PMDS603P Deep Learning Lab Experiment 3

July 2025

1 Work to do today

Note: Make a single PDF file of the work you are doing in a Jupyter notebook. Upload with the proper format. Please mention your name and roll no properly with the Experiment number on the first page of your submission.

Question1. Today, we will try to recall the work done in the previous lab first. The second problem attempted in the last lab was to use MNIST dataset which contains handwritten numbers (their images) from 0 to 9 digits. First try to fit a simple neural network model. Let us import the necessary modules required for this along with the dataset.

It contains 70000 handwritten images of digits from 0 to 9. So its a 10 class classification problem. Lets try to create a model that can do the classification task.

```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD
import matplotlib.pyplot as plt
batch_size = 128
num_classes = 10
epochs = 10
(x_train, y_train), (x_test, y_test) = mnist.load_data()
plt.imshow(x_train[3])
print(x_train[3])
```

Next let us reshape the whole images in to a single vector and normalize them.

```
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

• Do the one-hot encoding for the labels with appropriate functions for both training and testing data.
• Now here I have given the number of iterations as 25 but it can be changed, and the results

can be observed.
• Now create a simple neural network model using a sequential class with the following architecture.
Two hidden layers with 512 neurons in each layer with a sigmoid activation function. An output layer with 10 neurons with a softmax activation function. (Since it's a multiclass classification problem)
• Compile and fit the model with an appropriate loss function and use the SGD optimizer. You can change the learning rate in your optimizer SGD by defining
sgd1 = SGD(learning_rate = 0.01)
and then use sgd1 as your optimizer while compiling your model.
• You can print your model validation error and accuracy using You will get your validation

```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

set loss and accuracy.
• Now if you want to check one case, how your model is predicting you can try any one image print it and use your model to predict the same and print. The argmax function gives us the index of the class with highest probability.

2 Regularization Techniques

Next, we will see how we can use different regularization techniques to get better accuracy for the models.

- **Dropout Technique.**
  Now Already you have already created a model in the previous section. Now use this code to import Dropout layers.
  from keras.layers import Dropout
  Further, try to include a Dropout layer after your first and second hidden layers by using the code
  model.add(Dropout(0.2))
  Thus, in the layer before that, 20% of neurons become inactive during training in an epoch.
  But it's important to understand that the dropout parameter value that you see as chosen as 0.2 is a hyperparameter that can be tuned and found out. An appropriate parameter value will give you better results. Try manually whether you can come up with an appropriate dropout parameter value for your model by giving different values for the parameter.

- **Early Stopping:** Next, we can see how the early stopping technique can be implemented to reduce overfitting of the model and help it to generalize well.
  Now, for that, let's first divide the training data further into a train and a validation set. Do the one-hot encoding part for all three sets of data. Training, validation, and testing set.

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
X_subtrain,X_valid,y_subtrain,y_valid = train_test_split(x_train,y_train,test_size=0.10,random_state=1)
```

Now create your model with appropriate activation functions and loss functions, etc, and print the model summary of your model.
Further import the class early stopping from keras using
from keras.callbacks import EarlyStopping
Define an object of the early stopping class

```
estop = EarlyStopping(monitor = 'val_loss', min_delta = 1e-5,mode = 'min',patience=4,
    verbose=1,restore_best_weights=True)
```

Note:
monitor ='val_loss'
patience=4
min_delta: Minimum change in the monitored quantity to qualify as an improvement, i.e. an absolute change of less than min_delta, will count as no improvement.
mode: In min mode, training will stop when the quantity monitored has stopped decreasing;
in "max" mode it will stop when the quantity monitored has stopped increasing;
restore_best_weights=True
Further compile the model, and while fitting the model, use

```
model.fit(X_subtrain, y_subtrain,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(X_valid, y_valid),callbacks=[estop])
```

Here we have used the validation set for the validation part and then defined callbacks, as you see above, which will help perform early stopping.
Now you can evaluate your accuracy of the model and verify.

Challenging Question: Try for a scratch code for this case where you can create a custom neural network without using any inbuilt classes like sequential etc. Where you need to define a class neural network which has methods like forwardpass, backwardpass, and train. Figure out how we can do this.

This model has inputs as [0, 0, 1], [0, 1, 1], [1, 0, 1], [1, 1, 1] and the expected output as [0], [1], [1], [0] in each case. So there are three features in our dataset as you see above. The activation function is to be taken as sigmoid. The architecture is like we have only one hidden layer and an output layer with one neuron. Take the error function as $(1/2)(y - \hat{y})^2$.