[CMSC 678] Intro to Machine Learning HW 4

Shreya Date (PQ56297)

Implementation Details (Solution 3A)

The implementation of this model is kept simple and the code is highly modular.
There is one hidden layer.
The number of neurons in hidden layer, input layer and output layer are kept configurable.
A class is created for Neural Network, which takes number of input, hidden, output neurons and the learning rate as input paramters.
The activation function at each layer used is Sigmoid since it's giving good results whilst avoiding a complicated computation.
The cost function used is squared error loss.
The two sets of weights from input to hidden layer and that from hidden layer to output are initialized using the numpy.random.normal() function. The API samples the weights from a normal probability distribution centred around zero and with a standard deviation that is related to the number of incoming links into a node, 1/√(number of incoming links).

Algorithm:

1. We first have the feed forward part, where predictions are made.
2. These predictions are used to calculate the output error.
3. The output error is used to calculate the output gradient.
4. The weights from hidden layer to output layer are updated with the gradient.
5. Output gradient is nothing but a product of learning rate, output error, and derivative of activation function in the output layer (in this case, sigmoid).
6. Similar procedure is repeated for updating weights from input layer to the hidden layer by calculating the hidden error and hidden gradients.

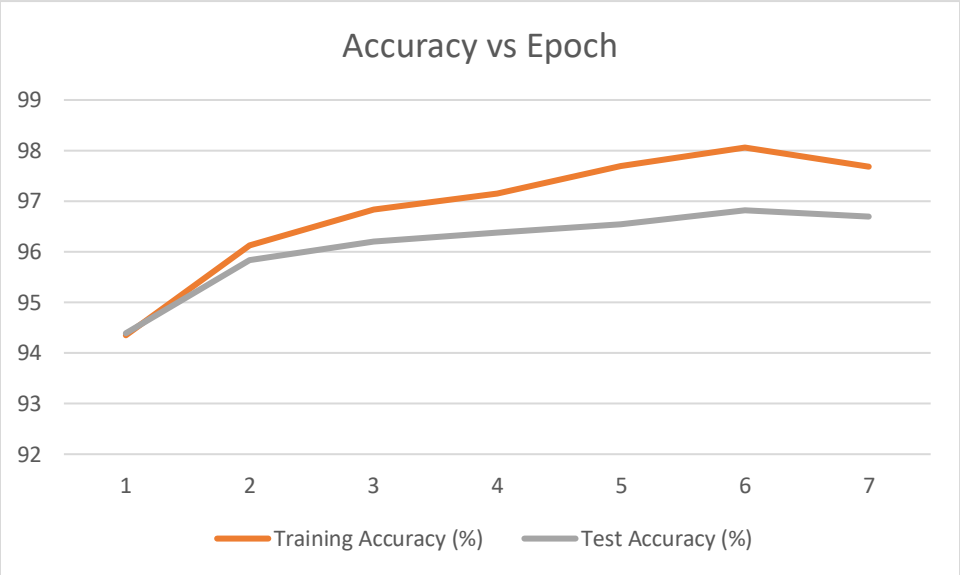
Source Code:
So the source consists of 4 files.
1. NeuralNetwork.py : This contains the actual network implementation
2. ActivationFunctions.py : This is a library sort of which contains the activation functions which can be used by any network. Currently, just the sigmoid is used but later we can make the neural network configurable to accommodate an activation function of the user's choice.
3. Main.py : This is the main file which loads the input data, trains it using an instance of Neural Network and calculates accuracy for training and test data.
4. XOR.py : This is created to feed XOR data to the network and validate that the built neiral network indeed works.
5. Perceptron.py : Baseline model for question 4.

Note: While running the code, make sure that the datasets are placed in data folder with the directory containing the source code.

Convergence Criteria (Solution 3A)

Accuracy vs Epoch

Epoch	Training Accuracy (%)	Test Accuracy (%)
1	94.35	94.39
2	96.13	95.83
3	96.84	96.20
4	97.15	96.38
5	97.70	96.55
6	98.06	96.82
7	97.68	96.70



As the accuracy dropped on increasing number of epochs from 6 to 7, number of epochs = 6 can be our model's convergence criteria. As number of epochs increase, the test accuracy does not increase much, but the training accuracy increases considerably. So epochs = 5 or 6 can be our model's convergence criteria.

Validation (Solution 3B)

To validate that the implementation of neural network is correct, verified it on XOR dataset. The accuracy was about 99% since the predicted values for data set [[0,0], [0,1], [1,0], [1,1]] were approximately [0.03, 0.97, 0.98, 0.02].

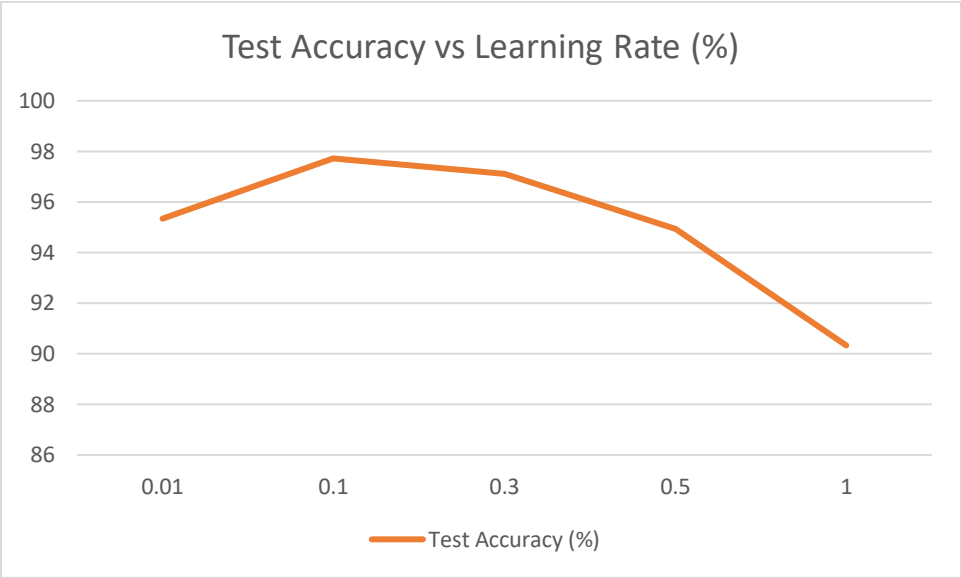
PFA XOR.py in src folder.

Experiments (Solution 3C)

1. Learning Rate vs Accuracy

Learning Rate	Test Accuracy (%)

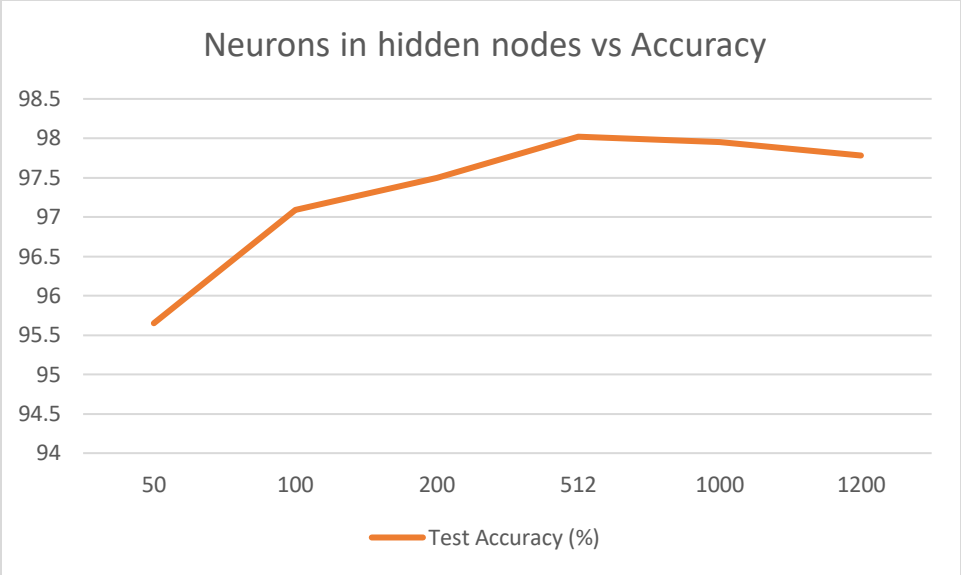
0.01	**95.33**
0.1	**97.72**
0.3	**97.12**
0.5	**94.93**
1	**90.33**



From the above data, we can see that the test accuracy drops as learning rate increases. A learning rate of 0.1 seems to be the best choice for getting good results.

2. Neurons in hidden nodes vs Accuracy

Number of neurons in hidden layer	Test Accuracy (%)
50	**95.65**
100	**97.09**
200	**97.5**
512	**98.02**
1000	**97.95**
1200	**97.78**



From the above plot, we can see that the best accuracy is obtained when number of neurons in hidden nodes is 512. The accuracy drops when number of neurons is 1200. So, we can conclude that adding neurons does not necessarily mean that the accuracy will improve. The time required to train might increase unnecessarily because of a large number of neurons.

3. Input representation vs Accuracy

Input Representation	Test Accuracy (%)
Normalized	**97.72**
Not Normalized	**23.72**

There is a drastic drop in accuracy when input image arrays are not normalized. Normalizing input arrays seems to be a good idea when input data ranges over a wide area.

4. Bias vs Accuracy

Bias	Test Accuracy (%)
Present	**96.28**
Not Present	**97.72**
Learned	**96.66**
Not Learned	**97.39**

Presence of bias indicates a slight drop in accuracy. This is where a bias-variance trade off comes into picture. As bias reduces, variance increases. Higher bias tipped off the model in a wrong direction and the accuracy dropped. Bias measures how far the models' predictions are from the correct value.

Comparisons with base model – Perceptron (Solution 4)

Test accuracy for single epoch, of baseline model, Perceptron: 87.53
Test accuracy for single epoch, of neural network model : 94.39 (with bias, learning rate = 0.1, sigmoid activation, squared error cost)

Average precision-recall score for single epoch, of baseline model, Perceptron: 0.87
Confusion matrix for single epoch, of neural network model:

	precision	recall	f1-score	support
0	0.925	0.987	0.955	6903
1	0.969	0.981	0.975	7877
2	0.952	0.945	0.948	6990
3	0.931	0.947	0.939	7141
4	0.948	0.942	0.945	6824
5	0.993	0.831	0.905	6313
6	0.925	0.983	0.953	6876
7	0.974	0.946	0.960	7293
8	0.904	0.953	0.928	6825
9	0.947	0.929	0.938	6958
accuracy			0.946	70000
macro avg	0.947	0.944	0.944	70000
weighted avg	0.947	0.946	0.945	70000

As we can see, the non-linearity introduced because of the hidden layer in the neural network model gives us a much better accuracy.

References:

1. Tariq Rashid(2016), Make Your Own Neural Network, CreateSpace Publishing
2. https://zahidhasan.github.io/2018-12-21-Bias-Variance-Trade-off-LearningCurve/
3. https://deepnotes.io/softmax-crossentropy
4. https://www.youtube.com/watch?v=QJoa0JYaX1I