径向基函数与自组织特征映射神经网络

机器学习研究室

计算机科学与技术学院
吉林大学

大纲

- 径向基函数
- 径向基函数神经网络
- 广义径向基函数神经网络
- 广义径向基函数神经网络训练算法
- 径向基函数神经网络与多层感知器比较
- 径向基函数神经网络用于分类
- 径向基函数神经网络应用：函数逼近
- 自组织特征映射神经网络

神经网络的典型应用

- 分类问题

$$l = f(\mathbf{x}) \qquad \mathbf{x} \in X \subset R^m \quad l \in C \subset N$$

- 函数逼近(回归问题)

$$\mathbf{y} = f(\mathbf{x}) \qquad \mathbf{x} \in X \subset R^n \quad \mathbf{y} \in Y \subset R^m$$

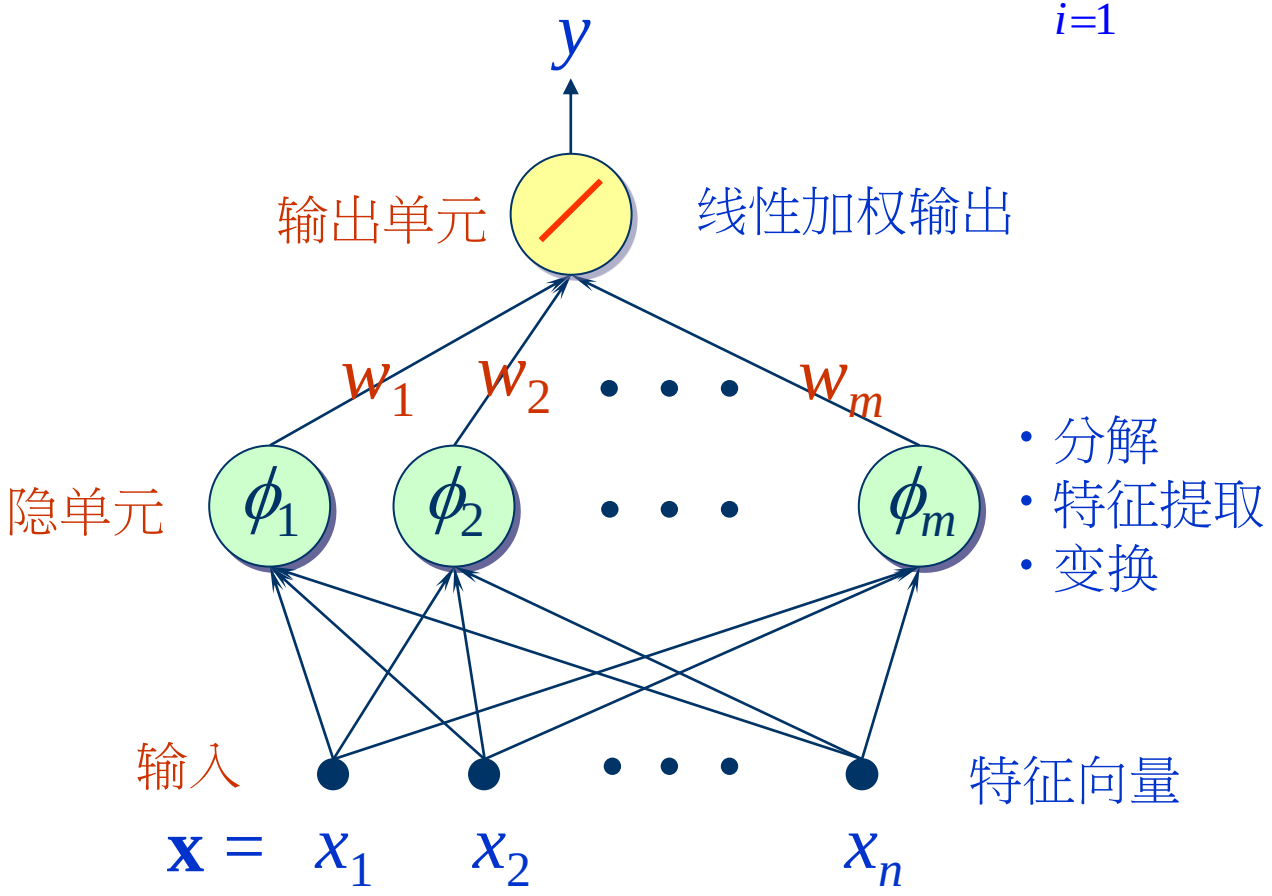
- 时间序列分析

$$\mathbf{x}(t) = f(\mathbf{x}_{t-1}, \mathbf{x}_{t-2}, \mathbf{x}_{t-3}, \ldots)$$

神经网络的典型应用

- 函数逼近(回归问题)

$$f(\mathbf{x}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x})$$



径向基函数

径向基函数（Radial basis function）

- 径向基函数是一个它的值(y)只依赖于变量(x)距原点距离的函数，即

$$\phi(X) = \phi(\|X\|)$$

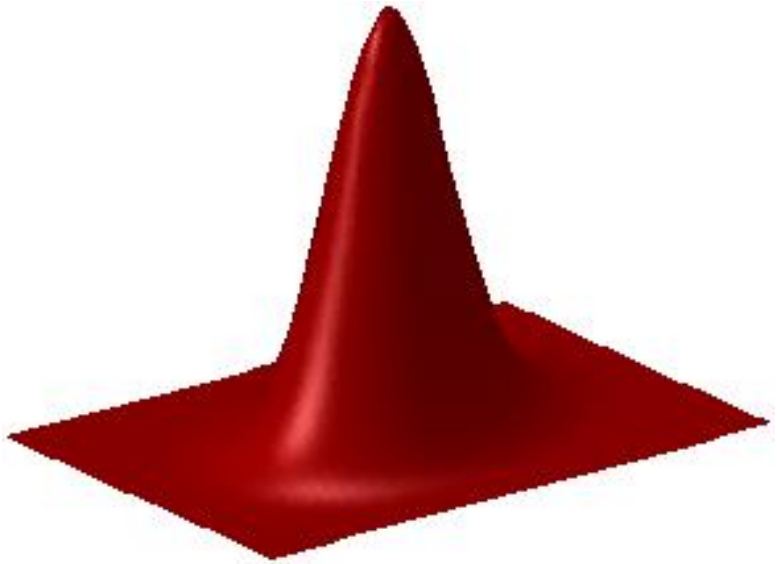
也可以是距其他某个中心点的距离，即

$$\phi(X) = \phi(\|X - c\|)$$

任一满足$\phi(X) = \phi(\|X - c\|)$的函数都可称作径向函数。其中，范数一般为欧几里得距离，不过亦可使用其他距离函数。

径向基函数（Radial basis function）

- 径向函数$\phi(X) = \phi(\|X - c\|)$包含三个参数：
- 中心：c;
- 距离度量：$r = \|X - c\|$;
- 形状：ϕ。



径向基函数（Radial basis function）

- 常见的径向基函数包括：$(r = \|X - c\|)$

- 高斯函数：

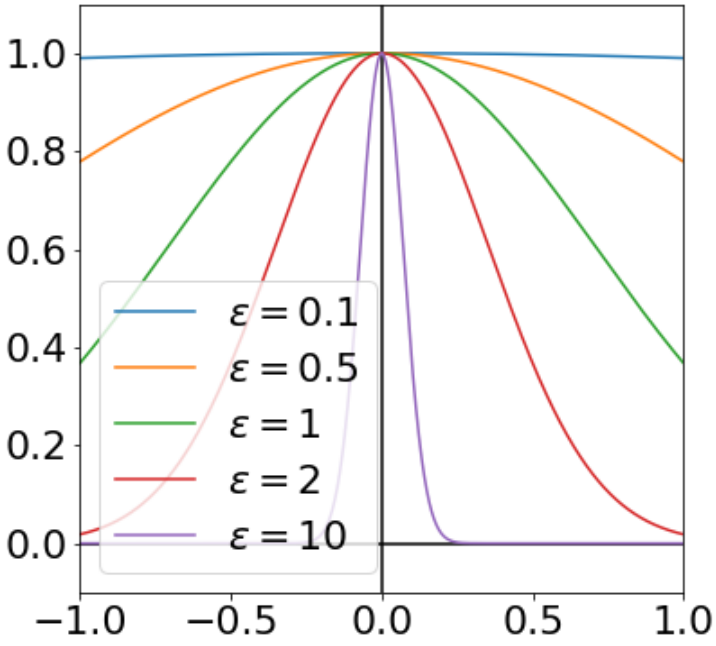
$$\phi(r) = e^{-(\varepsilon r)^2}$$

- 多二次函数（multiquadric）：

$$\phi(r) = \sqrt{1 + (\varepsilon r)^2}$$

- 逆二次函数（inverse quadratic）：

$$\phi(r) = \frac{1}{1 + (\varepsilon r)^2}$$



- 逆多二次函数（inverse multiquadric）：

$$\phi(r) = \frac{1}{\sqrt{1 + (\varepsilon r)^2}}$$

径向基(RBF)神经网络

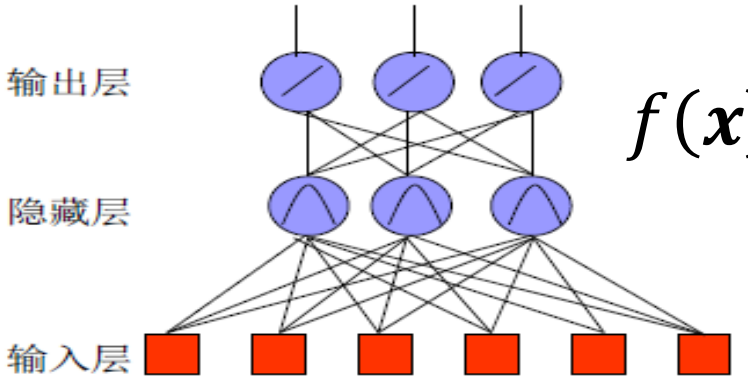
径向基(RBF)神经网络

- 1988--- D Lowe, D Broomhead

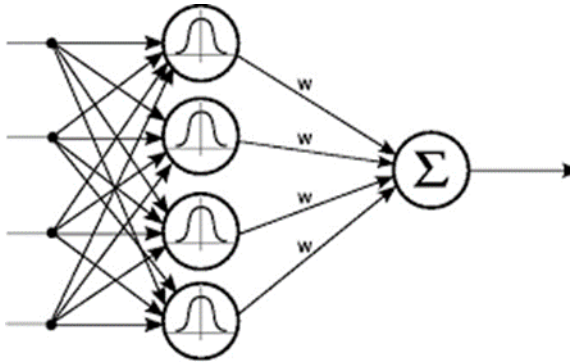
  Multivariable functional interpolation and adaptive networks（Complex systems）被引用次数：6733

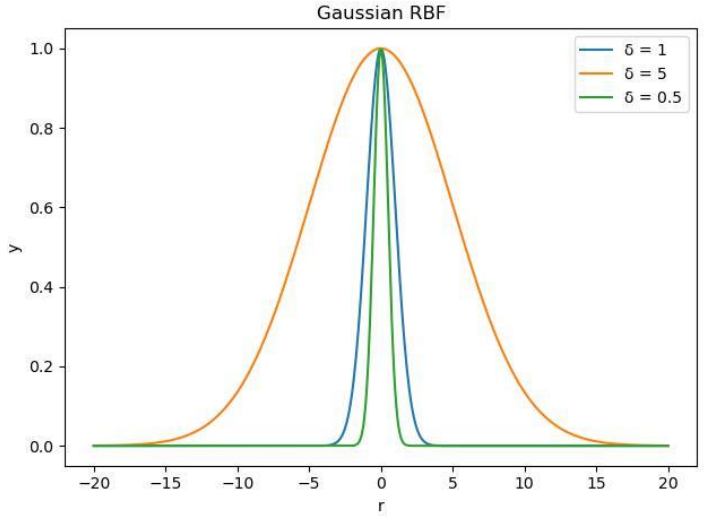
- 径向基函数网络是一种以径向基函数作为激活函数的人工神经网络。
- 网络的输出是输入和神经元参数的径向基函数的线性组合，能够以任意精度逼近任意连续函数。
- 径向基函数网络有许多用途，包括函数逼近、时间序列预测、分类和系统控制。
- 一个隐层：激活函数为径向函数，例如，高斯函数.
- 神经元的输入离该中心点越远，神经元的激活程度就越低。隐节点的这一特性常被称为“局部特性”。

径向基(RBF)神经网络



$$f(\boldsymbol{x}) = \sum_{i=1}^{m} w_m \phi(\|\boldsymbol{x} - c_m\|)$$

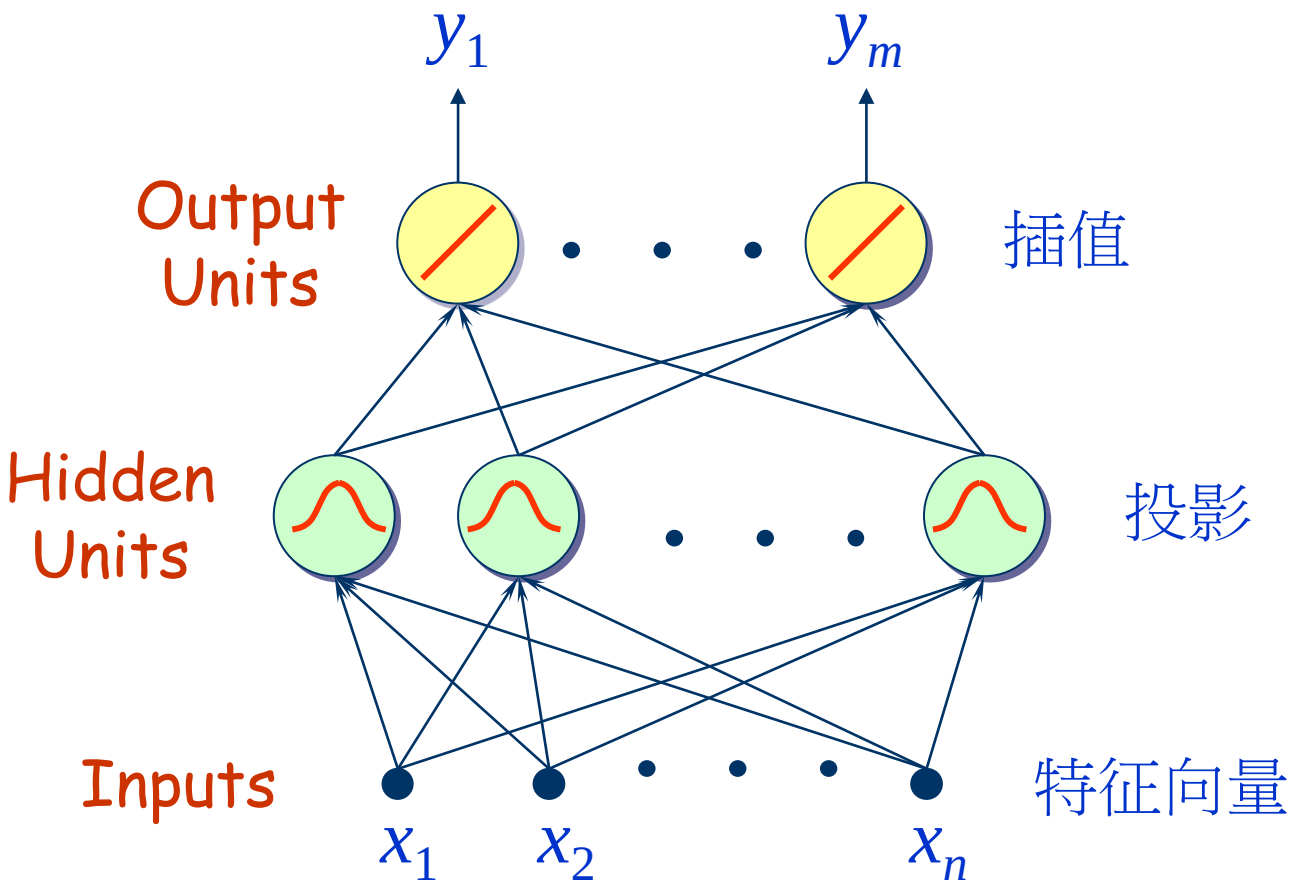




径向基(RBF)神经网络

- 函数逼近(回归问题)

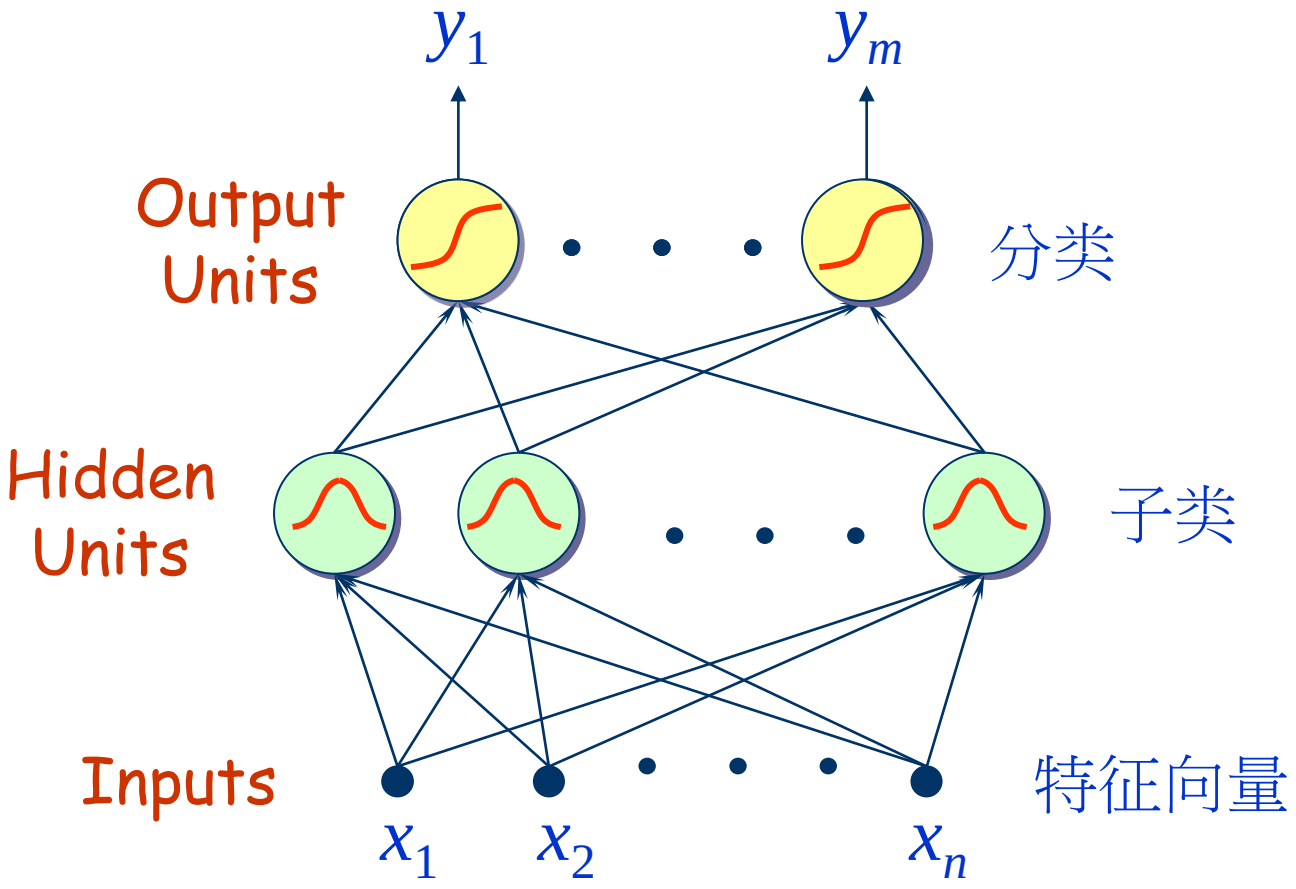
$$f(\boldsymbol{x}) = \sum_{i=1}^{m} w_m \phi(\|\boldsymbol{x} - \boldsymbol{x}_m\|)$$



径向基(RBF)神经网络

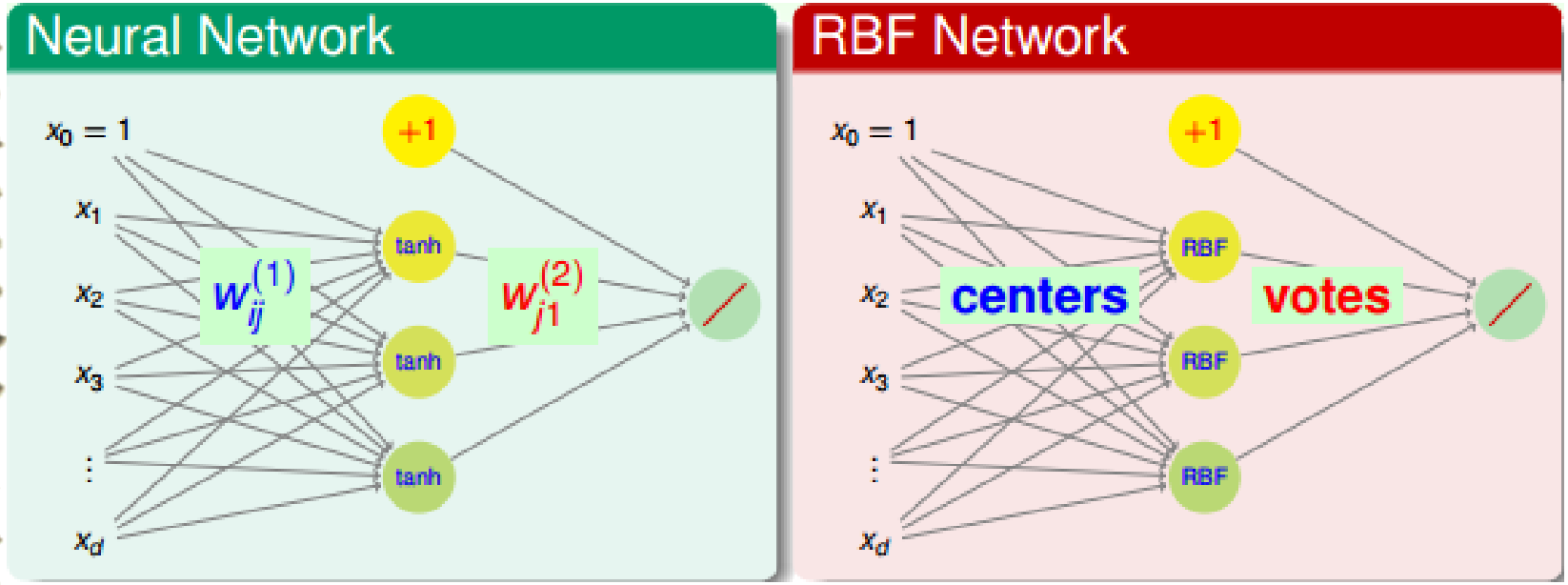
- 分类问题

$$f(\boldsymbol{x}) = \sum_{i=1}^{m} w_m \phi(\|\boldsymbol{x} - c_m\|)$$



径向基(RBF)神经网络

- RBF网络与经典神经网络的区别



- hidden layer different:
  (inner-product + tanh) versus (distance + Gaussian)
- output layer same: **just linear aggregation**

完全径向基(RBF)神经网络

- 径向基神经网络局部函数逼近
- 插值问题描述:
  - 考虑N维空间到一维空间的映射.
- 设N维空间有P个输入向量$\boldsymbol{x}_p$, $p = 1,2,\ldots,P$. 它们在输入空间相应的目标值为d_p, $p = 1,2,\ldots,P$. 插值的目的是寻找一个非线性映射函数$\mathrm{F}(\boldsymbol{x})$,使得满足下述插值条件

$$\mathrm{F}(\boldsymbol{x}_p) = d_p, p = 1,2,\ldots,P$$

完全径向基(RBF)神经网络

径向基函数解决插值问题---1985年Powell

选择P个基函数,对应每个训练数据:

$$\phi(\|\boldsymbol{x}-\boldsymbol{x}_p\|), p=1,2,\cdots,P$$

基函数的自变量为$\boldsymbol{x}$与中心$\boldsymbol{x}_p$的距离,由于距离是径向同性的,因此称径向基函数.基于径向基函数的插值函数定义为基函数的线性组合:

$$f(\boldsymbol{x})=\sum_{p=1}^{P} w_p\phi(\|\boldsymbol{x}-\boldsymbol{x}_p\|), p=1,2,\cdots,P$$

对于任意数据$\boldsymbol{x}_i$代入插值条件，可得:

$$\sum_{p=1}^{P} w_p\phi(\|\boldsymbol{x}_i-\boldsymbol{x}_p\|)=d_i, i=1,2,\cdots,P$$

完全径向基(RBF)神经网络

- 中心点为所有的样本点，ϕ为参数固定的径向基函数，因此需要求解的参数只有w_p。

令

$$\phi_{ip} = \phi\left(\left\|\boldsymbol{x}_i - \boldsymbol{x}_p\right\|\right), i, p = 1,2,\cdots,P$$

则可得到关于w_p的P阶线性方程组.

$$\begin{array}{c} \\ \text{样本1} \\ \text{样本2} \\ \\ \text{样本}P \end{array} \begin{array}{c} \begin{array}{cccc} \text{中心1} & \text{中心2} & & \text{中心}P \end{array} \\ \begin{pmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1P} \\ \phi_{21} & \phi_{22} & \cdots & \phi_{2P} \\ \vdots & \vdots & & \vdots \\ \phi_{P1} & \phi_{P2} & \cdots & \phi_{PP} \end{pmatrix} \end{array} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_P \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_P \end{pmatrix}$$

完全径向基(RBF)神经网络

写成向量形式为: $\Phi W = \boldsymbol{d}$

称Φ为插值矩阵，若其可逆，则可由上式解出W:

$$W = \Phi^{-1}\boldsymbol{d}$$

Micchelli定理给出了Φ的可逆性条件：

对于一大类函数，如果$\phi_1, \phi_2, \ldots, \phi_P$各不相同，则其可逆. 许多径向基函数满足Micchelli定理，例如

1）高斯函数:

$$\phi(r) = \exp(-\frac{r^2}{2\delta^2})$$

完全径向基(RBF)神经网络

2)Reflected Sigmoidal(反演S型)函数

$$\phi(r) = \frac{1}{1+\exp(-\frac{r^2}{\delta^2})}$$

3)Inverse multiquadrics (逆多二次)函数

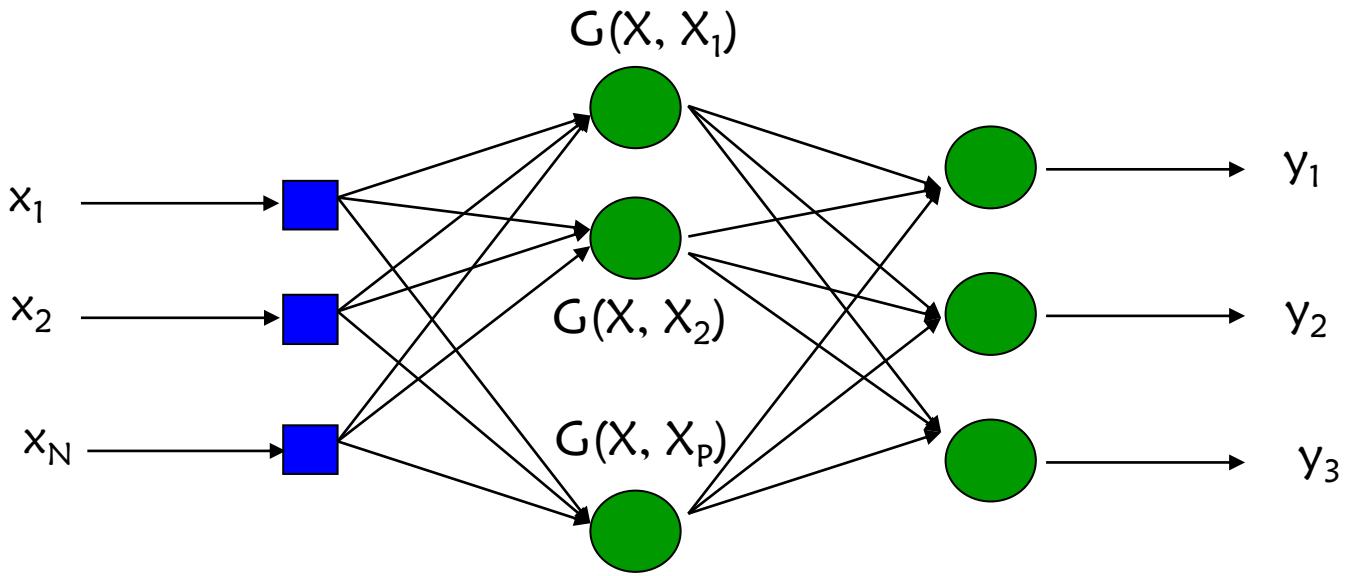
$$\phi = \frac{1}{(r^2+\delta^2)^{1/2}}$$

完全径向基(RBF)神经网络

完全内插存在的问题(正则化RBF网络):

1)经过所有训练数据点,当存在噪声时,泛化能力差

2)径向基函数数目与训练样本数相同,当训练样本数远远大于系统的固有自由度时,问题是超定的,插值矩阵求逆容易不稳定



广义径向基(RBF)神经网络

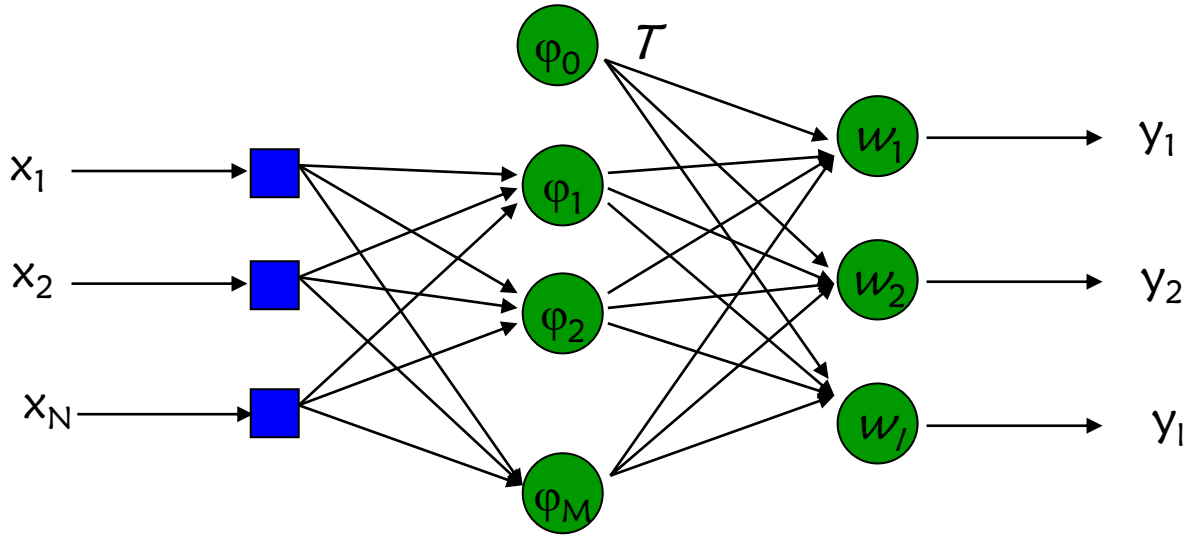
广义RBF网络

1)径向基函数数目M与训练样本数P不同,且一般M<<P

2)径向基函数的中心不再限制在数据点上,由训练确定

3)各径向基函数**的扩展常数不再统一**,由训练确定

4)输出函数的线性中包含阈值参数,用于补偿函数在样本集上的平均值与目标之平均值之间的差别

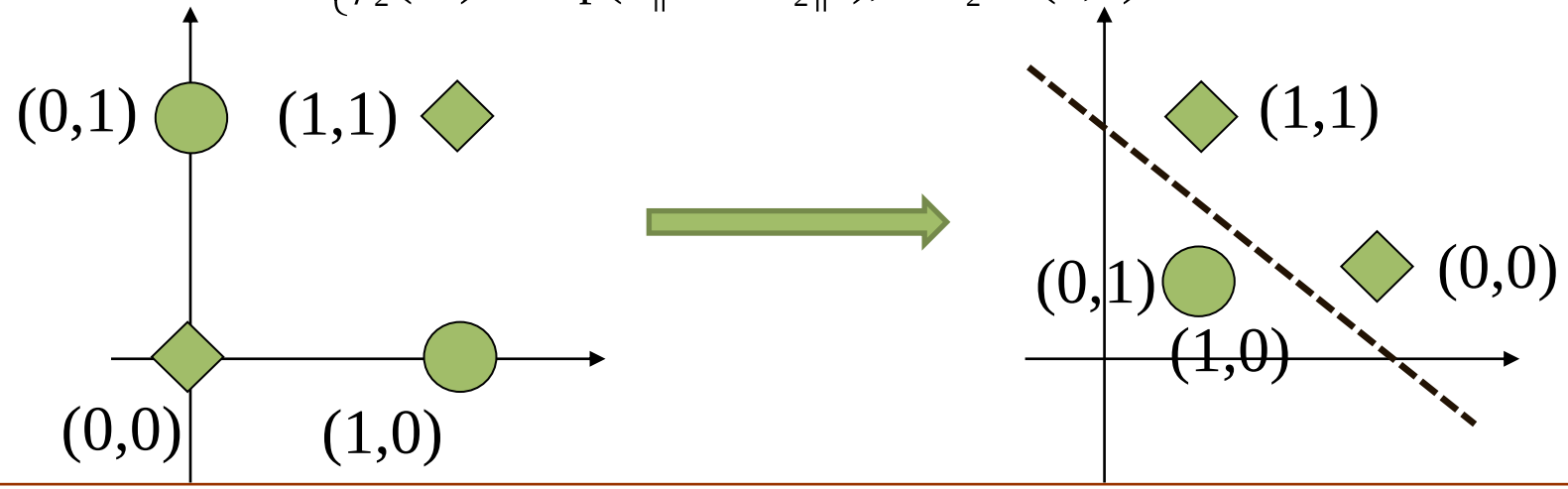


$$f(\boldsymbol{x}) = \boldsymbol{\alpha}\left(\sum_{i=1}^{m} w_m \phi(\|\boldsymbol{x} - \boldsymbol{c}_m\|)\right) \qquad \phi(r) = \exp(-\frac{r^2}{2\delta^2})$$

广义RBF网络

RBF网络将输入空间的模式点**非线性映射**到一个**高维空间**的做法是:设置一隐层,令 $\phi(X)$ 为隐节点的激活函数,并令隐节点数M大于等于输入节点数N.若M足够大,则在隐空间是线性可分的.从隐层到输出层可采用与感知器类似的解决线性可分问题的算法.如:

$$\begin{cases} \phi_1(X) = \exp(-\|X - C_1\|^2), & C_1 = (1,1) \\ \phi_2(X) = \exp(-\|X - C_2\|^2), & C_2 = (0,0) \end{cases}$$



广义径向基神经网络的 训练算法

广义RBF网络的学习算法

结构设计(多凭经验)和参数学习(3种参数:**各基函数的中心,扩展常数以及输出节点的权值**)

- 输出节点的权值一般由有监督的学习算法确定
- 各基函数的中心及扩展常数可由下列三种方法确定:

1)数据中心从样本中选取:样本密集的地方中心多些,稀疏的地方少些.若数据均匀分布,中心也可均匀分布.总之,选出的数据中心应有代表性.扩展常数由分布确定,如$\delta = d_{max}/\sqrt{2M}$, d_{max}是数据中心的最大距离,M是中心数目.

2)数据中心的聚类选择(k-means, SOM)

3)数据中心的监督学习算法

广义RBF网络的聚类学习算法

如何确定各基函数的中心？

首先估计中心的数目M,设C(k)表示第k次迭代时的中心.

1)初始化中心:$c_1(0), c_2(0), \cdots, c_M(0)$

2)计算各样本点与聚类中心的欧氏距离:

$$\|X_p - c_j(k)\|,\ p = 1,2,\cdots,P;\ j = 1,2,\cdots,M$$

3)相似匹配: 当$\|X_p - c_{j^*}(k)\| = \min_j \|X_p - c_j(k)\|,\ p = 1,2,\cdots,P$时，$X_p$被归为第$j^*$类.

4)更新各类聚类中心

(i) 均值方法(k-means) $c_j(k+1) = \dfrac{1}{N_j} \sum_{X \in U_j(k)} X$

(ii) 竞争学习算法(SOM) $c_j(k+1) = \begin{cases} c_j(k) + \eta[X_p - c_j(k)] & j = j^* \\ c_j(k) & j \neq j^* \end{cases}$

广义RBF网络的聚类学习算法

5) k++，若不满足终止条件（C(k)的改变两小于阈值），则转2

扩展常数的确定：设 $d_j = \min_i \|c_j - c_i\|$（所有中心点最小距离），则扩展常数可取为 $\delta_j = \lambda \cdot d_j$

输出层权值的确定：

(i) 最小均方算法(类似线性回归)

(ii) 伪逆法：令 $\varphi_{pj} = \varphi(\|X_p - c_j\|),\ p = 1,2,\cdots,P;\ j = 1,2,\cdots,M$

则隐层输出矩阵为 $\hat{\Phi} = (\varphi_{pj})_{P\times M}$，

令 $W = (w_1, w_2, \cdots, w_M)$，则

$$F(X) = \hat{\Phi}\ W = d \longrightarrow W = \hat{\Phi}^+\ d$$

$$\hat{\Phi}^+ = (\hat{\Phi}^T\hat{\Phi})^{-1}\hat{\Phi}^T$$

广义RBF网络的数据中心的监督学习算法

- 广义RBF网络的数据中心的监督学习算法

类似BP算法的梯度下降方法(假定单输出):

$$E=\frac{1}{2}\sum_{i=1}^{P}e_i^{\,2}=\frac{1}{2}\sum_{i=1}^{P}(d_i-F(x_i)^2=\frac{1}{2}\sum_{i=1}^{P}\left(d_i-\sum_{j=1}^{M}w_jG(\|X_i-c_i\|)\right)^2$$

$$\Delta c_j=-\eta\frac{\partial E}{\partial c_j}\qquad \Delta\delta_j=-\eta\frac{\partial E}{\partial \delta_j}\qquad \Delta w_j=-\eta\frac{\partial E}{\partial w_j}$$

$$G(r)=\exp(-\frac{r^2}{2\delta^2})$$

$$\Delta c_j=\eta\frac{w_j}{\delta_j^2}\sum_{i=1}^{P}e_iG(\|X_i-c_j\|)(X_i-c_j)$$

$$\Delta\delta_j=\eta\frac{w_j}{\delta_j^3}\sum_{i=1}^{P}e_iG(\|X_i-c_j\|)\|X_i-c_j\|^2$$

$$\Delta w_j=\eta\sum_{i=1}^{P}e_iG(\|X_i-c_j\|)$$

广义RBF网络的数据中心的监督学习算法

类似BP算法的梯度下降方法(每个数据修正一次):

$$E = \frac{1}{2}e^2$$

$$\Delta c_j = \eta \frac{w_j}{\delta_j^2} eG\left(\left\|X - c_j\right\|\right)(X - c_j)$$

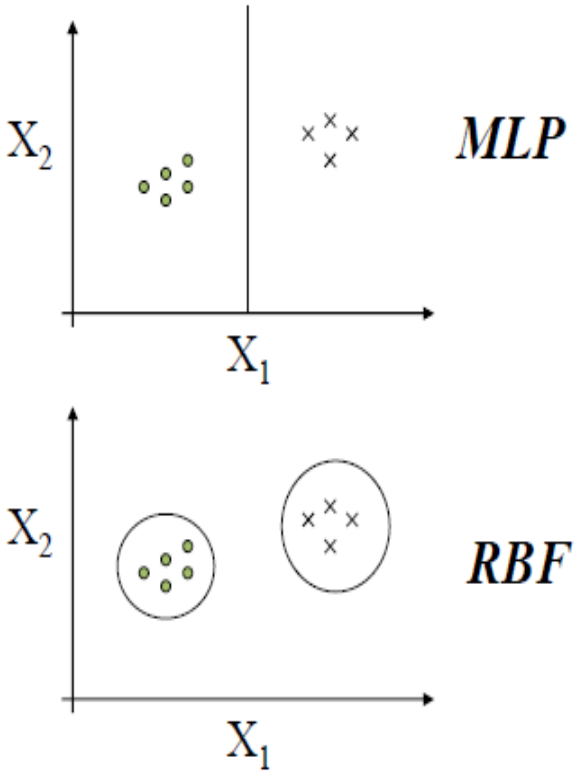
$$\Delta \delta_j = \eta \frac{w_j}{\delta_j^3} eG\left(\left\|X - c_j\right\|\right)\left\|X - c_j\right\|^2$$

$$\Delta w_j = \eta eG\left(\left\|X - c_j\right\|\right)$$

广义径向基神经网络与多层感知器比较

RBF网络与多层感知器MLP的比较

- **Classification**
  - –MLPs 通过**超平面**分类
  - –RBFs 通过**超球体**分类
- **Learning**
  - –MLPs 使用全局化学习
  - –RBFs 使用局部化学习
  - –RBFs 训练更快
  - – RBFs： 局部近似快速学习
- **Structure**
  - –MLPs 有一个或多个隐藏层
  - –RBFs 只有一个隐藏层
  - –RBFs 需要更多的隐藏神经元 => 维数灾难



RBF网络与多层感知器MLP的比较

- **Relationship**
  - RBF:隐层为非线性，输出层为线性或非线性。
  - MLP:隐层和输出层对于分类是非线性的，而输出层对于非线性回归是线性的。
- **Solution**
  - RBF: 求解析解简单
  - MLP: 求解析解困难

径向基函数神经网络用于分类

RBFN for classification

- 通过测量输入与训练集样本的相似度 。
- 每个RBFN神经元存储一个代表，它只是训练集中的一个例子。
- 每个RBF神经元计算输入与其代表向量(取自训练集)之间的相似性度量。
- 当对新输入进行分类时，每个神经元计算输入与其代表之间的欧几里得距离，输出一个介于0到1之间的值，作为相似性的度量。
- 相似函数的选择==激活函数的选择。
- 粗略地说，如果输入更接近于A类代表而不是B类代表，则将其归类为A类。

径向基函数神经网络应用实例：函数逼近

应用实例：函数逼近

- 构造一个RBF网络类

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

class RBFnetwork(object):
    def __init__(self, hidden_nums, r_w, r_c, r_sigma):
        self.h = hidden_nums          #隐含层神经元个数
        self.w = 0                    #线性权值
        self.c = 0                    #神经元中心点
        self.sigma = 0                #高斯核宽度
        self.r = {"w":r_w,
                  "c":r_c,
                  "sigma":r_sigma}    #参数迭代的学习率
        self.errList = []             #误差列表
        self.n_iters = 0              #实际迭代次数
        self.tol = 1.0e-5             #最大容忍误差
        self.X = 0                    #训练集特征
        self.y = 0                    #训练集结果
        self.n_samples = 0            #训练集样本数量
        self.n_features = 0           #训练集特征数量
```

应用实例：函数逼近

- 定义训练过程中用到的函数

```
#计算径向基距离函数
def guass(self, sigma, X, ci):
    return np.exp(-np.linalg.norm((X-ci), axis=1)**2/(2*sigma**2))

#将原数据高斯转化成新数据
def change(self, sigma, X, c):
    newX = np.zeros((self.n_samples, len(c)))
    for i in range(len(c)):
        newX[:,i] = self.guass(sigma[i], X, c[i])
    return newX

#初始化参数
def init(self):
    sigma = np.random.random((self.h, 1))                #(h,1)
    c = np.random.random((self.h, self.n_features))      #(h,n)
    w = np.random.random((self.h+1, 1))                  #(h+1,1)
    return sigma, c, w
```

应用实例：函数逼近

- 定义训练过程中用到的函数

```
#给输出层的输入加一列截距项
def addIntercept(self, X):
    return np.hstack((X,np.ones((self.n_samples,1))))

#计算整体误差
def calSSE(self, prey, y):
    return 0.5*(np.linalg.norm(prey - y))**2

#求L2范数的平方
def l2(self, X, c):
    m,n = np.shape(X)
    newX = np.zeros((m, len(c)))
    for i in range(len(c)):
        newX[:,i] = np.linalg.norm((X-c[i]), axis=1)**2
    return newX
```

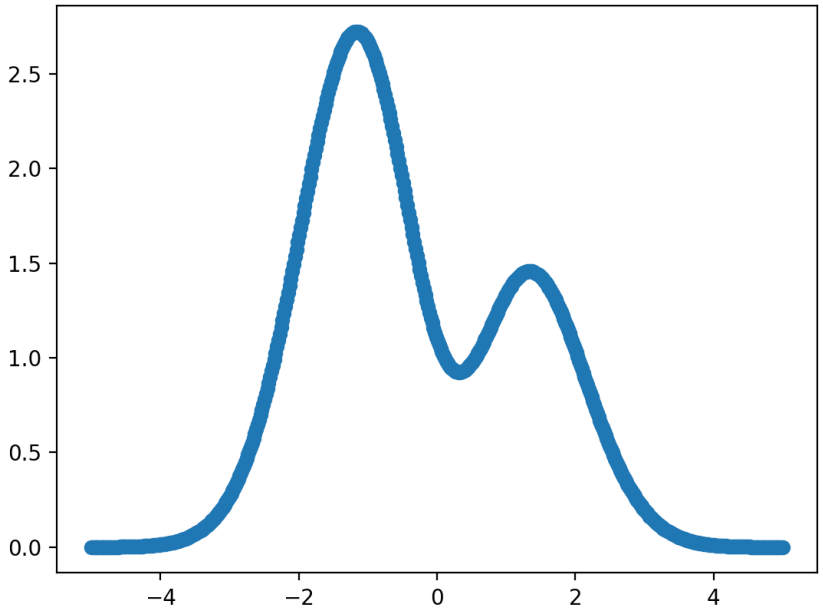
应用实例：函数逼近

- 训练函数

```
#训练
def train(self, X, y, iters):
    self.X = X
    self.y = y.reshape(-1,1)
    self.n_samples, self.n_features = X.shape
    sigma, c, w = self.init()                          #初始化参数
    for i in range(iters):
        ##正向计算过程
        hi_output = self.change(sigma,X,c)             #隐含层输出(m,h)，即通过径向
        yi_input = self.addIntercept(hi_output)        #输出层输入(m,h+1)，因为是线
        yi_output = np.dot(yi_input, w)                #输出预测值(m,1)
        error = self.calSSE(yi_output, y)              #计算误差
        if error < self.tol:
            break
        self.errList.append(error)                     #保存误差
        ##误差反向传播过程
        deltaw = np.dot(yi_input.T, (yi_output-y))     #(h+1,m)x(m,1)
        w -= self.r['w']*deltaw/self.n_samples
        deltasigma = np.divide(np.multiply(np.dot(np.multiply(hi_output,self.l2(X,
                    (yi_output-y)), w[:-1]), sigma**3)  #(h,m)x(m,1)
        sigma -= self.r['sigma']*deltasigma/self.n_samples
        deltac1 = np.divide(w[:-1],sigma**2)           #(h,1)
        deltac2 = np.zeros((1,self.n_features))                    #(1,n)
        for j in range(self.n_samples):
            deltac2 += (yi_output-y)[j]*np.dot(hi_output[j], X[j]-c)
        deltac = np.dot(deltac1,deltac2)               #(h,1)x(1,n)
        c -= self.r['c']*deltac/self.n_samples
    self.c = c
    self.w = w
    self.sigma = sigma
    self.n_iters = i
```

应用实例：函数逼近

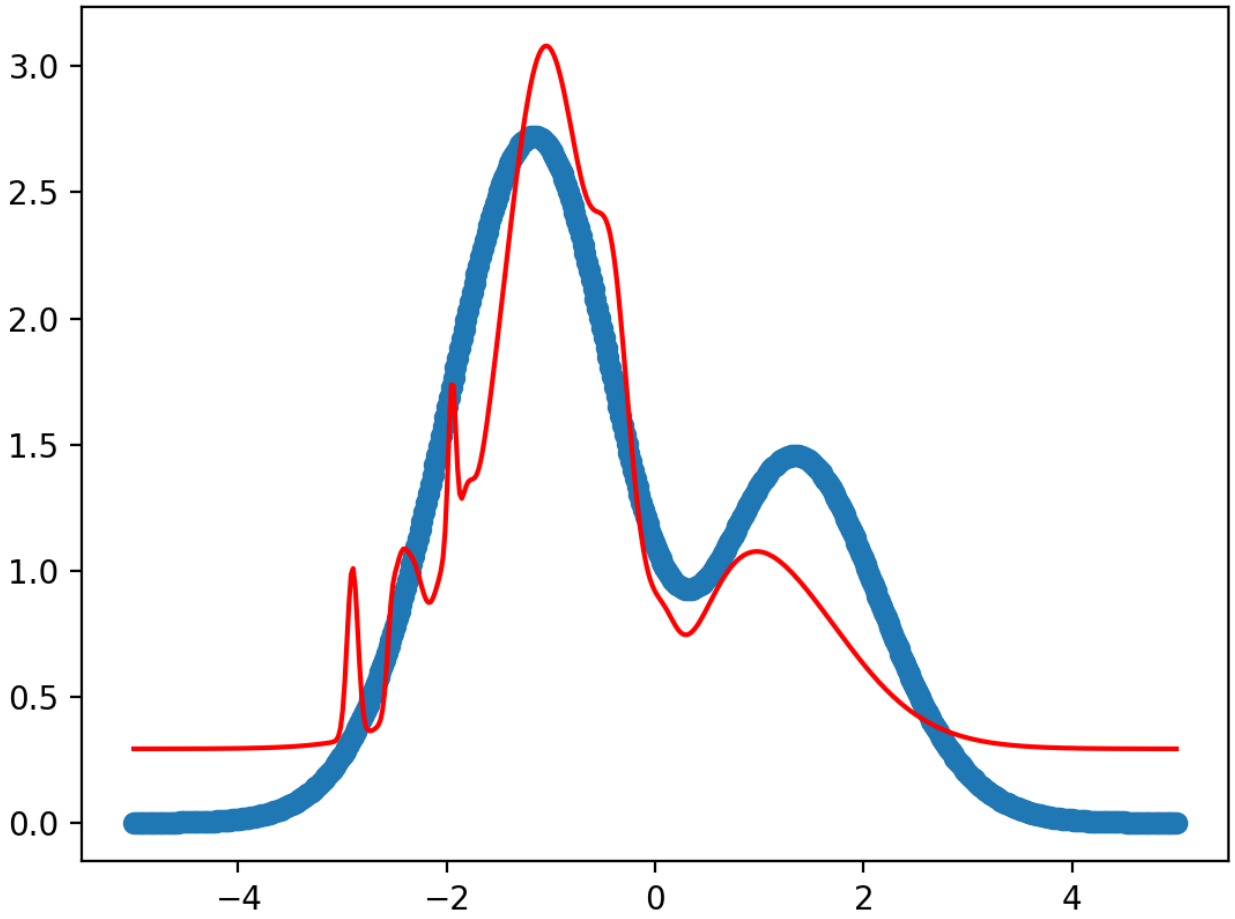
- 拟合的是Hermit多项式




```
if __name__ == "__main__":
    #拟合Hermit多项式
    X = np.linspace(-5, 5 , 500)[:, np.newaxis]
    y = np.multiply(1.1*(1-X+2*X**2),np.exp(-0.5*X**2))
    rbf = RBFnetwork(50, 0.1, 0.2, 0.1)
    rbf.train(X, y, 1000)
```

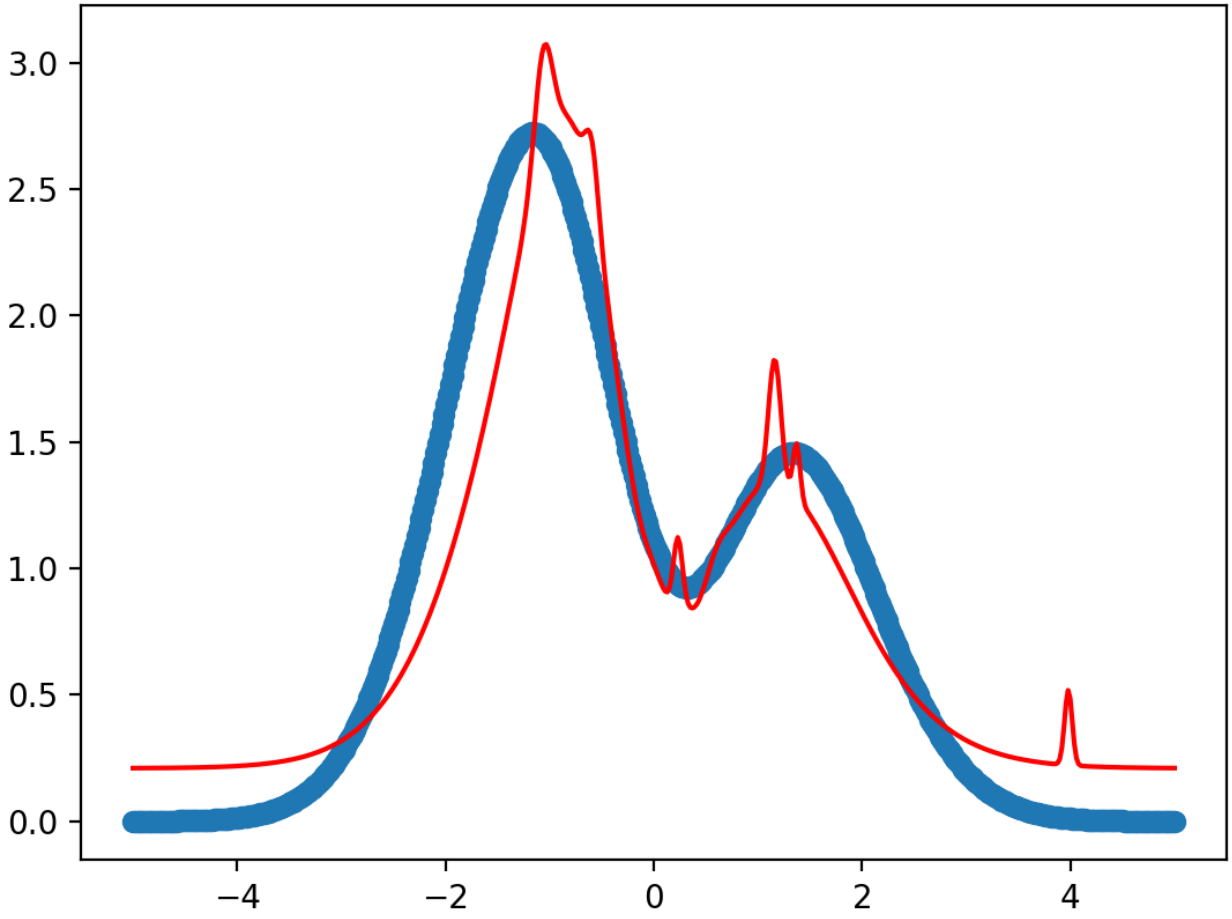
应用实例：函数逼近

- 第0轮迭代:



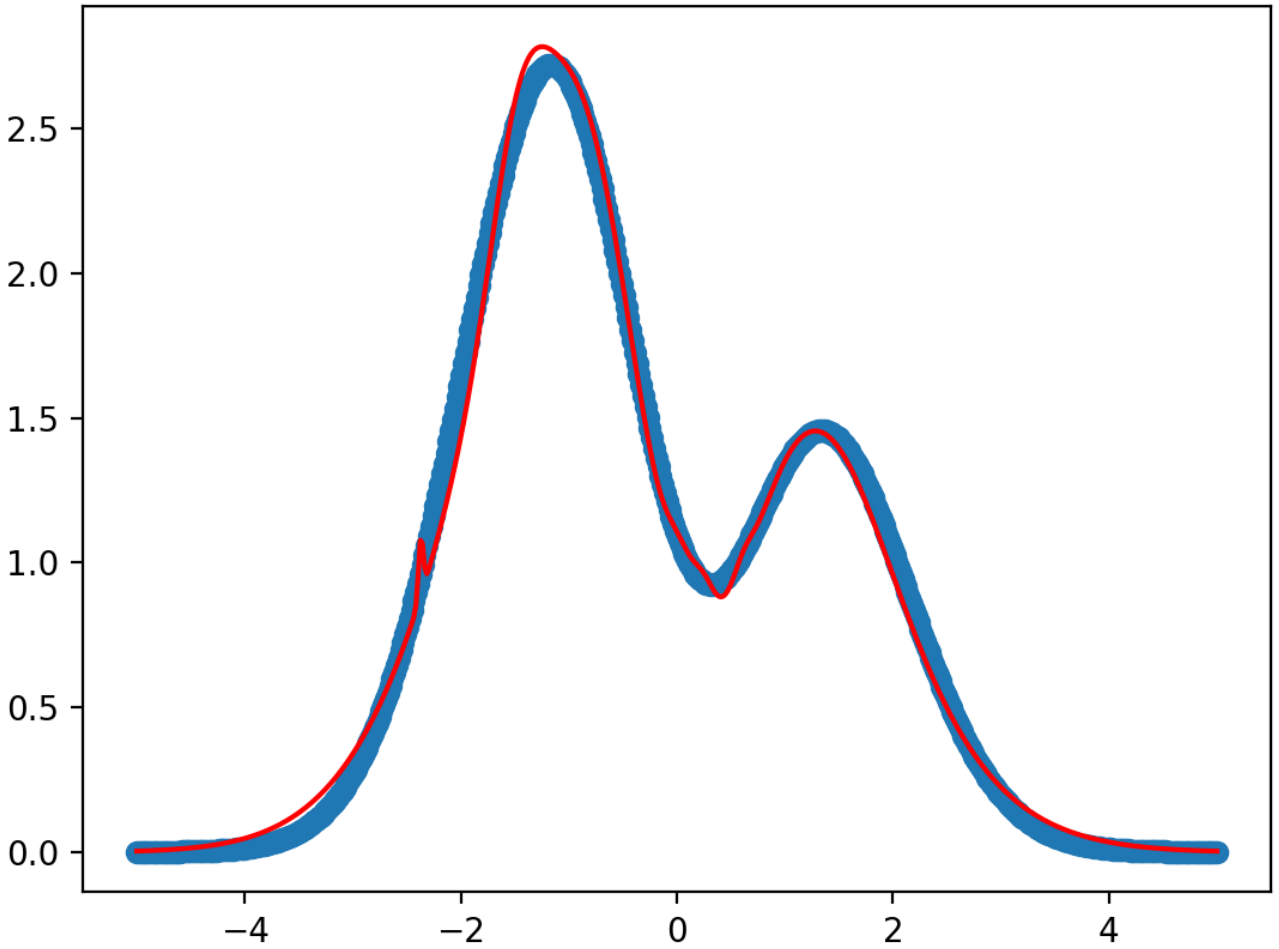
应用实例：函数逼近

- 第100轮迭代:



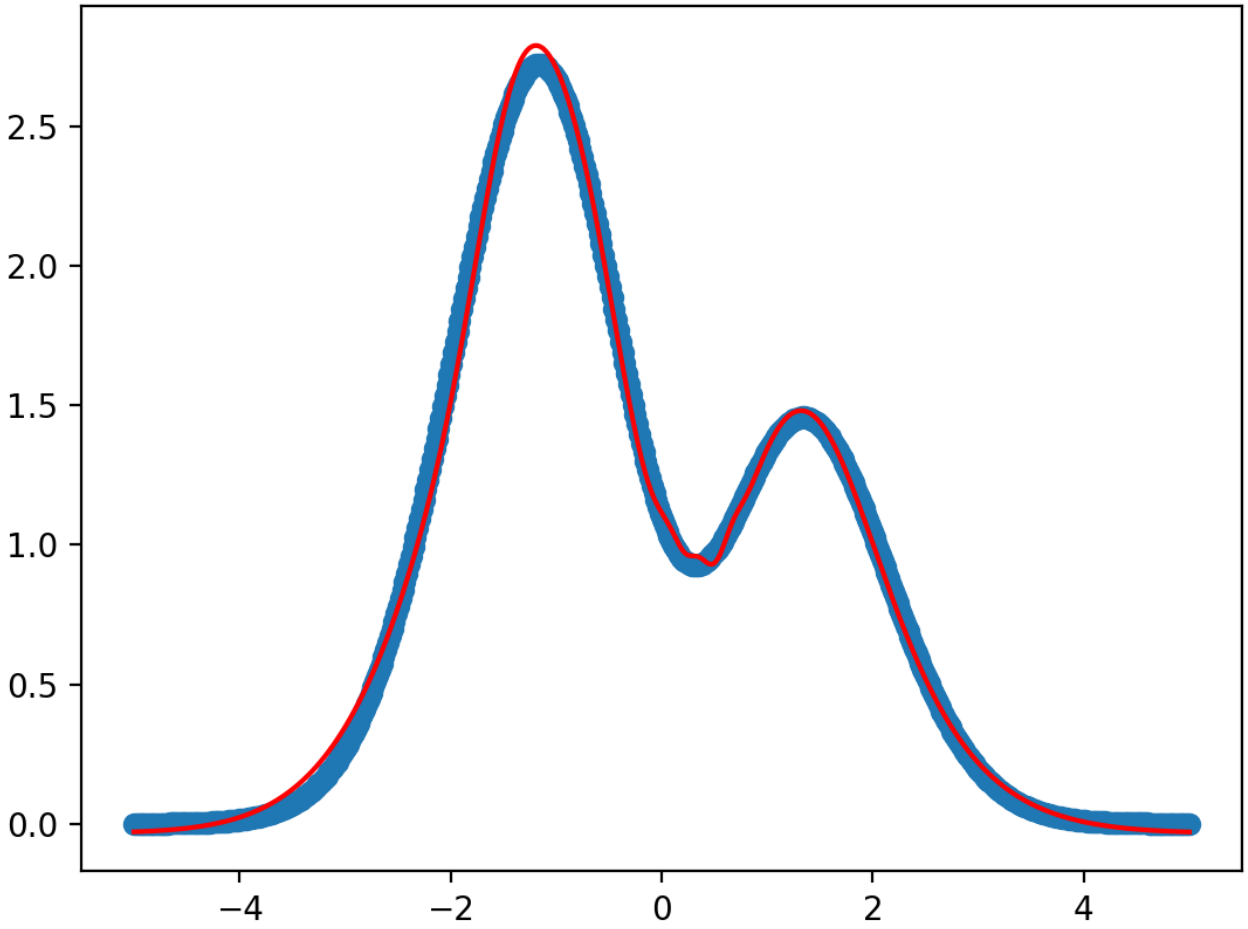
应用实例：函数逼近

- 第500轮迭代：



应用实例：函数逼近

- 第1000轮迭代：



自组织特征映射神经网络

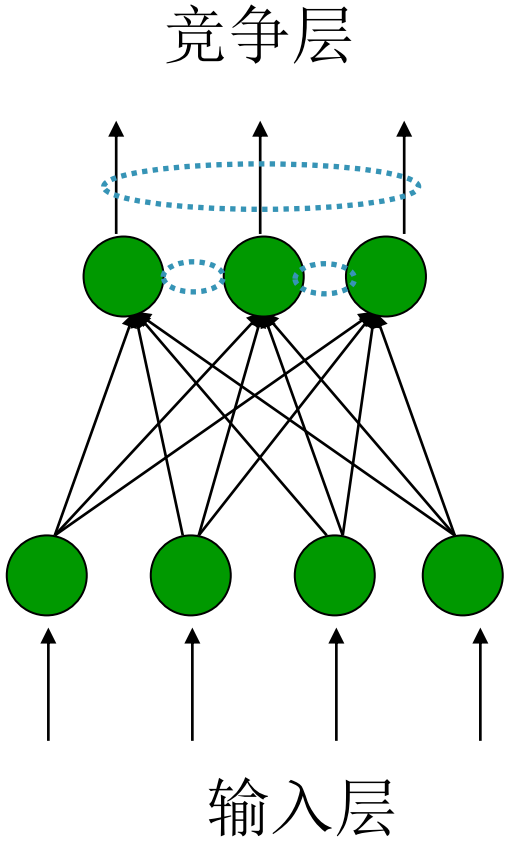
自组织特征映射神经网络

- 自组织映射(SOM)或自组织特征映射(SOFM)由Kohonen教授于1981年提出，是一种无监督机器学习技术，用于生成高维数据集的低维(通常是二维)表示，同时保留数据的拓扑结构。
- 不同于一般神经网络基于损失函数的反向传递来训练，它运用竞争学习(competitive learning)策略,依靠神经元之间互相竞争逐步优化网络。且使用近邻关系函数(neighborhood function)来维持输入空间的拓扑结构。

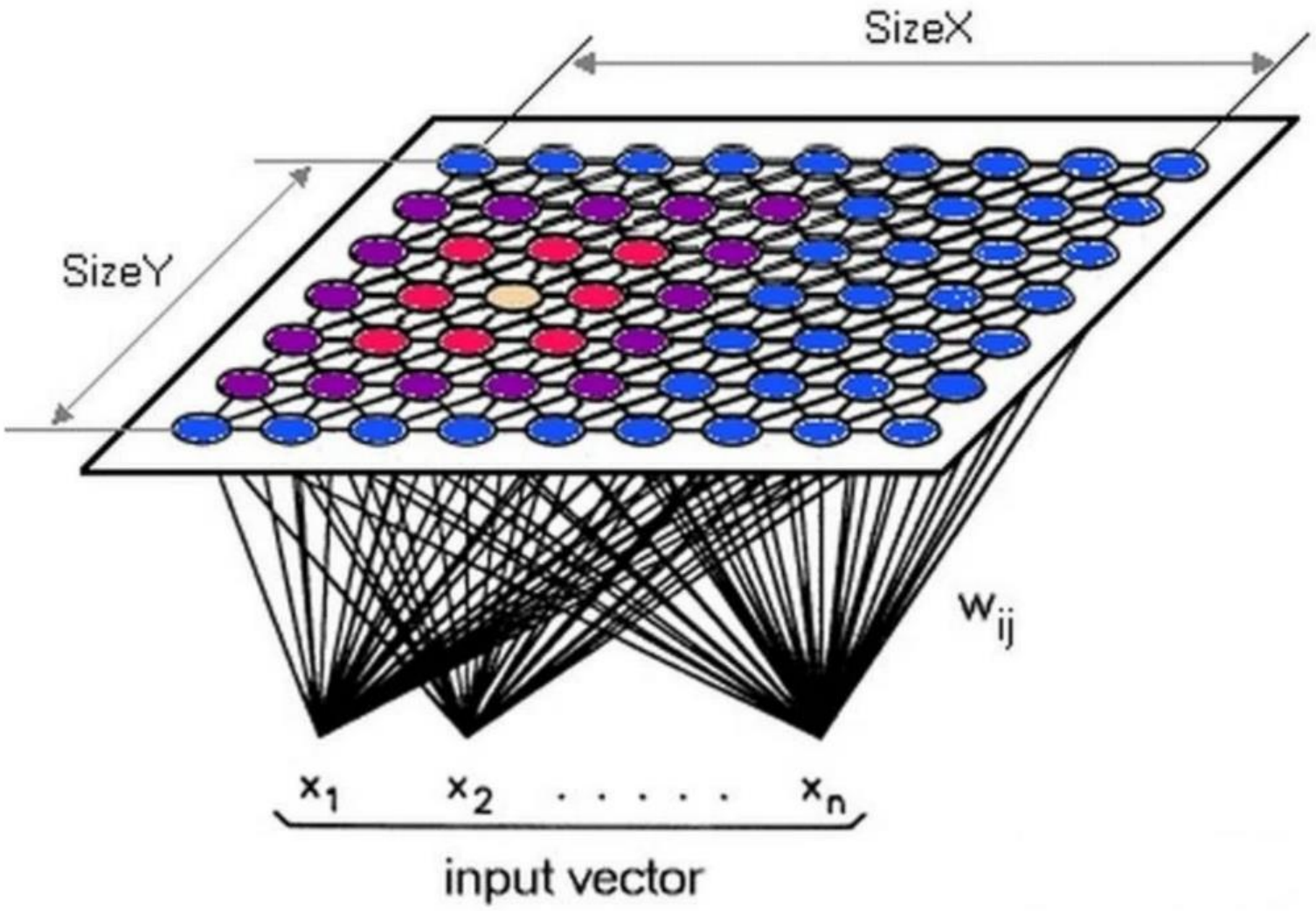
自组织特征映射神经网络

主要特点：

- 神经网络，竞争学习策略
- 无监督学习，不需要额外标签
- 非常适合高维数据的可视化，能够维持输入空间的拓扑结构
- 具有很高的泛化能力，它甚至能识别之前从没遇过的输入样本
- SOM的权重存储了竞争层每个节点的特征



自组织特征映射神经网络



自组织特征映射神经网络

竞争学习策略

侧抑制与竞争：在生物的神经细胞中存在一种侧抑制现象，即当一个神经细胞兴奋后，会对其周围的神经细胞产生抑制作用，从而产生竞争。开始时可能多个细胞同时兴奋，但一个兴奋程度最强的神经细胞对周围神经细胞的抑制作用也越强，其结果使周围细胞兴奋度减弱。这种抑制作用一般满足某种分布关系。最简单是“胜者为王”。

自组织特征映射神经网络

- Kohonen规则聚类

1) 先随机初始化k个聚类中心点；
2) 然后每次选出一个样本，将离它最近的聚类中心往它移动，使该聚类中心更靠近它，如此反复m次，具体更新法则如下：

$$w_k = w_k + \mathrm{lr} * (x - w_k)$$

其中

w_k：离样本最近的聚类中心点

lr：学习率

自组织特征映射神经网络

- “胜者为王”学习策略

1）向量归一化：$X \longrightarrow \hat{X}, w_j \longrightarrow \hat{w}_j$

2）寻找获胜神经元（欧式距离）：

$$\|\hat{X} - \hat{w}_{j^*}\| = \min_{1 \le j \le m} \{\|\hat{X} - \hat{w}_j\|\}$$

$$\|\hat{X} - \hat{W}_{j*}\| = \sqrt{(\hat{X} - \hat{W}_{j*})^T (\hat{X} - \hat{W}_{j*})} = \sqrt{\hat{X}^T \hat{X} - 2\hat{W}_{j*}{}^T \hat{X} + \hat{W}_{j*}{}^T \hat{W}_{j*}} = \sqrt{2(1 - \hat{W}_{j*}{}^T \hat{X})}$$

等价于求最大点积问题（正是竞争层神经元的输入）

$$\hat{W}_{j*}{}^T \hat{X} = \max_{j \in \{1,2,\cdots,m\}} (\hat{W}_j{}^T \hat{X})$$

3）网络输出与权值调整

$$\begin{cases} W_{j*}(t+1) = \hat{W}_{j*}(t) + \alpha\left(\hat{X} - \hat{W}_{j*}\right) \\ W_j(t+1) = W_j(t) \qquad\qquad j \neq j* \end{cases}$$

$$O_j(t+1) = \begin{cases} 1 & j = j* \\ 0 & j \neq j* \end{cases}$$

自组织特征映射神经网络

自组织映射神经网络(SOFM)的学习算法:

1)权重初始化: $W_j \to \hat{W}_j$, $j=1,2,\cdots,m$; $N_{j^*}(0)$; $\eta(0)$

2)样本选择: 从训练集中随机选取一个输入样本并归一化

$$X^p \to \hat{X}^p, \quad p \in \{1,2,\cdots,P\}$$

3)寻找获胜神经元（竞争过程）: $\hat{W}_{j^*}{}^T \hat{X}^p = \max_{j\in\{1,2,\cdots,m\}} (\hat{W}_j{}^T \hat{X}^p)$

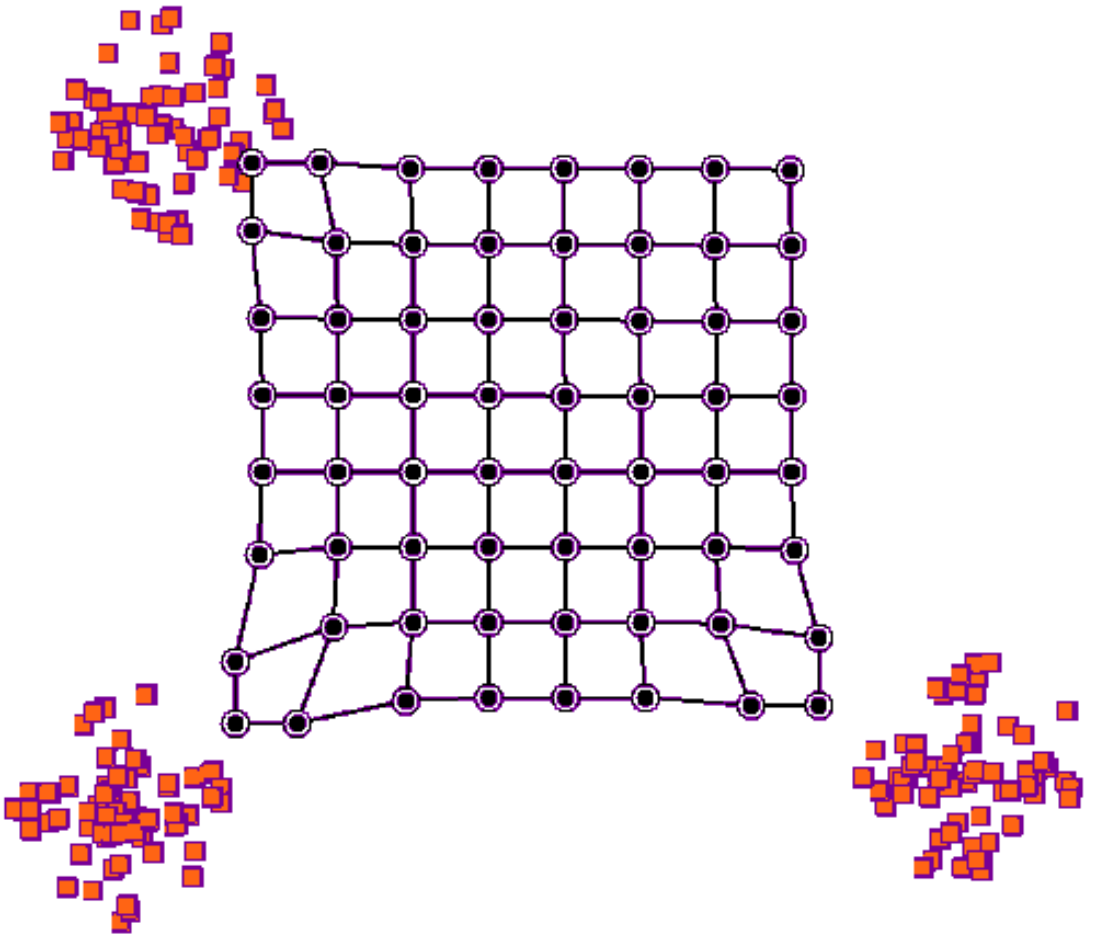
4)定义优胜邻域(自组织过程) $N_{j^*}(t)$: 一般初始时较大,以后逐渐收缩, 可正方形, 六角形等

5)调整权值: $w_{ij}(t+1) = w_{ij}(t) + \eta(t,N)[x_i^p - w_{ij}(t)], \quad i=1,2,\cdots,n;\ j \in N_{j^*}(t)$

$\eta(t,N)$: $t\uparrow \to \eta\downarrow, N\uparrow \to \eta\downarrow$ 如: $\eta(t,N) = \eta(t)e^{-N}$

6)结束检查: $\eta(t)$是否衰减到足够小(不存在输出误差概念)

自组织特征映射神经网络



自组织特征映射神经网络

SOM学习算法实现细节：

1) 权重初始化：

- 随机初始化 Random initialization
- 随机样本初始化 Initialization using initial samples
- 线性初始化 Linear initialization

2) 计算优胜节点：

- 优胜节点是指与样本特征相似度最高(距离最近)的竞争层节点。在预处理输入样本时可以考虑进行标准化(均值为0，标准差为1)；这样有助于使每个特征对于计算相似度的贡献相同。

自组织特征映射神经网络

SOM学习算法实现细节：

3) 计算邻域函数：

- 邻域函数用来确定优胜节点对其余所有节点的影响，以权重的形式体现。距离优胜节点越近的节点对应的权重越高，距离优胜节点越远的节点对应的权重越低。权重影响更新幅度，因此优胜邻域内的节点具有较大的更新程度。常用的邻域函数包括：

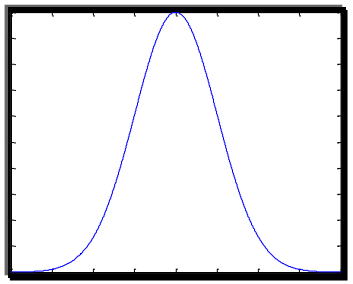
① Gaussian函数；
② Bubble函数。

自组织特征映射神经网络

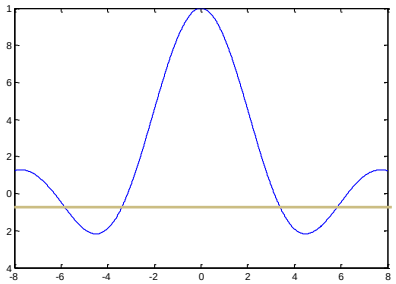
SOM学习算法实现细节：

3) 计算邻域函数：

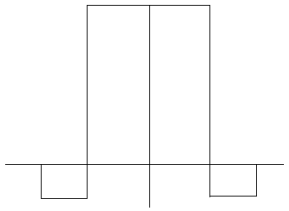
权值调整域: 由近及远, 由兴奋转为抑制



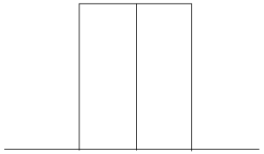
正态分布型



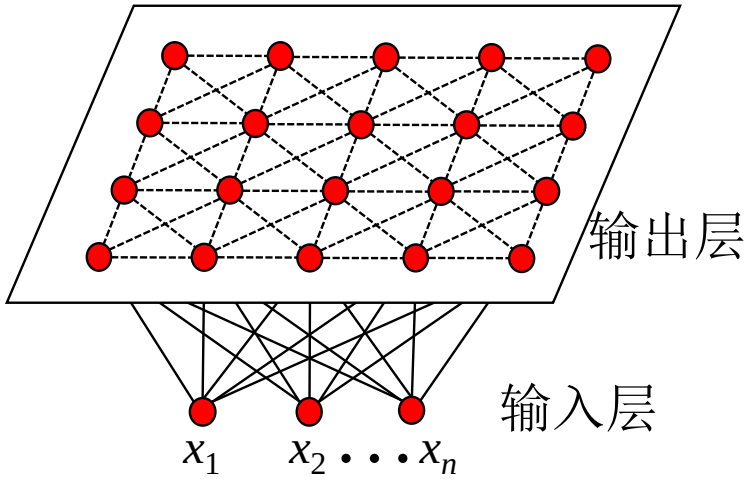
墨西哥帽型



大礼帽型



厨师帽型



自组织特征映射神经网络

SOM学习算法实现细节：

4) 学习率和邻域半径的衰减：

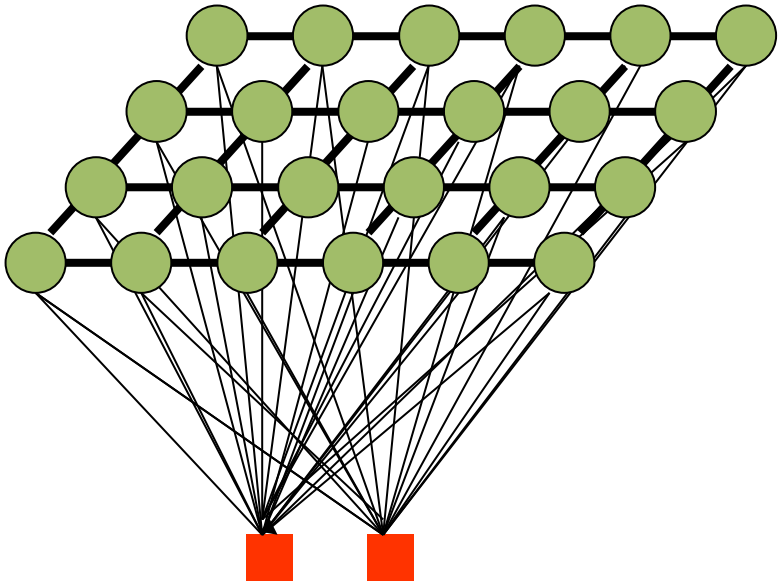
随着迭代次数的增大，学习率η应该逐步减少，以促进模型的收敛。邻域半径σ也应该逐渐减小，其目的是更精细地调整权重。

若记T为总迭代次数，t为当前迭代次数，则参数衰减公式表示为：

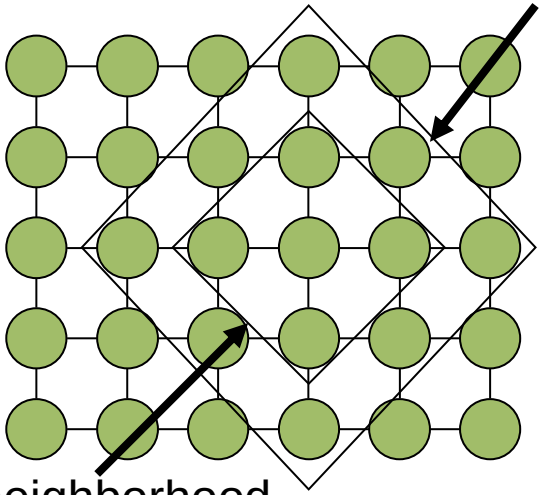
$$\eta = \frac{\eta_0}{1 + \frac{t}{T/2}}, \quad \sigma = \frac{\sigma_0}{1 + \frac{t}{T/2}}$$

领域参数设置：

- 神经元的激活在其直接邻域内扩散=>邻域对相同的输入模式变得更灵敏
- 邻居的规模最初很大，但随着时间的推移而减少=>网络特异性



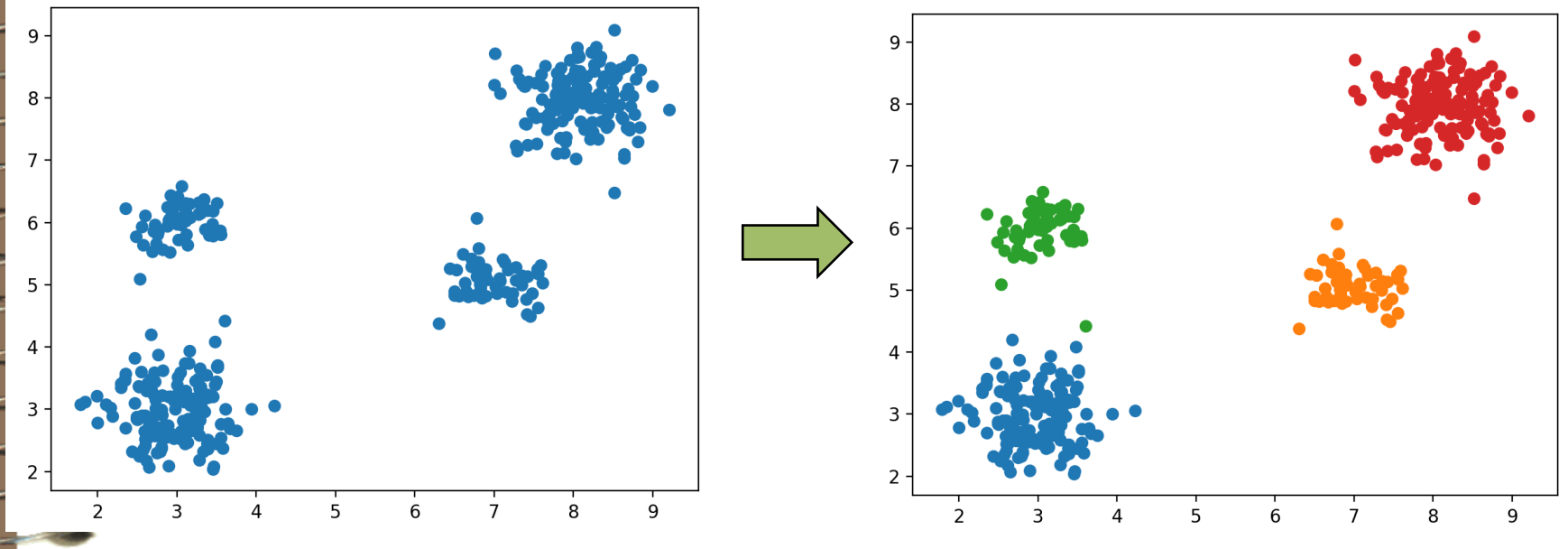
2nd neighborhood



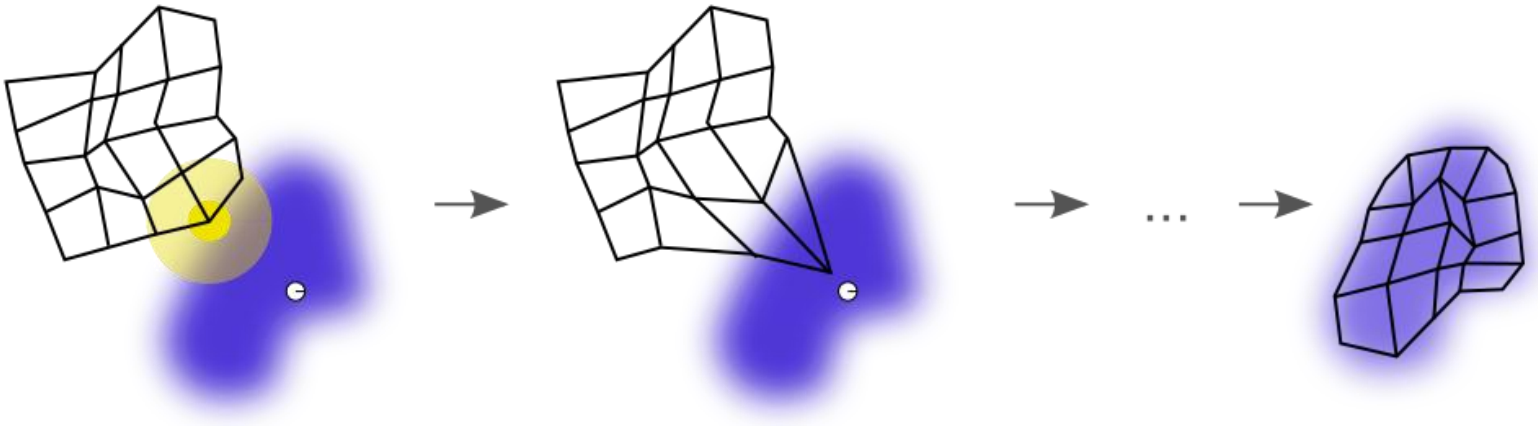
First neighborhood

自组织特征映射神经网络

自组织映射神经网络(SOFM)用于聚类:



自组织特征映射神经网络



训练自组织映射的图示。蓝色的区域是训练数据的分布，白色的圆点是从该分布中提取的当前训练数据。首先(左)SOM节点在数据空间中的位置是任意的。选择最接近训练数据的节点(以黄色突出显示)。它被移向训练基准，就像它在网格上的邻居一样(在较小程度上)。经过多次迭代，网格趋向于近似数据分布(右)。

https://en.wikipedia.org/wiki/Self-organizing_map

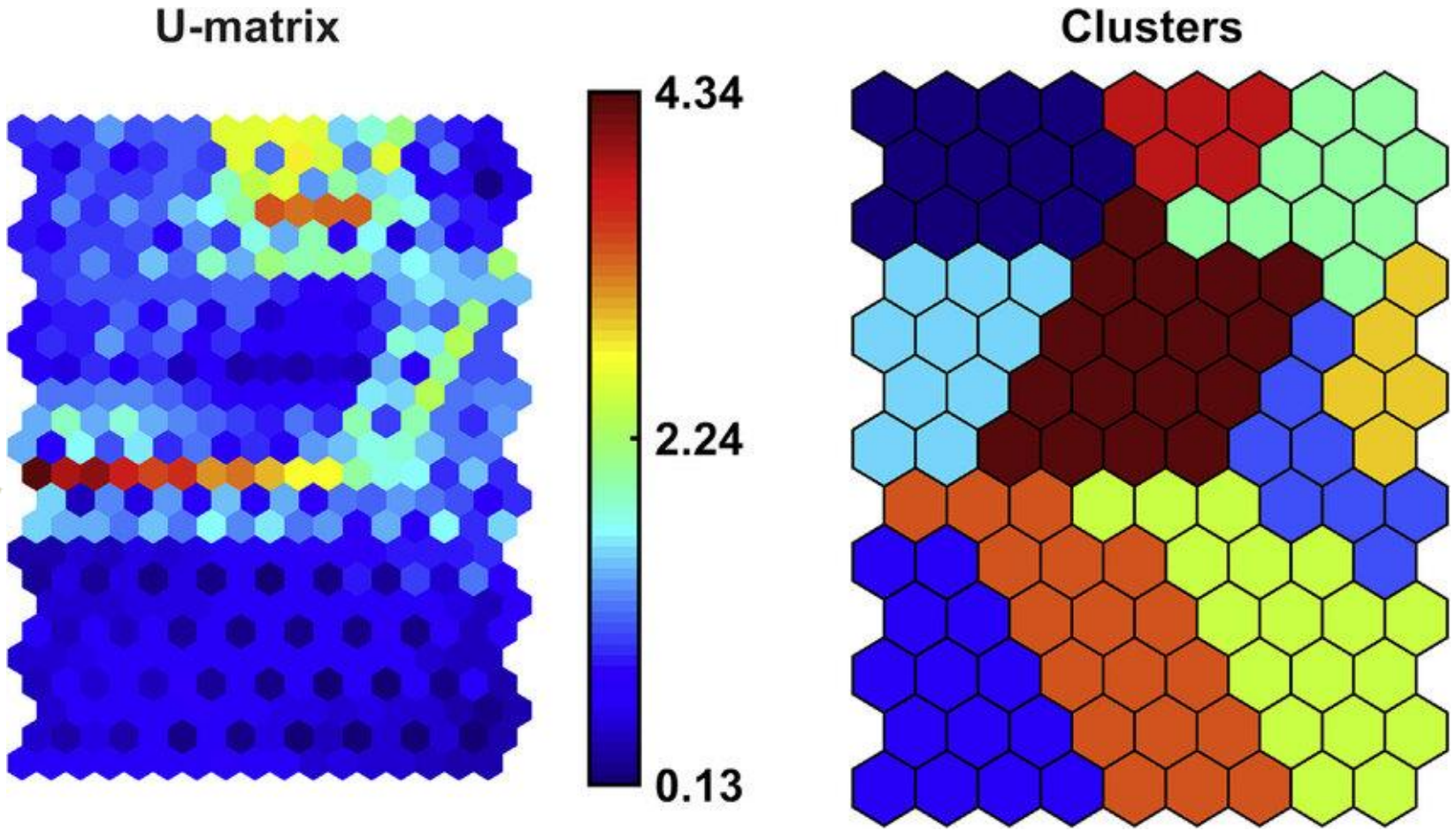
U-Matrix

- 在自组织映射（Self-Organizing Maps，简称SOM）中，U-Matrix（Unified Distance Matrix，统一距离矩阵）是一个用于描述映射中相邻单元之间距离关系的矩阵。它提供了关于SOM拓扑结构的重要信息，有助于揭示数据在映射上的分布和聚类情况。
- 具体来说，U-Matrix中的每个元素代表SOM网格中相邻单元之间的平均距离。这些距离可以是欧几里得距离、曼哈顿距离或其他合适的度量。通过可视化U-Matrix，研究人员可以观察到在SOM中哪些单元是紧密相邻的，以及它们之间的距离变化。通常，U-Matrix的可视化使用颜色编码来表示距离的大小，从而形成一个直观的拓扑图。
- U-Matrix在自组织映射中的应用广泛，它有助于：

1. **识别聚类**：通过观察U-Matrix中颜色或灰度级的变化，可以轻易地识别出数据在SOM上的聚类区域。聚类区域通常对应于U-Matrix中颜色较为一致或平滑的区域。
2. **发现边界**：U-Matrix中的高值区域通常表示数据在SOM上的边界或分隔区域。这些边界可以帮助用户理解数据的分布和拓扑结构。
3. **解释和解释SOM**：通过U-Matrix，研究人员可以更好地解释和解释SOM的输出结果。它提供了一个直观的方式来展示数据在SOM上的组织方式，从而有助于用户理解和利用这些信息。

- 总的来说，U-Matrix是自组织映射中一个重要的可视化工具，它帮助用户更好地理解和解释数据在SOM上的分布和聚类情况。通过U-Matrix的可视化，研究人员可以更加深入地了解数据的结构和特征，进而进行更有效的分析和决策。

U-Matrix



SOM实例

Iris Data Set

Download: Data Folder, Data Set Description

Abstract: Famous database; from Fisher, 1936

Data Set Characteristics:	Multivariate	**Number of Instances:**	150	**Area:**	Life
Attribute Characteristics:	Real	**Number of Attributes:**	4	**Date Donated**	1988-07-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	720483

Source:

Creator:

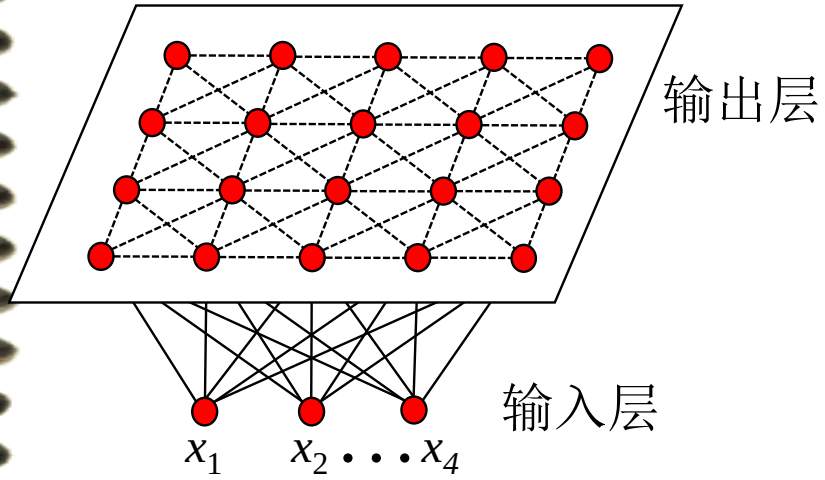
R.A. Fisher

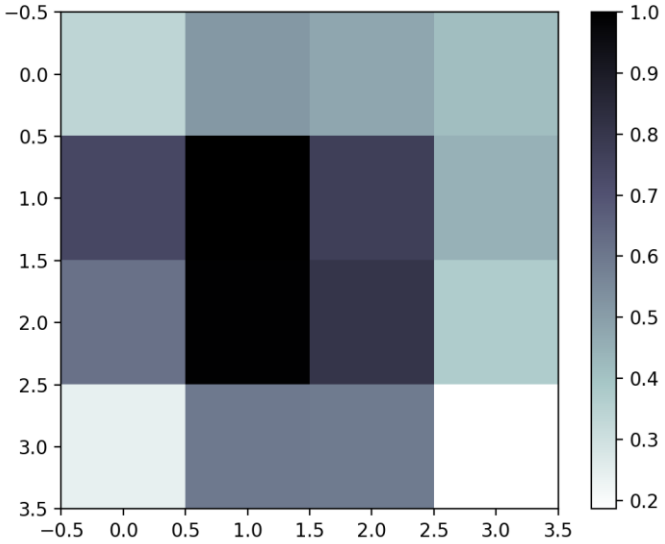
Donor:

Michael Marshall (MARSHALL%PLU '@' io.arc.nasa.gov)

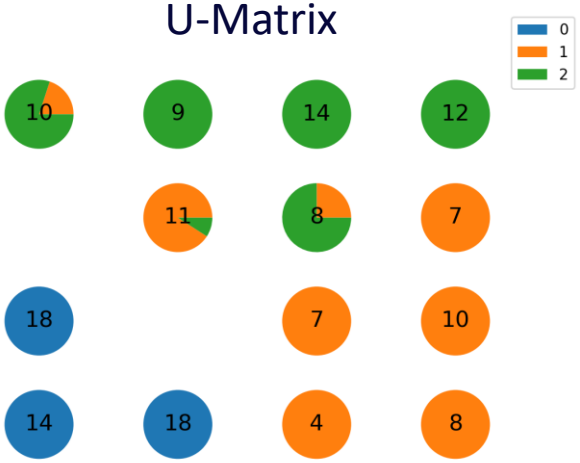
- 150个样本
- 属于鸢尾属下的三个亚属：山鸢尾、变色鸢尾和维吉尼亚鸢尾
- 四个特征：花萼和花瓣的长度和宽度

SOM实例





U-Matrix



输出节点