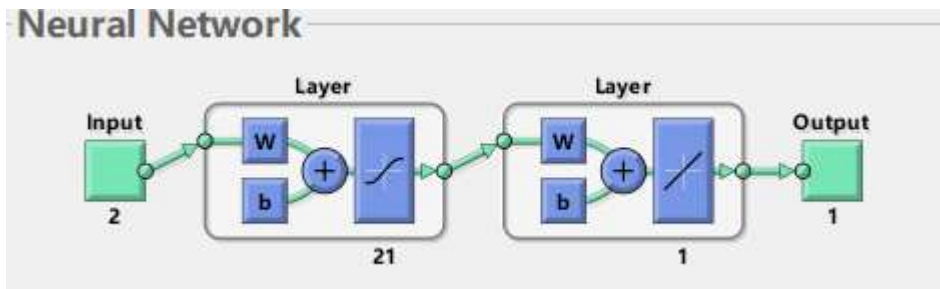


一．实验过程

1. 输入为二十五个矢量组成的张量：(-2, -2), (-2, -1), (-2, 0), (-2, 1), (-2, 2), (-1, -2), (-1, -1), (-1, 0), (-1, 1), (-1, 2), (0, -2), (0, -1), (0, 0), (0, 1), (0, 2), (1, -2), (1, -1), (1, 0), (1, 1), (1, 2), (2, -2), (2, -1), (2, 0), (2, 1), (2, 2), 输出为向下取整的矢量：(-2, -2, -1, -1, 0, -2, -1, -1, 0, 0, -1, -1, 0, 0, 1, -1, 0, 0, 1, 1, 0, 0, 1, 1, 2), 这个问题是想要拟合映射关系, 并达到进行泛化的目的, 也就是在给定的数据外给出的自变量, 我们希望训练出来的神经网络能给出在误差范围内的因变量。
2. 网络结构为输入层采用两个神经元, 输出层采用一个神经元, 然后由于是非线性函数, 所以我们需要采用隐含层, 具体神经元的个数不少于 2, 根据实验误差确认最好的隐藏层个数。其中激活函数的选择需要注意在输出层采用线性输出。



3. 学习方法：梯度下降法

1.1 初始化参数：

误差函数采用平方和函数

最大训练次数：100000 次

学习速率：0.001（学习速率低应该需要更大的 epoch 才能达到目标误差，但是如果是 0.01 的话就完全无法训练，程序运行时间非常短，而且随着 iterations 增加误差居然越来越大，所以选择这个就导致了经常 10000 次以内无法达到精度，但是我们只选择达到精度的数据进行观察……后来决定调整到 100000 不然会导致时间数据偏低，毕竟在 10000 次的时候观察 performance 已经很接近了）

目标误差：0.001

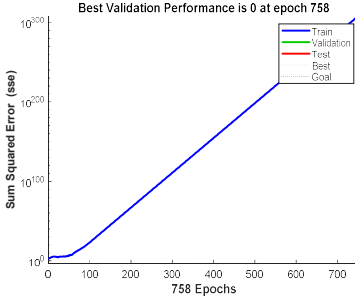
1.2 不同隐含层 S1 时的收敛速度与误差精度的对比分析

注：这里给出的值是给定了 S1 个数以后取平均值的结果

S1	时间 s	误差（通过减小学习速率等待时间变长这个误差可以小的多，但是我们达到 0.001 就停止训练了）
21(在上述 epoch 内几乎无法训练成功到精度内，我们只选择到了的六个数据做平均……但是后来决定增加数据实际上超过一万次以后大多几百几千次就到精度内了，但是我排除了一些因为随机	13.05067	0.001

初始值不好导致的异常数据, 尽量去除这个随机初始化的影响)		
20	13.65417	0.00106
19	12.0765	0.001
18	13.728333	0.001
17	16.26817	0.001
16	17.41633	0.001
…由于训练次数都比较大, 时间少的都是很偶然的情况, 这附近都是如此, 我跳几个数据		
12	训练了十多次, 无法在 30s 以内收敛, 并且差点很远, 推测要好几分钟	
从 11 试到了 4 发现一分多钟两分钟甚至十万次用完了都无法完成训练目标, 绝望之下尝试在 21 的基础上增加隐含层节点数进行训练		
22	7.785	0.001
23	7.801	0.001
24	5.844333	9.99E-04
25	4.238333	0.001
26	4.581833	9.99E-04
27	3.619667	9.99E-04
28	3.148	9.99E-04
29	2.642333	9.99E-04
30	3.3755	9.99E-04
31	1.717333	9.98E-04
32	2.639333	9.99E-04
33	2.144	9.98E-04
34	1.821667	9.98E-04

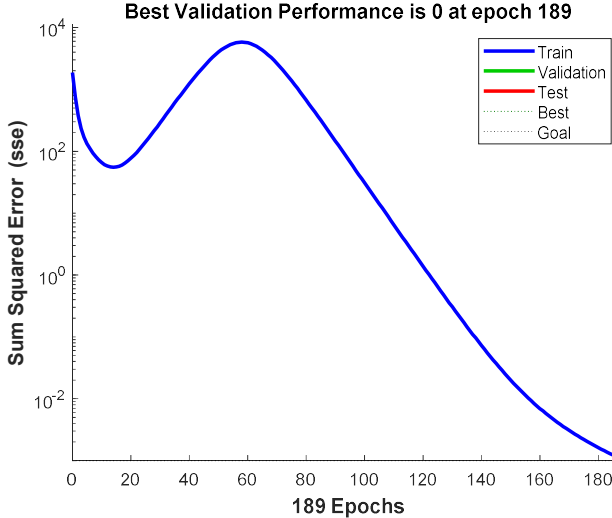
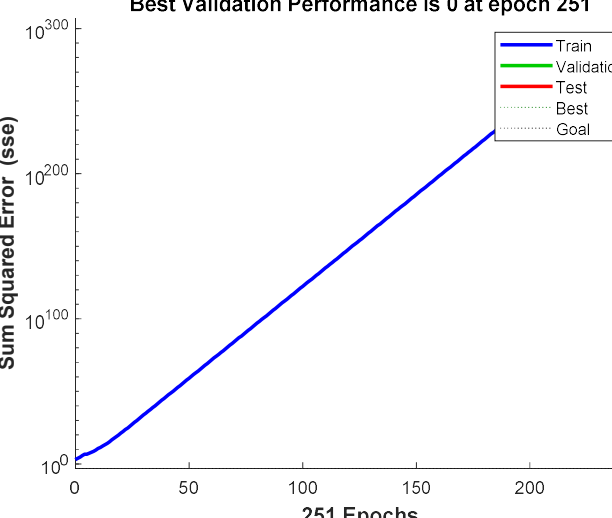
35	1.532667	9.98E-04
36	2.237667	9.99E-04
37	1.5305	9.99E-04
38	1.6185	9.97E-04
39	1.517167	9.98E-04
40	1.131167	9.98E-04
41	1.134333	9.99E-04
42	1.350333	9.98E-04
43	1.014667	9.97E-04
44	0.954667	9.97E-04
……快进到 50		
50	0.861667	9.96E-04
……快进到 60		
60	0.688	9.95E-04
……快进到 70		
70	0.549333	9.95E-04
80	0.499667	9.90E-04
90	0.437167	9.93E-04
100	0.363833	9.85E-04
150	0.329333	9.85E-04
140	0.291	9.69E-04
130	0.311667	9.73E-04
120	0.314667	9.84E-04

110	0.403333	9.89E-04
160	0.7680	
我猜测在 S1 等于 160 的时候发生了梯度消失（梯度无穷大）的现象，无法获得结果，并且可以看到，时间最少的 S1 数目在 140 附近，我再来寻找能完成拟合的最小的 S1 数目，但是以下只要拟合成功就记录，不再取平均值。		
12	79.0090	9.9994e-04
11（无法完成训练这里只记录达到100000次后的最终数据）	195.6750	0.0121
13	42.8210	9.9999e-04
但是在我下面改变了误差测试了几组数据以后回来改变 S1 时间发生了很大的变化，我把数据再次记录，这里我很疑惑，matlab 应该不会像精密仪器一样有回程误差？但是我比较相信这次的数据是对的因为一次迭代花的时间会比 S1 小的时候九很多，原因还没找到。		
100	8.37E+00	9.88E-04
110	6.72E+00	9.92E-04
120	1.00E+01	9.82E-04
130	4.89E+00	9.72E-04
140	4.58E+00	9.88E-04
150	6.76E+00	9.79E-04
160 和之前一样，花时间 39.9550s		
90	9.06E+00	9.89E-04
80	1.28E+01	9.90E-04
可以看到，训练最快的仍然是在 140 附近，推测是前面电脑电量足使用的 gpu 计算现在用的 cpu？		

- 测试过程中，我们减少隐藏层节点数确实能减少训练的时间，但是并不明显：隐藏层结点数为 21 的时候训练一万次要花差不多 8.1 秒的时间，而隐藏层结点数为 4 的时候要花 7.5 秒左右，可以发现差别并不大；但是如果隐藏层结点少，就无法拟合这样一个非线性程度非常大非常奇怪的函数，导致训练很久都无法达到精度，其实不是精度打不到，是网络结构限制，根本拟合不了。经过我的试验，在隐藏层节点数为 11 次的时候就已经无法拟合这个映射关系了、
- 注意到，nntraintool 里面，其实判断的停止训练的时候是达到了 0.00100，0.00101 都不会停止。

1.3 S1 设置为较好的情况下，在训练过程中取始终不变的学习速率 lr 时，对 lr 值为不同值的训练时间包括稳定性进行观察比较：

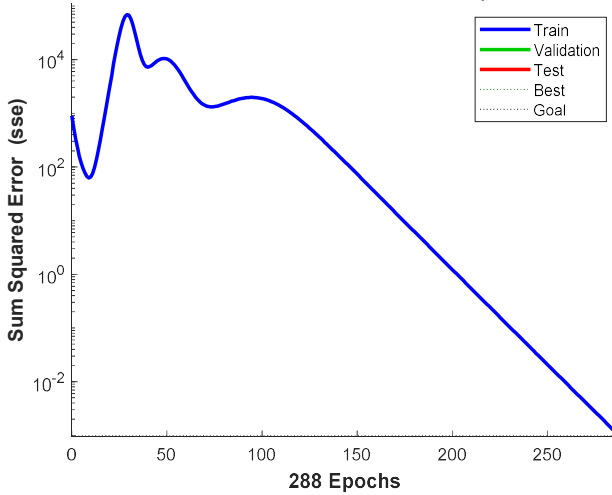
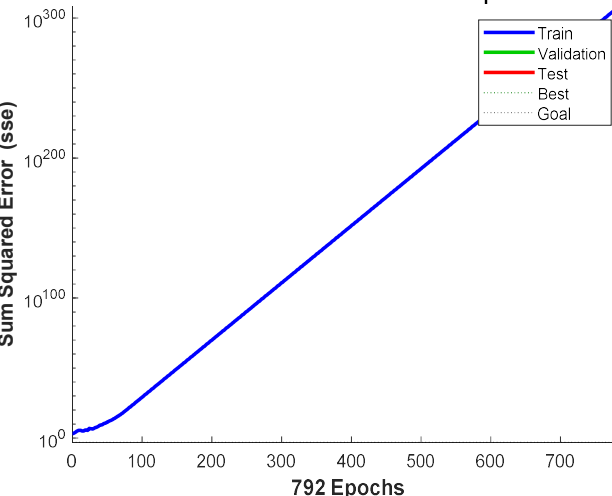
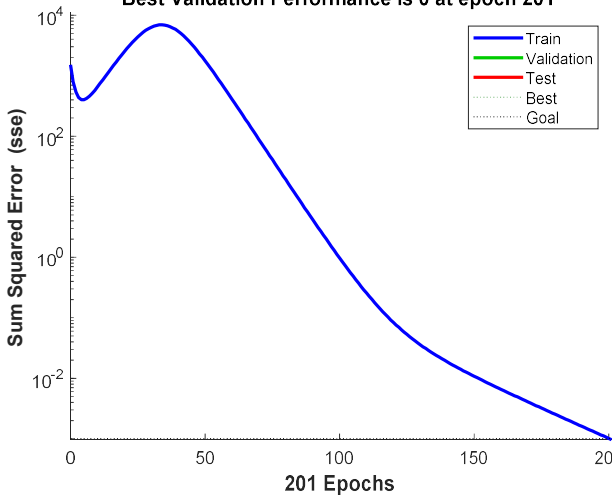
以下我们对每个 lr 值的训练时间不再取平均值因为只是定性分析，S1=140

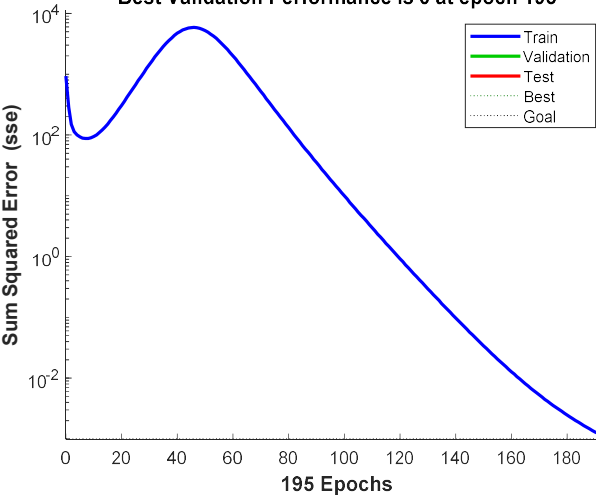
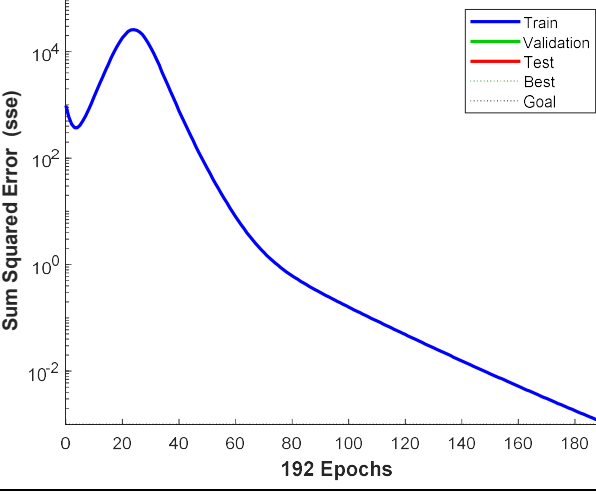
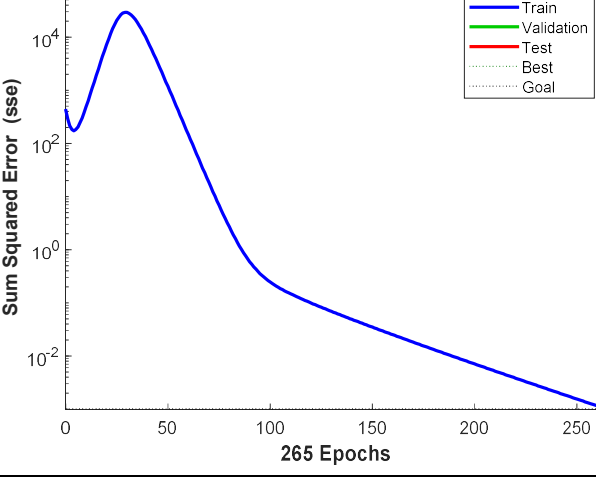
lr	训练时间	稳定性
0.001	29.3900	 <p>Best Validation Performance is 0 at epoch 189</p>
0.002	34.9190	 <p>Best Validation Performance is 0 at epoch 251</p>

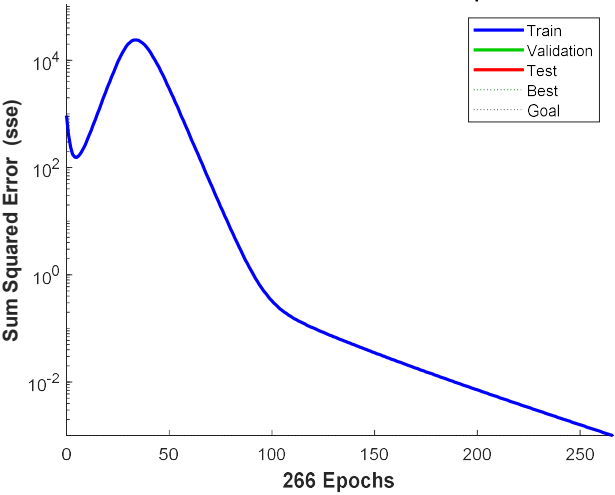
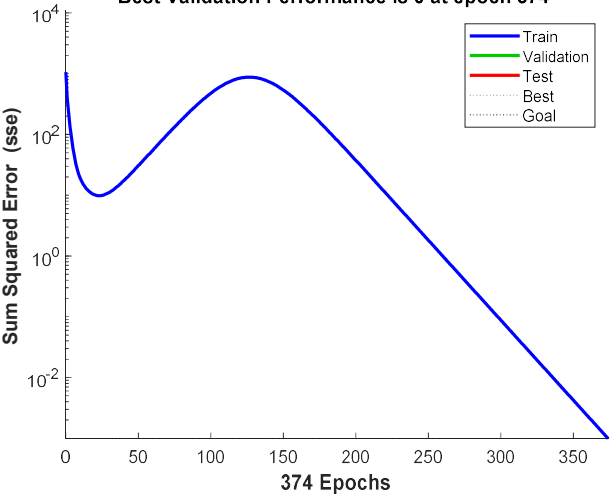
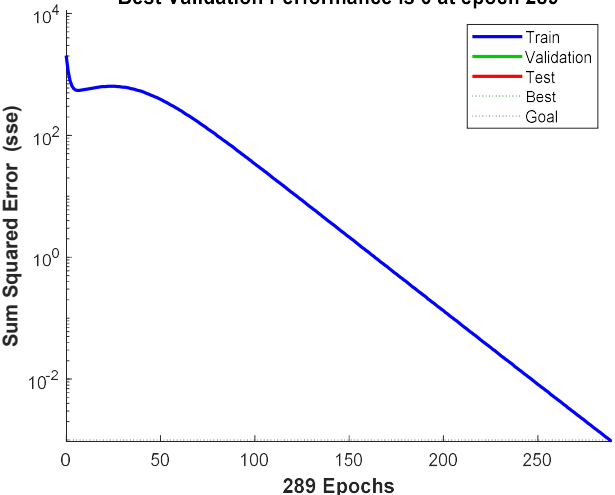
0.0009	30.6100	<p>Best Validation Performance is 0 at epoch 216</p> <p>Sum Squared Error (sse)</p> <p>216 Epochs</p> <p>Train Validation Test Best Goal</p>
0.0008	30.5690	<p>Best Validation Performance is 0 at epoch 244</p> <p>Sum Squared Error (sse)</p> <p>244 Epochs</p> <p>Train Validation Test Best Goal</p>
0.0007	26.3800	<p>Best Validation Performance is 0 at epoch 254</p> <p>Sum Squared Error (sse)</p> <p>254 Epochs</p> <p>Train Validation Test Best Goal</p>

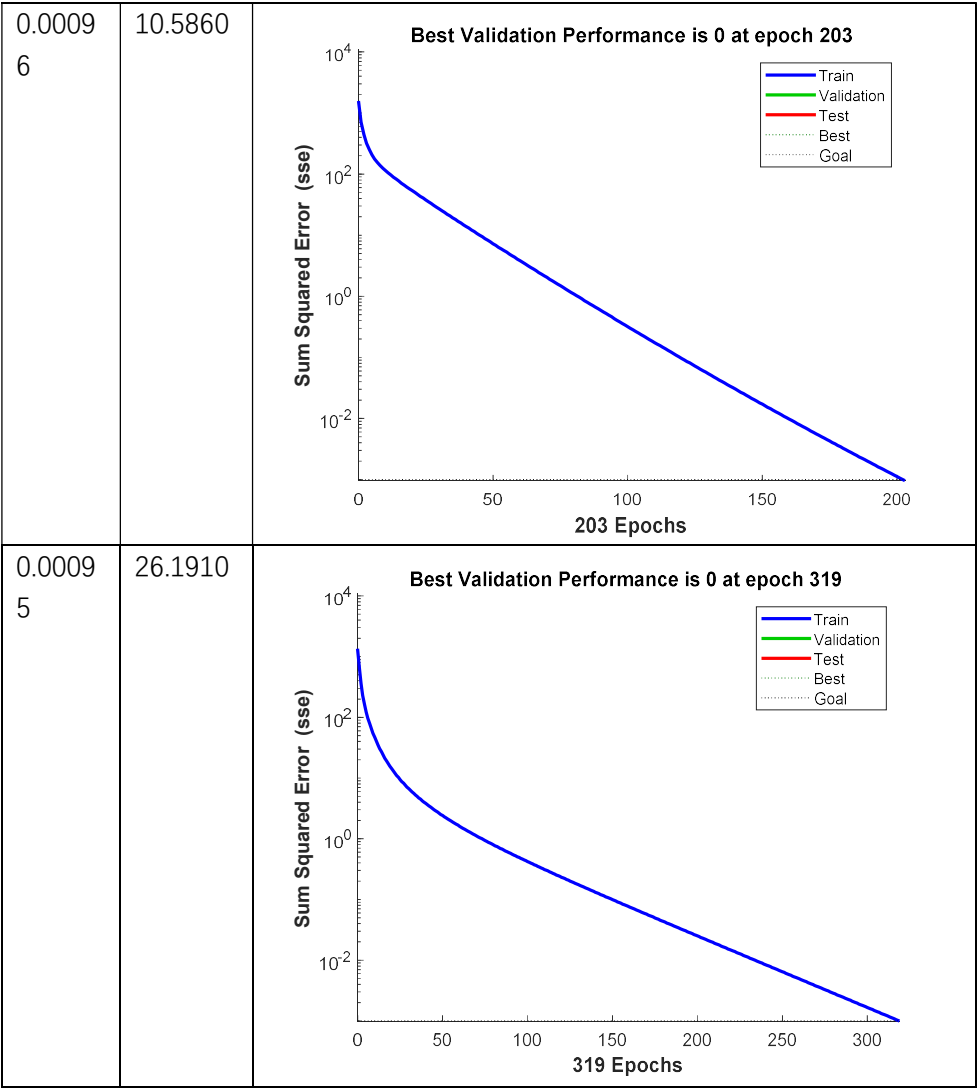
0.0006	55.5430	<p>Best Validation Performance is 0 at epoch 447</p> <p>Sum Squared Error (sse)</p> <p>447 Epochs</p> <p>Train Validation Test Best Goal</p>
0.0005	45.7100	<p>Best Validation Performance is 0 at epoch 357</p> <p>Sum Squared Error (sse)</p> <p>357 Epochs</p> <p>Train Validation Test Best Goal</p>
0.0004	78.7920	<p>Best Validation Performance is 0 at epoch 560</p> <p>Sum Squared Error (sse)</p> <p>560 Epochs</p> <p>Train Validation Test Best Goal</p>

0.0003	80.7490	<p>Best Validation Performance is 0 at epoch 585</p> <p>Sum Squared Error (sse)</p> <p>585 Epochs</p> <p>Train Validation Test Best Goal</p>
0.0002	263.623 0	<p>Best Validation Performance is 0 at epoch 1020</p> <p>Sum Squared Error (sse)</p> <p>1020 Epochs</p> <p>Train Validation Test Best Goal</p>
0.0001	383.325 0	<p>Best Validation Performance is 0 at epoch 2800</p> <p>Sum Squared Error (sse)</p> <p>2800 Epochs</p> <p>Train Validation Test Best Goal</p>

0.0011	35.8760	<p>Best Validation Performance is 0 at epoch 288</p> 
0.0012	107.166 0	<p>Best Validation Performance is 0 at epoch 792</p> 
0.0010 1	25.7850	<p>Best Validation Performance is 0 at epoch 201</p> 

0.0010 2	25.2700	<p>Best Validation Performance is 0 at epoch 195</p>  <p>Sum Squared Error (sse)</p> <p>195 Epochs</p>
0.0010 3	34.2060	<p>Best Validation Performance is 0 at epoch 192</p>  <p>Sum Squared Error (sse)</p> <p>192 Epochs</p>
0.0010 4	47.4370	<p>Best Validation Performance is 0 at epoch 265</p>  <p>Sum Squared Error (sse)</p> <p>265 Epochs</p>

0.0009 9	17.8560	<p>Best Validation Performance is 0 at epoch 266</p>  <p>Sum Squared Error (sse)</p> <p>266 Epochs</p>
0.0009 8	27.7380	<p>Best Validation Performance is 0 at epoch 374</p>  <p>Sum Squared Error (sse)</p> <p>374 Epochs</p>
0.0009 7	20.3190	<p>Best Validation Performance is 0 at epoch 289</p>  <p>Sum Squared Error (sse)</p> <p>289 Epochs</p>



虽然没有时间制表，但是数据基本收集完毕，观察上图的规律我们可以得到收敛最快最稳定的是 0.00096，往下变小应该是（没有找到更多小值的数据）肯定可以找到最小误差点，并且越小误差曲线越稳定下降，但是往大达到 0.0012 就无法训练收敛了。

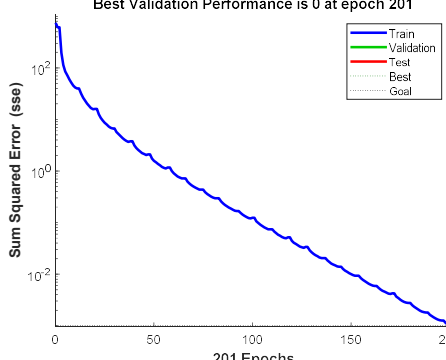
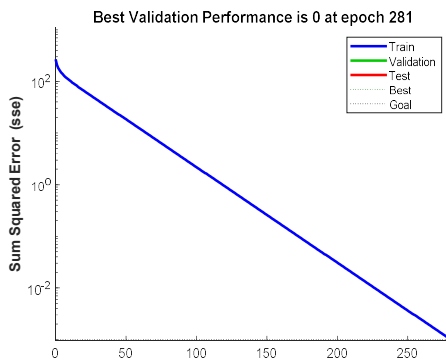
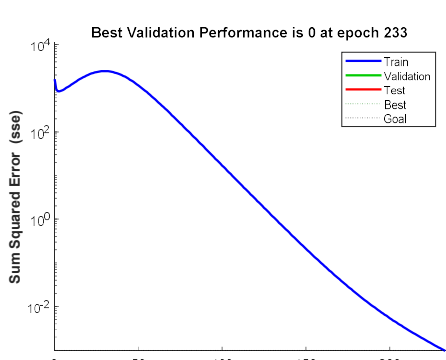
1.3 采用自适应学习速率，与单一固定的学习速率 lr 中最好的情况进行对比训练的观察。

代码改变：

```
traingda
net.trainParam.lr=0.001;
net.trainParam.lr_inc=1.05;
net.trainParam.lr_dec=0.7;
```

经过我粗略的测试大概还是在系统设定的初始值附近速度最快，我们列出结果并且和固定 lr 为 0.00096 的梯度下降法进行对比，我们取的是六次实验平均值数据，稳定性由于是定性观察只选取其中某次实验的结果：

方法	初始值	收敛时间 s	稳定性
----	-----	-----------	-----

Traingd	Lr=0.0009 6	1.94E+0 1	<p>Best Validation Performance is 0 at epoch 201</p> 
Traingda	Lr=0.001	2.90E+0 1	<p>Best Validation Performance is 0 at epoch 281</p>  <p>Best Validation Performance is 0 at epoch 233</p> 

可以看到，使用自适应值学习速率的方法时间会更快（我不确定是不是这样，因为一开始从 traingd 改成 traingda 的时候速度非常快在两秒以内，推测用的 cpu 不同的计算单元？），并且更加稳定，用 traingd 训练的时候偶尔可能出现较大的波动，但是 traingd 是平稳下降的。

二． 实验分析和实验结论

1. 给出网络的初始值、最终权值，以及最终误差与训练次数和所采用的算法：
输入层到隐含层的权值 =

-1.5617	-8.1340
3.3932	-7.5556
-6.5008	5.1322
-6.2374	-5.4492
3.9827	7.2621

-8.2775	0.2888
-6.0456	-5.6613
-3.4159	-7.5453
6.4480	5.1983
3.5837	7.4671
-5.5617	6.1374
-6.4255	-5.2262
-7.4601	-3.5982
-5.5567	-6.1419
-6.7688	-4.7732
5.8492	-5.8641
-6.2802	-5.4000
-5.9208	5.7917
-6.1036	5.5988
-8.2810	-0.1571
1.2011	8.1950
8.0064	-2.1207
-5.0726	6.5475
-6.3950	5.2635
-8.2730	-0.3978
7.6764	3.1102
-6.8824	4.6078
-4.9968	6.6055
-7.4326	3.6547
-8.0904	-1.7736
4.4618	-6.9780
-3.6194	-7.4498
-3.9506	7.2796
2.0757	8.0182
-7.3495	3.8190
7.4728	3.5717
-4.0712	-7.2129
5.4791	-6.2112
-8.1875	1.2506
7.6084	-3.2729
-6.5653	-5.0494
4.8857	6.6880
-6.8287	4.6872
-7.6798	3.1017
-7.9920	-2.1742
-1.7244	-8.1010
-1.1374	-8.2040
-6.5638	-5.0514
-6.5376	-5.0853

7.8434	2.6609
2.6045	-7.8624
7.0409	4.3619
1.2619	-8.1858
8.2501	-0.7316
-6.7583	-4.7880
-7.3107	3.8928
6.0810	5.6233
-5.5801	-6.1206
-7.9119	-2.4498
6.2628	5.4201
-5.7660	-5.9459
-5.3524	6.3208
7.9584	2.2944
-7.3477	-3.8225
7.9550	-2.3060
4.4247	-7.0016
6.2410	5.4451
-6.2447	-5.4409
2.1597	7.9960
-4.3986	-7.0180
4.3846	7.0267
-8.2712	-0.4327
3.5087	-7.5026
-7.5192	3.4730
-3.0319	-7.7076
6.4736	5.1665
-7.9050	2.4719
-5.5474	6.1503
3.8253	-7.3462
7.8474	-2.6493
-5.8605	-5.8527
4.4013	7.0163
8.0867	-1.7904
6.0462	-5.6607
5.0601	6.5571
-5.2481	-6.4076
-3.2026	7.6383
2.5983	-7.8644
-8.2450	-0.7870
6.2843	5.3952
7.9959	2.1601
0.5875	-8.2617
8.2682	-0.4868

7.7749	2.8549
-6.4421	-5.2057
-3.7325	-7.3938
-4.5626	-6.9125
-6.5000	5.1332
-7.9377	-2.3650
5.6215	6.0826
6.5852	5.0234
1.3273	-8.1755
-6.1148	5.5865
-4.1159	-7.1874
7.7985	-2.7898
4.4869	-6.9619
5.2216	6.4292
5.0914	6.5328
5.3717	6.3043
8.1938	-1.2087
3.9309	-7.2903
-6.2183	-5.4711
-8.1761	-1.3234
-7.1524	-4.1764
-2.9277	7.7478
-8.2522	0.7077
-3.4566	-7.5267
-8.2825	-0.0076
-2.2820	-7.9619
-4.1095	-7.1911
4.9301	-6.6554
-7.9003	2.4870
-8.2615	-0.5902
-4.8086	6.7437
-4.1158	7.1875
-7.4918	-3.5317
-6.9180	4.5543
-3.3456	-7.5767
7.2551	3.9955
-1.2073	-8.1940
8.1696	-1.3627
-6.4194	-5.2336
-6.3372	5.3329
1.9804	8.0423
2.5123	-7.8923
-3.2699	7.6097
5.1855	6.4584

5.3276	-6.3417
7.9721	-2.2461
-2.4942	-7.8980

隐含层到输出层的权值 =

列 1 至 7

-0.1007	0.8051	-0.9667	0.8232	-0.3827	0.7019	-0.8653
---------	--------	---------	--------	---------	--------	---------

列 8 至 14

-0.5114	-0.3077	-0.1083	-0.7473	0.0463	0.6885	0.3428
---------	---------	---------	---------	--------	--------	--------

列 15 至 21

0.8566	-0.1067	-0.6231	-0.1821	-0.7806	-0.7258	-0.7736
--------	---------	---------	---------	---------	---------	---------

列 22 至 28

0.5923	0.8440	-0.1201	-0.2931	0.5029	-0.0372	0.1337
--------	--------	---------	---------	--------	---------	--------

列 29 至 35

-0.6265	-0.1879	-0.2575	-0.8025	0.8582	0.5056	-0.8196
---------	---------	---------	---------	--------	--------	---------

列 36 至 42

-0.3097	-0.4715	-0.0406	-0.0126	0.8415	-0.6706	0.6778
---------	---------	---------	---------	--------	---------	--------

列 43 至 49

0.5334	0.0766	-0.7118	0.2026	0.8015	-0.5356	0.2059
--------	--------	---------	--------	--------	---------	--------

列 50 至 56

0.3852	-0.9354	-0.3537	0.1918	0.1130	-0.4808	-0.5117
--------	---------	---------	--------	--------	---------	---------

列 57 至 63

-0.3376	0.7669	-0.0784	-0.9038	0.2529	0.8715	0.2650
---------	--------	---------	---------	--------	--------	--------

列 64 至 70

0.4276	-0.9290	-0.0837	-0.4216	0.9305	-0.1634	-0.9075
--------	---------	---------	---------	--------	---------	---------

列 71 至 77

0.2391	-0.0738	-0.5471	0.9092	0.2102	0.6508	-0.6993
--------	---------	---------	--------	--------	--------	---------

列 78 至 84

0.1890	0.5121	-0.7535	0.9760	-0.8629	-0.6020	-0.5427
--------	--------	---------	--------	---------	---------	---------

列 85 至 91

-0.3655	0.0184	-0.2283	0.9055	-0.3358	0.3293	0.6122
---------	--------	---------	--------	---------	--------	--------

列 92 至 98

-0.8446	-0.8421	-0.8845	-0.2580	0.2559	0.6665	-0.2328
---------	---------	---------	---------	--------	--------	---------

列 99 至 105

0.0214	-0.4508	-0.2123	0.5885	-0.7995	-0.6904	-0.8939
--------	---------	---------	--------	---------	---------	---------

列 106 至 112

-0.1167	0.9708	-0.8028	0.6348	0.2568	-0.0202	-0.0742
---------	--------	---------	--------	--------	---------	---------

列 113 至 119

0.0773	-0.2716	0.3660	0.8687	-0.0189	0.1564	-0.4122
--------	---------	--------	--------	---------	--------	---------

列 120 至 126

0.2999	0.7884	0.5807	-0.9772	0.2700	-0.8772	-0.1768
--------	--------	--------	---------	--------	---------	---------

列 127 至 133

0.8687	-0.9290	0.5586	0.3914	-0.9996	-0.0251	-0.5095
--------	---------	--------	--------	---------	---------	---------

列 134 至 140

0.2007	0.2737	0.3774	-0.2723	0.2925	0.1690	-0.9853
--------	--------	--------	---------	--------	--------	---------

隐含层的偏置 =

16.5650
-16.3267
16.0883
15.8500
-15.6116
15.3733
15.1349
14.8966
-14.6583
-14.4199
14.1816
13.9432
13.7049
13.4665
13.2282
-12.9898
12.7515
12.5131
12.2748
12.0365
-11.7981
-11.5598
11.3214
11.0831
10.8447
-10.6064
10.3680
10.1297
9.8913
9.6530
-9.4147
9.1763
8.9380
-8.6996
8.4613
-8.2229
7.9846
-7.7462
7.5079
-7.2695
7.0312
-6.7929

6.5545
6.3162
6.0778
5.8395
5.6011
5.3628
5.1244
-4.8861
-4.6477
-4.4094
-4.1710
-3.9327
3.6944
3.4560
-3.2177
2.9793
2.7410
-2.5026
2.2643
2.0259
-1.7876
1.5492
-1.3109
-1.0726
-0.8342
0.5959
-0.3575
0.1192
0.1192
-0.3575
0.5959
-0.8342
-1.0726
1.3109
-1.5492
-1.7876
2.0259
2.2643
-2.5026
2.7410
2.9793
3.2177
3.4560
-3.6944

-3.9327
4.1710
-4.4094
4.6477
4.8861
5.1244
5.3628
5.6011
-5.8395
-6.0778
-6.3162
-6.5545
-6.7929
7.0312
7.2695
7.5079
-7.7462
-7.9846
8.2229
8.4613
8.6996
8.9380
9.1763
9.4147
9.6530
-9.8913
-10.1297
-10.3680
-10.6064
-10.8447
-11.0831
-11.3214
-11.5598
-11.7981
12.0365
-12.2748
-12.5131
-12.7515
-12.9898
-13.2282
-13.4665
-13.7049
13.9432
-14.1816

14.4199
-14.6583
-14.8966
15.1349
15.3733
-15.6116
15.8500
16.0883
16.3267
-16.5650

输出层的偏置 =

-0.7913

随机初始化网络以后给定训练数据的输入得到的输出的误差的平方和 =

25.8007

输入层到隐含层的权值 =

-1.5649	-8.1389
3.3910	-7.6167
-6.4087	4.9365
-6.2431	-5.4363
4.0757	7.3079
-8.1724	0.4459
-6.0419	-5.6617
-3.2961	-7.4835
6.4461	5.1976
3.6045	7.4769
-5.5637	6.1545
-6.4246	-5.2268
-7.4314	-3.5695
-5.5638	-6.1504
-6.7402	-4.7928
5.8295	-5.8270
-6.2782	-5.3645
-5.9121	5.7908
-6.2044	5.7673
-8.2809	-0.1572

1.2719	8.2301
8.0372	-2.0037
-4.9463	6.3874
-6.3960	5.2705
-8.2732	-0.3986
7.6054	3.0493
-6.8885	4.5894
-4.9971	6.6012
-7.4265	3.6840
-8.0958	-1.7786
4.4789	-6.9466
-3.6059	-7.4597
-3.9590	7.2304
2.0598	8.0453
-7.3945	3.9313
7.4939	3.5406
-4.0878	-7.2500
5.4655	-6.2401
-8.2016	1.2312
7.7013	-3.4126
-6.6065	-5.0510
4.8871	6.7854
-6.7989	4.6844
-7.6793	3.0690
-8.0811	-2.0934
-1.7265	-8.1008
-1.2289	-8.1586
-6.5697	-5.1301
-6.5370	-5.0560
7.8885	2.6218
2.5095	-7.9223
7.0495	4.2967
1.2820	-8.1772
8.2502	-0.7318
-6.6987	-4.8842
-7.1985	4.0422
6.0819	5.6206
-5.5780	-6.1198
-7.9015	-2.4775
6.2993	5.3800
-5.7659	-5.9459
-5.3673	6.3084
7.9519	2.3315
-7.3520	-3.8093

7.9525	-2.2870
4.3997	-7.0118
6.2182	5.4679
-6.1686	-5.5169
2.1593	7.9960
-4.4012	-7.0152
4.3809	7.0279
-8.2712	-0.4058
3.5858	-7.4633
-7.6731	3.1672
-3.0300	-7.7082
6.4883	5.1550
-7.8553	2.5171
-5.5417	6.1559
3.8332	-7.3539
7.9498	-2.5154
-5.8606	-5.8518
4.3224	7.0954
8.1023	-1.7857
6.0432	-5.6653
5.0857	6.5335
-5.2217	-6.4238
-3.2150	7.6339
2.5421	-7.9218
-8.2446	-0.7861
6.2228	5.5209
7.9963	2.0210
0.6157	-8.2464
8.2759	-0.4665
7.7588	3.0841
-6.4650	-5.2055
-3.7340	-7.3913
-4.5621	-6.9282
-6.5986	5.1048
-7.9505	-2.3595
5.6678	5.9822
6.5820	5.0125
1.2989	-8.1757
-6.2618	5.5177
-4.0774	-7.1767
7.7114	-2.7861
4.4953	-6.9761
5.1436	6.5808
5.1464	6.4304

5.3661	6.3200
8.2243	-1.2121
3.9286	-7.2962
-6.2184	-5.4666
-8.1784	-1.3346
-7.1255	-4.2155
-2.8113	7.8398
-8.2571	0.7248
-3.4573	-7.5063
-8.2825	-0.0075
-2.2977	-8.0464
-4.0692	-7.1168
4.9453	-6.6699
-7.8554	2.5378
-8.2583	-0.5869
-4.8062	6.7405
-4.1156	7.1864
-7.4765	-3.5084
-6.9432	4.5544
-3.4707	-7.6195
7.2022	3.9122
-1.3163	-8.0354
7.9522	-1.4706
-6.4198	-5.2336
-6.3357	5.3308
1.9900	8.0310
2.5119	-7.8906
-3.2816	7.6176
5.1418	6.4434
5.3112	-6.3323
7.9731	-2.2445
-2.4542	-7.6972

隐含层到输出层的权值 =

列 1 至 7

-0.3551	0.2497	-0.5528	0.8739	-0.2408	0.8700	-0.8170
---------	--------	---------	--------	---------	--------	---------

列 8 至 14

-0.7102	-0.3564	-0.0387	-0.2399	0.0926	0.6955	0.4008
---------	---------	---------	---------	--------	--------	--------

列 15 至 21

0.8774	-0.5776	-0.5631	0.2282	-0.4412	-0.6100	-0.4309
--------	---------	---------	--------	---------	---------	---------

列 22 至 28

0.1444	0.8895	0.0350	-0.1767	0.3904	0.1548	0.0175
--------	--------	--------	---------	--------	--------	--------

列 29 至 35

-0.1543	0.0426	0.0902	-0.4972	0.2864	0.3015	-0.4434
---------	--------	--------	---------	--------	--------	---------

列 36 至 42

-0.5868	-0.2385	0.2057	0.1403	0.8355	-0.4403	0.5563
---------	---------	--------	--------	--------	---------	--------

列 43 至 49

0.6041	-0.0040	-0.5781	0.2334	0.7692	-0.4920	0.2235
--------	---------	---------	--------	--------	---------	--------

列 50 至 56

0.3271	-0.3610	-0.4566	0.3799	-0.1800	-0.5542	-0.7485
--------	---------	---------	--------	---------	---------	---------

列 57 至 63

-0.1291	0.5561	-0.0143	-0.7026	0.0402	0.2962	0.1926
---------	--------	---------	---------	--------	--------	--------

列 64 至 70

0.1248	-0.7991	0.3268	-0.1834	0.6808	0.0547	-1.0519
--------	---------	--------	---------	--------	--------	---------

列 71 至 77

0.3698	-0.0739	-0.4658	0.6725	0.0531	0.8926	-0.7987
--------	---------	---------	--------	--------	--------	---------

列 78 至 84

0.0451	0.4080	-0.5275	0.7300	-0.5405	-0.3025	-0.3539
--------	--------	---------	--------	---------	---------	---------

列 85 至 91

-0.0482	-0.2656	0.2197	0.4919	-0.6403	0.4264	0.9168
---------	---------	--------	--------	---------	--------	--------

列 92 至 98

-0.9399 -0.5305 -0.7712 -0.2134 0.0829 0.5602 -0.6038

列 99 至 105

-0.2765 -0.5155 -0.4046 -0.0197 -0.7788 -0.8706 -0.6088

列 106 至 112

-0.2848 0.8836 -0.9082 0.5037 0.4651 -0.1632 0.0348

列 113 至 119

0.1803 -0.3667 0.5943 0.6503 0.0060 -0.0656 -0.3874

列 120 至 126

0.6057 0.6985 0.4145 -1.1986 0.3700 -0.7747 -0.1256

列 127 至 133

0.9853 -0.5096 0.5224 0.7812 -0.8570 0.0323 -0.4082

列 134 至 140

-0.4193 0.3079 0.3469 -0.3728 0.2123 0.1388 -0.5966

隐含层的偏置 =

16.5626

-16.2957

16.1842

15.8524

-15.5654

15.4259

15.1362

14.9575

-14.6593

-14.4098

14.1719

13.9434

13.7260

13.4588
13.2325
-13.0182
12.7702
12.5180
12.1401
12.0365
-11.7623
-11.5571
11.4649
11.0790
10.8445
-10.6754
10.3714
10.1322
9.8843
9.6476
-9.4294
9.1689
8.9710
-8.6726
8.3594
-8.2180
7.9392
-7.7311
7.4939
-7.1077
6.9884
-6.6941
6.5887
6.3323
5.9882
5.8399
5.6467
5.2792
5.1539
-4.8382
-4.6020
-4.4649
-4.1779
-3.9326
3.6782
3.5300
-3.2194

2.9827
2.7447
-2.4962
2.2646
2.0115
-1.7717
1.5503
-1.3241
-1.0985
-0.8585
0.6530
-0.3621
0.1770
0.0988
-0.3771
0.7138
-0.6955
-1.0730
1.2748
-1.6454
-1.7853
1.9873
2.0923
-2.5029
2.6468
2.9411
3.2168
3.4617
-3.7037
-3.9293
4.0916
-4.4110
4.5835
4.9612
5.1398
5.3550
5.5071
-5.8168
-6.0797
-6.3055
-6.4843
-6.7817
7.0853
7.2842

7.5076
-7.6681
-8.0172
8.3117
8.4450
8.6257
8.9865
9.1677
9.3842
9.6492
-9.8935
-10.1274
-10.3742
-10.5667
-10.8398
-11.0964
-11.3214
-11.4988
-11.8616
12.0217
-12.2914
-12.5156
-12.7544
-12.9899
-13.2417
-13.4543
-13.6494
13.9902
-14.2610
14.5286
-14.6581
-14.8978
15.1407
15.3744
-15.6048
15.8694
16.0969
16.3262
-16.6653

输出层的偏置 =

-0.6377

t =

0.3670

训练完成以后的网络给定训练数据的输入得到的输出的误差的平方和 =

9.9459e-04

训练次数 iterations=260

所采用的算法：这里我们使用目前为止测试最好的算法——自适应学习速率法，其中需要调整的缺省值只有学习速率（0.001）。

2. 这里先用插值法给出测试数据，即给出验证的 A 与 T 值：

首先需要说明一下，关于这个插值法的要求，如果是我理解的插值法，我个人认为它与神经网络一样，都是一种泛化的方法，所以这个使用插值法来制作测试数据集的要求是让我很迷惑的，根据开头这个实验的目标，我原以为应该是另外选取 25 个符合该映射的数据计算误差平方和，可是再看要求觉得可能这个模糊控制器自变量的选取只能取整数，不是整数就没有这个映射了，所以这里是为了验证这个网络的泛化能力是否和插值法一样强。

内插值：

A: e=ec={-1.5,-0.5,0.5,1.5}

T={}

3. 值得注意的是即使所有的初始化参数都完全相同，也会出现 epoch 次数不同的情况，可能两千多次精度就已经在 0.0001 了，也有可能八千多次才达到，甚至超过了最大循环次数 10000 次都无法达到，这都是由于使用 newff 网络建立的时候权值和偏置的参数是随机选取的，如果靠近最小值点，训练速度就会比较快。
4. 心得体会：要有耐心，会算到手指都痛，网络上讨论的太少了，要自己一点点试，就比如那个插值函数，半天搞不定，可能是我能力不够。