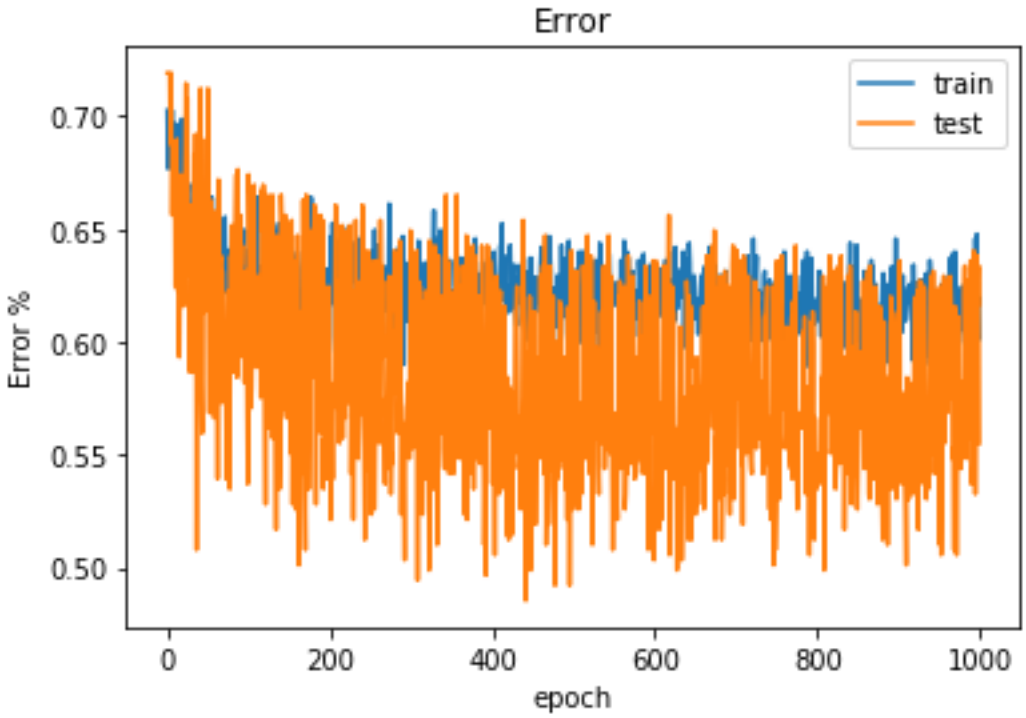
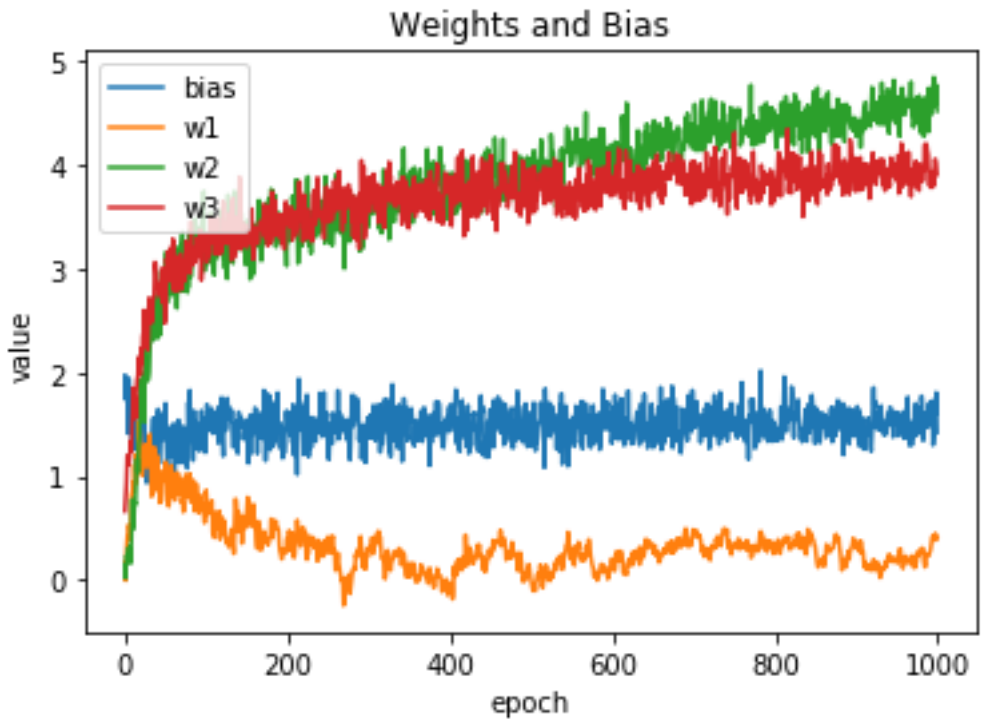
ECS 171 – Homework 2

Note: I am only submitting one python script because scripts for the later questions depend on variables in earlier scripts. However, I have indicated in the comments the parts of the script that corresponds to each question.

1. For my 3-layer ANN, the hidden layer uses a sigmoid activation function. However, the output layer uses a softmax activation function because the probabilities of each class should sum to 1. I also used dropout for the hidden layer, which means that a percentage of the nodes are "dropped" for each training sample. Dropout is used as a regularization technique so that training samples are not encoded in specific nodes and weights sequences.



Both the error for the training set and test set gradually decreased as expected when training the ANN over 1000 epochs. There is a lot more variance for the test error because it is not used to train the ANN, although it is surprising that the test set error is often lesser than the training set. That could just be due to variance from an insufficient amount of training data.



The weights and bias for "CYT" gradually find their optimum values as training progresses (i.e. they seem to be converging).

2. After training on all the available data, the error % was 0.6340970397. If we compare this to the error of only training on the training set by looking at the above graph, the new error % does not seem like an improvement. I expected an decrease in error due to the fact that more data means the ANN can learn to better differentiate the classes. However, with only one hidden layer and three hidden nodes, it appears the ANN could not handle the complexity of the data and could not use the additional data to decrease its error. See handwritten submission for activation function formula.

3. For this question, I changed the structure of my ANN to use only sigmoid as the activation function because I am more familiar with computing gradients for it. I also got rid of any regularization or optimization features. To obtain the weight updates of keras, I used a callback function to obtain weights before and after training for one sample. To calculate by hand, I manually computed values for the forward propagation, back propagation and gradient descent update. Although I said "by hand," these calculations were done using python and numpy to ensure accuracy and speed. The calculations I provided using pen and pencil are the derivation of the gradients that I implemented using python and numpy. This was the ouput from manually updating the weights and bias:

My_W1:

[[0.67811859 -0.14960622 -0.64784516 0.41809467 0.11151346 0.28291029 -0.4496897 -0.41054997]

 [-0.14657533 0.42386738 -0.16817758 0.15312098 -0.67414042 0.54192394 -0.06316818 -0.20181173]

[0.45847673 -0.09419264 0.3840241 -0.16693023 -0.1856012 -0.1647408 0.56241256 -0.57499275]]

My_W2:

[[0.00992303 -0.34705424 0.45786721]

 [-0.23040379 0.20313021 -0.14386142]

 [0.19479067 0.33041922 -0.53591323]

 [0.46285357 0.63425502 -0.27451311]

 [0.18832358 -0.24825921 -0.26220822]

 [-0.08444584 0.5957568 -0.03700102]

 [0.06468397 -0.21673113 -0.3223974]

 [-0.6108867   0.67674021 -0.52079222]

 [-0.21610719 0.12807139 0.04319534]

 [0.07695607 -0.39976691 -0.56660653]]

My_B1:

[[3.39413452e-06]

 [-2.64006509e-03]

 [3.20524540e-03]]

My_B2:

[[0.00463147]

 [-0.00470189]

 [-0.00474816]

 [-0.0057965]

 [-0.00458309]

 [-0.00545   ]

 [-0.00436796]

 [-0.00412251]

 [-0.00493712]

 [-0.00383233]]

These are the weights resulting from keras carrying out gradient descent:

Keras_W1:

[[0.67813414 -0.14959823 -0.64783156 0.41810289 0.11152427 0.28291029 -0.44968104 -0.41054696]

[-0.1450779 0.4246369 -0.16686733 0.15391129 -0.67310053 0.54192394 -0.06233627 -0.20152056]

[0.45670629 -0.09510245 0.38247496 -0.16786464 -0.18683068 -0.1647408 0.56142896 -0.57533699]]

Keras_W2:

[[0.00966327 -0.34726468 0.4575417]

[-0.22831072 0.2048258 -0.14123854]

[0.19690664 0.33213335 -0.53326166]

[0.46550021 0.63639903 -0.27119654]

[0.19035807 -0.24661106 -0.25965875]

[-0.08197715 0.59775668 -0.03390745]

[0.06661315 -0.2151683 -0.31997991]

[-0.6090765 0.67820668 -0.51852381]

[-0.21389727 0.12986164 0.04596465]

[0.07862724 -0.39841309 -0.56451237]]

Keras_B1:

[[2.50002904e-05]

[-5.60301705e-04]

[7.46279198e-04]]

Keras_B2:

[[0.00411247]

[-0.00051998]

[-0.00052051]

[-0.00050859]

[-0.0005182]

[-0.00051761]

[-0.00051349]

[-0.00050576]

[-0.00052175]

[-0.00049337]]

These are the differences between each entry above:

My_W1 – Keras_W1:

[[-1.55568259e-05 -7.99779133e-06 -1.35973215e-05 -8.21716972e-06 -1.08050070e-05   0.00000000e+00 -8.65592651e-06 -3.01169289e-06]

 [-1.49742773e-03 -7.69519130e-04 -1.31024740e-03 -7.90316945e-04 -1.03989072e-03   0.00000000e+00 -8.31905123e-04 -2.91169401e-04]

 [1.77044082e-03   9.09814831e-04   1.54914316e-03   9.34409254e-04   1.22948272e-03   0.00000000e+00   9.83598099e-04   3.44247414e-04]]

My_W2 – Keras_W2:

[[0.00025976 0.00021044 0.00032551]

 [-0.00209307 -0.00169559 -0.00262288]

 [-0.00211597 -0.00171413 -0.00265158]

 [-0.00264664 -0.00214402 -0.00331657]

 [-0.00203449 -0.00164815 -0.00254947]

 [-0.00246869 -0.00199989 -0.00309358]

 [-0.00192918 -0.00156283 -0.00241749]

 [-0.0018102 -0.00146647 -0.00226841]

 [-0.00220991 -0.00179025 -0.0027693]

 [-0.00167117 -0.00135382 -0.00209416]]

My_B1 – Keras_B1:

[[-2.16061559e-05]

 [-2.07976339e-03]

 [2.45896620e-03]]

My_B2 – Keras_B2:

[[0.000519]

 [-0.00418191]

 [-0.00422766]

 [-0.00528791]

 [-0.00406488]

 [-0.00493239]

 [-0.00385446]

 [-0.00361675]

 [-0.00441537]

 [-0.00333896]]

By looking at the above differences, we see that keras differs up to 10e-3 with my calculations, which is not too bad. There could be slight difference in how keras computes gradients to ensure numerical stability or for some other reason.

4. After running each configuration for 1000 epochs, the error matrix is the following:
(rows = 1, 2, 3 hidden layers; columns = 3, 6, 9, 12 hidden nodes)

[[0.59999999 0.52808988 0.51460674 0.48539329]

[0.66516855 0.60449439 0.59550563 0.56179774]

[0.68764046 0.65842697 0.71910113 0.71910113]]

The optimal configuration appears to be 1 hidden layer and 12 hidden nodes because it has the lowest error percentage on the testing set, meaning it's generalizing well to the test set. In general, increasing the number of hidden nodes reduces the error, except for 3 layers of hidden nodes. This suggests that more nodes per layer allows the ANN to better handle the complexity of the data. However, in general, it seems error increases with each additional layer of hidden nodes. Thus, for this specific problem, it appears one hidden layer with many nodes is the best at preventing overfitting and minimizing test set error.

5. The most likely class is CYT. CYT has the highest probability with 46.9%. NUT has the next highest percentage with 42.8%. Then there is a large dropoff in percentage as the next highest is ME3 with 3.9%. It is interesting to note that CYT and NUT was so close. We potentially need more data for the NN to better differentiate between these two classes with more certainty.

6. I see two significant factors affecting uncertainty. For any class prediction, (i) a higher predicted probability for a class means more certainty in the prediction. Example: 90% prediction is more certain than 80%. This makes sense as the probability is basically how certain the NN is that the input belongs to that class; (ii) a greater difference between the probability of a class and the probabilities of other classes indicate more certainty in the prediction for the concerned class. Example: 1st scenario - P(A) = 60%, P(B) = 39% and P(C) = 1%. 2nd scenario - P(A) = 60%, P(B) = 20% and P(C) = 20%. Although P(A) is the same for both scenarios, scenario 1 is less certain for class A because B is relatively closer than in scenario 2.

Here is a basic formula: For class A,

$$uncertainty(A) = 1 - P(A) + \sum_{i=1}^{\#classes\_sorted\_descending} P(i) - P(i-1).$$

Basically, uncertainty decreases if P(A) is high but decreases if the sum of successive probabilities are also high. This formula is far from perfect. For example, it does not work properly if P(A) is not the highest probability class, but it does show some concepts that are relevant to determining uncertainty.