第4章　神经网络

刘家锋

哈尔滨工业大学

第4章　神经网络

4.1 神经网络结构

神经网络与机器学习

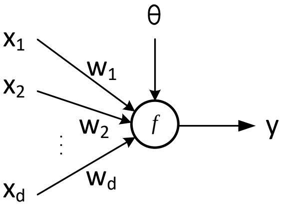
- **什么是神经网络？**
  - 神经网络是一个多学科交叉的研究领域
  - Kohonen对神经网络的定义：神经网络是由具有适应性的简单单元组成的广泛并行互连的网络，它的组织能够模拟生物神经系统对真实世界事物所作出的交互反映
- **机器学习中的神经网络**
  - 机器学习以神经网络作为学习机，完成分类或回归任务
  - 神经网络可能是到目前为止能力最强的机器学习工具

神经元

- **神经元模型**
    - 神经元是组成神经网络的基本单元
    - 实现输入向量$\mathbf{x}$到输出y的映射，$\mathbf{x} = (x_1, \cdots, x_d)^t \to y$：

$$y = f(\mathbf{w}^t\mathbf{x} - \theta) = f\left(\sum_{i=1}^{d} w_i x_i - \theta\right)$$

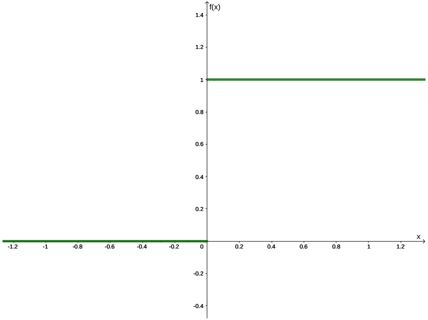
其中，f为激活函数，$\mathbf{w}$为权值向量，θ为偏置或阈值



激活函数

- **阶跃函数**

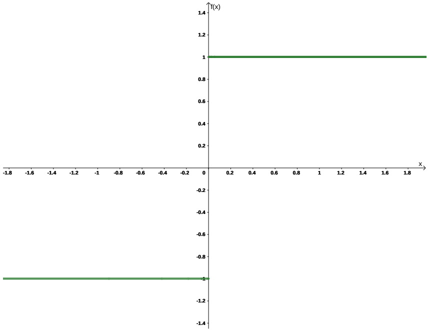
$$f(x) = \begin{cases} 0, & x < 0 \\ +1, & x \geq 0 \end{cases}$$



激活函数

- **符号函数**

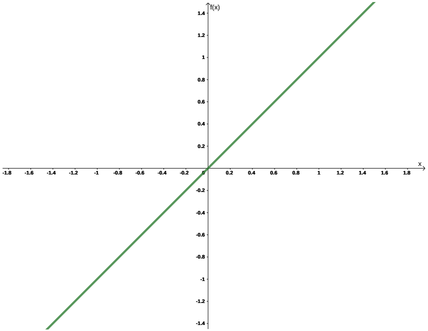
$$f(x) = \begin{cases} -1, & x < 0 \\ +1, & x \geq 0 \end{cases}$$



激活函数

- **线性函数**

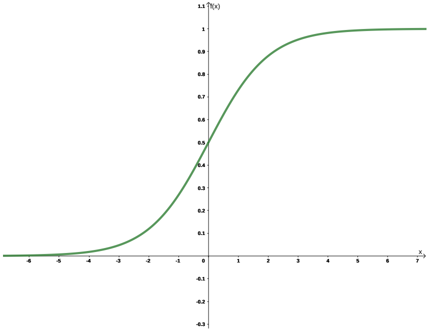
$$f(x) = x$$



激活函数

- **对数Sigmoid函数**

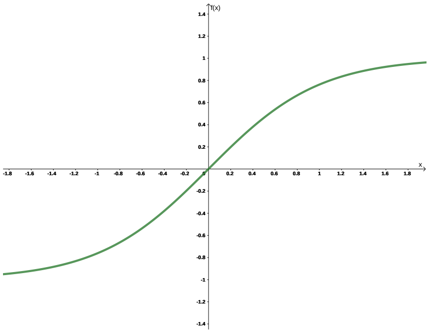
$$f(x) = \frac{1}{1+e^{-x}}$$



激活函数

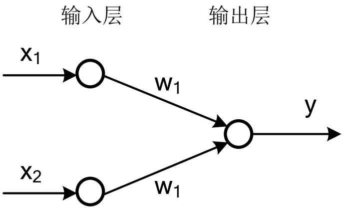
- **双曲正切Sigmoid函数**

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



两层神经网络

- **感知机由两层神经元组成**
  - 输入层：d个神经元，负责接收输入信号，传递给输出层
  - 输出层：1个神经元，完成分类或回归任务
  - 线性回归任务：输出层使用线性激活函数
  - 线性分类任务：输出层使用阶跃或符号激活函数

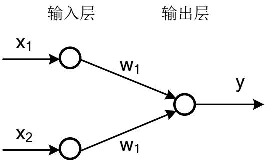


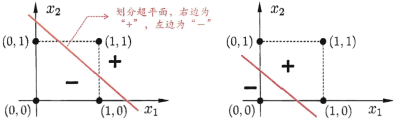
两层神经网络

- **感知机的局限性**
  - 感知机的本质是一个线性映射：

$$y = w_1x_1 + w_2x_2 - \theta$$

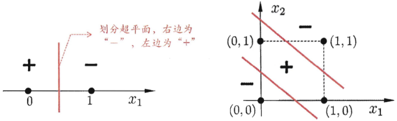
  - 可以完成线性分类和回归，但无法完成非线性分类和回归；





(a) “与”问题 ($x_1 \wedge x_2$)

(b) “或”问题 ($x_1 \vee x_2$)

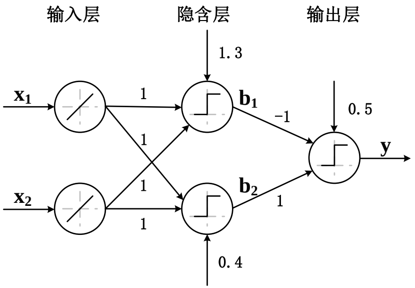


(c) “非”问题 ($\neg x_1$)

(d) “异或”问题 ($x_1 \oplus x_2$)

多层神经网络

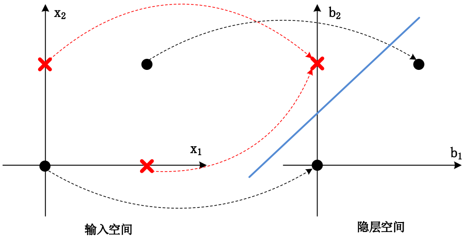
- 多层神经网络
  - 提高神经网络能力的方法是增加网络的层数
  - 在输入层和输出层之间增加一个隐含层，可以很容易地解决异或问题
  - 隐含层和输出层的神经元均使用阶跃激活函数



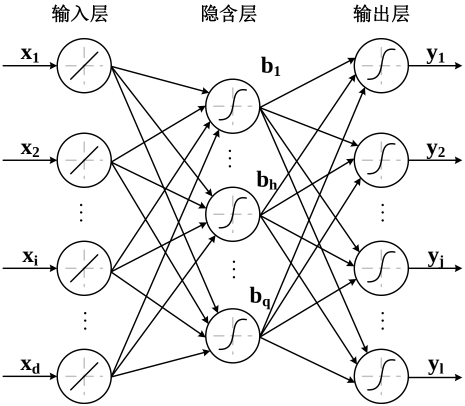
多层神经网络

- **多层网络非线性分类的原理**
  - 隐含层实现对输入空间的非线性映射
  - 输出层实现线性分类
  - XOR问题的映射过程

$$\begin{aligned} (0,0),(1,1) &\rightarrow (0,0),(1,1) &&\rightarrow 0 \\ (1,0),(0,1) &\rightarrow (0,1) &&\rightarrow 1 \end{aligned}$$



三层神经网络



多层神经网络

- **网络层数的选择**
  - 理论上来说，只要隐含层神经元的数量足够多，三层网络可以解决任何问题
  - 实际应用中，增加网络的层数可以减少总的神经元数量，对于复杂问题，现在更倾向于设计深层的网络
- **神经元数量的选择**
  - 输入层：输入属性的数量d，使用线性激活函数
  - 隐含层：根据需要来设定，采用线性或非线性的激活函数
  - 输出层
    - 回归问题：输出的数量l，可以采用线性或Sigmoid激活函数
    - 分类问题：类别编码长度，采用Sigmoid 激活函数，现在更多采用Softmax函数

4.2 BP算法

神经网络的学习

- **学习问题**
  - 训练集：$D = \{(\mathbf{x}_1, \mathbf{y}_1), \cdots, (\mathbf{x}_m, \mathbf{y}_m)\}, \mathbf{x}_i \in \mathbb{R}^d, \mathbf{y}_i \in \mathbb{R}^l$
  - 学习网络中每个神经元的权向量和阈值
  - 网络学习的目标是输入训练集中的$\mathbf{x}_k$时，网络的输出$\hat{\mathbf{y}}_k$能够尽量地接近训练数据的期望输出$\mathbf{y}_k$
- **优化目标函数**
  - 对于训练样例$(\mathbf{x}_k, \mathbf{y}_k)$，平方误差作为目标损失函数来优化是合理的选择：

$$E_k = \frac{1}{2}\sum_{j=1}^{l}(\hat{y}_j^k - y_j^k)^2$$

    其中，$\mathbf{y}_k = (y_1^k, \cdots, y_l^k)^t, \hat{\mathbf{y}}_k = (\hat{y}_1^k, \cdots, \hat{y}_l^k)^t$

  - 对于训练集D，目标损失函数为：

$$E = \frac{1}{m}\sum_{k=1}^{m}E_k$$

神经网络的学习

- **误差逆传播算法**
  - BP(Back Propagation)算法是神经网络学习最经典的算法
  - BP算法的本质就是对平方误差函数的梯度下降优化
  - 网络中任意参数v(权向量元素，阈值)的迭代优化公式：

$$v \leftarrow v + \Delta v = v - \eta \frac{\partial E}{\partial v}$$

  - 以三层网络为例，推导一个样例的平方误差E_k关于网络中每个参数的梯度公式
  - 需要学习的网络参数：隐含层权向量和阈值，输出层权向量和阈值

BP算法

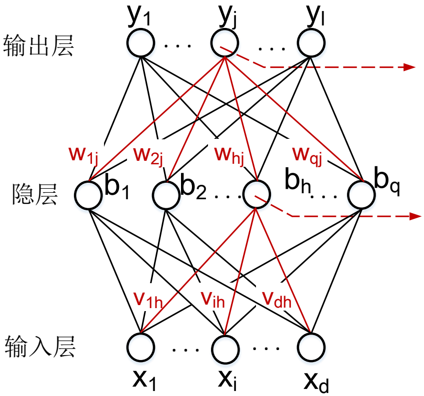
- **定义符号**
  - 网络的输入$\mathbf{x} = (x_1, \cdots, x_d)^t$，期望输出$\mathbf{y} = (y_1, \cdots, y_l)^t$
  - 隐含层有q个神经元，输入层第i个神经元与隐含层第h个神经元的连接权为v_{ih}，隐含层第h个神经元的阈值为γ_h
  - 隐含层第h个神经元接收到的输入α_h和输出b_h：

$$\alpha_h = \sum_{i=1}^{d} v_{ih} x_i, \quad b_h = f(\alpha_h - \gamma_h)$$

  - 输出层有l个神经元，隐含层第h个神经元与输出层第j个神经元的连接权为w_{hj}，输出层第j个神经元的阈值为θ_j
  - 输出层第j个神经元接收到的输入β_j和输出$\hat{y}_j$：

$$\beta_j = \sum_{h=1}^{q} w_{hj} b_h, \quad \hat{y}_j = f(\beta_j - \theta_j)$$

BP算法



第j个输出层神经元的输入

$$\beta_j = \sum_{h=1}^{q} w_{hj} b_h$$

第h个隐层神经元的输入

$$\alpha_h = \sum_{i=1}^{d} v_{ih} x_i$$

输出层参数的梯度

平方误差关于权w_{hj}的梯度：

$$\frac{\partial E_{\mathbf{x}}}{\partial w_{hj}} = \frac{\partial E_{\mathbf{x}}}{\partial \hat{y}_j} \cdot \frac{\partial \hat{y}_j}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial w_{hj}}$$

其中：

$$E_{\mathbf{x}} = \frac{1}{2}\sum_{j=1}^{l}(\hat{y}_j - y_j)^2 \quad \Rightarrow \quad \frac{\partial E_{\mathbf{x}}}{\partial \hat{y}_j} = \hat{y}_j - y_j$$

$$\hat{y}_j = f(\beta_j - \theta_j) \quad \Rightarrow \quad \frac{\partial \hat{y}_j}{\partial \beta_j} = f'(\beta_j - \theta_j)$$

$$\beta_j = \sum_{t=1}^{q} w_{tj} b_t \quad \Rightarrow \quad \frac{\partial \beta_j}{\partial w_{hj}} = b_h$$

代入得到：

$$\frac{\partial E_{\mathbf{x}}}{\partial w_{hj}} = \left[(\hat{y}_j - y_j) f'(\beta_j - \theta_j)\right] b_h = -g_j b_h$$

类似可得：

$$\frac{\partial E_{\mathbf{x}}}{\partial \theta_j} = -(\hat{y}_j - y_j) f'(\beta_j - \theta_j) = g_j$$

隐含层参数的梯度

平方误差关于权v_{ih}的梯度：

$$\frac{\partial E_{\mathbf{x}}}{\partial v_{ih}} = \frac{\partial E_{\mathbf{x}}}{\partial b_h} \cdot \frac{\partial b_h}{\partial \alpha_h} \cdot \frac{\partial \alpha_h}{\partial v_{ih}}$$

其中：

$$b_h = f(\alpha_h - \gamma_h) \quad \Rightarrow \quad \frac{\partial b_h}{\partial \alpha_h} = f'(\alpha_h - \gamma_h)$$

$$\alpha_h = \sum_{t=1}^{d} v_{th} x_t \quad \Rightarrow \quad \frac{\partial \alpha_h}{\partial v_{ih}} = x_i$$

由于：$E_{\mathbf{x}} = \frac{1}{2} \sum_{j=1}^{l} (\hat{y}_j - y_j)^2$，因此：

$$\begin{aligned} \frac{\partial E_{\mathbf{x}}}{\partial b_h} &= \sum_{j=1}^{l} (\hat{y}_j - y_j) \cdot \frac{\partial \hat{y}_j}{\partial b_h} \\ &= \sum_{j=1}^{l} (\hat{y}_j - y_j) \cdot \frac{\partial \hat{y}_j}{\partial \beta_j} \cdot \frac{\partial \beta_j}{\partial b_h} \\ &= \sum_{j=1}^{l} (\hat{y}_j - y_j) f'(\beta_j - \theta_j) w_{hj} = -\sum_{j=1}^{l} w_{hj} g_j \end{aligned}$$

隐含层参数的梯度

代入得到：

$$\frac{\partial E_{\mathbf{x}}}{\partial v_{ih}} = -\left[f'(\alpha_h - \gamma_h) \sum_{j=1}^{l} w_{hj} g_j \right] x_i = -e_h x_i$$

类似得到：

$$\frac{\partial E_{\mathbf{x}}}{\partial \gamma_h} = f'(\alpha_h - \gamma_h) \sum_{j=1}^{l} w_{hj} g_j = e_h$$

Sigmoid激活函数的导数：$f(x) = \frac{1}{1+e^{-x}}$

$$f'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}} - \left(\frac{1}{1+e^{-x}}\right)^2 = f(x)\left(1 - f(x)\right)$$

由$\hat{y}_j = f(\beta_j - \theta_j), b_h = f(\alpha_h - \gamma_h)$ 得到：

$$f'(\beta_j - \theta_j) = \hat{y}_j(1 - \hat{y}_j)$$
$$f'(\alpha_h - \gamma_h) = b_h(1 - b_h)$$

BP算法

- **BP算法的迭代公式**
  - 输出层：

$$w_{hj} \leftarrow w_{hj} + \eta g_j b_h, \qquad \theta_j \leftarrow \theta_j - \eta g_j$$

其中

$$g_j = -\hat{y}_j(1-\hat{y}_j)(\hat{y}_j - y_j)$$

  - 隐含层：

$$v_{ih} \leftarrow v_{ih} + \eta e_h x_i, \qquad \gamma_h \leftarrow \gamma_h - \eta e_h$$

其中

$$e_h = b_h(1-b_h)\sum_{j=1}^{l} w_{hj} g_j$$

BP算法

Algorithm 1 标准BP算法

```
Input: 数据集$D = \{(\mathbf{x}_k, \mathbf{y}_k)\}_{k=1}^{m}$，学习率$\eta$
Output: 多层网络的参数
随机初始化网络中所有连接权和阈值
repeat
    for all $(\mathbf{x}, \mathbf{y}) \in D$ do
        使用当前网络参数，前馈计算隐含层的输出$\{b_h\}$
        计算输出层的输出$\{\hat{y}_j\}$
        计算输出层的梯度项$\{g_j\}$
        计算隐含层的梯度项$\{e_h\}$
        更新连接权$\{w_{hj}\}, \{v_{ih}\}$，阈值$\{\theta_j\}, \{\gamma_h\}$
    end for
until 达到停止条件
```

BP算法

Algorithm 2 累积BP算法

```
Input: 数据集$D = \{(\mathbf{x}_k, \mathbf{y}_k)\}_{k=1}^{m}$，学习率$\eta$
Output: 多层网络的参数
随机初始化网络中所有连接权和阈值
repeat
    $\Delta w_{hj} \leftarrow 0, \Delta v_{ih} \leftarrow 0, \Delta\theta_j \leftarrow 0, \Delta\gamma_h \leftarrow 0$
    for all $(\mathbf{x}, \mathbf{y}) \in D$ do
        使用当前网络参数，计算输出$\{b_h\}, \{\hat{y}_j\}$，梯度项$\{g_j\}, \{e_h\}$;
        $\Delta w_{hj} \leftarrow \Delta w_{hj} + g_j b_h, \quad \Delta v_{ih} \leftarrow \Delta v_{ih} + e_h x_i$
        $\Delta\theta_j \leftarrow \Delta\theta_j - g_j, \quad \Delta\gamma_h \leftarrow \Delta\gamma_h - e_h$
    end for
    更新连接权：$w_{hj} \leftarrow w_{hj} + \eta\Delta w_{hj}/m, \quad v_{ih} \leftarrow v_{ih} + \eta\Delta v_{ih}/m$
    更新阈值：$\theta_j \leftarrow \theta_j + \eta\Delta\theta_j/m, \quad \gamma_h \leftarrow \gamma_h + \eta\gamma_h/m$
until 达到停止条件
```

BP算法

Algorithm 3 随机梯度BP算法

```
Input: 数据集$D = \{(\mathbf{x}_k, \mathbf{y}_k)\}_{k=1}^{m}$，学习率$\eta$
Output: 多层网络的参数
随机初始化网络中所有连接权和阈值
repeat
    随机抽取包含$m'$个数据的集合$D' \in D$，$m' \ll m$
    $\Delta w_{hj} \leftarrow 0, \Delta v_{ih} \leftarrow 0, \Delta\theta_j \leftarrow 0, \Delta\gamma_h \leftarrow 0$
    for all $(\mathbf{x}, \mathbf{y}) \in D'$ do
        使用当前网络参数，计算输出$\{b_h\}, \{\hat{y}_j\}$，梯度项$\{g_j\}, \{e_h\}$
        累积增量$\Delta w_{hj}, \Delta v_{ih}, \Delta\theta_j, \Delta\gamma_h$
    end for
    更新连接权：$w_{hj} \leftarrow w_{hj} + \eta\Delta w_{hj}/m'$，  $v_{ih} \leftarrow v_{ih} + \eta\Delta v_{ih}/m'$
    更新阈值：$\theta_j \leftarrow \theta_j + \eta\Delta\theta_j/m'$，  $\gamma_h \leftarrow \gamma_h + \eta\gamma_h/m'$
until 达到停止条件
```

BP算法

- **标准BP算法**
    - 每个样例更新一次权值和阈值
    - 更新频率高，迭代次数多，但每一次迭代的计算量小
- **累积BP算法**
    - 整个训练集更新一次权值和阈值
    - 更新频率低，但每一次迭代的计算量大
- **随机梯度算法**
    - 在标准算法和累积算法之间折中
    - 特别适用于大数据集的学习，是深度学习中的常用方法

例4.1 神经网络回归(4-1 NN-Regression.ipynb)

- **仿真数据生成**
  - 输入数据x：在区间$[-3,3]$生成均匀分布的100个随机数据

$$x \sim U(-3,+3)$$

  - 真实的输出数据y_{pure}：

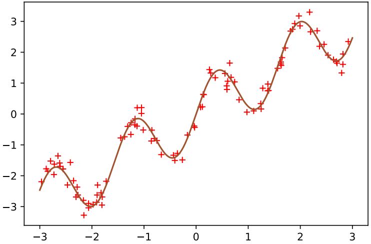
$$y_{pure} = \sin(4x) + x$$

  - 添加噪声的数据y：

$$y = y_{pure} + \delta, \qquad \delta \sim N(0, 0.25)$$

数据生成

```
import numpy as np
import matplotlib.pyplot as plt

rnd = np.random.RandomState(42)
x = rnd.uniform(-3,3,size=100)
y_pure = np.sin(4*x) + x
y = y_pure + rnd.normal(size=len(x))/4

plt.plot(x,y,'r+')

xx = np.linspace(-3, 3, 1000).reshape(-1, 1)
yy = np.sin(4*xx) + xx
plt.plot(xx,yy,'sienna')

plt.show()
```

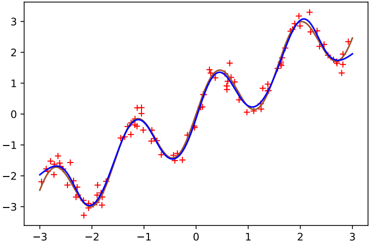


构建和学习神经网络

- **网络结构**
    - 输入层：1个神经元
    - 隐含层：20个神经元，对数Sigmoid激活函数
    - 输出层：1个神经元，线性激活函数
- **网络学习参数**
    - 学习算法：
        - lbfgs：拟牛顿法，速度快，适合小数据集学习
        - sgd，adam：随机梯度和改进的随机梯度，适合大数据集学习
    - 最大迭代次数：20000
    - 权值衰减系数：0.01

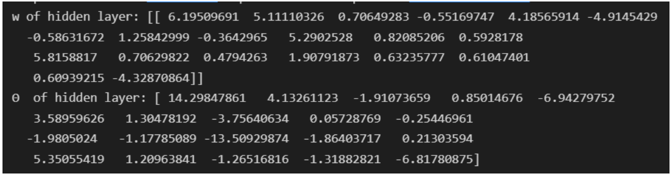
构建和学习神经网络

```
from sklearn.neural_network import MLPRegressor

mlp = MLPRegressor( hidden_layer_sizes=(20),\
      activation='logistic',solver='lbfgs',\
      alpha=0.01,max_iter=20000)
mlp.fit(x.reshape(-1,1),y)

plt.plot(x,y,'r+')
plt.plot(xx,yy,'sienna')

predict_yy = mlp.predict(xx)
plt.plot(xx, predict_yy, 'b')

plt.show()
```

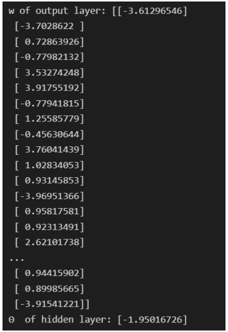


显示网络参数

```python
print("w of hidden layer:", mlp.coefs_[0])
print(chr(920)," of hidden layer:", mlp.intercepts_[0])

print("\nw of output layer:", mlp.coefs_[1])
print(chr(920)," of hidden layer:", mlp.intercepts_[1])
```



```
w of hidden layer: [[ 6.19509691  5.11110326  0.70649283 -0.55169747  4.18565914 -4.9145429
  -0.58631672  1.25842999 -0.3642965   5.2902528   0.82085206  0.5928178
   5.8158817   0.70629822  0.4794263   1.90791873  0.63235777  0.61047401
   0.60939215 -4.32870864]]
Θ  of hidden layer: [ 14.29847861   4.13261123  -1.91073659   0.85014676  -6.94279752
   3.58959626   1.30478192  -3.75640634   0.05728769  -0.25446961
  -1.9805024   -1.17785089 -13.50929874  -1.86403717   0.21303594
   5.35055419   1.20963841  -1.26516816  -1.31882821  -6.81780875]
```



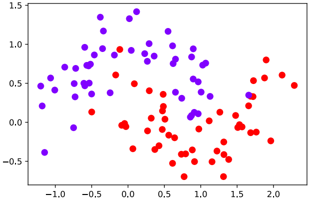
```
w of output layer: [[-3.61296546]
 [-3.7028622 ]
 [ 0.72863926]
 [-0.77982132]
 [ 3.53274248]
 [ 3.91755192]
 [-0.77941815]
 [ 1.25585779]
 [-0.45630644]
 [ 3.76041439]
 [ 1.02834053]
 [ 0.93145853]
 [-3.96951366]
 [ 0.95817581]
 [ 0.92313491]
 [ 2.62101738]
...
 [ 0.94415902]
 [ 0.89985665]
 [-3.91541221]]
Θ  of hidden layer: [-1.95016726]
```

例4.2 神经网络分类(4-2 NN-Classification.ipynb)

- **仿真数据生成**
  - 100个二分类2维数据
  - 每个类别的数据分布在一个半圆形的区域
  - 两个半圆形区域相互交叠
  - 数据集随机划分为训练集和测试集

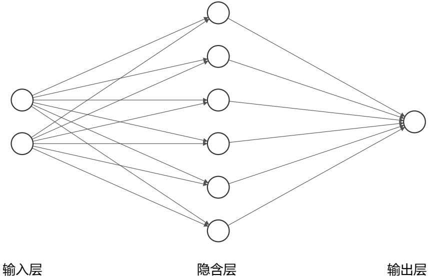
数据生成

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=100, noise=0.25, random_state=3)
X_train,X_test,y_train,y_test=train_test_split(X,y,stratify=y,random_state=42)

plt.scatter(X[:,0], X[:,1], c=y, s=50, cmap='rainbow')
plt.show()
```



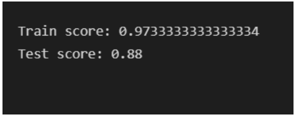
构建和学习神经网络

- **网络结构及学习参数**
  - 输入层：2个神经元，隐含层：6个神经元，输出层：1个神经元
  - 隐含层和输出层均使用对数Sigmoid激活函数
  - 学习算法：lbfgs，迭代次数：40

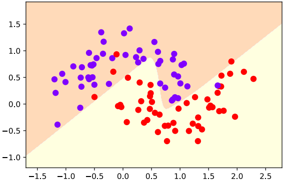


构建和学习神经网络

```
from sklearn.neural_network import MLPClassifier
from plot_decision_boundary import plot_decision_boundary

mlp=MLPClassifier(hidden_layer_sizes=(6),solver='lbfgs',activation='logistic',\
              max_iter=40, random_state=0).fit(X_train, y_train)

print('Train score:', mlp.score(X_train,y_train))
print('Test score:', mlp.score(X_test,y_test))
plot_decision_boundary(mlp,axis=[x_min,x_max,y_min,y_max])
plt.scatter(X[:,0], X[:,1], c=y, s=50, cmap='rainbow')
```



```
Train score: 0.9733333333333334
Test score: 0.88
```



4.3 神经网络学习

缓解过拟合

- **过拟合问题**
    - 神经网络是一个学习能力很强的模型，不可避免地会出现过拟合现象
    - 随着网络深度的增加，神经元数量的增多，过拟合现象会越发严重
    - 控制过拟合是神经网络学习需要重点关注的问题
- **控制过拟合的方法**
    - 增大训练数据集的规模
    - 正则化技术
    - 提前停止技术(early stopping)

正则化

- **正则化方法**
  - 在平方误差优化函数基础上，增加一个描述网络复杂程度的正则项：

    $$E_{wd} = \frac{1}{m}\sum_{k=1}^{m} E_k + \frac{\alpha}{2\eta}\sum_i w_i^2$$

    其中，$\{w_i\}$包括网络所有的参数，α 称为衰减系数，η为学习率
  - 可以证明，相应的梯度下降迭代公式为：

    $$w_i \leftarrow (1-\alpha)w_i - \eta\frac{\partial E}{\partial w_i}$$

    其中，$E = \frac{1}{m}\sum_{k=1}^{m} E_k$为标准的平方误差函数

算法的迭代次数

- **迭代次数对网络学习的影响**
  - BP算法的迭代次数对网络学习的结果有很大的影响
  - 迭代次数过少，网络会出现“欠学习”现象，分类边界过于简单，无法很好分类训练样本
  - 随着迭代次数的增加，模型会更复杂，能够更好地适应训练样本
  - 迭代次数过多，网络会出现“过学习”现象，分类边界过于复杂，对测试样本错误分类的风险增大

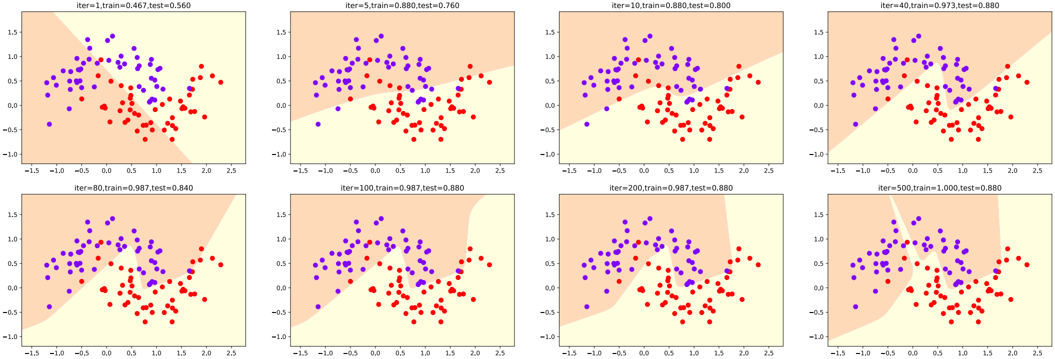
例4.3 提前停止(4-3 Early-Stop.ipynb)

- **迭代次数对分类边界的影响**
    - 使用与例4.2相同的仿真数据
    - 使用与例4.2相同的网络结构
    - 学习算法使用lbfgs
    - 迭代次数分别设置为：$1, 5, 10, 40, 80, 100, 200, 500$
    - 观察网络分类边界的变化

迭代次数对分类边界的影响

```
from sklearn.neural_network import MLPClassifier
from plot_decision_boundary import plot_decision_boundary

eps = 0.5
x_min,x_max = X[:,0].min()-eps, X[:,0].max()+eps
y_min,y_max = X[:,1].min()-eps, X[:,1].max()+eps

fig, axes = plt.subplots(2,4,figsize=(18, 6))
for iter,ax in zip([1,5,10,40,80,100,200,500],axes.reshape([8,1])):
    mlp = MLPClassifier(hidden_layer_sizes=(6), solver='lbfgs', \
            activation='logistic', max_iter=iter, \
            random_state=0).fit(X_train, y_train)
    plot_decision_boundary(mlp,axis=[x_min,x_max,y_min,y_max],ax=ax[0])
    ax[0].scatter(X[:,0], X[:,1], c=y, s=50, cmap='rainbow')
    ax[0].set_title('iter=%d,train=%4.3f,test=%4.3f' \
        %(iter,mlp.score(X_train,y_train),mlp.score(X_test,y_test)))
plt.show()
```

迭代次数的影响


iter=1,train=0.467,test=0.560
iter=5,train=0.880,test=0.760
iter=10,train=0.880,test=0.800
iter=40,train=0.973,test=0.880
iter=80,train=0.987,test=0.840
iter=100,train=0.987,test=0.880
iter=200,train=0.987,test=0.880
iter=500,train=1.000,test=0.880
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5

提前停止

- **提前停止策略**
  - 保留validation_fraction(缺省值10%)的训练数据作为验证集
  - 每一轮迭代之后，测试验证集的正确率
  - 如果连续n_iter_no_change(缺省值10)轮迭代，验证集的正确率都不提高，则停止迭代，算法结束
  - 输出网络参数

权值初始化

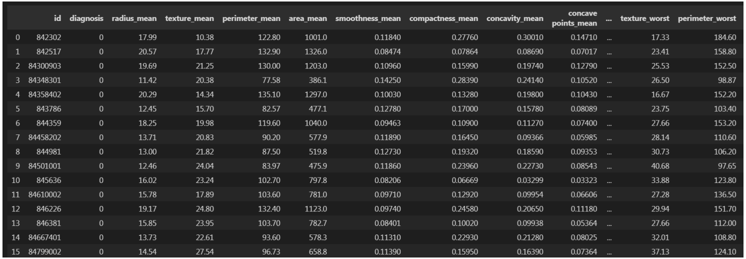
- **样本的规格化**
  - 方式1：每一维特征规格化到[0,1]之间
  - 方式2：每一维特征规格化为均值0，标准差1随机变量
- **权值的初始化**
  - 权值一般初始化在0附近
  - 例如，初始化为$[-\sqrt{6}/n_i, +\sqrt{6}/n_i]$之间均匀分布的随机数，$n_i$为神经元输入连接的数量

例4.4 样本规格化(4-4 Normalization.ipynb)

- **Breast Cancer Dataset**
  - University of Wisconsin的乳腺肿瘤数据
  - 原始属性数据来自于肿瘤细胞核图像的测量数据
    - 半径，纹理，周长，面积
    - 平滑性，紧致性，凹度，凹点数，对称性，分形维度
  - 由原始属性的均值、最大值、最小值等衍生出30维分类属性
  - 类别属性
    - Malignant：恶性
    - Benign：良性

数据准备

```
import numpy as np
import pandas as pd

data = pd.read_csv("BreastCancer.csv")

class_mapping = {’M’:0,’B’:1}
data[’diagnosis’] = data[’diagnosis’].map(class_mapping)
data.head(16)
```



	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	texture_worst	perimeter_worst
0	842302	0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	...	17.33	184.60
1	842517	0	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	...	23.41	158.80
2	84300903	0	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	...	25.53	152.50
3	84348301	0	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	...	26.50	98.87
4	84358402	0	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	...	16.67	152.20
5	843786	0	12.45	15.70	82.57	477.1	0.12780	0.17000	0.15780	0.08089	...	23.75	103.40
6	844359	0	18.25	19.98	119.60	1040.0	0.09463	0.10900	0.11270	0.07400	...	27.66	153.20
7	84458202	0	13.71	20.83	90.20	577.9	0.11890	0.16450	0.09366	0.05985	...	28.14	110.60
8	844981	0	13.00	21.82	87.50	519.8	0.12730	0.19320	0.18590	0.09353	...	30.73	106.20
9	84501001	0	12.46	24.04	83.97	475.9	0.11860	0.23960	0.22730	0.08543	...	40.68	97.65
10	845636	0	16.02	23.24	102.70	797.8	0.08206	0.06669	0.03299	0.03323	...	33.88	123.80
11	84610002	0	15.78	17.89	103.60	781.0	0.09710	0.12920	0.09954	0.06606	...	27.28	136.50
12	846226	0	19.17	24.80	132.40	1123.0	0.09740	0.24580	0.20650	0.11180	...	29.94	151.70
13	846381	0	15.85	23.95	103.70	782.7	0.08401	0.10020	0.09938	0.05364	...	27.66	112.00
14	84667401	0	13.73	22.61	93.60	578.3	0.11310	0.22930	0.21280	0.08025	...	32.01	108.80
15	84799002	0	14.54	27.54	96.73	658.8	0.11390	0.15950	0.16390	0.07364	...	37.13	124.10

划分数据集

```
from sklearn.model_selection import train_test_split

X = data.iloc[:,2:-1].to_numpy()
y = data.iloc[:,1].to_numpy()

print("Shape of X:", X.shape)
print("Number of Malignant:%d, Number of Benign:%d" %(y[y==0].size, y[y==1].size))

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

```
Shape of X: (569, 30)
Number of Malignant:212, Number of Benign:357
```

非规格化数据学习

```python
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(max_iter=1000,random_state=42)
mlp.fit(X_train, y_train)

print("Raw cancer data:")
print("\t Accuracy on training set: %4.3f" %(mlp.score(X_train, y_train)))
print("\t Accuracy on test set: %4.3f\n" %(mlp.score(X_test, y_test)))
```

```
Raw cancer data:
         Accuracy on training set: 0.939
         Accuracy on test set: 0.916
```

规格化数据

- **数据的规格化**
  - 原始数据属性的取值范围差距很大
  - 每一维属性单独规格化为标准高斯分布数据，使其均值为0，方差为1
  - 计算每一维属性的均值和方差：

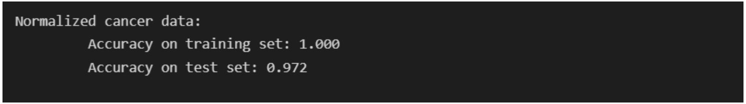
$$\mu = \frac{1}{n_{train}} \sum_{x \in X_{train}} x, \quad \sigma^2 = \frac{1}{n_{train}} \sum_{x \in X_{train}} (x - \mu)^2$$

  - 属性规格化：

$$x \longleftarrow \frac{x - \mu}{\sigma}$$

规格化数据学习

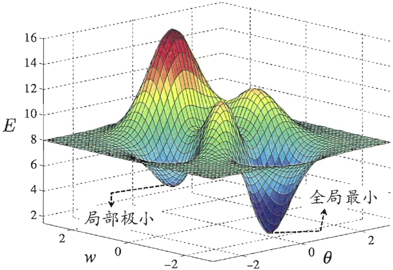
```
mean_on_train = X_train.mean(axis=0); std_on_train = X_train.std(axis=0)

X_train_scaled = (X_train - mean_on_train) / std_on_train
X_test_scaled = (X_test - mean_on_train) / std_on_train

mlp = MLPClassifier(max_iter=1000,random_state=0)
mlp.fit(X_train_scaled, y_train)

print("Normalized cancer data:")
print("\t Accuracy on training set: %3.2f" %(mlp.score(X_train_scaled, y_train)))
print("\t Accuracy on test set: %3.2f" %(mlp.score(X_test_scaled, y_test)))
```



```
Normalized cancer data:
         Accuracy on training set: 1.000
         Accuracy on test set: 0.972
```

全局最小与局部极小

- **BP算法的收敛性**
  - 神经网络的学习是一个参数寻优的过程
  - 平方误差函数在参数空间中存在多个“局部极小值”
  - BP算法只能保证收敛于某个局部极小值，不能保证收敛于“全局最小值”



全局最小与局部极小

- **避免局部极小**
  - 尝试不同的初始参数值，从多个学习结果中选择最好的
  - 使用随机优化的方法，例如：模拟退火，遗传算法等
  - 随机梯度算法在梯度计算中增加了随机性，有助于跳出局部极小，继续搜索
- **全局最小的必要性**
  - 寻找全局最小对神经网络学习来说并不是必须的
  - 全局最小是对训练数据而言的最优参数，而不是测试数据
  - 当前的研究更关注于网络泛化性能的提高，而不过分追求全局最优解

End