09
DEC
2020

ASSIGNMENT 3

SUBJECT : DEEP LEARNING [IT702]

SHANKARANARAYAN N

SUBJECT : DEEP LEARNING [IT702]

SHANKARANARAYAN N | M.TECH(RESEARCH) IT

ASSIGNMENT 3
SUBJECT : IT702 - DEEP LEARNING

AIM :

- **TASK0 : Creating a basic network**
- **TASK1 : Analyzing performance**
- **TASK2 : Building confusion matrix**
- **TASK3 : Overfitting**
- **TASK4 : Visualizing a neural network**

DATASET : MNIST DATASET

TASK 0 : CREATING A BASIC NETWORK

- A basic neural network with 2 hidden layers and with a learning rate of 0.01 and relu activation function is built.

TASK 1 : ANALYZING PERFORMANCE

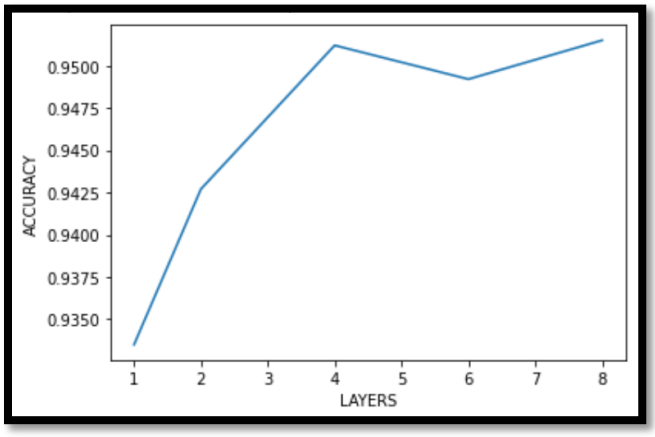
- Here we are analyzing the performance of the model by varying the parameters one by one as described below and keeping the rest of the values constant (and equal to the default values),
  - Batch size: 1, 2, 4, 8, 16, 32, 64, 128 (default 32).
  - Number of hidden layers: 1, 2, 4, 6, 8 (default 2).
  - Learning Rate: 0.01, 0.05, 0.1, 0.2, 0.4, 0.8 (default 0.01).
- We also plot the change in accuracy with respect to the change in parameters

TASK 1 : QUESTIONS

- Write a brief description of the variation observed in each graph and your hypothesis explaining the variation in your own words.
- Should accuracy alone be the criterion deciding a parameter setting?
- What could be other considerations in practice?

TASK 1: QUESTION 1 RESULT

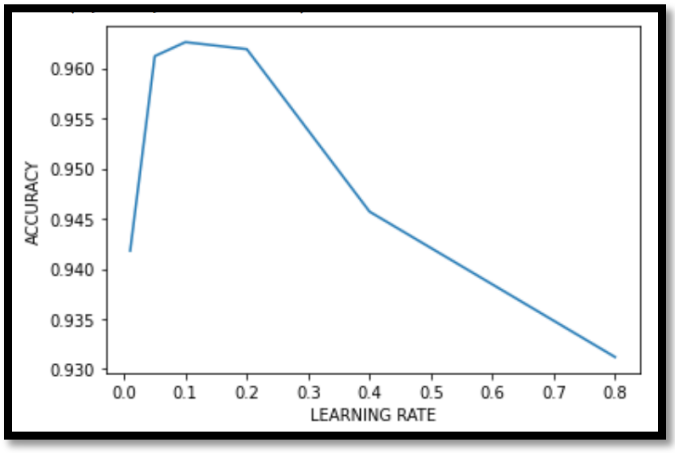
- Change in accuracy with respect to number of hidden layers



We obtain the following graph upon changing the number of hidden layers. The accuracy steadily increases with increase in number of layers, but after a contain point (in this case 4) the accuracy starts falling down and then raises again.

Own Hypothesis : This might be because introducing more layers may need more iterations to learn.

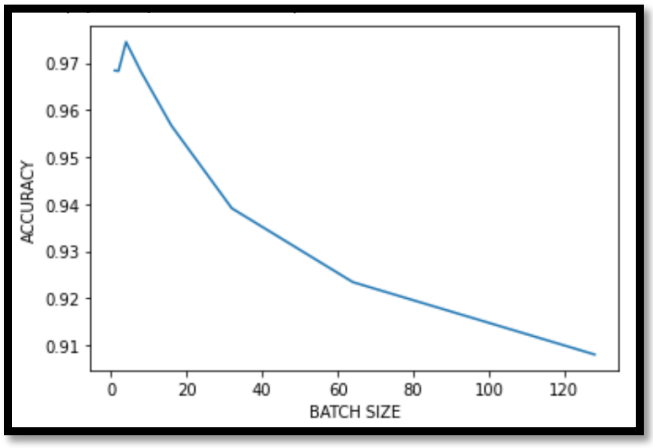
- Change in accuracy with change in learning rate



We obtain the following graph upon changing the learning rate. The accuracy steadily increases with increase in learning rate, but after a contain point (in this case 0.1) the accuracy starts falling down.

Own Hypothesis : This is similar to the concept of learning rate in grading dissent. High learning rate may some times affect our model.

- Change in accuracy with change in batch size



We obtain the following graph upon changing the learning rate. The accuracy increases with increase in learning rate, but after a contain point (in this case 0.4) the accuracy starts falling down.

Own Hypothesis : The accuracy decreases with increase in batchsize. But still if the batch size is too small the accuracy decreases again

IT IS OBSERVED THAT

[LEARNING RATE = 0.01 | BATCH SIZE=04 | Layer=2] Gives the best accuracy rate of more than 96%

TASK 1 : QUESTION 2 ANSWER [Should accuracy alone be the criterion deciding a parameter setting?]

- Accuracy can sometimes be misleading as a model with great accuracy rate doesn't necessarily mean that the model posses a great predictive power. In some cases, a model with low accuracy rate can have more predictive power.

- Example : In a problem, if there exist a large class imbalance. This model will be able to predict the value of the majority class for all predictions and achieve a high classification accuracy, the problem occurs when the test data is from the other lacking class.

TASK 1 : QUESTION 3 ANSWER [What could be other considerations in practice?]

- In the above mentioned scenario, additional parameters such as F1 Score, Precision, Recall can be used.

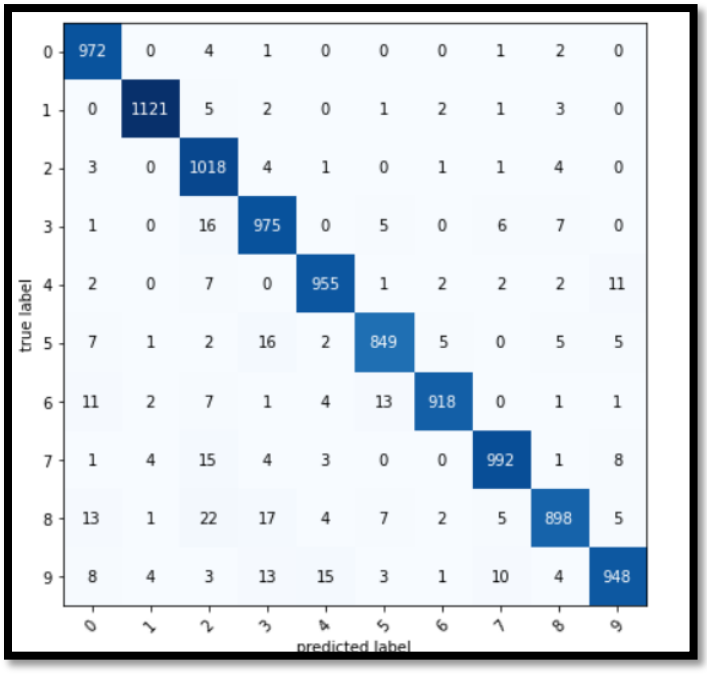
TASK 2 : PLOTTING A CONFUSION MATRIX

- Confusion matrix is a square matrix with one row and one column for each label. Each cell in the matrix contains the number of instances whose true label is that of row of the cell, but which our classifier predicted as the label corresponding to the column of the cell.
- Here, we plot a confusion matrix after training out model with the hyperparameters of the network to the values that resulted in the maximum validation accuracy in Task 1.

TASK 2 : QUESTIONS

1. Between which two classes does your model get the most confused? Which one of those is the true label and which one is the prediction?
2. What would you do if you wanted to make fewer misclassifications of one particular class?

TASK 2 : RESULT



Question 1 :

The model's accuracy rate is high and based on the confusion matrix we can conclude that the model isn't getting confused in most cases.

Question 2 :

Increasing the number of neurons and training set and avoiding underfitting/overfitting might help to decrease the number of miscalculations

TASK 2 : PLOTTING A CONFUSION MATRIX

- **Training the neural network on just the first 1000 of the 50,000 training examples, for 500 iterations.**

TASK 3 : QUESTIONS

1. How many parameters does your neural network created in the Task 2 have? Show the calculation
2. With so many parameters, would it always be the case that the neural network you have created generalizes well? Find out by training the neural network on just the first 1000 of the 50,000 training examples, for 500 iterations. Write down the loss and the accuracy on the training set as well as the test set for the trained network

TASK 3 : QUESTION 1 [Calculation done after Task 2]

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_20 (Dense)             (None, 100)               78500
_________________________________________________________________
dense_21 (Dense)             (None, 100)               10100
_________________________________________________________________
dense_22 (Dense)             (None, 100)               10100
_________________________________________________________________
dense_23 (Dense)             (None, 10)                1010
=================================================================
Total params: 99,710
Trainable params: 99,710
Non-trainable params: 0
_________________________________________________________________
```

CALCULATION

- This model has 2 hidden layer with 100 nodes using the relu activation function. The resulting architecture has 99,710 tunable parameters.
- From the input layer to the hidden layer there are 784*100 = 78400 weights.
- The hidden layer has 100 nodes so there are 1000 biases.
- This brings us to 78,400 + 100 = 78,500 parameters.

TASK 3 : QUESTION 2

- The Dataset is split in such a way that only the first 1000 training examples are used for training the model. Epoch is set to '500' for 500 iterations.

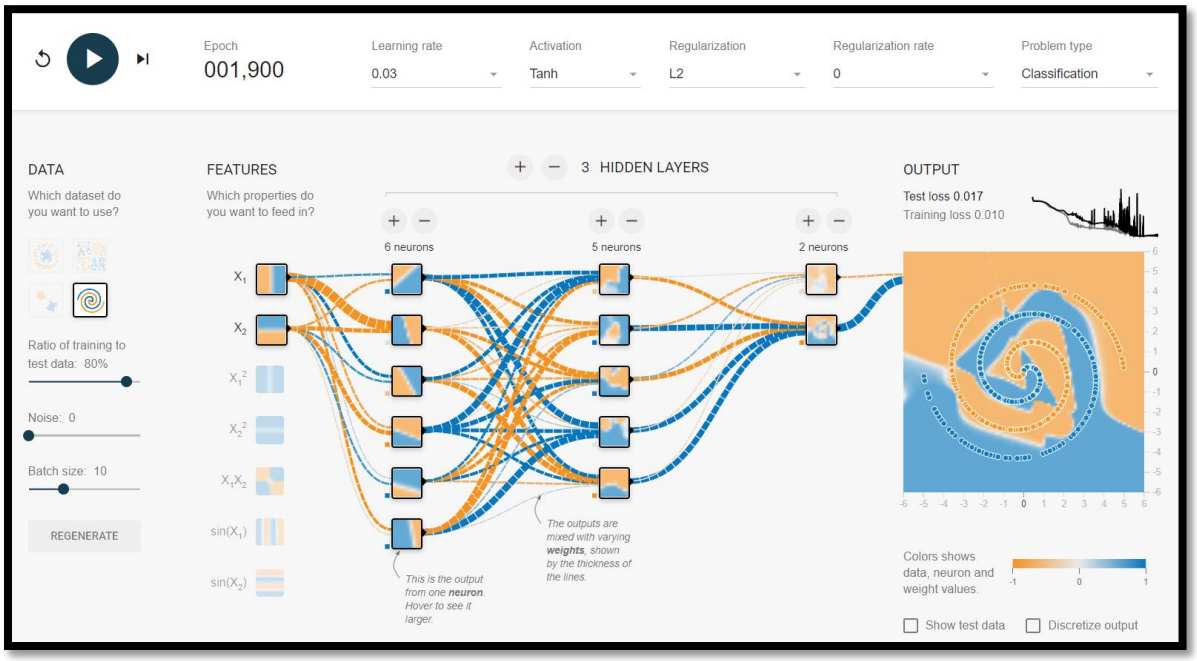
```
Training set score: 0.00010053347796201706
Training set accuracy: 1.0
Validation set score: 0.8988227844238281
Validation set accuracy: 0.8670338988304138
```

- The result shows that the neural network that we have created doesn't generalize well. The model is actually overfitting.
- Overfitting here can we best explained as state where an estimator has begun to learn the training set so well that it has started to model the noise in the training samples. In our case as we have used only 1000 training examples for 500 iterations which overfits the model. To prevent this we should use limited number of iterations or we could also reduce the number of parameters. And to increase the accuracy more training examples should be used.

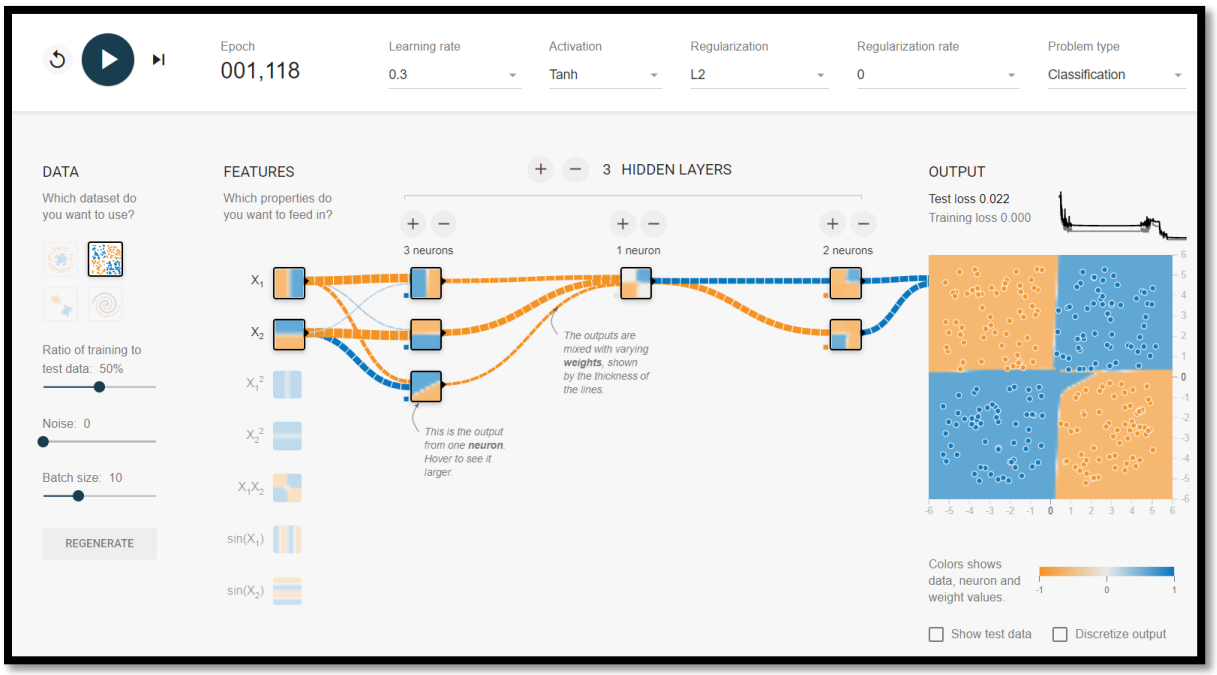
TASK 4 : TENSORFLOW PLAYGROUND

Here we should achieve a test loss of 0.07 or less. For Spiral model, Circle model, XOR model

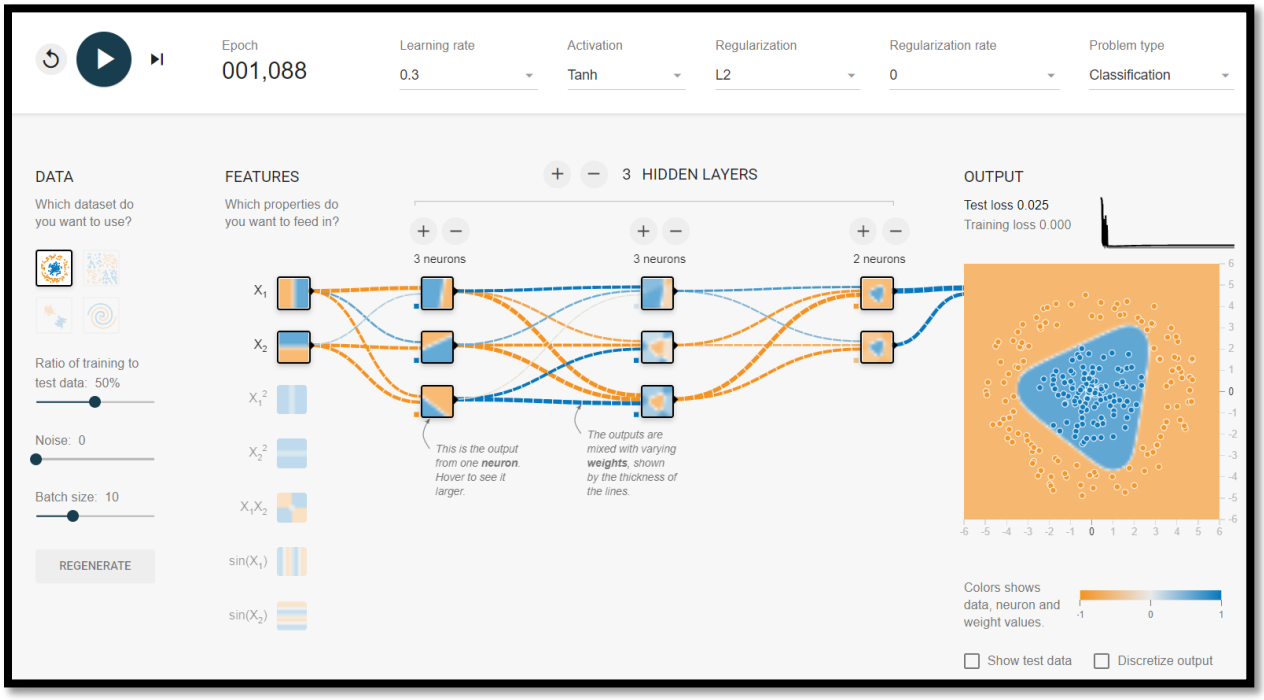
TASK 4 : SPIRAL [2 Hidden layers with 6 and 5 neurons respectively]



TASK 4 : XOR [2 Hidden layers with 3 and 1 neurons respectively]



TASK 4 : CIRCLE [2 Hidden layers with 3 and 1 neurons respectively]



[END OF ASSIGNMENT 3]

ALL THE GIVEN TASKES WERE SUCCESSFULLY COMPLETED AND THE RESULTS ARE DISPLAYED HERE. THE **CODE** is attached with this document as uploading of multiple documents is not supported in moodle.

ASSIGNMENT 3

SUBJECT : IT702 - DEEP LEARNING

SYBMITTED BY : SHANKARANARAYAN N, M.TECH(RESEARCH), DEPARTMENT OF INFORMATION TECHNOLOGY

TASK 0 : Creating a basic network (Ungraded)

TASK 1 : Analysing performance

TASK 1 : ANALYZING PERFORMANCE Here we are analyzing the performance of the model by varying the parameters one by one as described below and keeping the rest of the values constant (and equal to the default values),

- Batch size: 1, 2, 4, 8, 16, 32, 64, 128 (default 32).
- Number of hidden layers: 1, 2, 4, 6, 8 (default 2).
- Learning Rate: 0.01, 0.05, 0.1, 0.2, 0.4, 0.8 (default 0.01).

We also plot the change in accuracy with respect to the change in parameters

TASK 1 : QUESTIONS

- Write a brief description of the variation observed in each graph and your hypothesis explaining the variation in your own words.
- Should accuracy alone be the criterion deciding a parameter setting?
- What could be other considerations in practice?

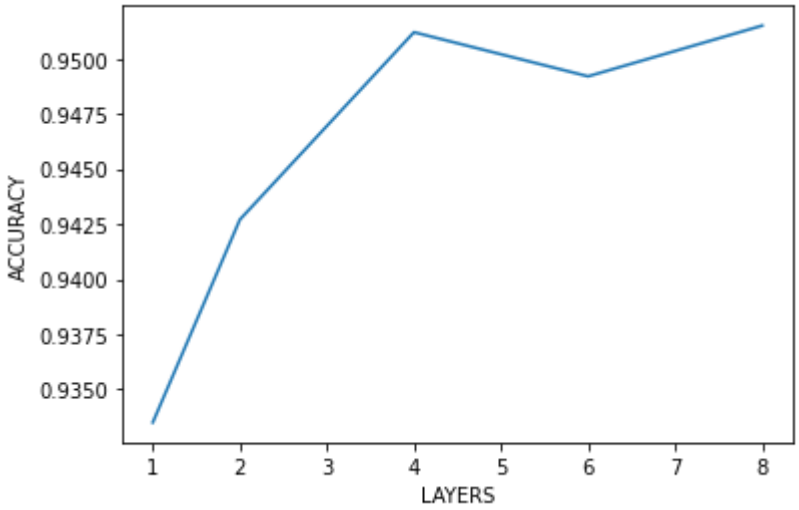
```
import numpy as np
# Use command line arguements for Task 1 (sys.argv)
import sys
prev=0.001
layers=[0,1, 2, 4, 6, 8 ]
batchsize=[0,1, 2, 4, 8, 16, 32, 64, 128]
learningrate=[0,0.01, 0.05, 0.1, 0.2, 0.4, 0.8]
print("")
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.utils import np_utils, to_categorical
from keras.optimizers import SGD
from keras.datasets import mnist
import matplotlib.pyplot as plt
```

```
# Load pre-shuffled MNIST data into train and test sets
(X_train_1, train_labels_1), (X_test, test_labels_1) = mnist.load_data()

# Preprocess input data
X_train_1 = X_train_1.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
X_train_1 = X_train_1.astype('float32')
X_test = X_test.astype('float32')

X_train_1 /= 255
X_test /= 255

# Divides the dataset into train and validation sets
X_valid = X_train_1[50000:60000]
X_train = X_train_1[:50000]
print(X_train.shape[0], 'train samples')
print(X_valid.shape[0], 'validation samples')

 #newly added



# Preprocess class labels
train_labels = np_utils.to_categorical(train_labels_1, 10)
test_labels = np_utils.to_categorical(test_labels_1, 10)
valid_labels = train_labels[50000:60000]
train_labels = train_labels[:50000]
va = [0.11111111111111111111]
vs=[0.0]
best=0.01


     50000 train samples
     10000 validation samples


va = [0.11111111111111111111]
for i in range(len(layers)):
  model = Sequential()
  model.add(Dense( 100, activation='relu',input_shape=(784,)))
  for a in range(layers[i]):
    model.add(Dense(100, activation='relu'))
  model.add(Dense(10, activation='softmax'))
  sgd = SGD(lr=0.01)
  model.compile(loss='categorical_crossentropy',
              optimizer=sgd,
              metrics=['accuracy'])
  print("\n\t_____________________________________________________\n")
  #print("LAYERS::",layers[i])
  model.fit(X_train,train_labels,batch_size=32, verbose=1, epochs=3)
  score = model.evaluate(X_valid, valid_labels, verbose=0)
  print('Validation score:', score[0])
  print('Validation accuracy:', score[1])
  sc=score[1]
  print(sc)
  va.append(sc)
  if(best<sc):
```

```
    best=sc
    print("BEST YET")
```

```
plt.plot(layers[1:], va[2:])
plt.xlabel('LAYERS')
plt.ylabel('ACCURACY')
```

```
Text(0, 0.5, 'ACCURACY')
```

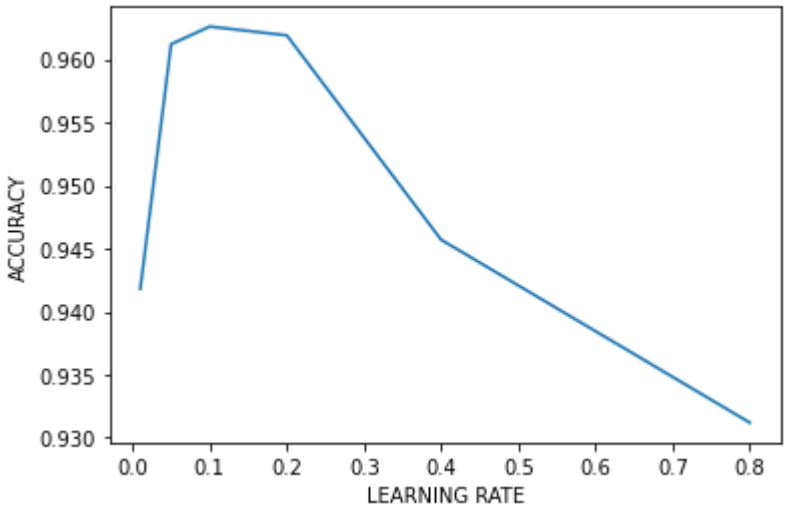



```
va = [0.11111111111111111111]
for j in range(len(learningrate)):
  model = Sequential()
  model.add(Dense(100, activation='relu',input_shape=(784,)))
  model.add(Dense(100, activation='relu'))
  model.add(Dense(100, activation='relu'))
  model.add(Dense(10, activation='softmax'))


  sgd = SGD(lr=learningrate[j])
  model.compile(loss='categorical_crossentropy',
              optimizer=sgd,
              metrics=['accuracy'])
  print("\n\t_____________________________________________________\n")
  print("LEARNING RATE::",learningrate[j])
  model.fit(X_train,train_labels,batch_size=32, verbose=1, epochs=3)   #BATCH SIZE
  score = model.evaluate(X_valid, valid_labels, verbose=0)
  print('Validation score:', score[0])
  print('Validation accuracy:', score[1])
  sc=score[1]
  print(sc)
  va.append(sc)
  if(best<sc):
    best=sc
    print("BEST YET")
```

```
plt.plot(learningrate[1:], va[2:])
```

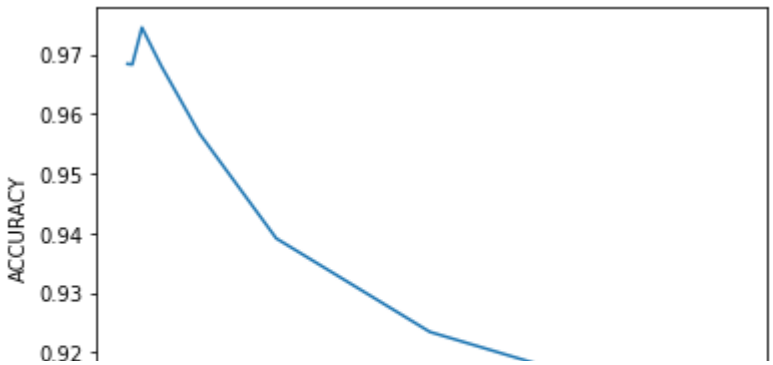
```
plt.xlabel('LEARNING RATE')
plt.ylabel('ACCURACY')
```

```
Text(0, 0.5, 'ACCURACY')
```




```
va = [0.11111111111111111111]
for k in range(len(batchsize)):
  model = Sequential()
  model.add(Dense(100, activation='relu',input_shape=(784,)))
  model.add(Dense(100, activation='relu'))
  model.add(Dense(100, activation='relu'))
  model.add(Dense(10, activation='softmax'))

  sgd = SGD(lr=0.01)
  model.compile(loss='categorical_crossentropy',
              optimizer=sgd,
              metrics=['accuracy'])

  print("\n\t_____________________________________________________\n")
  print("BATCH SIZE::",batchsize[k])
  model.fit(X_train,train_labels,batch_size=batchsize[k], verbose=1, epochs=3)   #BATCH SIZE
  score = model.evaluate(X_valid, valid_labels, verbose=0)
  print('Validation score:', score[0])
  print('Validation accuracy:', score[1])
  sc=score[1]
  print(sc)
  va.append(sc)
  if(best<sc):
    best=sc
    print("BEST YET")



plt.plot(batchsize[1:], va[2:])
plt.xlabel('BATCH SIZE')
plt.ylabel('ACCURACY')
```

```
Text(0, 0.5, 'ACCURACY')
```




```
# This is formatted as code
```

ACCURACY = 96.54% when [LEARNING RATE = 0.01 | BATCH SIZE=04 | Layer=2]

TASK 2 : CONFUSION MATRIX

Confusion matrix is a square matrix with one row and one column for each label. Each cell in the matrix contains the number of instances whose true label is that of row of the cell, but which our classifier predicted as the label corresponding to the column of the cell.Here, we plot a confusion matrix after training out model with the hyperparameters of the network to the values that resulted in the maximum validation accuracy in Task 1.

TASK 2 : QUESTIONS

1. Between which two classes does your model get the most confused? Which one of those is the true label and which one is the prediction?
2. What would you do if you wanted to make fewer misclassifications of one particular class?

```
model = Sequential()
model.add(Dense(100, activation='relu',input_shape=(784,)))
model.add(Dense(100, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(10, activation='softmax'))

sgd = SGD(lr=0.01)
model.compile(loss='categorical_crossentropy',
            optimizer=sgd,
            metrics=['accuracy'])
history=model.fit(X_train,train_labels,batch_size=4, verbose=1, epochs=3)   #BATCH SIZE
score = model.evaluate(X_valid, valid_labels, verbose=0)
print('Validation score:', score[0])
print('Validation accuracy:', score[1])


model.summary()
```

```
Model: "sequential_5"
```

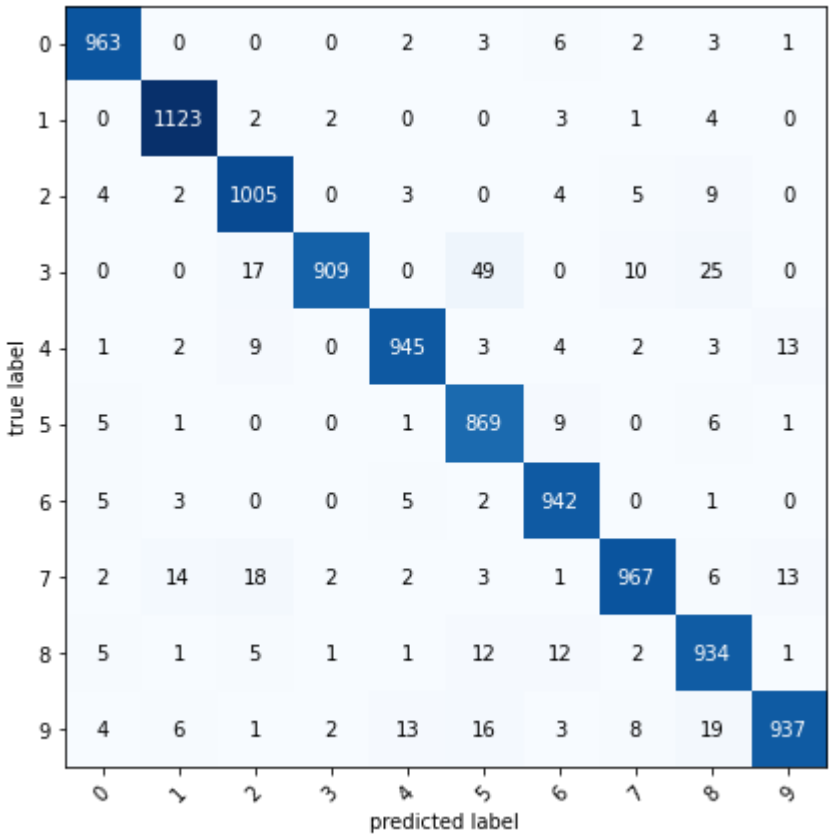
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_20 (Dense)             (None, 100)               78500
_________________________________________________________________
dense_21 (Dense)             (None, 100)               10100
_________________________________________________________________
dense_22 (Dense)             (None, 100)               10100
_________________________________________________________________
dense_23 (Dense)             (None, 10)                1010
=================================================================
Total params: 99,710
Trainable params: 99,710
Non-trainable params: 0
_________________________________________________________________
```

```
from sklearn.metrics import confusion_matrix
```

plot_confusion_matrix code copy from official site

```
y_pred=model.predict_classes(X_test)
class_names=["0","1","2","3","4","5","6","7","8","9"]
mat=confusion_matrix(test_labels_1,y_pred)
plot_confusion_matrix(conf_mat=mat, figsize=(7,7),class_names=class_names)
```

```
(<Figure size 504x504 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x7fc80225feb8>)
```



TASK 3 : OVERFITTING

QUESTIONS

1. How many parameters does your neural network created in the Task 2 have? Show the calculation
2. With so many parameters, would it always be the case that the neural network you have created generalizes well? Find out by training the neural network on just the first 1000 of the 50,000 training examples, for 500 iterations. Write down the loss and the accuracy on the training set as well as the test set for the trained network

```
#splitting dataset
(X_train_1, train_labels_1), (X_test, test_labels_1) = mnist.load_data()

X_train_1 = X_train_1.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
X_train_1 = X_train_1.astype('float32')
X_test = X_test.astype('float32')

X_train_1 /= 255
X_test /= 255

# Divides the dataset into train and validation sets
X_valid = X_train_1[1000:60000]
X_train = X_train_1[:1000]
print(X_train.shape[0], 'train samples')
print(X_valid.shape[0], 'validation samples')

 #newly added


# Preprocess class labels
train_labels = np_utils.to_categorical(train_labels_1, 10)
test_labels = np_utils.to_categorical(test_labels_1, 10)
valid_labels = train_labels[1000:60000]
train_labels = train_labels[:1000]

     1000 train samples
     59000 validation samples



model = Sequential()
model.add(Dense(100, activation='relu',input_shape=(784,)))
model.add(Dense(100, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(10, activation='softmax'))

sgd = SGD(lr=0.01)
model.compile(loss='categorical_crossentropy',
            optimizer=sgd,
            metrics=['accuracy'])
history=model.fit(X_train,train_labels,batch_size=4, verbose=1, epochs=500)   #BATCH SIZE



scorem = model.evaluate(X_train, train_labels, verbose=0)
score = model.evaluate(X_valid, valid_labels, verbose=0)
```

```
print('Training set score:', scorem[0])
print('Training set accuracy:', scorem[1])
print('Validation set score:', score[0])
print('Validation set accuracy:', score[1])
```

```
Training set score: 0.00010053347796201706
Training set accuracy: 1.0
Validation set score: 0.8988227844238281
Validation set accuracy: 0.8670338988304138
```

```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_12 (Dense)             (None, 50)                39250
_________________________________________________________________
dense_13 (Dense)             (None, 50)                2550
_________________________________________________________________
dense_14 (Dense)             (None, 50)                2550
_________________________________________________________________
dense_15 (Dense)             (None, 10)                510
=================================================================
Total params: 44,860
Trainable params: 44,860
Non-trainable params: 0
_________________________________________________________________
```