Brennan McFarland
10/02/18
Problem Set 2

Validating Computation

My analytic computation consistently matches or nearly matches the numerical approximation estimates returned by the numer_est_Wji and numer_est_Wkj functions, as evidenced by this sample of iterations' error estimates when running the fitting algorithm with an eta of .00001:

dWL_cum =

Columns 1 through 14

-1.3180 -1.6878 -1.1916 -1.6288
-0.8960 -1.3640 -1.2974 -1.4354 -1.3413
-1.1760 -0.7324 -0.7258 -1.5803 -0.6380
-5.2460 -6.7875 -4.7275 -6.6670
-3.6391 -5.5444 -5.2432 -5.6629 -5.4347
-4.8830 -3.0166 -2.9257 -6.5186 -2.7118

Columns 15 through 20

-0.7195 -1.3596 -1.5073 -1.5262
-0.6155 -0.9447
-2.9055 -5.4439 **-6.1261** -6.0529
-2.6510 -3.8806

est_dWkj =

Columns 1 through 14

-1.3180 -1.6878 -1.1916 -1.6288
-0.8960 -1.3640 -1.2974 -1.4354 -1.3413
-1.1760 -0.7324 -0.7258 -1.5803 -0.6380
-5.2460 -6.7875 -4.7275 -6.6670
-3.6391 -5.5444 -5.2432 -5.6629 -5.4347
-4.8830 -3.0166 -2.9257 -6.5186 -2.7118

Columns 15 through 20

-0.7195 -1.3596 -1.5073 -1.5262
-0.6155 -0.9447
-2.9056 -5.4439 **-6.1262** -6.0529
-2.6510 -3.8806

delta_L_cum =

-2.3113
-9.3758

delta_L_est =

-2.3113
-9.3759

dW_Lminus1_cum =

-0.4348 -0.6379
-0.1842 -0.2822
0.0721 0.1122
0.1340 0.1842

est_dWji =

-0.4348 -0.6379
-0.1842 -0.2822
0.0721 0.1122
0.1340 0.1842

0.3443	0.5027
0.2226	0.3296
-0.0422	-0.0670
0.2602	0.3697
0.1804	0.2691
-0.5525	-0.7933
0.0671	0.1073
0.2948	0.4342
-0.4102	-0.5838
0.1066	0.1680
0.3597	0.5302
0.1937	0.2800
0.3677	0.5240
0.3703	0.5384
-0.0016	0.0013
-0.2405	-0.3430

delta_Lminus1_cum =

-2.1812
-1.0194
0.4127
0.5737
1.7092
1.1407
-0.2522
1.2099
0.9405
-2.6332
0.4084
1.4951
-1.9146
0.6285
1.8211
0.9387
1.7190
1.8206
0.0214
-1.1282

0.3443	0.5027
0.2226	0.3296
-0.0422	-0.0670
0.2602	0.3697
0.1804	0.2691
-0.5525	-0.7933
0.0671	0.1073
0.2948	0.4342
-0.4102	-0.5838
0.1066	0.1680
0.3597	0.5302
0.1937	0.2800
0.3677	0.5240
0.3703	0.5384
-0.0016	0.0013
-0.2405	-0.3430

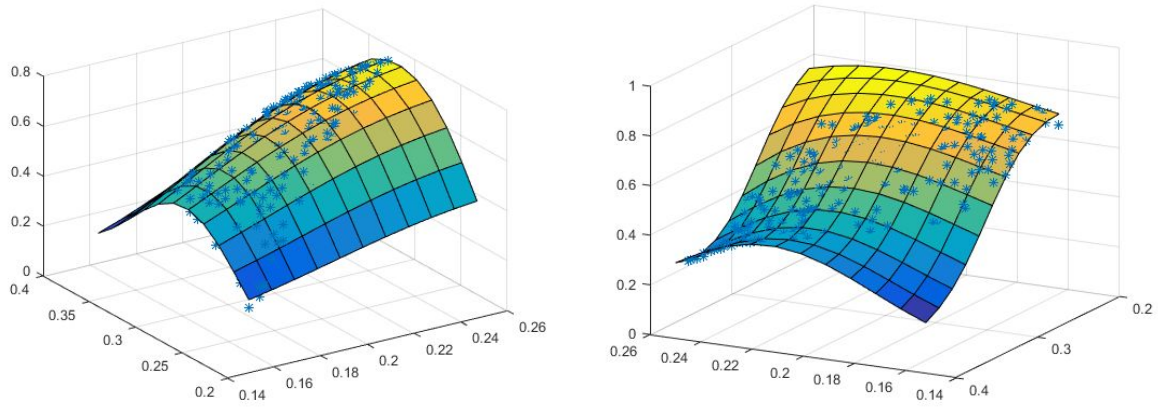
delta_Lminus1_est =

-2.1812
-1.0194
0.4127
0.5737
1.7092
1.1407
-0.2522
1.2099
0.9405
-2.6332
0.4084
1.4951
-1.9146
0.6285
1.8211
0.9387
1.7190
1.8206
0.0214
-1.1282

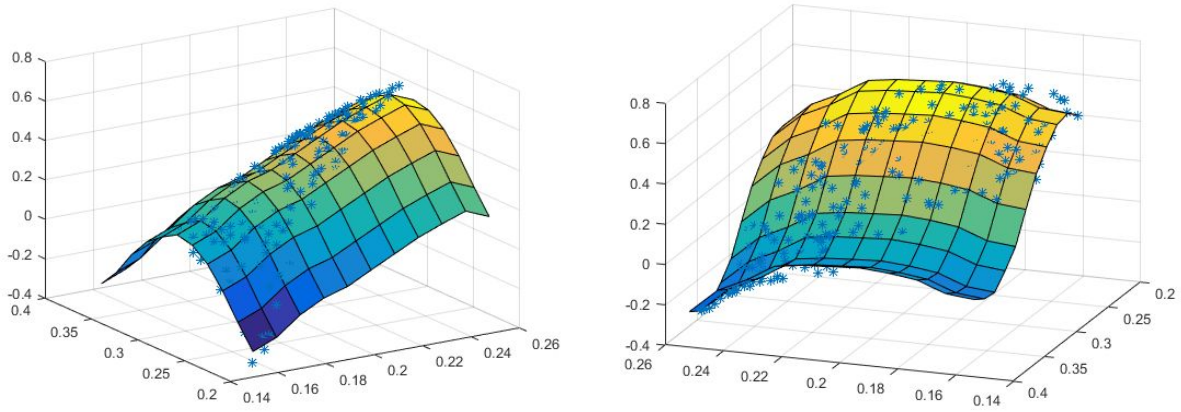
Discrepancies between the estimated and calculated values are highlighted in bold, but it can be seen that even in these cases the difference is only approximately .0001. In other cases the difference is small enough that it is not apparent in the number of significant digits displayed.

Comparing Hidden Layer Count

Running the ps2_fdfwd_net algorithm for approximately 8 hours (6336330 iterations) yielded the following graphs for the x and y hand positions (esqd=0.0542), demonstrating visually the effectiveness of the calculations:



And running the ps2_fdfwd_net algorithm for 1000 iterations (which only takes a couple of minutes, esqd= 0.0295, 0.2277) to fit the x and y hand positions yields:



It is therefore immediately apparent that the 3 layer network with partially preset weights and biases is far superior in terms of efficiency compared to the 1 layer network.

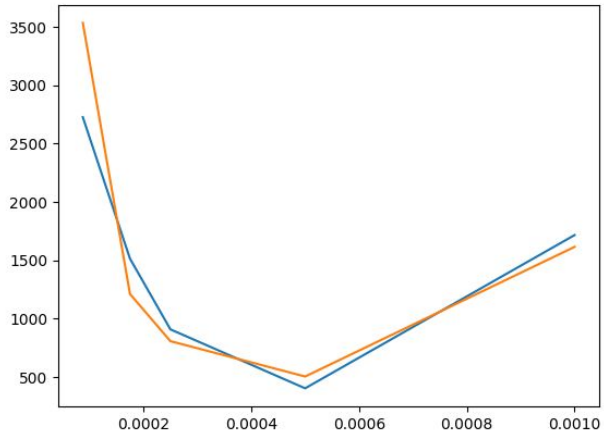
NOTE: As running the algorithm with the 1 hidden layer network takes the greater part of a day to converge, the following experiments will use only the network with 3 layers.

Changing the Learning Factor

The results above were obtained with an initial learning rate of eta=.001 and increasing or decreasing it dynamically. Setting the network to train until rms error drops below .05 and varying static values of eta yields the following results:

eta	Iterations for x	Iterations for y
.001	1717	1616
.0005	404	505
.00025	909	808
.000175	1515	1212
.0000875	2727	3535

Note that when eta was increased to above .001 the algorithm simply oscillated around the solution and ran for an indefinite amount of time. The solution graphs hardly changed in these cases, so increasing the error on convergence threshold within reason would not help. The iterations for x and y to convergence over varying values of eta is plotted as follows:



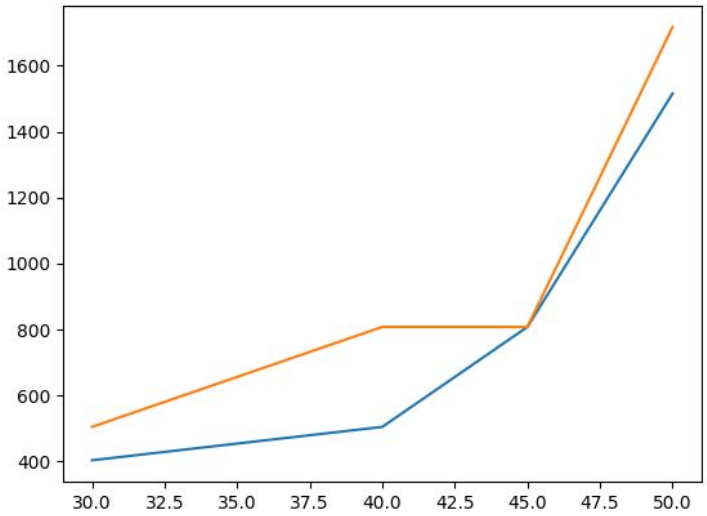
iterations for x iterations for y

As can be seen in the graph, for both dimensions the number of iterations to convergence first decreases as the algorithm takes larger steps then increases as it begins to overshoot the minimum error.

Accuracy Achieved in Training

The speed of convergence for the network can also be measured against varying values for the number of interneurons in the non-premapped layer (eta=.001):

nnodes_layer1	Iterations for x	Iterations for y
30	404	505
40	505	808
45	808	808
50	1515	1717



iterations for x iterations for y

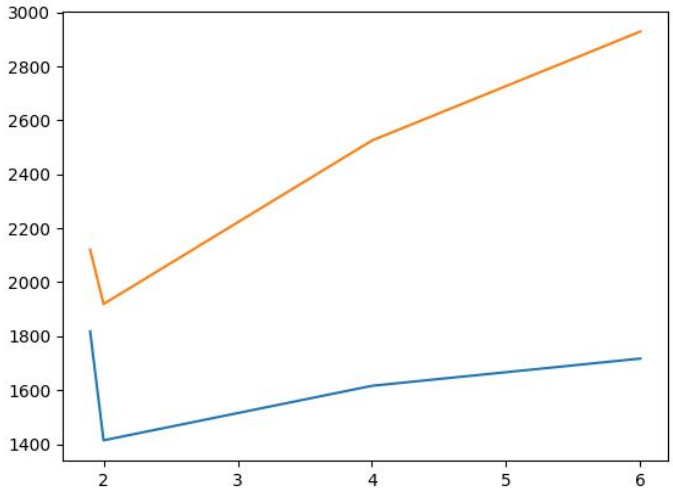
As can be seen, it appears that decreasing the number of nodes causes the algorithm to converge faster to within the margin of error, however it should be noted that this performance gain reaches a tangent and is as a tradeoff less capable of representing the target function more accurately given more time to run.

Alternative Values for the Fixed Weights

The network's efficiency can also be measured against changing the amplitude of the preset weights, ie, by varying the w_scale parameter. Datapoints and the corresponding graph are shown below:

W_scale (/dx)	Iterations for x	Iterations for y
1.9	1818	2121
2	1414	1919
4	1616	2525

6	1717	2929



iterations for x iterations for y

It is important to note that below w_scale=1.9 the algorithm almost immediately jumps to running indefinitely. This is likely because setting the scale below 2 causes all weights to quickly drop towards 0 (see Figure A., below) thus rendering most of the network silent. It can be seen though from the plot that after a sharp decrease in training time due to this effect, increasing the weights increases the number of iterations as the preset layers as the initial layers gradually tend towards always being active (see Figure B.). Either extreme allows the network to carry less information, so the optimal weight preset scale is at a happy medium, around 2.

Additional Observations

Dynamically updating the learning rate as implemented in the original code appears to give the best results, taking only 303 and 505 iterations for the x and y positions given eta=.001 and 50 interneurons in the non-preset layer. This is to be expected as the algorithm can dynamically adjust to the local area of the solution space and take smaller steps as it approaches the minimum.

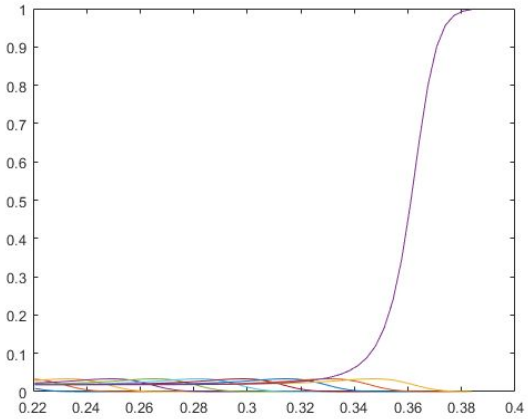


Figure A. w_scale=1

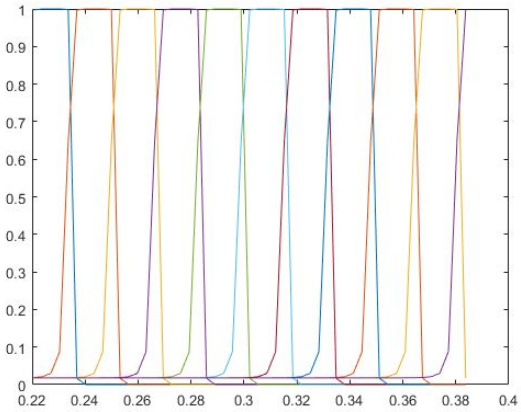


Figure B. w_scale=6