İZMİR KÂTİP ÇELEBİ UNIVERSITY

2010

INTRODUCTION TO MACHINE LEARNING

Homework 3

Binary Digit Classification with Multilayer Perceptron

Halime Özge KABAK

180403001

Step by Step Explanation Through Code

- First, I imported the necessary libraries.

```
#Halime Özge KABAK 180403001
#Machine Learning Homework 3
#Binary Digit Classification with Multilayer Perceptron



#importing libraries
import itertools
from keras import Sequential
from keras.datasets import mnist
import cv2
import numpy as np
from keras.layers import Dense
from keras.utils import np_utils
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
import sklearn.metrics as skmet
from sklearn.metrics import classification_report
```

- In the second step, I loaded the mnist dataset and obtained the necessary data for training and testing. In the incoming data, the size of each picture was 28x28, but the assignment required a 7x7 picture, so I resized each picture in the for loop and created a new matrix with these values.

```
(X_train, y_train), (X_test, y_test) = mnist.load_data() #load mnist train and test dataset
#print(X_train[1,:,:])
new_X=np.zeros([60000,7,7]) #create an empty array with  X_train.shape[0] and (7,7)
for i in range(X_train.shape[0]):
    new_X[i,:,:]=cv2.resize(X_train[i,:,:],(7,7)) #resize train dataset from 28x28 to 7x7
new_X2=np.zeros([10000,7,7]) #create an empty array with  X_test.shape[0] and (7,7)
for i in range(X_test.shape[0]):
    new_X2[i,:,:]=cv2.resize(X_test[i,:,:],(7,7)) #resize test dataset from 28x28 to 7x7
```

- In the next step, I extracted 0 and 1 from the dataset, since it was requested to be classified using only 0 and 1 values in the assignment.

```
train_filter = np.where((y_train == 0 ) | (y_train == 1)) # select 0 and 1 labels from y_train
X_train, Y_train = new_X[train_filter], y_train[train_filter] # train dataset which only have 0 and 1 digits
print(f'Images (x) Shape : {X_train.shape}')
print(f'Target (y) Shape : {Y_train.shape}')
test_filter = np.where((y_test == 0 ) | (y_test == 1)) # select 0 and 1 labels from y_test
X_test, Y_test = new_X2[test_filter], y_test[test_filter] # test dataset which only have 0 and 1 digits
print(f'Images (x) Shape : {X_test.shape}')
print(f'Target (y) Shape : {Y_test.shape}')
```

OUTPUT: In total, 12665 training data and 2115 test data were obtained.

```
Images (x) Shape : (12665, 7, 7)
Target (y) Shape : (12665,)
Images (x) Shape : (2115, 7, 7)
Target (y) Shape : (2115,)
```

- In this part, I plotted some of the images in the dataset and showed their pixel values.

```
#plotting an image from dataset
img = X_train[80].reshape(7, 7)

fig = plt.figure(figsize = (12,12))
ax = fig.add_subplot(111)
ax.imshow(img, cmap='gray')
width, height = img.shape
thresh = img.max()/2.5
for x in range(width):
    for y in range(height):
        val = round(img[x][y],2) if img[x][y] !=0 else 0
        ax.annotate(str(val), xy=(y,x),
                    horizontalalignment='center',
                    verticalalignment='center',
                    color='white' if img[x][y]<thresh else 'black')

plt.show()
```

- Before giving the data as input to the model, the matrices in 3d were converted to 2d, then the inputs were converted to float. Inputs converted to float state were normalized by dividing by 255. The reason for dividing by 255 is that the pixel values change between 0 and 255. Finally, twenty percent of the test dataset was set as validation dataset.

```
X_train=X_train.reshape(X_train.shape[0],X_train.shape[1]*X_train.shape[2]) #reshape X_train from 3d to 2d
X_test=X_test.reshape(X_test.shape[0],X_test.shape[1]*X_test.shape[2]) #reshape X_test from 3d to 2d
X_train = X_train.astype('float32') # change X_train type to float
X_test = X_test.astype('float32') # change X_test type to float
X_train = X_train/255 #normalization of train dataset
X_test = X_test/255 #normalization of test dataset
val_x, test_x, val_y, test_y = train_test_split(X_test, Y_test, test_size=0.8, stratify=Y_test) #splitting test dataset as 20% validation and 80% test dataset
print(f'Images (x) Shape : {val_x.shape}')
print(f'Target (y) Shape : {val_y.shape}')
print(f'Images (x) Shape : {test_x.shape}')
print(f'Target (y) Shape : {test_y.shape}')
```

OUTPUT:

```
Images (x) Shape : (423, 49)
Target (y) Shape : (423,)
Images (x) Shape : (1692, 49)
Target (y) Shape : (1692,)
```

- In the next step, the y-array containing the labels of the dataset was one hot encoded.

```
Y_train = np_utils.to_categorical(Y_train, 2)  #one hot incoding for Y_train
Y_test = np_utils.to_categorical(test_y, 2)    #one hot incoding for test_y
Y_val = np_utils.to_categorical(val_y, 2)      #one hot incoding for val_y
```

- At this stage, the model was built. The model consists of 3 layers and there are 49, 16 and 2 hidden units in each layer, respectively. Softmax and relu were used as the activation functions, and the model was trained with 50 epochs, but the training ends early when the validation accuracy does not change thanks to early stopping callback.

```
num_classes = 2
epochs = 50
batch_size = 32

#building the model
model = Sequential()
model.add(Dense(49, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(16, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

model.compile(loss='categorical_crossentropy',
              optimizer='Adam',
              metrics=['accuracy'])
callbacks = [

        EarlyStopping(monitor='val_loss', patience=20)
        #ModelCheckpoint(filepath=save_fname, monitor='val_loss', save_best_only=True)
        ]

history = model.fit(X_train, Y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(val_x, Y_val),callbacks=callbacks)
```

OUTPUT:

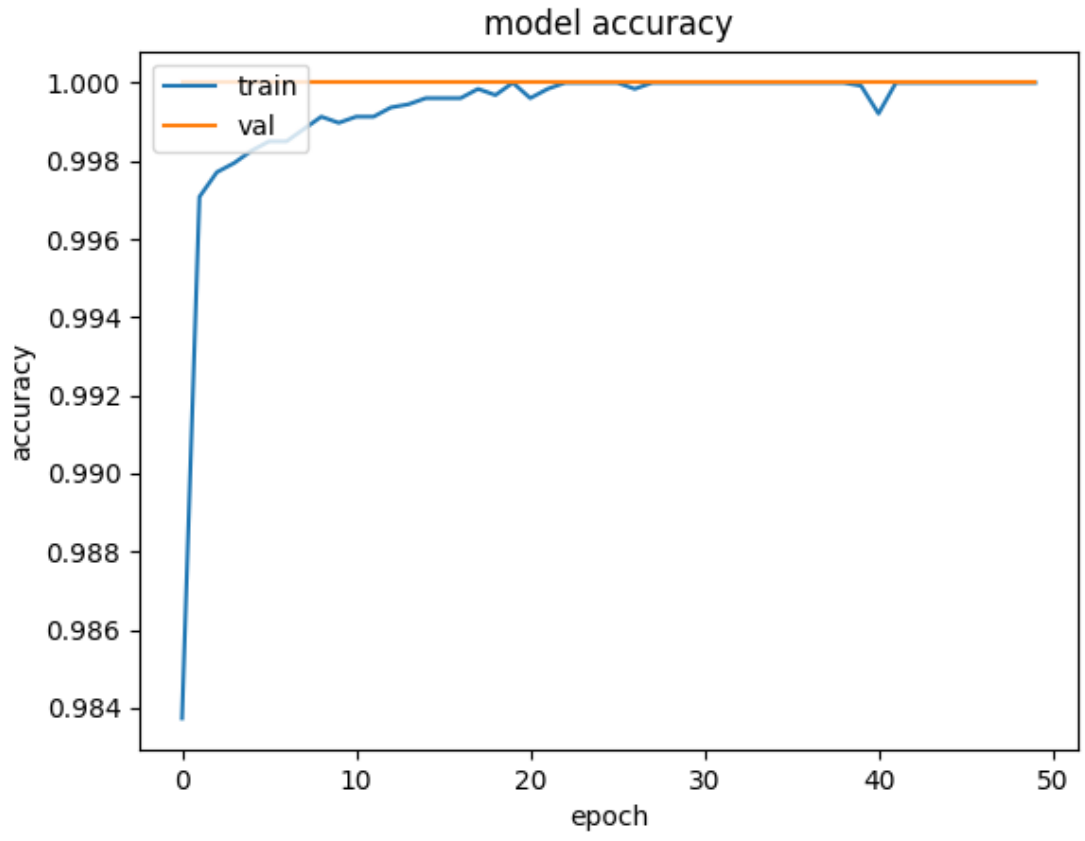
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 49)                2450

 dense_1 (Dense)             (None, 16)                800

 dense_2 (Dense)             (None, 2)                 34

=================================================================
Total params: 3,284
Trainable params: 3,284
Non-trainable params: 0
_________________________________________________________________
```

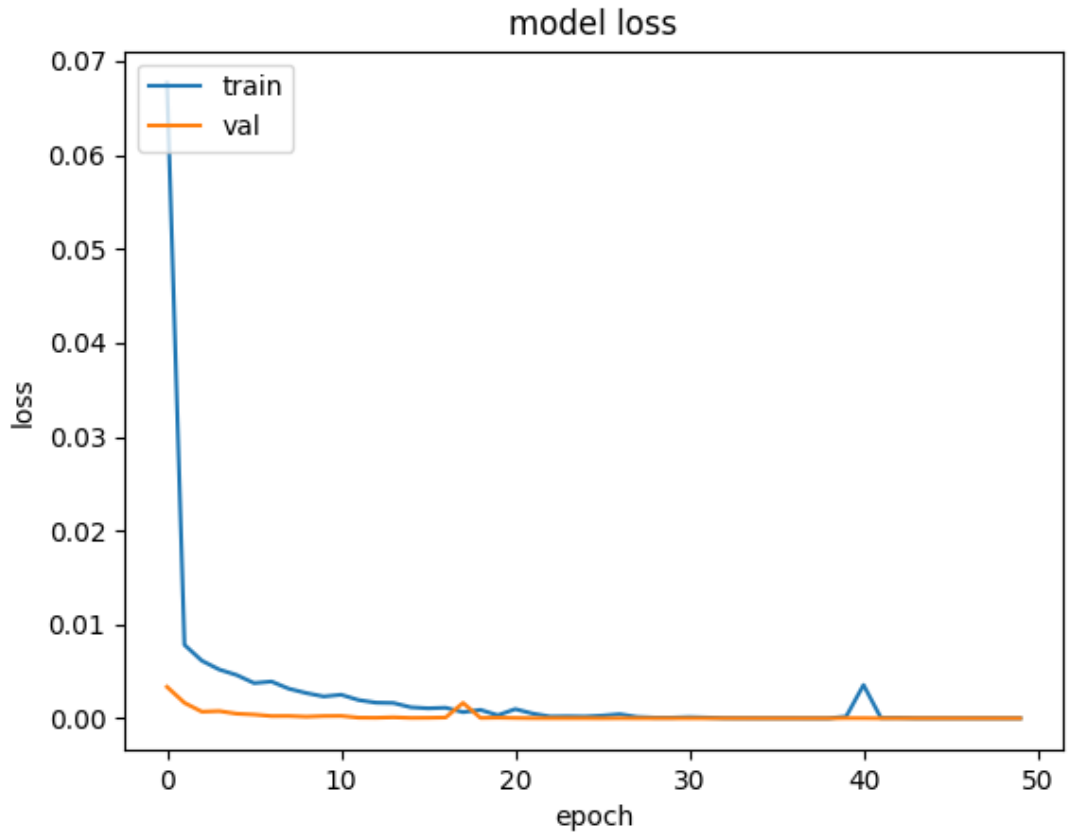
```
Epoch 9/50
396/396 [==============================] - 1s 2ms/step - loss: 0.0034 - accuracy: 0.9985 - val_loss: 0.0011 - val_accuracy: 1.0000
Epoch 10/50
396/396 [==============================] - 1s 1ms/step - loss: 0.0024 - accuracy: 0.9991 - val_loss: 0.0041 - val_accuracy: 0.9953
Epoch 11/50
396/396 [==============================] - 1s 1ms/step - loss: 0.0021 - accuracy: 0.9991 - val_loss: 9.8157e-04 - val_accuracy: 1.0000
Epoch 12/50
396/396 [==============================] - 1s 1ms/step - loss: 0.0022 - accuracy: 0.9991 - val_loss: 0.0014 - val_accuracy: 1.0000
Epoch 13/50
396/396 [==============================] - 1s 1ms/step - loss: 0.0019 - accuracy: 0.9994 - val_loss: 3.5481e-04 - val_accuracy: 1.0000
Epoch 14/50
396/396 [==============================] - 1s 1ms/step - loss: 0.0015 - accuracy: 0.9996 - val_loss: 7.2676e-04 - val_accuracy: 1.0000
Epoch 15/50
396/396 [==============================] - 1s 1ms/step - loss: 0.0012 - accuracy: 0.9996 - val_loss: 5.0760e-04 - val_accuracy: 1.0000
Epoch 16/50
 45/396 [==>...........................] - ETA: 0s - loss: 0.0029 - accuracy: 0.9986
```

- After completing the model training, the accuracy and loss graphs obtained during the training were plotted.

```
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], 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', 'val'], loc='upper left')
plt.show()
```

OUTPUT:





- After the model was evaluated with the test dataset, the loss and accuracy values were printed.

```
#printing loss and accuracy scores according to test dataset
score = model.evaluate(test_x, Y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

OUTPUT: According to this result, the accuracy is 99.94% and the loss is 0.65%.

```
Test loss: 0.006486227270215 75
Test accuracy: 0.9994089603424072
```

- Then the model was tested with the predict command and the result was decoded and made into a list. The classification report and confusion matrix, which measure the performance of the

model according to the predicted result of the model and the test dataset, was printed.

```
Y_pred=model.predict(test_x) #testing the model
y_pred=[Y_pred.argmax(1)] #one hot decodin for prediction results
y_pred=y_pred[0].tolist() # convert numpy array to list
test_y=test_y.tolist() # convert numpy array to list

print(classification_report(test_y, y_pred)) #printing classification report according to prediction results
cm = skmet.confusion_matrix(y_true=test_y, y_pred=y_pred) #confusion matrix
print(cm)
```

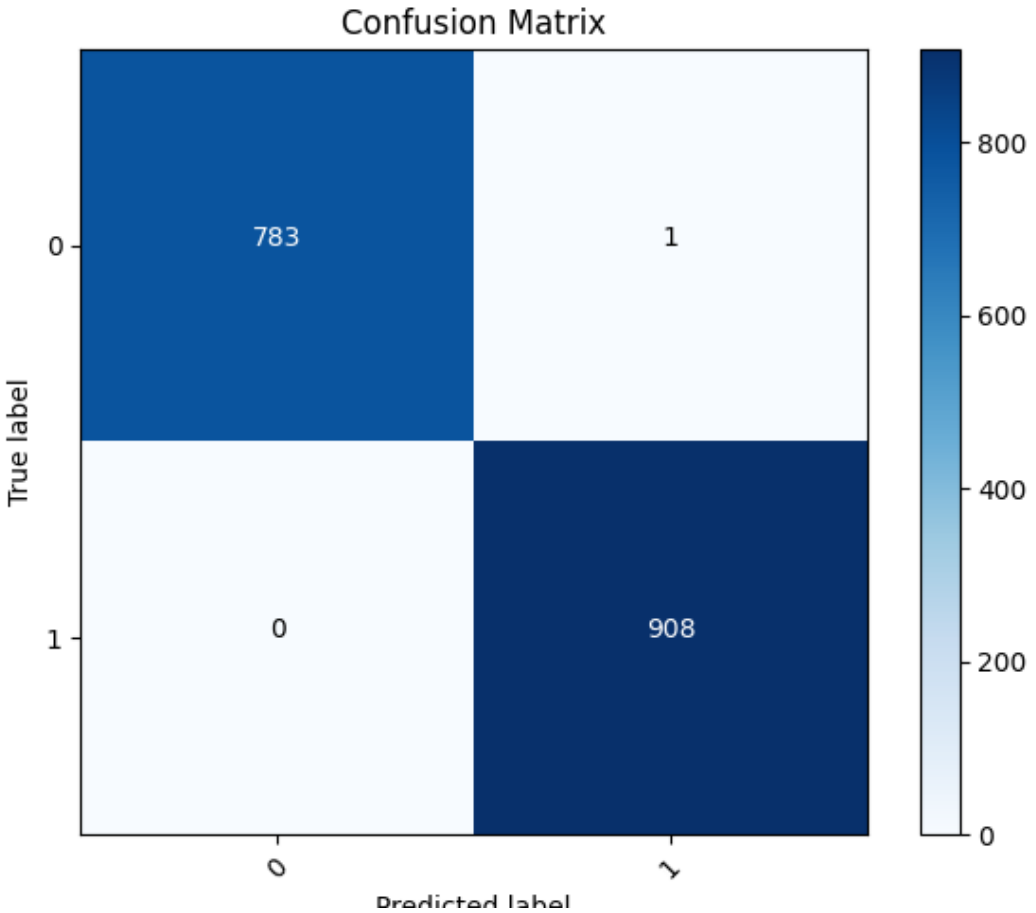
OUTPUT:

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       784
           1       1.00      1.00      1.00       908

    accuracy                           1.00      1692
   macro avg       1.00      1.00      1.00      1692
weighted avg       1.00      1.00      1.00      1692

[[783   1]
 [  0 908]]
```

- Confusion matrix plotted with the help of the confusion matrix plotting function, which was also written as plot_confusion_matrix.

```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

#plot confusion matrix
thresh = cm.max() / 2.
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
            horizontalalignment="center",
            color="white" if cm[i, j] > thresh else "black")
plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
cm_plot_labels = ['0','1']
plot_confusion_matrix(cm=cm, classes=cm_plot_labels, title='Confusion Matrix')
plt.show()
```

OUTPUT: According to this result, it was seen that 783 of the 784 data, which were actually 0, were classified correctly, and all 908 data, which were actually 1, were correctly classified, that is, the performance of the model was quite high.



- Finally, the results obtained according to the confusion matrix were printed and the model was saved in h5 format.

```
#print number of correct and incorrect predictions according to confusion matrix
print("\033[1m The result is telling us that we have: ",(cm[0,0]+cm[1,1]),"correct predictions.")
print("\033[1m The result is telling us that we have: ",(cm.sum()-(cm[0,0]+cm[1,1])),"incorrect predictions.")
print("\033[1m We have total predictions of:",(cm.sum()))

model.save("model_classifier.h5") #saving model as h5 file
```

OUTPUT:

```
The result is telling us that we have:  1691 correct predictions.
The result is telling us that we have:  1 incorrect predictions.
We have total predictions of: 1692
```

Architecture of the Saved Model