Xihao Zhu xz36

HW03

P1.

(1)&(2)

By using parameters set in (1), where learning parameter is 0.001, slope parameter is 1, my training runs 1,000,000 steps and then converges with

```
w =

    1.0370   -1.2786
    2.2212    2.5459
   -2.1534   -2.5470

v =

    2.4976
   -2.8639
    2.7966
```

with RMS error <0.2

(3)

(a)By setting up my own parameters, I use 0.003 as learn parameter, 60 as slope parameter. Stopping criteria is "stop when 15,000 learning steps are performed".

Initial weights are:

```
w =

    0.0020000    0.0020000
    0.0070000    0.0020000
    0.0100000   -0.0010000

v =

    0.0050000
   -0.0040000
    0.0040000
```

(b)

Learn step	Input vec			Desired	Actual	ek
1000	1	0	0	-1	-0.80169	-0.19831
2000	1	0	0	-1	-0.88773	-0.11227
3000	1	1	1	-1	0.48802	-1.488
4000	1	0	1	1	0.825	0.175
5000	1	0	1	1	0.82421	0.17579
6000	1	1	1	-1	-0.82716	-0.17284
7000	1	1	0	1	0.90349	0.096513
8000	1	1	1	-1	-0.90506	-0.094942
9000	1	0	0	-1	-0.93872	-0.061276
10000	1	1	0	1	0.93619	0.063806
11000	1	1	1	-1	-0.92737	-0.072631
12000	1	1	1	-1	-0.93679	-0.063207
13000	1	1	1	-1	-0.93679	-0.063213
14000	1	0	1	1	0.95429	0.045708
15000	1	0	1	1	0.96056	0.039441

(c)

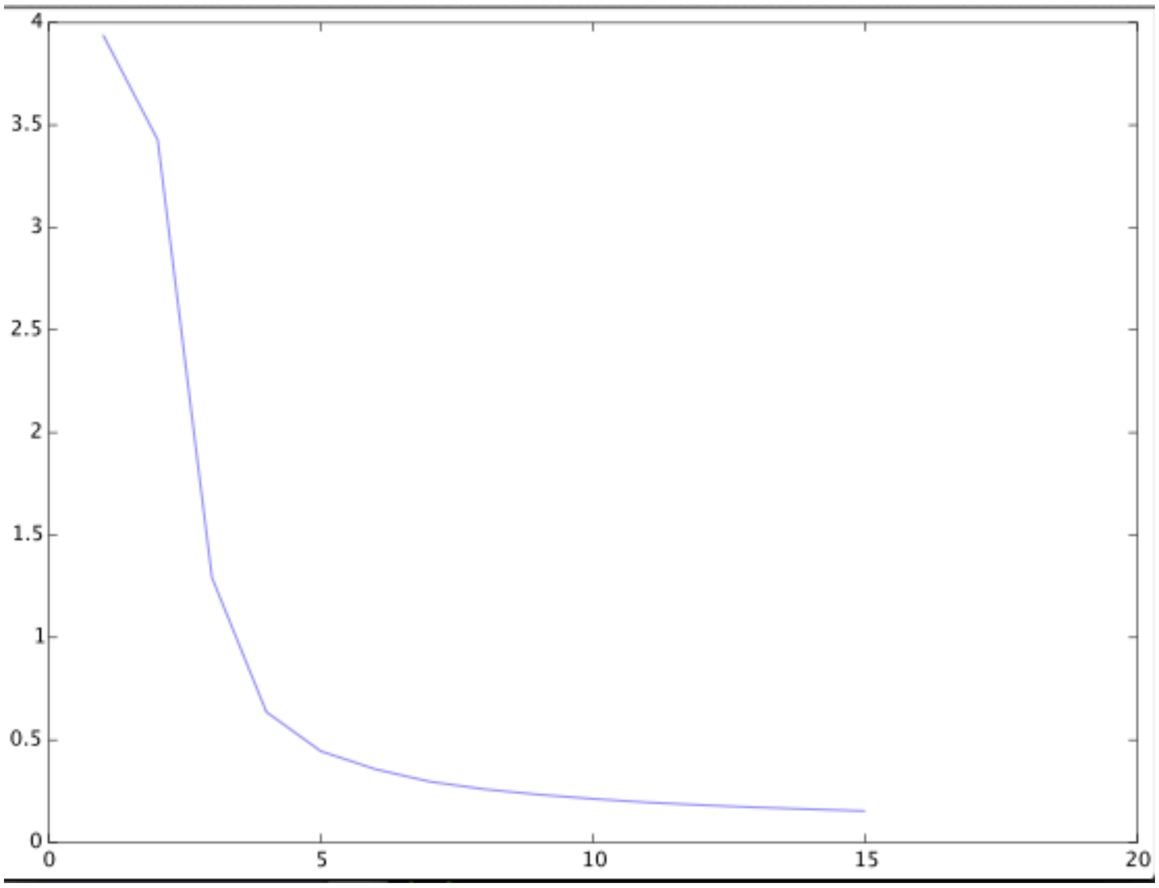
I use absolute error. The formula is

Error=abs(-1-y1)+abs(1-y2)+abs(1-y3)+abs(-1-y4) .

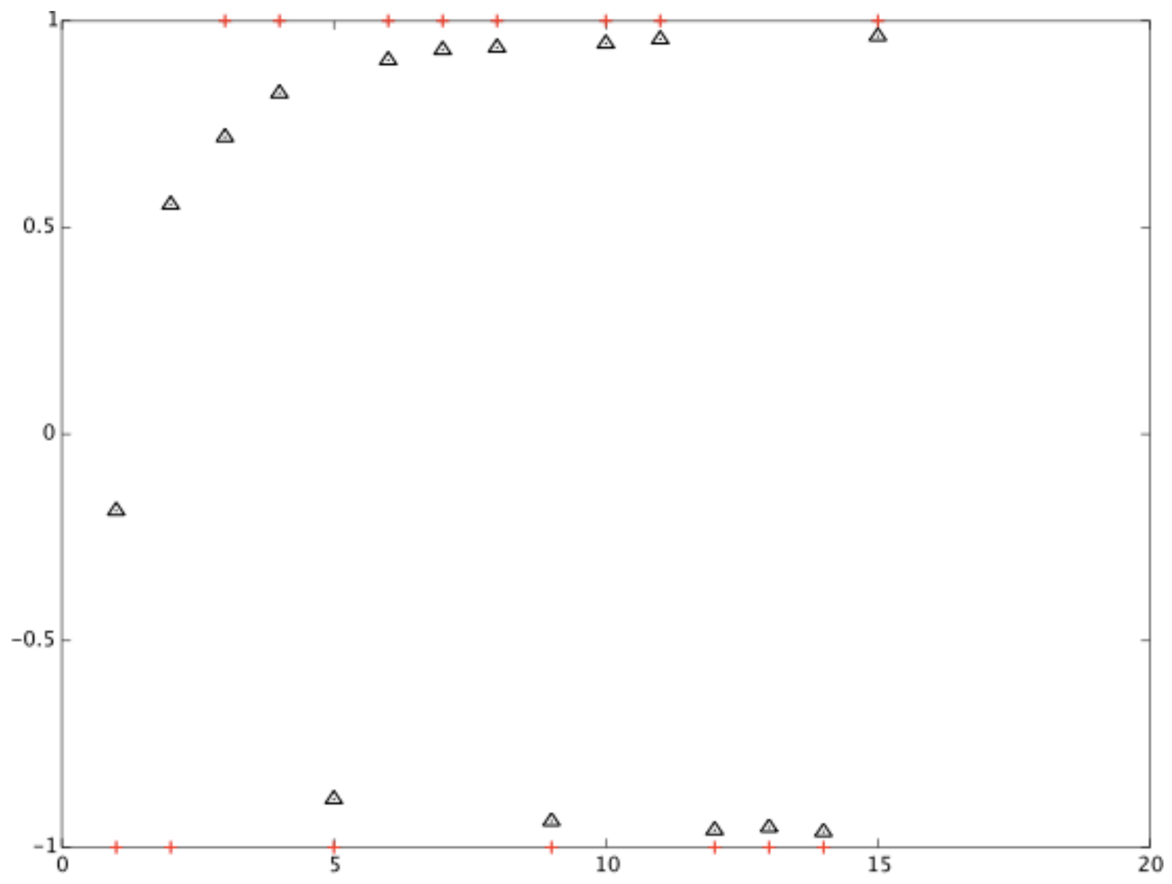
Where y1, y2, y3, y4 are outputs from neural network using current weights w, v, with inputs [1 1 1], [1 1 0], [1 0 1], [1 0 0].

My m is 15,

```
 err =

 Columns 1 through 8:

    3.93901     3.42967     1.29216     0.63663     0.44548     0.35740     0.29692
0.26010

 Columns 9 through 15:

    0.23363     0.21289     0.19567     0.18252     0.17139     0.16199     0.15371
```



This is a plot of Absolute error VS m step(m x1000 steps)



And this figure is a plot of desired outputs VS actual outputs. The red + is desired outputs; black triangle is actual output.

(d)
1,5000 learning steps total.

(e)
step.m is the related code.

P2.
(1)
changed code in step2.m
I use 7 hidden layers to train my data. I use the same parameters: 0.003 as learn parameter, 60 as slope parameter. I didn't use perfect match, so the initial weights are not same. They are just random weights.
I got resulting network's absolute error to be 0.12166. which is better than 2-hidden layer network, whose absolute error is 0. 15371(depicted in P1.3.c)

(2)
changed code in step2.m
call:
```
[w1,v1]=step2(3750,3,7)
r=errorRate(w1,v1)    where r is absolute error.
r=0.14
```

since previously step is 15000. Now epoch is 4, steps needed to run is 15000/4=3750. I found the error rate is slightly worse than 0.12166. When I use 4000 as step, that is, using
r is absolute error.
```
[w1,v1]=step2(4000,3,7)
r=errorRate(w1,v1)
r=0.115.
```
which is better than 0.12166. This is an interesting found~

(3)&(4)
It seems that the parameters provided don't converge for my code. I run 3000K steps but output doesn't converge.
(a)
By setting up my own parameters, I use 0.0001 as learn parameter, 0.001 as slope parameter. Stopping criteria is "stop when 150,000 learning steps are performed".
Initial weights are:

```
w =

  Columns 1 through 6:

   -0.040000     0.040000   -0.010000     0.060000     0.030000     0.070000
  -0.020000    -0.030000    -0.040000     0.090000    -0.080000     0.100000

 Columns 7 through 9:

   0.040000     0.030000     0.050000
   0.010000    -0.030000     0.030000
v =

   0.090000
  -0.020000
   0.080000
  -0.070000
   0.100000
   0.100000
   0.010000
   0.030000
  -0.070000
  -0.040000
```

(b)

Learn step	Input vec	Desired	Actual	ek
10000	1	1	8.0023	-7
20000	1	1	8.3239	-7.3
30000	1	1	8.8923	-7.9
40000	1	1	9.2323	-8.2
50000	1	1	9.9882	-8.9
60000	1	1	10	-9
70000	1	1	10	-9
80000	1	1	10	-9
90000	1	1	10	-9
100000	1	1	10	-9
110000	1	1	10	-9
120000	1	1	10	-9
130000	1	1	10	-9
140000	1	1	10	-9
150000	1	1	10	-9

(c)
the error becomes greater and greater as steps increase. And it converges to point where input pattern cannot influence output(output keeps at 10☹)
I think plot doesn’t make sense here.
(d)i use 150,000, and 3,000,000, neither works.

(e)step2.m