Xihao Zhu COMP502 HW5

P1.

(1)

(a)

For own parameters, I use 0.12 as learn parameter, 1 as slope parameter, 10 hidden layers, 0.2 as momentum term constant, 20 as epoch size. Stopping criteria is “stop when 150,000 learning steps are performed”.

Initial weights are w =

Columns 1 through 7:

0.40000 0.10000 0.50000 -0.40000 0.50000 -0.90000 -0.80000

-0.30000 0.90000 0.60000 0.80000 -0.70000 0.20000 -0.40000

Columns 8 and 9:

-0.90000 1.00000

-0.10000 -0.40000

v =

-0.050000

0.300000

-0.300000

0.700000

0.700000

0.600000

-0.400000

0.700000

-0.600000

0.800000

(b)

Learn step Input vec Desired Actual ek

10000 1 -0.43287 -0.471 -0.15428 -0.31672

20000 1 0.88708 -0.64456 -0.39194 -0.25262

30000 1 -0.82645 0.79887 0.72647 0.072402

40000 1 0.21186 -0.70298 -0.7276 0.024624

50000 1 0.012269 -0.9478 -0.79992 -0.14788

60000 1 0.020666 -0.94039 -0.80385 -0.13653

70000 1 -0.34321 -0.71574 -0.70555 -0.010198

80000 1 0.50266 -0.43696 -0.54967 0.11271

90000 1 0.71497 -0.49894 -0.47009 -0.028848

100000 1 0.77223 -0.54375 -0.51639 -0.027362

110000 1 0.043987 -0.91778 -0.85033 -0.067453

120000 1 0.16383 -0.7691 -0.75247 -0.016622

130000 1 -0.55097 -0.074666 -0.1077 0.033038

140000 1 -0.10421 -0.99997 -0.89701 -0.10296

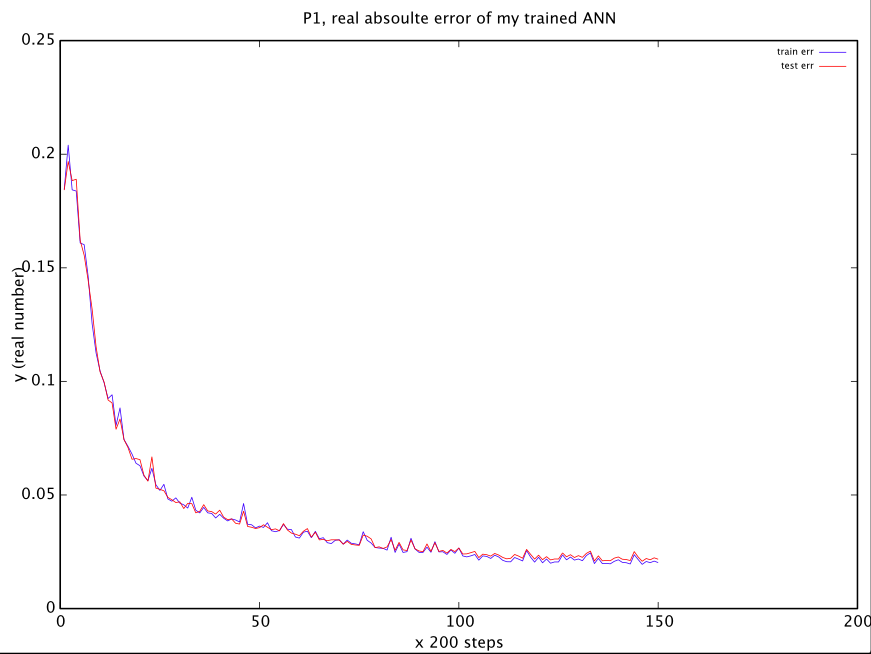
150000 1 -0.68276 0.38922 0.46882 -0.079593

(c)

The error measurement is like this:

I use absolute error. The formula is to get sum of absolute value of every network’s output minus desired output. Then scaling back the sum, so that the error is real error treating y as range of [-0.21, 1].

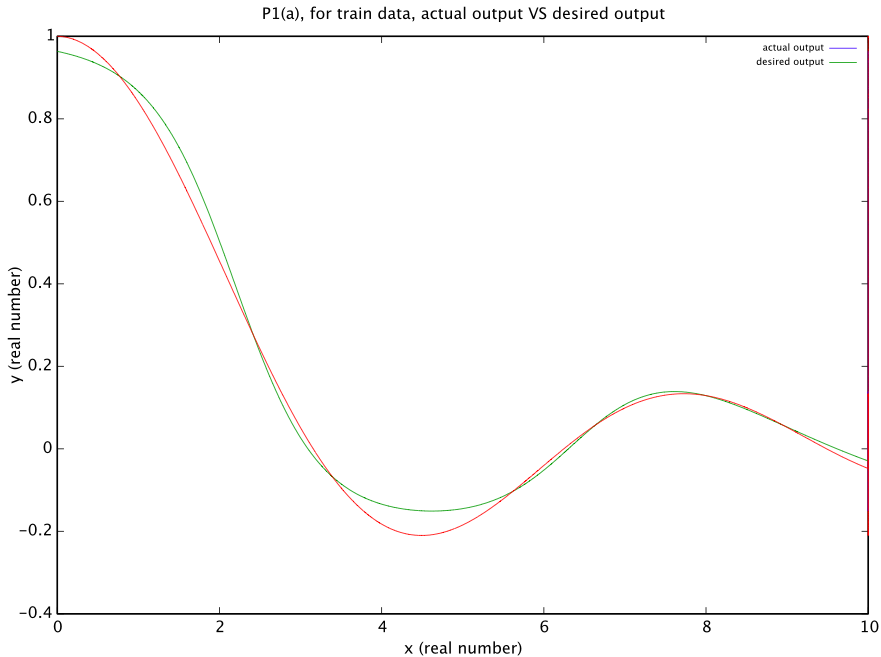
Here is the plot(y is absolute error)



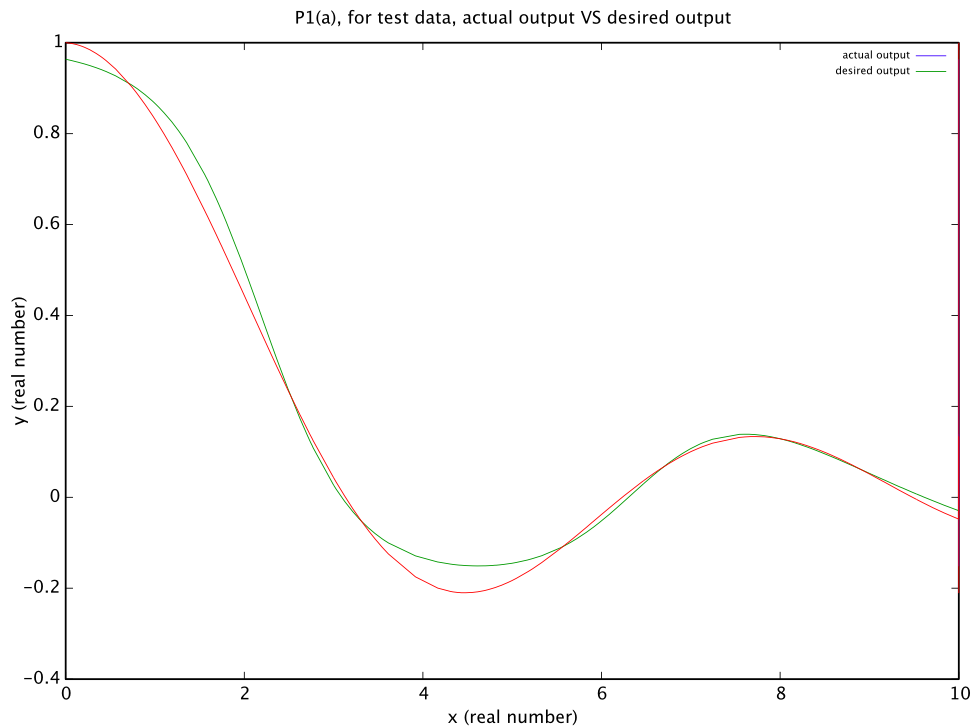
both the test error and train error are approaching 0.03. Therefore Error rate is about 0.03/(1-(-0.21))=2.48%

desired output VS calculated output is as follows(Matlab Color is wrong somehow, the red line is desired and green one is actual output)

For train data



For test data:



(2)

The code on owlspace doesn’t work for me.(I am using Octave..) So what I did was that I wrote my own cross validation function.

P2

(1)

successful parameter sets

(a)

For my parameters, I use 0.08 as learn parameter, 1 as slope parameter, 8 hidden layers, 0.15 as momentum term constant, 20 as epoch size. Stopping criteria is “stop when 70,000 learning steps are performed”. I run x from 1 to 40.

Initial weights are

w =

-2.40000 1.50000 0.30000 -0.30000 1.50000 1.20000 2.40000

0.30000 -1.80000 0.90000 -2.40000 2.70000 1.80000 1.80000

v =

2.40000

-2.40000

-0.30000

0.90000

0.30000

-0.30000

2.40000

1.80000

Yes, my initialized weights are big~

(b)

Learn step Input vec Desired Actual ek

6000 1 -0.052632 -0.30902 0.39015 -0.69917

12000 1 -0.15789 -0.80902 -0.48918 -0.31984

18000 1 0.052632 0.30902 0.39786 -0.088843

24000 1 -0.89474 0.80902 0.85527 -0.046252

30000 1 -0.26316 -1 -0.74219 -0.25781

36000 1 0.052632 0.30902 0.25875 0.050268

42000 1 -0.052632 -0.30902 -0.45773 0.14871

48000 1 -0.57895 0.30902 0.32889 -0.019874

54000 1 0.36842 0.80902 0.8231 -0.014081

60000 1 -0.26316 -1 -0.85871 -0.14129

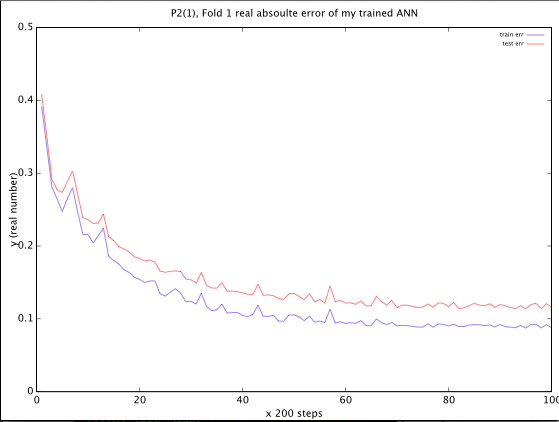
66000 1 -0.68421 0.80902 0.727 0.08202

(c)

The error measurement is like this:

I use absolute error. The formula is to get sum of absolute value of every network’s output minus desired output. Then scaling back the sum, so that the error is real error treating y as range of [-1, 1].

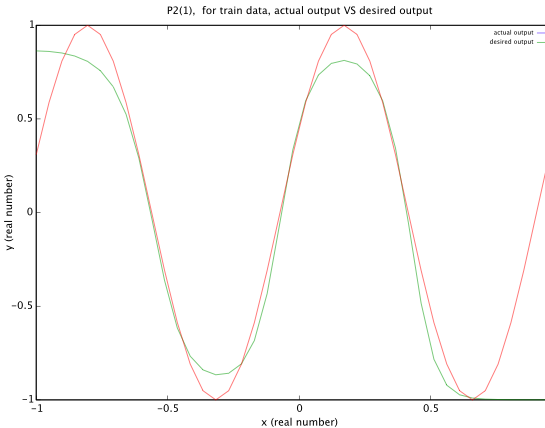
Here is the plot(y is absolute error)



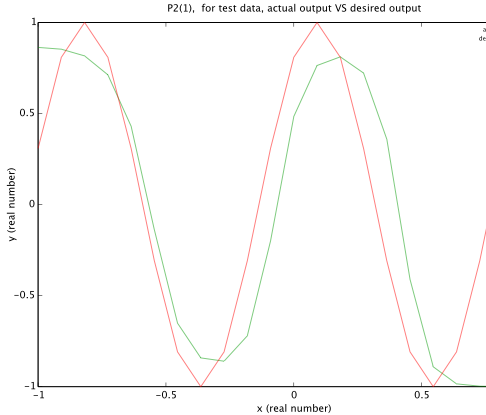
both the test error and train error are approaching 0.1. Therefore Error rate is about 0.1/(1-(-1))=5%

desired output VS calculated output is as follows(Matlab Color is wrong somehow, the red line is desired and green one is actual output)

For train data(x and y are unscaled here..)



For test data(x and y are unscaled here).



Unsuccessful parameter sets

(a)

For this set of parameters, I use 0.01 as learn parameter, 1 as slope parameter, 80 hidden layers, 0.1 as momentum term constant, 200 as epoch size. Stopping criteria is “stop when 70,000 learning steps are performed”. I run x from 1 to 40.

Initial weights are

w =

Columns 1 through 6:

0.200000 -0.900000 -0.400000 0.400000 0.500000 0.050000

0.800000 -0.900000 -0.500000 0.300000 -0.700000 -0.500000

Column 7:

0.600000

0.400000

v =

0.90000

-0.80000

0.50000

0.40000

-0.40000

-0.70000

0.70000

-0.60000

(b)

Learn step Input vec Desired Actual ek

6000 1 0.70732 -0.95106 -0.79142 -0.15963

12000 1 -0.70732 0.80902 0.44589 0.36313

18000 1 -0.41463 -0.80902 0.13622 -0.94523

24000 1 -0.41463 -0.80902 0.28218 -1.0912

30000 1 -0.36585 -0.95106 0.24687 -1.1979

36000 1 1 0.58779 -0.17167 0.75946

42000 1 -0.5122 -0.30902 0.3015 -0.61051

48000 1 -0.85366 0.95106 0.55124 0.39982

54000 1 -0.17073 -0.58779 0.025544 -0.61333

60000 1 -0.31707 -1 -0.092608 -0.90739

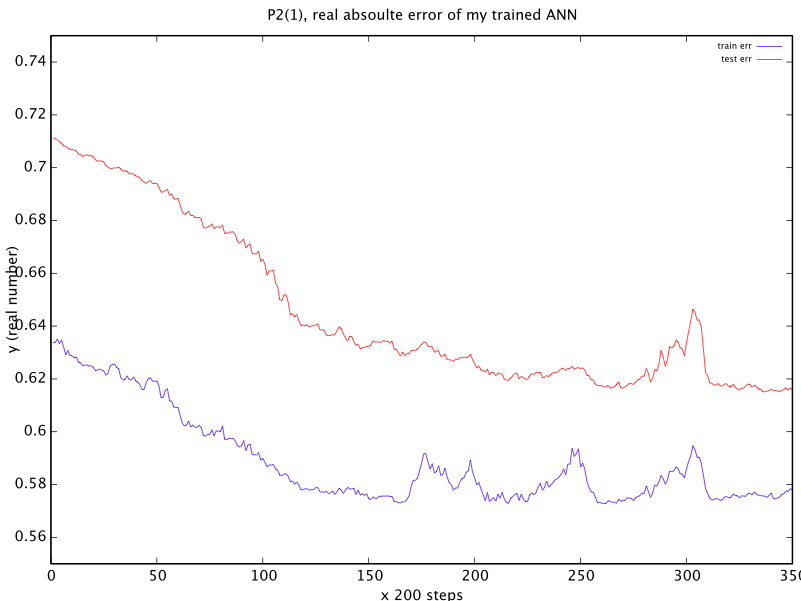
66000 1 0.90244 -4.4409e-16 -0.25086 0.25086

(c)

The error measurement is like this:

I use absolute error. The formula is to get sum of absolute value of every network’s output minus desired output. Then scaling back the sum, so that the error is real error treating y as range of [-1, 1].

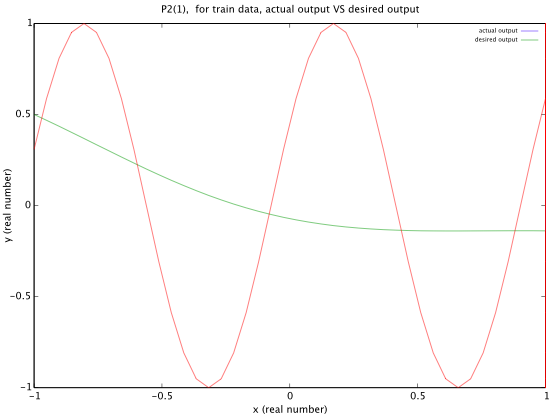
Here is the plot(y is absolute error)



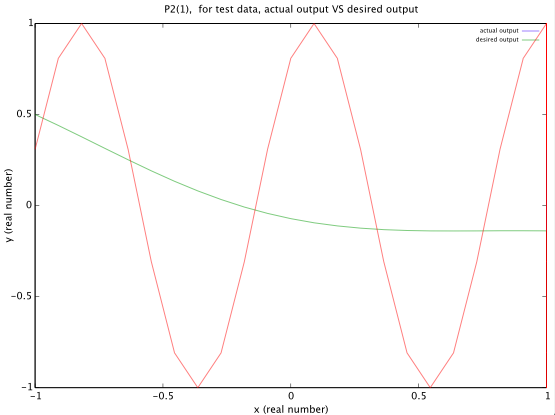
test error is nearly 0.64, train error is about 0.58. This result is really sad…

desired output VS calculated output is as follows(Matlab Color is wrong somehow, the red line is desired and green one is actual output)

For train data(x and y are unscaled here..)



for test data:

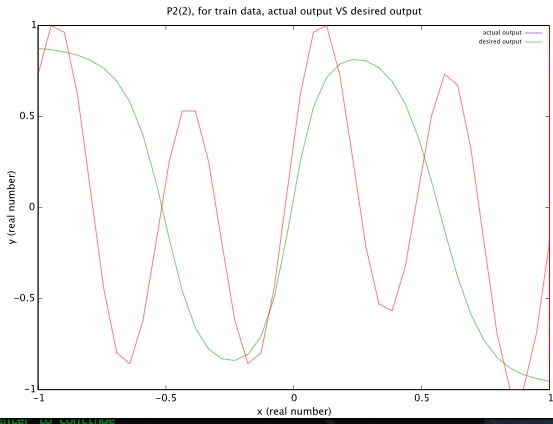


P2

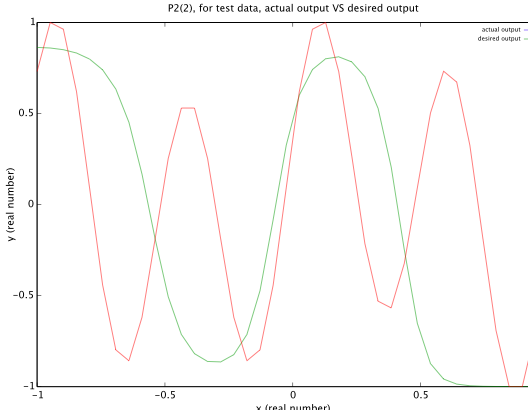
(2)

Unscaled actual output VS desired output is shown here

For train set:



for test data:



Seems that the green line(actual output) actually find out the sine sector(s(nT)) of the red line which represent the whole equation(Z(nT))

P3.

(a)

For my parameters, I use 0.05 as learn parameter, 1 as slope parameter, 10 hidden layers, 0.9 as momentum term constant, 200 as epoch size. Stopping criteria is “stop when 100,000 learning steps are performed”.

Initial weights are

w =

Columns 1 through 6:

0.080000 0.200000 -0.060000 -0.010000 0.160000 -0.160000

-0.100000 0.060000 -0.120000 0.040000 -0.140000 0.020000

0.200000 0.160000 -0.120000 -0.140000 0.020000 0.120000

0.140000 -0.020000 0.060000 0.160000 -0.020000 0.060000

0.100000 -0.140000 -0.060000 -0.040000 -0.100000 0.040000

Columns 7 through 9:

-0.120000 -0.040000 0.020000

0.180000 -0.080000 -0.010000

-0.040000 -0.060000 -0.060000

-0.140000 -0.060000 0.080000

0.140000 -0.060000 -0.020000

v =

0.080000 0.080000 0.200000

0.180000 0.020000 -0.080000

0.080000 0.020000 -0.100000

0.160000 -0.040000 0.160000

0.080000 0.200000 0.060000

0.140000 -0.140000 0.100000

-0.120000 -0.100000 -0.040000

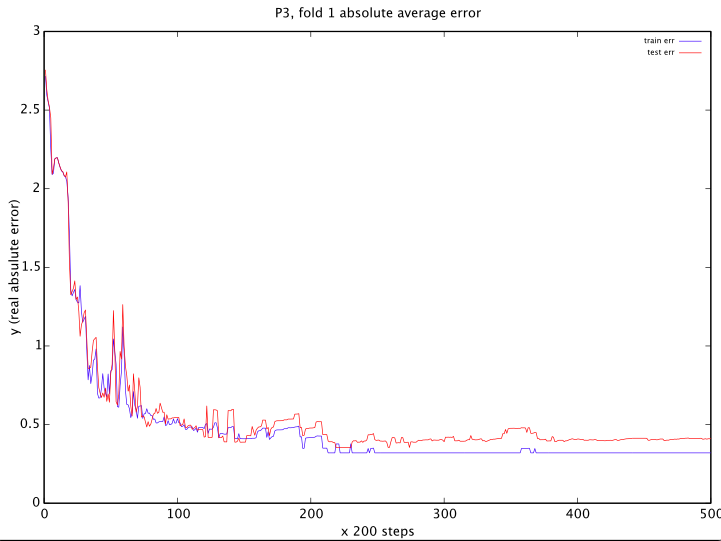
-0.140000 -0.180000 -0.020000

0.160000 -0.180000 -0.100000

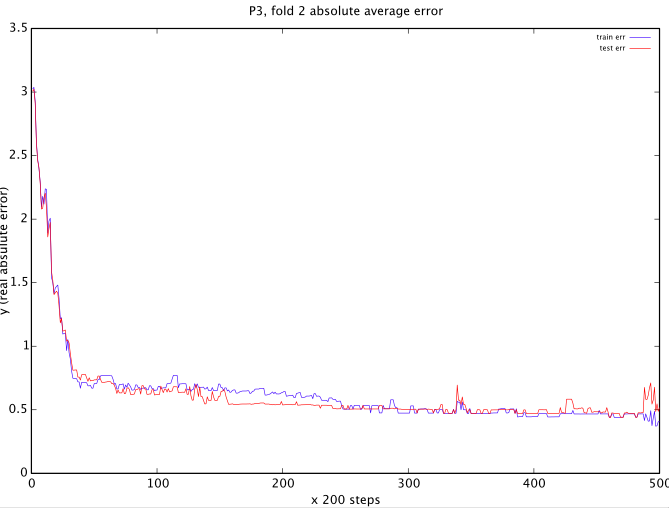
-0.040000 -0.100000 -0.120000

(b)&(c)

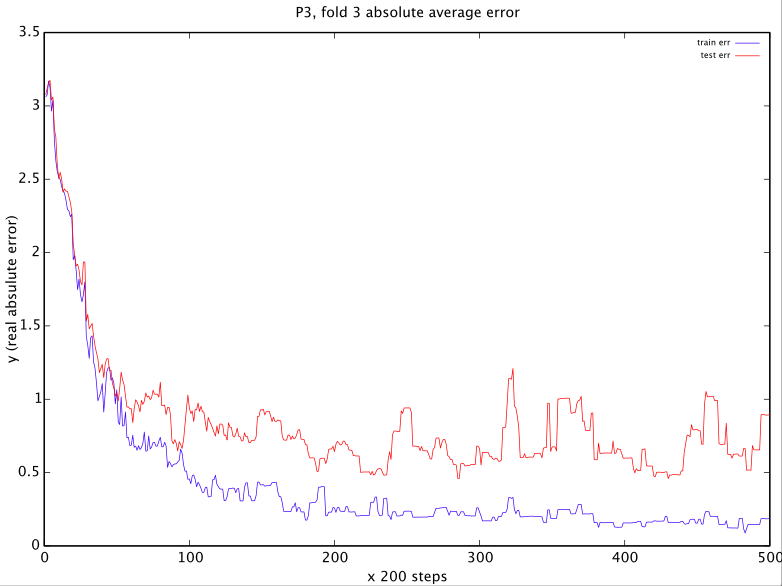
fold 1:



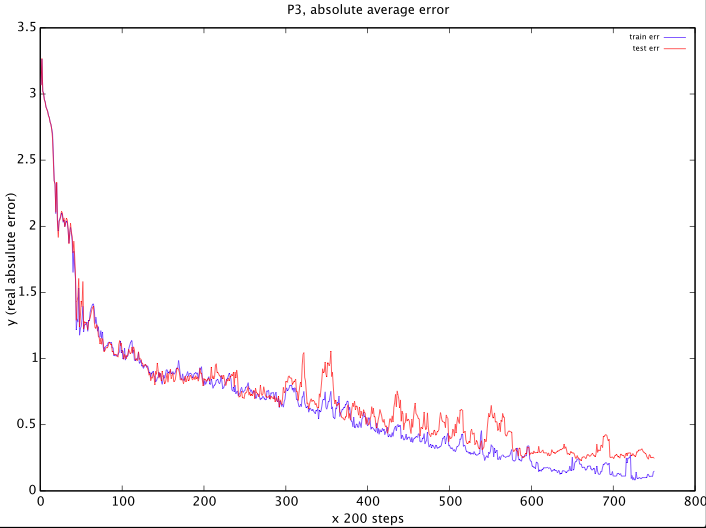
fold 2:



fold 3:



the following is using all test and train data:



Below I am plotting out Desired output VS actual output for 3 folds:

I forgot to add legends. + is desired output. Triangle is actual output.

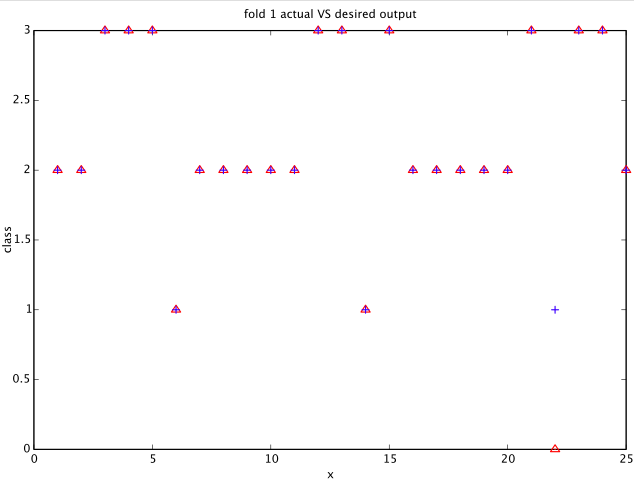
For y value,

1 is [1,0, 0] meaning Setosa

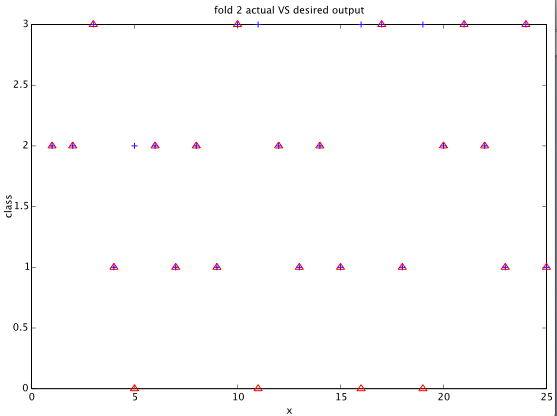
2 is [0 1 0] meaning Versacolor

3 is [0 0 1] meaning Virginica

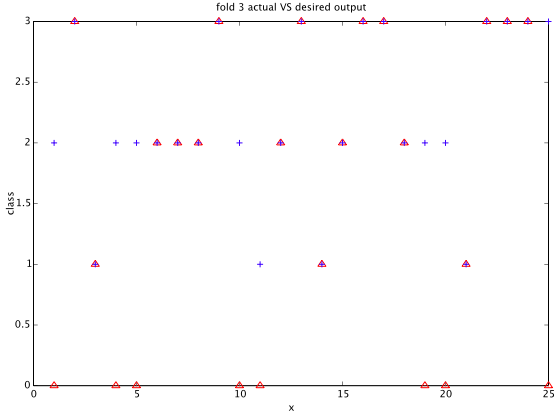
Fold 1:



fold 2:



fold3:



I think this plots are reasonable because fold 3 has high error rate, which we can tell also from its actual VS desired output~ Others are fine~