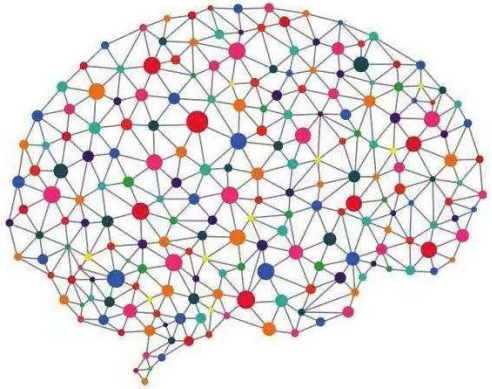
神经网络&深度学习
TENSORFLOW
Google

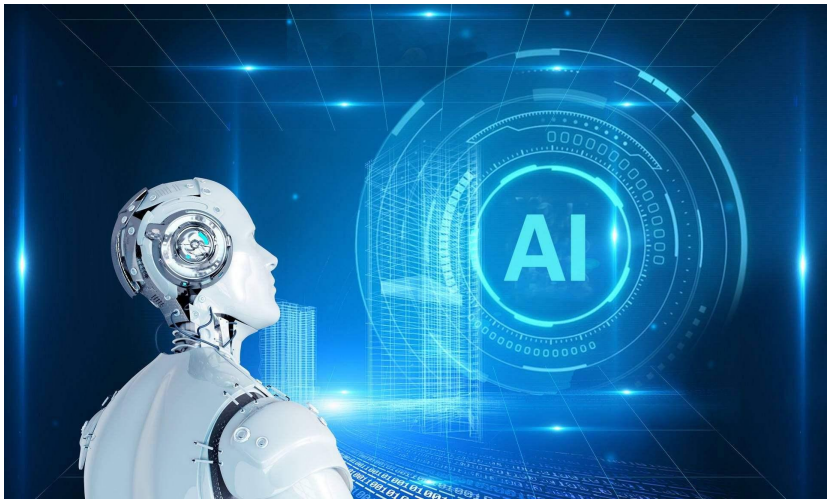


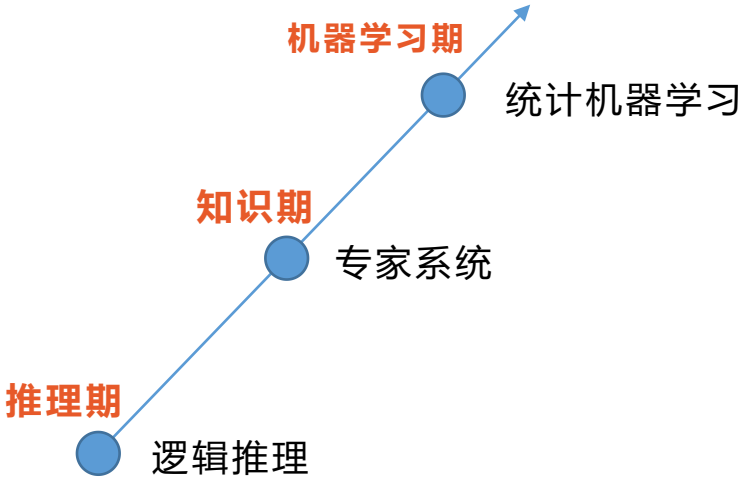
12 人工神经网络

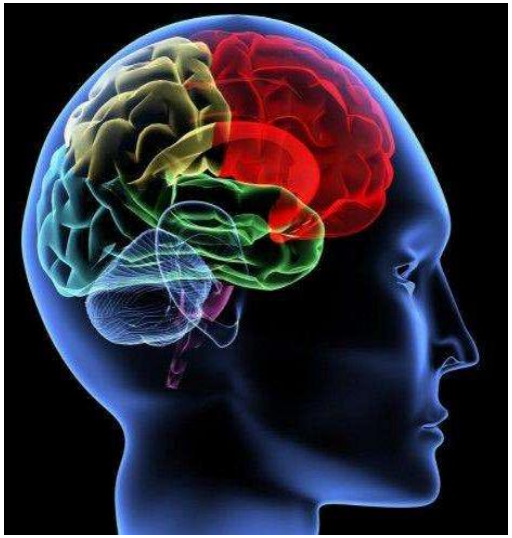
西安科技大学 牟琦
muqi@xust.edu.cn


tensorflow / tensorflow
TensorFlow: Open sou
Clone this repo:
git clone https://tensor
Branches
master

12.1 神经元与感知机

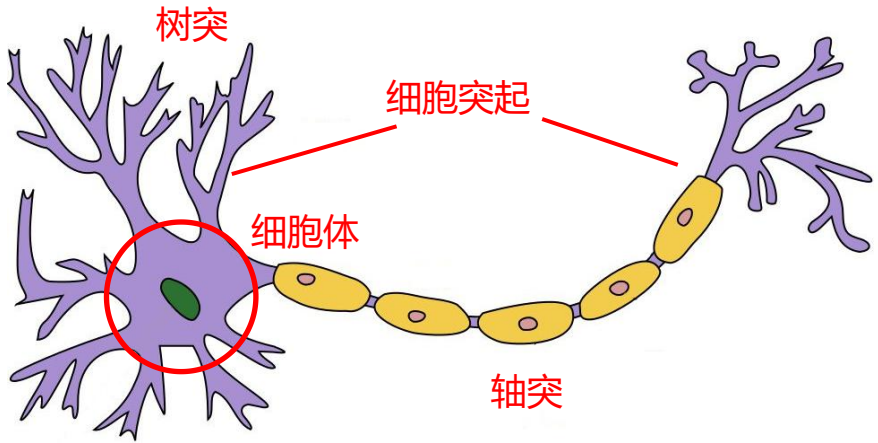

AI

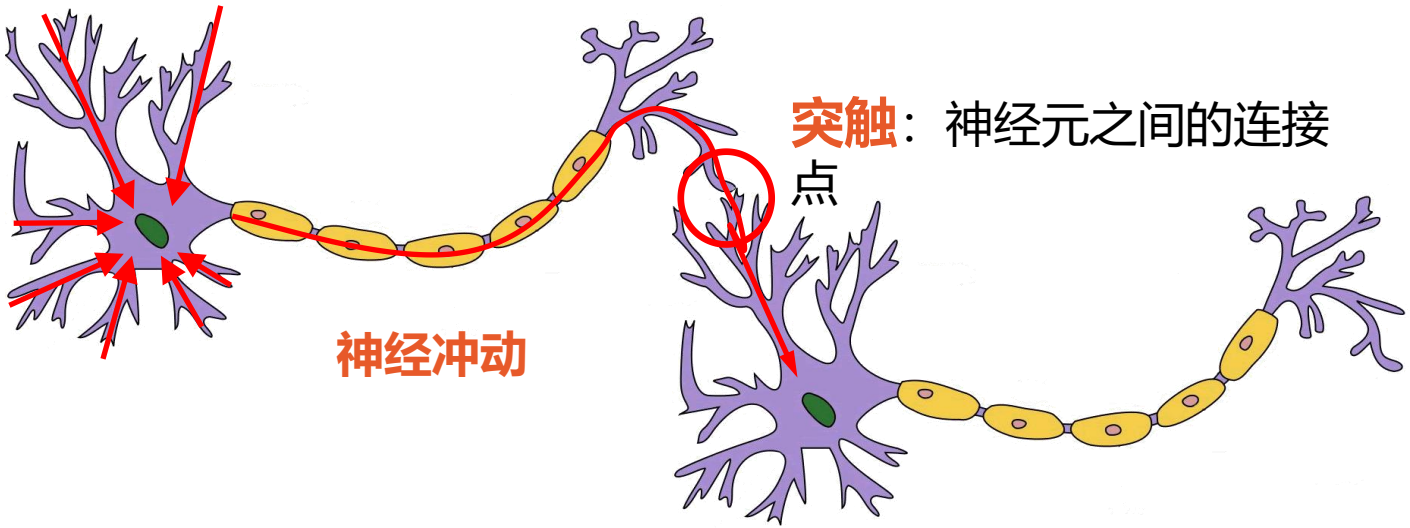

机器学习期
统计机器学习
知识期
专家系统
推理期
逻辑推理



神经网络：人脑智慧的物质基础

神经元/神经细胞：生物神经系统的基本单元

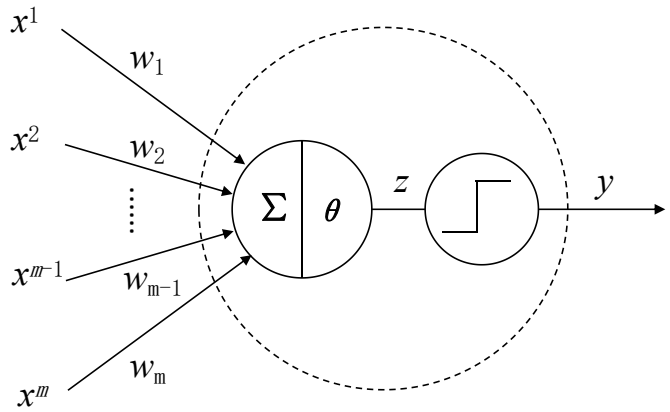




突触：神经元之间的连接点

抑制状态 → 兴奋状态

M-P神经元：1943，McCulloch，Pitts



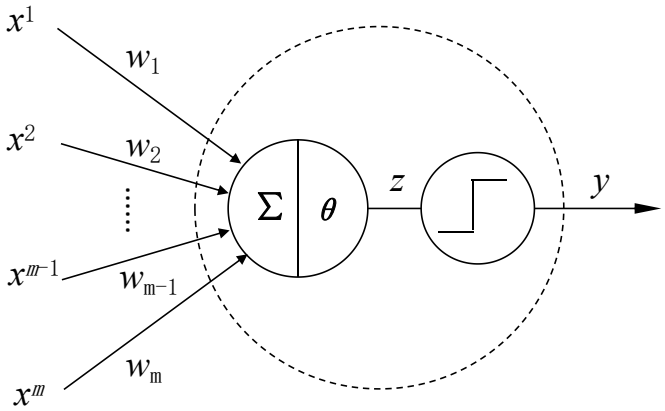
$$z = \sum_{j=1}^{m} w_j x^j - \theta$$

激活函数/激励函数 (activation function)

$$y = step(z)$$

$$step(z) = \begin{cases} 1 & , \quad z \geq 0 \\ 0 & , \quad z < 0 \end{cases}$$

M-P神经元：1943，McCulloch，Pitts



$$y = step(\sum_{j=1}^{m} w_j x^j - \theta) \qquad (j = 1,2,...,m)$$

令 $w_0 = -\theta$，$x^0 = 1$

$$y = step(w_0 x^0 + w_1 x^1 + w_2 x^2 + ... + w_m x^m)$$

$$y = step(W^T X)$$

$$W = (w_0, w_1, ..., w_m)^T$$

$$X = (x_0, x_1, x_2, ..., x_m)^T$$

权值向量W无法自动学习和更新，**不具备学习的能力**

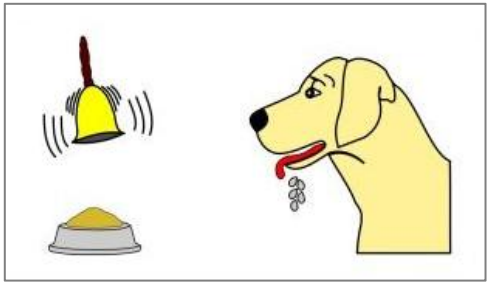
Donald Hebb，1949，神经心理学

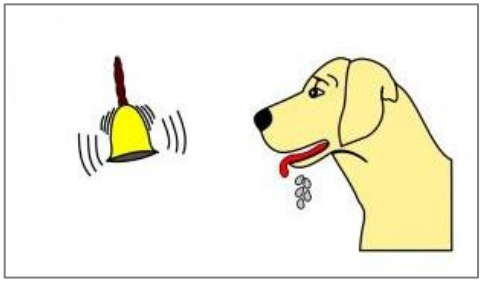
在**同一时间被激发**的神经元间的**联系会被强化**

如果两个神经元总是不能同步激发，它们之间的联系将会越来越弱，甚至消失

神经网络的**学习**过程是发生在神经元之间的**突触**部位

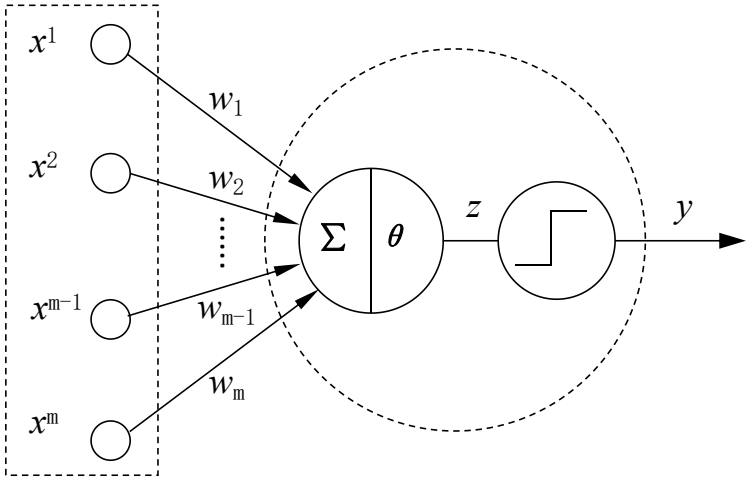
突触的**联结强度**与突触连接的两个神经元的**活性之和**成正比

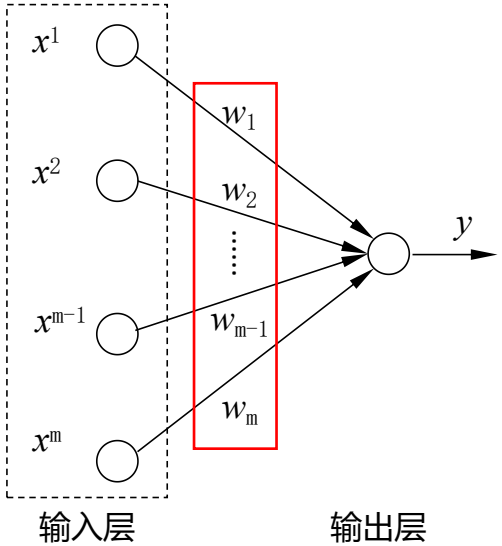




人工神经网络：通过**算法**调整神经元中的**权值**，模拟人类神经网络的学习能力

感知机：1957，Frank Rosenblatt





感知机训练法则 (Perceptron Training Rule)

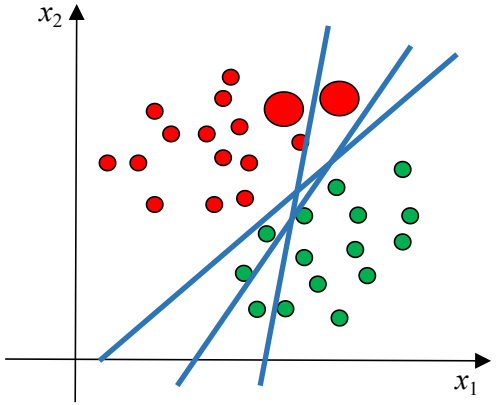
$$w_i^{(k+1)} = w_i^{(k)} + \Delta w_i$$

$$\Delta w_i = \eta (y - \hat{y}) x_i$$

y：训练样例的标记

$\hat{y}$：感知机的输出

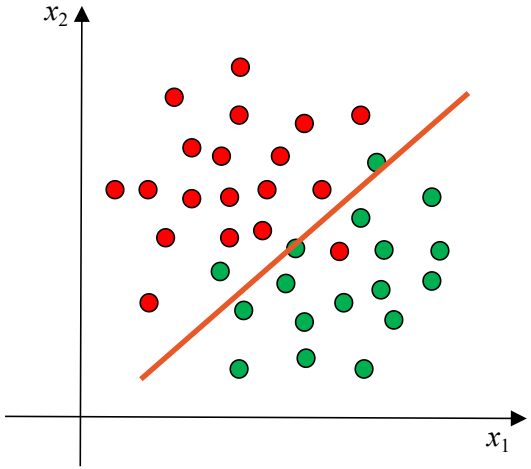
$\eta \in (0,1)$：学习率



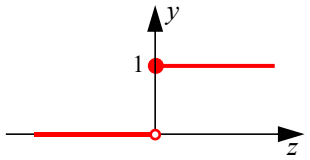
$$y = step(W^T X)$$

- 自动调整权值，具备**学习能力**
- 第一个用算法来精确定义的神经网络模型
- **线性二分类**的分类器
- 感知机算法存在**多个解**，受到**权值向量初始值**，**错误样本顺序**的影响
- 对于**非线性可分**的数据集，感知机训练法则**无法收敛**

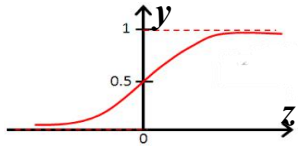
Delta法则：使用**梯度下降法**，找到能够**最佳拟合**训练样本集的的权向量



$$y = step\,(W^T X)$$

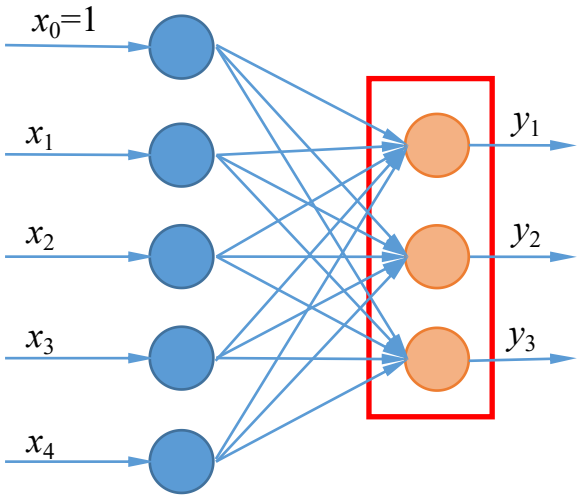


$$y = sigmoid\,(W^T X)$$



逻辑回归可以看做**单层神经网络**

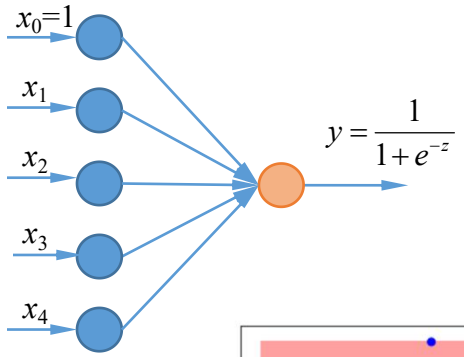
多分类问题：设置多个输出节点

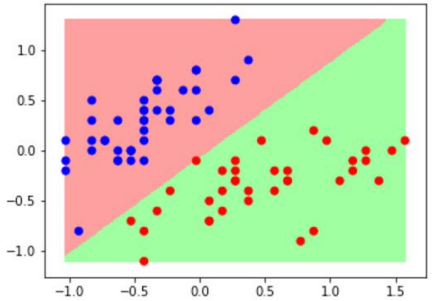


$$y = soft\max(W^T X)$$

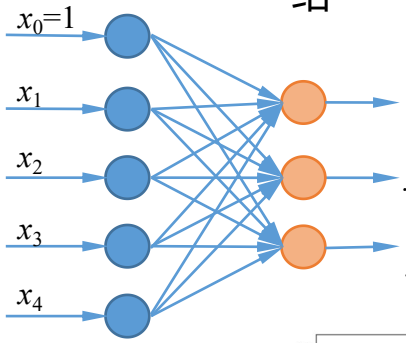
12.2 实例：实现单层神经网络

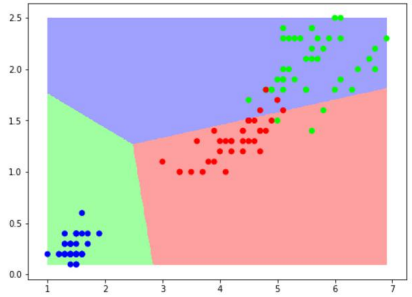
逻辑回归 —— 感知机





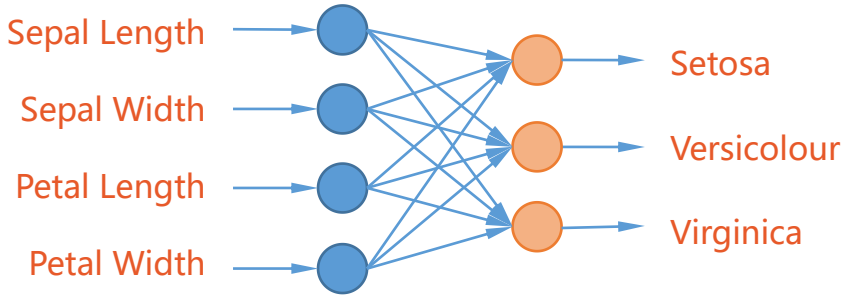
sofmax回归 —— 单层神经网络





口 神经网络的设计

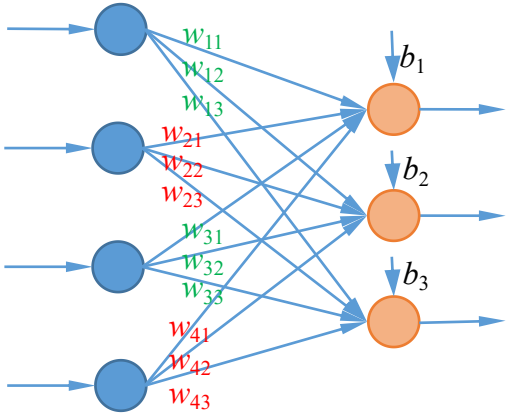
神经网络的结构	单层前馈型神经网络
激活函数	softmax函数
损失函数	交叉熵损失函数



独热编码

神经网络的实现

1	6.4	2.8	5.6	2.2
2	5	2.3	3.3	1
3	4.9	2.5	4.5	1.7
4	4.9	3.1	1.5	0.1
5	5.7	3.8	1.7	0.3
6	4.4	3.2	1.3	0.2
7	5.4	3.4	1.5	0.4
119	4.8	3	1.4	0.1
120	5.5	2.4	3.7	1



1	0	0	1	[3.04389629e-04, 1.29587591e-01, 8.70108008e-01]
2	0	1	0	[1.75134137e-01, 6.48399115e-01, 1.76466733e-01]
3	0	0	1	[7.53107527e-03, 3.44870061e-01, 6.47598863e-01]
4	1	0	0	[9.13596153e-01, 8.42635259e-02, 2.14024656e-03]
5	1	0	0	[8.66889775e-01, 1.27143607e-01, 5.96662890e-03]
6	1	0	0	[9.33885932e-01, 6.36122525e-02, 2.50180368e-03]
7	1	0	0	[8.95907521e-01, 9.70410407e-02, 7.05144275e-03]
119	1	0	0	[9.26600277e-01, 7.15029836e-02, 1.89674716e-03]
120	0	1	0	[9.72348675e-02, 7.34697282e-01, 1.68067887e-01]

(120, 4)

$$W=\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{31} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix}$$

(4, 3)

$B = [\, b_1, b_2, b_3 \,]$

(120, 3)

$Y=XW+B$

softmax函数

$Y=XW+B$

tf. nn. softmax()

tf.nn.softmax(tf.matmul(X_train,W)+b)

独热编码

tf. one_hot(indices, depth)

tf.one_hot(tf.constant(y_test, dtype=tf.int32), 3)

交叉熵损失函数

表示为独热编码的标签值

softmax函数的输出

tf.keras.losses.categorical_crossentropy(y_true, y_pred)

tf.reduce_mean(tf.keras.losses.categorical_crossentropy(y_true=Y_train,y_pred=PRED_train))

- 导入库

In [1]:
```
import tensorflow as tf
print("TensorFlow version:", tf.__version__)
```

```
TensorFlow version: 2.0.0
```

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [3]:
```
gpus = tf.config.experimental.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(gpus[0], True)
```

```
InternalError: Blas GEMM launch failed :
```

for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)

加载数据

```
In [4]: TRAIN_URL = "http://download.tensorflow.org/data/iris_training.csv"
        train_path = tf.keras.utils.get_file(TRAIN_URL.split('/')[-1], TRAIN_URL)

        TEST_URL = "http://download.tensorflow.org/data/iris_test.csv"
        test_path = tf.keras.utils.get_file(TEST_URL.split('/')[-1], TEST_URL)

In [5]: df_iris_train = pd.read_csv(train_path, header=0)
        df_iris_test = pd.read_csv(test_path, header=0)

In [6]: iris_train=np.array(df_iris_train)
        iris_test=np.array(df_iris_test)

In [7]: iris_train.shape,iris_test.shape

Out[7]: ((120, 5), (30, 5))
```

- 数据预处理

```
In [8]: x_train=iris_train[:,0:4]
        y_train=iris_train[:,4]

        x_test=iris_test[:,0:4]
        y_test=iris_test[:,4]

In [9]: x_train.shape,y_train.shape
Out[9]: ((120, 4), (120,))

In [10]: x_test.shape,y_test.shape
Out[10]: ((30, 4), (30,))

In [11]: x_train=x_train-np.mean(x_train,axis=0)
         x_test=x_test-np.mean(x_test,axis=0)
```

```
In [12]: x_train.dtype, y_train.dtype

Out[12]: (dtype('float64'), dtype('float64'))

In [13]: X_train=tf.cast(x_train, tf.float32)
         Y_train=tf.one_hot(tf.constant(y_train, dtype=tf.int32), 3)

         X_test=tf.cast(x_test, tf.float32)
         Y_test=tf.one_hot(tf.constant(y_test, dtype=tf.int32), 3)

In [14]: X_train.shape, Y_train.shape

Out[14]: (TensorShape([120,  4]), TensorShape([120,  3]))

In [15]: X_test.shape, Y_test.shape

Out[15]: (TensorShape([30,  4]), TensorShape([30,  3]))
```

- 设置超参数和显示间隔

In [16]:
```
learn_rate = 0.5
iter = 50

display_step =10
```

- 设置模型参数初始值

In [17]:
```
np.random.seed(612)
W = tf.Variable(np.random.randn(4,3),dtype=tf.float32)
B = tf.Variable(np.zeros([3]),dtype=tf.float32)
```

- 训练模型

```
In [18]: acc_train=[]
         acc_test=[]
         cce_train=[]
         cce_test=[]
```

```
for i in range(0,iter+1):
    with tf.GradientTape() as tape:
        PRED_train=tf.nn.softmax(tf.matmul(X_train,W)+B)
        Loss_train=tf.reduce_mean(tf.keras.losses.categorical_crossentropy(y_true=Y_train,y_pred=PRED_train))

    PRED_test=tf.nn.softmax(tf.matmul(X_test,W)+B)
    Loss_test=tf.reduce_mean(tf.keras.losses.categorical_crossentropy(y_true=Y_test,y_pred=PRED_test))

    accuracy_train=tf.reduce_mean(tf.cast(tf.equal(tf.argmax(PRED_train.numpy(),axis=1),y_train),tf.float32))
    accuracy_test=tf.reduce_mean(tf.cast(tf.equal(tf.argmax(PRED_test.numpy(),axis=1),y_test),tf.float32))

    acc_train.append(accuracy_train)
    acc_test.append(accuracy_test)
    cce_train.append(Loss_train)
    cce_test.append(Loss_test)

    grads = tape.gradient(Loss_train,[W,B])
    W.assign_sub(learn_rate*grads[0])
    B.assign_sub(learn_rate*grads[1])

    if i % display_step == 0:
        print("i: %i, TrainAcc:%f, TrainLoss: %f ,TestAcc:%f, TestLoss: %f" % (i,accuracy_train,Loss_train,accuracy_test,Loss_test))
```

dL_dw (4,3)

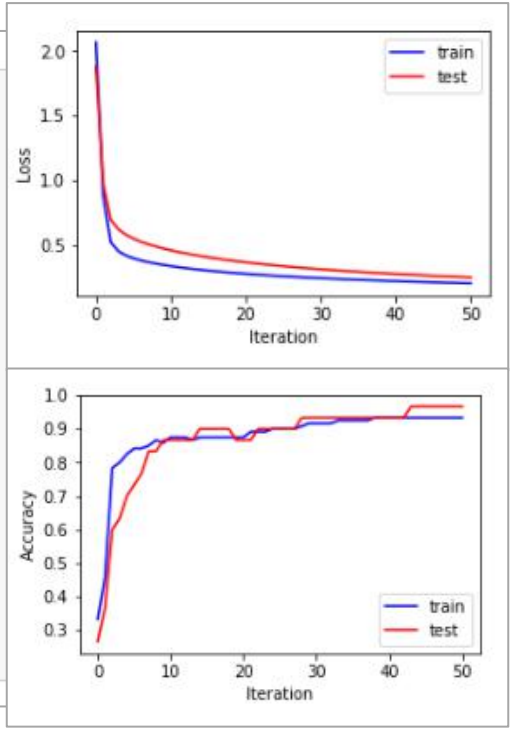
dL_db (3,)

训练结果

```
i: 0, TrainAcc:0.333333, TrainLoss: 2.066978 ,TestAcc:0.266667, TestLoss: 1.880856
i: 10, TrainAcc:0.875000, TrainLoss: 0.339410 ,TestAcc:0.866667, TestLoss: 0.461705
i: 20, TrainAcc:0.875000, TrainLoss: 0.279647 ,TestAcc:0.866667, TestLoss: 0.368414
i: 30, TrainAcc:0.916667, TrainLoss: 0.245924 ,TestAcc:0.933333, TestLoss: 0.314814
i: 40, TrainAcc:0.933333, TrainLoss: 0.222922 ,TestAcc:0.933333, TestLoss: 0.278643
i: 50, TrainAcc:0.933333, TrainLoss: 0.205636 ,TestAcc:0.966667, TestLoss: 0.251937
```

结果可视化

```
In [20]: plt.figure(figsize=(10,3))

         plt.subplot(121)
         plt.plot(cce_train, color="blue", label="train")
         plt.plot(cce_test, color="red", label="test")
         plt.xlabel("Iteration")
         plt.ylabel("Loss")
         plt.legend()

         plt.subplot(122)
         plt.plot(acc_train, color="blue", label="train")
         plt.plot(acc_test, color="red", label="test")
         plt.xlabel("Iteration")
         plt.ylabel("Accuracy")
         plt.legend()

         plt.show()
```

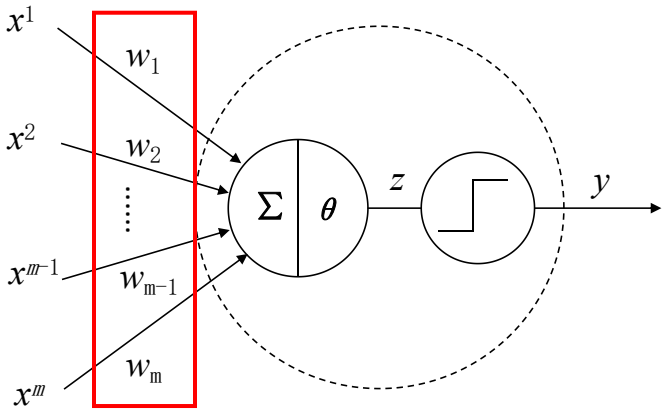


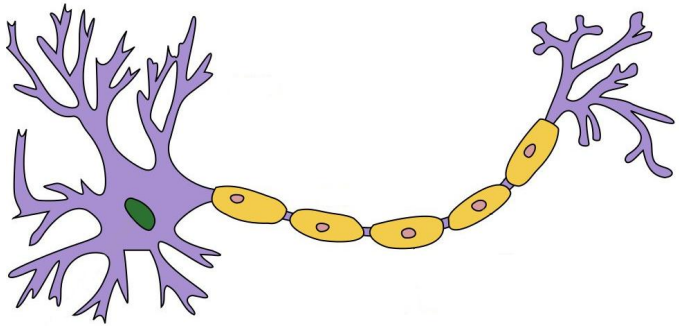

tensorflow / tensorflow
TensorFlow: Open sou
Clone this repo:
git clone https://tensor
Branches
master

12.3 多层神经网络

12.3.1 多层神经网络模型

M-P神经元





权值向量W是固定的，**不能够自动学习和更新**

感知机训练法则

$$w_i^{(k+1)} = w_i^{(k)} + \Delta w_i$$

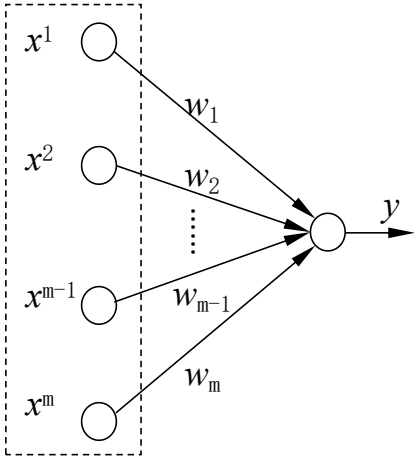
$$\Delta w_i = \eta (y - \hat{y}) x_i$$

y：训练样例的标记

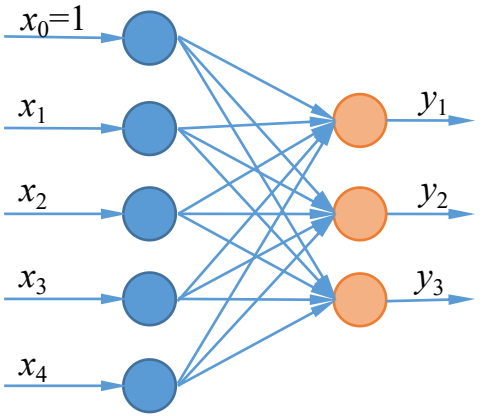
$\hat{y}$：感知机的输出

$\eta \in (0,1)$：学习率

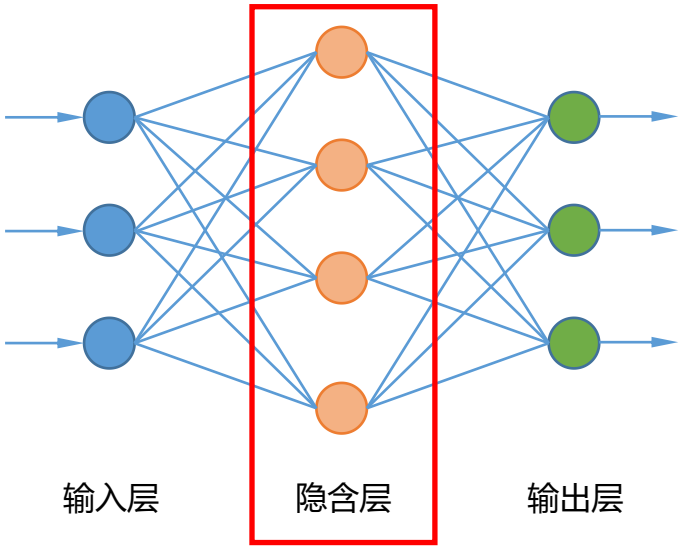
感知机：线性二分类



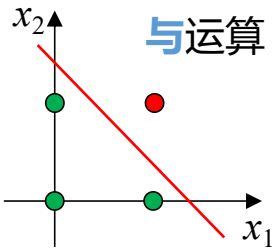
多分类：设置多个输出节点

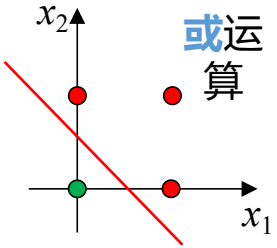


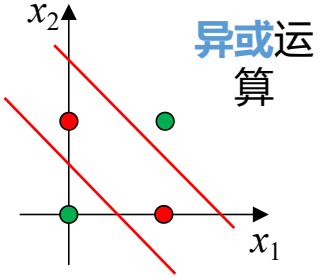
多层神经网络

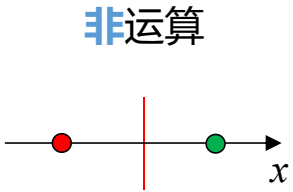


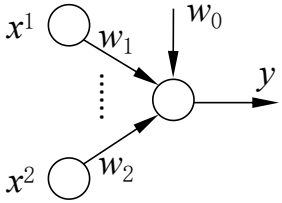
- 线性分类器



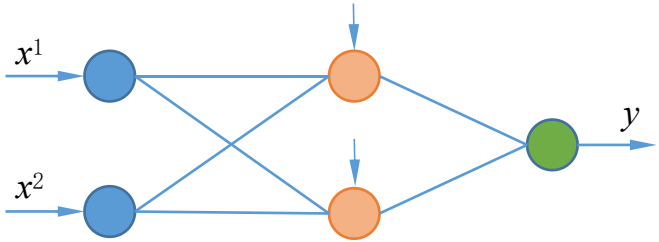


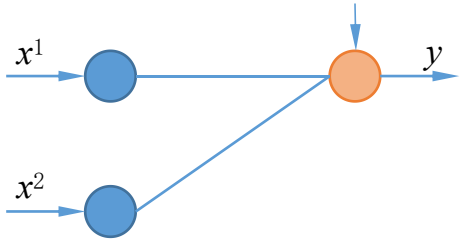


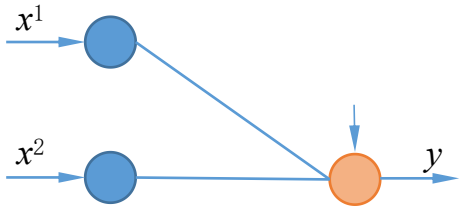


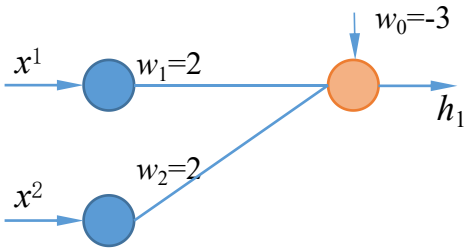


异或问题





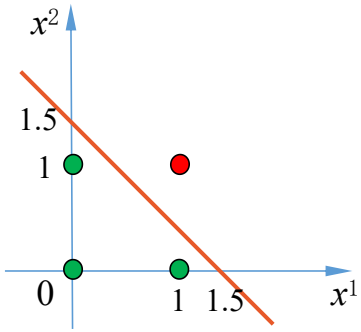


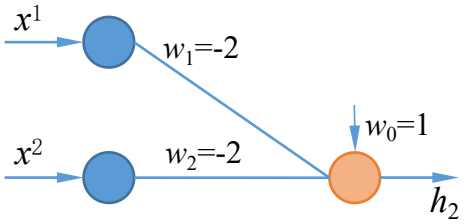


$z_1=2x^1+2x^2-3$

$h_1=step(2x^1+2x^2-3) \quad =x^1x^2$

(x^1, x^2)	z_1	h_1
(0, 0)	-3	0
(0, 1)	-1	0
(1, 0)	-1	0
(1, 1)	1	1

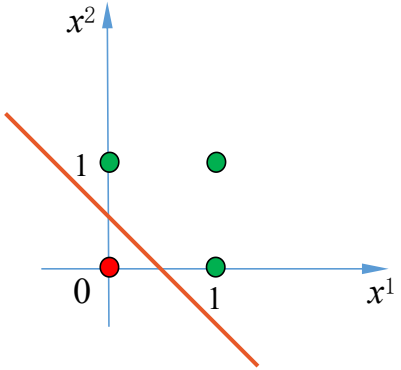




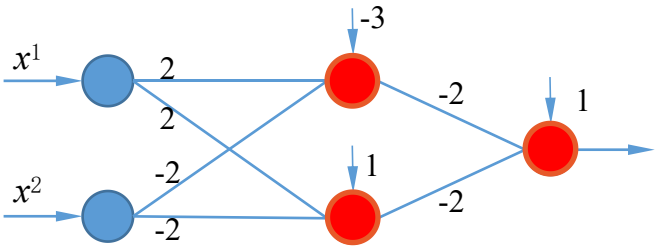
$$z_2 = -2x^1 - 2x^2 + 1$$

$$h_2 = step(-2x^1 - 2x^2 + 1) = \overline{x^1 + x^2}$$

(x^1, x^2)	z_2	h_2
(0, 0)	1	1
(0, 1)	-1	0
(1, 0)	-1	0
(1, 1)	-3	0



12.3.1 多层神经网络模型



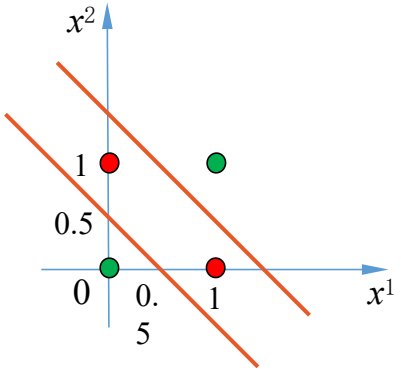
每个神经元3个模型参数：w_1, w_2, w_0

3个神经元共9个模型参数

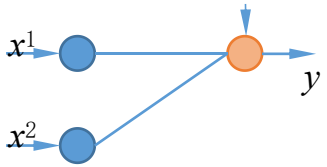
$z = -2h_1 - 2h_2 + 1$

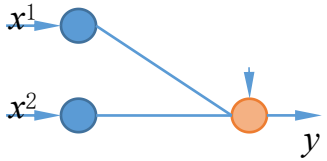
$y = step(-2x^1 - 2x^2 + 1)$

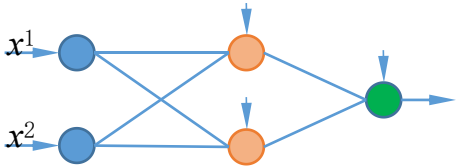
(x^1, x^2)	h_1	h_2	y
(0, 0)	0	1	0
(0, 1)	0	0	1
(1, 0)	0	0	1
(1, 1)	1	0	0

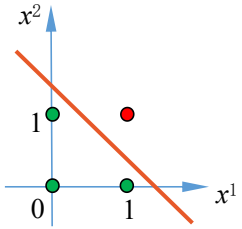


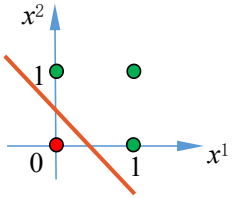
12.3.1 多层神经网络模型











(x^1, x^2)	h_1	h_2	y
(0, 0)	0	1	0
(0, 1)	0	0	1
(1, 0)	0	0	1
(1, 1)	1	0	0

$$h_1 = x^1 x^2$$

$$h_2 = \overline{x^1 + x^2}$$

$$\overline{h_1 + h_2} = \overline{x^1 x^2 + \overline{x^1 + x^2}}$$

$$= \overline{x^1 x^2}(x^1 + x^2)$$

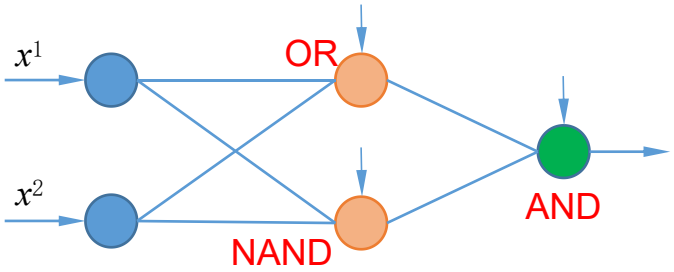
$$= (\overline{x^1} + \overline{x^2})(x^1 + x^2)$$

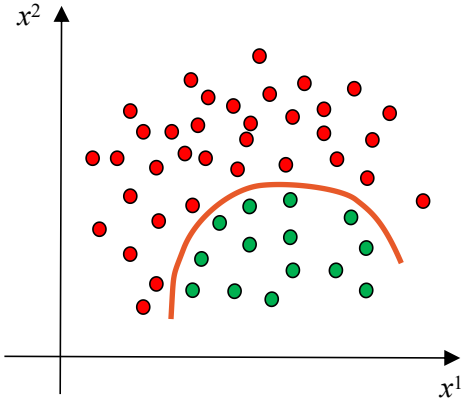
$$= (\overline{x^1} + \overline{x^2})x^1 + (\overline{x^1} + \overline{x^2})x^2$$

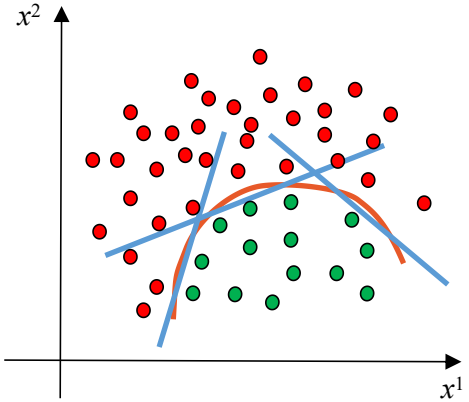
$$= x^1 \overline{x^2} + \overline{x^1} x^2$$

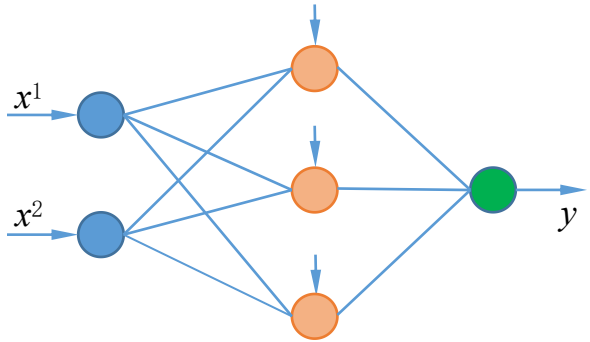
$$= x^1 \oplus x^2$$

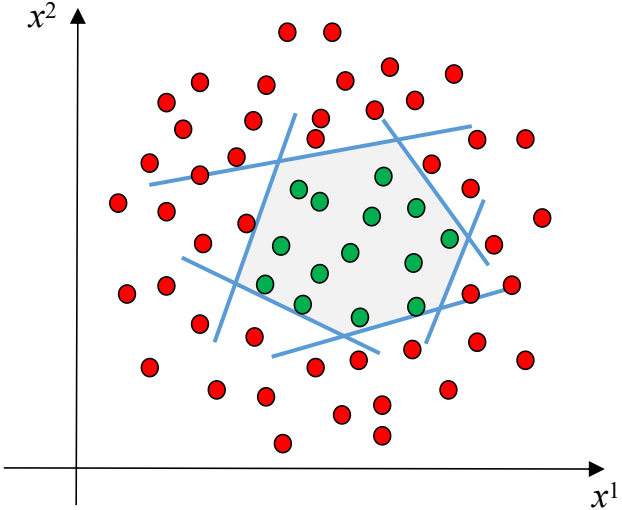
(x^1, x^2)	OR	NAND	AND
(0, 0)	0	1	0
(0, 1)	1	1	1
(1, 0)	1	1	1
(1, 1)	1	0	0

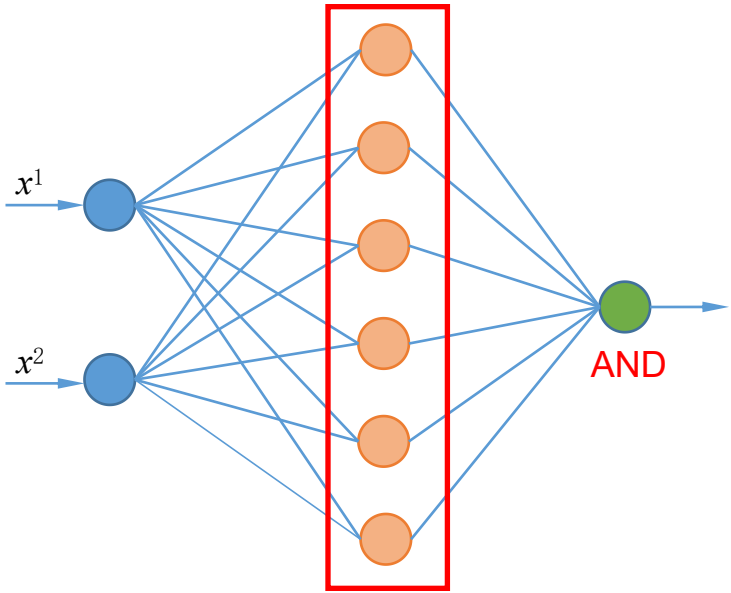


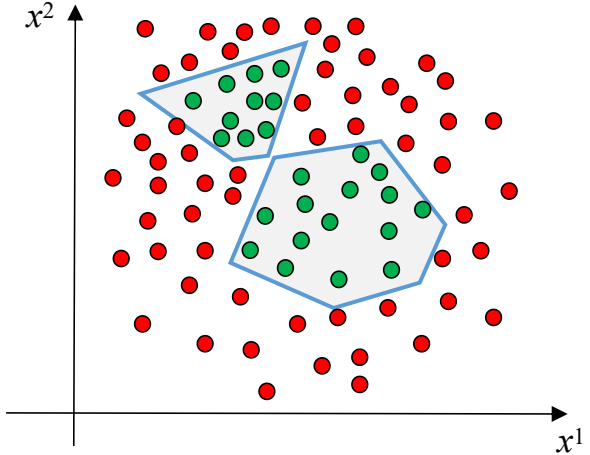

x^2
x^1

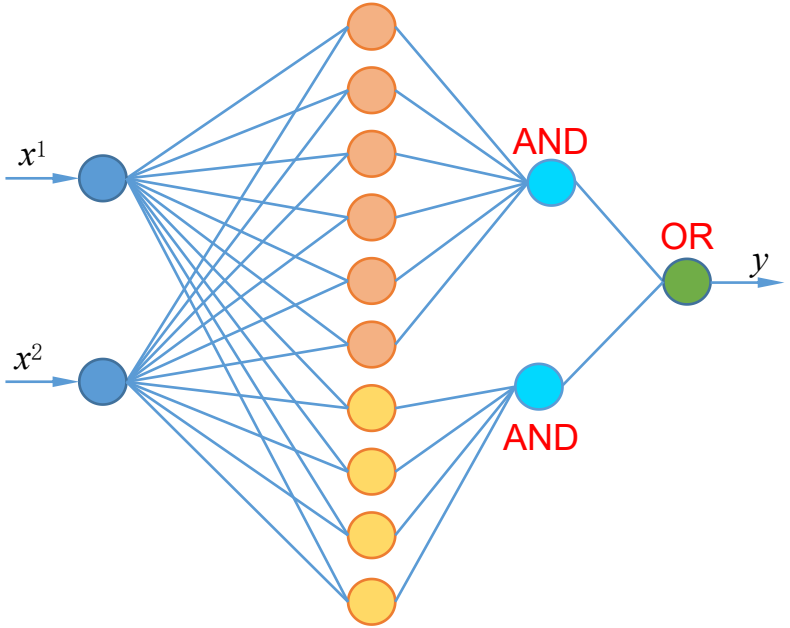

x^2
x^1


x^1
x^2
y

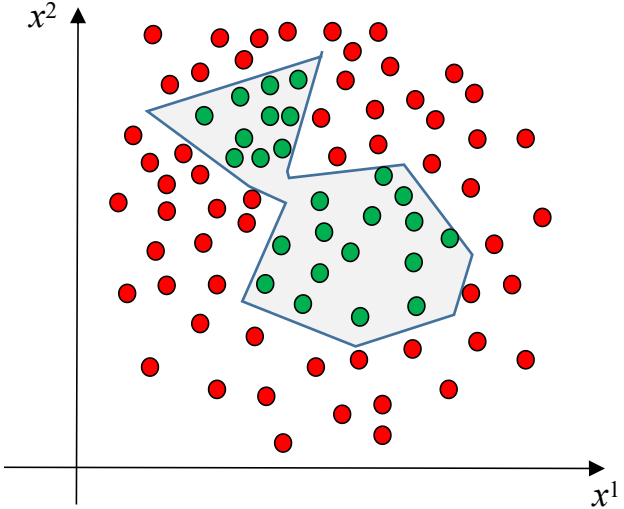

x^2
x^1

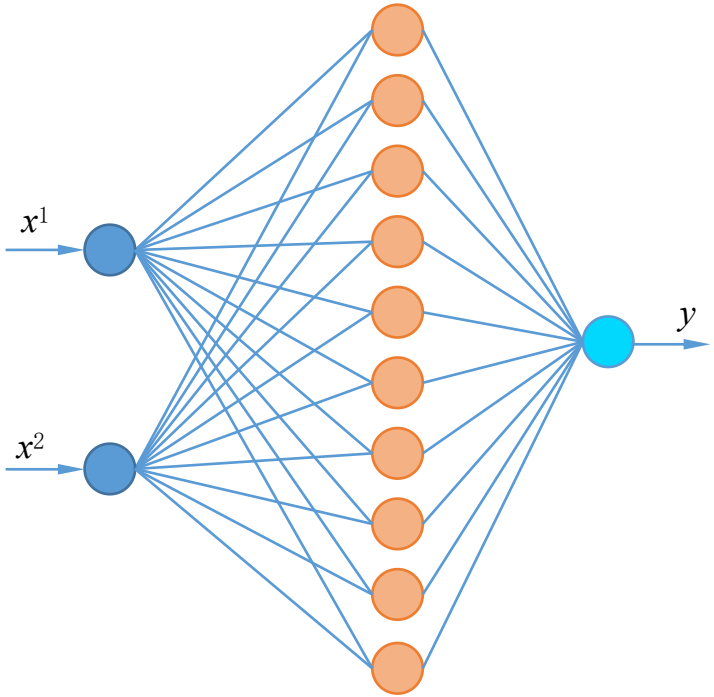

x^1
x^2
AND



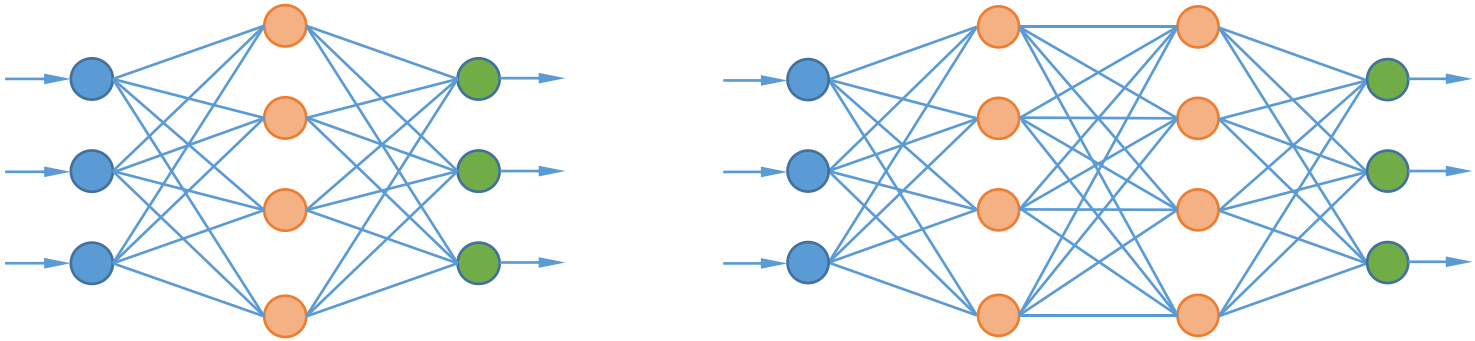


如果神经网络中有**足够的隐含层**，每个隐含层中有**足够多的神经元**，神经网络就可以表示**任意复杂函数**或**空间分布**


x^2
x^1

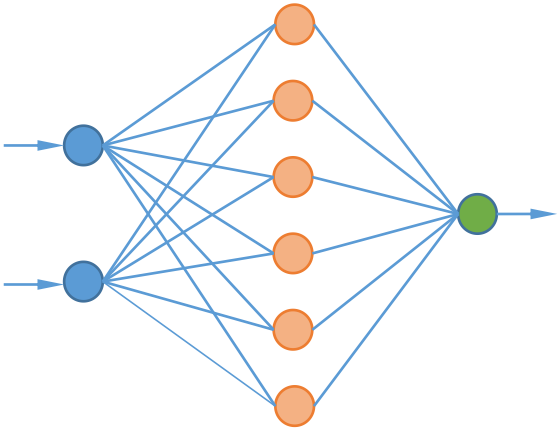

x^1
x^2
y

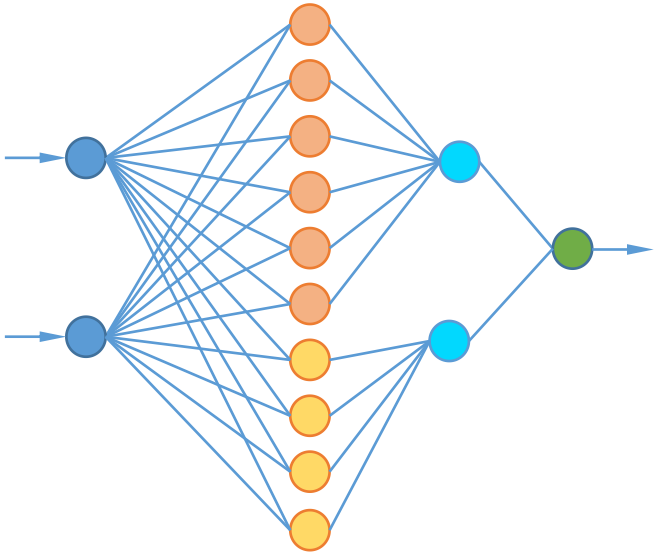
口 前馈神经网络 (Feedforward Neural Networks)



- 每层神经元只与**前一层**的神经元相连
- 处于**同一层**的神经元之间**没有连接**
- 各层间**没有反馈**，不存在跨层连接

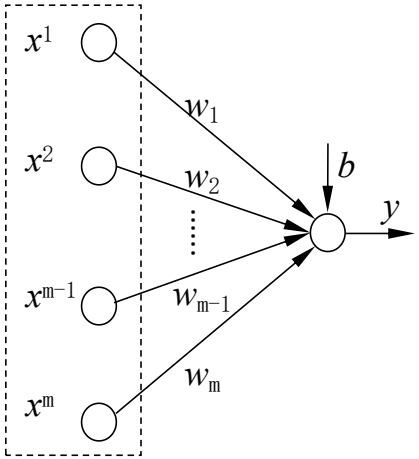
全连接网络 (Full Connected Network)

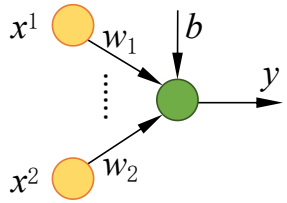




12.3.2 超参数和验证集

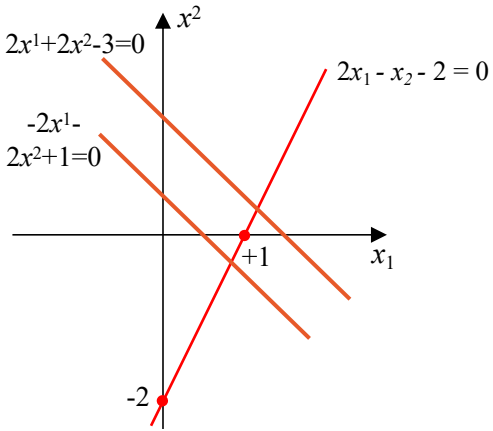
感知机

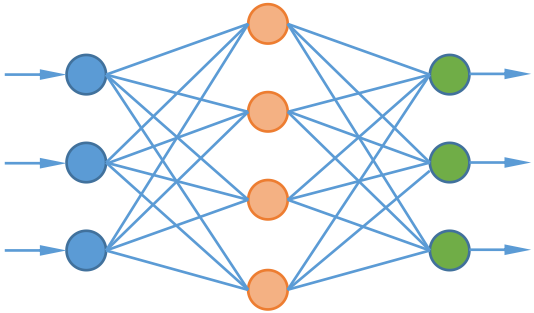


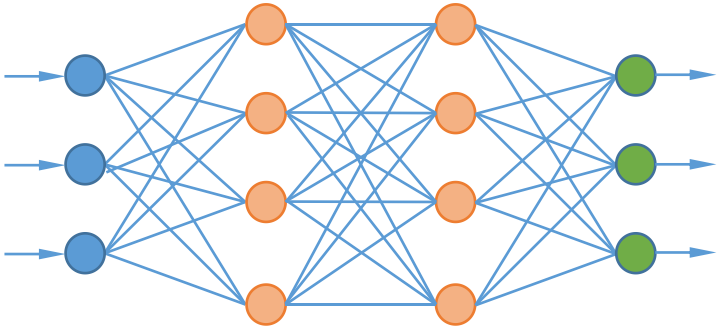


$$y=\sigma(z)=\sigma(w_1x^1+w_2x^2+b)$$

w_1, w_2, b	线性分类器	
2, - 1, 2	$y=\sigma(2x^1 - x_2 - 2)$	
2, 2, - 3	$y=\sigma(2x^1+2x^2-3)$	AND
-2, -2, 1	$y=\sigma(-2x^1-2x^2+1)$	NOR







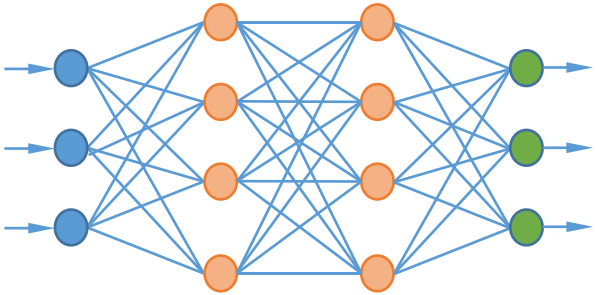
万能近似定理

在前馈型神经网络中，只要有**一个隐含层**，并且这个隐含层中有**足够多的神经元**，就可以逼近任意一个**连续的**函数或空间分布

多隐含层神经网络

- 能够表示**非连续的**函数或空间区域
- 减少泛化误差
- 减少每层神经元的数量

隐含层的设计



通过实验来确定超参数

layer∈(1,2,3)

node∈(4,16,32,64)

可能的组合：3×4=12种

- 使用**训练集**训练好所有模型
- 使用**同一个**测试集，得到不同模型上的误差
- 选出测试集上**误差最小**的模型

训练集、验证集和测试集

训练集：训练模型，确定模型参数

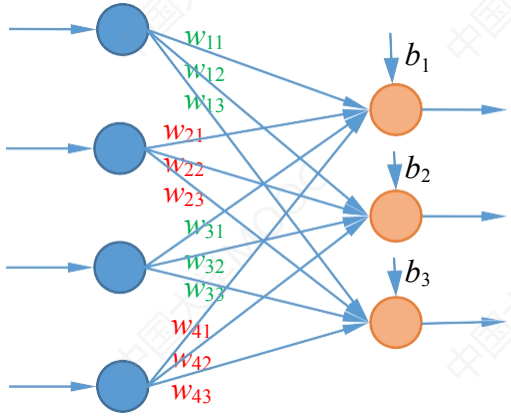
验证集：确定超参数

测试集：评估模型的泛化能力

12.4 误差反向传播算法

感知机 / 单层神经网络：线性分类

多层神经网络：非线性分类

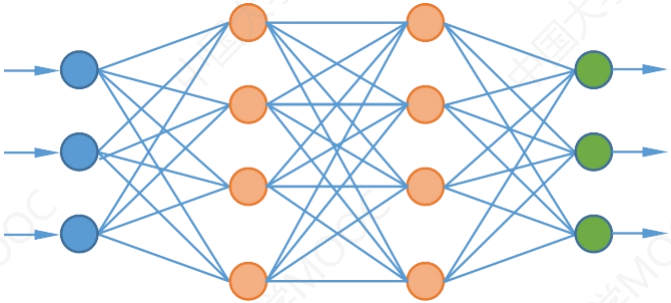


$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{31} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{bmatrix}$$

$$B = [\, b_1, b_2, b_3 \,]$$

$$W^{(k+1)} = W^{(k)} - \eta \frac{\partial Loss(W,B)}{\partial W}$$

$$B^{(k+1)} = B^{(k)} - \eta \frac{\partial Loss(W,B)}{\partial B}$$



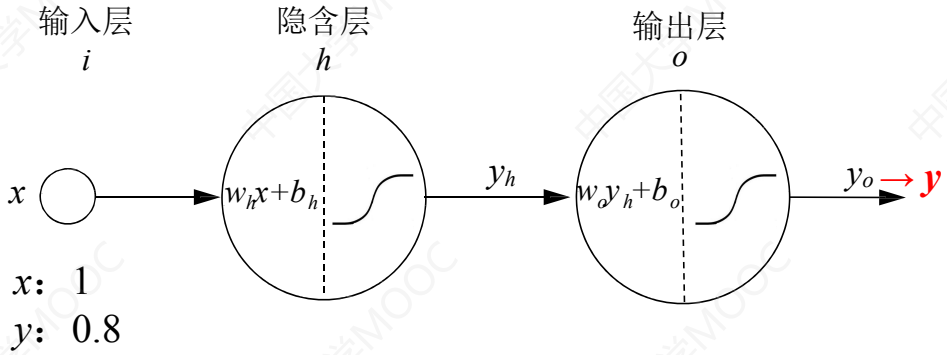
误差反向传播算法（Backpropagation，BP）

利用链式法则，反向传播损失函数的梯度信息，计算出损失函数对网络中所有模型参数的梯度

神经网络的训练

- 使用误差反向传播算法计算梯度
- 使用梯度下降法学习模型参数

1-1-1神经网络的误差反向传播

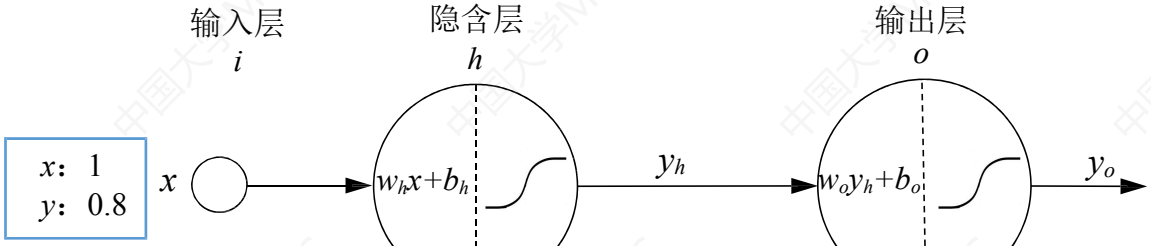


$$z_h = w_h x + b_h \qquad y_h = \frac{1}{1+e^{-z_h}}$$

$$z_o = w_o y_h + b_o \qquad y_o = \frac{1}{1+e^{-z_o}}$$

Step1: 设置模型参数初始值

w_h=0.2, b_h=0.1, w_o=0.3, b_o=0.2



Step2: 正向计算预测值

$$y_h=\frac{1}{1+e^{-(0.2\times1+0.1)}}=0.57$$

$$y_o=\frac{1}{1+e^{-(0.3\times0.57+0.2)}}=0.59$$

Step3: 计算误差

$$Loss=\frac{1}{2}(y-y_o)^2=0.02205$$

Step4: 误差反向传播

$$w_o^{(k+1)}=w_o^{(k)}-\eta\frac{\partial Loss}{\partial w_o}$$

$$b_o^{(k+1)}=b_o^{(k)}-\eta\frac{\partial Loss}{\partial b_o}$$

链式求导法则

$$\frac{\partial Loss}{\partial w_o} = \frac{\partial Loss}{\partial y_o} \cdot \frac{\partial y_o}{\partial z_o} \cdot \frac{\partial z_o}{\partial w_o} = -0.21 \times 0.2419 \times 0.57 = -0.02895543$$

$$Loss = \frac{1}{2}(y - y_o)^2 \qquad \frac{\partial Loss}{\partial y_o} = 2 \times \frac{1}{2} \times -(y - y_o) = -(0.8 - 0.59) = -0.21$$

$$y_o = \frac{1}{1 + e^{-z_o}} \qquad \frac{\partial y_o}{\partial z_o} = \frac{e^{-z_o}}{(1 + e^{-z_o})^2} = y_o(1 - y_o)$$

$$= 0.59 \times (1 - 0.59) = 0.2419$$

$$z_o = w_o y_h + b_o \qquad \frac{\partial z_o}{\partial w_o} = y_h = 0.57$$

$$\eta = 0.5$$

$$w_o^{(1)} = w_o^{(0)} - \eta \frac{\partial Loss}{\partial w_o}$$

$$= 0.3 - 0.5 \times (-0.02895543)$$

$$= 0.314477715$$

链式求导法则

$$\eta = 0.5$$

$$\frac{\partial Loss}{\partial w_o} = \frac{\partial Loss}{\partial y_o} \cdot \frac{\partial y_o}{\partial z_o} \cdot \frac{\partial z_o}{\partial w_o} = -0.21 \times 0.2419 \times 0.57 = -0.02895543$$

$$w_o^{(1)} = w_o^{(0)} - \eta \frac{\partial Loss}{\partial w_o} = 0.314477715$$

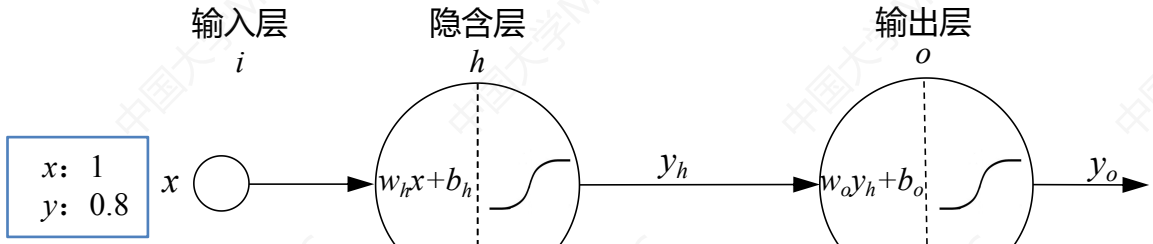
$$\frac{\partial Loss}{\partial b_o} = \frac{\partial Loss}{\partial y_o} \cdot \frac{\partial y_o}{\partial z_o} \cdot \frac{\partial z_o}{\partial b_o} = -0.21 \times 0.2419 = -0.050799$$

$$b_o^{(1)} = b_o^{(0)} - \eta \frac{\partial Loss}{\partial b_o} = 0.2253995$$

$$z_o = w_o y_h + b_o \qquad \frac{\partial z_o}{\partial b_o} = 1$$

Step1: 设置模型参数初始值

$w_h=0.2,\ b_h=0.1,\ w_o=0.3,\ b_o=0.2$



Step2: 正向计算预测值

$$y_h=\frac{1}{1+e^{-(0.2\times1+0.1)}}=0.57 \qquad y_o=\frac{1}{1+e^{-(0.3\times0.57+0.2)}}=0.59$$

Step3: 计算误差

$$Loss=\frac{1}{2}(y-y_o)^2=0.02205$$

Step4: 误差反向传播

$$w_h^{(k+1)}=w_h^{(k)}-\eta\frac{\partial Loss}{\partial w_h} \qquad b_h^{(k+1)}=b_h^{(k)}-\eta\frac{\partial Loss}{\partial b_h}$$

$$w_o^{(1)}=w_o^{(0)}-\eta\frac{\partial Loss}{\partial w_o}=0.314477715 \qquad b_o^{(1)}=b_o^{(0)}-\eta\frac{\partial Loss}{\partial b_o}=0.2253995$$

链式求导法则

$$\frac{\partial Loss}{\partial w_h}=\frac{\partial Loss}{\partial y_o}\cdot\frac{\partial y_o}{\partial z_o}\cdot\frac{\partial z_o}{\partial y_h}\cdot\frac{\partial y_h}{\partial z_h}\cdot\frac{\partial z_h}{\partial w_h}=-0.21\times0.2419\times0.3\times0.2451\times1=-0.00373525$$

$$Loss=\frac{1}{2}(y-y_o)^2 \qquad \frac{\partial Loss}{\partial y_o}=2\times\frac{1}{2}\times-(y-y_o)=-(0.8-0.59)=\boxed{-0.21}$$

$$y_o=\frac{1}{1+e^{-z_o}} \qquad \frac{\partial y_o}{\partial z_o}=\frac{e^{-z_o}}{(1+e^{-z_o})^2}=y_o(1-y_o)=\boxed{0.2419}$$

$$z_o=w_o y_h+b_o \qquad \frac{\partial z_o}{\partial y_h}=w_o=\boxed{0.3}$$

$$y_h=\frac{1}{1+e^{-z_h}} \qquad \frac{\partial y_h}{\partial z_h}=y_h(1-y_h)=0.57\times(1-0.57)=\boxed{0.2451}$$

$$z_h=w_h x+b_h \qquad \frac{\partial z_h}{\partial w_h}=x=\boxed{1}$$

链式求导法则

$$\frac{\partial Loss}{\partial b_h}=\frac{\partial Loss}{\partial y_o}\cdot\frac{\partial y_o}{\partial z_o}\cdot\frac{\partial z_o}{\partial y_h}\cdot\frac{\partial y_h}{\partial z_h}\cdot\frac{\partial z_h}{\partial b_h}=-0.21\times0.2419\times0.3\times0.2451\times1=-0.00373525$$

$$Loss=\frac{1}{2}(y-y_o)^2 \qquad \frac{\partial Loss}{\partial y_o}=2\times\frac{1}{2}\times-(y-y_o)=-(0.8-0.59)=-0.21$$

$$y_o=\frac{1}{1+e^{-z_o}} \qquad \frac{\partial y_o}{\partial z_o}=\frac{e^{-z_o}}{(1+e^{-z_o})^2}=y_o(1-y_o)=0.2419$$

$$z_o=w_o y_h+b_o \qquad \frac{\partial z_o}{\partial y_h}=w_o=0.3$$

$$y_h=\frac{1}{1+e^{-z_h}} \qquad \frac{\partial y_h}{\partial z_h}=y_h(1-y_h)=0.57\times(1-0.57)=0.2451$$

$$z_h=w_h x+b_h \qquad \frac{\partial z_h}{\partial b_h}=1$$

链式求导法则

$$\frac{\partial Loss}{\partial w_h}=\frac{\partial Loss}{\partial y_o}\cdot\frac{\partial y_o}{\partial z_o}\cdot\frac{\partial z_o}{\partial y_h}\cdot\frac{\partial y_h}{\partial z_h}\cdot\frac{\partial z_h}{\partial w_h}=-0.21\times0.2419\times0.3\times0.2451\times1=-0.00373525$$

$$\frac{\partial Loss}{\partial b_h}=\frac{\partial Loss}{\partial y_o}\cdot\frac{\partial y_o}{\partial z_o}\cdot\frac{\partial z_o}{\partial y_h}\cdot\frac{\partial y_h}{\partial z_h}\cdot\frac{\partial z_h}{\partial b_h}=-0.21\times0.2419\times0.3\times0.2451\times1=-0.00373525$$

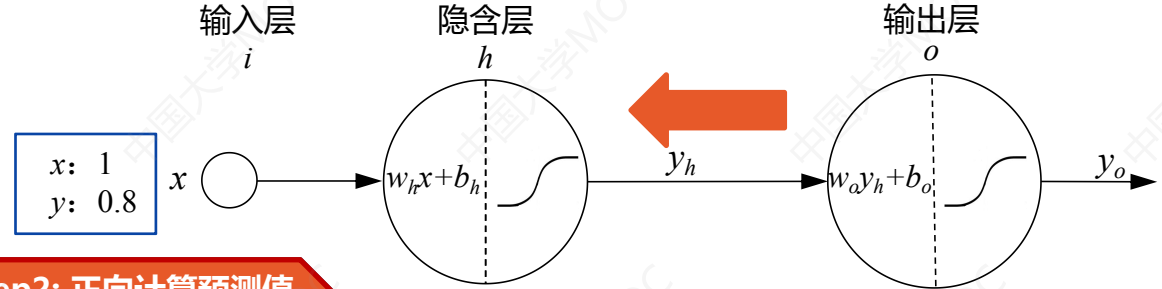
迭代公式

$$w_h^{(1)}=w_h^{(0)}-\eta\frac{\partial Loss}{\partial w_h}=0.2-0.5\times(-0.00373525)=0.201867625$$

$$b_h^{(1)}=b_h^{(0)}-\eta\frac{\partial Loss}{\partial b_h}=0.1-0.5\times(-0.00373525)=0.101867625$$

Step1: 设置模型参数初始值

$w_h=0.2,\ b_h=0.1,\ w_o=0.3,\ b_o=0.2$



Step2: 正向计算预测值

$$y_h=\frac{1}{1+e^{-(0.2\times1+0.1)}}=0.57 \qquad y_o=\frac{1}{1+e^{-(0.3\times0.57+0.2)}}=0.59$$

Step3: 计算误差

$$Loss=\frac{1}{2}(y-y_o)^2=0.02205$$

Step4: 误差反向传播

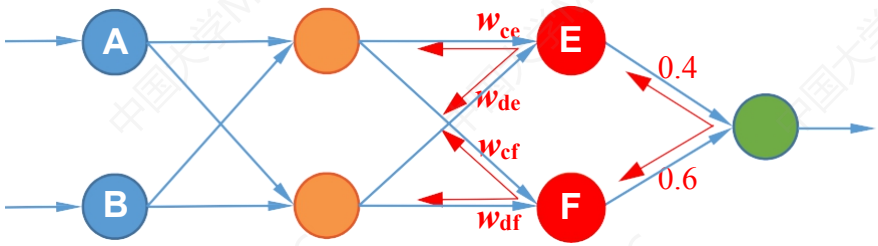
$$w_h^{(1)}=w_h^{(0)}-\eta\frac{\partial Loss}{\partial w_h}=0.201867625$$

$$b_h^{(1)}=b_h^{(0)}-\eta\frac{\partial Loss}{\partial b_h}=0.101867625$$

$$w_o^{(1)}=w_o^{(0)}-\eta\frac{\partial Loss}{\partial w_o}=0.314477715$$

$$b_o^{(1)}=b_o^{(0)}-\eta\frac{\partial Loss}{\partial b_o}=0.2253995$$

隐含层有多个神经元的误差反向传播



$$LossE = 0.4LossG$$
$$LossF = 0.6LossG$$

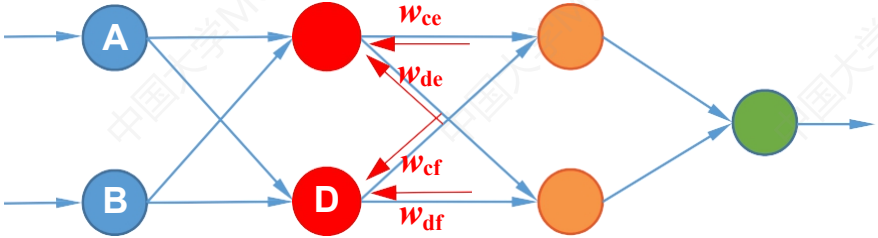
$$LossC_E = \frac{w_{ce}}{w_{ce}+w_{de}} LossE$$

$$LossD_E = \frac{w_{de}}{w_{ce}+w_{de}} LossE$$

$$LossC_F = \frac{w_{cf}}{w_{cf}+w_{df}} LossF$$

$$LossD_F = \frac{w_{df}}{w_{cf}+w_{df}} LossF$$

隐含层有多个神经元的误差反向传播



$$LossE = 0.4LossG$$
$$LossF = 0.6LossG$$

$$LossC = LossC_E + LossC_F = \frac{w_{ce}}{w_{ce}+w_{de}}LossE + \frac{w_{cf}}{wcf+wdf}LossF$$

$$LossD = LossD_E + LossD_F = \frac{w_{de}}{w_{ce}+w_{de}}LossE + \frac{w_{df}}{w_{cf}+w_{df}}LossF$$

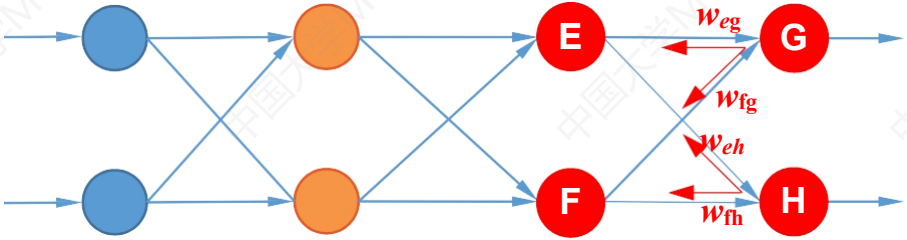
$$LossC_E = \frac{w_{ce}}{w_{ce}+w_{de}}LossE$$

$$LossD_E = \frac{w_{de}}{w_{ce}+w_{de}}LossE$$

$$LossC_F = \frac{w_{cf}}{w_{cf}+w_{df}}LossF$$

$$LossD_F = \frac{w_{df}}{w_{cf}+w_{df}}LossF$$

隐含层有多个神经元的误差反向传播



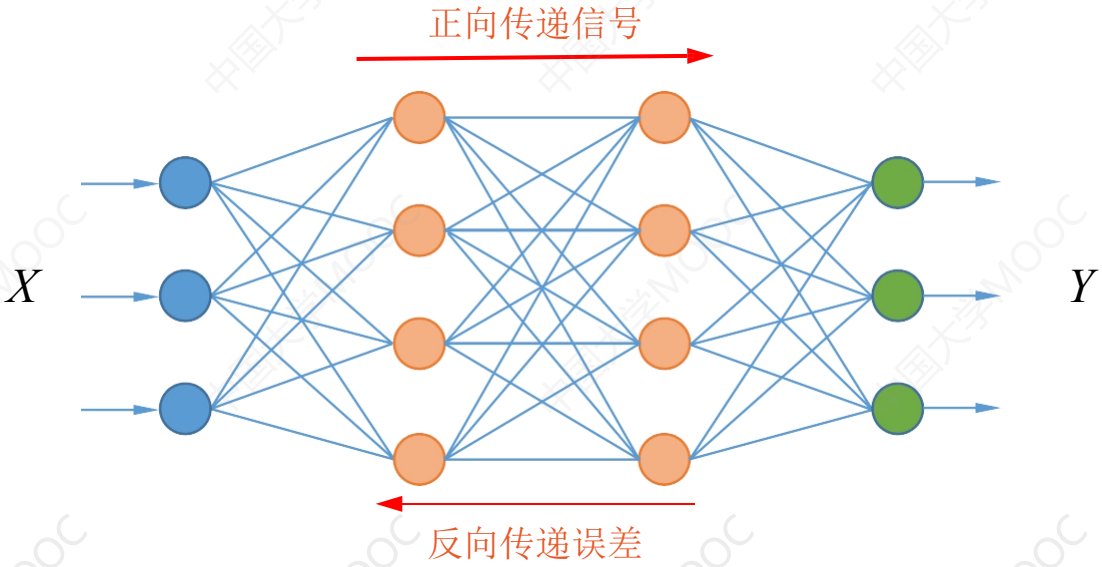
$$LossE = \frac{w_{eg}}{w_{eg} + w_{fg}} LossG + \frac{w_{eh}}{w_{eh} + w_{fh}} LossH$$

$$LossF = \frac{w_{fg}}{w_{eg} + w_{fg}} LossG + \frac{w_{fh}}{w_{eh} + w_{fh}} LossH$$

多层神经网络的训练

通过梯度下降法训练模型参数

通过误差反向传播法计算梯度



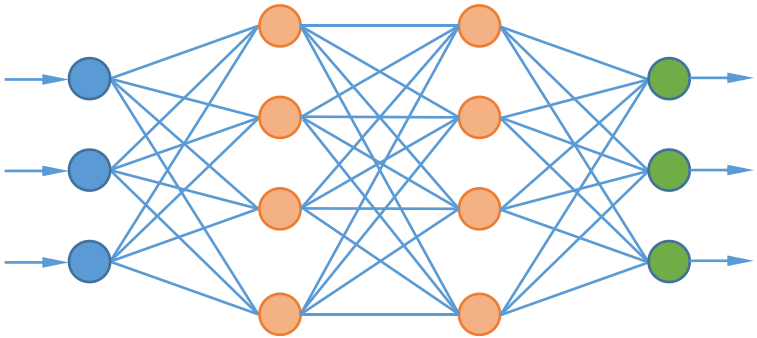

tensorflow / tensorflow
TensorFlow: Open sou
Clone this repo:
git clone https://tensor
Branches
master

12.5 激活函数

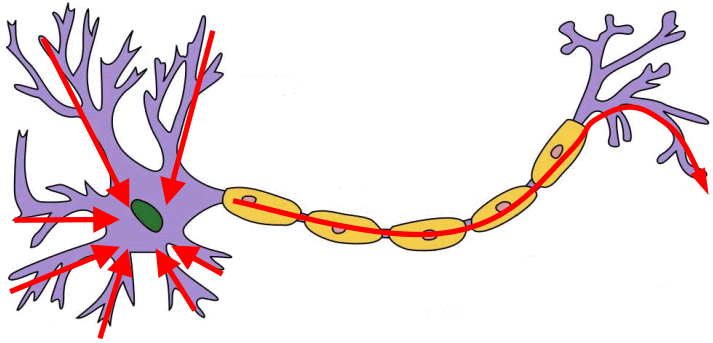
生物神经网络

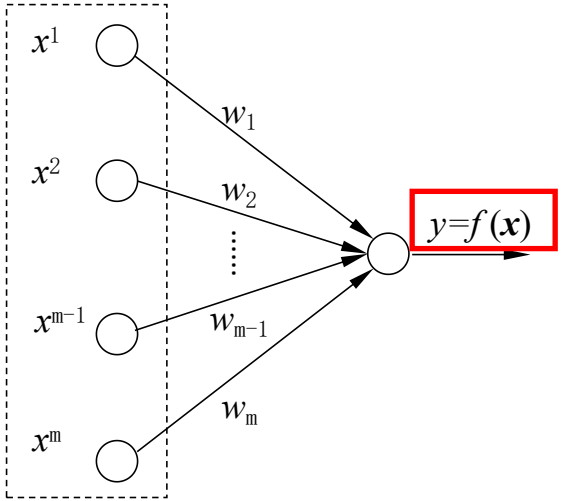
人工神经网络

由人工**神经元**和神经元之间的**连接**构成的一种**数学模型**



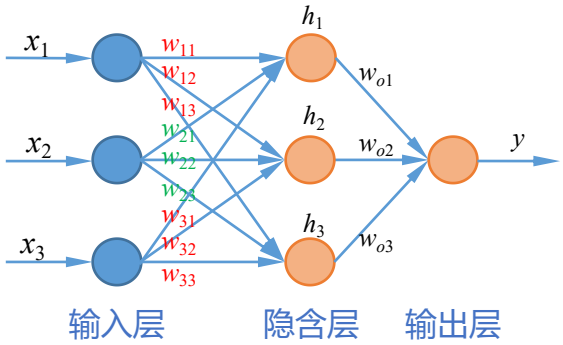
激活函数的作用





线性函数

无论有多少层隐含层，输出都是输入特征的**线性组合**



隐含层

$$h_1 = \sum_{j=1}^{3} w_{1j}x^j + w_{10} = W_1^T X$$

$$h_2 = \sum_{j=1}^{3} w_{2j}x^j + w_{20} = W_2^T X$$

$$h_3 = \sum_{j=1}^{3} w_{3j}x^j + w_{30} = W_3^T X$$

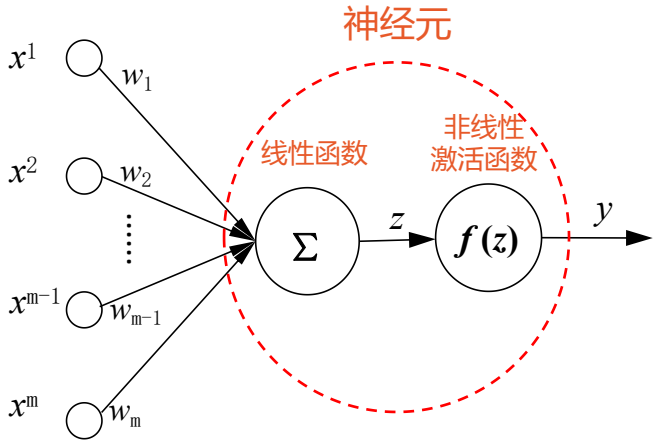
输出层

$$y = \sum_{j=1}^{3} w_{oj}h_j + w_o = W_o H_j$$

$$= W_o(W_1^T X + W_2^T X + W_3^T X)$$

任意函数

- 灵活、功能强大；
- 过于复杂，难于训练；
- 通用性差



- 所有输入的**线性组合**

$$z = \sum_{j=1}^{m} w_j x^j + w_0 = W^T X$$

- 使用一个**非线性**的**激活函数**

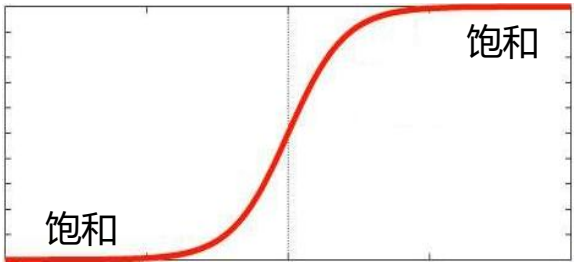
$$y = f(z) = f(W^T X)$$

激活函数的性质

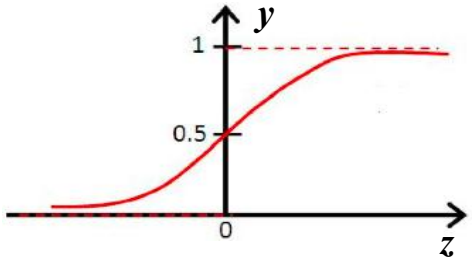
- 简单的非线性函数
- 连续并可导
- 单调函数

常用的激活函数

- sigmoid函数

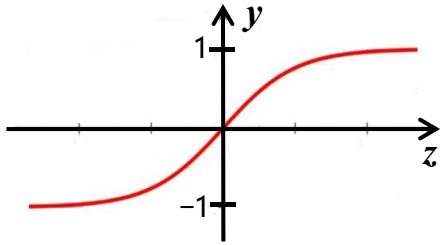


- 对数几率函数 logistic



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

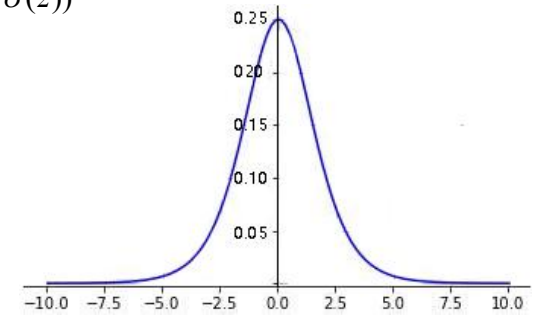
- 双曲正切函数 Tanh



$$tanh(z)=\frac{sinh\,z}{cosh\,z}=\frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$$

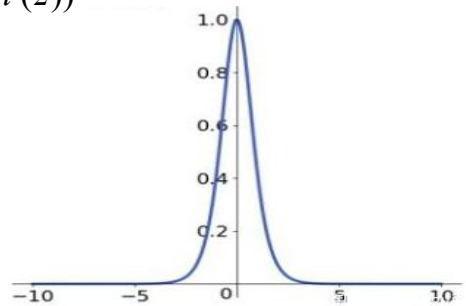
- logistic函数的导函数

$$\sigma'(z) = \sigma(z)(1-\sigma(z))$$



- Tanh函数的导函数

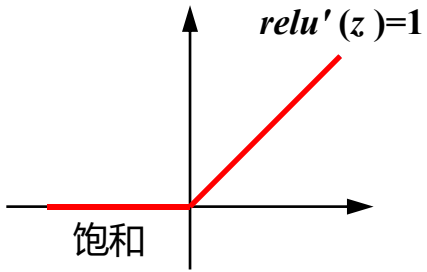
$$tanh'(z) = 1 - (tanh(z))^2$$



梯度消失问题

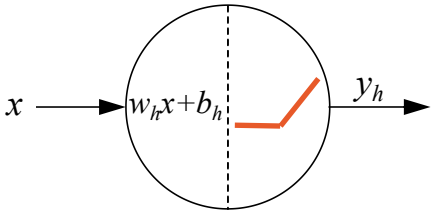
- 多层神经网络，误差反向传播
- 使用链式法则，误差经过每一层会不断衰减
- 网络层数较深时，梯度值趋近于0，参数更新几乎停滞

ReLU函数 (Rectified Linear Unit，修正线性单元)



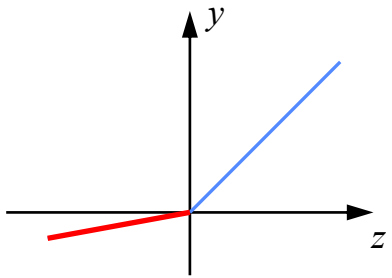
$$y = \begin{cases} z & z \geq 0 \\ 0 & z < 0 \end{cases}$$

$$= max(0, z)$$



- z>0时，导数等于1，缓解了梯度消失问题
- 不存在幂运算，计算速度快
- 导数恒等于1，训练模型收敛速度快
- 输出不是以0为均值的，会影响收敛的速度
- z<0时，梯度为0，神经元死亡

Leaky-ReLU函数



$$y=\begin{cases} z & z\geq 0 \\ \dfrac{z}{a} & z<0 \end{cases} \qquad a\in(1,+\infty)$$

- 避免了ReLU神经元死亡
- 神经网络的计算和训练速度快
- 超参数a需要人工调整

PReLU函数 和 RReLU函数

PReLU（Parameteric Rectified Linear Unit，参数化修正线性单元）

$$y = \begin{cases} z & z \geq 0 \\ \alpha z & z < 0 \end{cases}$$

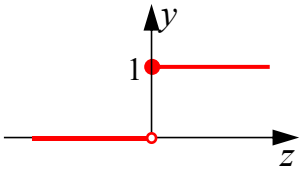
α：可训练参数

RReLU (Randomized Leaky Rectified Linear Unit，随机纠正线性单元)

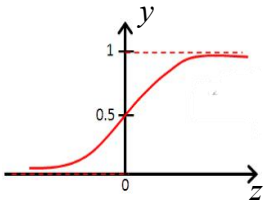
- 训练阶段，负值部分的斜率是随机分配的（均匀分布）
- 测试阶段，负值的斜率是固定的（训练阶段所有α的平均值）

激活函数的特点

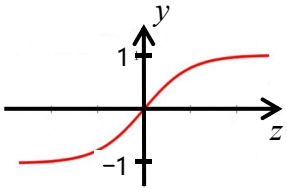
- 包含充分的梯度信息
- 能够识别阈值



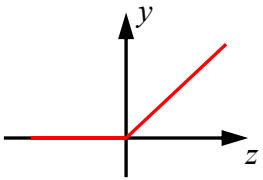
阶跃函数



Logistic函数



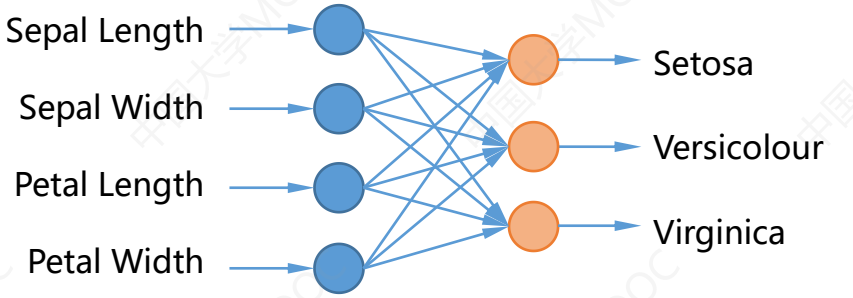
Tanh 函数



ReLU 函数

12.6 实例：实现多层神经网络

单层神经网络



- 激活函数 softmax函数
- 损失函数 交叉熵损失函数

$$Y = X\,W + B$$

输出 输入属性 权值矩阵 阈值

```
np.random.seed(612)
W = tf.Variable(np.random.randn(4,3),dtype=tf.float32)
B = tf.Variable(np.zeros([3]),dtype=tf.float32)
```

```
for i in range(0,iter+1):
    with tf.GradientTape() as tape:
        PRED_train=tf.nn.softmax(tf.matmul(X_train,W)+B)
        Loss_train=tf.reduce_mean(tf.keras.losses.categorical_crossentropy(y_true=Y_train,y_pred=PRED_train))

    PRED_test=tf.nn.softmax(tf.matmul(X_test,W)+B)
    Loss_test=tf.reduce_mean(tf.keras.losses.categorical_crossentropy(y_true=Y_test,y_pred=PRED_test))

    accuracy_train=tf.reduce_mean(tf.cast(tf.equal(tf.argmax(PRED_train.numpy(),axis=1),y_train),tf.float32))
    accuracy_test=tf.reduce_mean(tf.cast(tf.equal(tf.argmax(PRED_test.numpy(),axis=1),y_test),tf.float32))

    acc_train.append(accuracy_train)
    acc_test.append(accuracy_test)
    cce_train.append(Loss_train)
    cce_test.append(Loss_test)

    grads = tape.gradient(Loss_train,[W,B])
    W.assign_sub(learn_rate*grads[0])
    B.assign_sub(learn_rate*grads[1])

    if i % display_step == 0:
        print("i: %i, TrainAcc:%f, TrainLoss: %f ,TestAcc:%f, TestLoss: %f" % (i,accuracy_train,Loss_train,accuracy_test,Loss_test))
```

$Y = XW + B$

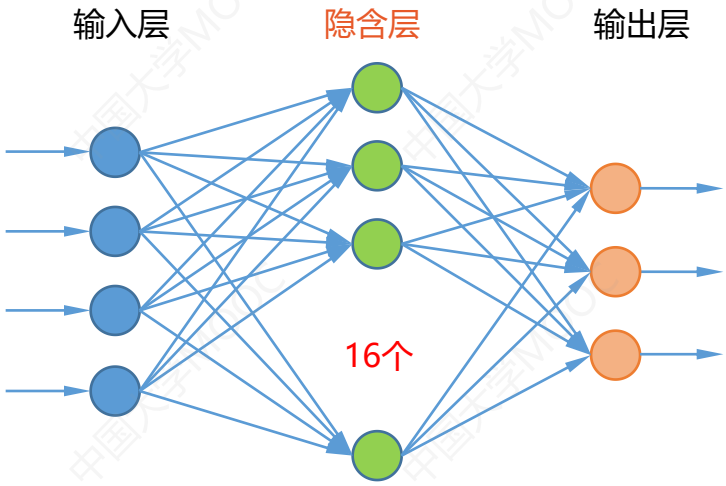
- 设置超参数

```
learn_rate = 0.5
iter = 50

display_step =10
```

- 训练结果

```
i: 0, TrainAcc:0.333333, TrainLoss: 2.066978 ,TestAcc:0.266667, TestLoss: 1.880856
i: 10, TrainAcc:0.875000, TrainLoss: 0.339410 ,TestAcc:0.866667, TestLoss: 0.461705
i: 20, TrainAcc:0.875000, TrainLoss: 0.279647 ,TestAcc:0.866667, TestLoss: 0.368414
i: 30, TrainAcc:0.916667, TrainLoss: 0.245924 ,TestAcc:0.933333, TestLoss: 0.314814
i: 40, TrainAcc:0.933333, TrainLoss: 0.222922 ,TestAcc:0.933333, TestLoss: 0.278643
i: 50, TrainAcc:0.933333, TrainLoss: 0.205636 ,TestAcc:0.966667 TestLoss: 0.251937
```

多层神经网络结构设计



W_1 (4,16)　B_1 (16,)　W_2 (16,3)　B_2 (3,)

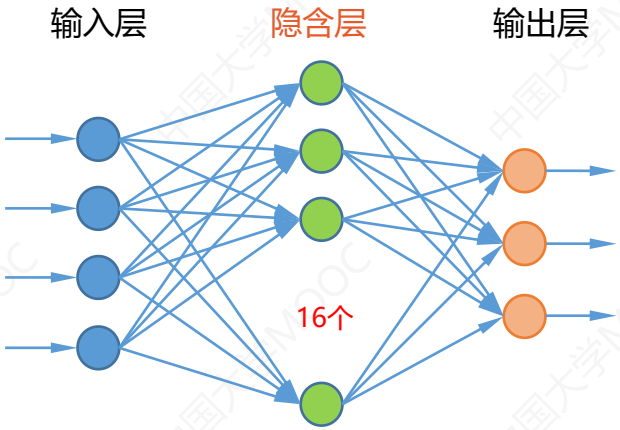
$H = X W_1 + B_1$　　$Y = H W_2 + B_2$

隐含层激活函数　relu函数

输出层激活函数　softmax函数

损失函数　交叉熵损失函数

多层神经网络的实现



W_1 (4,16)　B_1 (16,)　W_2 (16,3)　B_2 (3,)

$H = \mathrm{Relu}(XW_1 + B_1)$　$Y = Softmax(\mathrm{HW}_2 + B_2)$

设置模型参数

```
np.random.seed(612)
W1 = tf.Variable(np.random.randn(4,16),dtype=tf.float32)
B1= tf.Variable(tf.zeros([16]),dtype=tf.float32)
W2 = tf.Variable(np.random.randn(16,3),dtype=tf.float32)
B2= tf.Variable(tf.zeros([3]),dtype=tf.float32)
```

定义网络结构

```
Hidden_train=tf.nn.relu(tf.matmul(X_train,W1)+B1)
PRED_train=tf.nn.softmax(tf.matmul(Hidden_train,W2)+B2)
Loss_test=tf.reduce_mean(tf.keras.losses.categorical_crossentropy
                         (y_true=Y_test,y_pred=PRED_test))
```

$H = XW_1 + B_1$

$Y = HW_2 + B_2$

更新模型参数

```
grads = tape.gradient(Loss_train, [W1,B1,W2,B2])

W1.assign_sub(learn_rate*grads[0])
B1.assign_sub(learn_rate*grads[1])
W2.assign_sub(learn_rate*grads[2])
B2.assign_sub(learn_rate*grads[3])
```

完整程序：多层神经网络实现鸢尾花分类

- 导入库，设置GPU模式

```
In [1]: import tensorflow as tf
        print("TensorFlow version:", tf.__version__)

        TensorFlow version: 2.0.0

In [2]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt

In [3]: gpus = tf.config.experimental.list_physical_devices('GPU')
        tf.config.experimental.set_memory_growth(gpus[0], True)
```

- 加载数据，转换为NumPy数组

```
In [4]: TRAIN_URL = "http://download.tensorflow.org/data/iris_training.csv"
        train_path = tf.keras.utils.get_file(TRAIN_URL.split('/')[-1], TRAIN_URL)

        TEST_URL = "http://download.tensorflow.org/data/iris_test.csv"
        test_path = tf.keras.utils.get_file(TEST_URL.split('/')[-1], TEST_URL)

In [5]: df_iris_train = pd.read_csv(train_path, header=0)
        df_iris_test = pd.read_csv(test_path, header=0)

In [6]: iris_train=np.array(df_iris_train)
        iris_test=np.array(df_iris_test)

In [7]: iris_train.shape,iris_test.shape
Out[7]: ((120, 5), (30, 5))
```

- 数据预处理

```
In [8]: x_train=iris_train[:,0:4]
        y_train=iris_train[:,4]

        x_test=iris_test[:,0:4]
        y_test=iris_test[:,4]

In [9]: x_train.shape,y_train.shape
Out[9]: ((120, 4), (120,))

In [10]: x_test.shape,y_test.shape
Out[10]: ((30, 4), (30,))

In [11]: x_train=x_train-np.mean(x_train,axis=0)
         x_test=x_test-np.mean(x_test,axis=0)
```

```
In [12]: X_train=tf.cast(x_train, tf.float32)
         Y_train=tf.one_hot(tf.constant(y_train, dtype=tf.int32), 3)

         X_test=tf.cast(x_test, tf.float32)
         Y_test=tf.one_hot(tf.constant(y_test, dtype=tf.int32), 3)

In [13]: X_train.shape, Y_train.shape

Out[13]: (TensorShape([120, 4]), TensorShape([120, 3]))

In [14]: X_test.shape, Y_test.shape

Out[14]: (TensorShape([30, 4]), TensorShape([30, 3]))
```

- 设置超参数和显示间隔

```
In [15]: learn_rate = 0.5
         iter = 50
         display_step =10
```

- 设置模型参数初始值

```
In [16]: np.random.seed(612)
         W1 = tf.Variable(np.random.randn(4,16),dtype=tf.float32)
         B1= tf.Variable(tf.zeros([16]),dtype=tf.float32)
         W2 = tf.Variable(np.random.randn(16,3),dtype=tf.float32)
         B2= tf.Variable(tf.zeros([3]),dtype=tf.float32)
```

- 训练模型

```
In [17]: acc_train=[]
         acc_test=[]
         cce_train=[]
         cce_test=[]
```

```
for i in range(0,iter+1):
    with tf.GradientTape() as tape:
        Hidden_train=tf.nn.relu(tf.matmul(X_train,W1)+B1)
        PRED_train=tf.nn.softmax(tf.matmul(Hidden_train,W2)+B2)
        Loss_train=tf.reduce_mean(tf.keras.losses.categorical_crossentropy(y_true=Y_train,y_pred=PRED_train))

        Hidden_test=tf.nn.relu(tf.matmul(X_test,W1)+B1)
        PRED_test=tf.nn.softmax(tf.matmul(Hidden_test,W2)+B2)
        Loss_test=tf.reduce_mean(tf.keras.losses.categorical_crossentropy(y_true=Y_test,y_pred=PRED_test))

    accuracy_train=tf.reduce_mean(tf.cast(tf.equal(tf.argmax(PRED_train.numpy(),axis=1),y_train),tf.float32))
    accuracy_test=tf.reduce_mean(tf.cast(tf.equal(tf.argmax(PRED_test.numpy(),axis=1),y_test),tf.float32))

    acc_train.append(accuracy_train)
    acc_test.append(accuracy_test)
    cce_train.append(Loss_train)
    cce_test.append(Loss_test)

    grads = tape.gradient(Loss_train,[W1,B1,W2,B2])
    W1.assign_sub(learn_rate*grads[0])
    B1.assign_sub(learn_rate*grads[1])
    W2.assign_sub(learn_rate*grads[2])
    B2.assign_sub(learn_rate*grads[3])

    if i % display_step == 0:
        print("i: %i, TrainAcc:%f, TrainLoss: %f ,TestAcc:%f, TestLoss: %f" % (i,accuracy_train,Loss_train,accuracy_test,Loss_test))
```

- 训练结果

learn_rate=0.5，2层神经网络

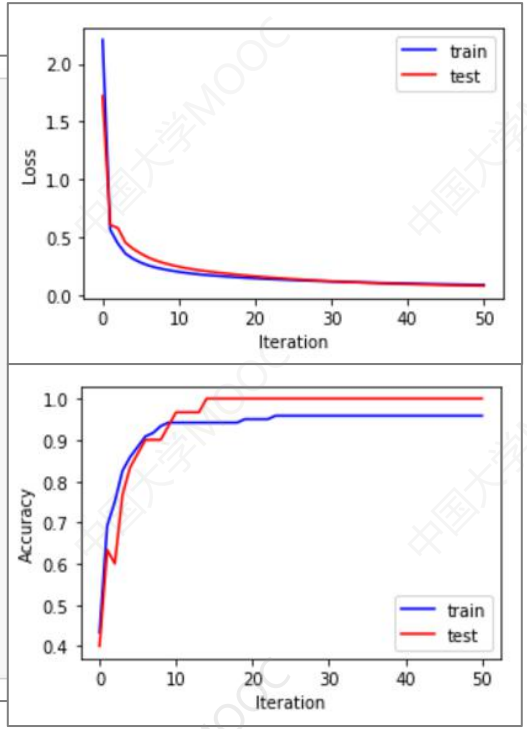
```
i: 0,  TrainAcc:0.433333,  TrainLoss: 2.205641 ,TestAcc:0.400000,  TestLoss: 1.721138
i: 10,  TrainAcc:0.941667,  TrainLoss: 0.205314 ,TestAcc:0.966667,  TestLoss: 0.249661
i: 20,  TrainAcc:0.950000,  TrainLoss: 0.149540 ,TestAcc:1.000000,  TestLoss: 0.167103
i: 30,  TrainAcc:0.958333,  TrainLoss: 0.122346 ,TestAcc:1.000000,  TestLoss: 0.124693
i: 40,  TrainAcc:0.958333,  TrainLoss: 0.105099 ,TestAcc:1.000000,  TestLoss: 0.099869
i: 50,  TrainAcc:0.958333,  TrainLoss: 0.092934 ,TestAcc:1.000000,  TestLoss: 0.084885
```

learn_rate=0.5，单层神经网络

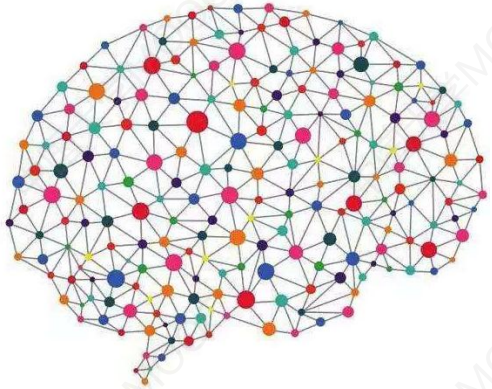
```
i: 0,  TrainAcc:0.333333,  TrainLoss: 2.066978 ,TestAcc:0.266667,  TestLoss: 1.880856
i: 10,  TrainAcc:0.875000,  TrainLoss: 0.339410 ,TestAcc:0.866667,  TestLoss: 0.461705
i: 20,  TrainAcc:0.875000,  TrainLoss: 0.279647 ,TestAcc:0.866667,  TestLoss: 0.368414
i: 30,  TrainAcc:0.916667,  TrainLoss: 0.245924 ,TestAcc:0.933333,  TestLoss: 0.314814
i: 40,  TrainAcc:0.933333,  TrainLoss: 0.222922 ,TestAcc:0.933333,  TestLoss: 0.278643
i: 50,  TrainAcc:0.933333,  TrainLoss: 0.205636 ,TestAcc:0.966667,  TestLoss: 0.251937
```

- 结果可视化

```
In [19]: plt.figure(figsize=(10,3))

         plt.subplot(121)
         plt.plot(cce_train,color="blue",label="train")
         plt.plot(cce_test,color="red",label="test")
         plt.xlabel("Iteration")
         plt.ylabel("Loss")
         plt.legend()

         plt.subplot(122)
         plt.plot(acc_train,color="blue",label="train")
         plt.plot(acc_test,color="red",label="test")
         plt.xlabel("Iteration")
         plt.ylabel("Accuracy")
         plt.legend()

         plt.show()
```



神经网络&深度学习
TENSORFLOW
Google

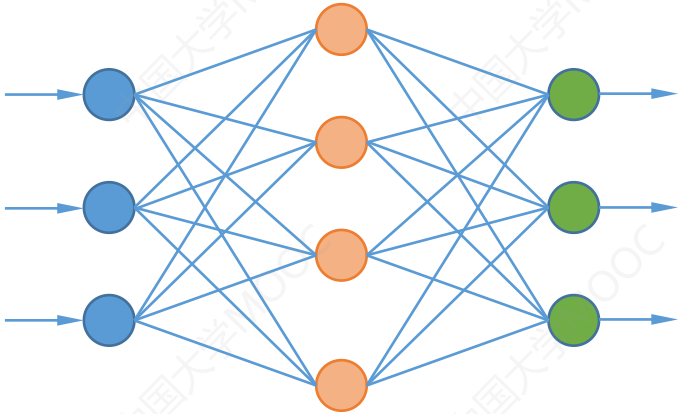


13 人工神经网络(2)

西安科技大学 牟琦
muqi@xust.edu.cn

13.1 小批量梯度下降法

多层神经网络——非线性分类问题



- 损失函数**不是凸函数**，很难计算解析解
- 通常采用**梯度下降法**，得到**数值解**

梯度下降法：求解函数极值问题

- **批量**梯度下降
- **随机**梯度下降
- **小批量**梯度下降

一元线性回归

$$Loss=\frac{1}{2}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2=\frac{1}{2}\sum_{i=1}^{n}(y_i-(wx_i+b))^2$$

n：样本数

批量梯度下降（Batch Gradient Descent，BGD）

- **每次迭代**都使用**所有样本**来计算偏导数

$$w^{(k+1)} = w^{(k)} - \eta \frac{\partial Loss(w,b)}{\partial w} = w^{(k)} - \eta \sum_{i=1}^{n} x_i (w^{(k)} x_i + b - y_i)$$

$$b^{(k+1)} = b^{(k)} - \eta \frac{\partial Loss(w,b)}{\partial b} = b^{(k)} - \eta \sum_{i=1}^{n} (w^{(k)} x_i + b - y_i)$$

- 由**所有样本**确定梯度方向
- 每一步都是准确地向着极值点趋近，**迭代次数少**
- 收敛于**全局极小值**或**局部极小值**点
- 可以利用**向量运算**进行**并行计算**
- 计算量大，训练时间长，不适合大规模数据集

样本集：**20万**个样本，**10次**迭代
计算量：20万×10 = **200万**

随机梯度下降（Stochastic Gradient Descent, SGD）

- **每次迭代**只选择**一个样本**训练模型，使网络的输出尽可能逼近这个样本的标签值
- **一轮**：使用**所有样本**训练一遍
- 反复**训练多轮**，直到网络对所有样本的误差足够小
- **参数更新非常频繁**，**无法快速收敛**
- 不易于实现并行计算

随机梯度下降通常是指**小批量梯度下降**算法

- **小批量梯度下降**（Mini-Batch Gradient Descent, MBGD）
- **小批量随机梯度下降**（Mini-Batch SGD）
  - 把数据分为多个**小批量**，每次迭代使用**一个小批量**来训练模型

$$Loss=\frac{1}{2}\sum_{i=1}^{t}(y_i-\hat{y}_i)^2=\frac{1}{2}\sum_{i=1}^{t}(y_i-(w^{(k)}x_i+b^{(k)}))^2$$

t：每一批中的样本数量

$$w^{(k+1)}=w^{(k)}-\eta\frac{\partial Loss(w^{(k)},b^{(k)})}{\partial w^{(k)}}=w^{(k)}-\eta\sum_{i=1}^{t}x_i(w^{(k)}x_i+b^{(k)}-y_i)$$

$$b^{(k+1)}=b^{(k)}-\eta\frac{\partial Loss(w^{(k)},b^{(k)})}{\partial b^{(k)}}=b^{(k)}-\eta\sum_{i=1}^{t}(w^{(k)}x_i+b^{(k)}-y_i)$$

- **小批量梯度下降**（Mini-Batch Gradient Descent, MBGD）
- **小批量随机梯度下降**（Mini-Batch SGD）
  - 把数据分为多个**小批量**，每次迭代使用**一个小批量**来训练模型
  - 每个**小批量中的所有样本**共同决定了本次迭代中的梯度方向
  - **一轮**：使用**所有小批量**训练一遍
  - 需要**训练多轮**，使网络对**所有样本**的误差足够小
  - **每次迭代**的**训练样本数固定**，与整个训练集的样本数量无关
  - 可以实现**并行运算**
  - 训练**大规模**数据集

样本集：**2000**个样本，10批，每个批中的样本数量：**200**

200000个样本，1000批，每个批中的样本数量：**200**

抽样（Sampling）

从数据集中**随机抽取**出来一部分样本，它们的特征可以**在一定程度上代表完整数据集的特征**

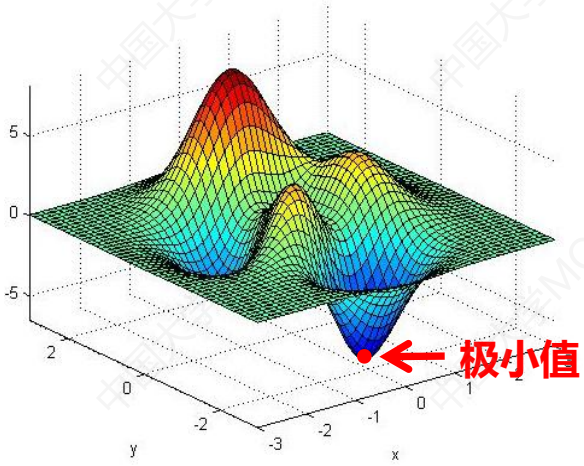
独立同分布：小批量样本能够代表整个样本集的特征

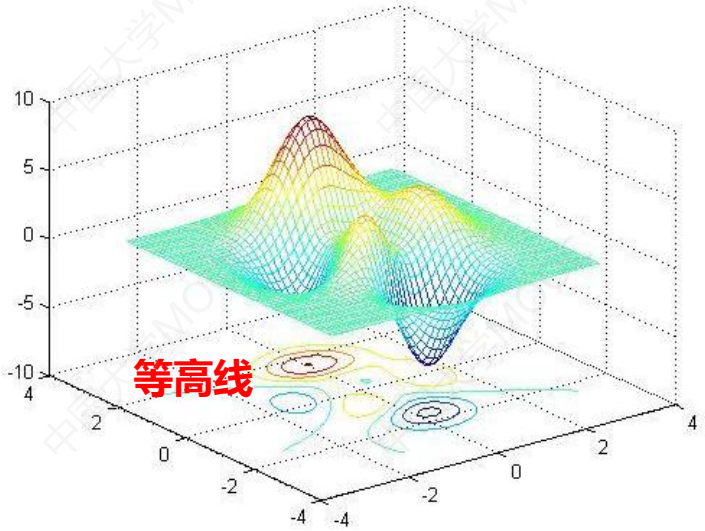
随机抽取：小批量样本的特征和整体样本的特征存在差别

小批量梯度下降

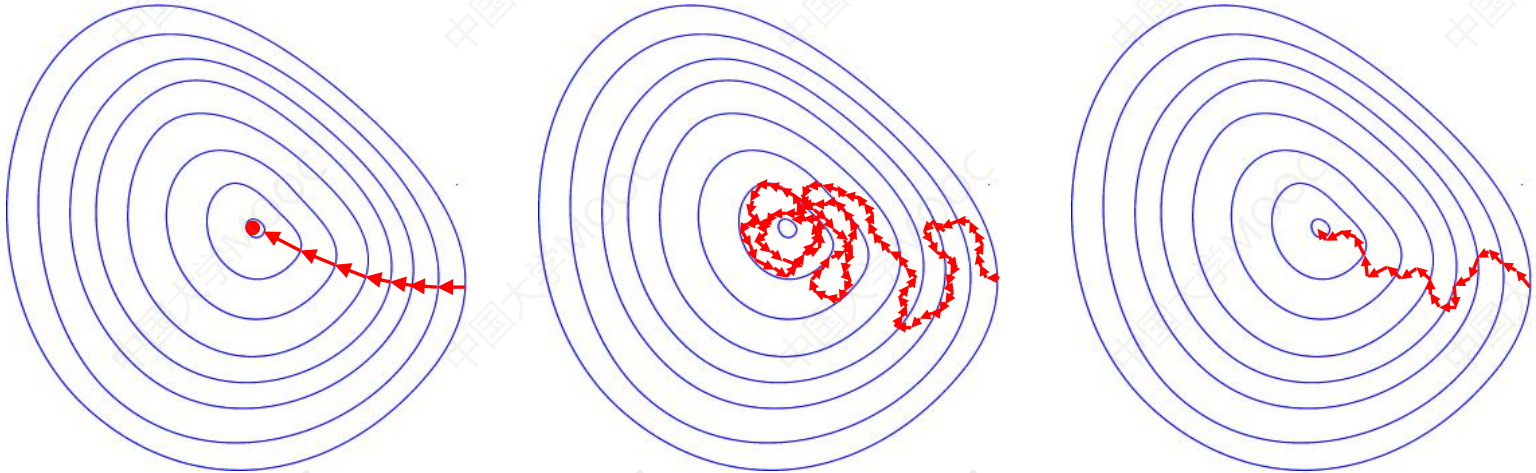
- 小批量样本计算出的**梯度**和使用全体样本计算出的**标准梯度**之间存在偏差
- 总体向最优化的方向前进
- 提高模型的**泛化能力**

损失函数： z=f(x,y)





损失函数： z=f(x,y)



13.2 梯度下降法的优化

梯度下降法：求解函数极值的方法

- 函数在某一点的**梯度**：**向量**
- **梯度方向**：函数值**变化最快**的方向

二元函数

$$\nabla = \begin{pmatrix} \frac{\partial f(w,b)}{\partial w} \\ \frac{\partial f(w,b)}{\partial b} \end{pmatrix}$$

多元函数

$$\nabla Loss(W) = \begin{pmatrix} \frac{\partial Loss(W)}{\partial w_0} \\ \frac{\partial Loss(W)}{\partial w_1} \\ \cdots \\ \frac{\partial Loss(W)}{\partial w_n} \end{pmatrix}$$

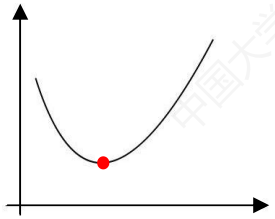
- 沿着**损失函数梯度方向**更新权值

$$W = W - \eta \nabla Loss(W)$$

$$W^{(k+1)} = W^{(k)} - \eta \nabla Loss(W^{(k)})$$

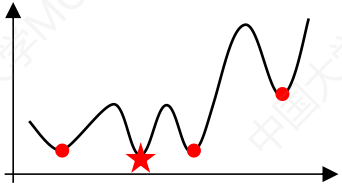
- **梯度为零**时，停止迭代

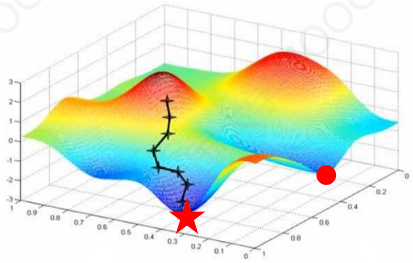
驻点：导数为0的点

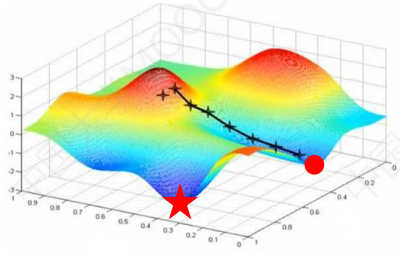


非凸函数

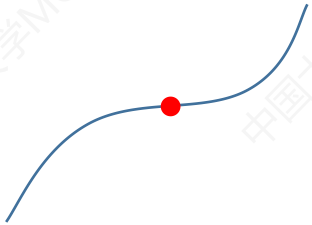
局部极小值点

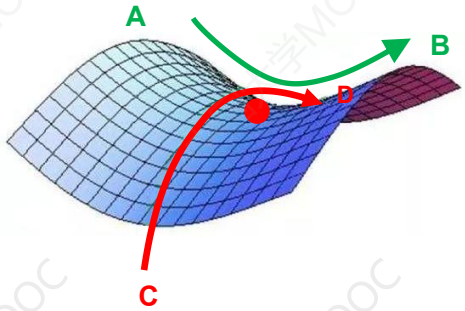


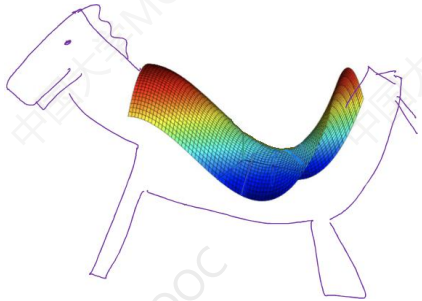




鞍点







多层神经网络使用梯度下降法，**无法保证一定可以收敛于全局最小值点**

口 小批量梯度下降法

一元线性回归

$$Loss = \frac{1}{2}\sum_{i=1}^{t}(y_i - \hat{y}_i)^2 = \frac{1}{2}\sum_{i=1}^{t}(y_i - (wx_i + b))^2$$

$$w^{(k+1)} = w^{(k)} - \eta \frac{\partial Loss(w^{(k)}, b^{(k)})}{w^{(k)}}$$

$$b^{(k+1)} = b^{(k)} - \eta \frac{\partial Loss(w^{(k)}, b^{(k)})}{\partial b^{(k)}}$$

影响小批量梯度下降法的主要因素

- 小批量样本的选择
- 批量大小
- 学习率
- 梯度

小批量样本的选择

$$Loss = \frac{1}{2}\sum_{i=1}^{t}(y_i - \hat{y}_i)^2 = \frac{1}{2}\sum_{i=1}^{t}(y_i - (wx_i + b))^2$$

在**每轮**训练之前，**打乱样本顺序**

数据集的**连续的样本之间有高度的相关性**

	WBC	RBC	HGB	HCT	MCV	MCH	MCHC	PLT	LYMPHF	NEUTP	MONOP
1	7.3	4.7	151.0	45.6	92.6	31.7	342.0	215.0	32.1	57.2	9.1
2	9.0	4.0	151.8	44.0	8.5	30.6	356.6	266.9	32.6	5.7	5.1
3	6.5	4.0	153.9	50.5	9.3	33.0	337.4	207.7	33.9	5.2	4.9
4	7.7	4.9	150.5	41.8	8.6	32.6	343.7	273.9	34.6	6.3	5.3
5	7.3	4.0	149.1	42.7	8.6	31.6	350.7	221.0	34.2	6.4	4.7
6	7.5	4.6	151.1	45.1	9.2	32.4	363.8	289.1	31.6	5.2	5.3
7	8.8	4.2	145.7	47.3	8.7	31.6	357.4	262.7	31.5	5.7	5.5
8	7.8	5.0	151.7	44.1	8.6	30.9	354.0	284.2	31.3	5.9	5.9
9	8.7	4.1	146.4	44.0	8.8	31.1	354.5	277.4	35.0	6.2	6.4
10	8.2	4.8	149.0	49.1	8.6	32.7	340.0	219.4	31.9	6.2	4.9
11	8.2	4.2	153.8	43.4	8.8	31.4	363.3	255.9	31.1	5.6	5.5
12	8.4	4.7	150.1	48.3	8.8	32.6	352.7	261.5	29.9	6.1	4.7
13	9.0	4.7	145.1	45.2	8.9	31.5	345.2	294.8	34.2	5.4	5.7
14	7.9	4.7	148.8	41.1	9.4	30.1	352.0	256.0	30.6	5.5	5.1
15	8.5	4.0	152.3	41.6	8.5	32.2	353.0	227.5	29.5	5.8	5.8
16	8.0	4.8	150.3	48.3	9.0	29.7	344.7	223.1	33.6	5.7	6.1
17	8.6	4.0	145.4	46.8	8.6	29.2	360.3	287.7	28.5	5.2	5.6
18	8.6	4.2	145.1	40.7	8.5	30.5	344.1	247.0	31.8	5.9	6.4
19	7.7	4.3	150.6	47.5	9.3	29.1	361.8	212.1	32.2	5.8	6.2
20	7.9	4.1	145.2	51.1	9.1	31.8	346.8	259.8	33.3	5.3	5.6
21	7.0	5.0	153.5	42.6	8.6	30.4	357.0	294.4	31.6	5.2	5.7
22	7.5	4.1	146.4	46.9	8.5	29.1	342.9	294.9	28.9	5.3	4.5
23	8.2	4.8	145.9	40.8	8.6	30.9	339.8	292.1	29.7	6.5	6.5
24	8.2	4.2	147.5	44.1	9.3	32.8	339.9	200.6	28.7	6.0	6.1
25	8.6	4.1	148.3	51.1	8.9	31.5	338.6	202.4	28.5	6.5	5.9
26	6.7	4.8	148.9	46.2	9.4	30.4	344.1	242.4	29.4	5.2	5.1
27	7.2	4.5	145.8	49.8	8.5	29.1	358.3	218.4	30.0	5.4	5.7
28	8.4	4.2	151.2	48.1	9.0	31.1	342.8	200.5	34.3	5.4	4.6
29	6.5	4.6	146.3	40.8	8.5	29.9	362.6	224.3	29.2	5.8	5.7
30	6.7	4.2	150.1	40.2	8.5	32.6	364.3	244.9	32.5	6.2	5.4
31	7.1	4.0	147.6	40.2	8.7	32.3	345.7	281.0	29.8	6.5	5.3
32	6.8	4.2	148.1	49.8	8.8	29.0	340.7	219.1	29.4	5.5	5.2
33	6.9	4.0	149.2	42.7	9.1	32.1	345.9	223.8	32.3	6.3	4.8
34	9.0	5.0	145.3	45.8	8.9	32.8	354.9	216.1	29.7	6.4	5.2
35	8.8	4.7	147.8	43.3	8.7	31.7	356.6	237.0	31.0	6.1	5.5
36	6.5	4.2	152.5	45.5	8.7	29.7	359.3	234.3	34.9	5.7	4.7
37	7.0	4.3	150.3	50.1	8.8	32.2	342.2	272.3	28.5	6.3	6.1
38	7.6	4.0	146.4	43.5	9.4	29.2	338.9	292.7	30.8	5.3	4.6
39	6.9	4.3	147.6	50.1	8.6	31.1	359.3	280.0	28.1	6.1	4.7

- **小批量中样本的数量**（mini-batch size）
  - 批量中的**样本数量越多**，**梯度方向越准确**，迭代次数越少

    批量梯度下降法：小批量样本=整个数据集
  - 批量中的**样本数量越少**，**随机性越大**，迭代次数越多

    随机梯度下降法：小批量样本数=1
  - 充分利用处理器资源，进行**并行计算**

    2的幂数：32、64、128、256

学习率

$$w^{(k+1)} = w^{(k)} - \eta \frac{\partial Loss(w^{(k)}, b^{(k)})}{w^{(k)}}$$

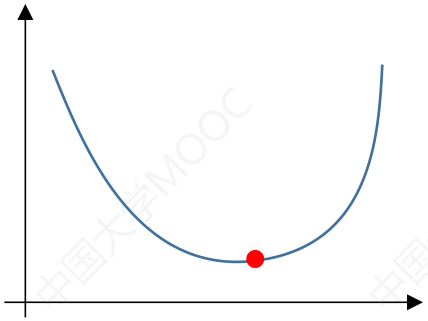
$$b^{(k+1)} = b^{(k)} - \eta \frac{\partial Loss(w^{(k)}, b^{(k)})}{\partial b^{(k)}}$$

固定学习率

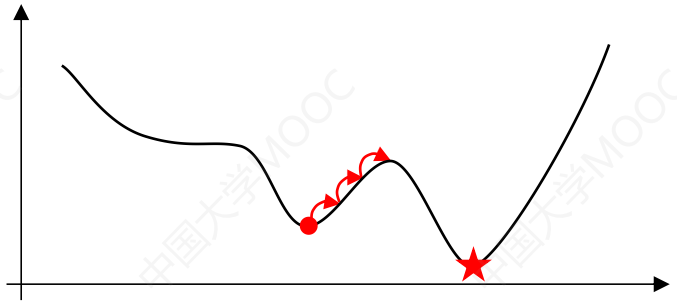
学习率设置**过小**，收敛速度慢；
学习率设置**过大**，震荡，无法收敛

动态调整学习率：在训练过程中，动态的调整学习率

- **学习率衰减**（Learning Rate Decay）/**学习率退火**（Learning Rate Annealing）
  - **开始训练**时，设置**较大**学习率，加快收敛速度
  - **迭代过程**中，学习率随着迭代次数**逐渐减小**，避免震荡



凸函数



非凸函数

- **调节学习率**——非凸函数
  - **周期性**的增大学习率
  - **自适应**调整学习率
  - 自适应调整**每个参数**的学习率

$$w^{(k+1)} = w^{(k)} - \eta \frac{\partial Loss(w^{(k)}, b^{(k)})}{w^{(k)}}$$

$$b^{(k+1)} = b^{(k)} - \eta \frac{\partial Loss(w^{(k)}, b^{(k)})}{\partial b^{(k)}}$$

- **自适应学习率算法**
  - **AdaGrad**
    - 全局学习率 η
    - 各个**参数自适应**的调整学习率 $\frac{\eta}{\sqrt{\sum_{r=1}^{t}(grad_r)^2 + \varepsilon}}$
    - 学习率随着迭代次数的增加而**单调减小**
  - **RMSprop**、和**AdaDelta**
    - 改进了AdaGrad算法**调整学习率**的方法
    - 各参数的学习率**不是单调递减**的

梯度估计

$$w^{(k+1)} = w^{(k)} - \eta \frac{\partial Loss(w^{(k)}, b^{(k)})}{w^{(k)}}$$

$$b^{(k+1)} = b^{(k)} - \eta \frac{\partial Loss(w^{(k)}, b^{(k)})}{\partial b^{(k)}}$$

当前梯度
历史梯度 → 参数更新

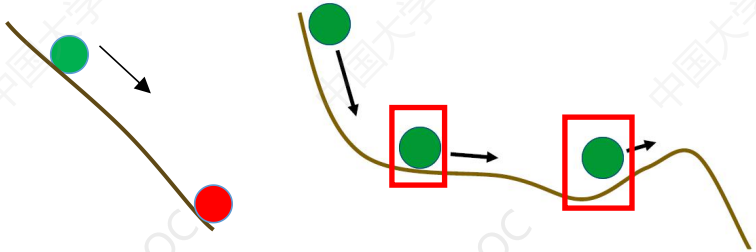
动量梯度下降法（Momentum）

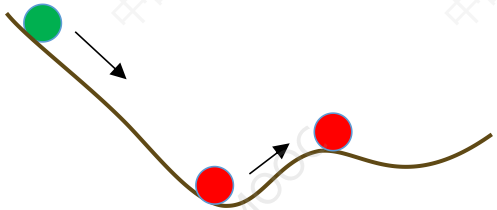
动量=质量×速度

在更新参数时，可以在一定程度上**保留之前的更新方向**

$$\begin{cases} W^{(k+1)} = W^{(k)} + v^{(k+1)} \\ v^{(k+1)} = \alpha v^{(k)} - \eta \nabla Loss\,(W^{(k+1)}, X^{(i)}, Y^{(i)}) \end{cases}$$

梯度方向不变：步长更大，收敛加快
梯度方向改变：步长变小，更新变慢



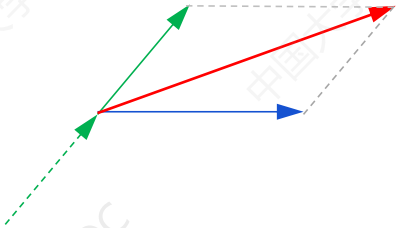


牛顿加速梯度算法（Nesterov Accelerated Gradient，NAG）

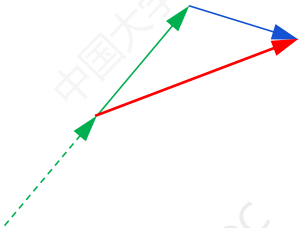
- 根据当前更新方向，估算下一步的梯度方向
- 在新位置计算梯度，修正梯度方向

$$\begin{cases} W^{(k+1)} = W^{(k)} + v^{(k+1)} \\ v^{(k+1)} = \alpha v^{(k)} - \eta \nabla Loss(W^{(k+1)} - \mu v^{(k)}, X^{(i)}, Y^{(i)}) \end{cases}$$

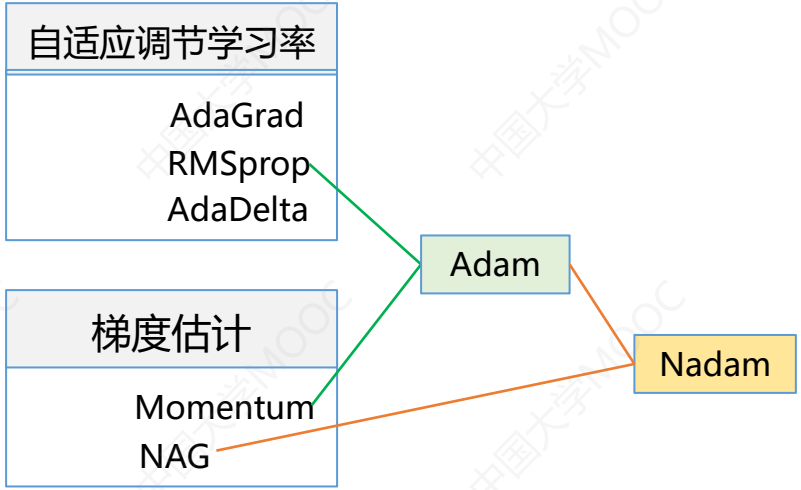
Momentum

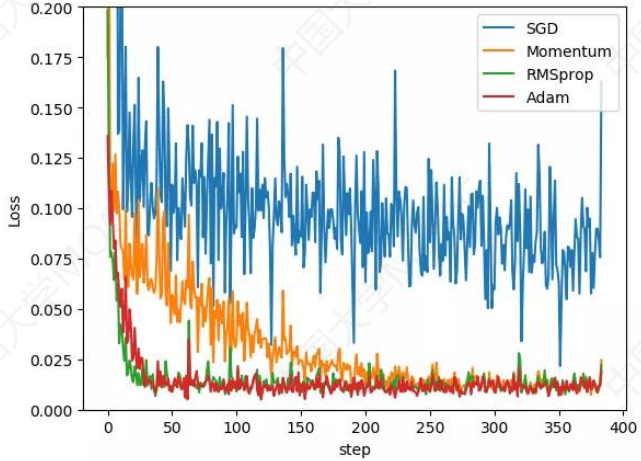


NAG



自适应学习率+梯度估计


自适应调节学习率
AdaGrad
RMSprop
AdaDelta
梯度估计
Momentum
NAG
Adam
Nadam

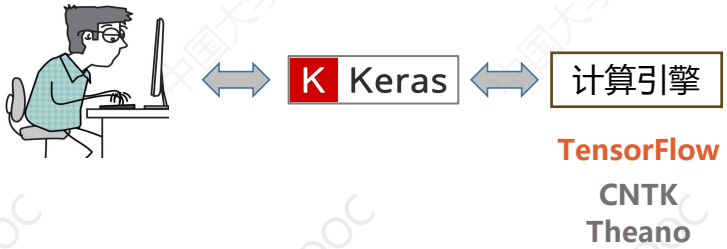

SGD
Momentum
RMSprop
Adam
Loss
step
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0
50
100
150
200
250
300
350
400

13.3 keras与tf.keras

Keras

由Python编写的**开源人工神经网络库**

- 面向对象，完全**模块化**
- 支持**神经网络**和**深度学习**的主流算法
- 支持**多操作系统**下的**多GPU并行计算**
- 作为深度学习库的**前端**



Keras with TensorFlow Backend

- 安装Keras
- TensorFlow作为**依赖项**自动安装

tf.keras：TensorFlow的**高阶API**

- 安装tensorflow
- tf.keras作为**子模块**被自动安装

TensorFlow 2.0 + tf.keras

- 解决了1.x版本中存在的问题
- 大量的优化和扩展

TensorFlow具备Keras的**简单性**
Keras具备TensorFlow的**强大性**

13.4 Sequential模型

tf.keras

- TensorFlow的**高阶API**
- 快速**搭建**和**训练**神经网络模型
- 主要数据结构是**模型**(model)

Sequential模型

- 神经网络框架
- 只有**一组输入**和**一组输出**
- 各个**层**按照先后顺序**堆叠**

建立Sequential模型

```
model=tf.keras.Sequential()
```

```
model
<tensorflow.python.keras.engine.sequential.Sequential at 0x2cfb865f4e0>
```

添加层

全连接层，卷积层，池化层.....

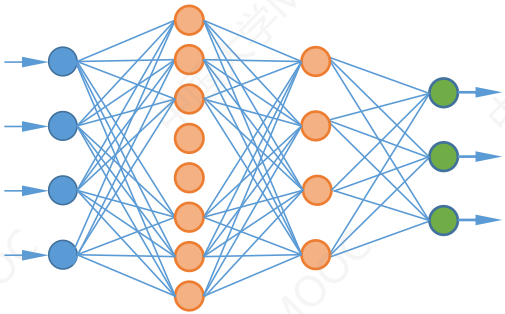
```
model.add (tf.keras.layers.... )
```

```
tf.keras.layers.Dense(
    inputs # 输入该网络层的数据
    activation # 激活函数  'relu', 'softmax','sigmoid','tanh'
    input_shape #输入数据的形状 )
```

查看摘要

```
model.summary()
```

多分类任务——三层神经网络




```
model=tf.keras.Sequential()
model.add(tf.keras.layers.Dense(8,activation="relu",input_shape=(4,)))
model.add(tf.keras.layers.Dense(4,activation="relu"))
model.add(tf.keras.layers.Dense(3,activation="softmax"))
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 8)                 40
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 36
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 15
=================================================================
Total params: 91
Trainable params: 91
Non-trainable params: 0
```

隐含层1

隐含层2

输出层

配置训练方法

model.compile(loss, optimizer, metrics)

损失函数

均方差损失函数	'mse'	tf.keras.losses.mean_squared_error() tf.keras.losses.MeanSquaredError()
多分类交叉熵损失函数	'categorical_crossentropy' 'sparse_categorical_crossentropy'	tf.keras.losses.categorical_crossentropy(from_logits=False) tf.keras.losses.sparse_categorical_crossentropy(from_logits=False) tf.keras.losses.CategoricalCrossentropy() tf.keras.losses.SparseCategoricalCrossentropy()
二分类交叉熵损失函数	'binary_crossentropy'	tf.keras.losses.binary_crossentropy() tf.keras.losses.BinaryCrossentropy()

配置训练方法

model.compile(loss, optimizer, metrics)

优化器

'sgd'	tf.keras.optimizers.SGD (lr, momentum)
'adagrad'	tf.keras.optimizers.Adagrad (lr)
'adadelta'	tf.keras.optimizers.Adadelta (lr)
'adam'	tf.keras.optimizers.Adam (lr, beta_1=0.9, beta_2=0.999)

tensorflow1.x tf.train.optimizer

tensorflow2.0+ tf.keras.optimizer

配置训练方法

- keras模型性能评估函数
- 自定义性能评估函数

```
model.compile(loss, optimizer, metrics)
```

标签值：数值 预测值：数值	'accuracy'	tf.keras.metrics.Accuracy()
二分类 标签值：数值 预测值：概率	'binary_accuracy '	tf.keras.metrics.binary_accuracy(threshold=0.5) tf.keras.metrics.BinaryAccuracy(threshold=0.5)
多分类 标签值：独热编码 预测值：独热编码	'categorical_accuracy'	tf.keras.metrics.categorical_accuracy() tf.keras.metrics.CategoricalAccuracy()
多分类 标签值：数值 预测值：独热编码	'sparse_categorical_accuracy'	tf.keras.metrics.sparse_categorical_accuracy() tf.keras.metrics.SparseCategoricalAccuracy()
多分类 前k种标签 标签值：数值 预测值：独热编码	'top_k_categorical_accuracy'	tf.keras.metrics.top_k_categorical_accuracy() tf.keras.metrics.TopKCategoricalAccuracy()
多分类 前k种标签 标签值：数值 预测值：独热编码	'sparse_top_k_categorical_accuracy'	tf.keras.metrics.sparse_top_k_categorical_accuracy() tf.keras.metrics.SparseTopKCategoricalAccuracy()

配置训练方法

model.compile(loss, optimizer, metrics)

AUC SUM Mean Precision	BinaryCrossentropy CategoricalCrossentropy SparseCategoricalCrossentropy Hinge CategoricalHinge	MeanAbsoluteError MeanAbsolutePercentageError MeanRelativeError MeanSquareError .....

https://tensorflow.google.cn/versions/r2.0/api_docs/python/tf/keras/metrics/

Iris

```
model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.1),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              metrics=[tf.keras.metrics.SparseCategoricalAccuracy()])
```

IRIS

标签值： 1 ⟶ **[0, 1, 0]**

预测值：[0.1, 0.7, 0.1]

loss=tf.keras.losses.CategoricalCrossentropy(from_logits=False),
metrics=[tf.keras.metrics.CategoricalAccuracy()])

Mnist

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['sparse_categorical_accuracy'])
```

训练模型

```
model.fit (训练集的输入特征, 训练集的标签,
          batch_size=批量大小 ,
          epochs= 迭代次数,
        shuffle=是否每轮训练之前打乱数据,
          validation_data=(测试集的输入特征，测试集的标签),
          validation_split=从训练集划分多少比例给测试集,
          validation_freq = 测试频率
        verbose=日志显示形式
        )
```

```
verbose = 0   不在标准输出流输出
verbose = 1   输出进度条记录
verbose = 2   每个epoch输出一行记录
```

```
model.fit (x=None, y=None, batch_size=32 , epochs= 1,  shuffle=True,
          validation_data=None,  validation_split=0.0,  validation_freq = 1,
        verbose=1)
```

手写数字识别

```
model.fit(train_x, train_y, batch_size=32,epochs=5,validation_split=0.2) validation_freq = 1
Train on 48000 samples, validate on 12000 samples
Epoch 1/5
48000/48000 [==============================] - 4s 75us/sample - loss: 0.2813 - accuracy: 0.9198 - val_loss: 0.1512 - val_accuracy: 0.9562
Epoch 2/5
48000/48000 [==============================] - 3s 61us/sample - loss: 0.1239 - accuracy: 0.9632 - val_loss: 0.1179 - val_accuracy: 0.9639
Epoch 3/5
48000/48000 [==============================] - 3s 67us/sample - loss: 0.0880 - accuracy: 0.9740 - val_loss: 0.1008 - val_accuracy: 0.9688
Epoch 4/5
48000/48000 [==============================] - 3s 66us/sample - loss: 0.0648 - accuracy: 0.9806 - val_loss: 0.0950 - val_accuracy: 0.9713
Epoch 5/5
48000/48000 [==============================] - 3s 62us/sample - loss: 0.0510 - accuracy: 0.9849 - val_loss: 0.0915 - val_accuracy: 0.9726
<tensorflow.python.keras.callbacks.History at 0x13de91f7748>
```

History.history 损失函数 + **性能指标**
compile.metrics

```
In [20]: print(model.metrics_names)
['loss', 'accuracy']
```

■ 手写数字识别

```
model.fit(X_train, y_train, batch_size=32, epochs=5, validation_split=0.2)
```

validation_freq = 1

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/5
48000/48000 [==============================] - 5s 103us/sample - loss: 0.2846 - sparse_categorical_accuracy: 0.9183 - val_loss: 0.1582 - val
_sparse_categorical_accuracy: 0.9533
Epoch 2/5
48000/48000 [==============================] - 3s 72us/sample - loss: 0.1272 - sparse_categorical_accuracy: 0.9624 - val_loss: 0.1199 - val_
sparse_categorical_accuracy: 0.9646
Epoch 3/5
48000/48000 [==============================] - 3s 67us/sample - loss: 0.0860 - sparse_categorical_accuracy: 0.9743 - val_loss: 0.1020 - val_
sparse_categorical_accuracy: 0.9712
Epoch 4/5
48000/48000 [==============================] - 3s 66us/sample - loss: 0.0632 - sparse_categorical_accuracy: 0.9804 - val_loss: 0.1007 - val_
sparse_categorical_accuracy: 0.9701
Epoch 5/5
48000/48000 [==============================] - 3s 67us/sample - loss: 0.0486 - sparse_categorical_accuracy: 0.9851 - val_loss: 0.0901 - val_
sparse_categorical_accuracy: 0.9748

<tensorflow.python.keras.callbacks.History at 0x1cc5e7f5080>
```

```
model.metrics_names
```

model.compile(metrics=[ ])

```
['loss', 'sparse_categorical_accuracy']
```

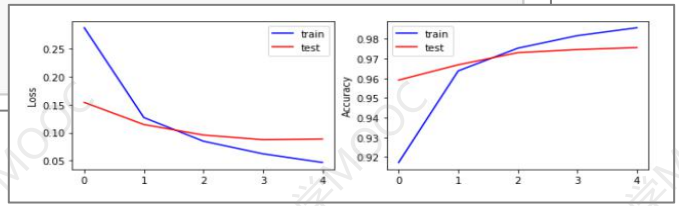
■ 手写数字识别

```
history.history
```

```
{'loss': [0.2865329485485951,
  0.12679306019532183,
  0.0847434730535994,
  0.06226444375965123,
  0.046863987305822474],
 'sparse_categorical_accuracy': [0.91714585,
  0.9636667,
  0.9751875,
  0.9815625,
  0.9855625],
 'val_loss': [0.15371777984748283,
  0.1143894852194935,
  0.09567862997855991,
  0.08741304606727014,
  0.0886051772281838],
 'val_sparse_categorical_accuracy': [0.959,
  0.96675,
  0.97291666,
  0.9745,
  0.9755]}
```

```
loss = history.history['loss']
val_loss = history.history['val_loss']
acc = history.history['sparse_categorical_accuracy']
val_acc = history.history['val_sparse_categorical_accuracy']
```

```
plt.figure(figsize=(10,3))

plt.subplot(121)
plt.plot(loss,color="blue",label="train")
plt.plot(val_loss,color="red",label="test")
plt.ylabel("Loss")
plt.legend()

plt.subplot(122)
plt.plot(acc,color="blue",label="train")
plt.plot(val_acc,color="red",label="test")
plt.ylabel("Accuracy")

plt.legend()
plt.show()
```



手写数字识别

```
history=model.fit(X_train, y_train, batch_size=32,epochs=5,validation_split=0.2)
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/5
48000/48000 [==============================] - 5s 98us/sample - loss: 0.2865 - sparse_categorical_accuracy: 0.9171 - val_loss: 0.1537 - val_
sparse_categorical_accuracy: 0.9590
Epoch 2/5
48000/48000 [==============================] - 4s 84us/sample - loss: 0.1268 - sparse_categorical_accuracy: 0.9637 - val_loss: 0.1144 - val_
sparse_categorical_accuracy: 0.9668
Epoch 3/5
48000/48000 [==============================] - 4s 86us/sample - loss: 0.0847 - sparse_categorical_accuracy: 0.9752 - val_loss: 0.0957 - val_
sparse_categorical_accuracy: 0.9729
Epoch 4/5
48000/48000 [==============================] - 4s 91us/sample - loss: 0.0623 - sparse_categorical_accuracy: 0.9816 - val_loss: 0.0874 - val_
sparse_categorical_accuracy: 0.9745
Epoch 5/5
48000/48000 [==============================] - 4s 84us/sample - loss: 0.0469 - sparse_categorical_accuracy: 0.9856 - val_loss: 0.0886 - val_
sparse_categorical_accuracy: 0.9755
```

```
type(history)
```

```
tensorflow.python.keras.callbacks.History
```

手写数字识别

```
model.fit(train_x, train_y, batch_size=64,epochs=5) validation_split=0.0

Train on 60000 samples
Epoch 1/5
60000/60000 [==============================] - 4s 60us/sample - loss: 0.3001 - accuracy: 0.9157
Epoch 2/5
60000/60000 [==============================] - 2s 34us/sample - loss: 0.1333 - accuracy: 0.9614
Epoch 3/5
60000/60000 [==============================] - 2s 31us/sample - loss: 0.0921 - accuracy: 0.9730
Epoch 4/5
60000/60000 [==============================] - 2s 34us/sample - loss: 0.0693 - accuracy: 0.9798
Epoch 5/5
60000/60000 [==============================] - 2s 32us/sample - loss: 0.0548 - accuracy: 0.9836

<tensorflow.python.keras.callbacks.History at 0x22d06a91080>
```

评估模型

model.evaluate（test_set_x，test_set_y ,batch_size, verbose）

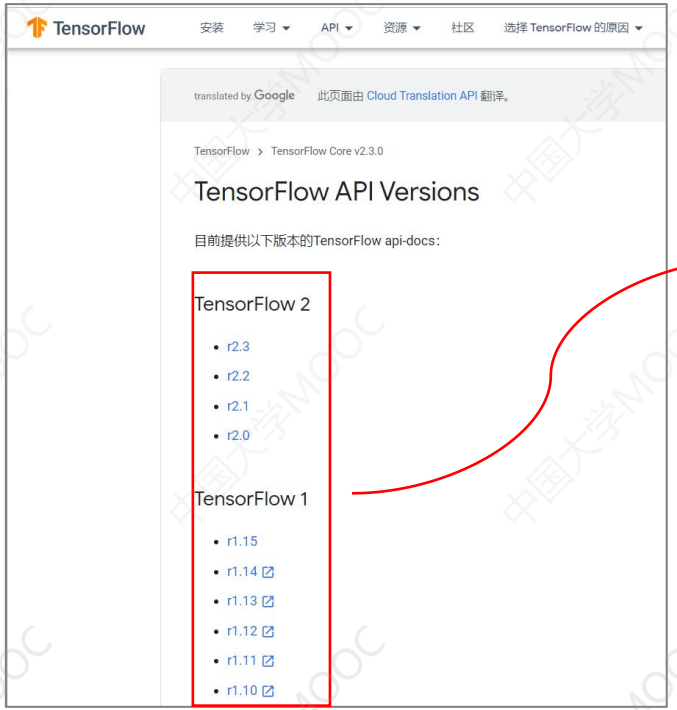
- **Mnist**
  - 训练集：60000
  - 测试集：10000

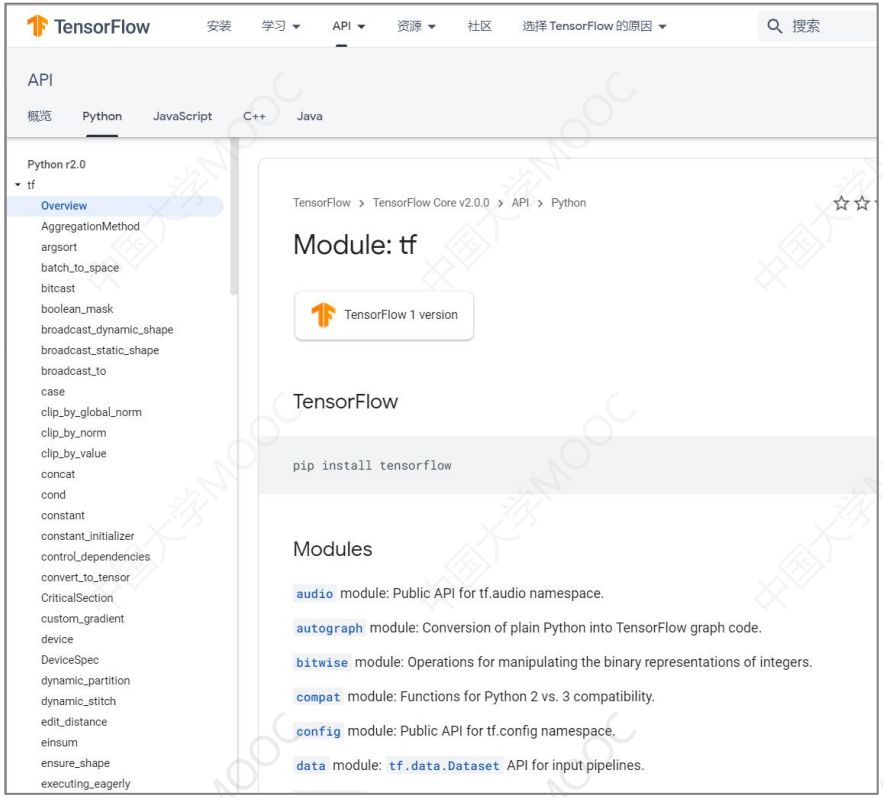
```
In  [17]: model.evaluate(test_x, test_y, batch_size=32, verbose=2)
          10000/1 - 0s - loss: 0.0427 - accuracy: 0.9722
Out[17]: [0.08461135778911412, 0.9722]
```

使用模型

model.predict（x, batch_size, verbose）

https://tensorflow.google.cn/versions





13.5 实例：Sequential模型实现手写数字识别

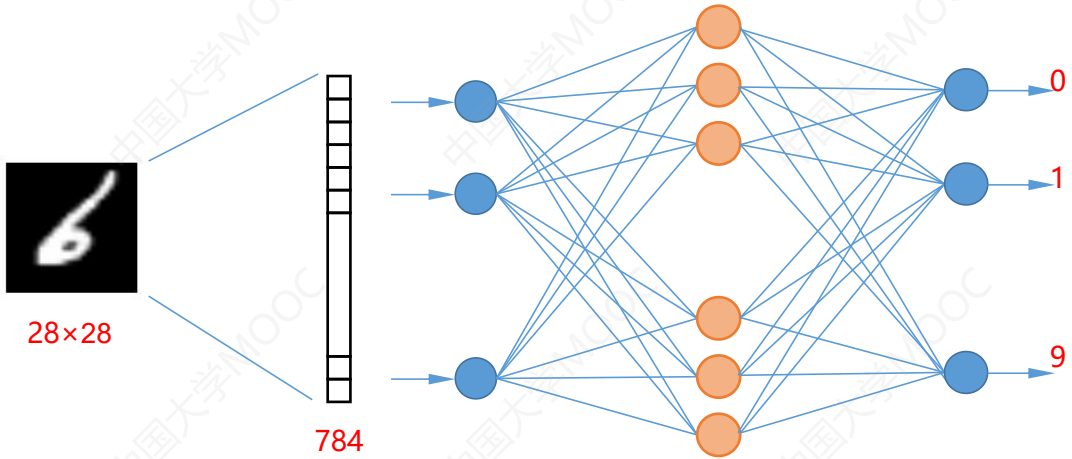
Sequential模型

- 只有**一组输入**和**一组输出**
- 各个**层**按照先后顺序**堆叠**

- 建立模型
  ```
  model=tf.keras.Sequential( )
  model.add( )
  ```
- 查看摘要
  ```
  model.summary( )
  ```
- 配置训练方法
  ```
  model.compile( )
  ```
- 训练模型
  ```
  model.fit( )
  ```
- 评估模型
  ```
  model.evaluate( )
  ```
- 使用模型
  ```
  model.predict( )
  ```

手写数字数据集MNIST

设计神经网络结构



784×128+128=10048

128×10+10=1290

0

100480+1290=**101770**

标签值：0~9

损失函数：

SparseCategoricalCrossentropy

导入库

```
In [1]: import tensorflow as tf
        tf.__version__, tf.keras.__version__

Out[1]: ('2.0.0', '2.2.4-tf')

In [2]: import numpy as np
        import matplotlib.pyplot as plt

In [3]: gpus = tf.config.experimental.list_physical_devices('GPU')
        tf.config.experimental.set_memory_growth(gpus[0], True)
```

加载数据

```
In [4]: mnist=tf.keras.datasets.mnist
        (train_x,train_y),(test_x,test_y)=mnist.load_data()

In [5]: print(train_x.shape)
        print(train_y.shape)
        print(test_x.shape)
        print(test_y.shape)

        (60000, 28, 28)
        (60000,)
        (10000, 28, 28)
        (10000,)

In [6]: type(train_x),type(train_y)
Out[6]: (numpy.ndarray, numpy.ndarray)

In [7]: type(test_x),type(test_y)
Out[7]: (numpy.ndarray, numpy.ndarray)

In [8]: train_x.min(),train_x.max()
Out[8]: (0, 255)

In [9]: train_y.min(),train_y.max()
Out[9]: (0, 9)
```

数据预处理

```
In [10]: X_train=train_x.reshape((60000, 28*28))
         X_test=test_x.reshape((10000, 28*28))
```

tf.keras.layers.Flatten( )

```
In [11]: print(X_train.shape)
         print(X_test.shape)

         (60000, 784)
         (10000, 784)
```

```
In [10]: X_train, X_test=tf.cast(train_x/255.0, tf.float32), tf.cast(test_x/255.0, tf.float32)
         y_train, y_test=tf.cast(train_y, tf.int16), tf.cast(test_y, tf.int16)

In [11]: type(X_train), type(y_train)

Out[11]: (tensorflow.python.framework.ops.EagerTensor,
          tensorflow.python.framework.ops.EagerTensor)
```

- 建立模型

```
In [12]: model=tf.keras.Sequential()
         model.add(tf.keras.layers.Flatten(input_shape=(28,28)))
         model.add(tf.keras.layers.Dense(128,activation="relu"))
         model.add(tf.keras.layers.Dense(10,activation="softmax"))

In [13]: model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 128)               100480
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

配置训练方法

```
In [14]: model.compile(optimizer='adam',
                       loss='sparse_categorical_crossentropy',
                       metrics=['sparse_categorical_accuracy'])
```

标签值：0~9

预测值：概率分布

训练模型

```
In [15]: model.fit(X_train, y_train, batch_size=64,epochs=5,validation_split=0.2)

         Train on 48000 samples, validate on 12000 samples
         Epoch 1/5
         48000/48000 [==============================] - 2s 45us/sample - loss: 0.3374 - sparse_categorical_accuracy:
         0.9052 - val_loss: 0.1874 - val_sparse_categorical_accuracy: 0.9473
         Epoch 2/5
         48000/48000 [==============================] - 2s 33us/sample - loss: 0.1601 - sparse_categorical_accuracy:
         0.9536 - val_loss: 0.1367 - val_sparse_categorical_accuracy: 0.9616
         Epoch 3/5
         48000/48000 [==============================] - 2s 35us/sample - loss: 0.1111 - sparse_categorical_accuracy:
         0.9676 - val_loss: 0.1162 - val_sparse_categorical_accuracy: 0.9643
         Epoch 4/5
         48000/48000 [==============================] - 2s 34us/sample - loss: 0.0843 - sparse_categorical_accuracy:
         0.9756 - val_loss: 0.1018 - val_sparse_categorical_accuracy: 0.9696
         Epoch 5/5
         48000/48000 [==============================] - 1s 31us/sample - loss: 0.0661 - sparse_categorical_accuracy:
         0.9806 - val_loss: 0.0930 - val_sparse_categorical_accuracy: 0.9722
Out[15]: <tensorflow.python.keras.callbacks.History at 0x26e31b2d9b0>
```

评估模型

```
In  [16]: model.evaluate(X_test,  y_test,  verbose=2)
          10000/1 - 0s - loss: 0.0446 - sparse_categorical_accuracy: 0.9730
Out[16]: [0.0874457276863046, 0.973]
```

- 使用模型

```
In [17]: plt.axis("off")
         plt.imshow(test_x[0],cmap="gray")
         plt.show()
```

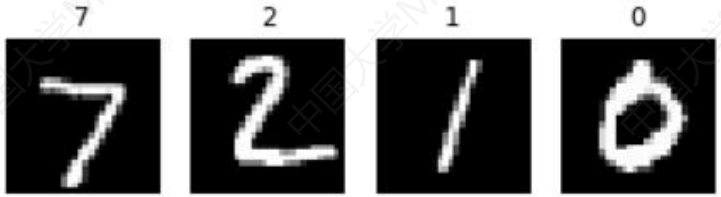
```
In [18]: y_test[0]
Out[18]: <tf.Tensor: id=15906, shape=(), dtype=int16, numpy=7>
```

使用模型

```
In  [19]: model.predict([[X_test[0]]])

Out[19]: array([[1.5193305e-07, 1.4296833e-07, 5.6485705e-06, 6.1900186e-04,
                 3.7725050e-09, 6.8479079e-07, 1.1266648e-10, 9.9935585e-01,
                 9.8319993e-07, 1.7526505e-05]], dtype=float32)

In  [20]: np.argmax(model.predict([[X_test[0]]]))

Out[20]: 7
```

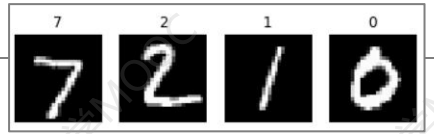
- 使用模型——测试集中**前4个**数据

```
In [21]: for i in range(4):

             plt.subplot(1,4,i+1)
             plt.axis("off")
             plt.imshow(test_x[i],cmap='gray')
             plt.title(test_y[i])

         plt.show()
```



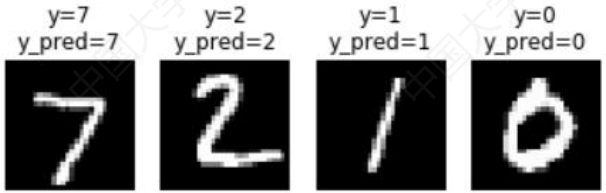
- 使用模型——测试集中**前4个**数据

```
In [22]: model.predict(X_test[0:4])

Out[22]: array([[1.51932895e-07, 1.42968190e-07, 5.64855463e-06, 6.19000697e-04,
                 3.77249032e-09, 6.84789484e-07, 1.12666480e-10, 9.99355853e-01,
                 9.83197992e-07, 1.75264722e-05],
                [2.39225937e-08, 1.46651606e-03, 9.98372793e-01, 1.20054741e-04,
                 2.44312660e-11, 1.68191150e-06, 2.73193677e-06, 3.58970839e-11,
                 3.62173996e-05, 5.76994064e-10],
                [8.24031304e-06, 9.95415926e-01, 4.08277119e-04, 2.54331389e-04,
                 5.98004612e-04, 5.25890682e-05, 4.77820053e-04, 1.27647584e-03,
                 1.49090553e-03, 1.73707685e-05],
                [9.99874830e-01, 1.23330821e-08, 4.77576832e-05, 1.29433559e-07,
                 4.56634234e-06, 1.37567758e-06, 3.26714071e-05, 2.61713703e-05,
                 1.33380125e-08, 1.23481877e-05]], dtype=float32)

In [23]: np.argmax(model.predict(X_test[0:4]),axis=1)

Out[23]: array([7, 2, 1, 0], dtype=int64)
```



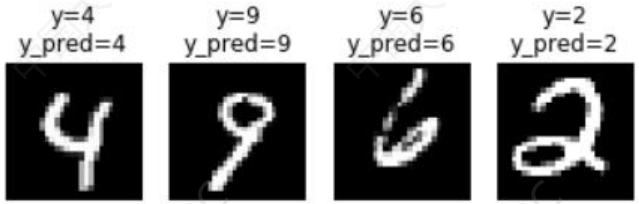
- 使用模型——测试集中**前4个**数据

```
In [24]: y_pred=np.argmax(model.predict(X_test[0:4]),axis=1)

In [25]: for i in range(4):

             plt.subplot(1,4,i+1)
             plt.axis("off")
             plt.imshow(test_x[i],cmap='gray')
             plt.title("y="+str(test_y[i])+"\ny_pred="+str(y_pred[i]))

         plt.show()
```



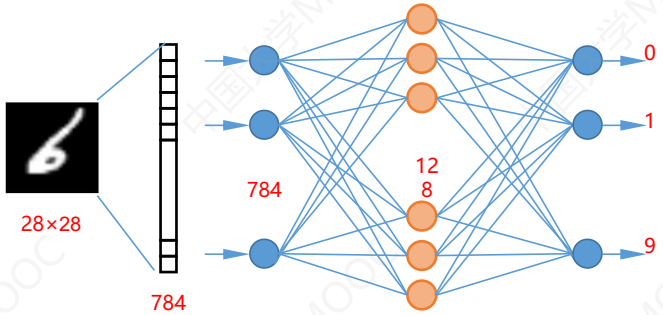
- 使用模型——测试集中**随机**取4个数据

In [26]:
```
for i in range(4):
    num = np.random.randint(1,10000)

    plt.subplot(1,4,i+1)
    plt.axis("off")
    plt.imshow(test_x[num],cmap='gray')
    y_pred=np.argmax(model.predict([[X_test[num]]]))
    title="y="+str(test_y[num])+"\ny_pred="+str(y_pred)
    plt.title(title)

plt.show()
```



13.6 实例：模型的保存和加载

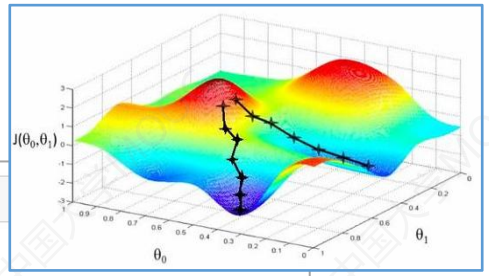
实例：使用Sequential模型实现手写数字识别




```
model=tf.keras.Sequential()
model.add(tf.keras.layers.Flatten(input_shape=(28,28)))
model.add(tf.keras.layers.Dense(128,activation="relu"))
model.add(tf.keras.layers.Dense(10,activation="softmax"))
```

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['sparse_categorical_accuracy'])
```

训练模型




```
In [15]: model.fit(X_train, y_train, batch_size=64,epochs=5,validation_split=0.2)

         Train on 48000 samples, validate on 12000 samples
         Epoch 1/5
         48000/48000 [==============================] - 2s 45us/sample - loss: 0.3374 - sparse_categorical_accuracy:
         0.9052 - val_loss: 0.1874 - val_sparse_categorical_accuracy: 0.9473
         Epoch 2/5
         48000/48000 [==============================] - 2s 33us/sample - loss: 0.1601 - sparse_categorical_accuracy:
         0.9536 - val_loss: 0.1367 - val_sparse_categorical_accuracy: 0.9616
         Epoch 3/5
         48000/48000 [==============================] - 2s 35us/sample - loss: 0.1111 - sparse_categorical_accuracy:
         0.9676 - val_loss: 0.1162 - val_sparse_categorical_accuracy: 0.9643
         Epoch 4/5
         48000/48000 [==============================] - 2s 34us/sample - loss: 0.0843 - sparse_categorical_accuracy:
         0.9756 - val_loss: 0.1018 - val_sparse_categorical_accuracy: 0.9696
         Epoch 5/5
         48000/48000 [==============================] - 1s 31us/sample - loss: 0.0661 - sparse_categorical_accuracy:
         0.9806 - val_loss: 0.0930 - val_sparse_categorical_accuracy: 0.9722

Out[15]: <tensorflow.python.keras.callbacks.History at 0x26e31b2d9b0>
```

保存模型参数

```
model.save_weights( filepath,
                    overwrite=True,
                    save_format=None)
```

HDF5格式

.h5 ,.keras

save_format=None

分层数据格式（Hierarchical Data Format）

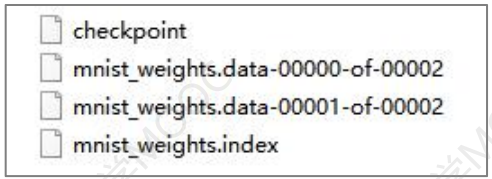
- group
- dataset

SavedModel

save_format= "tf"

TensorFlow序列化文件格式

```
model.save_weights("mnist_weights", save_format="tf")
```



- 保存模型参数

```
model.save_weights( filepath,
                    overwrite=True,
                    save_format=None)
```

```
model.save_weights("mnist_weights.h5",overwrite=False)
[WARNING] mnist_weights.h5 already exists - overwrite? [y/n]
```

- 加载模型参数

```
model.load_weights( filepath)
```

实例： 手写数字识别

In [1]:
```
import tensorflow as tf
tf.__version__, tf.keras.__version__
```

Out[1]:
```
('2.0.0', '2.2.4-tf')
```

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [3]:
```
gpus = tf.config.experimental.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(gpus[0], True)
```

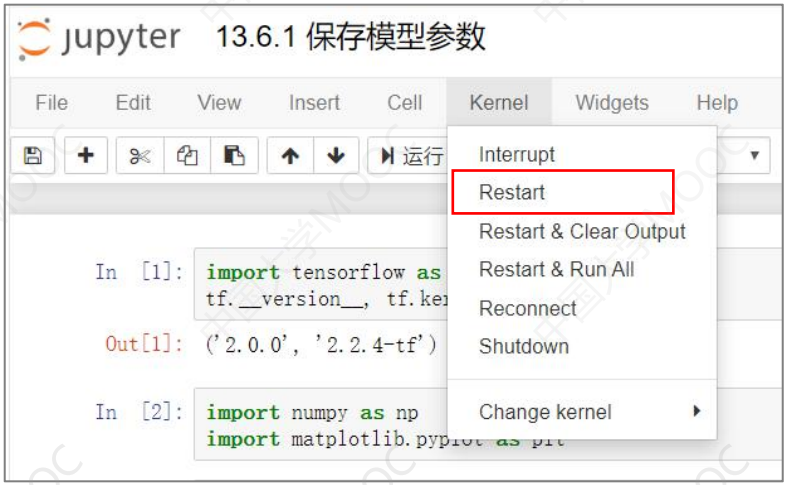
In [4]:
```
mnist=tf.keras.datasets.mnist
(train_x,train_y),(test_x,test_y)=mnist.load_data()
```

In [5]:
```
X_train,X_test=tf.cast(train_x/255.0,tf.float32),tf.cast(test_x/255.0,tf.float32)
y_train,y_test=tf.cast(train_y,tf.int16),tf.cast(test_y,tf.int16)
```

```
In [6]: model=tf.keras.Sequential()
        model.add(tf.keras.layers.Flatten(input_shape=(28,28)))
        model.add(tf.keras.layers.Dense(128,activation="relu"))
        model.add(tf.keras.layers.Dense(10,activation="softmax"))

In [7]: model.compile(optimizer='adam',
                      loss='sparse_categorical_crossentropy',
                      metrics=['sparse_categorical_accuracy'])
```

```
In [8]: model.fit(X_train, y_train, batch_size=64,epochs=5)

        Train on 60000 samples
        Epoch 1/5
        60000/60000 [==============================] - 3s 44us/sample - loss: 0.3035 - sparse_categorical_accuracy: 0.9148
        Epoch 2/5
        60000/60000 [==============================] - 2s 34us/sample - loss: 0.1368 - sparse_categorical_accuracy: 0.9597
        Epoch 3/5
        60000/60000 [==============================] - 2s 34us/sample - loss: 0.0917 - sparse_categorical_accuracy: 0.9731
        Epoch 4/5
        60000/60000 [==============================] - 2s 34us/sample - loss: 0.0697 - sparse_categorical_accuracy: 0.9795
        Epoch 5/5
        60000/60000 [==============================] - 2s 32us/sample - loss: 0.0546 - sparse_categorical_accuracy: 0.9836

Out[8]: <tensorflow.python.keras.callbacks.History at 0x1418e333438>

In [9]: model.evaluate(X_test,  y_test, verbose=2)

        10000/1 - 1s - loss: 0.0402 - sparse_categorical_accuracy: 0.9749

Out[9]: [0.07920550688984804, 0.9749]
```

```
In [10]: model.save_weights("mnist_weights.h5")
```



```
In [1]: import tensorflow as tf
        tf.__version__, tf.keras.__version__
Out[1]: ('2.0.0', '2.2.4-tf')

In [2]: import numpy as np
        import matplotlib.pyplot as plt

In [3]: gpus = tf.config.experimental.list_physical_devices('GPU')
        tf.config.experimental.set_memory_growth(gpus[0], True)

In [4]: mnist=tf.keras.datasets.mnist
        (train_x,train_y),(test_x,test_y)=mnist.load_data()

In [5]: X_train,X_test=tf.cast(train_x/255.0,tf.float32),tf.cast(test_x/255.0,tf.float32)
        y_train,y_test=tf.cast(train_y,tf.int16),tf.cast(test_y,tf.int16)
```

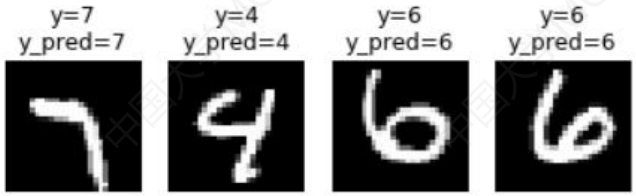
```
In [6]: model=tf.keras.Sequential()
        model.add(tf.keras.layers.Flatten(input_shape=(28,28)))
        model.add(tf.keras.layers.Dense(128,activation="relu"))
        model.add(tf.keras.layers.Dense(10,activation="softmax"))

In [7]: model.compile(optimizer='adam',
                      loss='sparse_categorical_crossentropy',
                      metrics=['sparse_categorical_accuracy'])
```

```
In [8]: model.load_weights("mnist_weights.h5")

In [9]: model.evaluate(X_test, y_test, verbose=2)
10000/1 - 1s - loss: 0.0402 - sparse_categorical_accuracy: 0.9749
Out[9]: [0.07920550688984804, 0.9749]
```

上次测试的结果

```
In [9]: model.evaluate(X_test, y_test, verbose=2)
10000/1 - 1s - loss: 0.0402 - sparse_categorical_accuracy: 0.9749
Out[9]: [0.07920550688984804, 0.9749]
```

In [10]:
```
for i in range(4):
    num = np.random.randint(1,10000)

    plt.subplot(1,4,i+1)
    plt.axis("off")
    plt.imshow(test_x[num],cmap='gray')
    y_pred=np.argmax(model.predict([[X_test[num]]]))
    title="y="+str(test_y[num])+"\ny_pred="+str(y_pred)
    plt.title(title)

plt.show()
```



save_weights()方法仅保存了神经网络的**模型参数**，
使用load_weights方法之前，需要首先定义一个完全相同的神经网络模型

```
In [6]: model.load_weights("mnist_weights.h5")
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-6-8028adecf0e0> in <module>
----> 1 model.load_weights("mnist_weights.h5")

NameError: name 'model' is not defined
```

model.load_weights( ) ← model.fit( )

保存整个模型

```
model.save ( filepath,
             overwrite=True,
             include_optimizer=True,
             save_format=None
           )
```

- 神经网络的结构
- 模型参数
- 配置信息（优化器，损失函数等）
- 优化器状态

HDF5格式

```
**.h5 ,**.keras
save_format=None
```

SavedModel

```
save_format= "tf"
```

```
----mnist_model_tf
    |-----assets
    |-----variables
          |------variables.data-00000-of-00002
          |------variables.data-00001-of-00002
          |------variables.index
    |-----saved_model.pb
```

加载模型

```
tf.keras.models.load_model()
```

实例：保存**手写数字识别**模型

```
In [11]: model.save("mnist_model.h5")
```

In [1]:
```
import tensorflow as tf
tf.__version__, tf.keras.__version__
```

Out[1]:
```
('2.0.0', '2.2.4-tf')
```

In [2]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [3]:
```
gpus = tf.config.experimental.list_physical_devices('GPU')
tf.config.experimental.set_memory_growth(gpus[0], True)
```

In [4]:
```
mnist=tf.keras.datasets.mnist
(train_x,train_y),(test_x,test_y)=mnist.load_data()
```

In [5]:
```
X_train,X_test=tf.cast(train_x/255.0,tf.float32),tf.cast(test_x/255.0,tf.float32)
y_train,y_test=tf.cast(train_y,tf.int16),tf.cast(test_y,tf.int16)
```

```
In [6]: model=tf.keras.models.load_model('mnist_model.h5')

In [7]: model.summary()
```

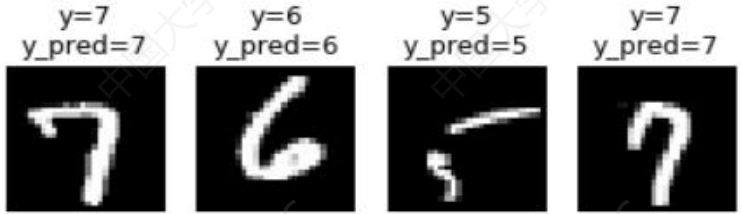
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               100480
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

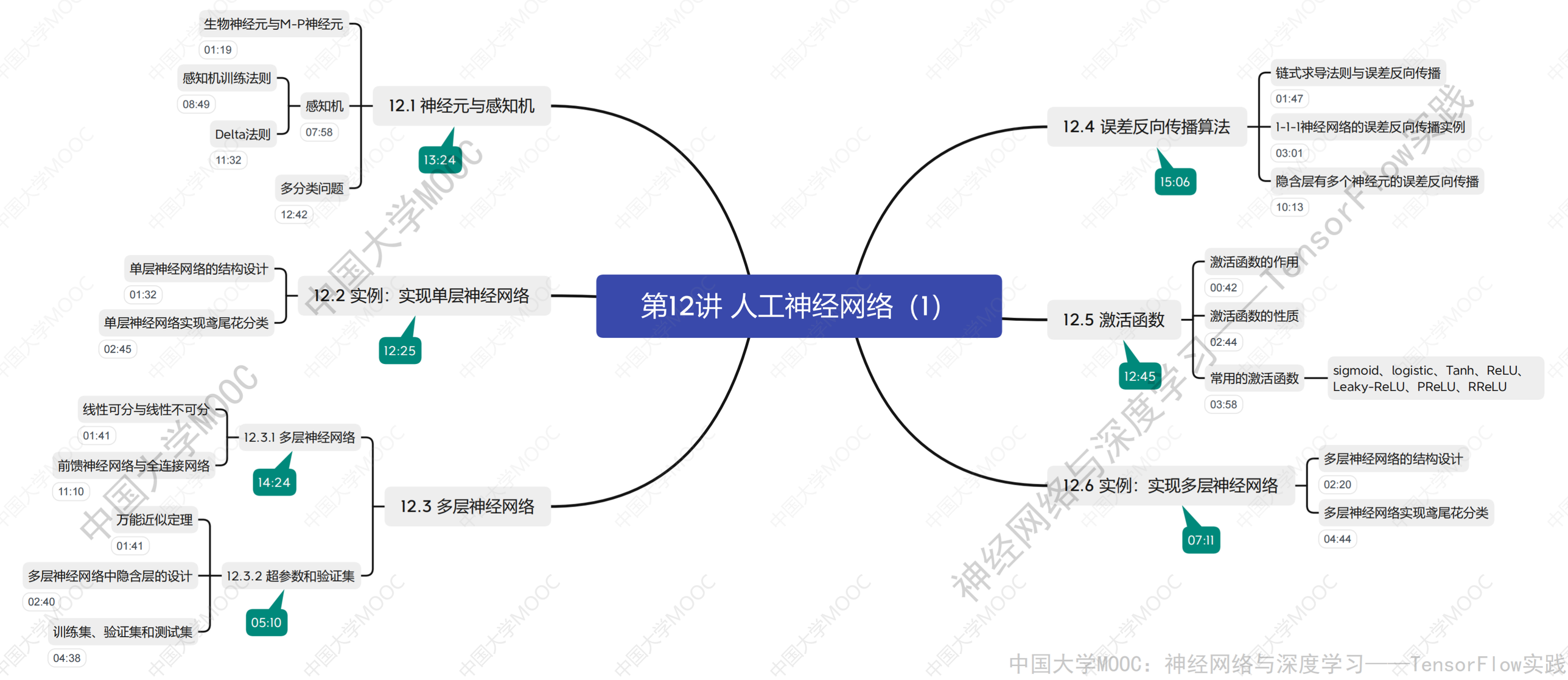
```
In [8]: model.evaluate(X_test, y_test, verbose=2)
10000/1 - 1s - loss: 0.0402 - sparse_categorical_accuracy: 0.9749
Out[8]: [0.07920550688984804, 0.9749]
```

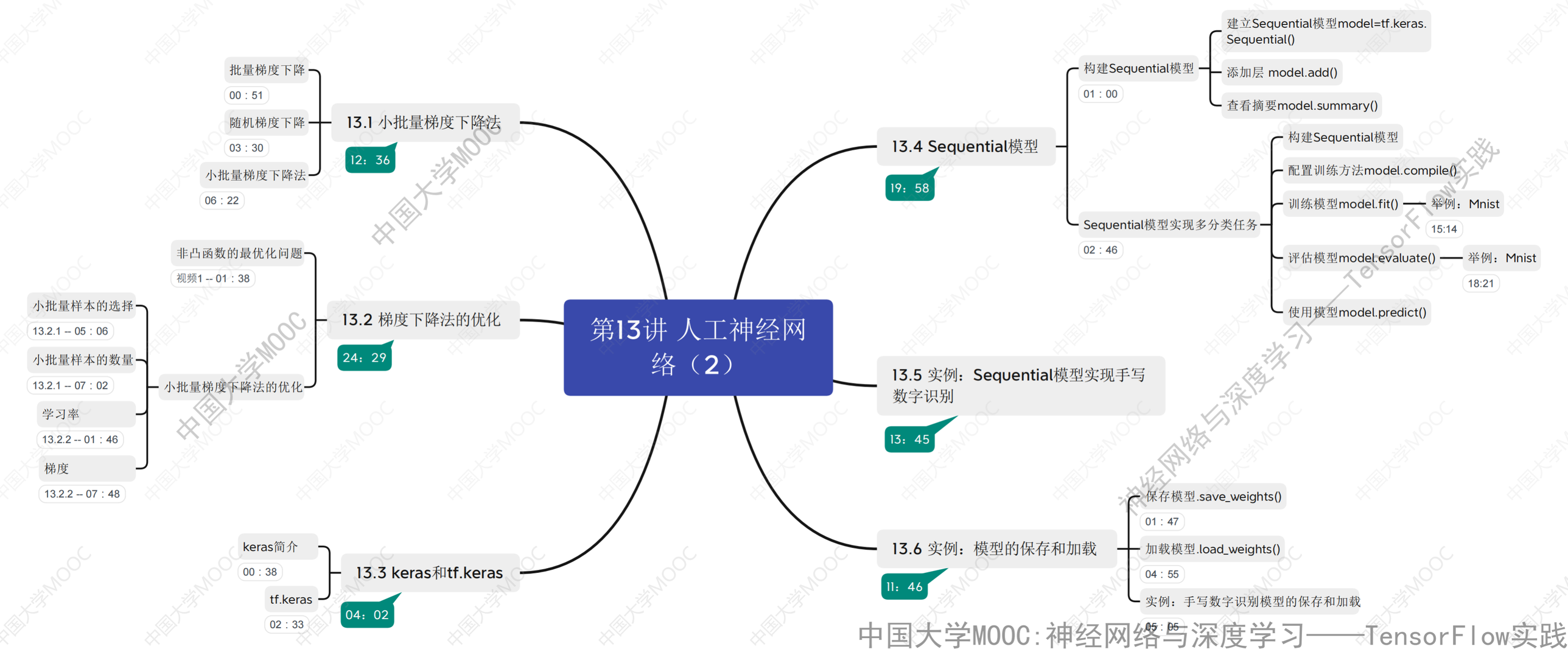
上次测试的结果

```
In [9]: model.evaluate(X_test, y_test, verbose=2)
10000/1 - 1s - loss: 0.0402 - sparse_categorical_accuracy: 0.9749
Out[9]: [0.07920550688984804, 0.9749]
```

In [9]:
```
for i in range(4):
    num = np.random.randint(1, 10000)

    plt.subplot(1,4,i+1)
    plt.axis("off")
    plt.imshow(test_x[num], cmap='gray')
    y_pred=np.argmax(model.predict([[X_test[num]]]))
    title="y="+str(test_y[num])+"\ny_pred="+str(y_pred)
    plt.title(title)

plt.show()
```




第12讲 人工神经网络（1）
12.1 神经元与感知机
13:24
生物神经元与M-P神经元
01:19
感知机
07:58
感知机训练法则
08:49
Delta法则
11:32
多分类问题
12:42
12.2 实例：实现单层神经网络
12:25
单层神经网络的结构设计
01:32
单层神经网络实现鸢尾花分类
02:45
12.3 多层神经网络
12.3.1 多层神经网络
14:24
线性可分与线性不可分
01:41
前馈神经网络与全连接网络
11:10
12.3.2 超参数和验证集
05:10
万能近似定理
01:41
多层神经网络中隐含层的设计
02:40
训练集、验证集和测试集
04:38
12.4 误差反向传播算法
15:06
链式求导法则与误差反向传播
01:47
1-1-1神经网络的误差反向传播实例
03:01
隐含层有多个神经元的误差反向传播
10:13
12.5 激活函数
12:45
激活函数的作用
00:42
激活函数的性质
02:44
常用的激活函数
03:58
sigmoid、logistic、Tanh、ReLU、Leaky-ReLU、PReLU、RReLU
12.6 实例：实现多层神经网络
07:11
多层神经网络的结构设计
02:20
多层神经网络实现鸢尾花分类
04:44
中国大学MOOC
神经网络与深度学习——TensorFlow实践
中国大学MOOC：神经网络与深度学习——TensorFlow实践


第13讲 人工神经网络（2）
13.1 小批量梯度下降法
12：36
批量梯度下降
00：51
随机梯度下降
03：30
小批量梯度下降法
06：22
13.2 梯度下降法的优化
24：29
非凸函数的最优化问题
视频1 -- 01：38
小批量梯度下降法的优化
小批量样本的选择
13.2.1 -- 05：06
小批量样本的数量
13.2.1 -- 07：02
学习率
13.2.2 -- 01：46
梯度
13.2.2 -- 07：48
13.3 keras和tf.keras
04：02
keras简介
00：38
tf.keras
02：33
13.4 Sequential模型
19：58
构建Sequential模型
01：00
建立Sequential模型model=tf.keras.Sequential()
添加层 model.add()
查看摘要model.summary()
Sequential模型实现多分类任务
02：46
构建Sequential模型
配置训练方法model.compile()
训练模型model.fit()
举例：Mnist
15:14
评估模型model.evaluate()
举例：Mnist
18:21
使用模型model.predict()
13.5 实例：Sequential模型实现手写数字识别
13：45
13.6 实例：模型的保存和加载
11：46
保存模型.save_weights()
01：47
加载模型.load_weights()
04：55
实例：手写数字识别模型的保存和加载
05：05
中国大学MOOC
神经网络与深度学习——TensorFlow实践
中国大学MOOC：神经网络与深度学习——TensorFlow实践

第 13 讲 人工神经网络(2)拓展题目

题目一：

请使用低阶 API 分别实现随机梯度下降法和小批量梯度下降法。

①代码
②实验结果
③实验小结