CS557 - Project Two

Andrew Struthers, Austin Laverdure, Bridget Smith

May 2023

What To Do:

Investigate the use of the Multi-Layer Perceptron (MLP, Sections 4.2.1 and 4.4.1 in textbook) to approximate (a regression problem) the following function:

$$f(x, y) = \sin(\pi * 10 * x + \frac{10}{1 + y^2}) + \ln(x^2 + y^2)$$

where $1 \leq x \leq 100$ and $1 \leq y \leq 100$ are real numbers.

1. Set up two sets of data, one for training and one for testing.
2. Use the training set to compute the weights of the network.
3. Evaluate the approximation accuracy (the root-mean squared error) of the network on the test data.
4. Use a variable number of hidden layers (1-2), each with a variable number of neurons (0-19).
5. Investigate how the network's performance is affected by varying the number of hidden layers and the number of hidden neurons in each layer. It is not necessary to try all 20 x 20 combinations of neurons in the two hidden layers; try only some of them.
6. Investigate how the network's performance is affected by varying the number of training patterns.

1 Implementation and Results

1. Describe which implementation you used and, if it is your code, attach it

    - We used the Multi-Layer Perceptron regressor provided by the Scikit-Learn package. We also used the Numpy package to assist with the math and MatPlotLib for graphing some of the results. In the test cases reported, we used $10,000$ training cases with $1,000$ test cases. We ran the code in a nested for loop that iterated over every given combination of hidden layers in the range of $[1, 2]$ and neurons per layer in the range of $[0, 19]$, printing out the root mean squared error of the test data each time. We used the logistic sigmoid function for the activation function, with a constant learning rate of $n = 0.001$. The weight optimization solver we used was *adam*, which is a stochastic gradient-based optimizer. This provided us the best results compared with the other solvers provided by Scikit-Learn. We also ran the regressor with a fixed random number seed so that we could get consistent results during the training phase in each iteration. The code used for this lab has been submitted in addition to this lab report.

2. The approximation accuracy for the different MLP architectures used

Table 1: Root-mean squared errors for different networks in one trial

# Neurons	1 Layer	2 Layers
1	0.71987	0.72790
2	0.70839	0.72012
3	0.71174	0.71917
4	0.70611	0.69723
5	0.70794	0.69194
6	0.70595	0.69205
7	0.70695	0.69208
8	0.70566	0.69391
9	0.70767	0.69538
10	0.70435	0.69169
11	0.70729	0.69178
12	0.69501	0.69322
13	0.71808	0.69293
14	0.70517	0.69328
15	0.69843	0.69260
16	0.70514	0.69635
17	0.70473	0.69506
18	0.69381	0.69157
19	0.70892	0.69420

Table 2: Count of lowest root-mean squared errors by network for 15 trials (only networks with lowest RMSE during any trials are shown)

Network	# of trials with smallest root-mean squared error
2 layers with 7 neurons per layer	1
2 layers with 9 neurons per layer	1
2 layers with 10 neurons per layer	3
2 layers with 15 neurons per layer	1
2 layers with 16 neurons per layer	2
2 layers with 17 neurons per layer	2
2 layers with 18 neurons per layer	2
2 layers with 19 neurons per layer	3

The network with the smallest root-mean squared error always had two hidden layers but the number of neurons changed; although those with more neurons per layer tended to outperform those with less. Only in 2 out of 15 trials did a network with less than 10 layers have the smallest root-mean squared error. Two layers with one neuron per layer performed with the lowest accuracy almost every test.

The test that gave us these outputs was a run of the code with $10,000$ training cases and $1,000$ test cases. In the figure below, we can see the error of the best approximation. In this case, it turned out the best approximation was achieved with 2 layers and 18 neurons per layer, and we got an RMSE value of 0.69157. We graphed a scatter plot of

$$|\text{calculated output} - \text{predicted output}|$$

for each of the $1,000$ test cases and then a horizontal line at the RMSE value.

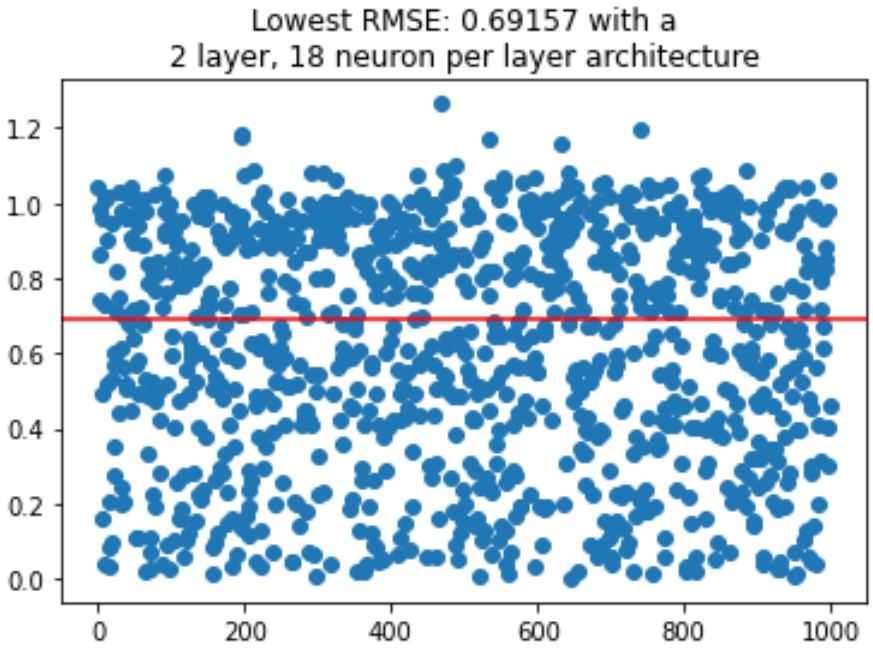

Figure 1: Error of approximation to calculated values with RMSE graphed

3. Your conclusions about how the number of hidden layers and neurons influences the approximation error

    - The conclusions we were able to gather from running all of these trials is that as the number of layers increase, and as the number of neurons in each layer increase, the function approximator becomes more and more accurate. However, there is a bit of diminishing returns as we increase the number of layers and neurons drastically. As we add more layers and neurons, the runtime and iterations required to converge during the training phase increases, without necessarily giving us better estimation accuracy. We can show these conclusions with the following two figures:

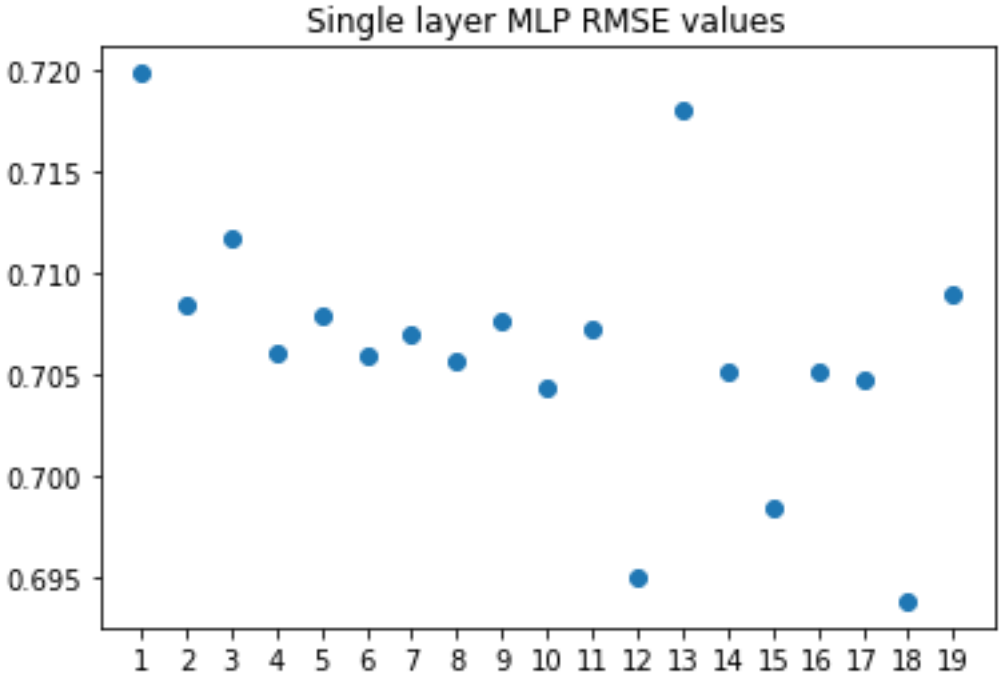

Figure 2: RMSE values for prediction after training with 1 layer and x neurons

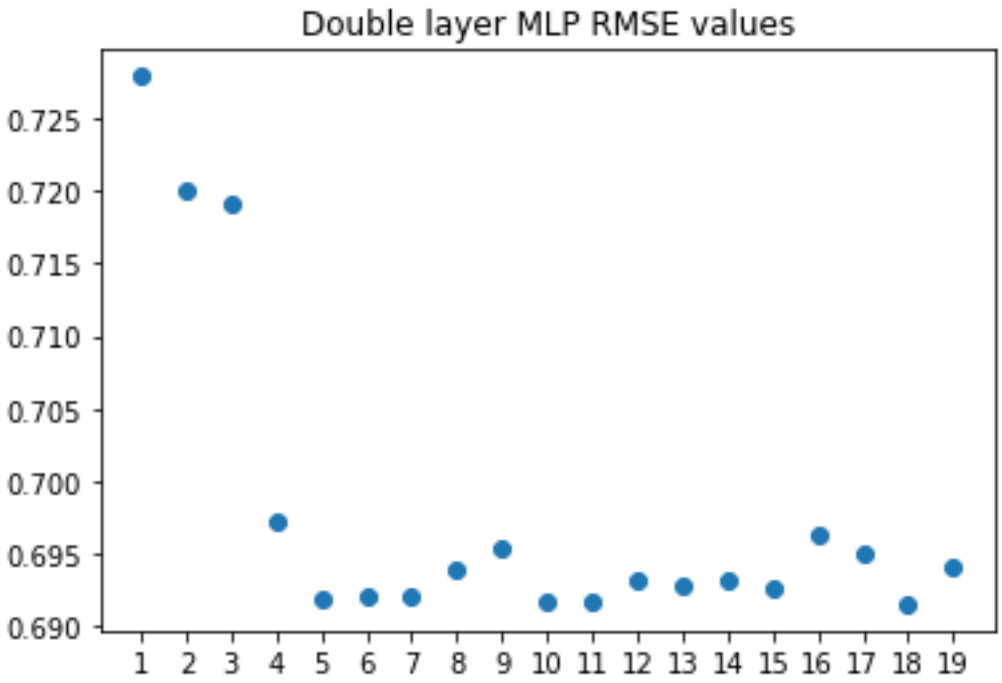

Figure 3: RMSE values for prediction after training with 2 layer and x neurons

We can see that, in both cases, as we add more neurons per layer, the function approximation has a lower error. In the case of the double layer architecture, however, we can see that after 4 neurons, the RMSE value starts to stagnate. The training phase takes longer the more neurons we have in each layer, and since we don't see any better approximation after a 2 layer, 5 neuron architecture, we could target MLP architectures around those levels.