NEURAL NETWORK & DEEP LEARNING(CS-5720)

(CRN:31196)

ASSIGNMENT -2

Name : Vamsi Krishna Mekala
ID : 700742751
Date : 07/11/2023
Github : https://github.com/vamsi-mekala/Neural-networks-icp-2
Google Drive: https://drive.google.com/file/d/1e0Az26ooIATtz1c7AESxWwFO8AOgeC4W/view?usp=sharing

Question 1:

1. Add more Dense layers to the existing code and check how the accuracy changes.
2. Change the data source to Breast Cancer dataset * available in the source code folder and make required changes. Report accuracy of the model.
3. Normalize the data before feeding the data to the model and check how the normalization change your accuracy

```
In [138]: np.random.seed(155)
          my_first_nn = Sequential() # create model
          my_first_nn.add(Dense(20, input_dim=8, activation='relu')) # hidden layer
          my_first_nn.add(Dense(16, input_dim=8, activation='relu'))
          my_first_nn.add(Dense(8, input_dim=8, activation='relu'))

          my_first_nn.add(Dense(1, activation='sigmoid')) # output layer
          my_first_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
          my_first_nn_fitted = my_first_nn.fit(X_train, Y_train, epochs=100,
                                               initial_epoch=0)
```

```
Epoch 91/100
576/576 [==============================] - 0s 207us/step - loss: 0.4549 - acc: 0.8177
Epoch 92/100
576/576 [==============================] - 0s 58us/step - loss: 0.4671 - acc: 0.7743
Epoch 93/100
576/576 [==============================] - 0s 55us/step - loss: 0.4573 - acc: 0.7882
Epoch 94/100
576/576 [==============================] - 0s 30us/step - loss: 0.4686 - acc: 0.7778
Epoch 95/100
576/576 [==============================] - 0s 30us/step - loss: 0.4550 - acc: 0.7812
Epoch 96/100
576/576 [==============================] - 0s 29us/step - loss: 0.4729 - acc: 0.7656
Epoch 97/100
576/576 [==============================] - 0s 33us/step - loss: 0.4693 - acc: 0.7743
Epoch 98/100
576/576 [==============================] - 0s 50us/step - loss: 0.4623 - acc: 0.7847
Epoch 99/100
576/576 [==============================] - 0s 32us/step - loss: 0.4799 - acc: 0.7865
Epoch 100/100
576/576 [==============================] - 0s 28us/step - loss: 0.4451 - acc: 0.7934
```

In the above screen shot we added 2 more dense layers to existing code and the results are at 79% accuracy.

```
In [146]: np.random.seed(155)
          my_first_nn = Sequential() # create model
          my_first_nn.add(Dense(30, input_dim=30, activation='relu')) # hidden layer

          my_first_nn.add(Dense(1, activation='sigmoid')) # output layer
          my_first_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
          my_first_nn_fitted = my_first_nn.fit(X_train, Y_train, epochs=100,
                                               initial_epoch=0)

Epoch 50/100
426/426 [==============================] - 0s 63us/step - loss: 0.2156 - acc: 0.9202
Epoch 51/100
426/426 [==============================] - 0s 36us/step - loss: 0.1687 - acc: 0.9343
Epoch 52/100
426/426 [==============================] - 0s 45us/step - loss: 0.1756 - acc: 0.9296
Epoch 53/100
426/426 [==============================] - 0s 39us/step - loss: 0.1789 - acc: 0.9366
Epoch 54/100
426/426 [==============================] - 0s 74us/step - loss: 0.1943 - acc: 0.9366
Epoch 55/100
426/426 [==============================] - 0s 40us/step - loss: 0.1755 - acc: 0.9484
Epoch 56/100
426/426 [==============================] - 0s 33us/step - loss: 0.1673 - acc: 0.9390
Epoch 57/100
426/426 [==============================] - 0s 34us/step - loss: 0.2249 - acc: 0.9249
Epoch 58/100
426/426 [==============================] - 0s 30us/step - loss: 0.2676 - acc: 0.9131
Epoch 59/100
426/426 [==============================] - 0s 12us/step - loss: 0.1694 - acc: 0.9272
```

In the above screenshot we can see the model performance for the breast cancer dataset.

```
my_first_nn.add(Dense(30, input_dim=30, activation='relu')) # hidden layer
my_first_nn.add(Dense(1, activation='sigmoid')) # output layer
my_first_nn.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
my_first_nn_fitted = my_first_nn.fit(X_train, Y_train, epochs=100,
                                     initial_epoch=0)

Epoch 91/100
426/426 [==============================] - 0s 33us/step - loss: 0.0172 - acc: 0.9953
Epoch 92/100
426/426 [==============================] - 0s 39us/step - loss: 0.0171 - acc: 0.9953
Epoch 93/100
426/426 [==============================] - 0s 39us/step - loss: 0.0167 - acc: 0.9953
Epoch 94/100
426/426 [==============================] - 0s 39us/step - loss: 0.0171 - acc: 0.9953
Epoch 95/100
426/426 [==============================] - 0s 38us/step - loss: 0.0164 - acc: 0.9953
Epoch 96/100
426/426 [==============================] - 0s 39us/step - loss: 0.0162 - acc: 0.9953
Epoch 97/100
426/426 [==============================] - 0s 39us/step - loss: 0.0159 - acc: 0.9953
Epoch 98/100
426/426 [==============================] - 0s 39us/step - loss: 0.0155 - acc: 0.9953
Epoch 99/100
426/426 [==============================] - 0s 39us/step - loss: 0.0154 - acc: 0.9953
Epoch 100/100
426/426 [==============================] - 0s 36us/step - loss: 0.0151 - acc: 0.9953

In [150]: print(my_first_nn.summary())
          print(my_first_nn.evaluate(X_test, Y_test))

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_72 (Dense)             (None, 30)                930
_________________________________________________________________
dense_73 (Dense)             (None, 1)                 31
=================================================================
Total params: 961
Trainable params: 961
Non-trainable params: 0
_________________________________________________________________
None
143/143 [==============================] - 0s 2ms/step
[0.1936580974322099, 0.9650349654517807]
```

Here is the model performance for the breast cancer dataset after scaling and normalizing the data. And the accuracy went up to 99%

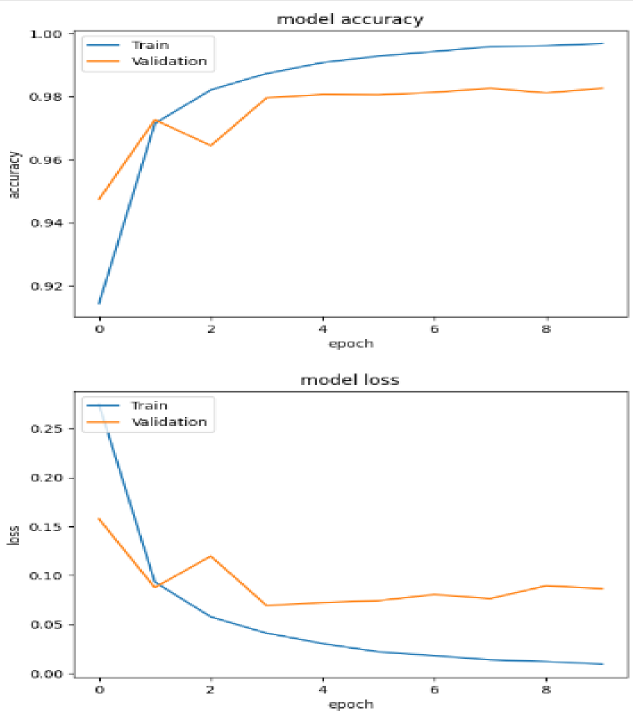
Question 2:

Use Image Classification on the handwritten digits data set (mnist)

1. Plot the loss and accuracy for both training data and validation data using the history object in the source
code.
2. Plot one of the images in the test data, and then do inferencing to check what is the prediction of the model
on that single image.
3. We had used 2 hidden layers and Relu activation. Try to change the number of hidden layer and the
activation to tanh or sigmoid and see what happens.
4. Run the same code without scaling the images and check the performance?

In [165]:
```
import matplotlib.pyplot as plt

# summarize history for accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```



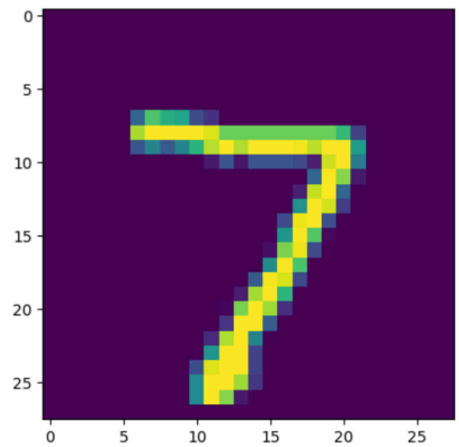
Here is the plot for the loss and accuracy for the test and train data.

Plot one of the images in the test data, and then do inferencing to check what is the prediction of the model on that single image

In [186]:
```
predictions = model.predict(test_data)
```

In [187]:
```
image = np.array(test_images[0]).reshape(28, 28)
plt.imshow(image)
```

Out[187]: <matplotlib.image.AxesImage at 0x23b64c32b08>



Here is the inference of the prediction and the test data that was predicted by the model.

```
print(dimData)
train_data = train_images.reshape(train_images.shape[0],dimData)
test_data = test_images.reshape(test_images.shape[0],dimData)

#convert data to float and scale values between 0 and 1
train_data = train_data.astype('float')
test_data = test_data.astype('float')
#scale data
train_data /=255.0
test_data /=255.0
#change the labels frominteger to one-hot encoding. to_categorical is doing the same thing as LabelEncoder()
train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)

#creating network
model = Sequential()
model.add(Dense(512, activation='tanh', input_shape=(dimData,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(512, activation='sigmoid'))
model.add(Dense(10, activation='softmax'))

model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(train_data, train_labels_one_hot, batch_size=256, epochs=10, verbose=1,
                   validation_data=(test_data, test_labels_one_hot))
```

```
(28, 28)
784
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 9s 150us/step - loss: 0.3345 - acc: 0.8947 - val_loss: 0.1824 - val_acc: 0.9429
Epoch 2/10
60000/60000 [==============================] - 7s 112us/step - loss: 0.1234 - acc: 0.9616 - val_loss: 0.1162 - val_acc: 0.9630
Epoch 3/10
60000/60000 [==============================] - 7s 112us/step - loss: 0.0816 - acc: 0.9747 - val_loss: 0.0900 - val_acc: 0.9712
Epoch 4/10
60000/60000 [==============================] - 7s 117us/step - loss: 0.0626 - acc: 0.9800 - val_loss: 0.0811 - val_acc: 0.9739
Epoch 5/10
60000/60000 [==============================] - 6s 102us/step - loss: 0.0466 - acc: 0.9850 - val_loss: 0.0854 - val_acc: 0.9726
Epoch 6/10
60000/60000 [==============================] - 6s 104us/step - loss: 0.0370 - acc: 0.9874 - val_loss: 0.0904 - val_acc: 0.9743
Epoch 7/10
60000/60000 [==============================] - 6s 101us/step - loss: 0.0293 - acc: 0.9902 - val_loss: 0.1062 - val_acc: 0.9695
Epoch 8/10
60000/60000 [==============================] - 6s 93us/step - loss: 0.0231 - acc: 0.9921 - val_loss: 0.0837 - val_acc: 0.9782
Epoch 9/10
60000/60000 [==============================] - 7s 111us/step - loss: 0.0210 - acc: 0.9928 - val_loss: 0.0870 - val_acc: 0.9764
Epoch 10/10
60000/60000 [==============================] - 6s 100us/step - loss: 0.0173 - acc: 0.9942 - val_loss: 0.0872 - val_acc: 0.9775
```

Now, we added some more dense layers and changed the activation functions from relu to tanh and sigmoid functions and can observe the accuracy dropped from 98 - 97%.

```
In [189]: from keras import Sequential
from keras.datasets import mnist
import numpy as np
from keras.layers import Dense
from keras.utils import to_categorical

(train_images,train_labels),(test_images, test_labels) = mnist.load_data()

print(train_images.shape[1:])

#convert data to float and scale values between 0 and 1
train_data = train_data.astype('float')
test_data = test_data.astype('float')
#scale data
train_data /=255.0
test_data /=255.0
#change the labels frominteger to one-hot encoding. to_categorical is doing the same thing as LabelEncoder()
train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)

#creating network
model = Sequential()
model.add(Dense(512, activation='tanh', input_shape=(dimData,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(512, activation='sigmoid'))
model.add(Dense(10, activation='softmax'))

model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(train_data, train_labels_one_hot, batch_size=256, epochs=10, verbose=1,
                   validation_data=(test_data, test_labels_one_hot))
```

```
(28, 28)
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 9s 156us/step - loss: 2.3203 - acc: 0.0990 - val_loss: 2.3290 - val_acc: 0.0958
Epoch 2/10
60000/60000 [==============================] - 7s 116us/step - loss: 2.3172 - acc: 0.1022 - val_loss: 2.3233 - val_acc: 0.1010
Epoch 3/10
60000/60000 [==============================] - 6s 102us/step - loss: 2.3173 - acc: 0.1013 - val_loss: 2.3281 - val_acc: 0.1028
Epoch 4/10
60000/60000 [==============================] - 6s 107us/step - loss: 2.3178 - acc: 0.1003 - val_loss: 2.3402 - val_acc: 0.1135
Epoch 5/10
60000/60000 [==============================] - 6s 99us/step - loss: 2.3174 - acc: 0.1006 - val_loss: 2.3226 - val_acc: 0.1032
Epoch 6/10
60000/60000 [==============================] - 6s 98us/step - loss: 2.3173 - acc: 0.1019 - val_loss: 2.3671 - val_acc: 0.1032
Epoch 7/10
60000/60000 [==============================] - 6s 93us/step - loss: 2.3165 - acc: 0.1027 - val_loss: 2.3301 - val_acc: 0.0958
Epoch 8/10
60000/60000 [==============================] - 5s 90us/step - loss: 2.3168 - acc: 0.1024 - val_loss: 2.3246 - val_acc: 0.0892
Epoch 9/10
60000/60000 [==============================] - 6s 99us/step - loss: 2.3162 - acc: 0.1032 - val_loss: 2.3334 - val_acc: 0.1010
Epoch 10/10
60000/60000 [==============================] - 6s 97us/step - loss: 2.3174 - acc: 0.1025 - val_loss: 2.3149 - val_acc: 0.0958
```

Now the same code is executed without scaling the images and now the accuracy is much dropped to 95%, so preprocessing and scaling all the images to the uniform size makes the model more accurate in training.