Problem 2: Learning to implement Neural Network

Colab link:- https://colab.research.google.com/drive/1FMafrelBpmbE-nFJLdi7HZWmTPvuuDpG#scrollTo=mGMlu1A6wbxK

In this exercise a simple neural network has been developed **from scratch** in python to recognise gurumukhi digits. First I have uploaded all training and test images of gurumukhi digits onto my google drive to be mounted for colab notebook.Then those images have been converted into 20X20 matrix of pixels using cv2 package to be used as training and validation set for subsequent processing. For this model, I decided to go with sigmoid and softmax as activation functions to keep things simple. The neural network is going to be a simple network of four layers. The input layer consists of 400 units corresponding to every pixel in the 20 by 20 image. The second layer(*hidden layer*) drops down to 128 units , the third layer drops down to 64 units and lastly the final layer with 10 units corresponding to digits 0–9. Functions can be generalized as processes that takes in an input and spits out an output. Activation functions are those functions that take in weighted sum of neurons as input(*varying in magnitude*) and turn it into meaningful data that can be easily understood or fed into the next layer. Sigmoid and softmax are two such activation functions that are used here. In my model, I use the sigmoid function to squish the random outputs given out by layer into numbers between 0 and 1. A Softmax function takes in a vector as input and spits out a vector of same size having elements that sum up to 1. Every element in the output vector is between 0 and 1, and thus these values can be interpreted as probabilities. In my model, the output layer spits out a vector of shape 10 having different magnitudes. Hence, I use softmax to normalize my result. I have used backpropagation to update the weights.

I have defined a class named DNN where there is init method which is basically a constructor that accepts layer size,epochs and learning rate.this method initializes the weights using random numbers and hidden layers.Then there are sigmoid and softmax functions which are defined as per their formula. There is a forward pass function which basically caculates output of each layer.Here it caculates first the dot product of weight and input from previous layer and then it passes the result to sigmoid function to get the output between 0 & 1 of each hidden layer.After that backward pass function is there which basically calculates gradient of weights of each layer to be used in SGD process.

I have trained the model with various combinations of epochs and learning rates to come up with optimum accuracy. First I have tried with epoch as 500 and learning rate as 0.001 but it gives me accuracy around 75%. Then I have tried with epoch as 1000 and learning rate as 0.001 but it gives me accuracy around 85%. Finally I have used epoch as 2000 and learning rate as 0.01 and it gives me accuracy around 93%.

```
dnn = DNN(sizes=[400, 128, 64, 10], epochs=2000, lr=0.01)
dnn.train(train_list, test_list, 10)
```

```
Epoch: 0, Time Spent: 0.70s, Accuracy: 9.55%
Epoch: 200, Time Spent: 170.39s, Accuracy: 59.55%
Epoch: 400, Time Spent: 338.60s, Accuracy: 78.09%
Epoch: 600, Time Spent: 506.61s, Accuracy: 87.08%
Epoch: 800, Time Spent: 677.01s, Accuracy: 93.82%
Epoch: 1000, Time Spent: 846.08s, Accuracy: 93.26%
Epoch: 1200, Time Spent: 1014.55s, Accuracy: 92.70%
Epoch: 1400, Time Spent: 1183.54s, Accuracy: 92.70%
Epoch: 1600, Time Spent: 1352.80s, Accuracy: 92.70%
Epoch: 1800, Time Spent: 1521.87s, Accuracy: 92.70%
Epoch: 2000, Time Spent: 1689.79s, Accuracy: 92.70%
```