Neural Networks-Experiment1

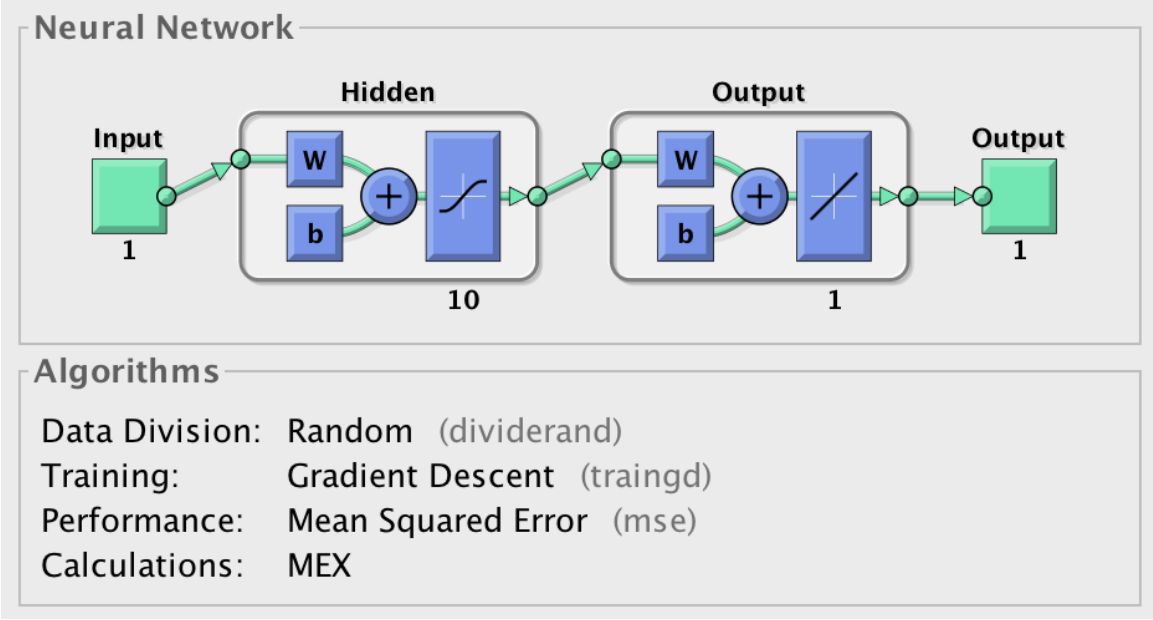
tnlin

1. 试利用 BP 算法和 BPM 算法设计一个多层前向神经网络,实现某一函数的逼近。(50’)

开发环境：Matlab 2015b

目标函数：实现cosine函数的逼近

我设计了一个1-10-1三层前向网络，并且用Gradient Descent加以训练



本来一开始采用newff函式来实作，但这个函式已经在2010年时候被Matlab弃置

所以我最后使用feedforwardnet函式将程序重新改写

训练参数设置如下：

```
net.trainParam.lr=0.03;      % Learning rate
net.trainParam.mu=0.9;       % Momentum constant
net.trainParam.epochs=3000; % Maximum number of epochs to train
net.trainParam.show=20;      % Epochs between displays
```

由上而下分别是学习速度、动量因子、最大训练步数和隔几步显示一次训练结果

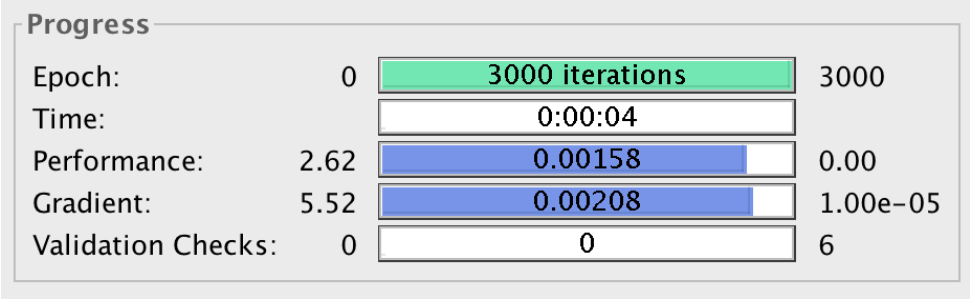
还会有预设的validation check=6, 当cost上升6次后将自动停止训练

网络权值：训练后得到的是一个10x1的Vector

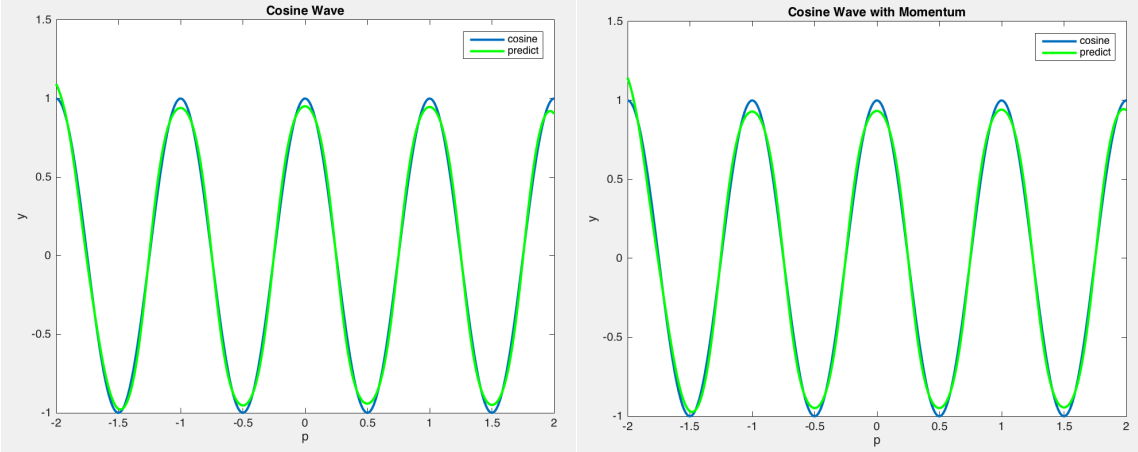
为了方便观察，这边印出weight’ (1x10)

	1	2	3	4	5	6	7	8	9	10
1	7.0985	6.2483	7.0281	-5.3246	-6.0768	-8.0579	5.9906	-8.5312	-6.3551	8.8954

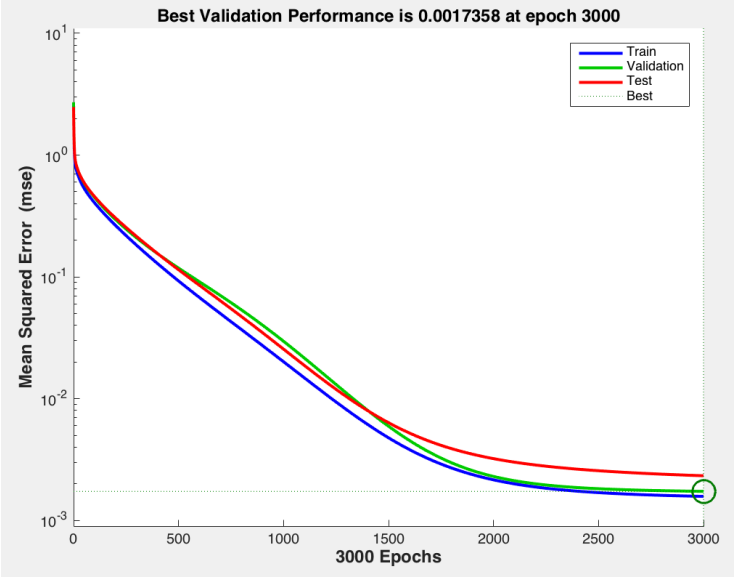
逼近效果图：图片分成两个部分呈现
第一部分（蓝色）是原本的cosine函数
第二部分（绿色）是预测的cosine函数，给定一段输入



输入p：从-2到2，步伐为0.01的一个1x401矩阵
目标y：从$-4\pi \sim 4\pi$的cosine函数，1x401矩阵



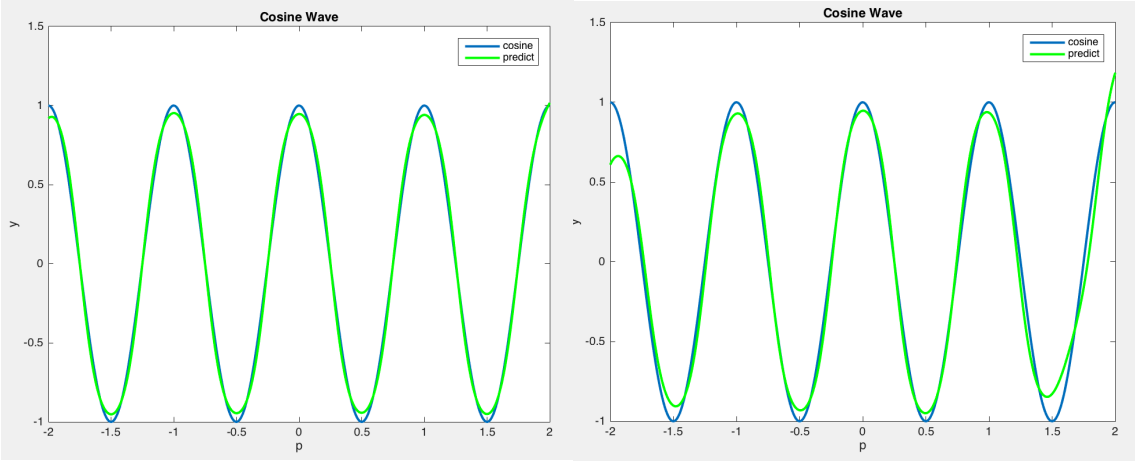
左边是BP，右边是BPM，训练结果看来是差不多的
不得不吐槽Gradient descanting后面的收敛速度实在是太慢了。。。
逼得我把学习速率和训练步数都*3(预设是0.01,1000步)，才能得到满意的结果。



这是Mean Square Error的视觉化，可以看到大概在2000轮后的收敛速度极慢无比

训练步长对算法性能的影响

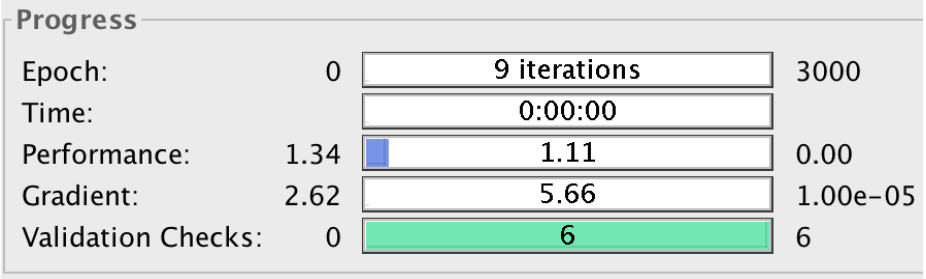
将net.trainParam.lr改成0.1和0.01，进行比较

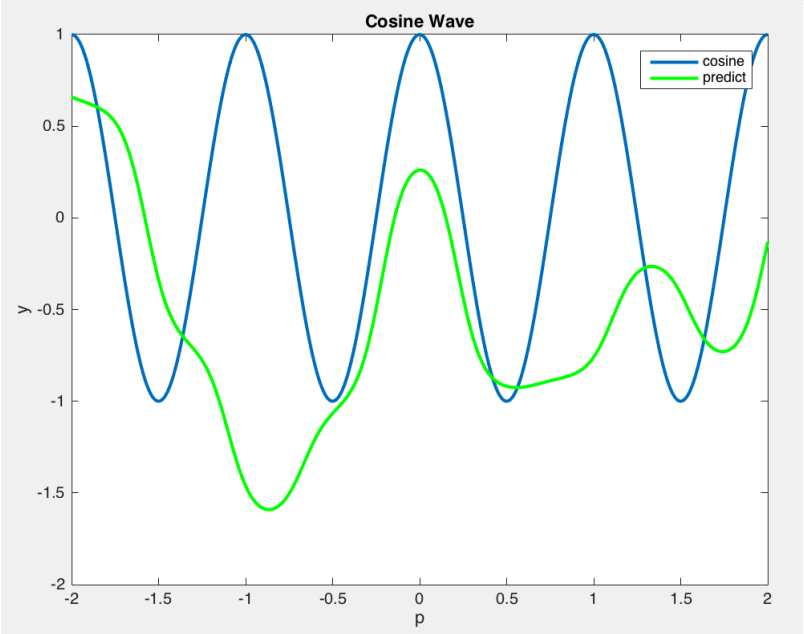


左边是lr=0.1，右边是lr=0.01

可以看到当步长从0.03改成0.1的时候，逼近的效果变好了，而改成0.01的效果就不怎么样，估计梯度下降的收敛速度还是太慢了。

另外我还发现到一个现象，当步长过大时(如改成0.3)，很容易出现cost上升的情形，进而导致训练提前终止，所以选择合适的学习步长还是非常重要的。



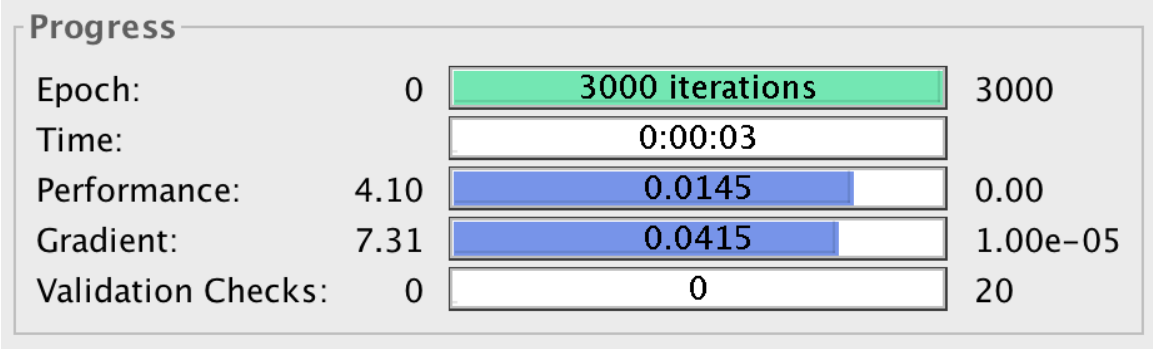


动量因子对算法性能的影响

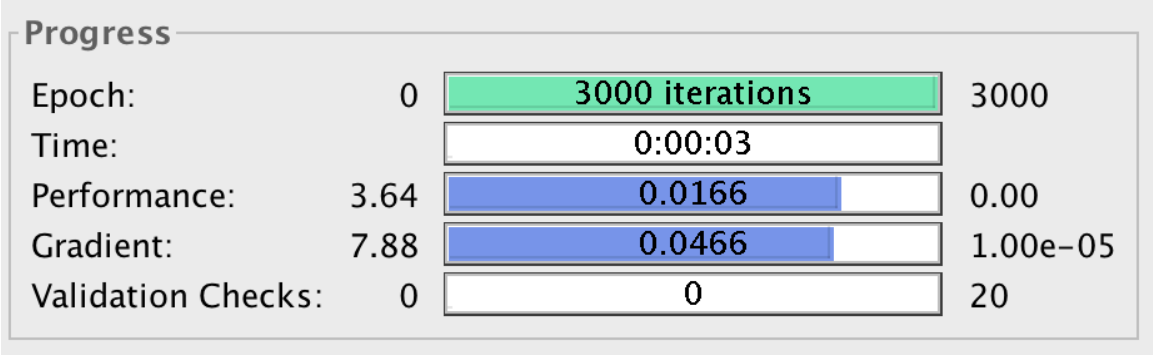
BPM的特色就是加入动量因子，因为梯度下降有陷入局部极小的问题存在，加入动量因子后，能让反馈的误差信号使神经元的权值重新振荡起来。

计算公式：dX = mc*dXprev + lr*(1-mc)*dperf/dX

Momentum constant 预设为0.9，测试改成0.09和9



mc = 0.09

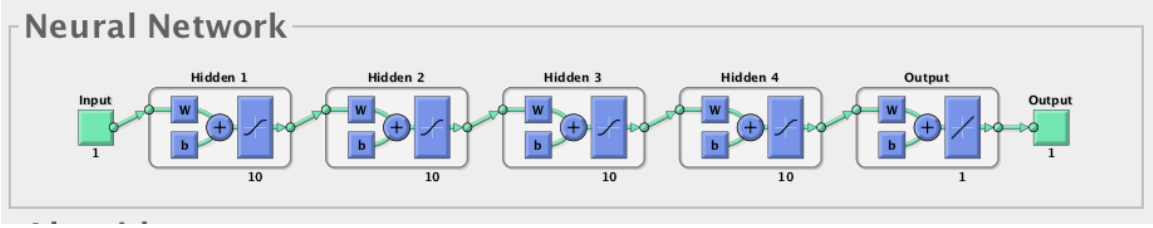


mc = 9

可能因为函数是简单的cosine，所以动量因子在这里并没有造成显著的影响

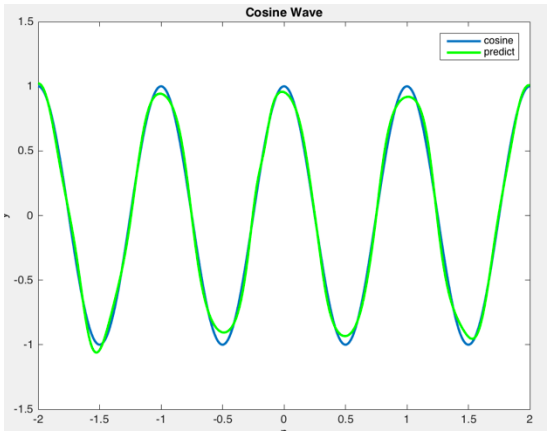
隐层数目对算法性能的影响

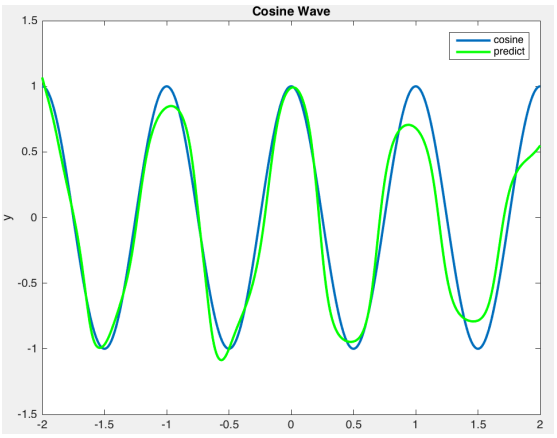
隐层预设为2，将其改为4, 6来做测试



当隐层=4时，有八成机率能跑完3000轮

但训练出来的结果并没有比2层来得好，在尖端的地方失真了

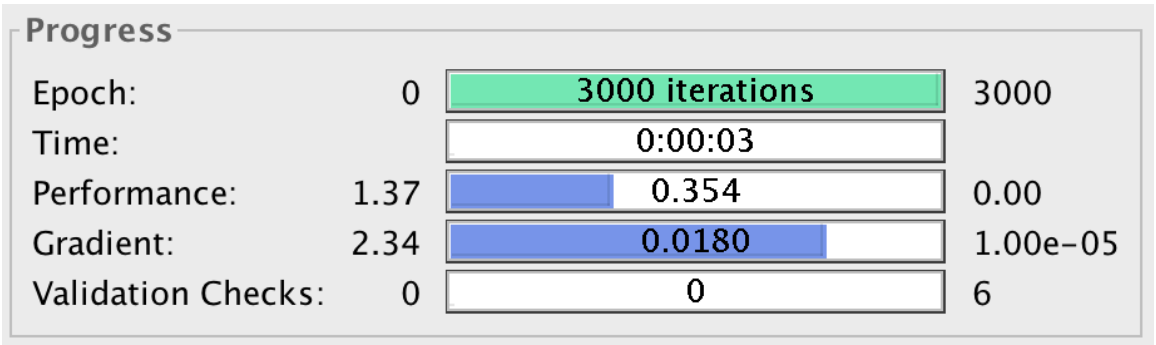




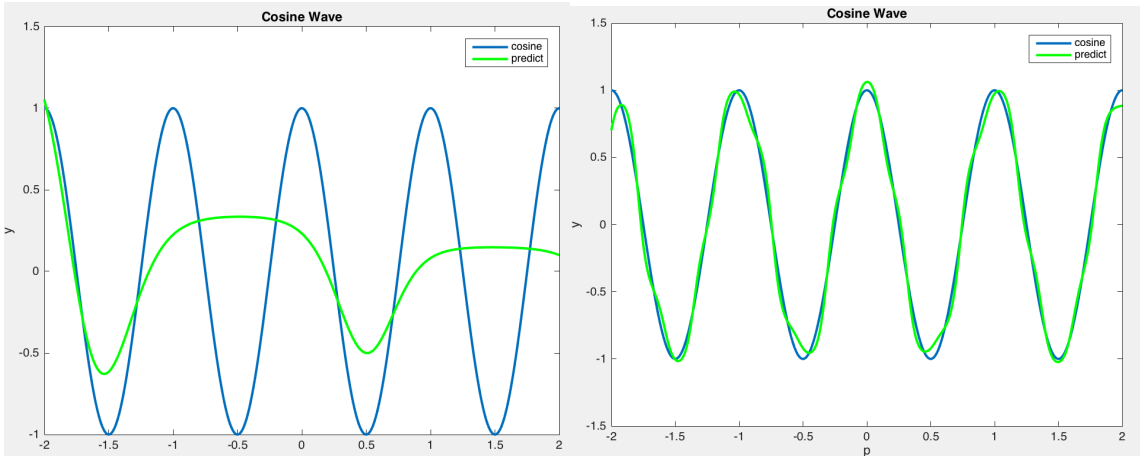
左边是隐层=4，右边是隐层=6

当隐层=6时，八成的网络都会在300轮前终止，而且训练的成果特别差
自己推断可能是因为函数所需要学习的特徵少，维度小，在隐层多的情况下反而学习效果差，隐层还是适当就好。
如果是拥有多维特徵的数据集，多隐层的效果应该会比较好。

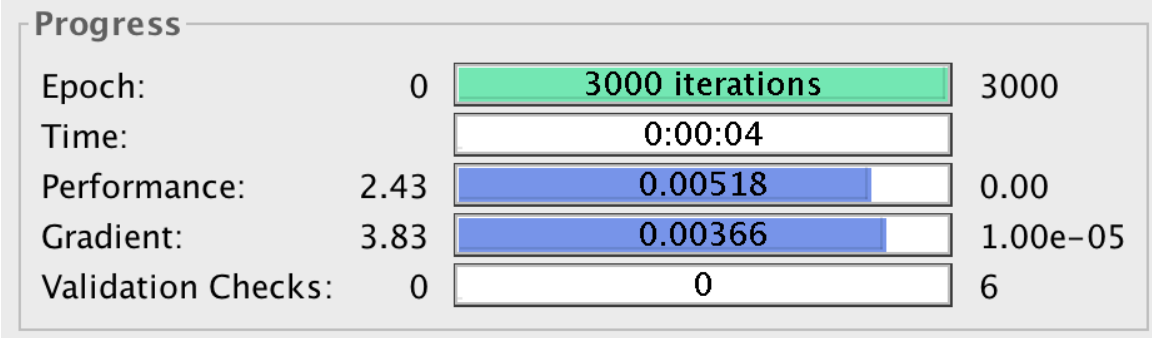
隐单元数目对算法性能的影响
原本的隐单元数目=10，更改成5,20来进行测试



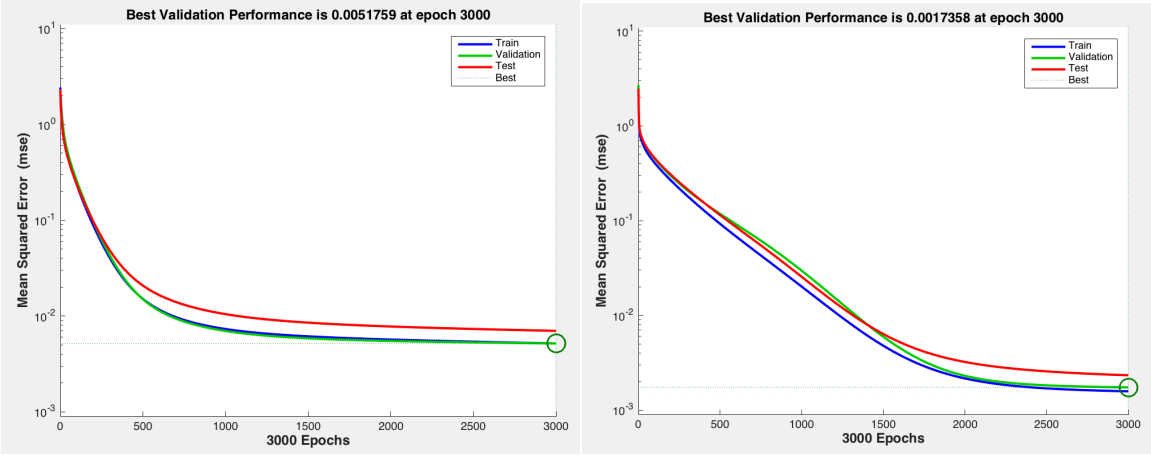
当隐单元数目=5，训练的效率极差无比，学习成果十分糟糕



左边是隐单元数=5，右边是隐单元数=20



当隐单元数=20,收敛的速度极快，下图是一个的比较图



左边是隐单元数=20，右边是隐单元数=10

可以看到左边大概在700轮左右就已经收敛到10^2，而右边约1300轮才收敛到10^2

但我认为就结果而言，隐单元数=20可能会有overfitting的问题

正常版的隐单元数=10看起来跟我所预期的cosine函数更为接近

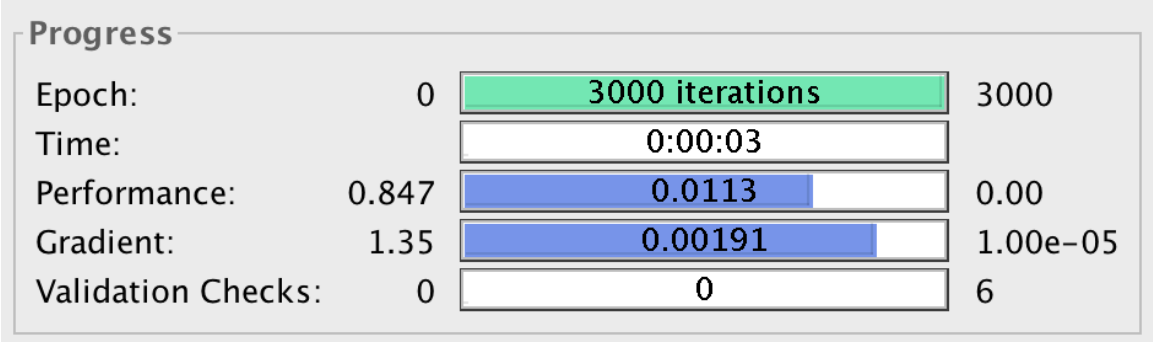
网络的推广性

根据内部函式dividerand所分，资料将拿70%去做Training、15%做Validation, 15%做test，所以在训练的过程中，也同时也保证了网络的推广性良好。

网络的抗噪性

在cosine函数的地方，加上一个0.1倍高斯分布的随机数当做噪音

我想来试试看ANN到底有多强大，能否成功预测cosine函数呢？

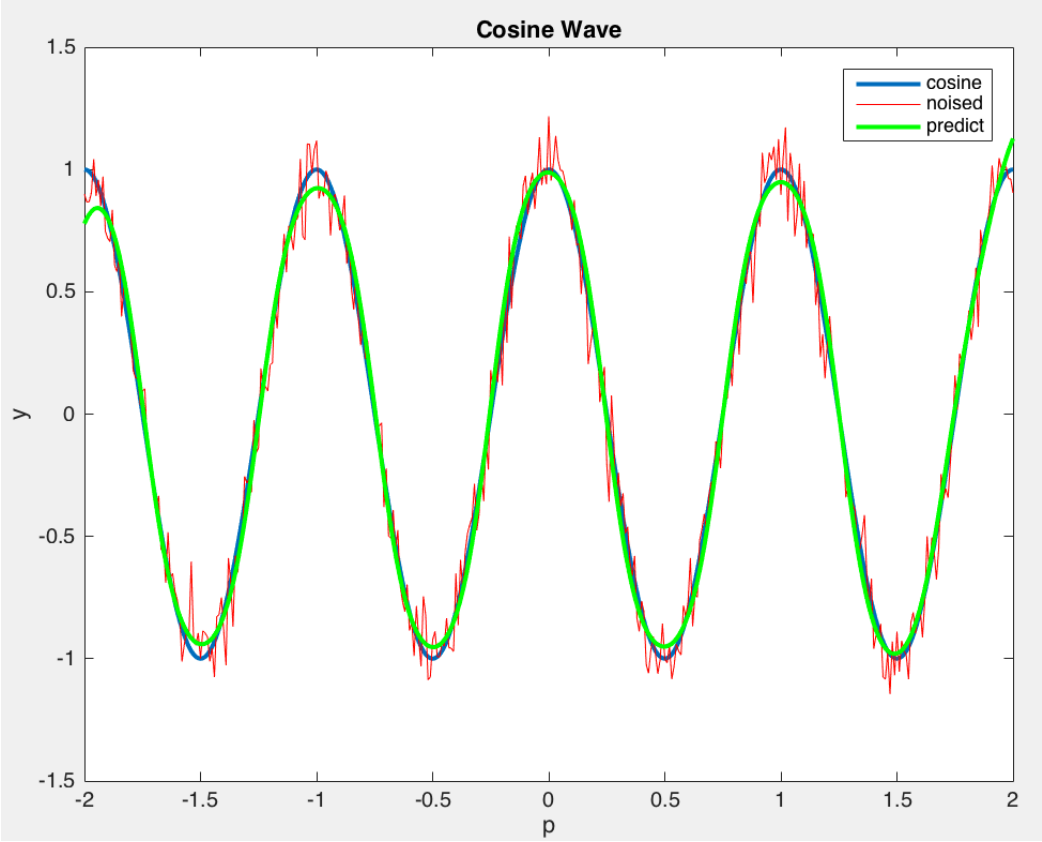


逼近效果图：图片分成三个部分呈现

第一部分是原本的cosine函数

第二部分是加噪的cosine函数，输入到training set

第三部分是预测的cosine函数，给定一段输入



网络满成功的将cosine函数预测出来，本身的抗噪能力看起来还挺不错的！

不过如果将学习步长加大＆轮数增加，那肯定就会Overfitting

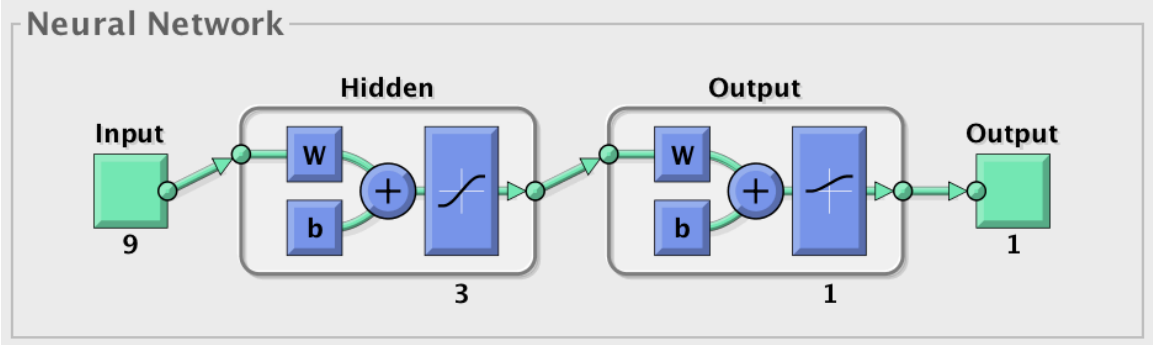
如果改用别的优化算法如：Levenberg-Marquardt、BFGS Quasi-Newton等

不仅训练速度快、还能够避免overfitting

用过之后的心得只能說：Gradient Descanting 的确是个经典，但如果要应用到实战的话还是换个比较现代/先进的算法吧！

2. 利用三层网络解决字母 T 与 L 的识别问题,假设每个字母用 3*3 的二维二值图表示,令 黑方格为 1,白方格为 0,每个字母有 4 个样本,包括字母正常位置以及旋转 90,180,270 度的图像,希望输入不同的位置下的 T 时网络输出为 1,输入 L 时网络输出为 0。试利用 BP 算法构造解决该问题的 9-3-1 网络。(50’)

开发环境：Matlab 2015



这是我设计的9-3-1网络，并且采用Gradient Descent来训练网络

参数设定

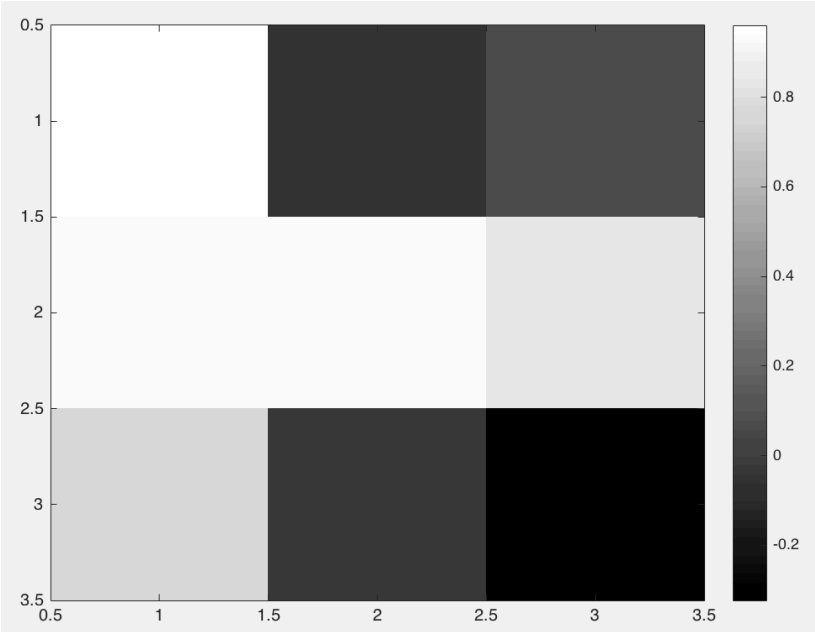
```
%% Training Params
net.trainParam.lr=0.01;      % Learning rate
net.trainParam.epochs=1000; % Maximum number of epochs to train
net.trainParam.show=20;      % Epochs between displays
% Set all data to Training
net.divideParam.trainRatio=1;  %Train(default=0.7)
net.divideParam.valRatio=0;    %Validation(default=0.15)
net.divideParam.testRatio=0;   %Test(default=0.15)
```

这边特别注意divideParam，我必须把所有的样本都拿来training
因为我们的样本数实在太少了，只有8个
如果再切给Test，训练出来的识别效果极差无比

输入和输出

输入x：有九个维度, 9x8矩阵（T跟L的原始数据，3x3=9维）
输入x_noised：在输入x加上一个0.2倍高斯分布的随机数噪音
输入t：[1,1,1,1,0,0,0,0], 1x8矩阵, T=1, L=0

加上噪音后的输入x_noised



输出y：输入x后所得到的预测值，1x8矩阵

	1	2	3	4	5	6	7	8
1	0.6194	0.7279	0.9758	0.9537	0.0344	0.0318	0.1541	0.0464

输出y_noised：输入x_noised后所得到的预测值，1x8矩阵

	1	2	3	4	5	6	7	8
1	0.5893	0.7082	0.9851	0.9581	0.0369	0.2521	0.1696	0.0698

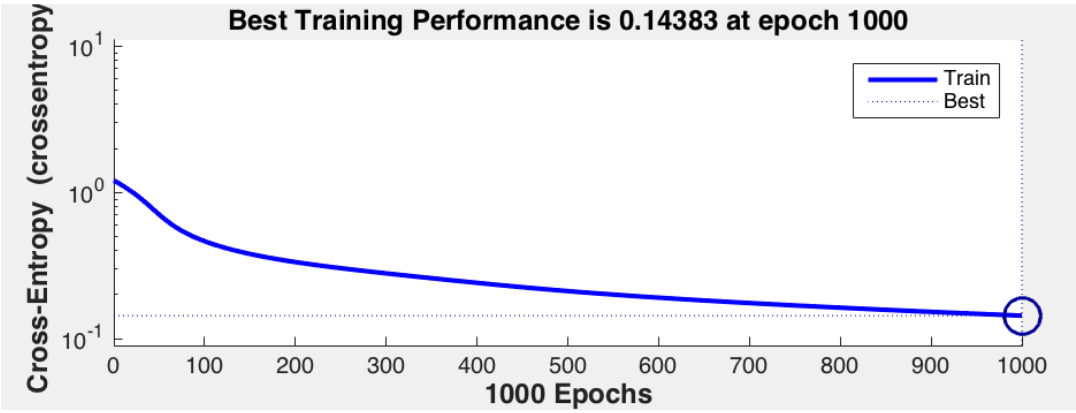
越靠近1/0，就代表网络觉得输入x比较像 T/L

经过四舍五入后，可以得到分类的结果

	1	2	3	4	5	6	7	8
1	1	1	1	1	0	0	0	0

跟原本的输入t完全相同，分类成功

训练的Performance，Gradient Descent的收敛速度越后面越缓慢……



随机图形预测

写一个叫做predict_rand_graph的函式，随机产生1x9由0,1所组成的阵列
并且丢入网络中去预测，效果图如下

```
x_rand_shape =          y_rand =

     1     1     1          0.5500
     0     1     0
     1     1     0     Result = T
```

分类的效果还是很不错的

心得

这是第一次完整地用Matlab实战了函数逼近、模式辨认的问题，逐渐感受到学习网络的强大，只要给他越多、越完整的训练资料，他就能做出更准确、全面的预测。
但美中不足的是，目前懂得学习算法只有Gradient Descent，为了写这次作业在网上翻了不少文档，看到以下的这张图，用不同算法来逼近sine函数的效能比较

Algorithm	Mean Time (s)	Ratio	Min. Time (s)	Max. Time (s)	Std. (s)
LM	1.14	1.00	0.65	1.83	0.38
BFG	5.22	4.58	3.17	14.38	2.08
RP	5.67	4.97	2.66	17.24	3.72
SCG	6.09	5.34	3.18	23.64	3.81
CGB	6.61	5.80	2.99	23.65	3.67
CGF	7.86	6.89	3.57	31.23	4.76
CGP	8.24	7.23	4.07	32.32	5.03
OSS	9.64	8.46	3.97	59.63	9.79
GDX	27.69	24.29	17.21	258.15	43.65

最下面那个是Gradient descent with momentum and adaptive learning rate backpropagation
很明显的落后了其他算法好几条街，整体的效能是很差的，期许未来能够学习更多的算法和网络，以期建立更有效率的网络
PS : Matlab文档，是最好的参考资料

给我自己的参考资料：Matlab的神经网络，可以使用的参数

Params	Default	Description
net.trainParam.epochs	1000	Maximum number of epochs to train
net.trainParam.goal	0	Performance goal
net.trainParam.lr	0.01	Learning rate
net.trainParam.lr_inc	1.05	Ratio to increase learning rate
net.trainParam.lr_dec	0.7	Ratio to decrease learning rate
net.trainParam.max_fail	6	Maximum validation failures
net.trainParam.max_perf_inc	1.04	Maximum performance increase
net.trainParam.mc	0.9	Momentum constant
net.trainParam.min_grad	1e-5	Minimum performance gradient
net.trainParam.show	25	Epochs between displays (NaNfor no displays)
net.trainParam.showCommandLine	false	Generate command-line output
net.trainParam.showWindow	true	Show training GUI
net.trainParam.time	inf	Maximum time to train in seconds