Short Report.

Development of a neural network from scratch.

Initially 1 class and 2 functions were created. The class was used for creating an instance of the neural network and the functions for training the network and to query the network respectively. Brief description of the three are mentioned below -

1. **class neuralNetwork:**

   Libraries imported are - **numpy, matplotlib, scipy**.

   - Inside this class is the first **__init__** function with **4 parameters**. The parameters are: input, hidden, output nodes and the learning rate respectively. The parameters were received externally from the instance of the class.

   The inputs are connected to the hidden nodes via weight matrix. Two links had to be established. One to weight matrix to link the inputs with the hidden nodes (named W_{IH}) and the other to link the hidden nodes with the output (named W_{HO}).

   {Include hand drawn figure}

   In the above diagram, W_{IH} implies the connection of any input node I with any hidden node H.

   W_{HO} implies connection of any hidden node H with any output node O.

   The weights are initially set randomly using the **random distribution** library from numpy. The function takes **3 parameters** which are - center of the distribution, standard deviation and the size of the array. Here, we initialized it centering around 0, the standard deviation was set to $\frac{1}{\sqrt{incoming\ node\ links}}$ and the size of the two-dimensional array was set to be equal to the number of hidden and input nodes. In the similar pattern, weight was assigned randomly for the link between the hidden and the output nodes. The row and

column size for the weight matrix was similarly set equal to the number of hidden nodes and output nodes respectively.

A technical detail to be included here is that, the row and column for input or output with the hidden nodes needed to be transposed to match those for matrix multiplications. For instance, $Row_{IH} = hidden\ nodes,\ Column_{IH} = input\ nodes$.

Activation Function - Sigmoid is used here for the first trial. Named as **expit()** this logistic sigmoid function is a special function from scipy library. It takes an input and returns the output from a sigmoid value squashed between [0, 1].

2. **def train() -**

This function takes two parameters - inputs and the target outputs (the output classes). The inputs are fed by the user into the network and it distributes the inputs into the number of classes obtained from the target value. For instance, if the input contains 500 values and the target is set to be 5, the function will distribute those 500 inputs into 5 different target classes. Both the input and target arrays are converted into a two-dimensional matrix and transposed.

After that the signals emerging in the hidden layer was calculated by doing a dot product between W_{IH} and the inputs. This hidden input was passed through the sigmoid activation function to calculate the signals generated in the hidden layer. In similar manner, dot product was done between W_{HO} and hidden outputs which were again passed through the activation function to calculate the signals generating from the final layer.

A technical detail here is to transpose the input and target matrix to match with the hidden inputs and outputs for matrix multiplication.

Error calculation and weight update

Error was calculated simply taking the difference between the target and final output. The weight was adjusted with respect to the learning rate and was re-distributed to the nodes. At first, the updated weight was calculated between the output errors and final outputs, and then distributed to the links between W_{HO} links. In similar pattern, $W_{IH}$was calculated and distributed.

3. **def query() -**

   This function takes one parameter - the input list. It performs the same computations as done in the train function to calculate the signals emerging in and from the hidden layer neurons to the next layer. It would return the final output i.e. the emerging signals from the final output layer which are basically probability values obtained from the activation map.

An object of the neuralNetwork class was instantiated where the user defined input, hidden and output nodes along with the learning rate was passed.

Training Phase

The MNIST dataset was used for the training purpose. The data points for the 10 classes were separated using commas (.). Afterwards, the inputs were scaled to bring the values between 0.01 to 0.99. All targets (output class values) were assigned the value 0.01 except the one with the highest similarity. That means if any input had the highest similarity with that target class then the output probability value from that class would be 0.99 whereas for no similarity the value would be 0.01. In this manner, the neural network was trained with the inputs and the targets.

Testing Phase

First 60000 data points were used for training whereas the remaining 10000 data points served as the test set. Similar splitting and scaling was done as the one done in the training phase. The network was queried with the inputs and the returned values were stored. A library from numpy was used to figure out the greatest values from the network. The network is supposed to show the maximum value on the neuron that has the highest math with the existing data point. For instance if the data point is 3 the 3rd neuron will fire the greatest value.

{Include hand drawn matrix with corresponding values}

A scorecard list was also included in this part to be used as the accuracy metric. For this, we had to match the network produced label with the actual/correct label. If it predicted the correct label 1 was appended to the scorecard and for incorrect classification 0 was appended. The scorecard was then used to calculate the network accuracy percentage.

Learning Rate Variation

For the final part the learning rate was varied keeping the other parameters constant and a loop was run to check the variation of the network accuracy along with different learning rate values. The learning rates were assigned randomly and only one epoch was run to obtain the results presented in the table below -

Learning Rate	**Network Performance**
0.01	91.97
0.02	93.38
0.03	93.77
0.04	94.42
0.05	94.75
0.06	94.67
0.11	95.41
0.21	94.83
0.31	93.95
0.41	92.23
0.51	91.38
0.61	90.14
0.71	85.39
0.81	85.00
0.91	84.85
0.99	84.53

References

1. Make Your Own Neural Networks by Tariq Rashid.
2. Dive Into Deep Learning by Aston Zhang et al.