Beijing Forest Studio

北京理工大学信息系统及安全对抗实验中心

深度学习讨论会

陈倩柔　博士

2017年11月30日

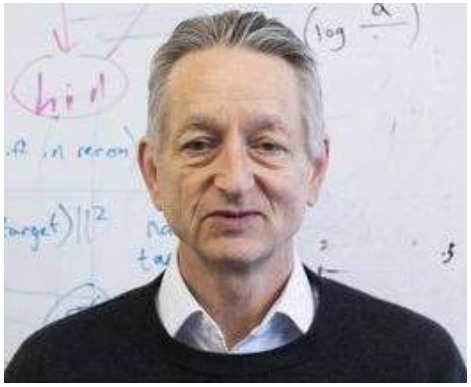
内容提要

- 背景简介
  - 什么是深度学习
  - 深度学习的步骤
  - 深度学习的优点
- 算法原理
  - 一个简单的深度神经网络（DNN）
  - 基于Keras的实现
- 参考文献

背景简介 什么是深度学习

- 最早提出：由Geoffrey Hinton等于2006年在《Sciences》上发表的文章中。

- Deep Learning的含义
  - 最初/一般：指基于深度神经网络的学习，Deep对应的指神经网络的深度（层数），但没有具体地定义大于多少层的算深度网络
  - 从学习能力层面理解：能学到深层次（传统机器学习或者说显示特征无法描述）的内容

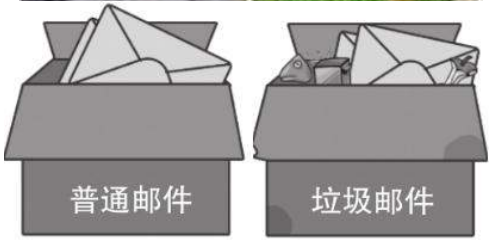


背景简介 深度学习的过程3-1

- 机器学习的应用：

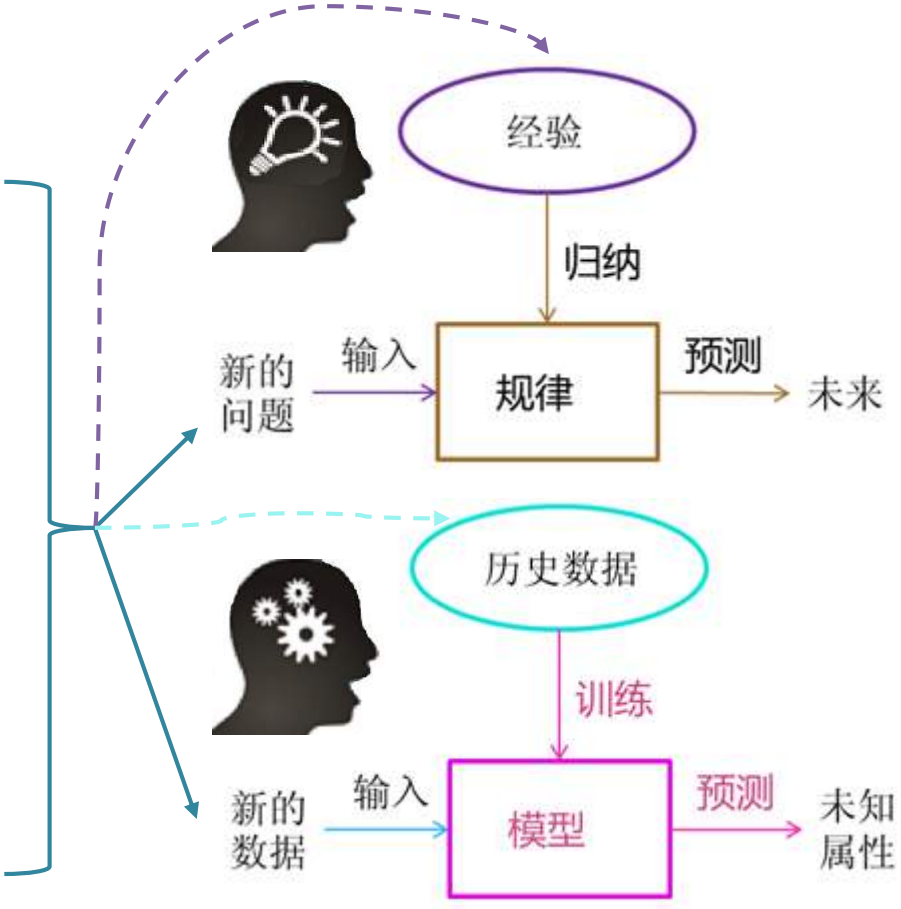
图像识别

文本分类



异常检测

疾病预测



背景简介 深度学习的过程3—2

- 机器学习过程：

标签：0猫，1狗

训练

训练数据

Step1：构建模型

函数集合
f1, f2, …

$$f = \frac{1}{1+e^{-\theta^T x}}$$

评价函数

Step2：定义目标函数

均值损失函数
对数损失函数

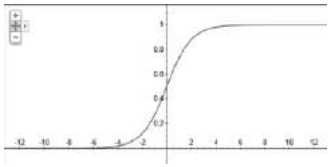
Step3：找到最优函数f*

梯度下降
拟牛顿法

新数据

使用f*

0=猫



f*（  ）=猫

背景简介 深度学习的过程3-3

- 深度学习过程：

训练

训练数据

标签：0猫，1狗

Step1：构建模型

函数集合
f1, f2, …

$$f = \frac{1}{1+e^{-\theta^T x}}$$

评价函数

Step2：定义目标函数

均值损失函数
对数损失函数

Step3：找到最优函数f*

梯度下降
拟牛顿法

新数据

使用f*

0=猫

f*（  ）=猫

背景简介 深度学习的优点

- 不用再提取特征

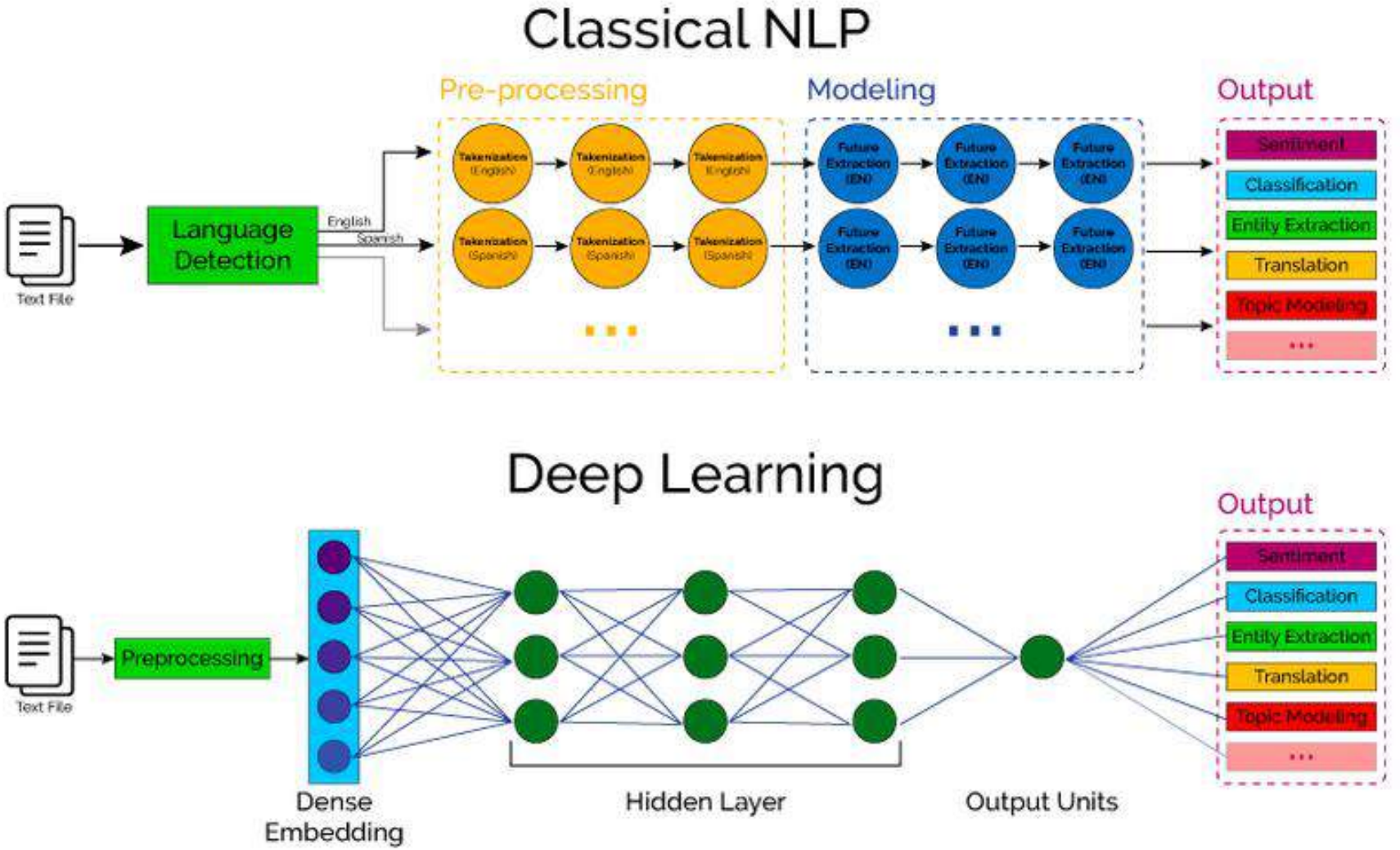
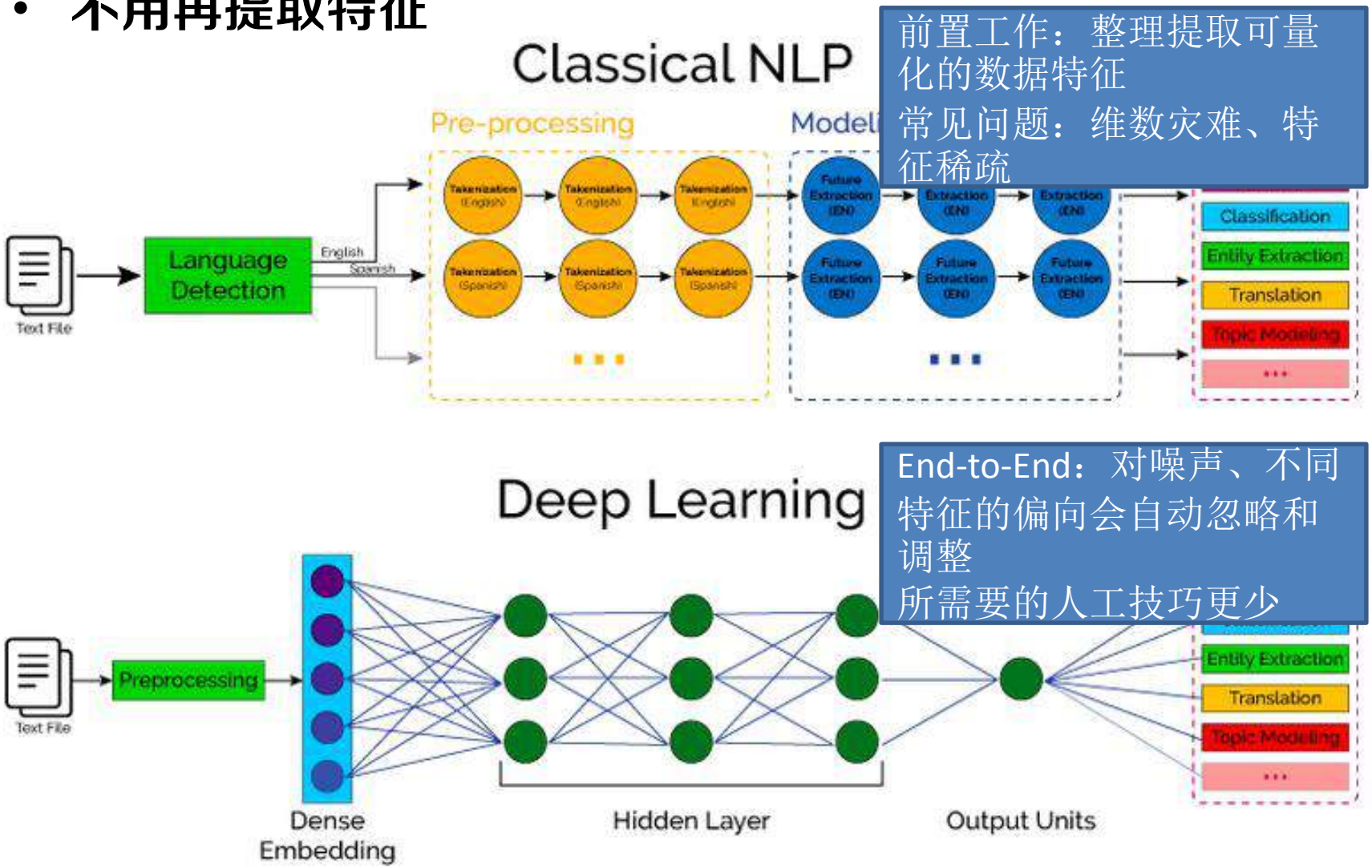


Image Source - blog.aylien.com

背景简介 深度学习的优点

- 不用再提取特征



前置工作：整理提取可量化的数据特征

常见问题：维数灾难、特征稀疏

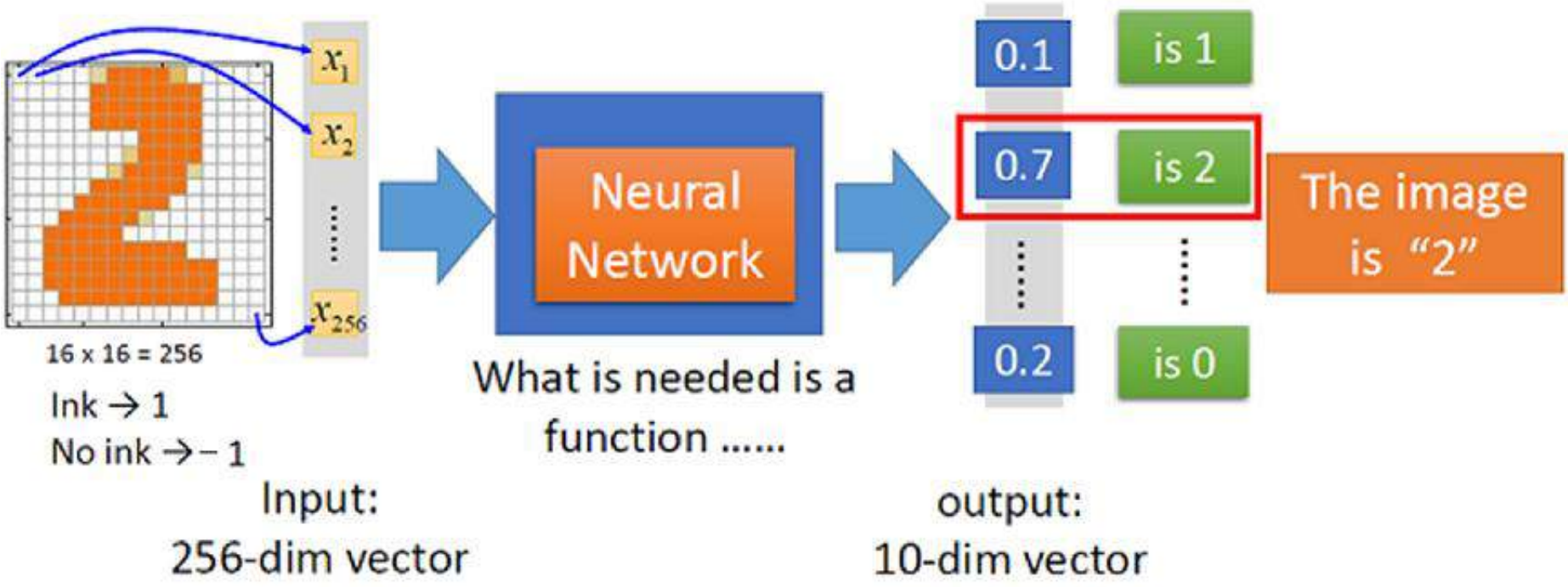
End-to-End：对噪声、不同特征的偏向会自动忽略和调整

所需要的人工技巧更少

Image Source - blog.aylien.com

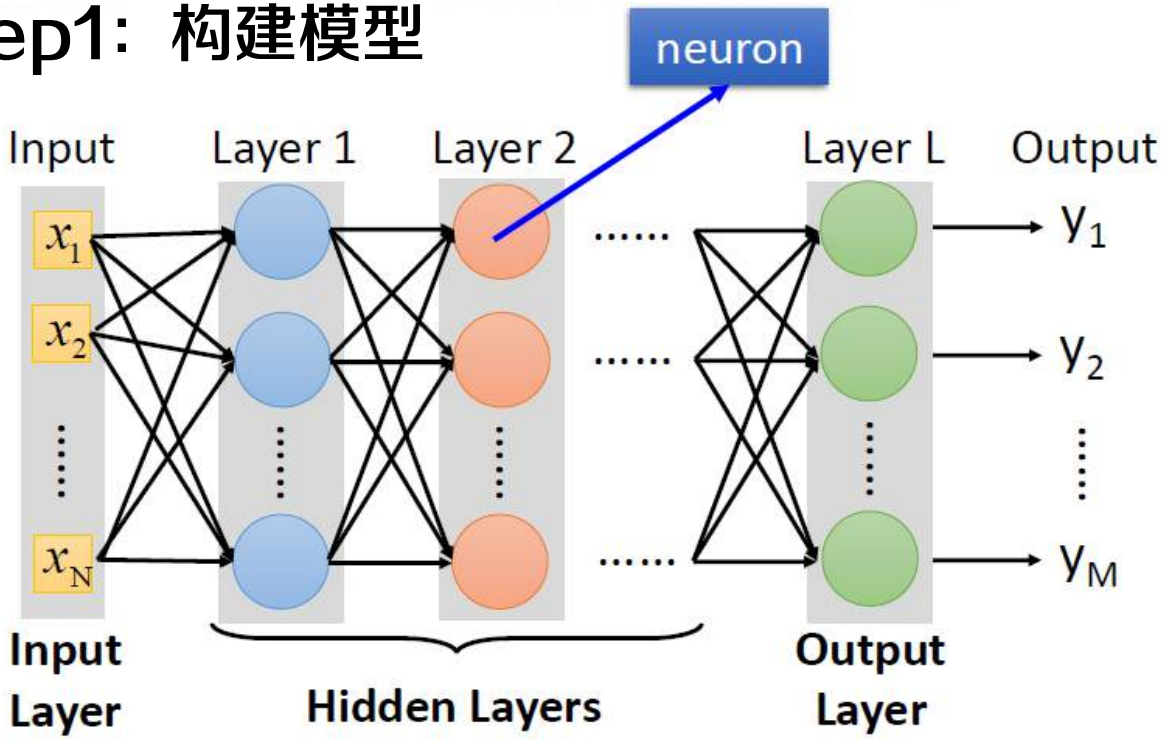
算法原理 一个简单的DNN9-1

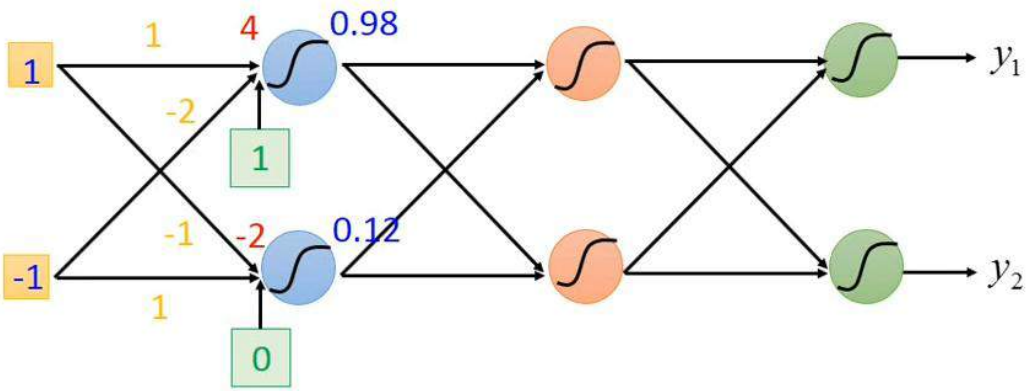
- 手写数字识别



算法原理 一个简单的DNN9-2

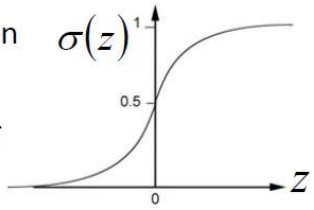
- Step1：构建模型





Sigmoid Function

$$\sigma(z)=\frac{1}{1+e^{-z}}$$

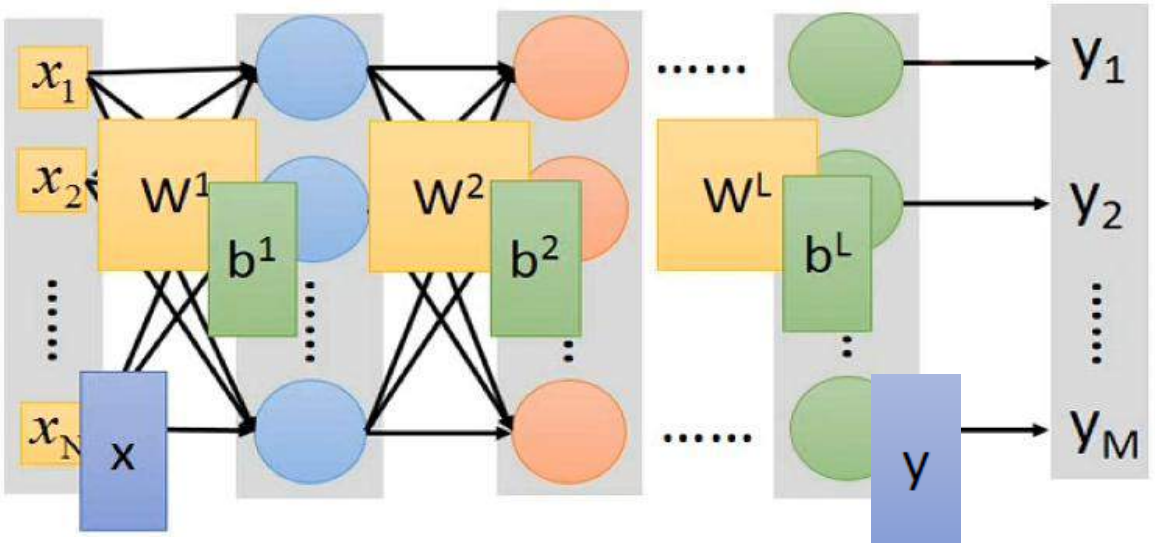


$$\sigma\left(\begin{bmatrix}1 & -2\\ -1 & 1\end{bmatrix}\begin{bmatrix}1\\ -1\end{bmatrix}+\begin{bmatrix}1\\ 0\end{bmatrix}\right)=\begin{bmatrix}0.98\\ 0.12\end{bmatrix}$$

$$\begin{bmatrix}4\\ -2\end{bmatrix}$$

算法原理 一个简单的DNN9-3

- Step1：构建模型



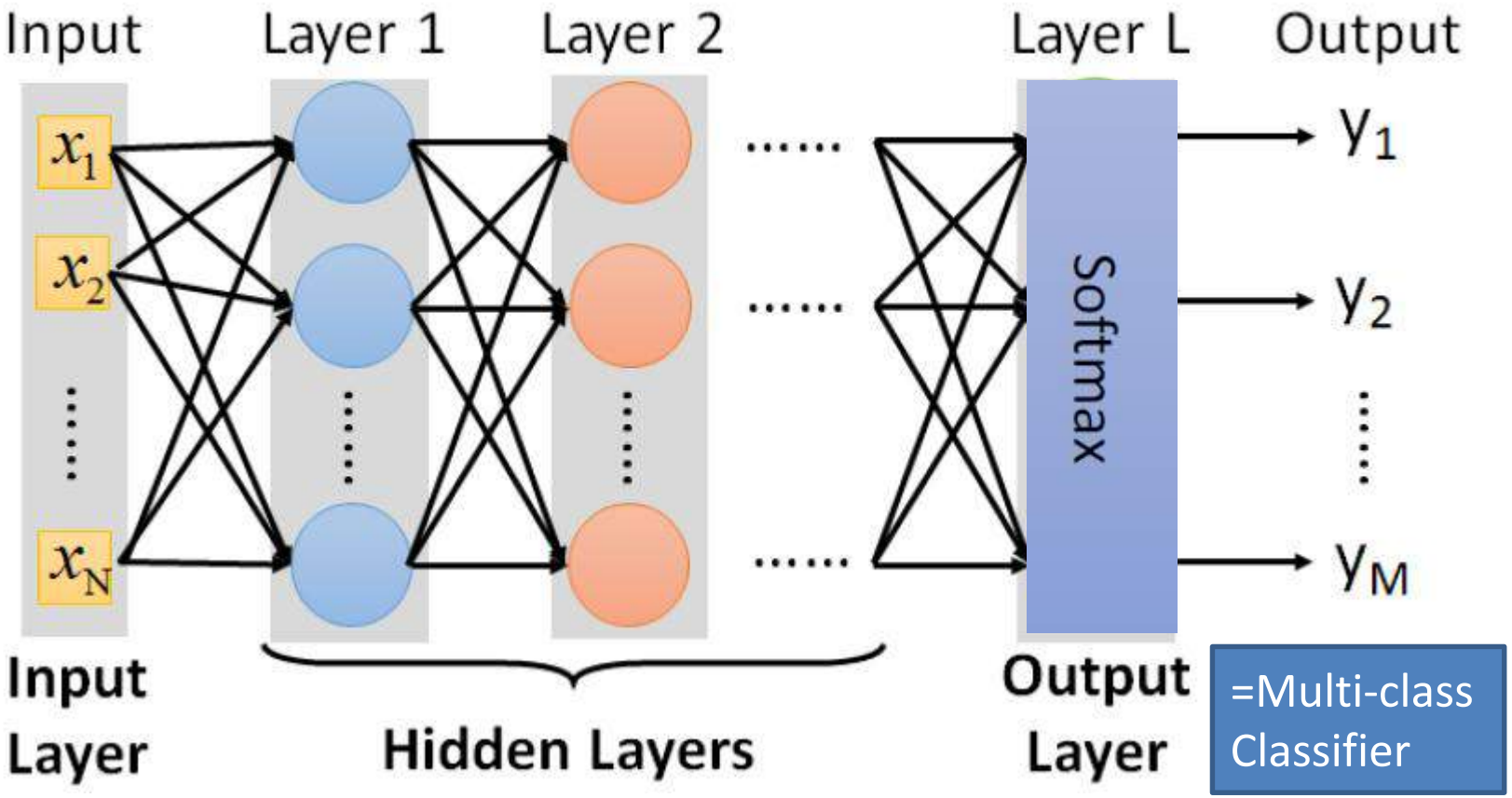
$$\sigma(\ W^1\ x\ +\ b^1\ )$$

多少层？每层多少神经元？
设计其它网络结构？

$$y = f(x) = \sigma(W^L \cdots \sigma(W^2\ \sigma(W^1\ x\ +\ b^1\ )+\ b^2\ ) \cdots +\ b^L\ )$$

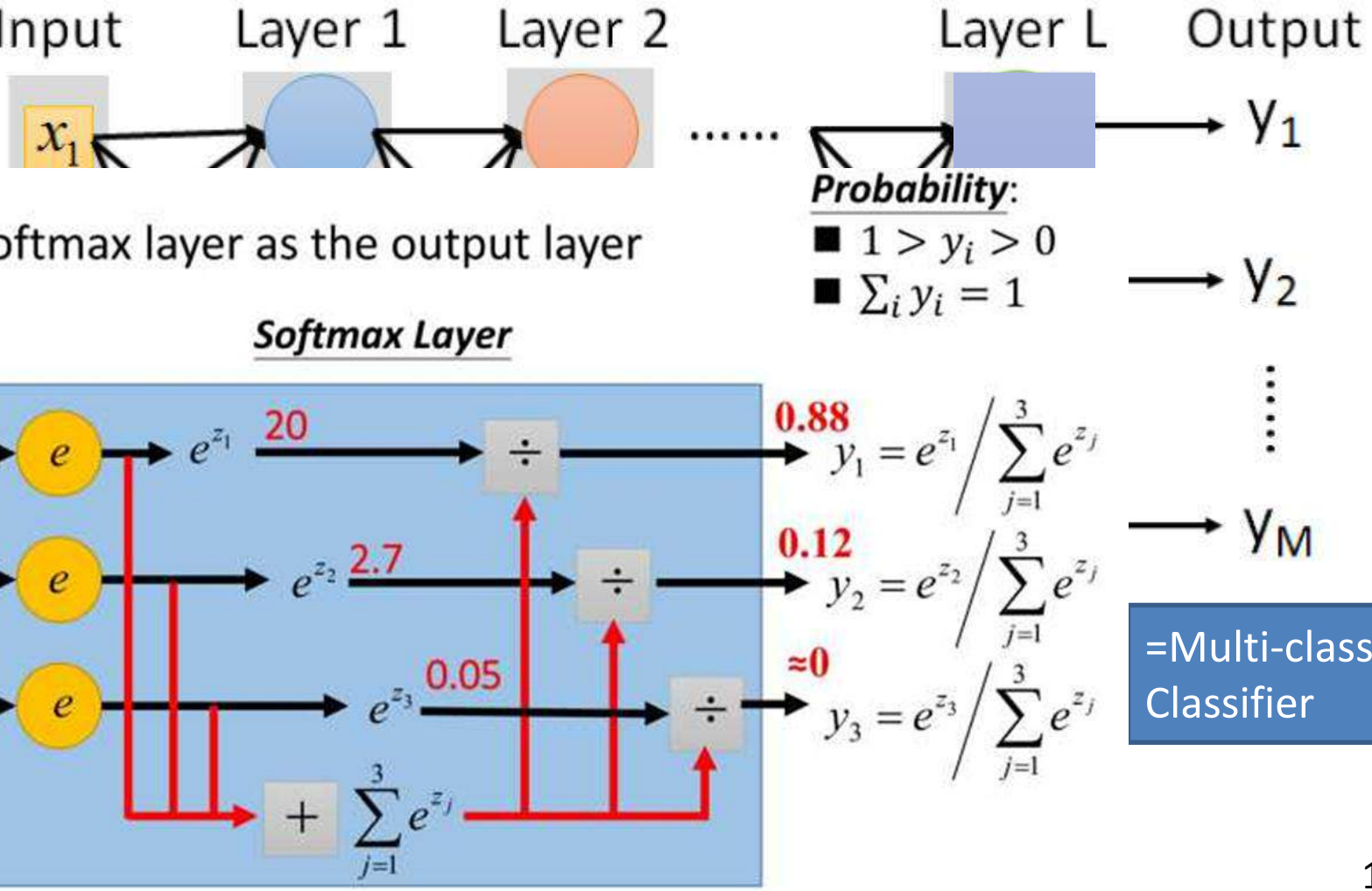
算法原理 一个简单的DNN9-5

- Step1：构建模型



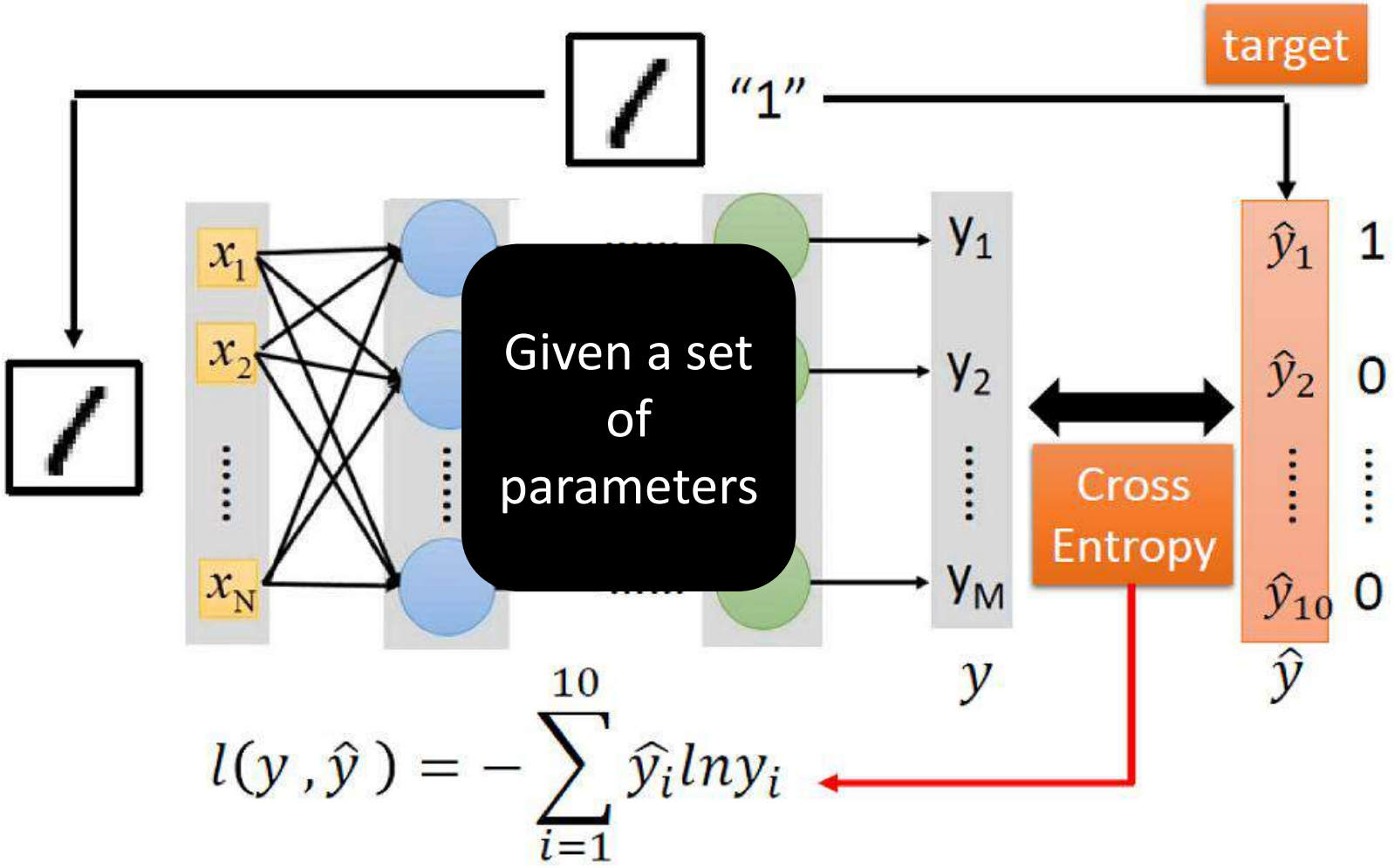
算法原理 一个简单的DNN9-6

- Step1：构建模型



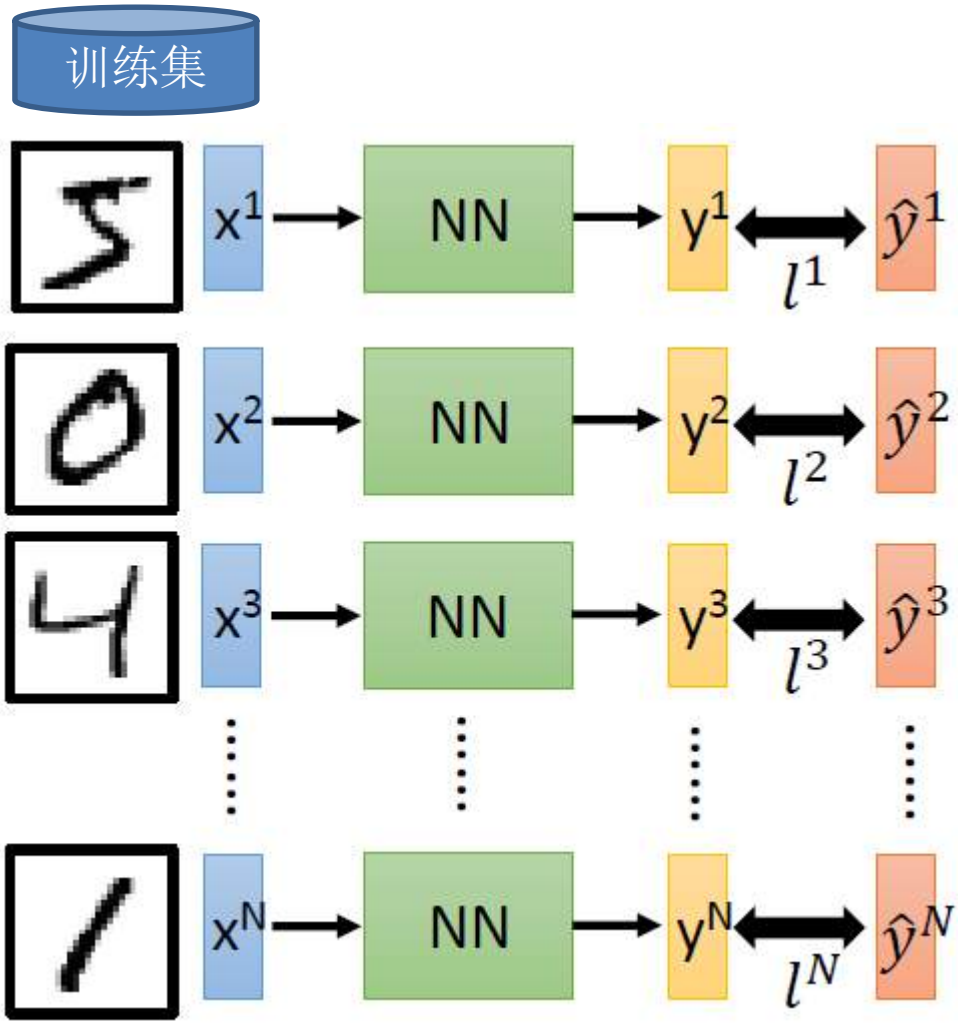
算法原理 一个简单的DNN9-7

- Step2：定义目标函数



算法原理 一个简单的DNN9-8

- Step2：定义目标函数



Total Loss:

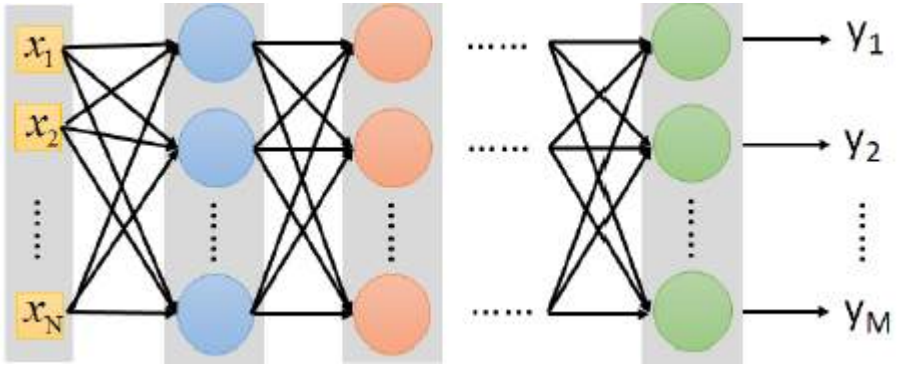
$$L = \sum_{n=1}^{N} l^n$$

找到f*使total loss最小

找到network参数W*使total loss最小

算法原理 一个简单的DNN9-9

- Step3：找到最优函数f*



Gradient Descent

θ

w_1 0.2 → 0.15 Compute $\partial L/\partial w_1$ $-\mu\,\partial L/\partial w_1$

w_2 -0.1 → 0.05 Compute $\partial L/\partial w_2$ $-\mu\,\partial L/\partial w_2$

b_1 0.3 → 0.2 Compute $\partial L/\partial b_1$ $-\mu\,\partial L/\partial b_1$

$$\nabla L = \begin{bmatrix} \frac{\partial L}{\partial w_1} \\ \frac{\partial L}{\partial w_2} \\ \vdots \\ \frac{\partial L}{\partial b_1} \\ \vdots \end{bmatrix}$$

gradient

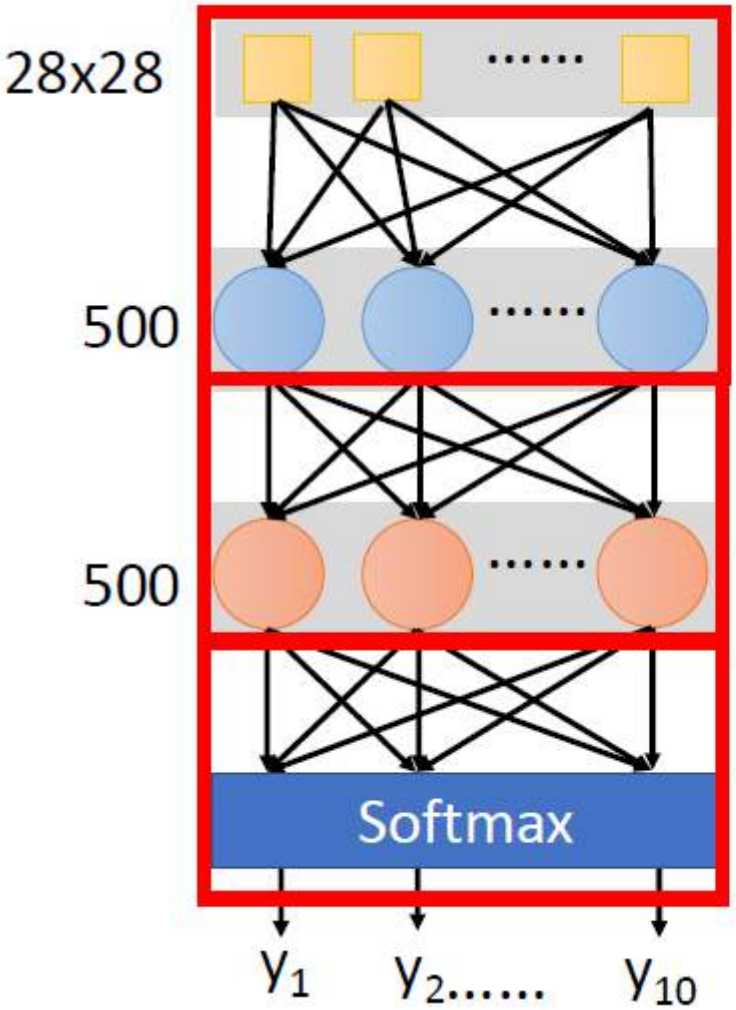
$$L = \sum_{n=1}^{N} l^n$$

$$l(y, \hat{y}) = -\sum_{i=1}^{10} \hat{y}_i \ln y_i$$

Backpropagation

算法原理 基于Keras的实现6-1

- Step1：构建模型




```
model = Sequential()
```

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

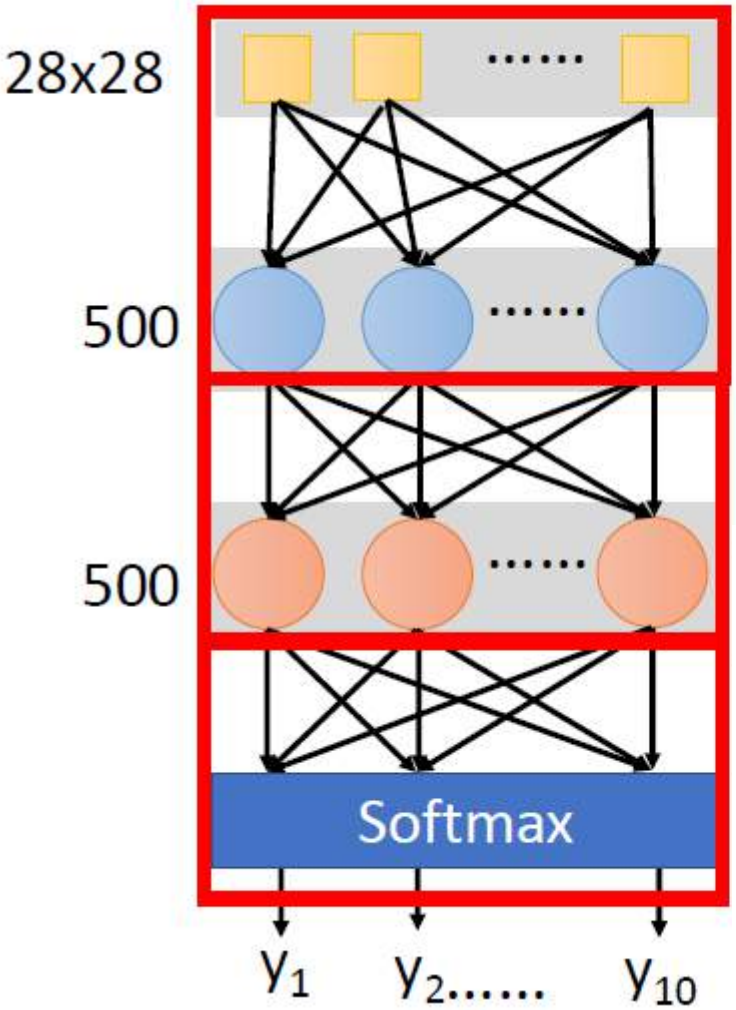
softplus, softsign, relu, tanh,
hard_sigmoid, linear

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

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

算法原理 基于Keras的实现6-2

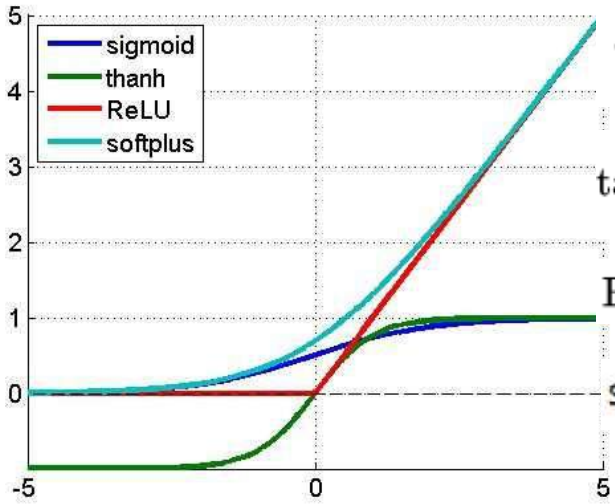
- Step1：构建模型




```
model = Sequential()
```

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

softplus, softsign, relu, tanh,
hard sigmoid, linear



$$\sigma(x) = \frac{1}{1+e^{-x}}$$

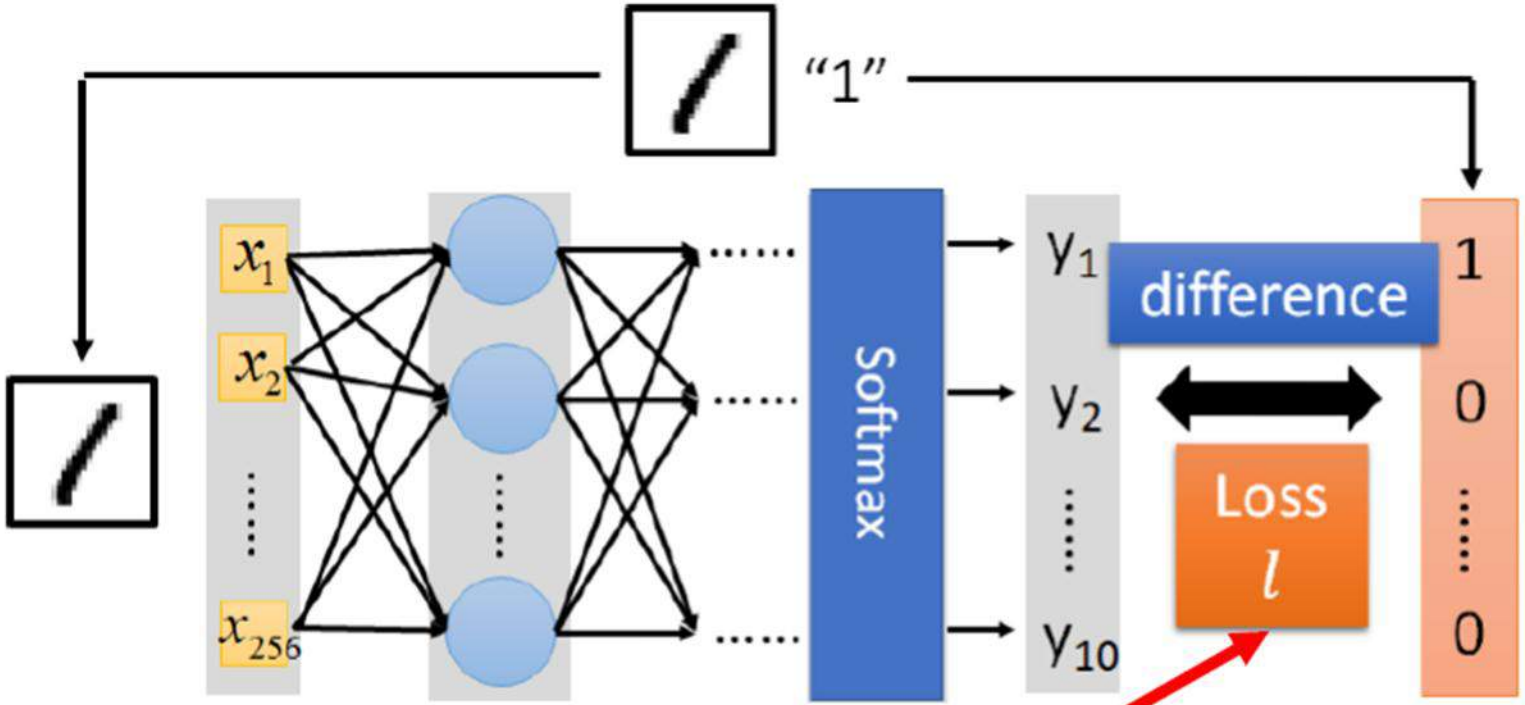
$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$\mathrm{ReLU} = \max(0, x)$$

$$\text{softplus} = log(1+e^x)$$

算法原理 基于Keras的实现6-3

- Step2：定义目标函数




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

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

算法原理 基于Keras的实现6-4

- Step3：找到最优函数f*

Step 3.1: Configuration

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

SGD, RMSprop, Adagrad, Adadelta, Adam, Adamax, Nadam

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)

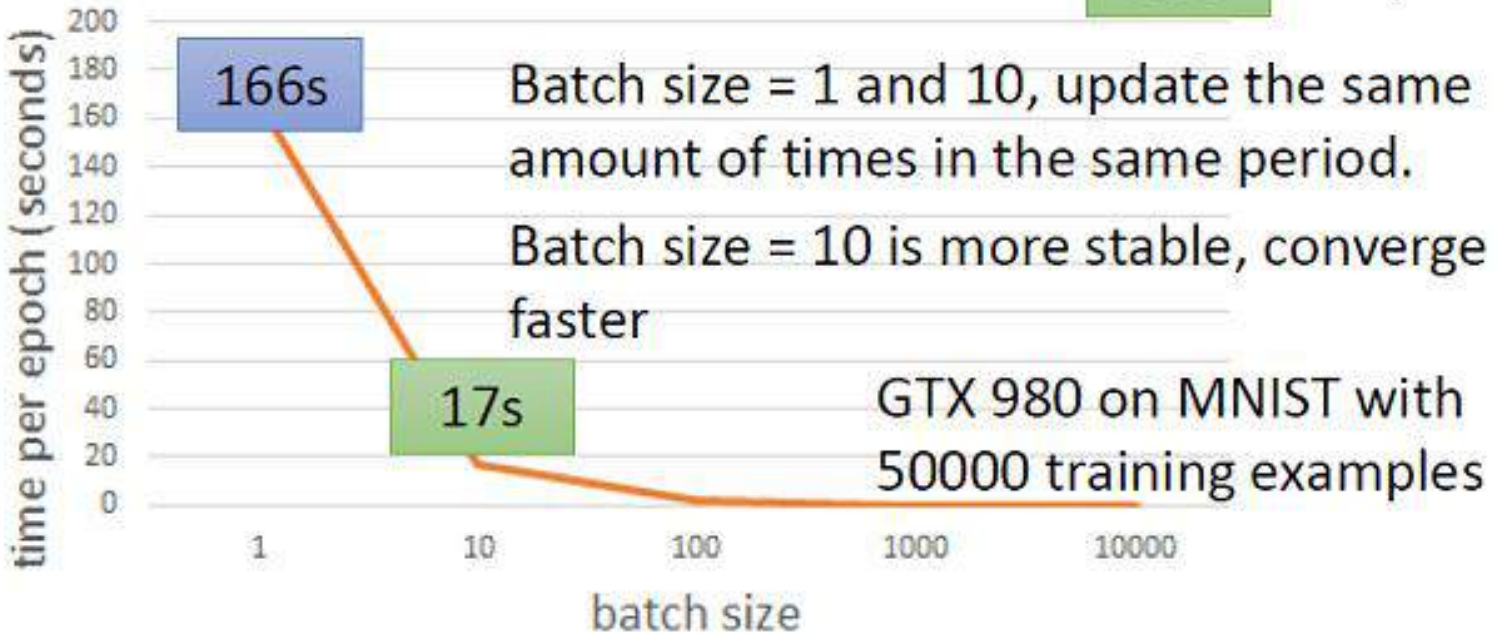
实际没有去最小化整个训练集的total loss，而是把整个训练集随机分为多个batch；计算每个batch（100个样本）的total loss，更新参数，若有60个batch，则更新60次；将所有batch跑完，为1个epoch

算法原理 基于Keras的实现6-5

- Step3：找到最优函数f*

Smaller batch size means more updates in one epoch

- E.g. 50000 examples
- batch size = 1, 50000 updates in one epoch 166s 1 epoch
- batch size = 10. 5000 updates in one epoch 17s 10 epoch



算法原理 基于Keras的实现6-6

- 实验结果
  - 实验数据：http://yann.lecun.com/exdb/mnist/

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 0s 7us/step - loss: 0.4130 - acc: 0.8825
Epoch 2/10
60000/60000 [==============================] - 0s 7us/step - loss: 0.1943 - acc: 0.9440
Epoch 3/10
60000/60000 [==============================] - 0s 7us/step - loss: 0.1513 - acc: 0.9563
Epoch 4/10
60000/60000 [==============================] - 0s 7us/step - loss: 0.1247 - acc: 0.9646
Epoch 5/10
60000/60000 [==============================] - 0s 7us/step - loss: 0.1062 - acc: 0.9691
Epoch 6/10
60000/60000 [==============================] - 0s 7us/step - loss: 0.0924 - acc: 0.9731
Epoch 7/10
60000/60000 [==============================]                                  acc: 0.9762
Epoch 8/10
60000/60000 [==============================]                                  acc: 0.9784
Epoch 9/10
60000/60000 [==============================]                                  acc: 0.9809
Epoch 10/10
60000/60000 [==============================] - 0s 7us/step - loss: 0.0597      acc: 0.9826
('total loss on testing set:', 0.08160823175990954)
('accuracy of testing set:', 0.9745000000000003)
```

batch_size=100, epoch=10
每个epoch，更新参数600次
共更新参数10*600=6000次

参考文献

1. 高扬.白话深度学习与TensorFlow[M]. 机械工业出版社, 2017.
2. 李宏毅.一天搞懂深度学习[EB/OL]. https://www.zhihu.com/question/26006703, 2017-12-03.

知人者智，自知者明。胜人者有力，自胜者强。知足者富。强行者有志。不失其所者久。死而不亡者，寿。

谢谢！