

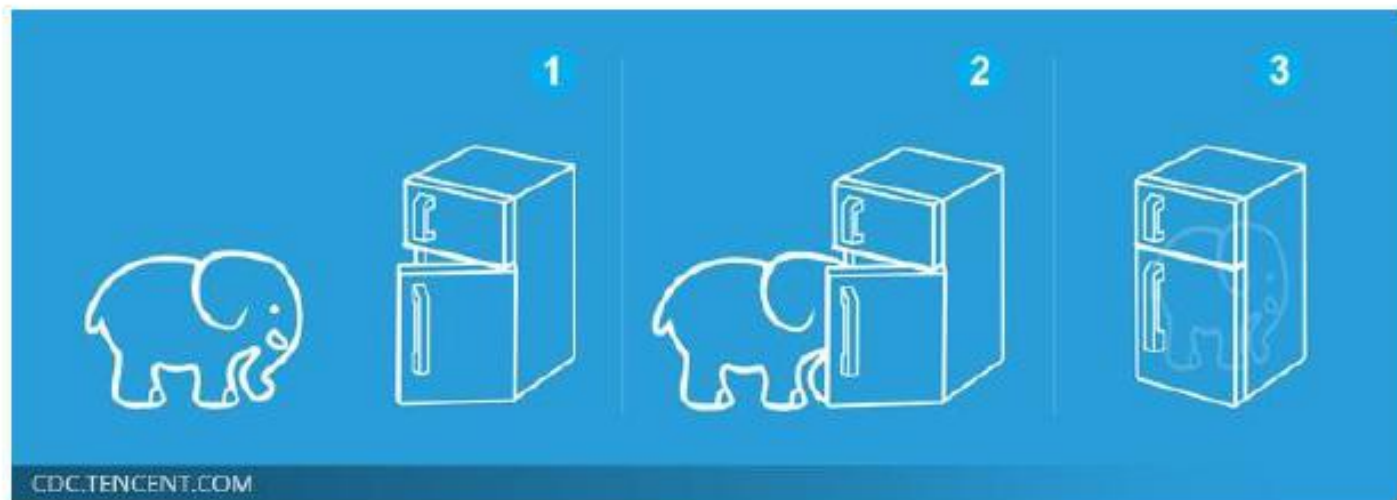
神经网络

一个简单实现深度学习的工具
--keras

Three Steps for Deep Learning



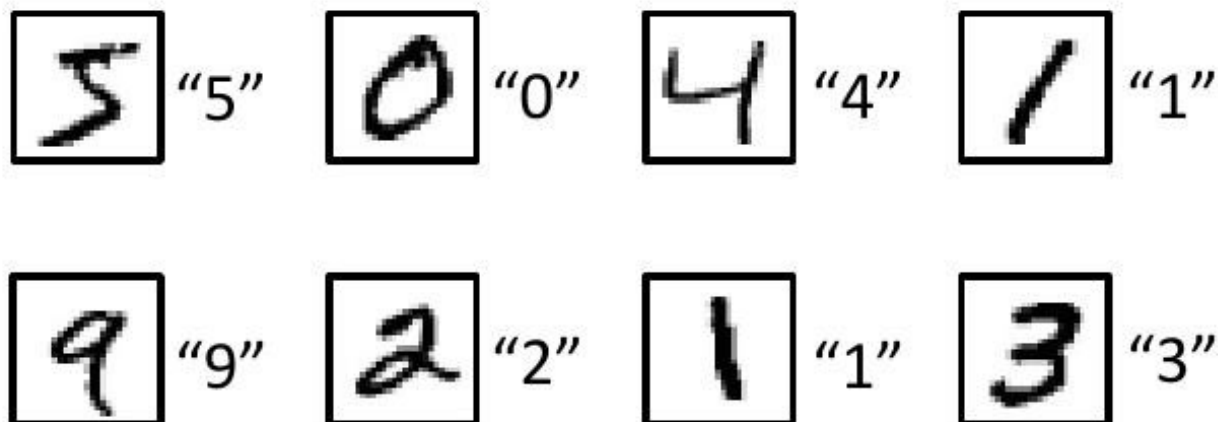
Deep Learning is so simple



数据集— mnist (手写数字识别)

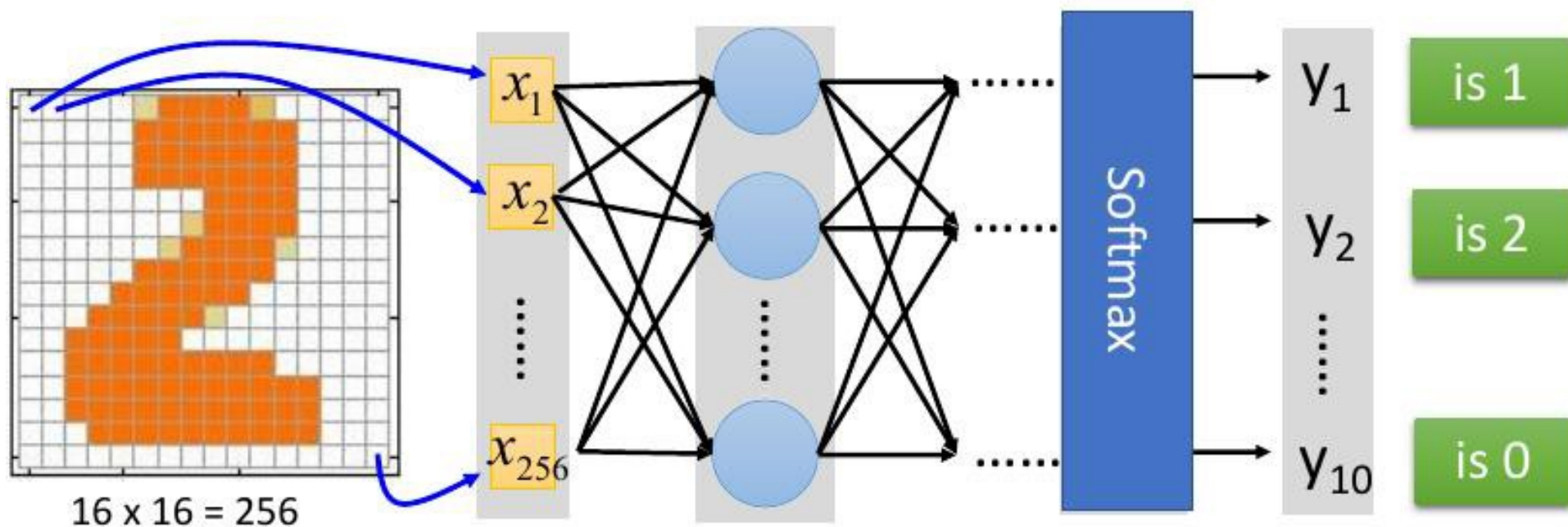
Training Data

- Preparing training data: images and their labels



The learning target is defined on the training data.


Learning Target




Ink \rightarrow 1

No ink \rightarrow 0

The learning target is

Input:  \rightarrow y_1 has the maximum value

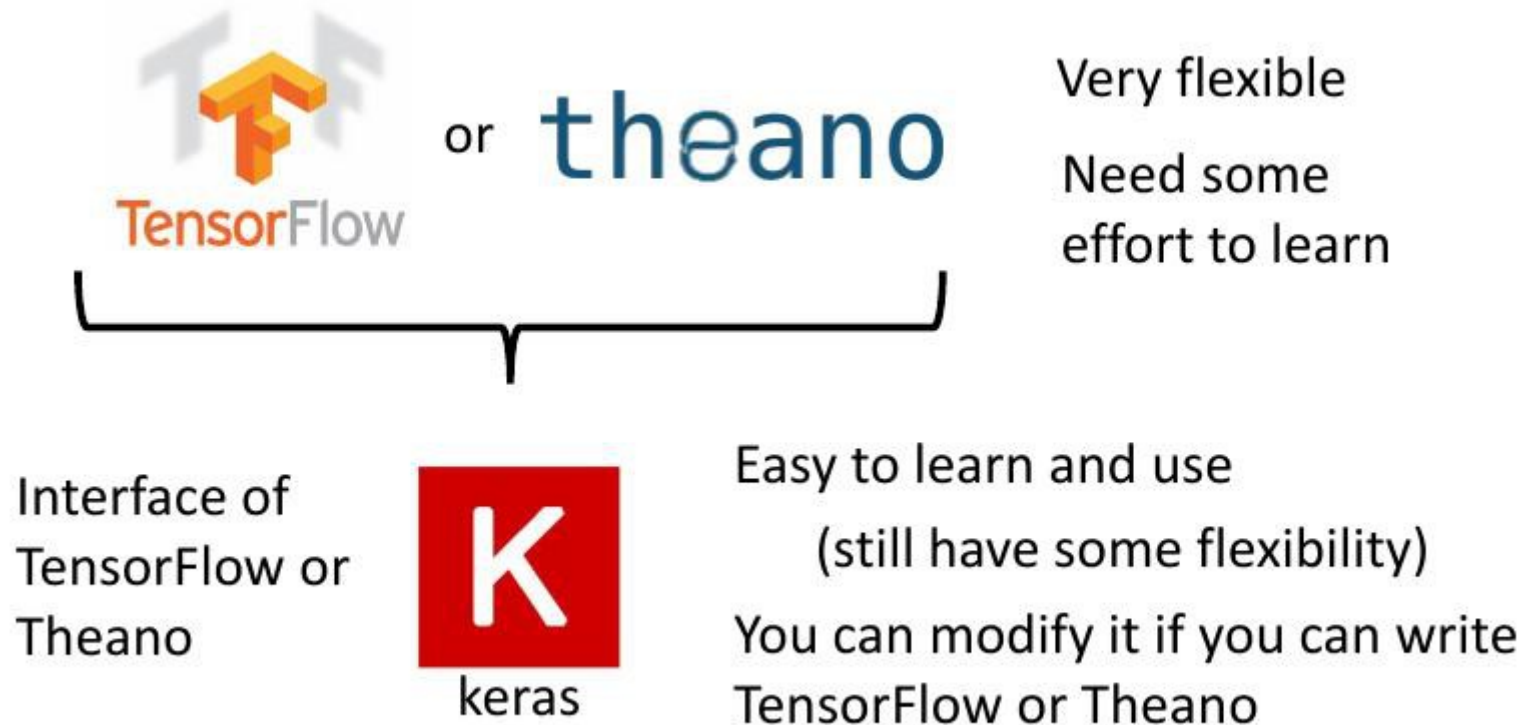
Input:  \rightarrow y_2 has the maximum value

mnist 数据的预处理

有一个脚本文件，是将图像数据转化为 keras 可以处理的 array 数据。具体的细节代码上有注释。

Keras

“Hello World” for Deep Learning



一些基本概念

- **符号计算**：符号主义的计算首先定义各种变量，然后建立一个“计算图”，计算图规定了各个变量之间的计算关系。建立好的计算图还需要编译一些内部细节。然而，此时的计算图还是一个“空壳子”，里面没有任何实际的数据，只有当你把需要运算的输入放进去后，才能在整个模型中形成数据流，从而形成输出值。
- **张量**：张量可以看作是向量、矩阵的自然推广，我们用张量来表示广泛的数据类型。

- **模型**：Keras 里面有两个模型。一种叫 Sequential-- 称为序贯模型，也就是单输入单输出，一条路通到底，层与层之间只有相邻关系，跨层连接统统没有。还有一种叫 functional model API-- 只要这个东西接收一个或一些张量作为输入，然后输出的也是一个或一些张量，那不管它是什么鬼，统统都称作“模型”。

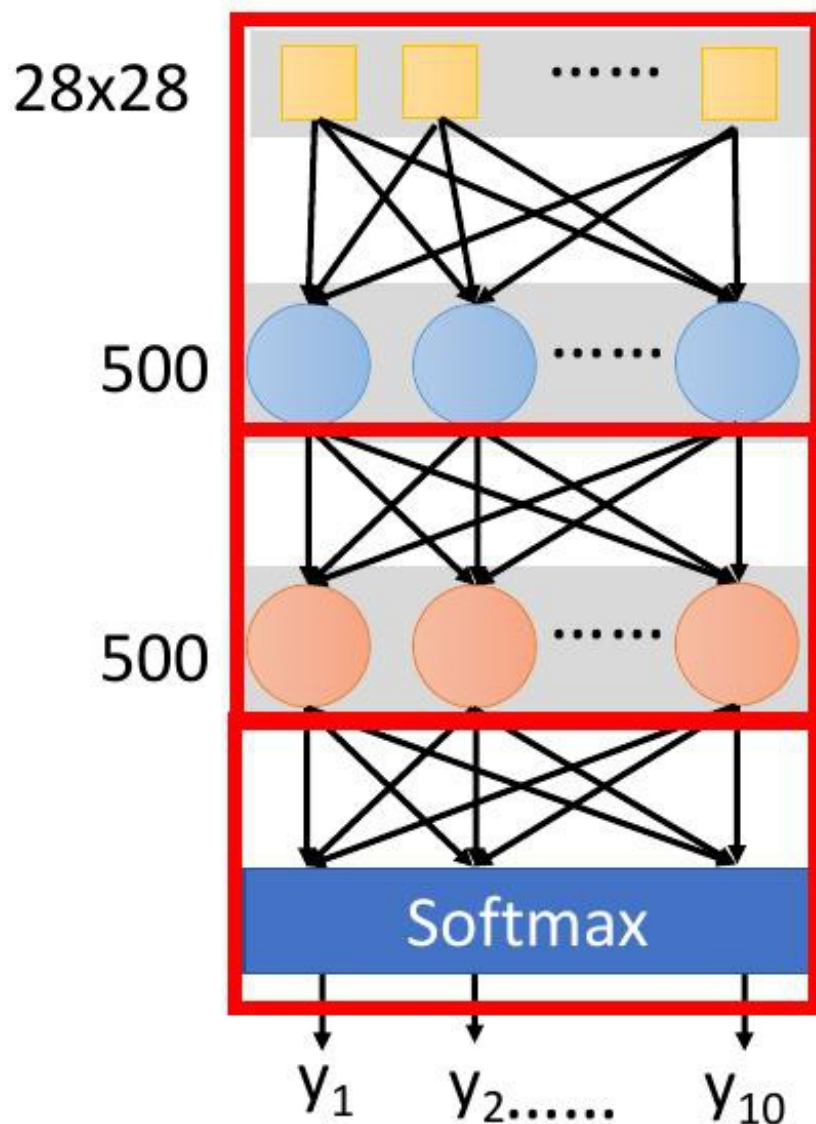
Keras-- 安装

安装方法很多，具体的写了一个 word 文档（都是从网上查出来的方法）。

注意：这种方法在 64 位的 ubuntu 系统下实现的。

Keras--Sequential

一、建立一个“计算图”



```
model = Sequential()
```

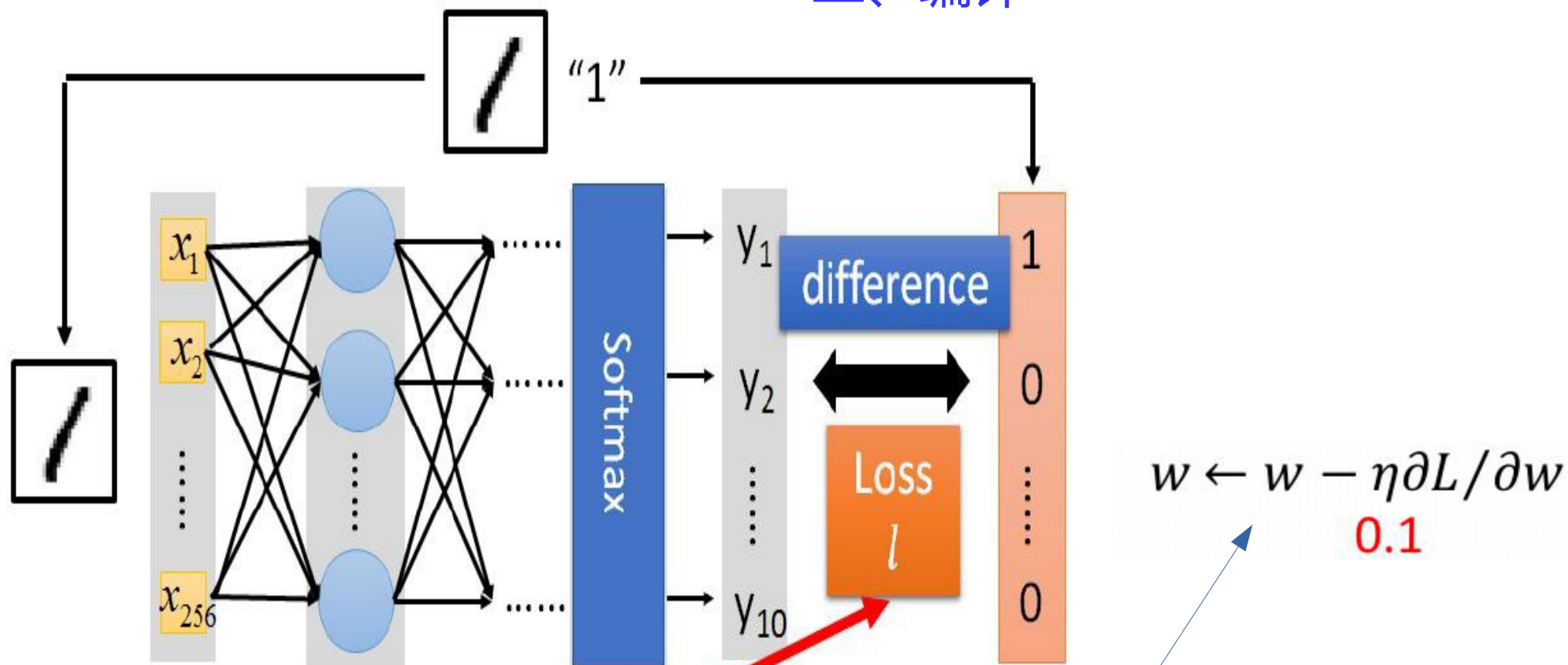
```
model.add( Dense( input_dim=28*28,  
                  output_dim=500 ) )  
model.add( Activation('sigmoid') )
```

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

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

Keras--Sequential

二、编译

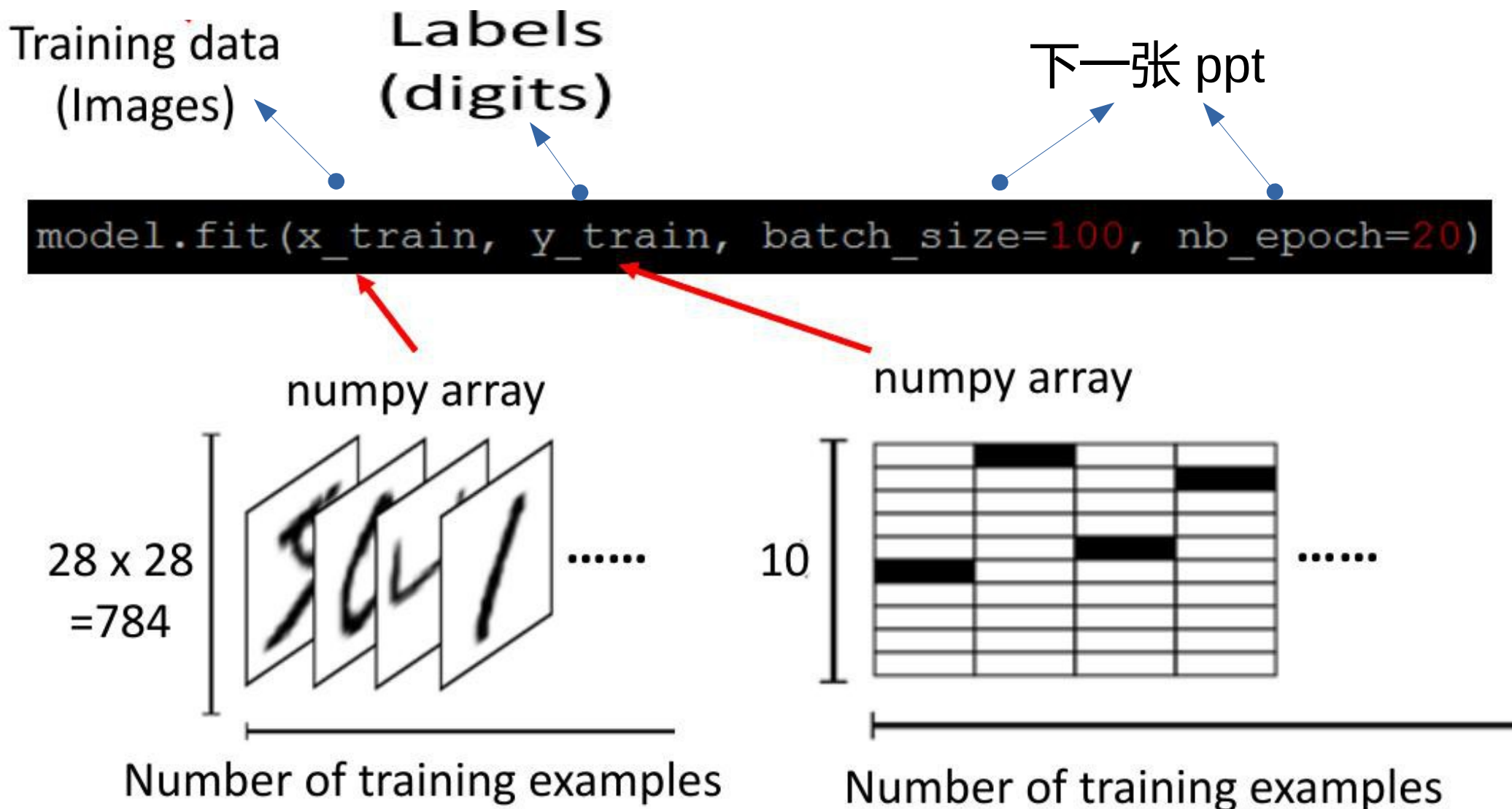


```
model.compile(loss='mse',  
              optimizer=SGD(lr=0.1),  
              metrics=['accuracy'])
```

评价指标

Keras--Sequential

三、训练模型

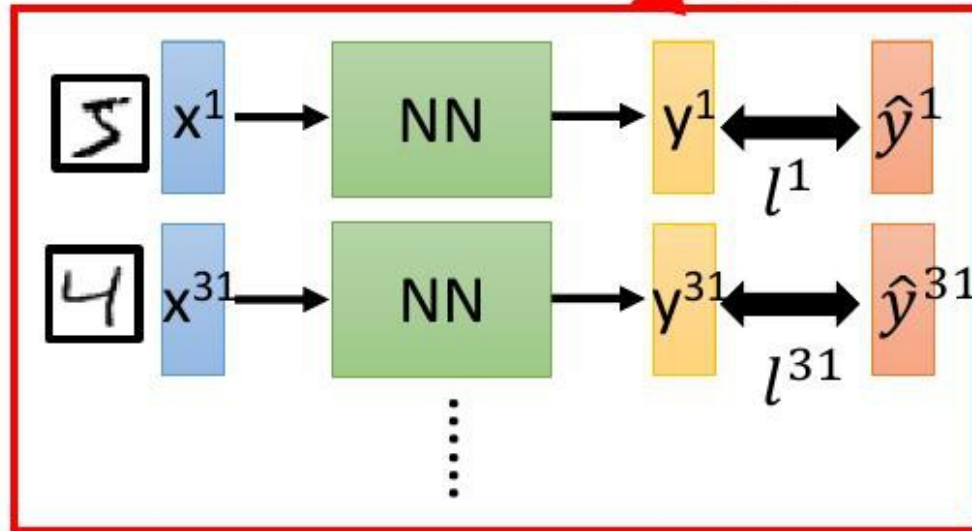


Keras-Sequential

Mini-batch

```
model.fit(x_train, y_train, batch size=100, nb epoch=20)
```

Mini-batch



100 examples in a mini-batch

Repeat 20 times

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

one epoch

Keras-Sequential

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])
```

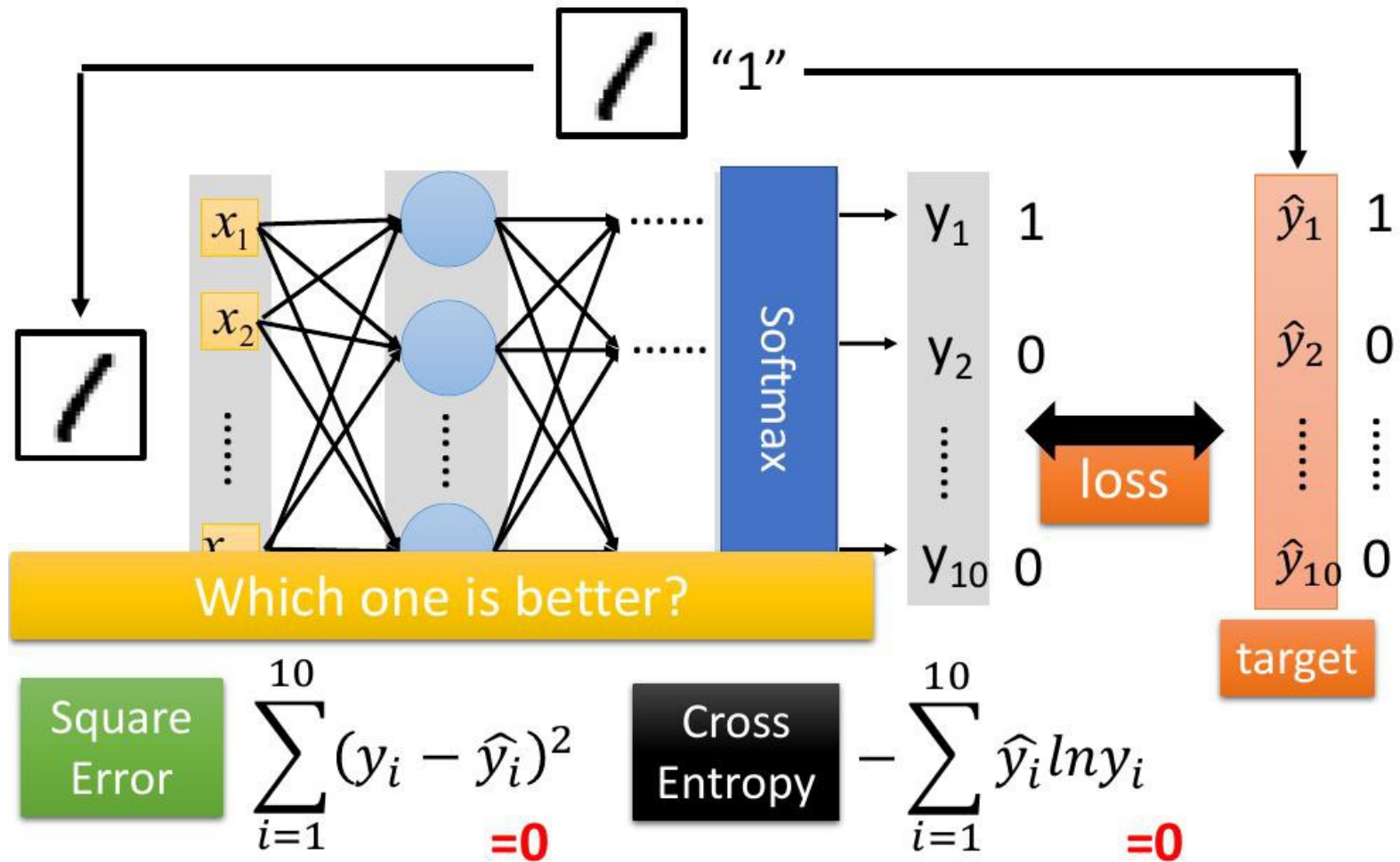
case 2:

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



Loss

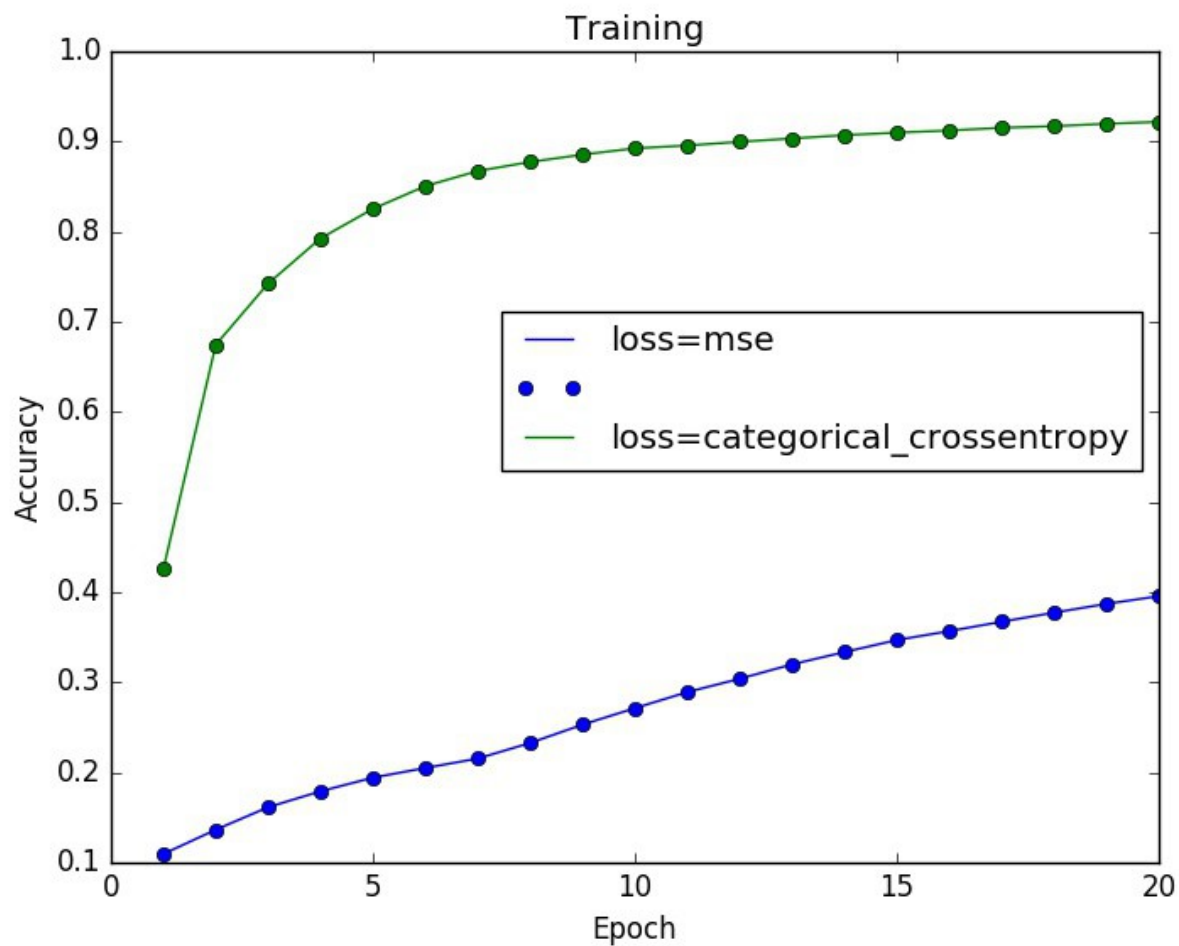
Choosing Proper Loss



```
model1.compile(loss='mse',  
optimizer=SGD(lr=0.01),metrics=['accuracy'])
```

```
model2.compile(loss='categorical_crossentropy',  
optimizer=SGD(lr=0.01),metrics=['categorical_accuracy'])
```


Lose



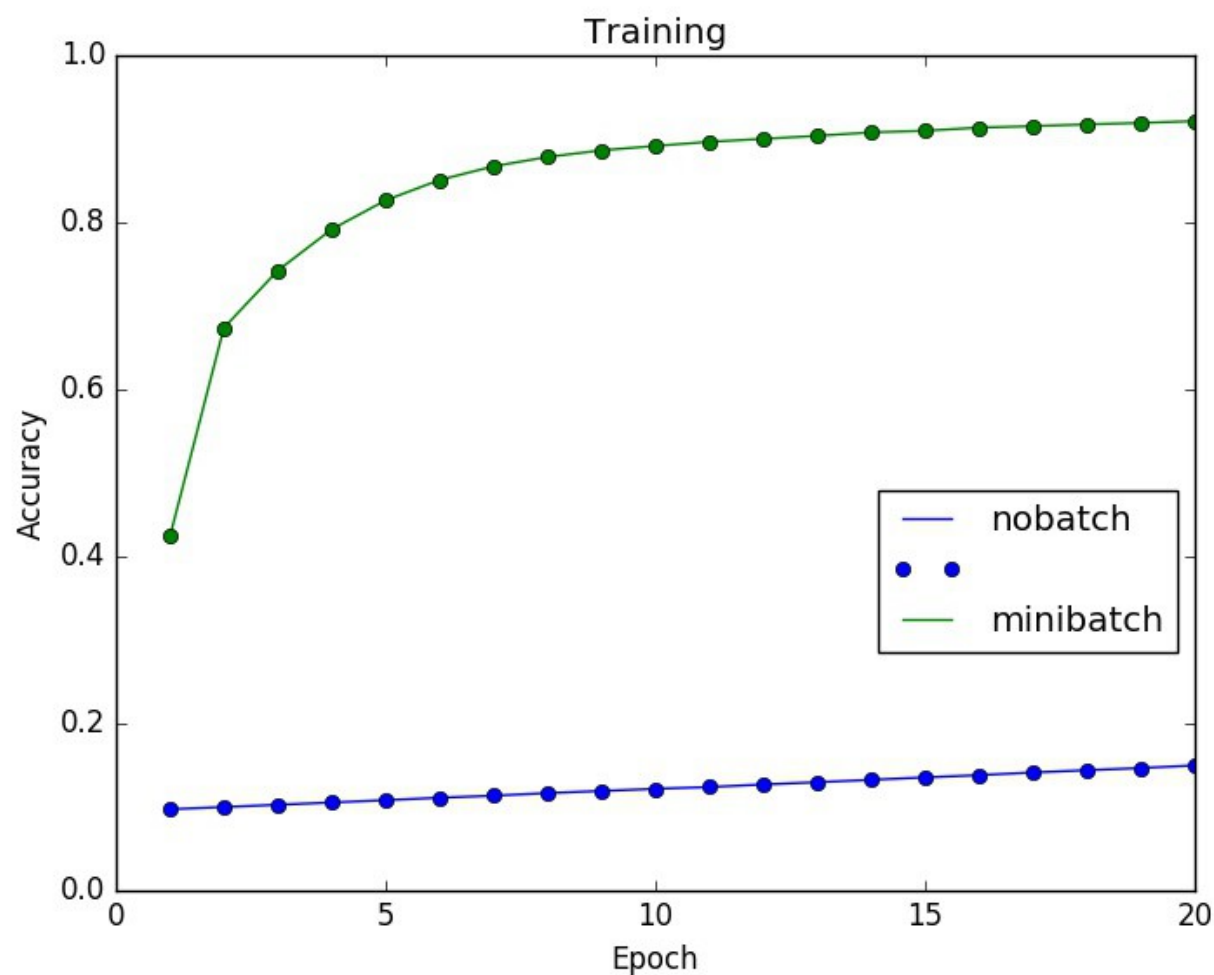
On Testset	Accuracy
Mse	0.395
Cross Entropy	0.919

当使用 softmax 作为输出层的激活函数时，对于多分类问题来说，使用交叉熵作为损失函数更好一些。



Minibatch

Minibatch



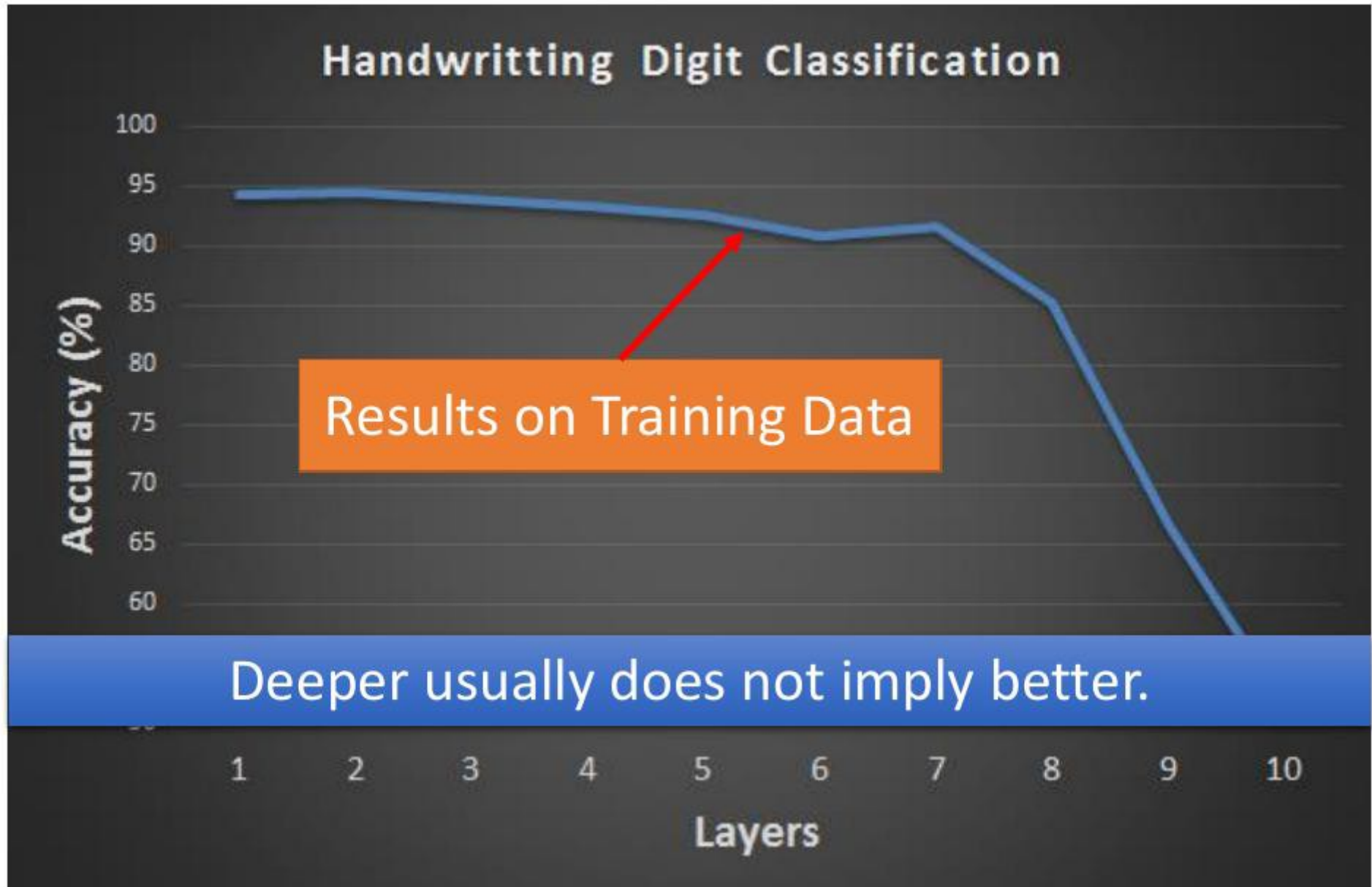
On Testset	Accuracy
nobatch	0.147
minibatch	0.919

图中可以看出有
batch 比没有
batch 效果差很多

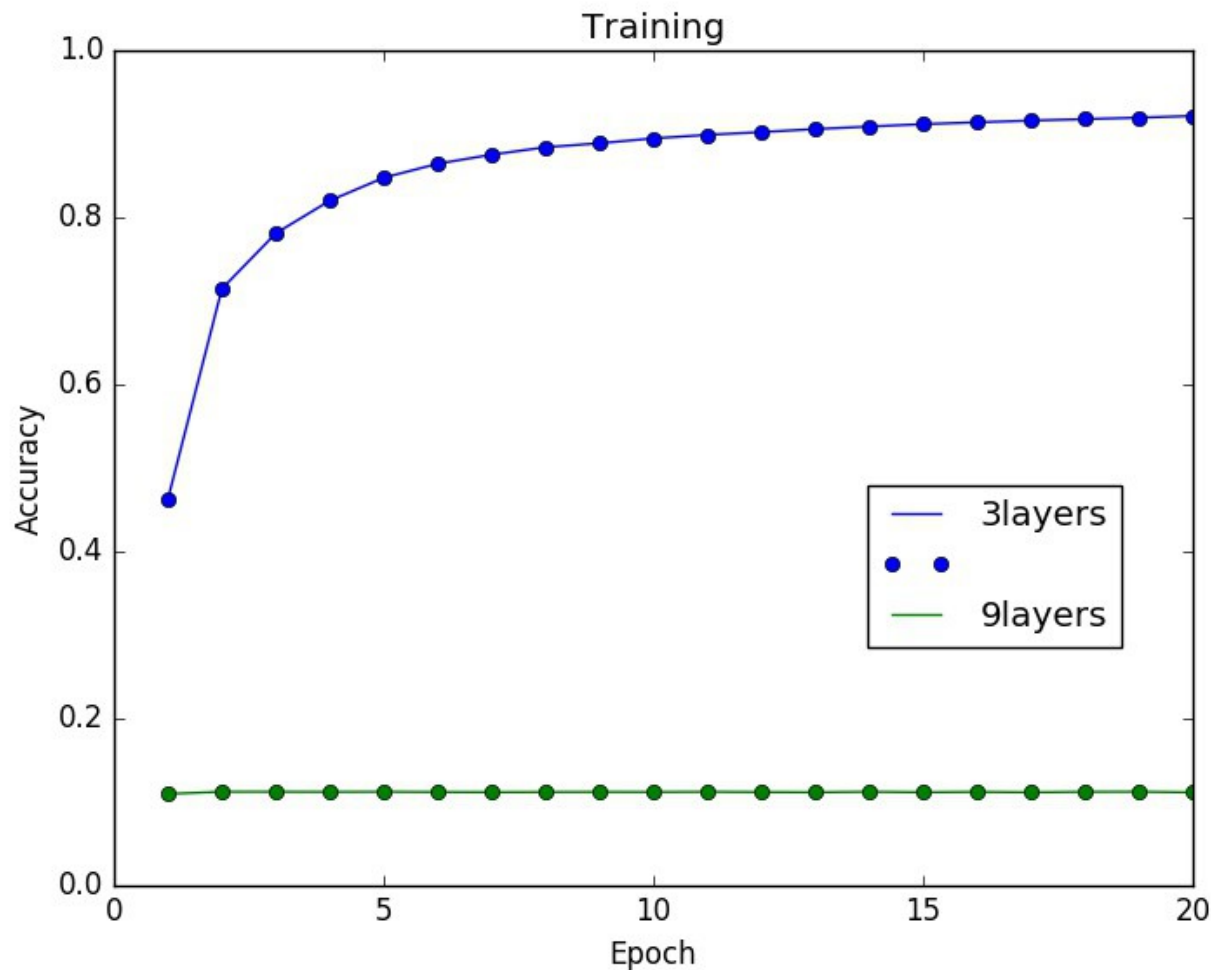


Layers and Activation

Hard to get the power of Deep ...



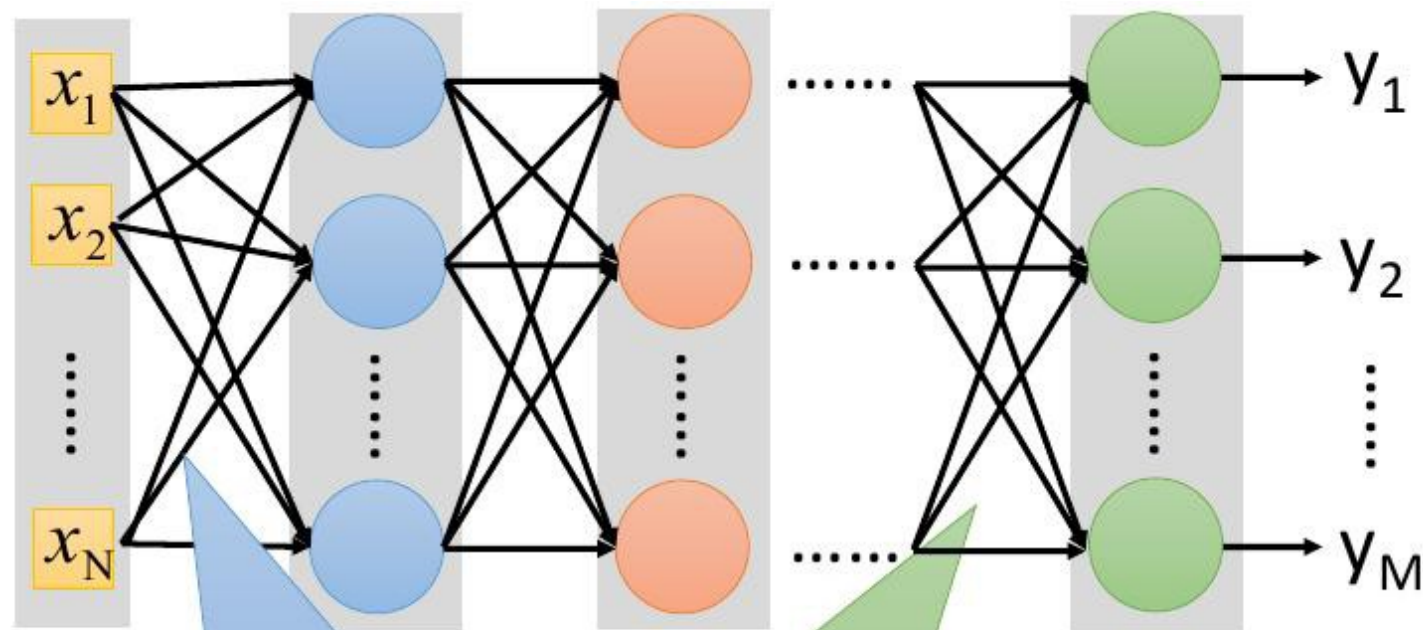
Layers



On Testset	Accuracy
3layers	0.919
9layers	0.114

当激活函数同为 sigmoid 时，隐层为 7 时的效果比仅有 1 层时差很多

多隐层时，会出现梯度弥散问题



Smaller gradients

Learn very slow

Almost random

Larger gradients

Learn very fast

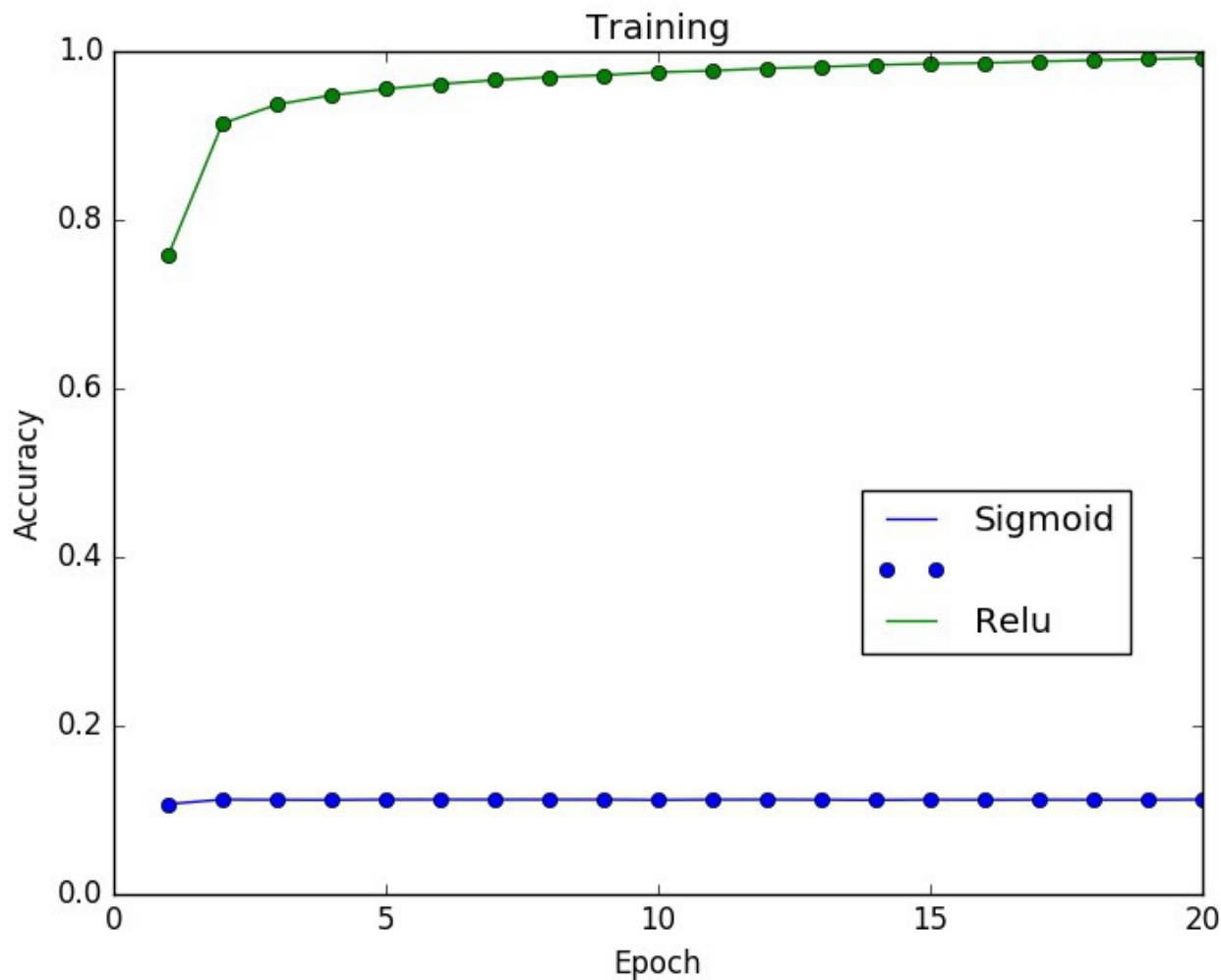
Already converge

改变激活函数的形式

In 2006, people used RBM pre-training.
In 2015, people use ReLU.

Activation

9layers



On Testset	Accuracy
Sigmoid	0.113
Relu	0.963

当隐层数较多时，
relu比sigmoid
好很多



Optimizer

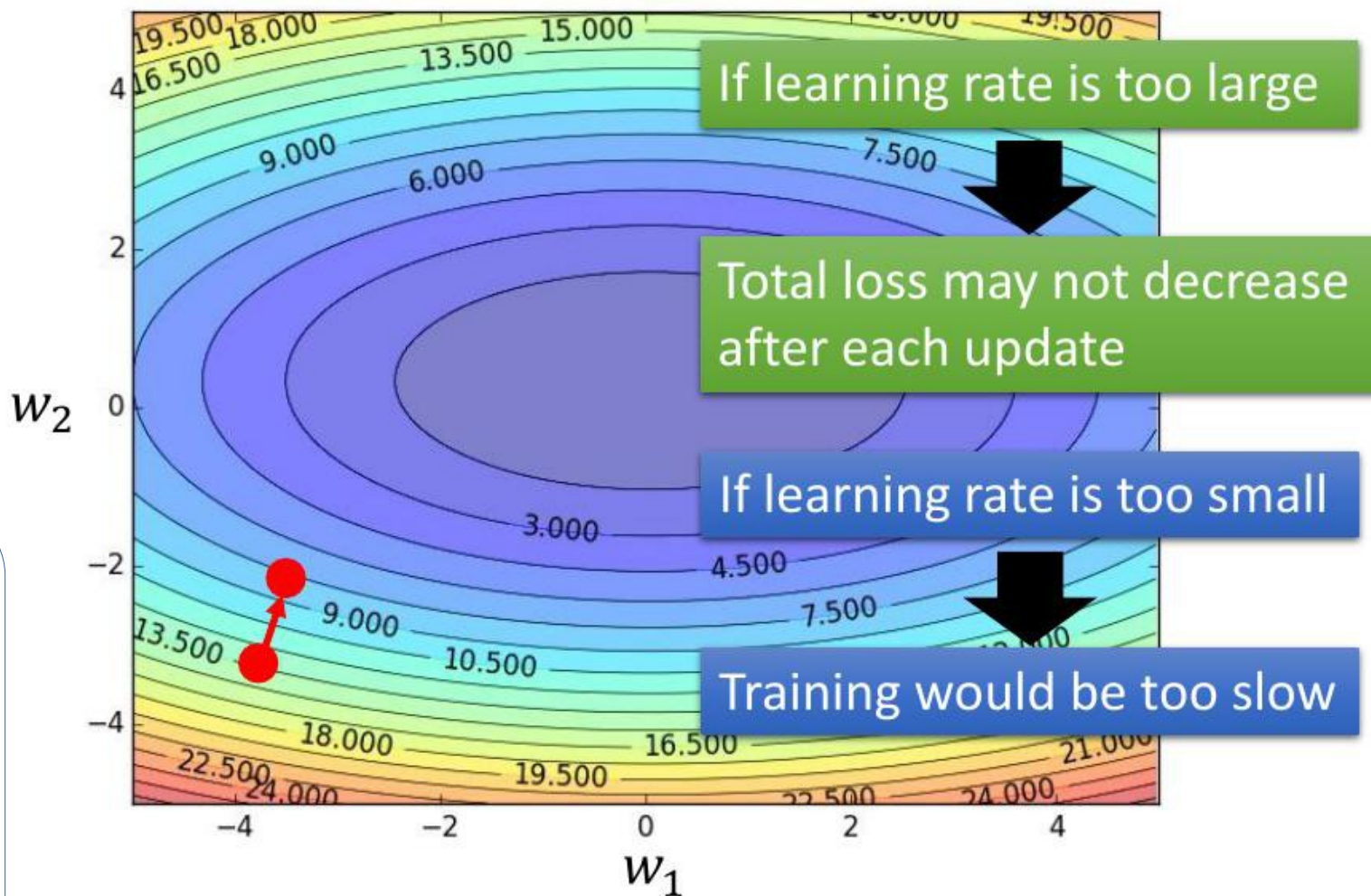
Learning Rates

Best idea

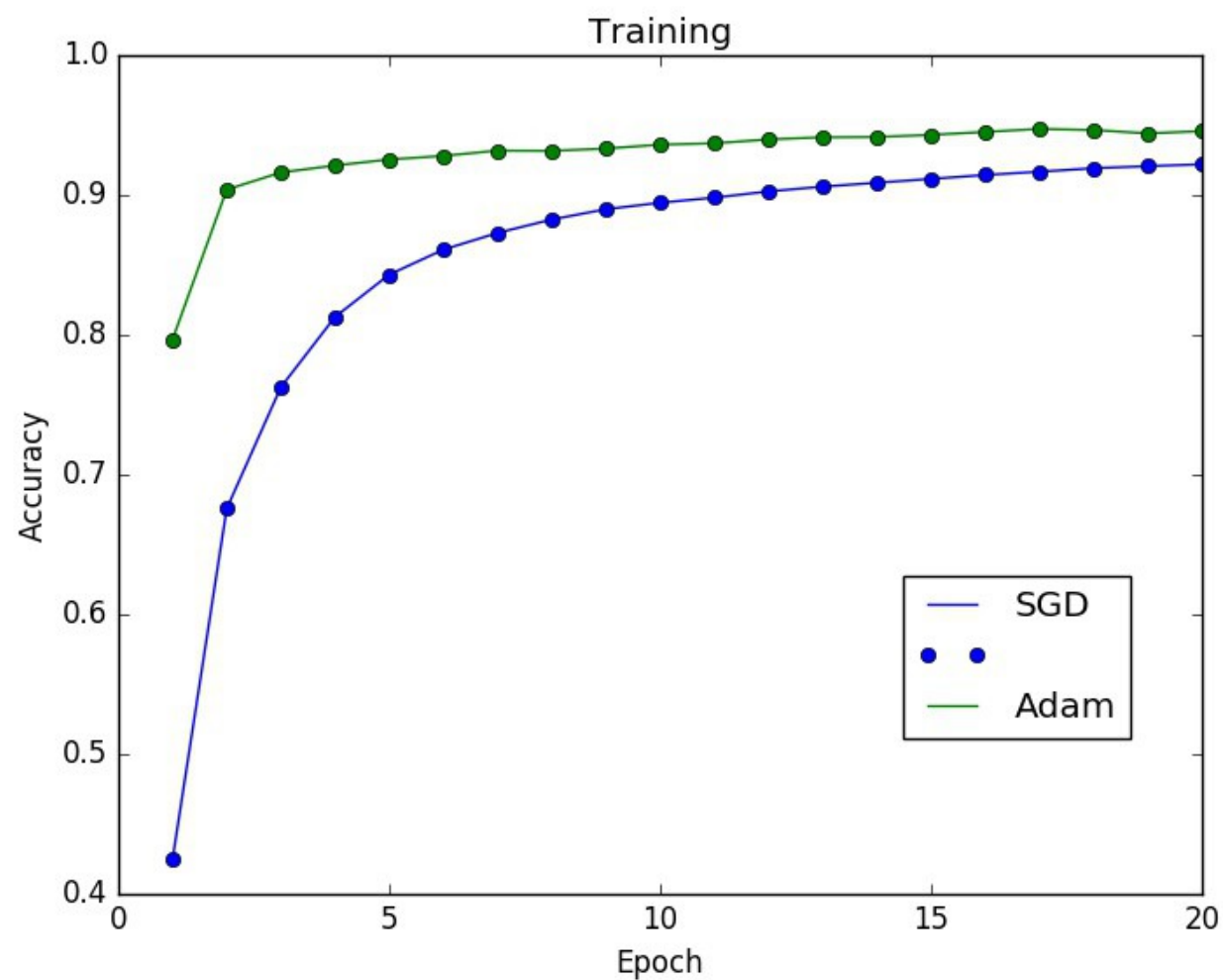
开始的时候，
离目标值较远，
需要一个较大的
学习率

在几次训练之后，
我们离目标值很
近了，就需要一个
很小的学
习率值了

Set the learning
rate η carefully



Optimizer



On Testset	Accuracy
SGD	0.922
Adam	0.937

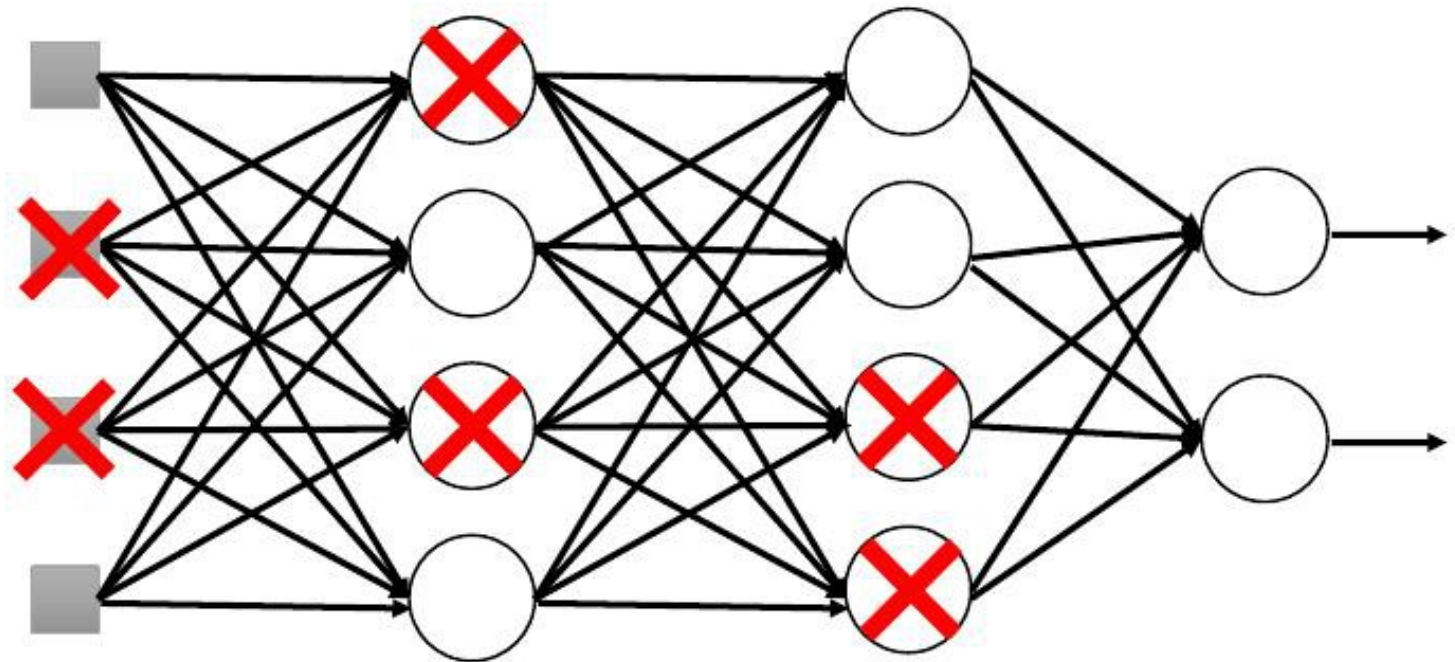
Adam 比 SGD 收敛
的速度更快



Dropout

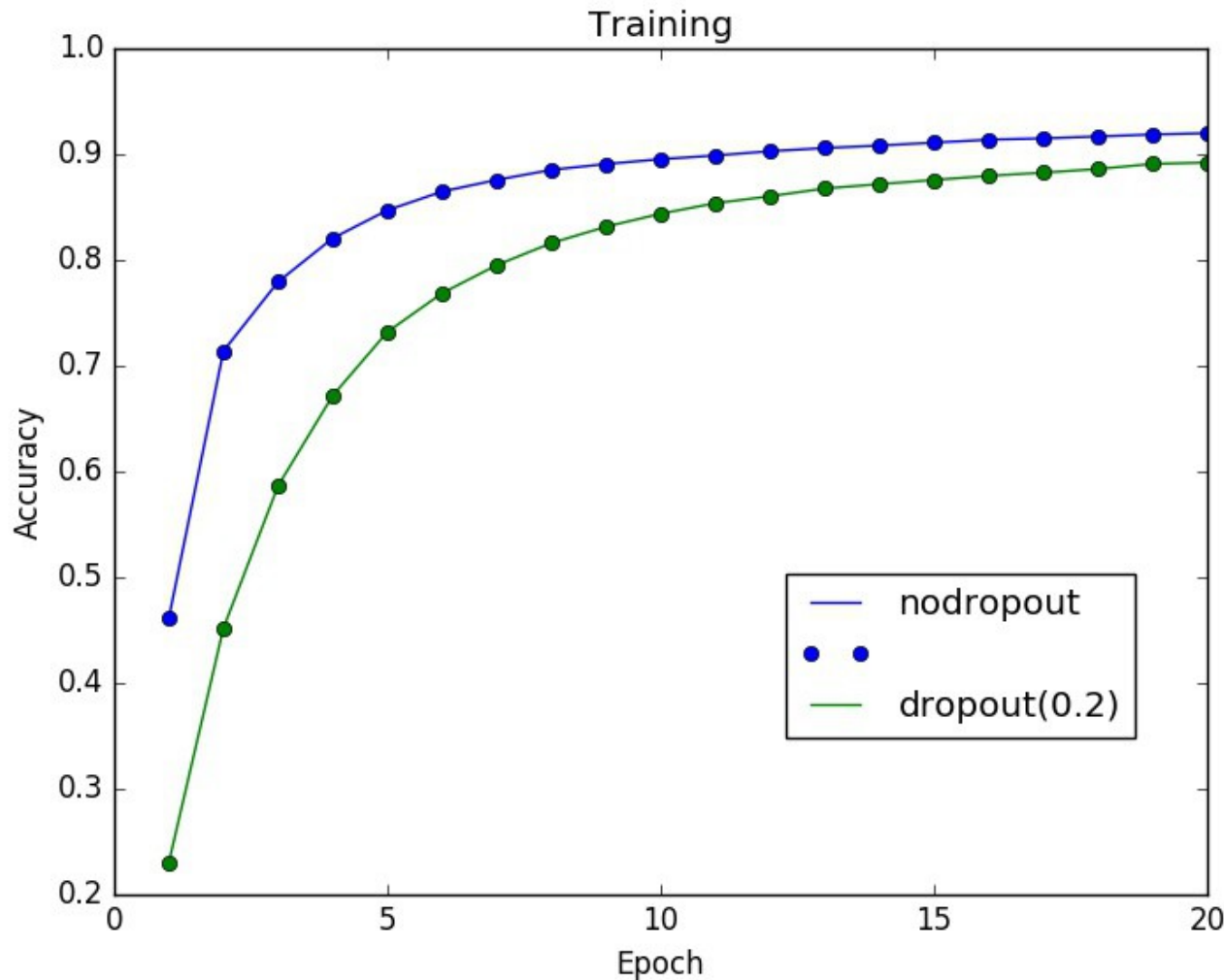
Dropout

Training:



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

Dropout

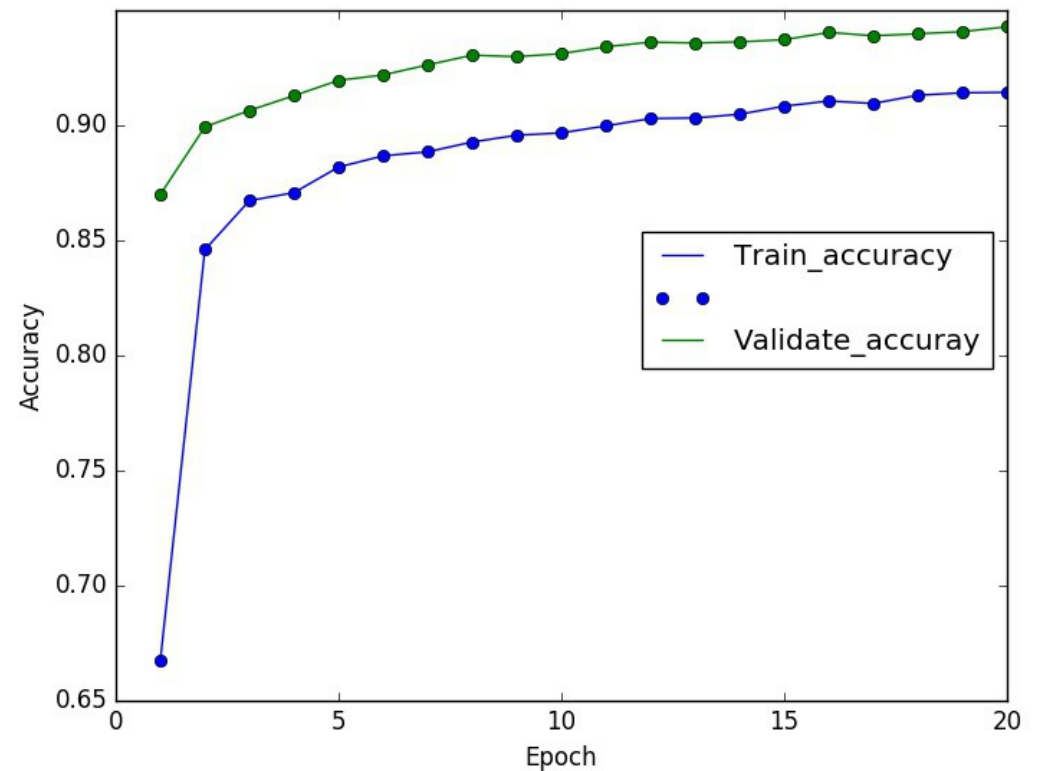
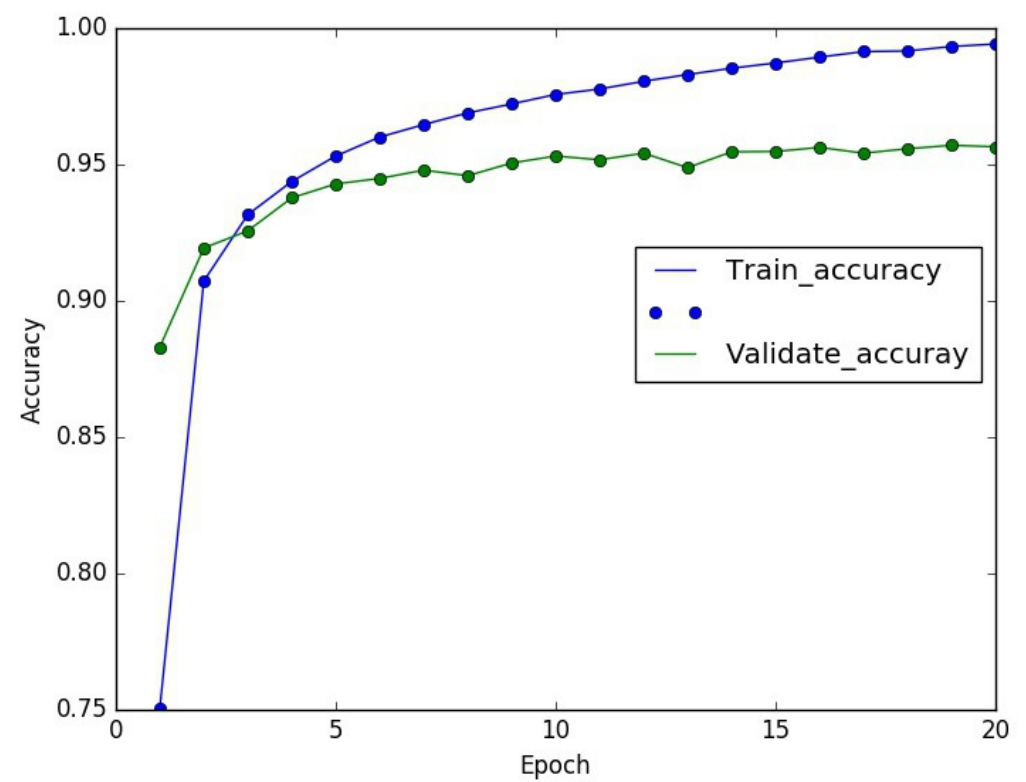


On Testset	Accuracy
nodropout	0.906
dropout	0.918

当断开比例为 0.2 时，在训练集上差不多，测试集上也差不多，从经验上说加 dropout 应该在测试集上的效果更好一点，但这也不是个确定标准

Step 3: pick
the best
function

On Testset	Accuracy
simple	0.94
fuzha	0.96



谢谢，大家！