**INTRODUCTION TO ARTIFICIAL INTELLIGENCE**

NESTOR ROMERO LEON

301133331

**NEURAL NETWORK ASSIGNMENT**

**Exercise 1 (10 data points, epochs=200, show=15, goal=0.00001)**

Epoch: 15; Error: 0.055049850446098066;

Epoch: 30; Error: 0.024593664619289447;

Epoch: 45; Error: 0.006163642879966952;

Epoch: 60; Error: 0.0032412563701455376;

Epoch: 75; Error: 0.0018684930407908416;

Epoch: 90; Error: 0.0017094236408314051;

Epoch: 105; Error: 0.0016000205568634002;

Epoch: 120; Error: 0.0015388619950171955;

Epoch: 135; Error: 0.0015049017675386417;

Epoch: 150; Error: 0.0014823917492332033;

Epoch: 165; Error: 0.0014633140112977022;

Epoch: 180; Error: 0.001451172856561653;

Epoch: 195; Error: 0.0014411816413149685;

The maximum number of train epochs is reached

**Predicted result for (0.1,0.2) neural network NN1. Result 1 = [[0.23498515]]**

**Exercise 2 (10 data points, epochs=1000, show=100, goal=0.00001)**

Epoch: 100; Error: 0.8599746586220043;

Epoch: 200; Error: 0.8246044813592557;

Epoch: 300; Error: 0.786599411313424;

Epoch: 400; Error: 0.6369767153777499;

Epoch: 500; Error: 0.32361404634162033;

Epoch: 600; Error: 0.21833384174272943;

Epoch: 700; Error: 0.1969065531949613;

Epoch: 800; Error: 0.18825597981276754;

Epoch: 900; Error: 0.18123570604869788;

Epoch: 1000; Error: 0.1730628747816391;

The maximum number of train epochs is reached

**Predicted result for (0.1,0.2) neural network NN2. Result 2 = [[-0.06966066]]**

When using a comparable set of parameters for training both neural networks, the results suggest that the **Gradient Descent with Momentum Backpropagation and Adaptive LR** training method used by default by the newff function (<https://pythonhosted.org/neurolab/intro.html>) allows better predictions achieving a lower error rate in 2/3 the number of epochs.

Neural Network 1 (NN1) starts with a considerably lower error rate and converges faster than Neural Network 2 (NN2). With the results, it is clear that N1 showcases a better example of the expected gradient descent behavior displaying a higher delta in error decrease for earlier iterations and converging to the final solution even since epoch 150. NN2 at the same epoch displayed poor performance with a higher error rate and lower delta in error decrease.

For NN2 it seems to be struggling to minimize the loss function with smaller steps and finalizing with a quite imprecise test/simulation value.

**Exercise 3 (100 data points, epochs=200, show=100, goal=0.00001)**

Epoch: 15; Error: 0.1419487203078696;

Epoch: 30; Error: 0.051418466145696035;

Epoch: 45; Error: 0.030727470794240368;

Epoch: 60; Error: 0.023976229310110224;

Epoch: 75; Error: 0.01813915368617672;

Epoch: 90; Error: 0.01757322557754569;

Epoch: 105; Error: 0.016290877951351066;

Epoch: 120; Error: 0.015180015885127678;

Epoch: 135; Error: 0.01497747415573295;

Epoch: 150; Error: 0.014868869611323082;

Epoch: 165; Error: 0.01436993648814377;

Epoch: 180; Error: 0.01431052133964017;

Epoch: 195; Error: 0.013977247302605675;

The maximum number of train epochs is reached

**Predicted result for (0.1,0.2) neural network NN3. Result 3 = [[0.29833818]]**

With the increase of data points, as one might expect, the resulting effect for NN3 is a lower error rate and a much better test/simulation value. The rest of the evolution of error rates is fairly similar to NN1.

**Exercise 4 (100 data points, epochs=1000, show=100, goal=0.00001)**

Epoch: 100; Error: 1.7480473244144719;

Epoch: 200; Error: 1.3445013185864663;

Epoch: 300; Error: 1.0756894254964537;

Epoch: 400; Error: 0.8585611585147372;

Epoch: 500; Error: 0.660734345009445;

Epoch: 600; Error: 0.47715715935020114;

Epoch: 700; Error: 0.31992164133339757;

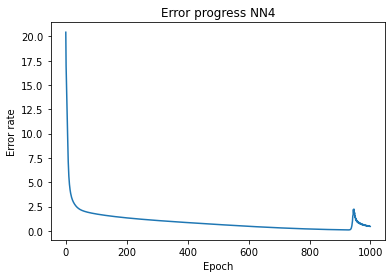
Epoch: 800; Error: 0.19910187309724708;

Epoch: 900; Error: 0.1186071074069235;

Epoch: 1000; Error: 0.46193908103292153;

The maximum number of train epochs is reached

**Predicted result for (0.1,0.2) neural network NN4. Result 4 = [[0.38923075]]**



For NN4 the test/simulation value is worse than NN3, even the error rate is several orders of magnitude higher than for NN3. In this case, the Gradient descent backpropagation training method seems to continue struggling and even missing the target of a local minimum when adjusting its weights (~Epoch 900).

**Exercise 5**

Epoch: 15; Error: 0.03355205351442765;

Epoch: 30; Error: 0.014393345631166812;

Epoch: 45; Error: 0.011974963908531134;

Epoch: 60; Error: 0.011873243513511034;

Epoch: 75; Error: 0.01176773742676583;

Epoch: 90; Error: 0.011587528662221434;

Epoch: 105; Error: 0.01133232479457341;

Epoch: 120; Error: 0.011293170859125151;

**Predicted result for (0.2,0.1,0.2) neural network NN5. Result 5 = [[0.81736183]]**

Epoch: 100; Error: 1.5339018050355262;

Epoch: 200; Error: 1.3969246168294747;

Epoch: 300; Error: 1.3382229311770186;

Epoch: 400; Error: 1.2992231666157448;

Epoch: 500; Error: 1.2694964449810013;

Epoch: 600; Error: 1.2447851126967258;

Epoch: 700; Error: 1.2228878568003343;

Epoch: 800; Error: 1.202573756394258;

Epoch: 900; Error: 1.1832047392569265;

Epoch: 1000; Error: 1.1646445694374254;

The maximum number of train epochs is reached

**Predicted result for (0.2,0.1,0.2) neural network NN6. Result 6 = [[0.60614152]]**

Even with more inputs, result 5 shows better behavior in terms of minimizing error, however, the test/simulation value is way off target. Result 6 seems closer but the higher error rates suggest an abnormal even lucky prediction possibly because of the particular set of data in some form of “overfitting” if even possible to call it that.