Backpropagation algorithm
MNIST digit data classification

Dataset:
MNIST dataset with 60000 training images and 10000 images.

Network details:

- Network topology:
  - The neural network built for the image classification problem is a 784 x N x 10 network. The value of N was experimented upon and the final value was fixed as 50. Thus, it is a 784 x 50 x 10 network.
- Output representation:
  - In order to represent the digits in the output layer, one-hot encoding was utilized to generate vectors of size 10 for each digit. Starting with 0 as [1, 0, … ,0] until 9 as [0, … , 1], each digit was converted to its corresponding vector values.
- Neuron activation functions:
  - Hidden layer: Sigmoid function is used as the activation function.
    $$S(x) = \frac{1}{1 + e^{-x}}$$
  - Output layer: Softmax function is used as the activation function since it is well suited to the classification problem.

    $$softmax,\ \sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$
- Energy function:
  - Cross entropy is used as the energy function.
    $$CE\ loss,\ f(s)_i = -\sum_{i}^{C} t_i\ log(s_i)$$
- Other optimization techniques:
  - None

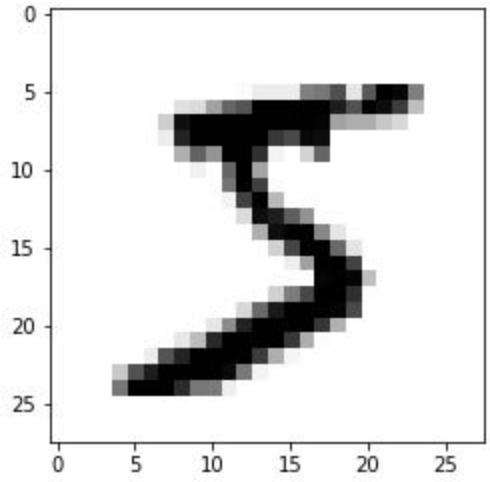
Hyperparameter tuning:

- Learning rate:
  - The learning rate is set to 0.5 for both the hidden layer and the output layer. Initially, the learning rates were set to be inversely proportional to the number of incoming neurons ($\propto$ 1/784 for the hidden layer and $\propto$ 1/50 for the output layer) but the convergence was observed to be slower (approximately 350 epochs).
  - On experimenting with the same values for both the layers, the convergence was reached earlier. A learning rate of 1 was too high, while 0.5 was found to be the optimum value.

    - Moreover, the difference between the subsequent energy costs was observed. If the energy cost increased, the learning rate of the hidden layer was reduced by a factor of 0.1 times the current learning rate.
- Number of layers:
    - Initially, the network was designed with only 1 hidden layer. Since this topology achieved good results, further hidden layers were not added.
- Number of units in the hidden layer:
    - The number of hidden units was fixed as 28 in the initial model.
    - Though this model performed well, other values were used to verify the model performance.
    - When the number of hidden units was increased to 100, overfitting was observed with the training accuracy reaching high 99% while the test accuracy reduced.
    - Thus, the model was trained with 50 hidden units and optimal performance was observed.

Design process:

- Data preparation:
    - The MNIST data had to be converted to the required format before being used to train and test the model.
    - In order to accommodate 784 neurons in the input layer, the data had to be reshaped into 784 x #examples matrix.
    - In order to verify that the inputs were processed correctly, random data points were plotted and verified manually along with their labels.
    - For example, the first data point in the training data was plotted as follows.



- Data normalization:
    - The training and test data had values in the range [0, 255]

    - In order to normalize the data to be in the range [0, 1], the data points were divided by 255.
- Weights initialization:
    - The weights for the different layers were drawn from a normal distribution with zero mean and variance = (1 / incoming connections count).
- Activation functions and cost function:
    - The first model was designed using the sigmoid function for the hidden layer and the hyperbolic tangent function for the output layer.
    - This model was trained to minimize the L2 norm value of D - Y, where D is the desired output and Y is the obtained output value.
    - However, it was observed that this design wasn't successful even when trained for a large number of epochs. The cost was higher and the convergence was slower. On observing the cost for random generated output points, the cost wasn't reduced as fast as required.
    - Thus, cross-entropy energy was utilized in the place of the L2 norm. This was more convenient since the problem at hand was a classification problem.
    - However, this meant that tanh or sigmoid functions cannot be used in the output layer since the log values for negative values were undefined. Softmax function was more suited to the task of generating a probability distribution of the likelihood of the output values.
    - Thus, the sigmoid function was used for the hidden layer and softmax function was used for the output layer.
- Batch learning vs online learning:
    - The model design began by accommodating all the training examples per each epoch since it was presumed that online training would take more time to train.
    - However, it was quickly observed that this was not a reliable approach since the updates to the weights were very small and the training stagnated seemingly at a local minima. The cost didn't reduce on training the training for numerous epochs.
    - Thus, the model was converted to process the input in batches.
    - Starting with 6000 as the batch size, the reduction in cost per epoch was observed. Since the subsequent changes in cost were very small for batch sizes 6000 and 600, the batch size was reduced to 60 and better results were observed.
    - Hence, 60 was chosen as the batch size.
- Convergence criteria:
    - The earlier version of the model was trained until the cost dropped below a certain threshold.
    - However, the difference between subsequent energy values was utilized finally since it was already being used to modify the learning rate, the same was utilized for checking convergence.
    - The model was considered to have completed training when the subsequent energy difference was less than $1e^{-4}$.

- Reducing the threshold further lead to overfitting the training data.

Pseudocode:

1. Load the MNIST data and labels, and reshape as follows:
   trainX $\in R^{784x60000}$, trainY $\in R^{60000x1}$
   testX $\in R^{784x10000}$, testY $\in R^{10000x1}$
2. Normalize the training and test data by dividing by 255 elementwise.
3. Convert the training and test labels into one-hot encoded values. For example, 0 is converted to $[1, 0, 0, 0, 0, 0, 0, 0, 0, 0]^T$
   Thus, trainY $\in R^{10x60000}$, testY $\in R^{10x10000}$
4. Set learning rates eta1 (for the hidden layer) and eta2 (for the output layer) as 0.5.
5. Set the batch_size = 60 and calculate the number of batches to be processed per epoch. Let us call this value 'batches'.
6. Initialize W1 $\in R^{50xbatch\_size}$ and b1 $\in R^{50x1}$ drawn separately from a normal distribution with mean 0 and variance (1 / 784). Similarly, initialize W2 $\in R^{10x50}$ and b2 $\in R^{10x1}$ from a normal distribution with mean 0 and variance (1/50).
7. Initialize cost_tracker and cost_tracker_test as empty arrays to save the cost values for each epoch for the train and test data. Similarly, initialize misclassifications_counter and misclassifications_counter_test as empty arrays to save the number of misclassifications for each epoch for the train and test data.
8. Do:
   a. For i = 0 to 'batches',
      i. Extract the trainX and trainY values corresponding to the current batch window and call them X_batch and Y_batch.
      ii. Perform the forward pass as follows:
         1. a1_induced = W1*X + b1
         2. a1_activated = sigmoid(a1_induced), where sigmoid(z) = $1 / (1-e^{-z})$
         3. a2_induced = W2*a1_activated
         4. a2_activated = softmax(a2_induced),
            where $softmax$, $\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
         5. y = a2_activated
      iii. Perform the backward pass as follows:
         1. y_prime = Y_batch - a2_activated
         2. a2_backward = 1
         3. a1_backward = sigmoid'(a1_activated)
         4. dW2 = - 1 / batch_size * y_prime * a1_activated.T * a2_backward
         5. db2 = - 1 / batch_size * $\sum_{i=1}^{batch\ size} y\ prime[i, :]$, i.e., the average of the y_prime values rowwise.
         6. tempval = W2.T * y_prime * a1_backward

7. dW1 = - 1 / batch_size * tempval * X_batch.T
8. db1 = - 1 / batch_size * $\sum_{i=1}^{batch\ size} tempval[i, :]$, i.e., the average of the value rowwise.

iv. Update the weights and biases as follows:
1. W1 = W1 - eta1 * dW1
2. b1 = b1 - eta1 * db1
3. W2 = W2 - eta2 * dW2
4. b2 = b2 - eta2 * db2

b. epoch ← epoch + 1
c. Generate Y values for each input using the forward pass in 8.a.ii.
d. Calculate the cost for training data using cross-entropy.

$$CE\ loss, f(s)_i = -\sum_{i}^{C} t_i\ log(s_i)$$

Append the value to cost_tracker. Similarly, generate the cost for the test data and append to cost_tracker_test.
e. Calculate the misclassifications for the train and test data and append to misclassifications_counter and misclassifications_counter_test correspondingly.
f. If the absolute difference subsequent cost values < threshold value (~ $1e^{47}$), break out of the loop.
g. If the current cost value > previous cost value, update eta1 = eta1 - 0.1 * eta1

Final results:

- Number of misclassifications on the training data: 12
- Accuracy on the training data: 99.98%
- Number of misclassifications on the test data: 283
- Accuracy on the test data: 97.17%