David D’Attile
Prof. Kauchak
CS158
3 April 2022

# Assignment 08: Two-Layer Neural Network

**1.** *I gave you an example network and showed what would happen if you trained the network on one example. Starting at the original network (Figure 1) using η = 0.5, calculate the following using your code.*

**1a.** *What are the node outputs if we input an example [0.5, 0.2] with the label −1? You should have three values, two for the hidden node outputs and one for the output node:*

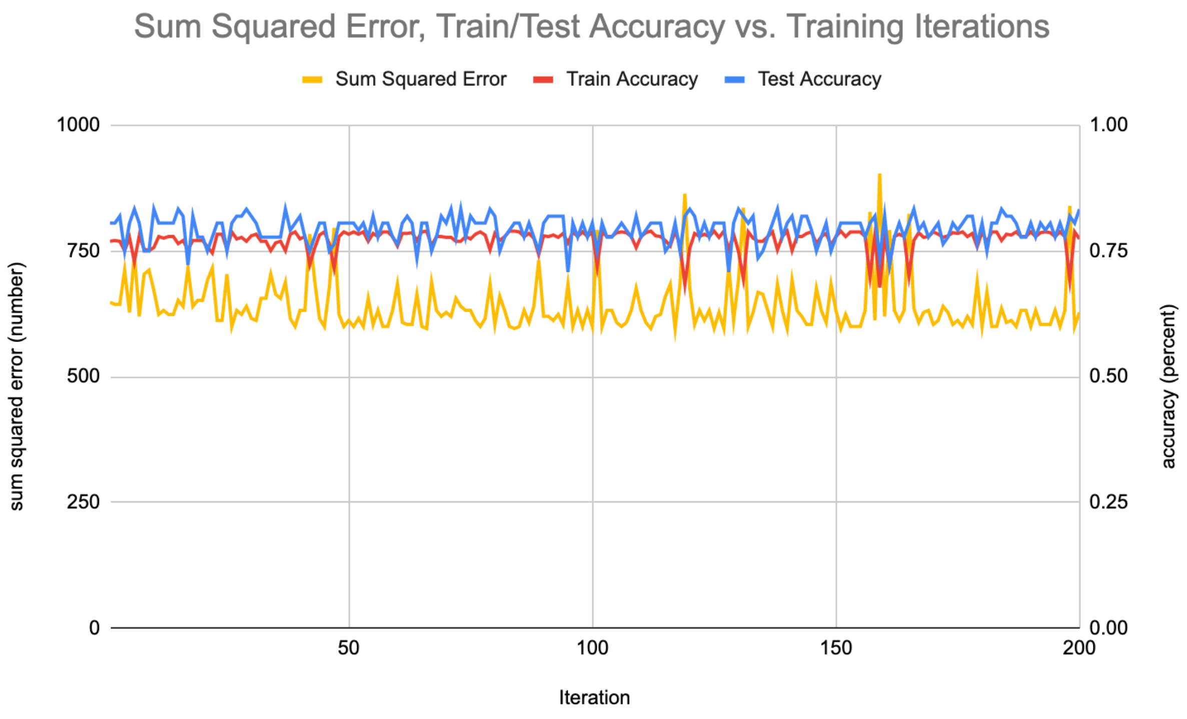
- Hidden Node 1: -0.949826
- Hidden Node 2: -0.852437
- Output Node: 0.997571809785583

**1b.** *What are the final weights if we train for one iteration on the example [0.5, 0.2] with the label −1? Write your input weights as a matrix, with the first row, the weights into the first hidden node, the second row the weights into the second hidden row, etc:*

Hidden Weights:
[ x1, x2, bias ]
[ -0.699739, 1.600104, -1.799479 ]
[ 0.030397, 0.600159, -1.399205 ]

Output weights:
[ v1, v2, bias ]
[ -1.095398, -0.595870, 1.795155 ]

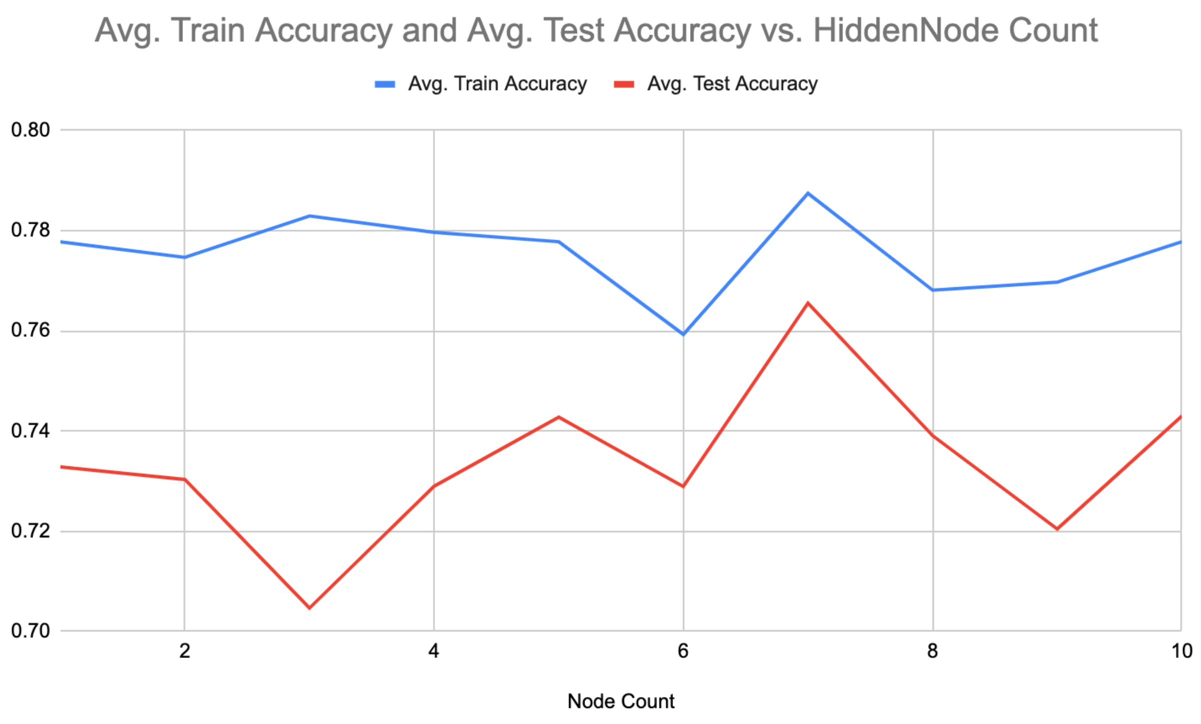
**2.** *Create a plot that has the number of training iterations on the x-axis and then plots three things: 1) the sum of the squared errors for the iteration, 2) the training accuracy, and 3) the testing accuracy. Use the default parameters and three hidden nodes. Do this for one random 90/10 split of the data (i.e. generate the values as the model is training). What we want to visualize is how the loss correlates with the other two. Write a few sentences describing this data. Anything interesting? Do you see signs of overfitting?*



Based on the data above, I noticed a few interesting things. First, from the relatively flat train/test accuracies and sum squared error values, it seems like the network is able to learn the binary titanic dataset fairly easily (after an iteration or two). Because of this, it's tricky to tell how loss correlates with the other two values. Interestingly, I did notice that in this example, the test data classification accuracy was consistently higher than the training data accuracy, which does not indicate overfitting.

**3.** *Experiment using 10-fold cross validation on how the number of hidden nodes impacts the performance. Create a table with values from 1 to 10 hidden nodes and include training and test averages with the default parameters. Note this can take a few minutes to do since training the models, particularly as the number of hidden nodes goes up, can be slow. Write a few sentences describing the data. What size network would you use?*

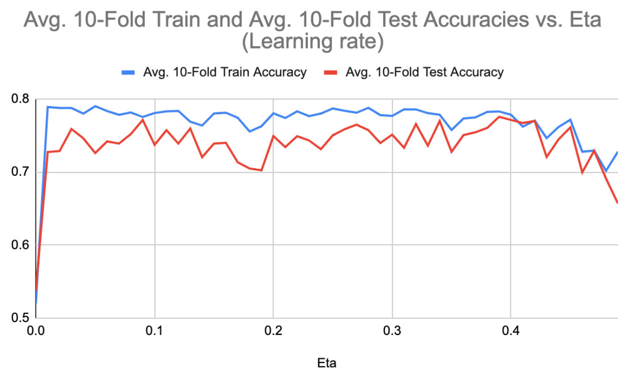
| Hidden Nodes | Avg. Train Accuracy | Avg. Test Accuracy |
|---:|---:|---:|
| 1 | 0.777767 | 0.732864 |
| 2 | 0.774666 | 0.730329 |
| 3 | 0.782908 | 0.704695 |
| 4 | 0.779639 | 0.728995 |
| 5 | 0.777774 | 0.742798 |
| 6 | 0.759273 | 0.72892 |
| 7 | 0.787422 | 0.76554 |
| 8 | 0.76811 | 0.73908 |
| 9 | 0.769693 | 0.720469 |
| 10 | 0.777773 | 0.743005 |

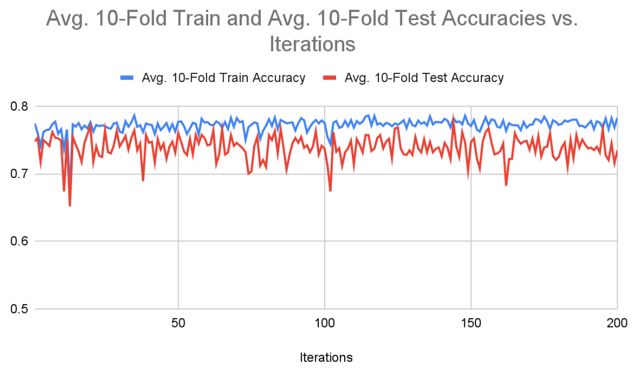


In the data above, we can see that 7 hidden nodes produces the best average training and testing accuracies for the 10-fold cross validation. Based on this data, I would be confident in choosing 7 as a hidden node count for this dataset. Further, 7 hidden nodes lines up with our in-class notion that the number of hidden nodes for a 2 layer network should fall somewhere below the number of examples (642 training) divided by the number of dimensions (6 features).

It’s worth noting that, given the upward trend in accuracy towards the right side of the graph, I can’t for sure rule out that a higher internal node count (say an experiment from 1 to 200 hidden nodes rather than 1 to 10) would yield higher average train/test classification accuracies.

**4.** *Experiment with some of the model parameters (i.e. number of hidden nodes, eta, and the number of iterations) to try and find the best model possible. State how you experimented, the final best model and its 10-fold accuracy, and any other observations that you had.*





For this experiment, I attempted to find the ideal 2-layer NN hyperparameters for the titanic dataset by trying to separately find the best-performing eta (learning rate) and iteration count hyperparameter values. To do this, I used a NN with 7 hidden nodes based on the results from experiment 2, and the default hyperparameter value for the parameter not being tested (i.e. iterations = 200 for the eta test; eta = 0.1 for the iterations test). For each experimental value, the best average training/testing data accuracies were gathered from a 10-fold cross validation and the averages are plotted in the graphs above. Based on these experiments, I obtained the following best-performing individual eta and iteration values:

| | Value: | Accuracy: |
|---|---:|---:|
| Best Eta (train): | 0.05 | 0.789909 |
| Best Eta (test): | 0.39 | 0.775399 |
| Best Iterations (train): | 115 | 0.786021 |
| Best Iterations (test): | 144 | 0.775399 |

I selected the “best” eta and iterations values based on test data averages accuracies, so the ideal network hyperparameter values I found were a hidden node count of 7, an eta value of 0.39, and an iterations value of 144. With the same 10-fold cross validation strategy, a network with these hyperparameter values yielded average training/testing accuracies of 77.358% and 77.1174%, respectively.

Given that these values do not quite surpass the accuracies gathered in the individual eta/iterations tests, I cannot conclude that it is the best 2-layer network for the binary titanic dataset. However, based on the graphical data throughout this writeup (experiments 2, 3, and 4), we can see that our current 2-layer network setup tends to have a performance cap at just below 80%. So, for a relatively small dataset such as this one, the best performing networks (>78% accuracy) could be due to data oddities such as a lucky shuffle during training iterations.

**5.** *Extra Credit: Add in the ability to switch between the tanh activation function and the sigmoid activation function. Add two methods setTanhActivation and setSigmoidActivation to switch between them. Do a small experiment to show how their performance differs.*

After adding in sigmoid activation functionality, I ran a small performance experiment by creating two NN objects (one using tanh activation and one using sigmoid) that both had 7 hidden nodes, eta values of 0.1, and iteration values of 200. I performed a 10-fold cross validation experiment with the networks that yielded the following test data accuracies:

| Fold | TanH Test Accuracy | Sigmoid Test Accuracy |
|---:|---:|---:|
| 0 | 0.704225 | 0.619718 |
| 1 | 0.816901 | 0.774648 |
| 2 | 0.507042 | 0.521127 |
| 3 | 0.859155 | 0.859155 |
| 4 | 0.816901 | 0.830986 |
| 5 | 0.71831 | 0.760563 |
| 6 | 0.690141 | 0.802817 |
| 7 | 0.746479 | 0.802817 |
| 8 | 0.71831 | 0.732394 |
| 9 | 0.866667 | 0.72 |
| avg | 0.744413 | 0.742423 |

Before any statistical analysis, it seems clear that these two networks perform very similarly. Unfortunately, this notion is confirmed by a paired t-test p-value of 0.9336438, so these networks’ respective performance differences are not statistically significant.