Homework 2
ECS171

Atharva Chalke

November 5, 2018

Problem 1

Read about outlier detection algorithms (e.g. one-class SVM, LOF, Isolation Forest, etc.)and perform at least two of the methods to this dataset. Answer the following questions:(a) are there any outliers on the dataset?, (b) do the methods agree and why?, (c) what are the assumptions behind each method? Remove any outliers and then continue with this new, revised dataset. [10]

Solution

Code: Part1.py , functions.py
a. Based on the methods there are outliers in the data set.

b. Using LOF we get 75.0 outliers
Using Isolation Forest we get 75.0 outliers
The methods categorize 142.000000 points differently.
No, the methods do not agree since they categorize 142 points differently, and agree for only 8 points. This is due to the nature of the algorithms. While, Local Outlier Factor is a density based method that looks at density of a point compared to its neighbors , Isolation forest makes random decision trees and picks outliers that are on an average relatively closer to the node. Pictorially, this can be thought of as outliers needing less lines to separate from other samples, while densely populated points would need a large number of lines to segment out.

c. Both methods assume that an abnormal point would have different features than normal points that belong to a particular class. Isolation Forest is based on the fact that anomalies are data points that are few and different.Thus,segmenting them out is easier than segmenting inliers.
Local Outlier Factor looks at density of a point compared to its neighbors to categorize if a point is an anomaly. Thus, LOF assumes that anomalies are not densely located; therefore, they would categorize a densely located anomaly to be normal or if an anomaly is located within a sparse cluster will be categorize to be normal.

In the code provided, we continued with LOF since points with category 9 were extremely low, and Isolation Forest has a high tendency to categorize them as outliers since they are easier to segment out. Upon visualization using T-Sne, the points looked more densely located using LOF.
Graphs using T-Sne can be loaded by running the code.

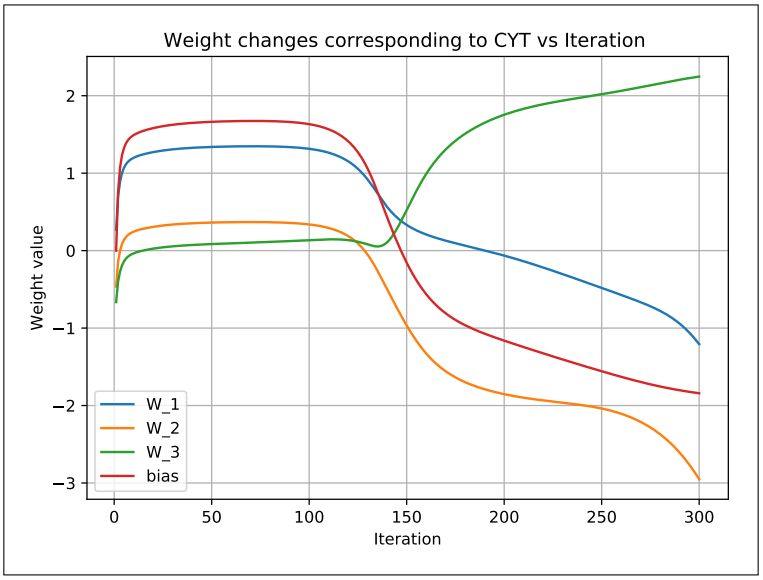
Problem 2

Construct a 4-layer artificial neural network (ANN) and specifically a feed-forward multilayer perceptron to perform multi-class classification. The two hidden layers should have 3 nodes each. Split your data into a random set of 70 % of the samples as the training set and the rest 30% as the testing set. Please note that you will never train with the testing set; the ANN will only take into account the training set for updating the weights.For the most popular class "CYT", provide 2 plots: (I) weight values per iteration for the last layer (3 weights and bias), (II) training and test error per iteration. Use stochastic gradient descent with back-propagation. [20pt]

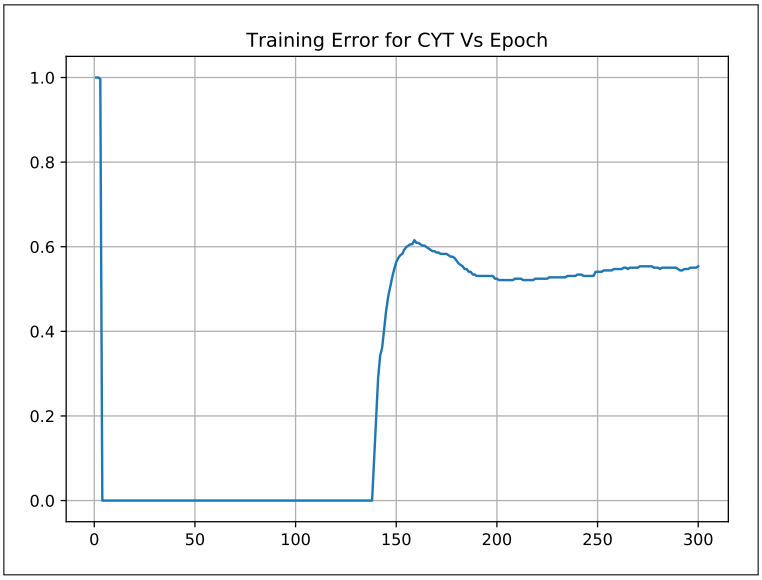
Solution

Code: Part2.py
Graphs: PlotWeights.py, Part2TrainTestPlot.py
The model used sigmoid activation for all layers with a batch-size of 1. The learning rate was set to 0.01 and trained for 300 epochs.

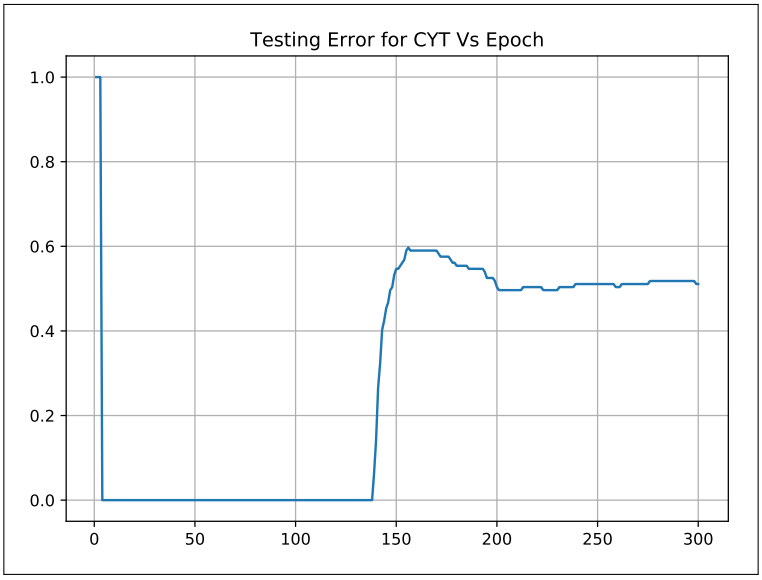
Weights + bias



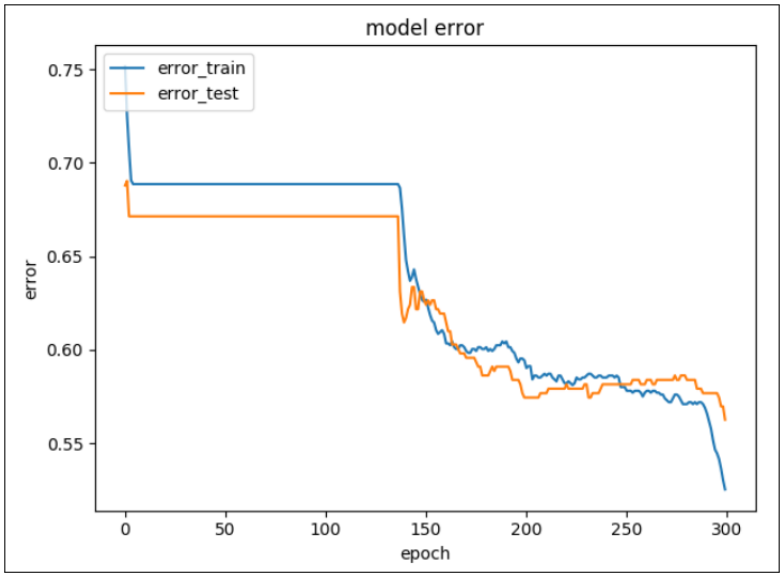
Training error for CYT vs Epoch



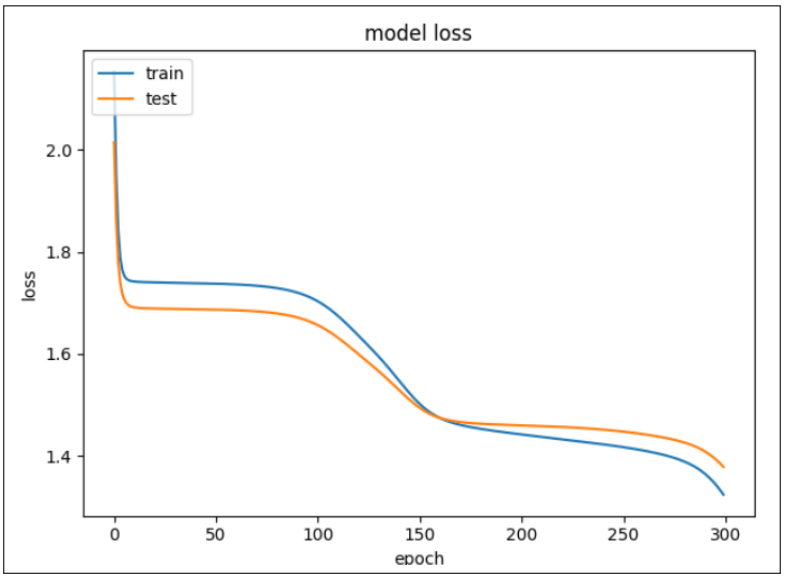
Testing error for CYT vs Epoch



Error vs Epoch



Loss vs Epoch

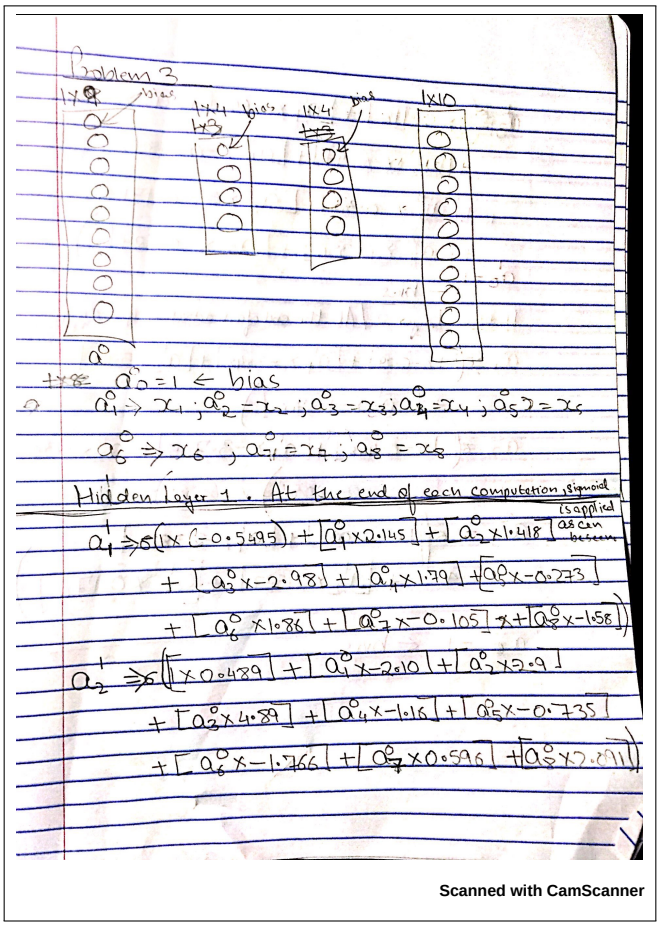


Problem 3

Now re-train the ANN with all your data (all 1484 samples). What is your training error?Provide the final activation function formula for class "CYT" after training (this includes the functional form and corresponding weights from hidden layers necessary to calculate activation of "CYT" in output layer). [10pt]

Solution

Code: Part3.py, Part3activation.py
Note: The weights can be found by running Part3.py and then running Part3activation.py. Make sure the model saved by Part3.py is in the same folder.
The final weight values might be different due to shuffling.
loss: 1.4605 - acc: 0.3908
Training error = 1 - 0.3908 = 0.6092 = 60.92%



$$a_3^1 = \sigma\Big([1 \times 0.116] + [a_1^0 \times 0.644] + [a_2^0 \times 0.51] + [a_3^0 \times -0.47] + [a_4^0 \times -0.248] + [a_5^0 \times -0.33] + [a_6^0 \times -0.171] + [a_7^0 \times 0.356] + [a_8^0 \times -0.627]\Big)$$

$a_0^1 = 1$ ← bias

Hidden Layer 2 / At the end of each activation, sigmoid is applied as can be seen.

$$a_1^2 = \sigma\Big([1 \times -0.29] + [a_1^1 \times -3.48] + [a_2^1 \times 5.101] + [a_3^1 \times -1.3709]\Big)$$

$$a_2^2 = \sigma\Big([1 \times -0.448] + [a_1^1 \times -3.705] + [a_2^1 \times 4.304] + [a_3^1 \times -0.4605]\Big)$$

$$a_3^2 = \sigma\Big([1 \times -0.085] + [a_1^1 \times -2.2825] + [a_2^1 \times 2.537] + [a_3^1 \times 0.176]\Big)$$

$a_0^2 = 1$ ← bias.

Out put layer

$$a_1^3 = \left([1 \times -1.522] + [a_1^2 \times 0.364] + [a_2^2 \times 0.849] + [a_3^2 \times -0.047]\right)$$

$$a_2^3 = \left([1 \times -2.905] + [a_1^2 \times 1.757] + [a_2^2 \times 1.327] + [a_3^2 \times -0.033]\right)$$

$$a_3^3 = \left([1 \times 0.841] + [a_1^2 \times -1.327] + [a_2^2 \times -1.739] + [a_3^2 \times -0.714]\right)$$

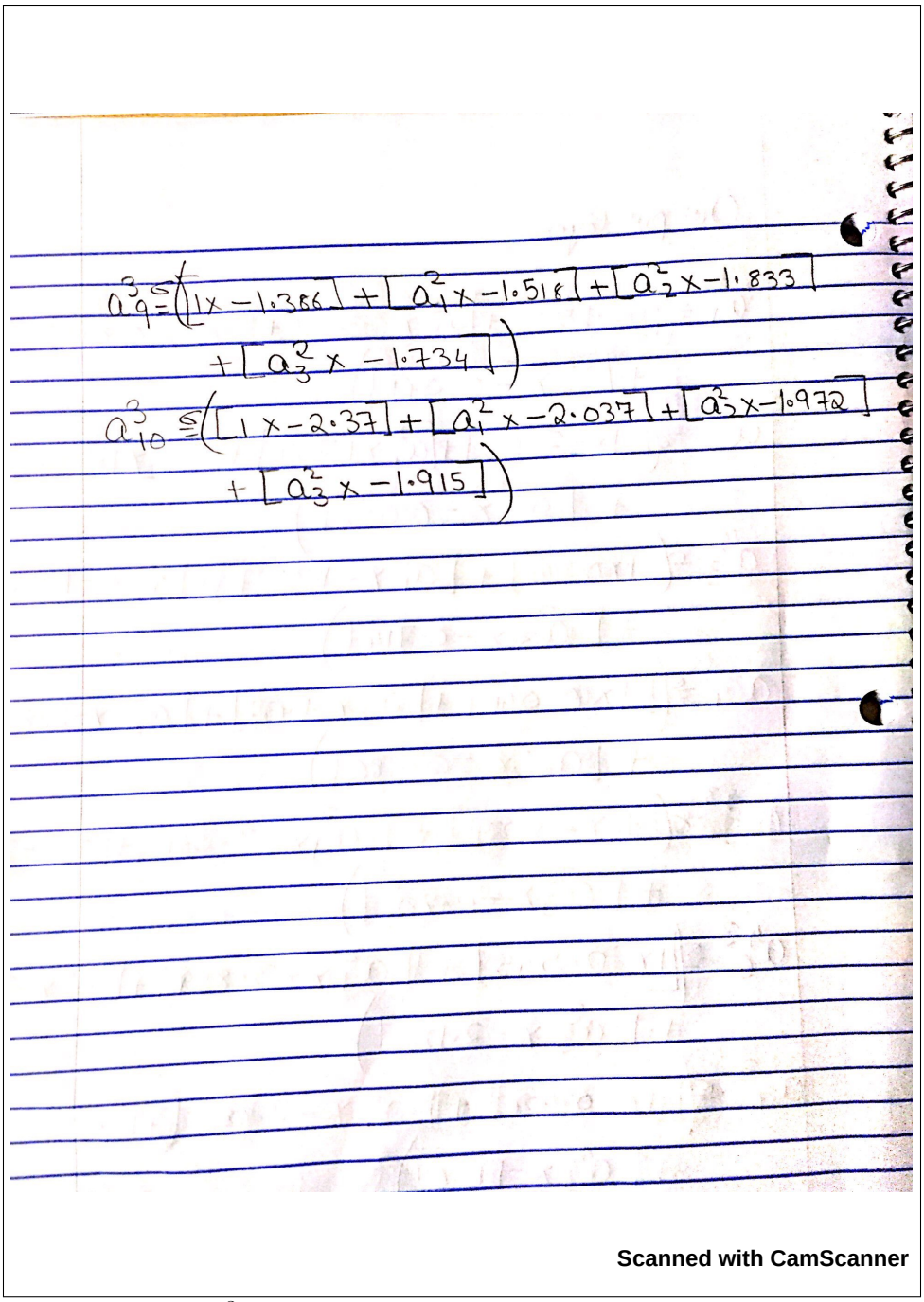
$$a_4^3 = \left([1 \times 0.041] + [a_1^2 \times -1.437] + [a_2^2 \times -1.28] + [a_3^2 \times -0.686]\right)$$

$$a_5^3 = \left([1 \times -0.021] + [a_1^2 \times -3.361] + [a_2^2 \times -2.283] + [a_3^2 \times -0.888]\right)$$

$$a_6^3 = \left([1 \times 0.545] + [a_1^2 \times -3.859] + [a_2^2 \times -3.37] + [a_3^2 \times -2.126]\right)$$

$$a_7^3 = \left([1 \times -0.406] + [a_1^2 \times -2.886] + [a_2^2 \times -2.577] + [a_3^2 \times -1.106]\right)$$

$$a_8^3 = \left([1 \times -1.615] + [a_1^2 \times -1.541] + [a_2^2 \times -0.634] + [a_3^2 \times -1.324]\right)$$



a^3_1 corresponds to activation for CYT.

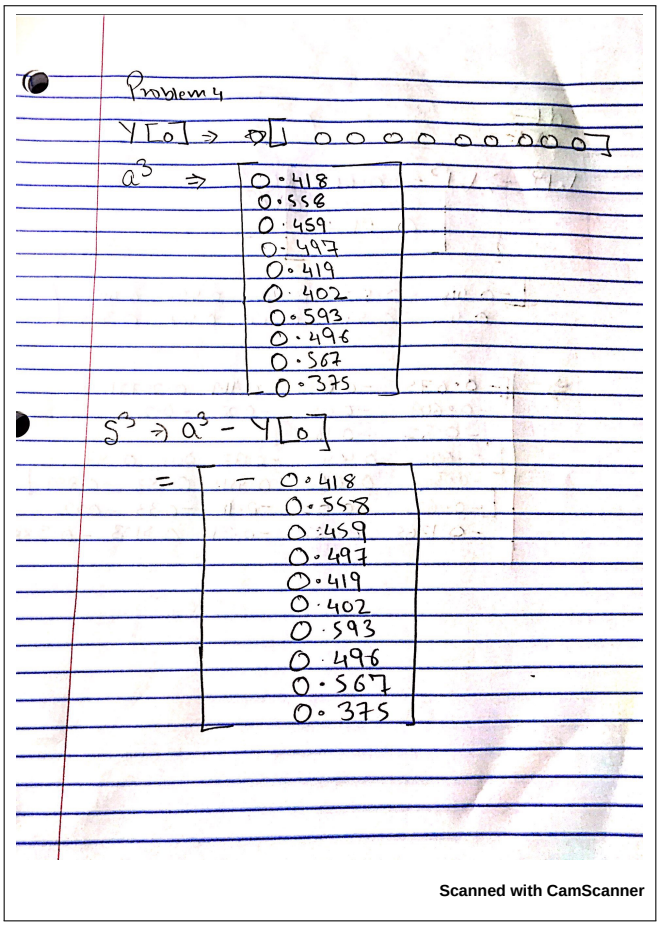
Problem 4

For the ANN that you have built (4 layers, 2 hidden layers, 3 nodes per hidden layer) calculate the first round of weight updates with back-propagation with paper and pencil for the two final layers (output to hidden; 2nd hidden to 1st hidden) for only the first sample.Confirm that the numbers that you calculated are the same with those produced by the code and provide both your calculations and the code output. Provide both calculations made by

hand (scanned image is fine) and corresponding output from the program that shows that both are in agreement. [30pt]

Solution

Code: Part4.py, Part4Val.py
Note: Weights from model can be calculated by running Part4Val.



$$\frac{dL}{dW^3} = \cancel{\log} a^2 \times [S^3]^T$$

$a^2 \Rightarrow$ $\begin{bmatrix} 0.4476 \\ 0.6432 \\ 0.6939 \end{bmatrix} \times [S^3]^T$

$\frac{dL}{dW^3} \Rightarrow$

$$\begin{bmatrix} -0.26 & 0.25 & 0.205 & 0.222 & 0.187 \\ 0.180 & 0.26 & 0.222 & 0.254 & 0.168 \\ -0.37 & 0.359 & 0.295 & 0.319 & 0.270 \\ 0.259 & 0.382 & 0.319 & 0.365 & 0.241 \\ -0.4038 & 0.387 & 0.319 & 0.345 & 0.291 \\ 0.279 & 0.412 & 0.344 & 0.394 & 0.260 \end{bmatrix}$$

$W^3 \Rightarrow$

$$\begin{bmatrix} 0.677 & -0.034 & 0.17 & 0.259 & -0.607 \\ -0.084 & -0.201 & -0.211 & -0.272 & -0.165 \\ 0.636 & 0.409 & -0.174 & 0.180 & 0.472 \\ -0.365 & 0.049 & 0.455 & 0.057 & -0.256 \\ -0.629 & -0.016 & -0.181 & -0.331 & -0.517 \\ -0.173 & 0.632 & -0.308 & 0.516 & -0.3863 \end{bmatrix}$$

$$W^3 = W^3 - 0.01 \times \frac{dL}{dW^3}$$

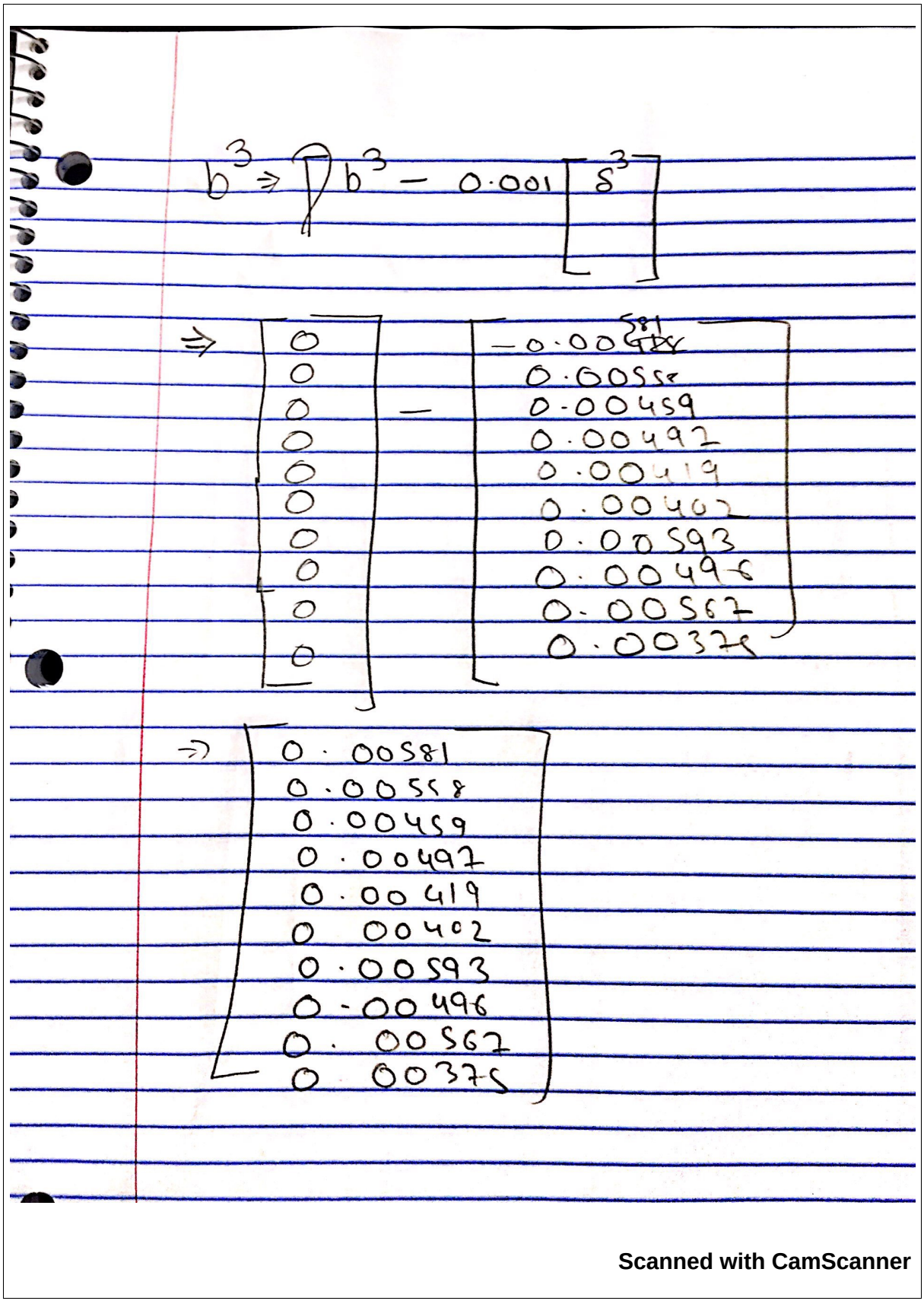
$$\begin{bmatrix} 0.6939 \end{bmatrix}$$

$$\begin{bmatrix} -0.418 & 0.\ldots & \ldots & 0.427 & 0.419 \\ 0.402 & 0.593 & 0.496 & 0.562 & 0.375 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} -0.675 & -0.037 & 0.169 & 0.25721 \\ -0.609 & -0.08 & -0.20 & -0.21 \\ -0.27 & -0.167 \end{bmatrix}$$

$$\begin{bmatrix} 0.610 & 0.404 & -0.177 & 0.15 & 0.47 \\ -0.368 & 0.046 & 0.4516 & 0.053 & -0.259 \end{bmatrix}$$

$$\begin{bmatrix} -0.625 & -0.02 & -0.18 & -0.33 & -052 \\ -0.1765 & 0.627 & -0.31 & 0.5123 & -0.388 \end{bmatrix}$$



$\delta^2 = W^3 \times S^3$ o $gg.a^2(1-a^2)$

$$= \begin{bmatrix} -0.036 \\ 0.023 \\ 0.0408 \end{bmatrix}$$

$\frac{dL}{dW^2} \Rightarrow S^2 \times [a^1]^T$

$$a^1 = \begin{bmatrix} 0.760 \\ 0.402 \\ 0.481 \end{bmatrix}$$

$$\frac{dL}{dW^2} = \begin{bmatrix} -0.027 & -0.0146 & -0.0175 \\ 0.0177 & 0.009 & 0.0112 \\ 0.031 & 0.016 & 0.0196 \end{bmatrix}$$

$$W^2 \Rightarrow \begin{bmatrix} -0.314 & 0.842 & 0.716 \\ 0.701 & -0.820 & 0.764 \\ -0.525 & 0.579 & -0.070 \end{bmatrix}$$

$W^2 = W^2 - 0.01 \frac{dL}{dW^2}$

$$\Rightarrow \begin{bmatrix} -0.314 & 0.842 & 0716 \\ 0.701 & -0.820 & 0.764 \\ -0.525 & 0.579 & -0.070 \end{bmatrix}$$

$$b^2 = b^2 - 0.001 \times \delta^2$$

$$= \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} -0.00036 \\ +0.00023 \\ +0.000408 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} +0.00036 \\ 0.00023 \\ 0.000408 \end{bmatrix}$$

Problem 5

Perform a parameter sweep (grid search) on the number of hidden layers (investigate 1,2or 3) and number of nodes in each hidden layer (investigate 3,6,9,12). Create a 3x4 matrix with the number of hidden layers as rows and the number of hidden nodes per layer as columns, with each element (cell) of the matrix representing the testing set error for that specific combination of layers/nodes. What is the optimal configuration? What you find the relationship between these attributes (number of layers, number of nodes) and the generalization error (i.e. error in testing data) to be? [25pt]

Solution

Code: Part5.py
All models were trained with 100 epochs and a learning rate of 0.01. The hidden layers used sigmoid activation and the output nodes used sigmoid activation. Stochastic Gradient descent was run with a categorical cross-entropy loss function.

	3 nodes	6nodes	9 nodes	12 nodes
1 layer	56.9%	61.4%	49.64%	46.80%
2 layers	61.2%	67.1%	63.5%	67.13%
3 layers	67.13%	67.13%	67.13%	67.13%

The optimal configuration is 1 hidden layers with 12 nodes.
There is no absolute correlation between generalization error, and number of nodes/layer; however , in this case the following happens.

Node Behavior with constant layers
1. For 1 layer , increasing the nodes, reduces increases generalization error till 6 nodes and then reduces as we go from 6 nodes to 9 nodes and then to 12 nodes.
2. For 2 layers, increasing the nodes from 3 to 6, increases the generalization error; however, the error decreases from 6 to 9 and then 12, increases the generalization error again.
3. For 3 layers, increasing the nodes doesn't change the generalization error.

Layer Behavior with constant nodes
1. For 3 nodes, the generalization error increases as we increase layers.
2. For 6 nodes, the generalization error increases with increasing layers.
3. For 9 nodes, the generalization error increases with increasing layers.
4. For 12 nodes, the error increases as we go to 2 layers, and then stays the same at 3 layers too.

Problem 6

Which class does the following sample belong to?[5pt]Unknown Sample 0.52 0.47 0.52 0.23 0.55 0.03 0.52 0.39

Solution

Code: Part6.py The class is 1.0 = NUC.

Problem 7

Can you come up with a quantitative measure of uncertainty for each classification? What is the uncertainty for the unknown sample of the previous question? Justify your assumptions and method [5pt bonus]

Solution

Code: Part6.py

Since, a sigmoid activation is used in the final layer, each activation gives its individual probability and we choose the value with the highest probability. Thus the values after activation give the confidence that the sample belongs to a particular class.

The uncertainty in this case can be defined to be (1− value of activation for that class), which gives us the probability that the sample does not belong to that class.

In case of our previous question, the final layer activations are:
[0.07467858, 0.10460108, 0.0081709, 0.01305773, 0.00107002, 0.00034136, 0.00054121, 0.00321237, 0.00163533, 0.00040831]

Thus, the uncertainty for each class will be (1 - final layer activations).
Therefore, uncertainty for the 10 classes are
[0.9253214, 0.8953989, 0.9918291, 0.9869423, 0.99893, 0.99965864, 0.9994588, 0.9967876, 0.9983647, 0.9995917], where the first element corresponds to the uncertainty for class 0, the second element for class 1 and so on.