

## freexyn 编程实例视频教程系列 43

### Matlab 与神经网络 函数拟合和分类

#### 43.0 概述

##### 1.主要内容

1.1 运用 Matlab 进行神经网络算法的编程；

1.2 通过实例体会运用 Matlab 处理神经网络问题的思路。

作者：freexyn

#### 43.1 从神经网络用于函数拟合开始讲起

1.拟合数据（神经网络拟合、多项式拟合、线性回归）

$X = [-5 \ -4 \ -3 \ -2 \ -1 \ 0 \ 1 \ 2 \ 3 \ 4 \ 5]$

$T = [-5 \ -3 \ -1 \ 1 \ 3 \ 5 \ 7 \ 9 \ 11 \ 13 \ 15]$

##### 2.认识函数

fitnet

view

train

##### 3.说明

##### 3.1 函数拟合（function fitting）

神经网络在函数拟合方面有很好的表现，一个相对简单的神经网络就可以拟合任意的函数；

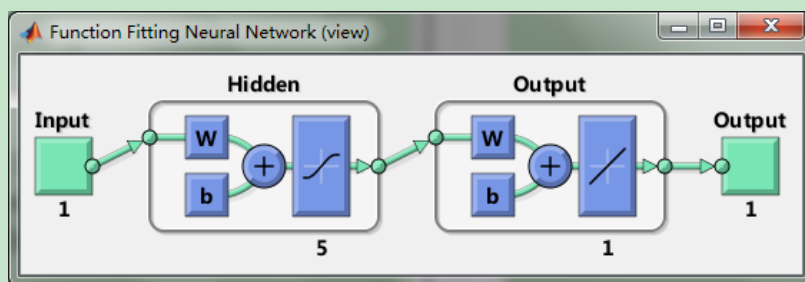
使用样本数据训练神经网络，建立输入-输出数据之间的关系，

训练之后，给出输入数据，使用训练好的网络计算输出结果；

### 3.2 输入输出数据形式

为神经网络拟合定义输入输出数据形式，给出  $n$  个输入向量，按列组成的矩阵，相应的，给出  $n$  个目标输出向量，按列组成矩阵。

### 3.3 网络结构

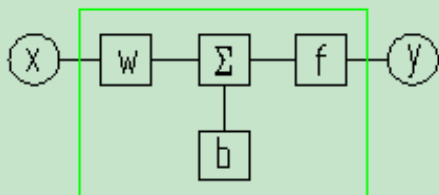


3.4 每次训练之后的神经网络的运算结果可能不一样。

## 43.2 神经网络结构（理论课）

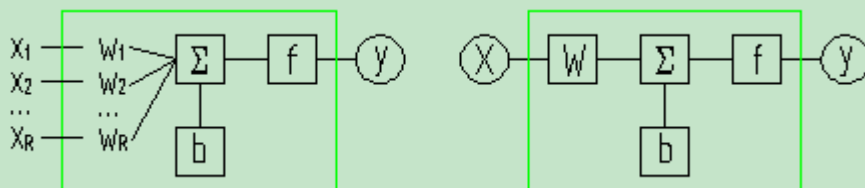
### 1. 神经元概念

简单的一次函数  $y=kx+b$  在神经网络算法中是怎么表达的呢？



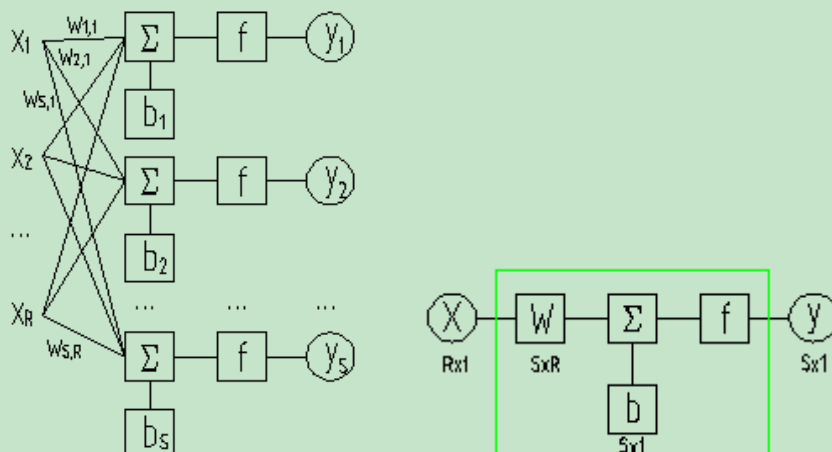
$$y=f(wx+b)$$

### 2. 输入为向量的神经元



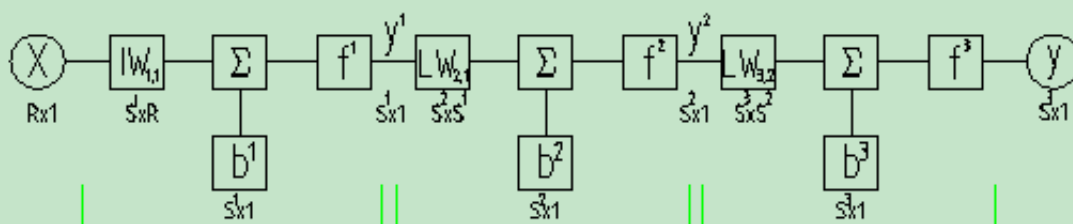
$$y=f(WX+b)$$

### 3.神经网络结构



$$y=f(WX+b)$$

### 4.多层神经网络



$$y=f_3(LW_3*f_2(LW_2*f_1(W_1X+b_1)+b_2)+b_3)$$

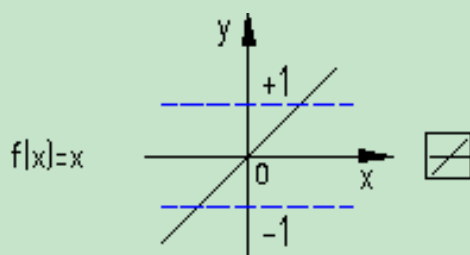
作者：freexyn

### 5. Matlab 中神经网络结构表示

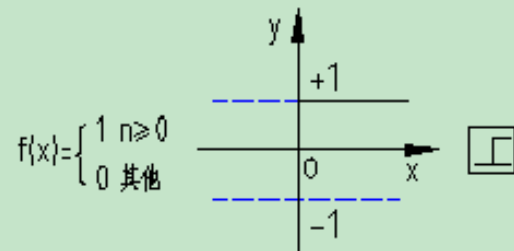
#### 43.3 传递函数

##### 1.常用的传递函数

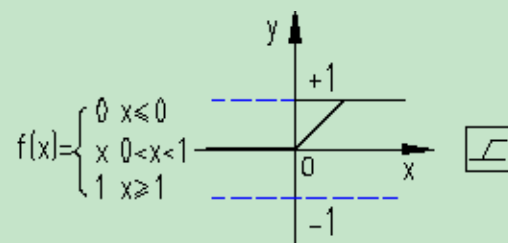
##### (1) 线性传递函数 purelin



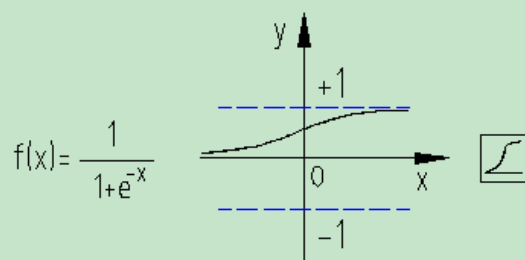
(2) 阈值型传递函数 hardlim



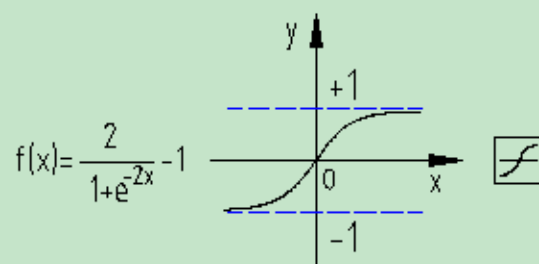
(3) 饱和线性传输函数 Satlin



(4) Log-sigmoid 型传递函数 logsig



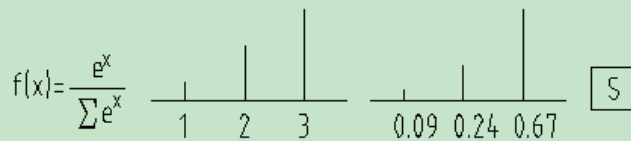
(5) Tan-sigmod 型传递函数 tansig



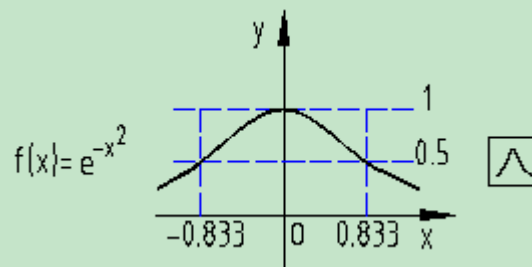
(6) 竞争型传递函数 compet



### (7) 柔性最大值传输函数 softmax



### (8) 径向基传输函数 radbas



## 2. fitnet 神经网络的传递函数

## 3.深度学习工具箱传递函数列表

>>help nntransfer

## 43.4 网络训练

### 1.介绍神经网络的训练过程（反向传播技术）

$x = [-0.5 \ 0.5];$

$t = [-0.5 \ 1.5];$

### 2.认识函数

train

adapt

### 3.说明

#### 3.1 简单理解

简单来说,算法的训练过程,就是不断更新参数直到收敛的过程,以最常见的梯度下降算法来说,沿着性能函数下降最快的方向,更新

参数，即，负梯度的方向，迭代更新公式：

$$x(k+1)=x(k)-a(k)g(k)$$

### 3.2 Matlab 中的训练过程

(1) 训练首先需要提供一组样本数据：输入  $x$  和目标  $t$ ；

(2) 使用 `train` 函数进行训练；

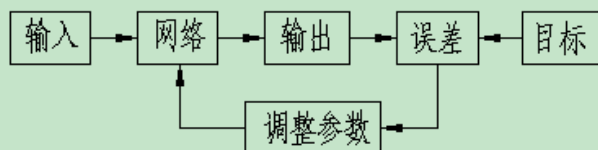
训练函数；

学习函数；

性能函数（误差函数）

$$F=mse=\frac{1}{n}\sum_{i=1}^n(t_i-y_i)^2$$

正向计算输出，误差反向传播更新参数；



### 3.3 训练方式

两种：增量方式和批量方式（incremental mode and batch mode）

增量方式中，输入数据每计算一次，参数就更新一次；

批量方式中，所有输入数据都计算完成后，再更新参数；

一般来讲，`train` 函数使用的是批量方式，增量方式对应的函数是 `adapt`，具体根据训练函数的设置；

对于大多数深度学习的问题，批量方式运算更快误差更小。

### 3.4 训练结束条件（收敛和终止）

以下情况发生时，训练结束：

到达最大的循环迭代次数（epochs）；

达到最大的运行时间 (time);

误差小于给定的目标值 (goal) ;

梯度小于给定值 (min\_grad);

验证数据误差经过多次循环迭代仍然不再改善 (max\_fail);

## 43.5 网络初始化和配置

1.编程演示网络初始化和配置

2.认识函数

init

configure

3.说明

网络创建后，需要初始化和配置网络；

初始化：初始化参数（权重和偏移量）；

配置：配置网络输入和输出大小(size)(匹配输入输出样本数据)，

并初始化参数；

## 43.6 数据预处理

1.介绍神经网络输入输出数据的预处理；

2.认识函数

mapminmax

mapstd

processpca

fixunknowns

removeconstantrows

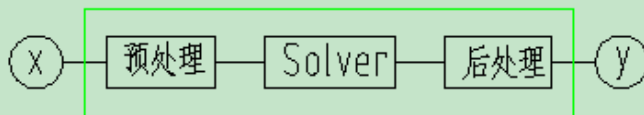
### 3.说明

#### 3.1 为什么要预处理

对输入输出数据做一些预处理，会使得网络训练更加高效。

#### 3.2 预处理

输入数据先进行预处理再应用到网络，相应的，输出数据做相反的变换，然后作为输出结果；



对于输入数据和输出数据，默认的处理函数都是 `mapminmax` 和 `removeconstantrows`，也可以自己更改默认设置；

作者：freexyn

#### 3.3 NaN 的用法

如果样本数据中有可以忽略的数据（离群值，未知点，不重要的点），可以使用 NaN 代替，传递到网络中会自动被忽略；

#### 3.4 程序中演示预处理函数的用法

#### 3.5 网络对象中预处理函数相关的属性用法

## 43.7 性能评价与作图

### 1.介绍神经网络的性能评价和图像函数的用法

### 2.认识函数



mse

perform

plotperform

ploterrhist

plotfit

plotregression

plottrainstate

3.说明

直接进入程序讲解。

## 43.8 数据分组

1.编程演示神经网络数据分组的用法

2.认识函数

dividerand

divideblock

divideint

divideind

3.说明

3.1 数据分组

训练网络时，通常会把样本数据分为三组：

- (1) 训练数据，用来计算梯度、更新网络参数；
- (2) 验证数据，计算误差，监控训练过程；

(3) 测试数据，不用于训练，但可用来比较不同的模型，可用来绘制误差变化趋势图；

### 3.2 对于训练的意义

训练开始后，训练数据误差会逐渐减小，验证数据的误差也会逐渐减小，随着训练过程的深入，当发生过拟合时，验证数据的误差开始上升，当误差大于某一最小值的迭代次数达到设定值时，训练停止，这是验证停止技术，它会把验证数据在最小值点的网络参数保存下来，训练停止后，网络参数再恢复到最小值点的参数值，作为训练结果；

### 3.3 四种分组函数原理（程序中演示）

### 3.4 Matlab 中的应用（程序中演示）

### 3.5 训练说明

每次训练之后，应用网络解决相同的问题都会得到不同的结果，这是因为每次训练过程中，数据分为训练、验证和测试不同等因素，不妨多次训练，以便获得最佳精度的网络。

## 43.9 工作流、属性和训练记录

1.介绍神经网络的设计工作流程、对象属性列表，及训练记录；

2.设计工作流

(1) 收集和整理数据；\*

(2) 创建神经网络；\*

(3) 配置网络；

(4) 初始化权重和偏移;

(5) 训练网络; \*

(6) 验证网络;

(7) 使用网络。

### 3.神经网络对象属性列表

以 fitnet 网络为例，程序中演示并说明;

### 4.神经网络训练记录列表

以 fitnet 网络为例，程序中演示并说明;

## 43.10 防止过拟合与改善网络性能

1.编程演示过拟合的情况，并学会如何改善网络训练结果;

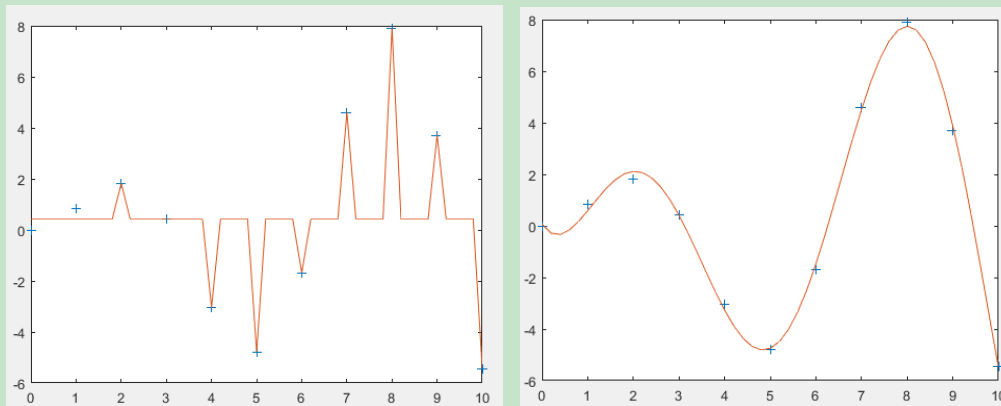
作者: freexyn

2.认识函数 (无)

3.说明

### 3.1 过拟合概念

训练神经网络会出现的一个问题就是过拟合，当样本数量少，但网络参数很多时，对于训练数据误差已经很小了，但使用新数据计算时，误差却很大，直接导致网络的泛化性不好。



### 3.2 防止过拟合的方法

- (1) 重新训练
- (2) 训练多个网络取平均值
- (3) Early Stopping
- (4) 正则化 (Regularization)

后面贝叶斯正则化算法再讲；

### 3.3 改善训练结果

如果训练后的网络, 运算精度不够, 可以尝试以下几种方式改善:

- (1) 重新初始化参数(init), 重新训练, 训练多次, 以找到最佳;
- (2) 增加隐含层神经元数量 (20 以上), 这意味着增加参数数量, 网络会具有更强大的功能;
- (3) 提供更多的训练样本数据;
- (4) 尝试不同的训练函数;

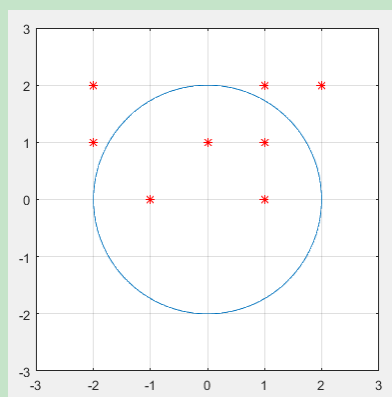
## 43.11 模式识别 pattennet

1.编程演示模式识别神经网络的用法, 对下面数据进行分类

$x = [-1 \ 0 \ 1 \ 1 \ -2 \ 2 \ -2 \ 1]$

```
0 1 1 0 2 2 1 2];
```

```
t=[1 1 1 1 2 2 2 2];
```



## 2.认识函数

patternnet

vec2ind

ind2vec

作者：freexyn

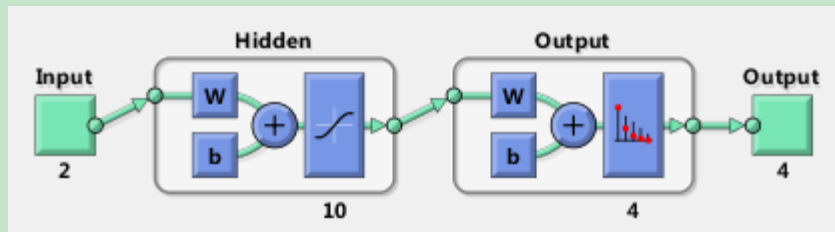
## 3 说明

### 3.1 模式识别

训练网络，对输入数据进行分类，输入数据对应的目标数据就是它的所属类别。

给出 n 组输入数据，按列排成矩阵，相应的，给出 n 组目标数据，表示每一组输入数据的所属类别，注意目标数据的元素只能是 0 或者 1；

### 3.2 网络结构



隐含层神经元数量，默认 10，隐含层传递函数 tansig，输出层传递函数 softmax（前面讲过）；

## 43.12 模式识别相关

1.编程演示模式识别神经网络的性能评价和图像函数的用法

2.认识函数

crossentropy

confusion

plotconfusion

roc

plotroc

3.说明

3.1 交叉熵

patternnet 网络默认的误差性能计算函数是交叉熵（crossentropy）

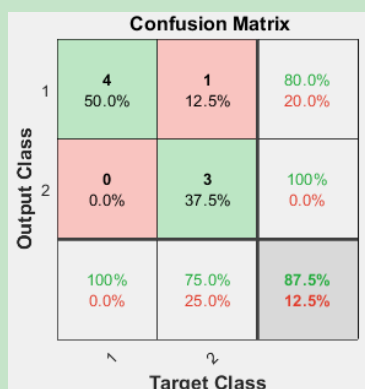
对于二元分类问题(0 或者 1)，计算公式如下

$$ce = -t .* \log(y) - (1-t) .* \log(1-y)$$

在具体编程中的用法，可以类比 mse，它也有 regularization 和 normalization 的属性。

3.2 混淆矩阵

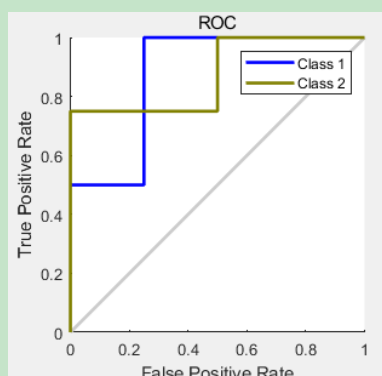
混淆矩阵（confusion）显示了正确和错误分类的百分比，是评价神经网络性能的一种方法。



### \*3.3 受试者工作特征图（只介绍函数，不讲理论）

受试者工作特征值(receiver operating characteristic, roc)，是评价神经网络性能的一种方法；

受试者工作特征图（plotroc），图像曲线越靠近左边和上边，表明分类越好。

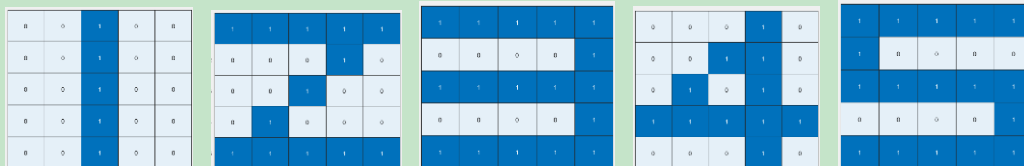


## 43.13 实例 数字图像识别

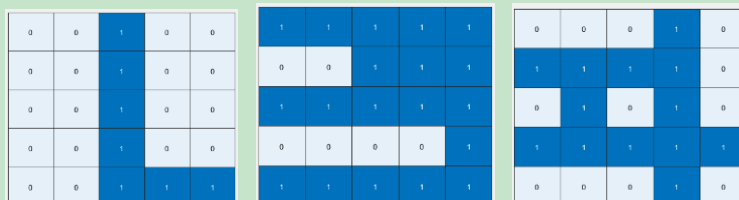
### 数字图像识别

使用 5\*5 的像素阵列表达图像，图像内容是数字 1~5，图像数据作为输入，识别出的数字作为输出；

### 样本数据集（训练）



待分类的数据（测试）



## 43.14 前馈神经网络

1.前馈神经网络和级联前馈神经网络

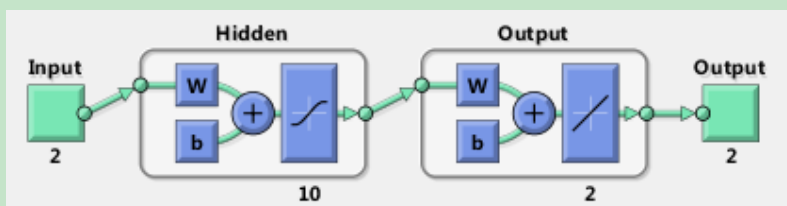
2.认识函数

feedforwardnet

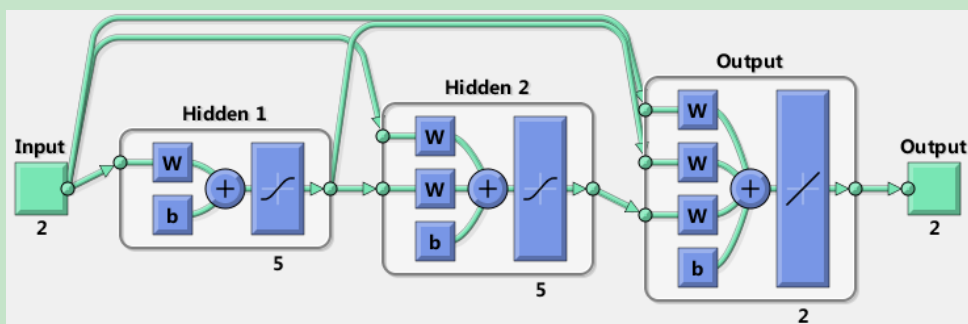
cascadeforwardnet

3.说明

feedforwardnet 神经网络，可用于拟合和分类；



cascadeforwardnet 神经网络





## 43.15 自定义神经网络

### 1.编程演示自定义神经网络

### 2.认识函数

network

### 3.说明

神经网络设计四个层次

- (1) 使用交互界面 GUI;
- (2) 使用基本的命令做神经网络, 可以使用或更改默认设置;
- (3) 自定义网络结构, 并充分使用属性设置和参数;
- (4) 更改或者编写工具箱函数

## 43.16\*算法前传（浅显理论课）

### 1.模型和损失函数

以线性回归（Linear Logistics）模型为例

#### (1) 模型（估计函数）

$$h(x) = h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

$$h_{\theta}(x) = \theta^T X$$

#### (2) 损失函数

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$$\min J_{\theta}$$

## 2. 梯度下降法

### 1. 梯度下降法

梯度下降法的优化思想是用当前位置负梯度方向作为搜索方向，因为该方向是当前位置下降最快的方向；

### 2. 算法流程

(1) 首先参数  $\theta$  初始化；

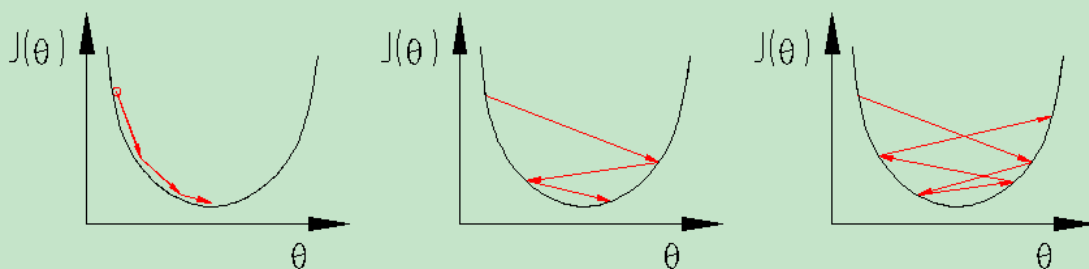
(2)  $J(\theta)$  按梯度下降的方向，更新（减小） $\theta$  的值；

$$\theta_i = \theta_{i-1} - \alpha \nabla J(\theta)$$

其中，

$$\nabla J(\theta) = \left( \frac{\partial J}{\partial \theta_1}, \frac{\partial J}{\partial \theta_2}, \dots, \frac{\partial J}{\partial \theta_n} \right)$$

### 3. 算法迭代过程可视化



### 4. 梯度下降法的缺点

(1) 靠近极小值时收敛速度明显变慢，所以利用梯度下降法求解需要很多次的迭代；

(2) 可能会“之字形”地下降；

(3) 在更新回归系数时要遍历整个数据集，是一种批处理方法，这样训练数据特别忙庞大时，需要很长运算时间；

(4) 如果误差曲面上有多个局极小值，那么不能保证这个过程

会找到全局最小值。

### 5.算例

$$J(\theta) = (\theta - 2)^2 - 1 \quad \nabla J(\theta) = 2(\theta - 2)$$

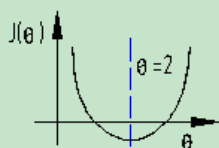
$$\theta_0 = 5 \quad \alpha = 0.3$$

$$\theta_1 = 5 - 0.3 \times 2(5 - 2) = 3.2$$

$$\theta_2 = 3.2 - 0.3 \times 2(3.2 - 2) = 2.48$$

$$\theta_3 = 2.48 - 0.3 \times 2(2.48 - 2) = 2.192$$

$$\theta_4 = 2.192 - 0.3 \times 2(2.192 - 2) = 2.0768$$



## 3.批量/随机/小批量/动量梯度下降法

### 1. 批量梯度下降 (BGD)

批量梯度下降法是最原始的形式, 在每一次迭代时使用所有样本来进行梯度的更新。

### 2. 随机梯度下降 (SGD)

随机梯度下降是每次迭代使用一个样本来对参数进行更新。

一般来讲, 随机梯度下降法的收敛速度比批量梯度下降法要快;

### 3.小批量梯度下降 (MBGD)

小批量梯度下降, 每次迭代使用一定数量的样本数据对参数进行更新, 它是对批量梯度下降以及随机梯度下降的一个折中办法。

总结, 这三种方法本质上遵循了梯度下降法的基本思想和算法步骤, 只是当遇到处理大批量数据的情况时, 考虑到运算时间和收敛速度的差别, 对数据采用单个逐次输入、分批输入和整体输入的方法做了细分。

### 4.动量梯度下降法

迭代  $n$  次的梯度序列为：

$$\{\nabla J_1, \nabla J_2, \nabla J_3, \dots, \nabla J_n\}$$

对应的动量梯度序列为：

$$\{V_0=0, V_1=\beta V_0+(1-\beta)\nabla J_1, V_2=\beta V_1+(1-\beta)\nabla J_2, \dots\}$$

迭代更新公式：

$$\theta(i) = \theta(i-1) - \alpha V(i)$$

## 4. 牛顿法

### 1. 牛顿法

牛顿法使用函数  $f(x)$  的泰勒级数的一阶展开来寻找方程  $f(x) = 0$

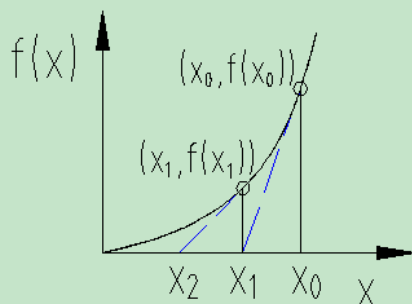
的根。  $f(x)$  在  $x_0$  点一阶泰勒展开式： $f(x) = f(x_0) + (x - x_0)f'(x_0)$

作者：freexyn

由  $f(x) = 0$ ，推导出迭代公式：

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

算法迭代过程可视化：



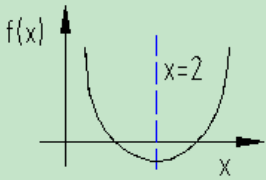
### 2. 牛顿法用于优化问题

对于优化问题，目标函数  $J(\theta)$ ，转化成  $J'(\theta) = 0$  的问题，进而可以使用牛顿法求解，迭代公式变成：

$$x_{n+1} = x_n - \frac{f'(x_n)}{f''(x_n)}$$

3. 牛顿法比梯度下降法收敛速度更快;

4. 算例

$$f(x) = (x-2)^2 - 1 \quad f'(x) = 2(x-2)$$


$$x_0 = 5$$

$$x_1 = 5 - 8/6 = 3.67$$

$$x_2 = 3.67 - (1.67 \times 1.67 - 1) / 2 / 1.67 = 3.13$$

$$x_3 = 3.13 - (1.13 \times 1.13 - 1) / 2 / 1.13 = 3.0075$$

## 5. 高斯牛顿法

可以看做牛顿法（适用一维数据）在多维数据上的延伸;

$$x_{n+1} = x_n - H_f(x_n)^{-1} \nabla f(x_n)$$

$$x_{i+1} = x_i - (J_f^T J_f)^{-1} J_f^T f$$

## 6. 拟牛顿法

基本思想是改善牛顿法每次需要求解复杂的 Hessian 矩阵的逆矩阵的缺陷，它使用正定矩阵来近似 Hessian 矩阵的逆，从而简化了运算的复杂度;

不同的拟牛顿法会有不同的处理，常用的拟牛顿法有 DFP 算法和 BFGS 算法。

## 7. 共轭梯度法 (Conjugate Gradient)

共轭梯度法是介于最速下降法与牛顿法之间的一个方法，它仅需利用一阶导数信息，但克服了最速下降法收敛慢的缺点，又避免了牛

顿法需要存储和计算 Hesse 矩阵并求逆的缺点，共轭梯度法不仅是解决大型线性方程组最有用的方法之一，也是解大型非线性最优化最有效的算法之一；

## 8. 贝叶斯理论

贝叶斯决策理论方法是统计模型决策中的一个基本方法；

其基本思想是：

1. 已知类条件概率密度参数表达式和先验概率；
2. 利用贝叶斯公式转换成后验概率；
3. 根据后验概率大小进行决策分类。

设  $D_1, D_2, \dots, D_n$  为样本空间  $S$  的一个划分，如果以  $P(D_i)$  表示事件  $D_i$  发生的概率，且  $P(D_i) > 0 (i=1, 2, \dots, n)$ ，那么，对于任一事件  $x$ ， $P(x) > 0$

$$P(D_j/x) = \frac{P(x/D_j)P(D_j)}{\sum_{i=1}^n P(x/D_i)P(D_i)}$$

## 43.17 LM 算法 trainlm

### 1. 算法

LM (Levenberg-Marquardt) 算法，中文为列文伯格-马夸尔特法；

它可以看成是最速下降法和高斯牛顿法的结合；

迭代公式如下：

$$x_{n+1} = x_n - (J_e^T J_e + \mu E)^{-1} J_e^T e$$

其中,  $J$  是雅克比矩阵,  $u$  是系数,  $E$  是单位矩阵,  $e$  是误差函数。

## 2. Matlab 中的 LM 算法函数 trainlm

在 matlab 中, 网络训练函数 trainlm 使用 LM 算法更新网络的参数值, 损失函数必须是 mse 或者 sse。

使用反向传播, 对于参数矩阵  $X$  (权重和偏差) 每个分量, 计算雅克比矩阵  $jX$  和损失函数 mse (性能评价函数 perf), 根据 LM 算法更新变量:

$$dX = -(jX * jX + E * \mu) \setminus jX * e$$

大多情况下, 该算法是运算速度最快的反向传播算法, 不妨作为优先选择, 但是它的内存消耗比较大;

## 3. 训练参数设置

设置属性 net.trainFcn='trainlm'

net.trainParam 的属性值列表如下:

net.trainParam.epochs	1000	最大循环次数
net.trainParam.goal	0	误差目标值
net.trainParam.max_fail	6	最大验证失败次数
net.trainParam.min_grad	1e-7	误差函数梯度最小值
net.trainParam.mu	0.001	Mu 初始值
net.trainParam.mu_dec	0.1	下降系数
net.trainParam.mu_inc	10	上升系数
net.trainParam.mu_max	1e10	Mu 最大值
net.trainParam.show	25	每 25 个循环显示 (NaN 不显示)
net.trainParam.showCommandLine	false	产生命令行窗口输出
net.trainParam.showWindow	true	显示训练界面

---

net.trainParam.time	inf	最大训练时间（秒）
---------------------	-----	-----------

---

以下情况发生时，训练结束：

到达最大的循环迭代次数（epochs）；

达到最大的运行时间（time）；

误差小于给定的目标值（goal）；

梯度小于给定值（min\_grad）；

Mu 大于给定值（mu\_max）；

验证数据误差经过多次循环迭代仍然不再改善（max\_fail）；

#### 4.学习函数

学习函数 `learngd` 使用梯度下降法更新参数（权重和偏移）；

$$W(i)=W(i-1)+dW$$

其中， $dW = lr * gW$

学习函数 `learngdm` 使用动量梯度下降法更新参数；

$$dW = mc * dW(i-1) + (1-mc) * lr * gW$$

Matlab 中的设置

设置属性 `net.inputWeights{1}.learnFcn='learngd'`；

设置参数：学习率和动量常数；

#### 5.性能函数

`net.performFcn='mse'`

`net.performParam`



## 43.18 贝叶斯正则化 trainbr

### 1.算法

#### (1) 正则化 (Regularization)

正则化，通过对需要优化的参数施加约束，以防止过拟合；

$$J^*(\theta) = J(\theta) + \alpha \Omega(\theta)$$

L2 正则化项：

$$\Omega(\theta) = (1/2) * \sum (w)^2$$

L1 正则化项：

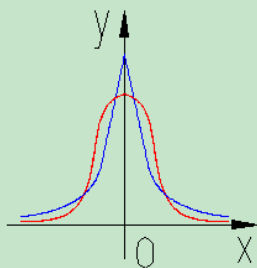
$$\Omega(\theta) = \sum |w|$$

#### (2) 贝叶斯正则化

从概率论角度讲， $w$  是先验概率， $J^*(\theta)$  是后验概率，贝叶斯方法通过最优化后验概率估计参数  $w$ ；

当先验概率分布满足正态分布（高斯分布）的时候，把  $J^*(\theta)$  代入贝叶斯理论的公式，可以推导出 L2 正则化项；

当先验概率分布满足拉普拉斯分布的时候，把  $J^*(\theta)$  代入贝叶斯理论的公式，可以推导出 L1 正则化项；



## 2. Matlab 中的 trainbr 函数

在 matlab 中，网络训练函数 trainbr 使用贝叶斯正则化算法更新网络的参数值，损失函数是方差和参数的联合值，公式：

$$\text{Reg\_mse} = (1 - \alpha) \text{mse} + \alpha \text{msw}$$

其中，

$$\text{msw} = \frac{1}{n} \sum_{j=1}^n w_j^2$$

## 3. 训练参数设置

作者：freexyn

设置属性 net.trainFcn='trainbr'

net.trainParam 的属性值列表如下：

net.trainParam.epochs	1000	最大循环次数
net.trainParam.goal	0	误差目标值
net.trainParam.mu	0.005	Mu 初始值【LM 算法的调整参数】
net.trainParam.mu_dec	0.1	下降系数
net.trainParam.mu_inc	10	上升系数
net.trainParam.mu_max	1e10	Mu 最大值
net.trainParam.max_fail	inf	最大验证失败次数
net.trainParam.min_grad	1e-7	误差函数梯度最小值
net.trainParam.show	25	每 25 个循环显示 (NaN 不显示)
net.trainParam.showCommandLine	false	产生命令行窗口输出
net.trainParam.showWindow	true	显示训练界面
net.trainParam.time	inf	最大训练时间 (秒)

注意，默认地，max\_fail=inf；

以下情况发生时，训练结束：

到达最大的循环迭代次数 (epochs);

达到最大的运行时间 (time);

误差小于给定的目标值 (goal) ;

梯度小于给定值 (min\_grad);

Mu 大于给定值 (mu\_max);

#### 4.学习函数

`net.learnFcn='learngdm';`

设置参数：学习率和动量常数；

#### 5.性能函数

`net.performFcn='mse'`

`net.performParam` 两个参数

正则化参数  $\alpha$  使用 `net.performParam.regularization` 设置；

### 43.19\*比例共轭梯度法 `trainscg`

#### 1.算法

Scaled conjugate gradient 量化（比例）共轭梯度法

#### 2.Matlab 中的 `trainscg` 函数

在 matlab 中，网络训练函数 `trainscg` 根据量化（比例）共轭梯度法更新网络参数；

使用反向传播，对于参数矩阵 (权重和偏差)每个分量，计算雅克比矩阵和性能评价函数，然后更新网络参数；

该算法与 `traincgp`, `traincgf`, and `traincgb` 算法一样，都是基于共轭梯度方向的；

### 3.训练参数设置

设置 `net.trainFcn = 'trainscg'`

`net.trainParam` 的属性值列表如下：

<code>net.trainParam.epochs</code>	1000	最大循环次数
<code>net.trainParam.show</code>	25	每 25 个循环显示 (NaN 不显示)
<code>net.trainParam.showCommandLine</code>	false	产生命令行窗口输出
<code>net.trainParam.showWindow</code>	true	显示训练界面
<code>net.trainParam.goal</code>	0	误差目标值
<code>net.trainParam.time</code>	inf	最大训练时间 (秒)
<code>net.trainParam.min_grad</code>	1e-6	误差函数梯度最小值
<code>net.trainParam.max_fail</code>	6	最大验证失败次数
<code>net.trainParam.sigma</code>	5.0e-5	*计算近似二阶导数的权重的改变量
<code>net.trainParam.lambda</code>	5.0e-7	*调整海塞矩阵不确定性的参数

以下情况发生时，训练结束：

到达最大的循环迭代次数 (`epochs`)；

达到最大的运行时间 (`time`)；

误差小于给定的目标值 (`goal`) ；

梯度小于给定值 (`min_grad`)；

验证数据误差经过多次循环迭代仍然不再改善 (`max_fail`)；

## 4.学习函数

设置属性 `net.learnFcn='learngdm'`;

设置参数：学习率和动量常数；

## 5.性能函数

`net.performFcn='mse'`

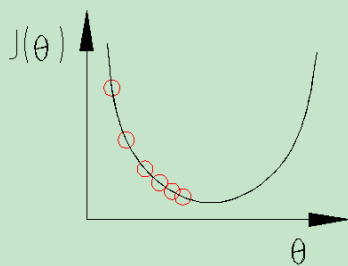
`net.performParam` 两个参数

## 43.20 弹性反向传播算法 `trainrp`

### 1.算法

`resilient backpropagation` 弹性反向传播算法；

从梯度下降法说起：



$$\theta(i+1) = \theta(i) - \alpha \Delta J$$

弹性反向传播算法的基本思想是，消除梯度（偏微分）幅度的影响因素，只使用梯度的符号变化，决定参数更新的方向，引入一个独立的变量  $\beta$ ，决定参数更新的大小；

$$\Delta J^* = \beta * \text{sign}(\Delta J)$$

## 2. Matlab 中的 trainrp 函数

在 matlab 中，训练算法 trainrp 根据弹性反向传播算法更新网络参数；

使用反向传播，对于参数矩阵 X(权重和偏差)每个分量，计算偏微分 gX 和性能评价函数 perf，然后更新网络参数，更新公式为：

$$dX = \text{deltaX} * \text{sign}(gX)$$

## 3. 训练参数设置

设置 net.trainFcn = 'trainrp'

net.trainParam 的属性值列表如下：

net.trainParam.epochs	1000	最大循环次数
net.trainParam.show	25	每 25 个循环显示 (NaN 不显示)
net.trainParam.showCommandLine	false	产生命令行窗口输出
net.trainParam.showWindow	true	显示训练界面
net.trainParam.goal	0	误差目标值
net.trainParam.time	inf	最大训练时间（秒）
net.trainParam.min_grad	1e-5	误差函数梯度最小值
net.trainParam.max_fail	6	最大验证失败次数
net.trainParam.delt_inc	1.2	权重变量的增加量
net.trainParam.delt_dec	0.5	权重变量的减小量
net.trainParam.delta0	0.07	权重变量的初始值
net.trainParam.deltamax	50.0	权重变量的最大值

以下情况发生时，训练结束：

到达最大的循环迭代次数（epochs）；

达到最大的运行时间（time）；

误差小于给定的目标值（goal）；

梯度小于给定值 (min\_grad);

验证数据误差经过多次循环迭代仍然不再改善 (max\_fail);

#### 4.学习函数

设置属性 net.learnFcn='learngdm';

设置参数：学习率和动量常数；

#### 5.性能函数

net.performFcn='mse'

net.performParam 两个参数

### 43.21 算法汇总比较

#### 1.训练函数列表

本深度学习工具箱里，训练函数列表如下：

训练函数	算法
trainlm	LM 算法
trainbr	贝叶斯正则化
trainbfg	BFGS 拟牛顿法
trainrp	弹性反向传播算法
trainscg	比例共轭梯度法
traincgb	Powell/Beale Restarts 共轭梯度法
traincgf	Fletcher-Powell 共轭梯度法
traincgp	Polak-Ribière 共轭梯度法
trainoss	One Step Secant
traingdx	可变学习率梯度下降法
traingdm	动量梯度下降法

训练函数	算法
traingd	梯度下降法

## 2.Fitnet 网络

对于 Fitnet 网络，默认训练函数是 trainlm，可选如上表所示；

其中，训练速度最快的是 trainlm，拟牛顿法 trainbfg 也比较快，但是，这两个对大型神经网络(上千个参数)效率不高，因为需要更多的内存，更多的运算时间，trainlm 在函数拟合(非线性回归)问题上的表现优于模式识别问题；

对于训练大型网络，或者对于训练模式识别问题，trainscg 和 trainrp 是最佳选择，内存消耗量小，比标准的梯度下降法速度快；

## 3.简单总结

哪种方法最快，很难说，这依赖于很多因素，包括问题的复杂性，样本数据的数量，网络参数的数量，误差目标值，网络是用于函数拟合（回归）还是模式识别（判别式分析）等等；

对于函数逼近问题，网络参数小于 100 个，LM 算法最快，当训练精度要求很高时，这个优势更加明显，在许多情况下，trainlm 比其他算法可以获得更小的误差；但是，网络参数增多时，优势就不明显了，而且在模式识别问题上表现不佳；内存需求比其他算法要大；

对于模式识别问题，rp 是最快的；但是，它对函数逼近问题表现不佳；目标误差值较小时（精度提高），它的优势也会减弱；内存需求比其他算法相对要小；



共轭梯度法，尤其是 scg，当网络参数很多的时候，对于很多问题都表现较好；在函数逼近问题上，与 LM 一样快（参数很多时，比 LM 还要快）；在模式识别问题上，与 rp 一样快；当目标误差值较小时（精度提高），它的优势并不像 rp 减弱的那么快；该算法比最速下降法运算速度快，共轭梯度法具有相对中等的内存需求；

可变学习率算法（traingdx）几乎是最慢的算法，与 rp 具有差不多的内存需求量；

## 43.22 生成函数和仿真模块

### 1.编程演示神经网络生成函数和 Simulink 的用法

### 2.认识函数

genFunction

gensim

### 3.说明

#### 3.1 生成函数

训练好的网络可以使用 genFunction 生成独立的函数，产生的函数包含了仿真网络所需要的信息，例如参数值、设置等；

#### 3.2 生成仿真模块

如果网络没有延时，那么可以指定第二个参数为-1（默认为 1），意思是连续型采样；

## 43.23 交互方式和样本数据

1.演示神经网络交互界面的用法和内置的样本数据;

2.认识函数

nnstart

nftool

nctool

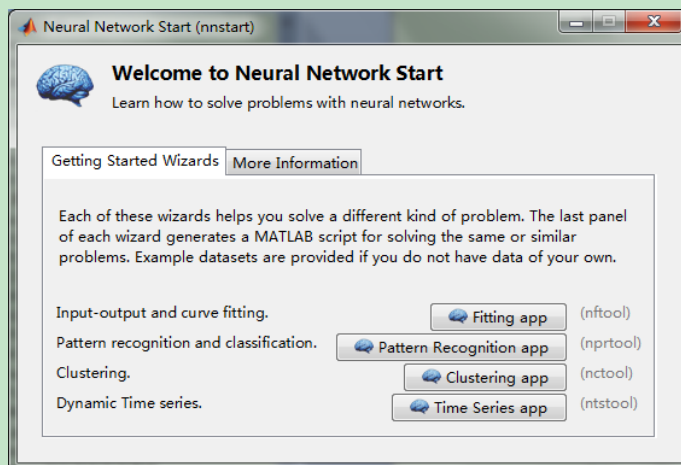
nprtool

ntstool

nndatasets

3.说明

神经网络交互界面提供一种不需要编程就可以使用神经网络工具处理一些基本问题的途径;



深度学习工具箱预设了很多样本数据,可以直接使用它们训练和测试网络;

## 43.24 感知器

1.使用感知器神经网络进行分类;

```
x=[1 2 2 0 -1 -1
```

```
0 0 1 1 0 1];
```

```
t=[0 0 0 1 1 1]
```

作者: freexyn

2.认识函数

perceptron

plotpv

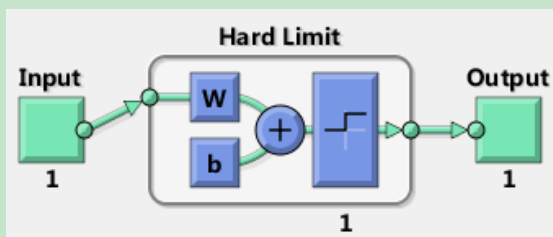
plotpc

3.说明

3.1 概念

感知器是一个简单的单层二元分类器，把输入数据进行线性分类;

3.2 感知器神经网络结构



传递函数是，阈值型传递函数 hardlim（前面讲过）;

Output=hardlim(Wx+b)

从网络结构中可以看出，感知器仅能够训练单个层，即单层神经网络，这也使得感知器解决问题的能力是有限的。

### 3.3 感知器算法原理

默认的权重更新规则：

$$\Delta w = (t - y)p^T$$

通过学习函数 learnp 实现；

$$\Delta w = (t - y) \frac{p^T}{\|p\|}$$

通过 learnpn 学习函数实现；

训练函数 trainc

性能函数 mae

### 3.4 局限性

输出值只有 0 和 1；

只能对线性可分数据进行二元分类；

## 43.25 线性神经网络

1.使用线性神经网络对下面样本数据进行函数拟合；

`x=[-5 -4 -3 -2 -1 0 1 2 3 4 5];`

`t=[-5 -3 -1 1 3 5 7 9 11 13 15];`

2.认识函数

`linearlayer`

`newlind`

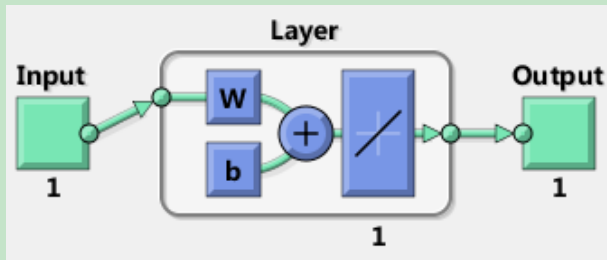
3.说明

#### 3.1 概念

线性层是一个单层的线性神经网络结构，可以处理线性相关（线

性拟合或线性分类) 的问题;

### 3.2 网络结构



$$a = \text{purelin}(wp+b) = wp+b$$

### 3.3 算法原理

最小均方算法 (LMS 算法, 也叫 Widrow-Hoff 学习算法), 是基于近似最速梯度过程的算法, 算法不详细讲述 (比较简单), 直接给出迭代公式:

$$\Delta w = lr * (t-y) * p'$$

$$\Delta b = lr * (t-y)$$

使用学习函数 learnwh 实现;

训练函数 trainb

性能函数 mse

### 3.4 局限性

线性网络只能处理输入输出之间的线性关系问题;

超定系统, 只能计算误差最小的近似解;

欠定系统, 网络可能无效;

## 43.26 自适应线性神经网络

1.使用自适应线性神经网络对下面采样数据进行时序预测;

```
x= {2 3 4 5};
```

```
xi={1};
```

```
t={5 7 10 12};
```

## 2.认识函数

con2seq

seq2con

## 3.说明

### 3.1 数据结构

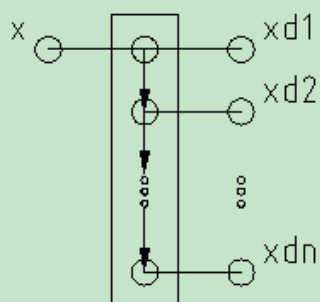
并行向量 (concurrent vector): 输入向量没有先后顺序之分;

有序向量 (sequential vector): 输入向量有先后的时序顺序;

深度学习工具箱使用矩阵表示并行数据,使用元胞数组表示有序数据,处理并行数据的网络是静态网络,处理有序数据的网络是动态网络;

### 3.2 抽头延迟线

抽头延迟线 (Tapped Delay Line, TDL)



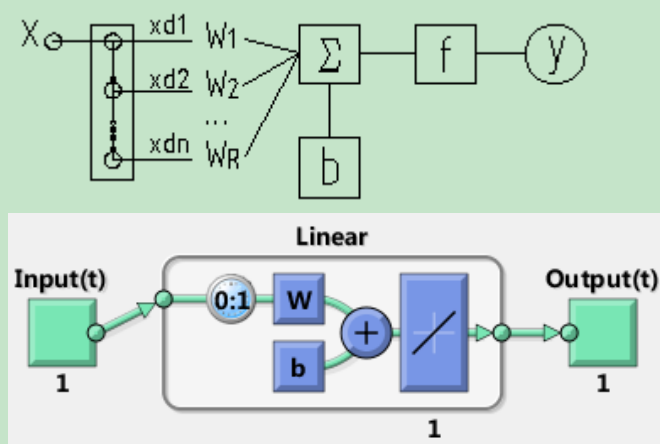
### 3.3 自适应线性过滤器 (概念和网络结构)

把 TDL 和线性网络组合起来创建自适应线性过滤器;

用于处理简单的线性时序预测问题,输入数据是有序数据,输入

延时大于 0;

网络结构如下



$a = \text{purelin}(WX+b)=?$

## 43.27 径向基神经网络

1. 使用径向基神经网络进行函数拟合

$x = [-5 -4 -3 -2 -1 0 1 2 3 4 5];$

$t = [-5 -3 -1 1 3 5 7 9 11 13 15];$

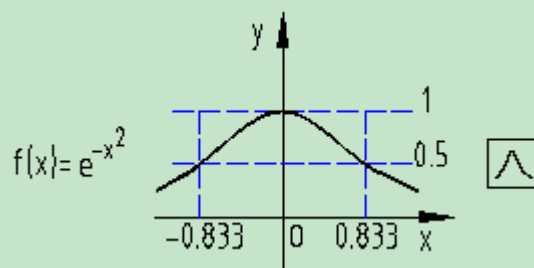
2. 认识函数

newrbe

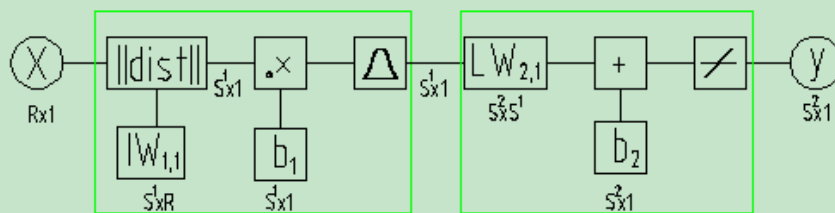
newrb

3. 说明

3.1 概念（回顾）

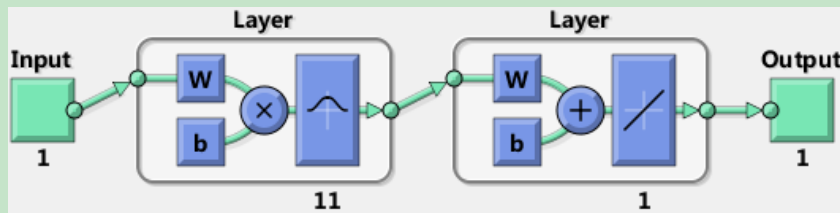


### 3.2 网络结构



这个径向基神经网络是由两个层组成：一个径向基函数作为传递函数的隐含层，一个线性函数作为传递函数的输出层；

$$a1 = \text{radbas}(\|IW(1,1) - X\| * b1), \quad y = \text{purelin}(LW(2,1) * a1 + b2)$$



### 3.3 newrbe 和 newrb 原理

使用函数 newrbe 或 newrb 创建径向基神经网络层；

newrbe 函数计算原理是，输入向量的数量有多少，这个函数就会在径向基层创建多少个神经元，并把这一层的权重参数初始化为  $X'$ ，把偏差参数  $b$  设置为  $0.8326/\text{spread}$ ；

一方面，为了尽可能的覆盖输入向量空间的所有距离，参数 spread 尽量选大，有助于使得网络函数变得平滑；

另一方面，参数也不能选的很大，因为会导致输出都在一个很高的水平上（接近于 1），网络功能可能失效；

newrb 函数的原理与 newrbe 相似，不同的是，newrb 在迭代过程中，每次往网络中添加一个神经元，直到误差平方和低于给定的参数 goal 为止，或者达到最大的神经元数目为止；



## 43.28 广义回归神经网络 GRNN

1. 使用广义回归神经网络进行函数拟合

```
x=[-5 -4 -3 -2 -1 0 1 2 3 4 5]; % 第一节的样本数据
```

```
t=[-5 -3 -1 1 3 5 7 9 11 13 15];
```

2. 认识函数

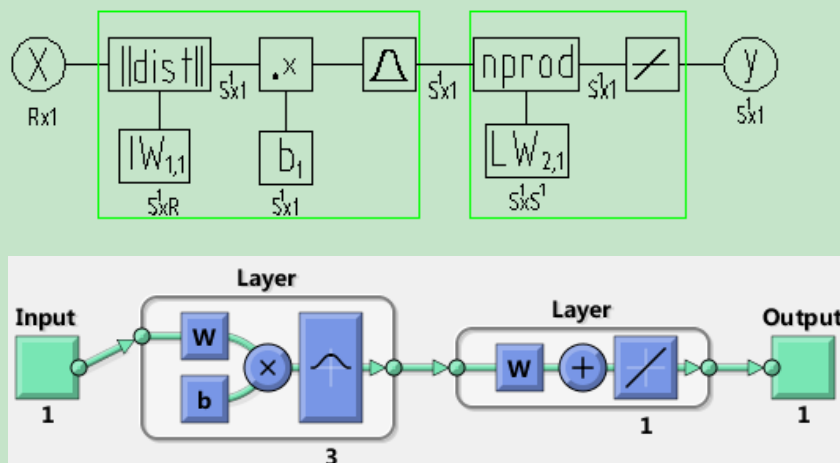
Newgrnn

作者: freexyn

3. 说明

3.1 概念和网络结构

广义回归神经网络 (Generalized regression neural network, GRNN), 可用于函数逼近 (拟合), 网络结构如下:



## 43.29 概率神经网络 PNN

1. 使用概率神经网络进行分类 (模式识别)

```
x=[-1 0 1 1 -2 2 -2 1
```

```
0 1 1 0 2 2 1 2]; % 第 11 节实例
```

```
t=[1 1 1 1 2 2 2 2];
```

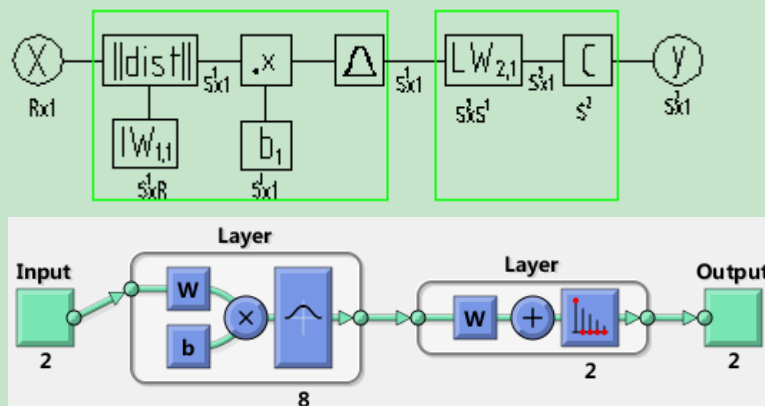
## 2.认识函数

newpnn

## 3.说明

### 3.1 概念和网络结构

概率神经网络 (probabilistic neural networks, PNN), 可用于分类问题, 网络结构如下:



## 43.30 学习向量量化 LVQ

### 1.使用学习向量量化神经网络进行分类 (模式识别)

```
x=[-1 0 1 1 -2 2 -2 1
```

```
0 1 1 0 2 2 1 2];
```

```
t=[1 1 1 1 2 2 2 2];
```

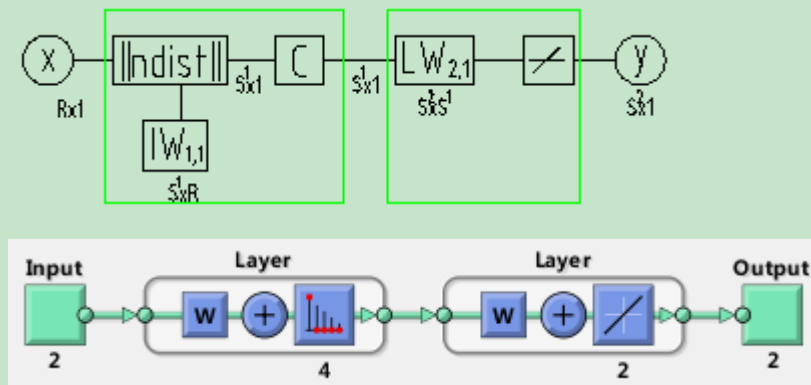
## 2.认识函数

lvqnet

## 3.说明

### 3.1 概念和网络结构

学习向量量化（Learning vector quantization, LVQ）用于分类问题，网络结构如下：



### 3.2 学习函数 learnlv1

通过上述神经网络计算之后，分类可能是正确的，也可能是错误的，接下来就是调整更新参数，在第一层中，如果  $x(i)$  分类为  $s(j)$  是正确的，那么就调整  $s(j)$  对应的神经元的权重参数往该输入向量  $x(i)$  靠近的方向移动；如果分类错误，就朝远离的方向移动；

参数更新迭代公式：

$$W(i) = W(i-1) + a(X(i) - W(i-1))$$

$$W(i) = W(i-1) - a(X(i) - W(i-1))$$

如上所述的内容就是 LVQ1 算法，在 matlab 中使用学习函数 learnlv1 实现；

### 3.3 学习函数 learnlv2

算法 LVQ2.1 的原理与 LVQ1 是相似的，只是在处理两个向量都靠近同一个神经元的情况时，会进行一些运算和处理，使得结果更稳定（鲁棒性高），目的是改善第一层的学习结果；

在 matlab 中实现这个算法的函数是 learnlv2；

### 3.4 训练函数 trainr

trainr, 基于学习函数的随机顺序增量训练方法;

## 43.31 自编码器

1.使用自编码器神经网络, 对随机数据进行重构(重现)

2.认识函数

trainAutoencoder

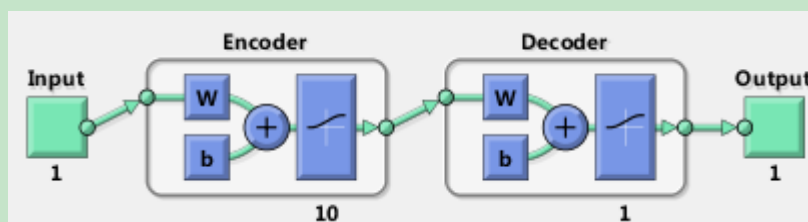
predict

3.说明

### 3.1 概念和网络结构

自编码器是一种神经网络, 用于模式识别, 该网络会尝试在其输出能够复现其输入。

它由一个编码器和一个解码器组成, 网络结构如下:



### 3.2 算法原理

对自动编码器的训练是无监督式的。

对于第一层(编码器层), 输入数据被映射成另一种数据, 计算公式:  $z1=h1(w1*x+b1)$

对于第二层(解码器层), 数据逆运算重构原始数据, 计算公式:  $x2=h2(w2*z1+b2)$

### 3.3 损失函数

训练过程仍然基于损失函数的优化。

$$J = \text{mse} + \lambda * \Omega(L2) + \beta * \Omega(s)$$

#### (1) 稀疏正则项

首先引入一个概念，隐含层神经元的平均活跃度（激活函数输出值的平均值）定义为：

$$\rho = \frac{1}{n} \sum_{i=1}^n z1(x_i)$$

一种计算方法是 Kullback-Leibler 散度：

$$\Omega(s) = \sum_{i=1}^D q \times \log\left(\frac{q}{p_i}\right) + (1-q) \times \log\left(\frac{1-q}{1-p_i}\right)$$

#### (2) L2 正则项

向损失函数中增加一个权重相关的正则项，称为 L2 正则项，计算公式：

$$\Omega(L2) = \frac{1}{2} \sum_l \sum_j^n \sum_i^k (w_{ji}^{(l)})^2$$

### 3.4 稀疏自编码器

这个稀疏性是针对自编码器的隐层神经元而言的，通过对隐层神经元的大部分输出进行抑制使网络达到一个稀疏的效果。

可以设置以下参数来控制正则项的影响：

L2WeightRegularization

SparsityRegularization

SparsityProportion

## 43.32 编码和解码

1.随机数据实例演示自编码器编码和解码应用。

2.认识函数

`encode`

`decode`

`plotWeights`

`network`

3.说明

上节已经讲过自编码器编码和解码的网络结构和原理；

在图像识别问题应用方面，样本数据经过编码之后的数据，一般称为特征数据，编码也就是特征提取的运算过程。

`plotWeights` 可以把自编码器学习的特征作图呈现出来；

## 43.33 堆叠自编码器

1.根据花的特征数据，使用堆叠自编码器网络进行分类；

作者：freexyn

2.认识函数

`trainSoftmaxLayer`

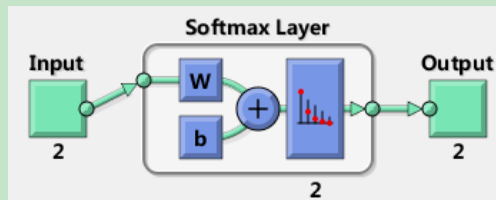
`stack`

3.说明

3.1 Softmax 分类神经网络层

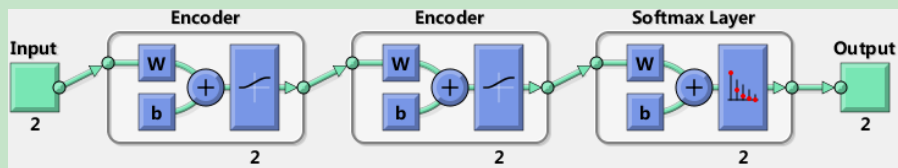
`softmax` 分类神经网络层可以简单理解成一种分类输出结果层，

传递函数 softmax 在前面课程已经讲过，默认的性能评价函数是交叉熵，默认的训练函数 trainscg，该网络层通过训练学习分类功能，网络结构如下：



### 3.2 自编码器堆叠

自编码器可首尾相连进行堆叠，可堆叠的对象要求是维度匹配，即，前一个输出层与后一个的输入层大小相同，其属性参数继承了原有网络对象的属性参数，最后可以连接 Softmax 分类神经网络层用于输出分类结果。



本系列教程结束

欢迎交流和留言

作者/旺旺/微信公众号/UP: freexyn

邮箱: freexyn@163.com (建议、提问、合作、供稿等, 请发邮件)

[点击 freexyn 官方淘宝小店 >>试看全部课程<<](#)

版权所有 侵权必究

**End**