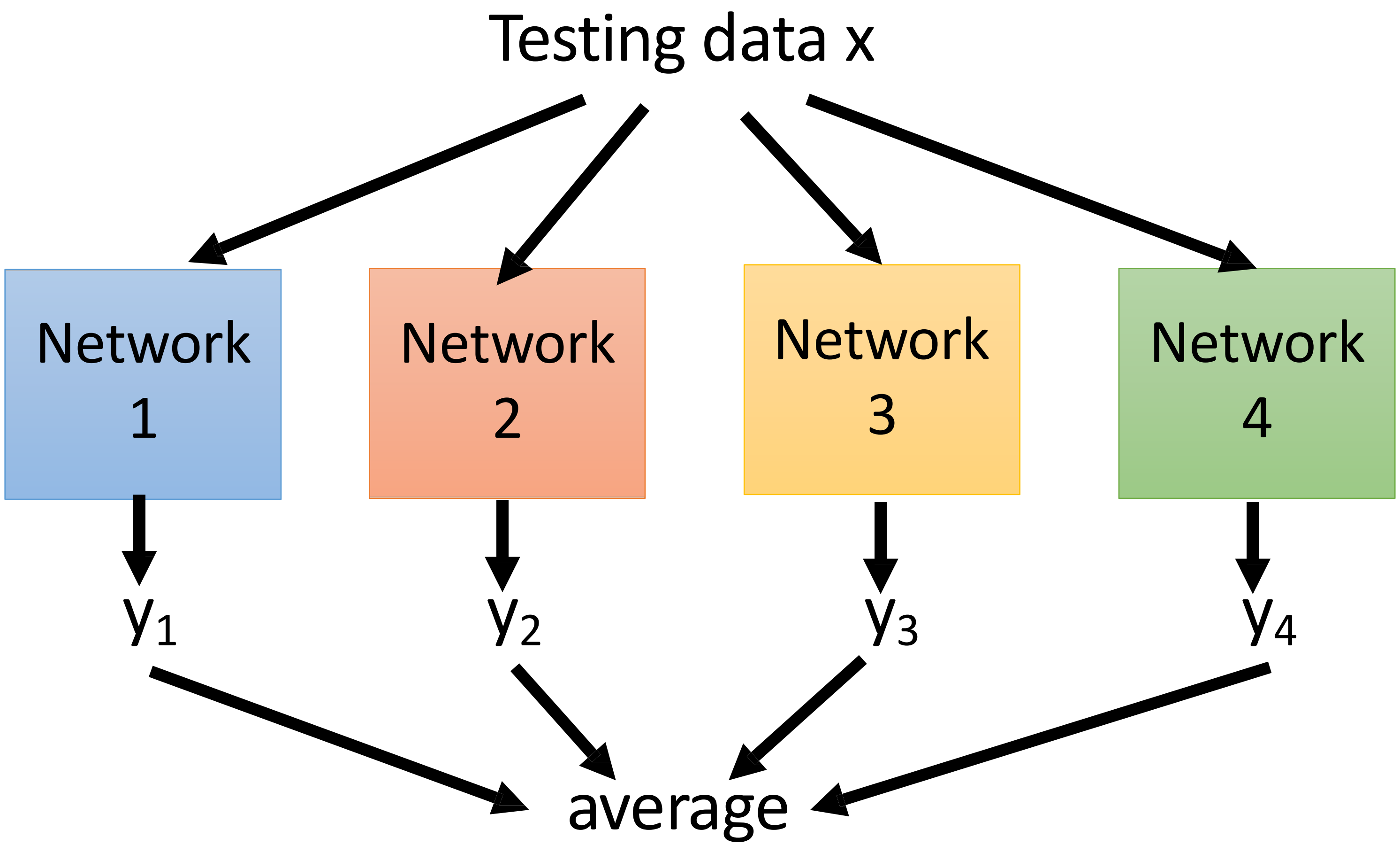
Dropout is a kind of ensemble.



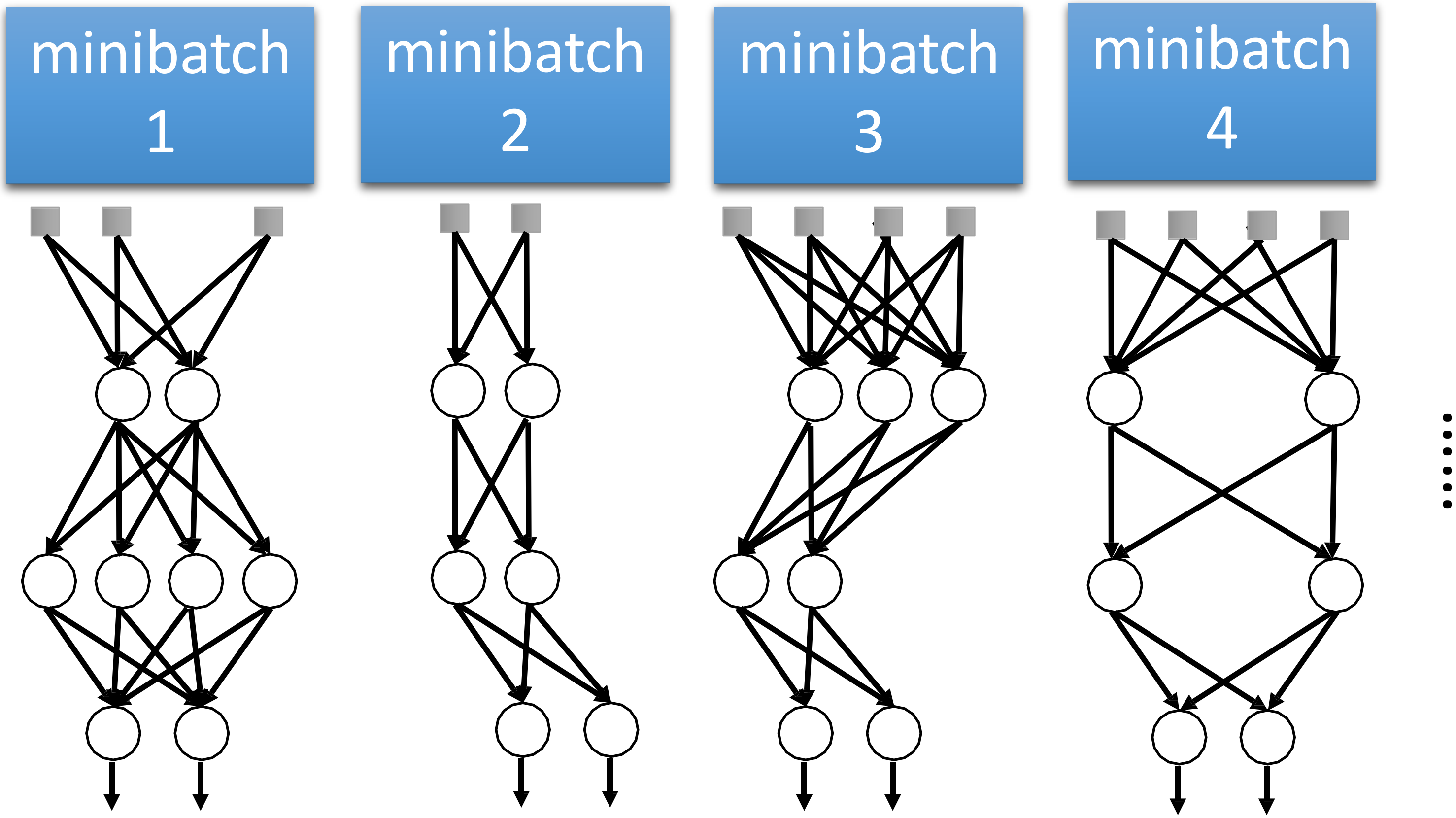
Train a bunch of networks with different structures

Dropout is a kind of ensemble.

Ensemble



Dropout is a kind of ensemble.



Training of Dropout

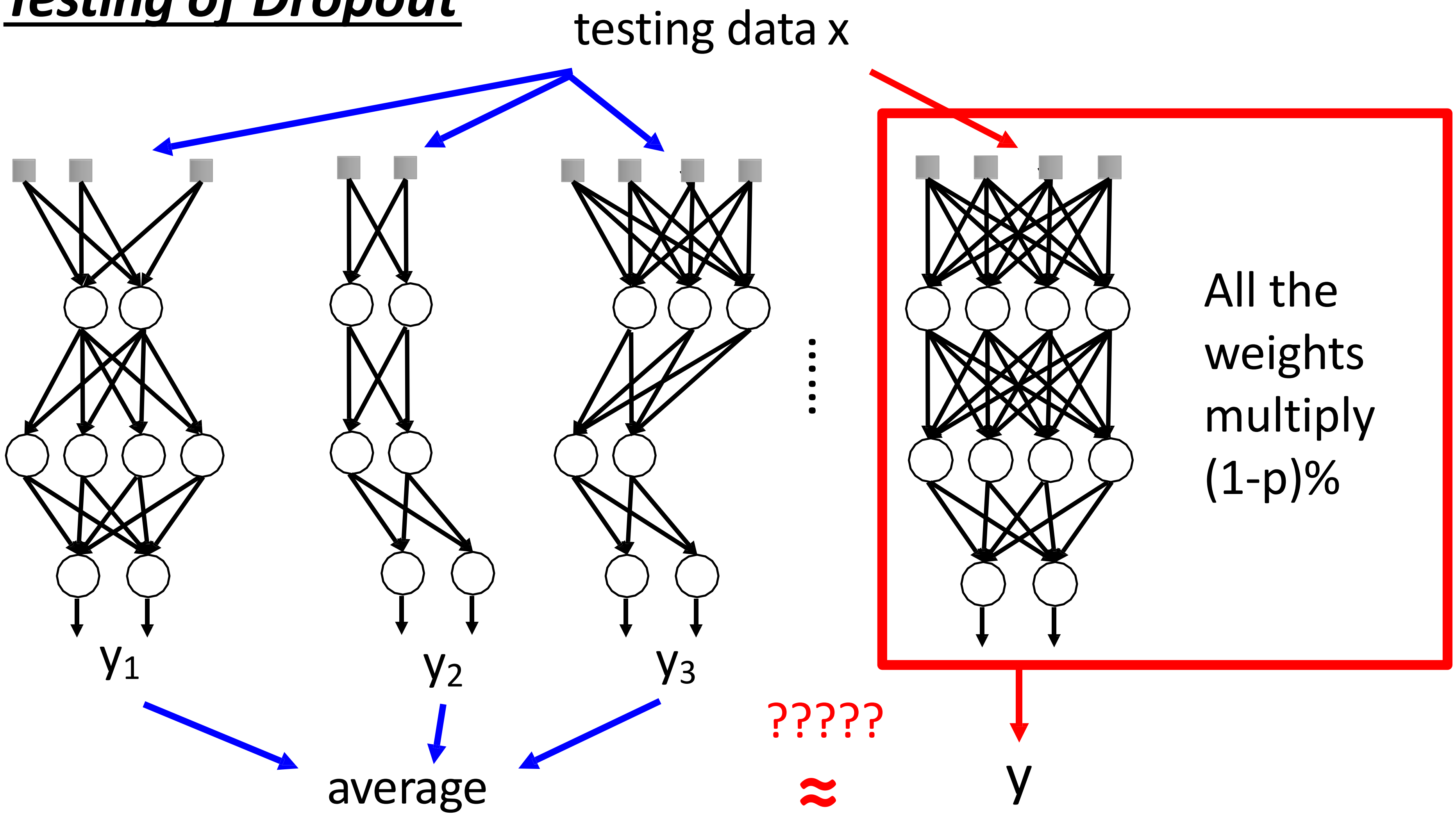
M neurons

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.

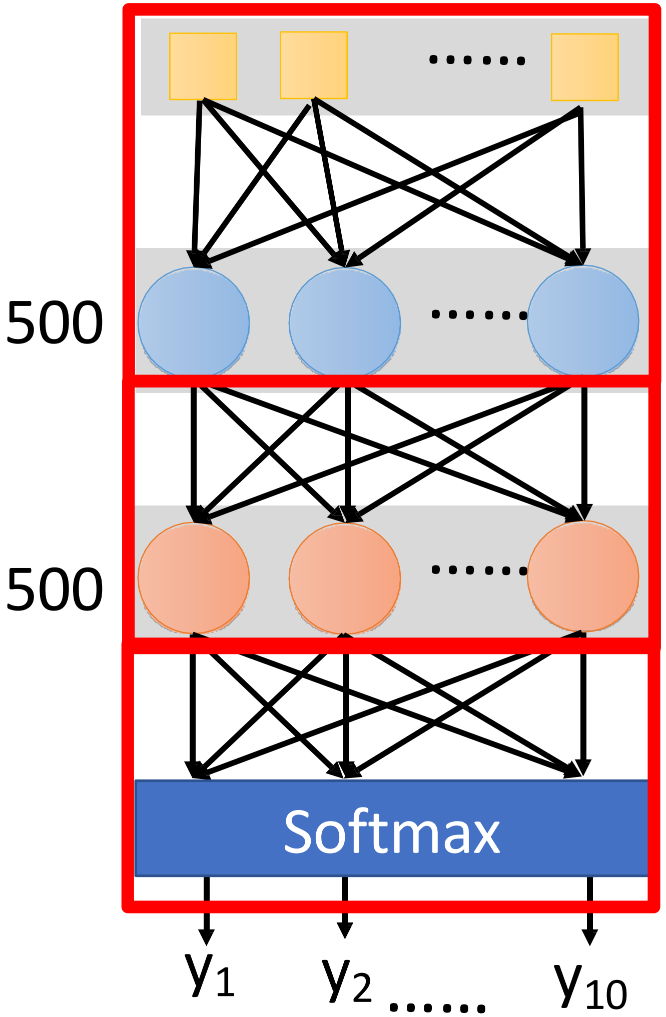
Testing of Dropout



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') )
```

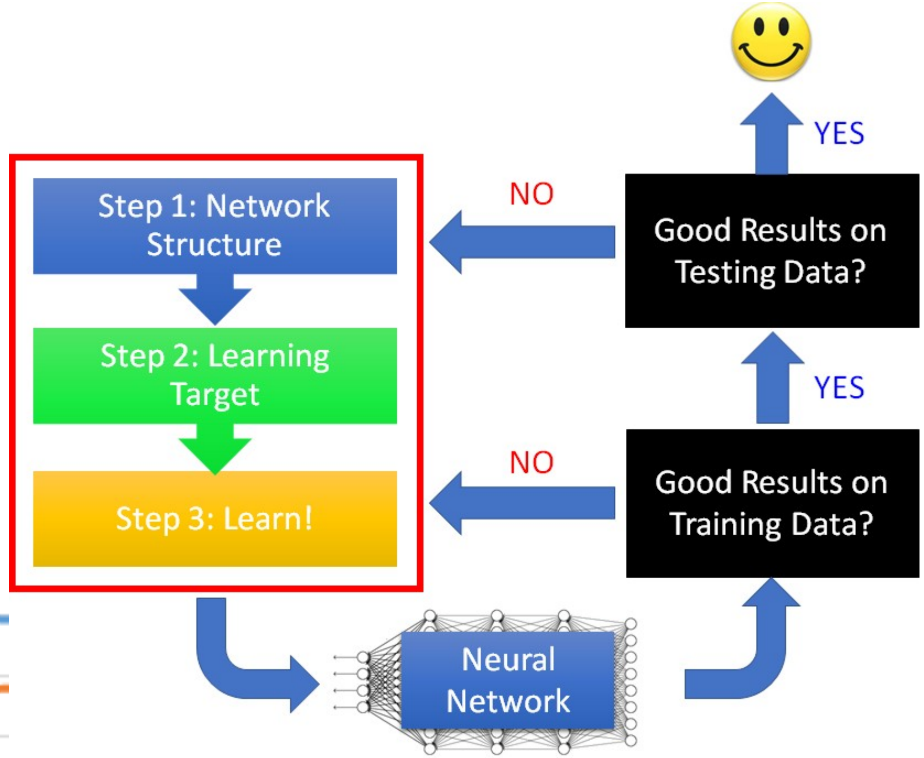
model.add(dropout(0.8))

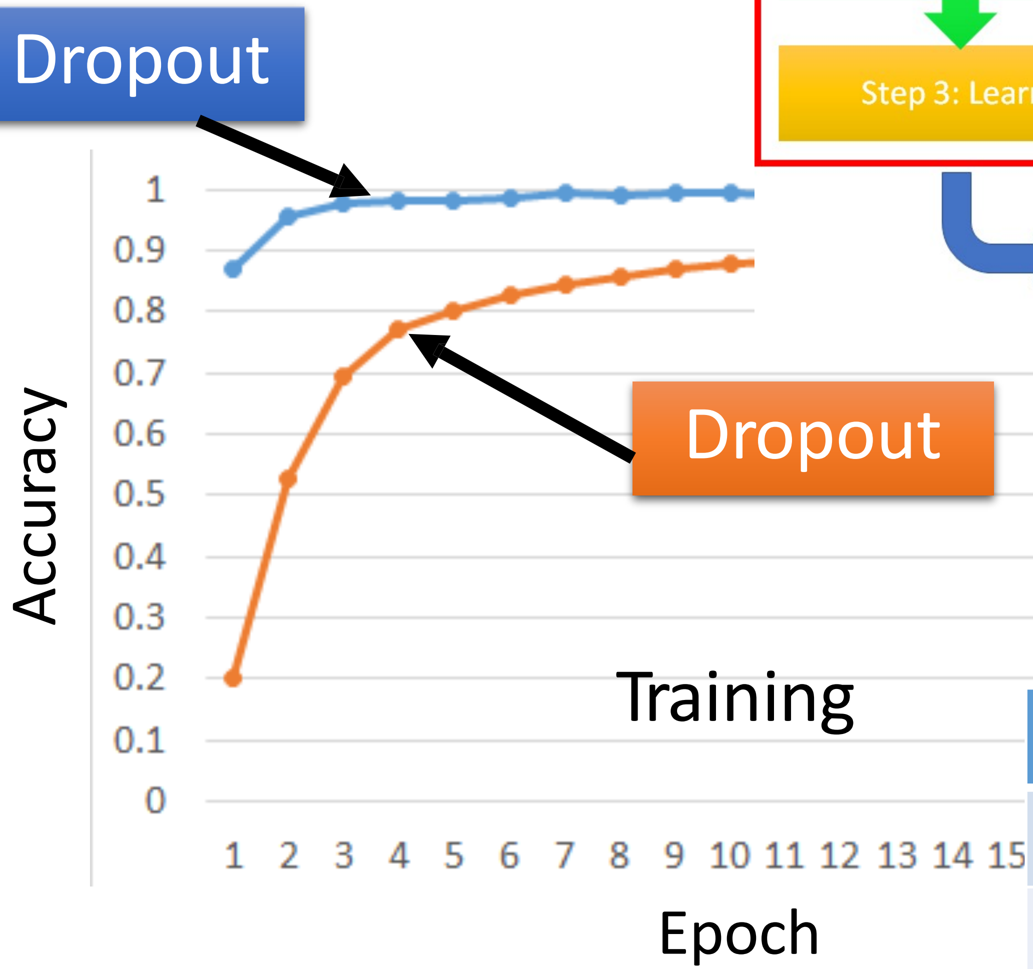
```
model.add( Dense( output_dim=500 ) )
model.add( Activation('sigmoid') )
```

model.add(dropout(0.8))

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

Let's try it

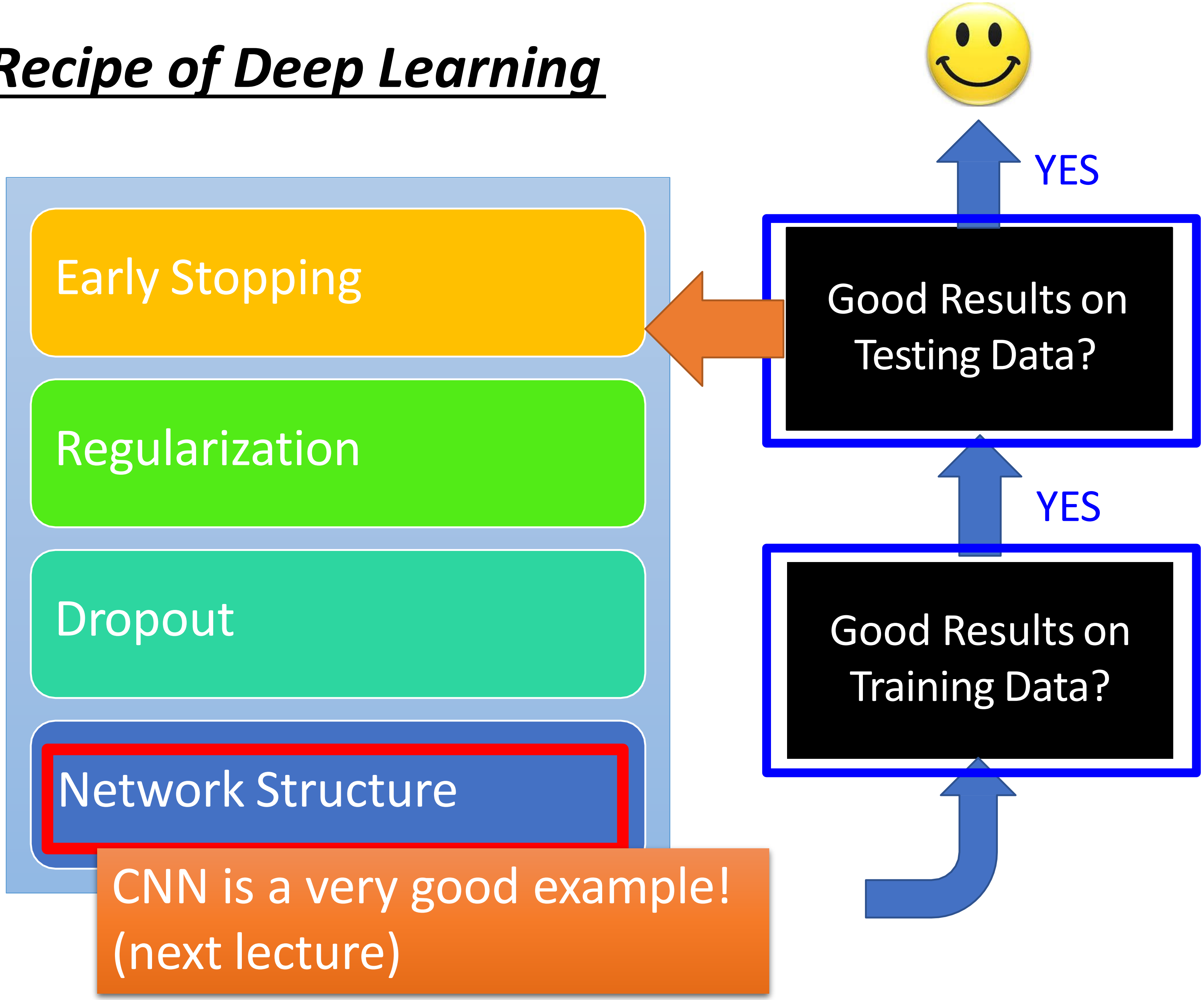




Testing:

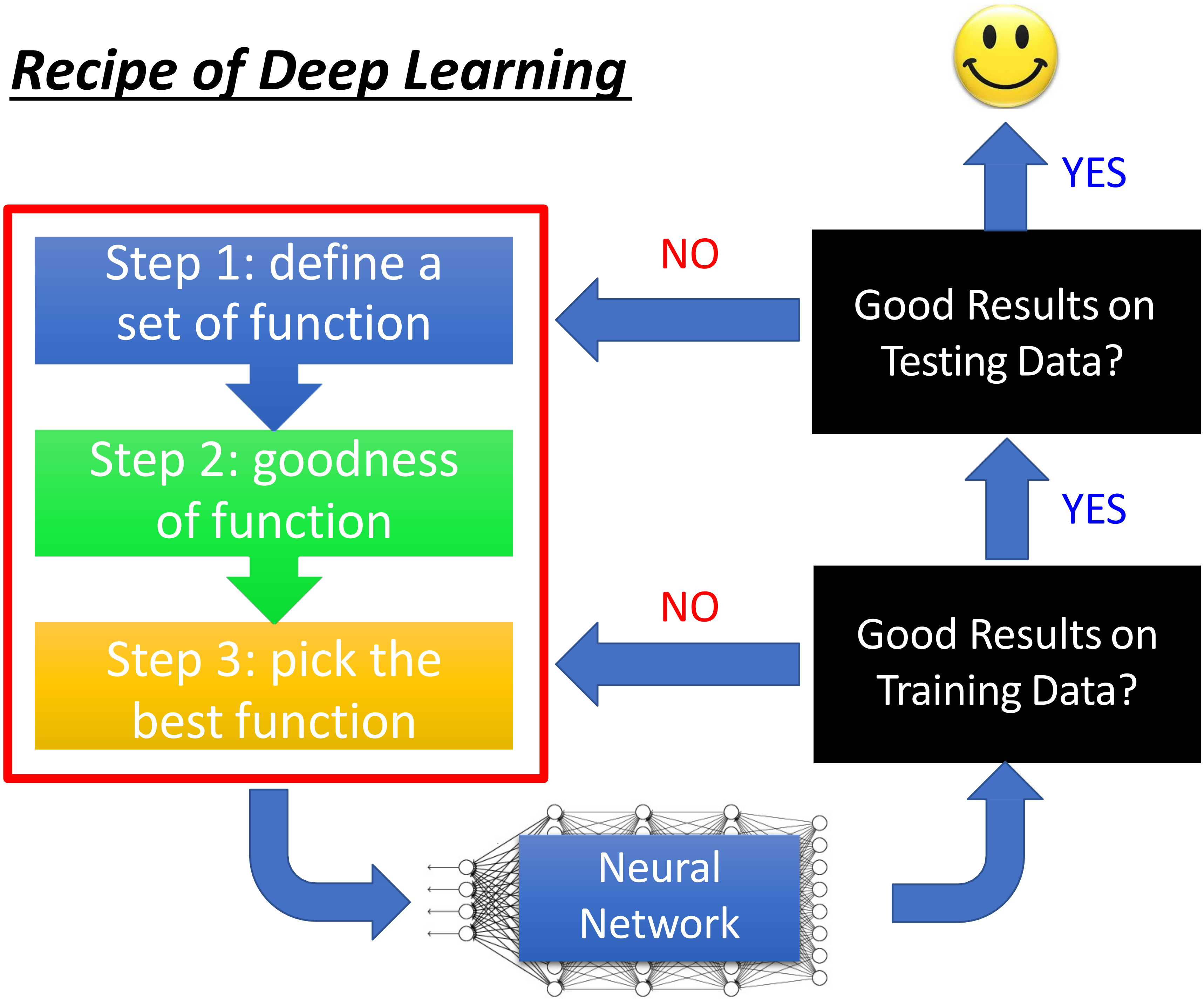
	Accuracy
Noisy	0.50
+ dropout	0.63

Recipe of Deep Learning



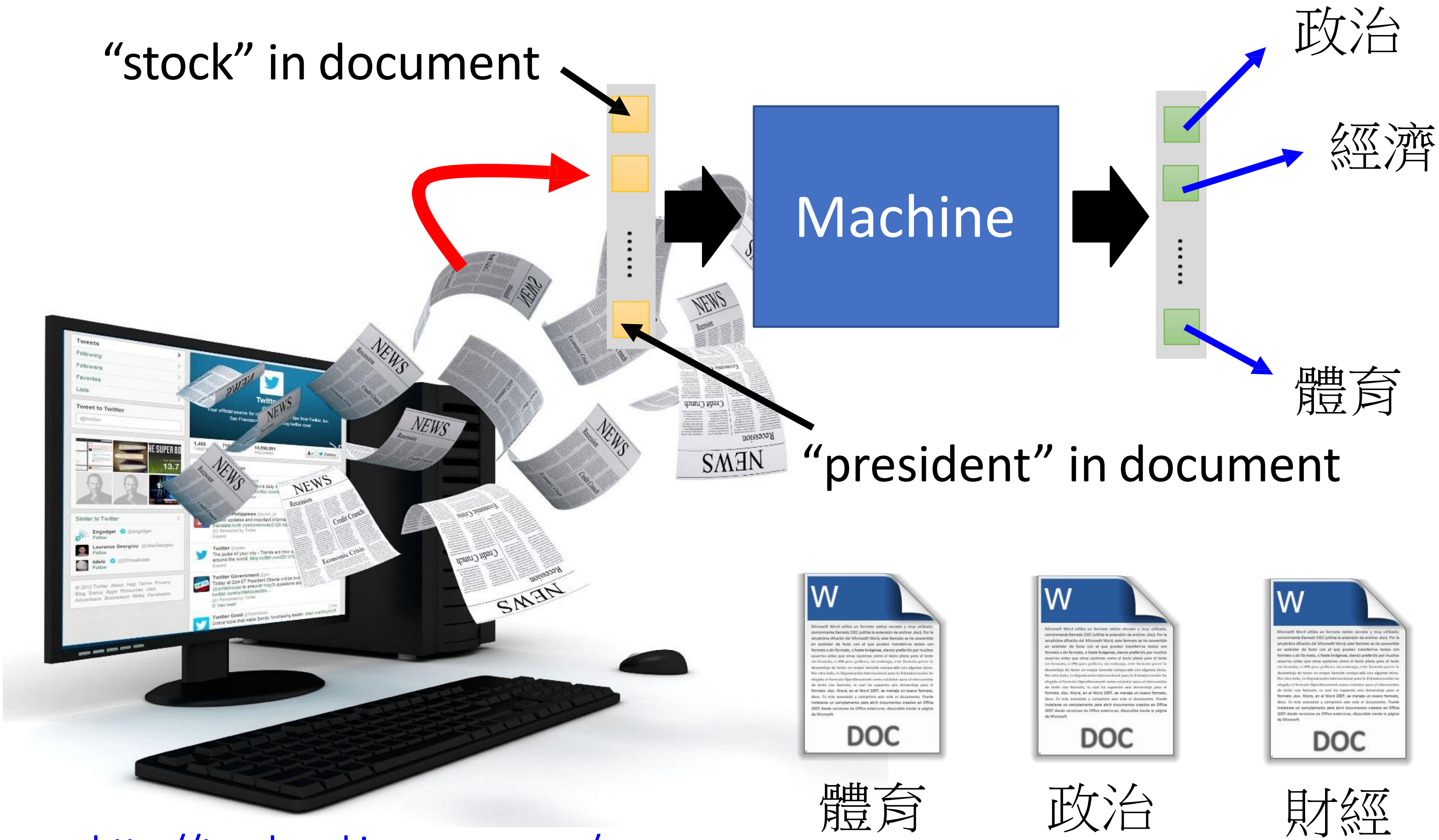
Concluding Remarks of Lecture II

Recipe of Deep Learning



Let’s try another task

Document Classification



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

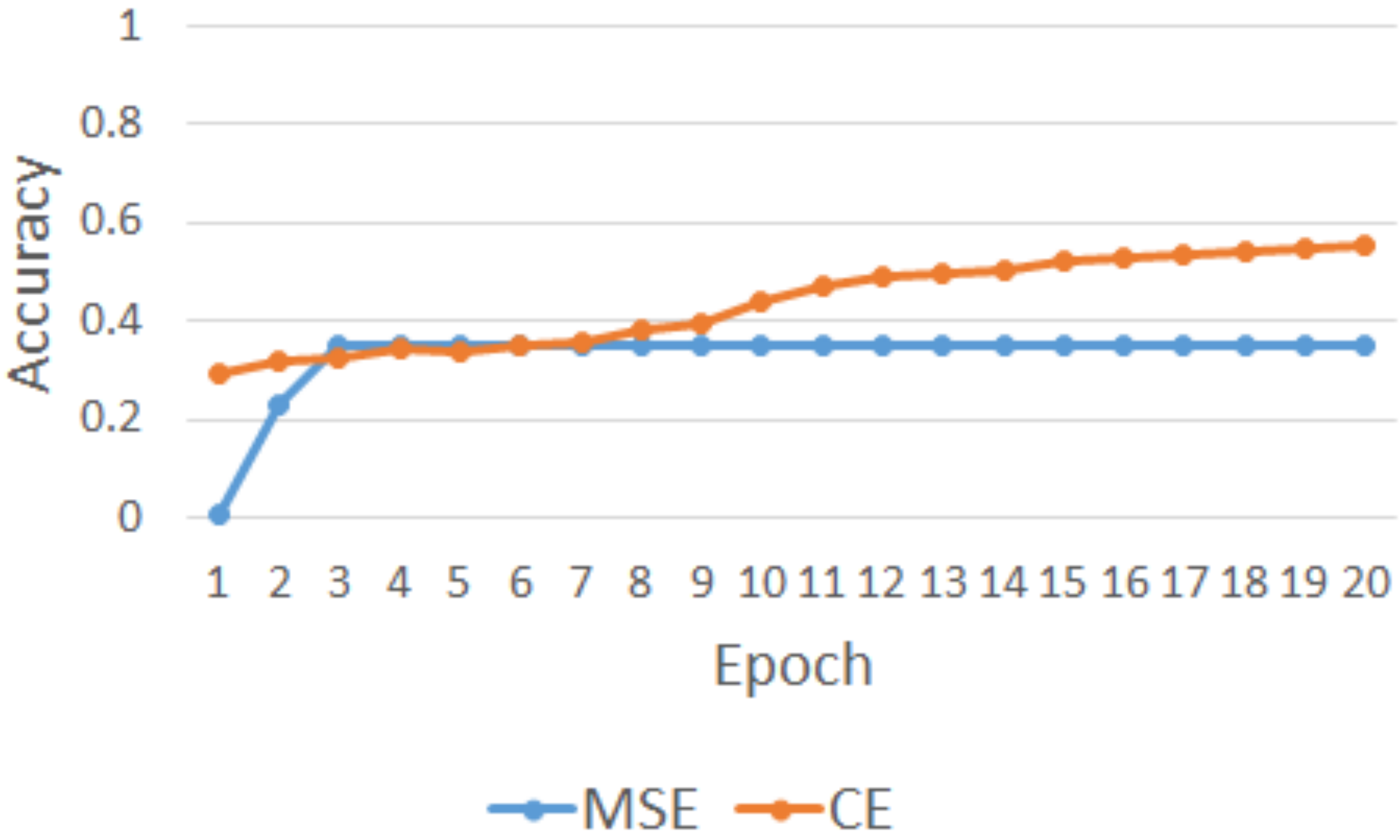
Data

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

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

In [10]: x_test.shape
Out[10]: (2246, 1000)

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

```
In [12]: x_train[0]
Out[12]:
array([ 0.,  1.,  1.,  0.,  1.,  1.,  1.,  1.,  1
        0.,  0.,  1.,  1.,  1.,  0.,  1.,  0.,  0
        1.,  0.,  0.,  1.,  1.,  0.,  1.,  0.,  0
        1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0
        1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0
        0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  1.,  1.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,
        0.,  0.,  0.,  0.,  0.,  1.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  1.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  1.,  1.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,
        0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
```

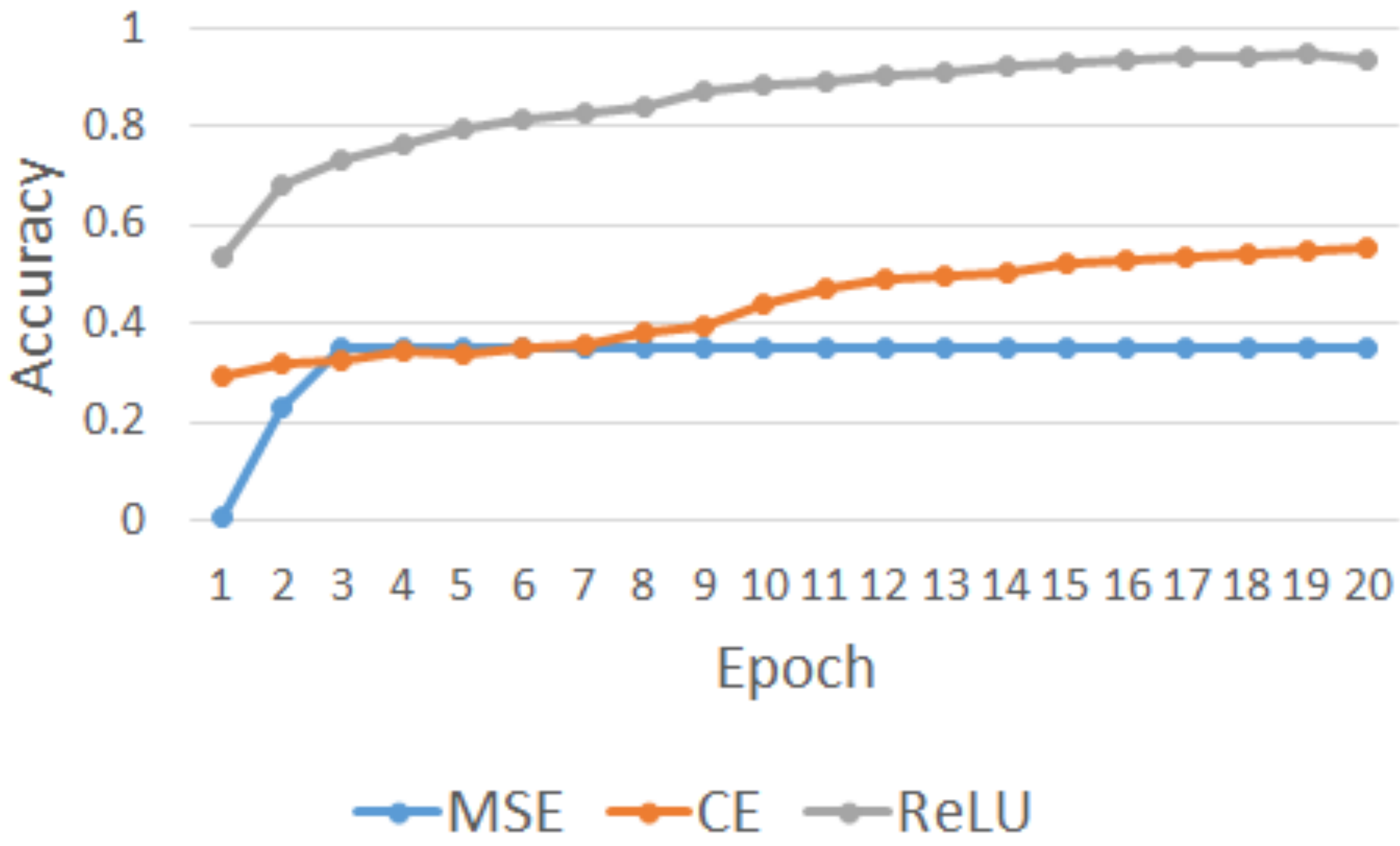
```
In [13]: y_train[0]
Out[13]:
array([ 0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.])
```

MSE



1
0.8
0.6
0.4
0.2
0
Accuracy
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Epoch
MSE
CE

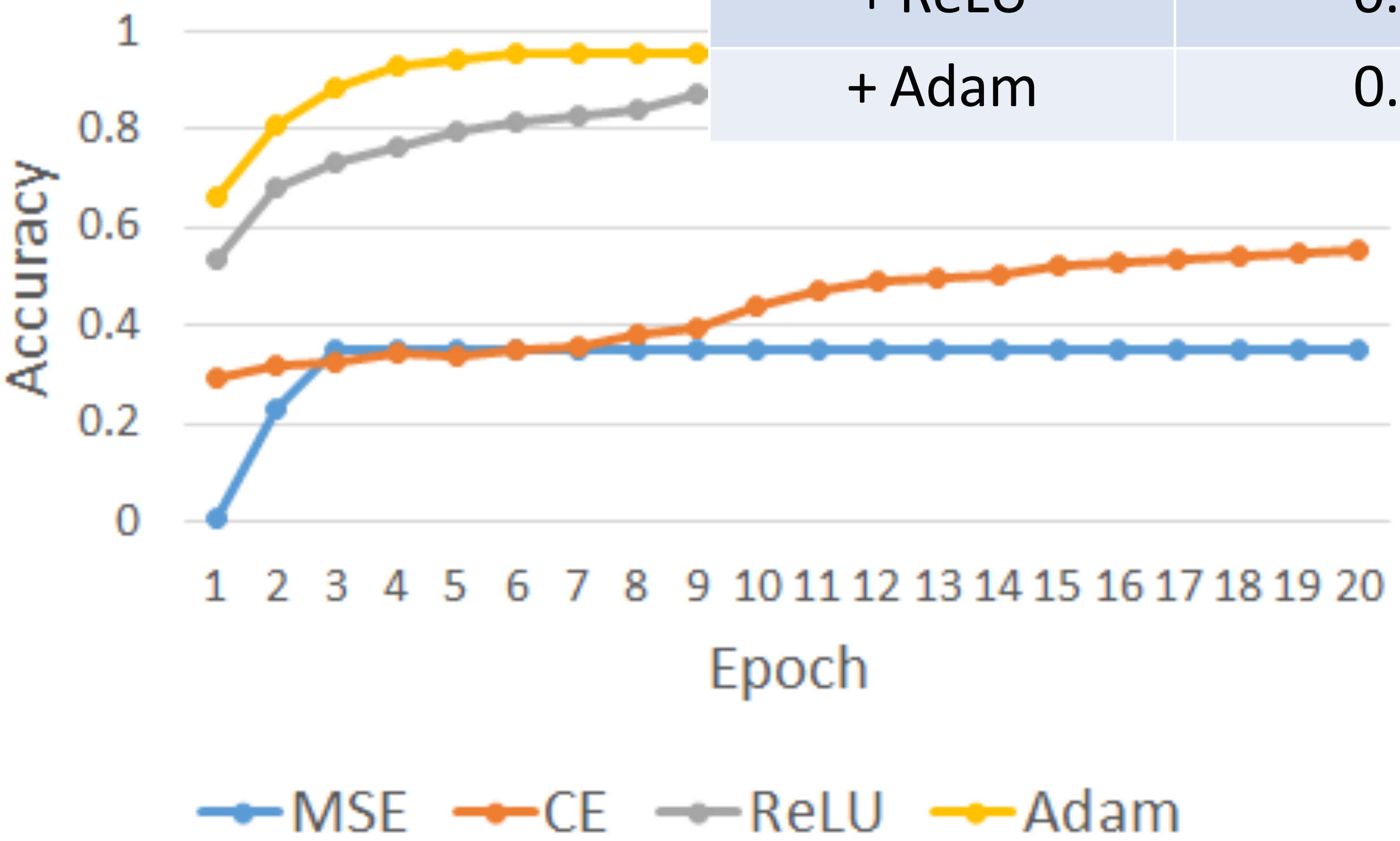
ReLU



1
0.8
0.6
0.4
0.2
0
Accuracy
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Epoch
MSE
CE
ReLU

Adaptive Learning

	Accuracy
MSE	0.36
CE	0.55
+ ReLU	0.75
+ Adam	0.77



Dropout

	Accuracy
Adam	0.77
+ dropout	0.79