[1]:
```
# Loading Libraries
```

[2]:
```
import numpy as np
import idx2numpy
import time as tm
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.utils import to_categorical
```

[3]:
```
# Creating function to load EMNIST data
def importer(location, dataset):
    x_train = idx2numpy.convert_from_file(location + '/emnist-' + dataset +␣
 ↪'-train-images-idx3-ubyte')
    y_train = idx2numpy.convert_from_file(location + '/emnist-' + dataset +␣
 ↪'-train-labels-idx1-ubyte')
    x_test = idx2numpy.convert_from_file(location + '/emnist-' + dataset +␣
 ↪'-test-images-idx3-ubyte')
    y_test = idx2numpy.convert_from_file(location + '/emnist-' + dataset +␣
 ↪'-test-labels-idx1-ubyte')
    x_train, x_test = x_train / 255.0, x_test / 255.0
    break_point = (x_train.shape[0] - x_test.shape[0])
    x_val = x_train[break_point:]
    x_train = x_train[:break_point]
    y_val = y_train[break_point:]
    y_train = y_train[:break_point]

    return x_train, y_train, x_val, y_val, x_test, y_test
```

```
loc = "/Users/7mza/Desktop/EMNIST Data"
x_train, y_train, x_val, y_val, x_test, y_test = importer(loc, 'digits')
```

[4]:
```
# Modifying variables
x_train = x_train.reshape(200000, 28*28)
x_train = x_train.astype('float32')

x_test = x_test.reshape(40000, 28*28)
x_test = x_test.astype('float32')

x_val = x_val.reshape(40000, 28*28)
x_val = x_val.astype('float32')

y_train = to_categorical(y_train)
y_val = to_categorical(y_val)
y_test = to_categorical(y_test)
```

[5]:
```
# Generating the input to hidden layer weights
input_length = x_train.shape[1]
hidden_units = 2000

Win = np.random.normal(size = [input_length, hidden_units])
print('Input weight shape: {shape}'.format(shape = Win.shape))
```

```
Input weight shape: (784, 2000)
```

[6]:
```
# Computing the hidden layer to output weights
def input_to_hidden(x):
    a = np.dot(x, Win)
    a = np.maximum(a,0,a)
    return a
```

[7]:
```
# Attempting to minimize the least square error between the predicted labels␣
↪and the training labels
X = input_to_hidden(x_train)
Xt = np.transpose(X)
Wout = np.dot(np.linalg.inv(np.dot(Xt,X)), np.dot(Xt, y_train))
print('Output weights shape: {shape}'.format(shape = Wout.shape))
```

```
Output weights shape: (2000, 10)
```

[8]:
```
# Creating function to predict the output
def predict(x):
    x = input_to_hidden(x)
    y = np.dot(x, Wout)
    return y
```

[9]:
```
# Testing the model
y = predict(x_test)
correct = 0
total = y.shape[0]
ELMt = tm.time()
for i in range(total):
    predicted = np.argmax(y[i])
    test = np.argmax(y_test[i])
    correct = correct + (1 if predicted == test else 0)
ELMt2 = tm.time() - ELMt
print('Time taken: ', round(ELMt2, 2) , 'seconds')
print('Accuracy: ', round((correct/total)*100, 2), "%")
```

```
Time taken:  0.14 seconds
Accuracy:  96.75 %
```

1.4 MLP

[10]:
```
# Importing Libraries
import tensorflow as tf
import numpy as np
import idx2numpy
import time as tm
import keras
from keras import models
from keras import layers
from tensorflow.keras.utils import to_categorical
from matplotlib import pyplot as plt
```

[11]:
```
# Loading dataset and seperating into train and test variables
def importer(location, dataset):
    x_train = idx2numpy.convert_from_file(location + '/emnist-' + dataset +␣
 ↪'-train-images-idx3-ubyte')
    y_train = idx2numpy.convert_from_file(location + '/emnist-' + dataset +␣
 ↪'-train-labels-idx1-ubyte')
    x_test = idx2numpy.convert_from_file(location + '/emnist-' + dataset +␣
 ↪'-test-images-idx3-ubyte')
    y_test = idx2numpy.convert_from_file(location + '/emnist-' + dataset +␣
 ↪'-test-labels-idx1-ubyte')
    x_train, x_test = x_train / 255, x_test / 255
    break_point = (x_train.shape[0] - x_test.shape[0])
    x_val = x_train[break_point:]
    x_train = x_train[:break_point]
    y_val = y_train[break_point:]
    y_train = y_train[:break_point]

    return x_train, y_train, x_val, y_val, x_test, y_test
```

```
loc = "/Users/7mza/Desktop/EMNIST Data"
x_train, y_train, x_val, y_val, x_test, y_test = importer(loc, 'digits')
print("Training data sizes: ", x_train.shape, y_train.shape)
print("Validation data sizes: ", x_val.shape, y_val.shape)
print("Holdout/test data sizes: ", x_test.shape, y_test.shape)
```

```
Training data sizes:  (200000, 28, 28) (200000,)
Validation data sizes:  (40000, 28, 28) (40000,)
Holdout/test data sizes:  (40000, 28, 28) (40000,)
```

[12]:
```
# Modifying variables
x_train = x_train.reshape(200000, 28*28)
x_train = x_train.astype('float32')

x_test = x_test.reshape(40000, 28*28)
x_test = x_test.astype('float32')

x_val = x_val.reshape(40000, 28*28)
x_val = x_val.astype('float32')

y_train = to_categorical(y_train)
y_val = to_categorical(y_val)
y_test = to_categorical(y_test)
```

[13]:
```
# Creating model
network = models.Sequential()
network.add(layers.Dense(16, activation = 'relu', input_shape = (28 * 28,)))
network.add(layers.Dense(10, activation = 'softmax'))

network.compile(optimizer = 'rmsprop',
                loss = 'categorical_crossentropy',
                metrics = ['accuracy'])
```

```
2022-06-19 22:41:05.819334: I tensorflow/core/platform/cpu_feature_guard.cc:193]
This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical
operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate
compiler flags.
```

[14]:
```
# Running the model
MLPT = tm.time()
TrainingHistory = network.fit(x_train, y_train, epochs = 10, batch_size = 32,␣
 ↪validation_data = (x_val, y_val), verbose = 1)
network.evaluate(x_test, y_test)
```

```
history_dict = TrainingHistory.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']

epochs = range(1, len(loss_values) + 1)
MLPTD = tm.time() - MLPT
```

```
Epoch 1/10
6250/6250 [==============================] - 8s 1ms/step - loss: 0.2440 -
accuracy: 0.9341 - val_loss: 0.1788 - val_accuracy: 0.9497
Epoch 2/10
6250/6250 [==============================] - 8s 1ms/step - loss: 0.1682 -
accuracy: 0.9552 - val_loss: 0.1553 - val_accuracy: 0.9574
Epoch 3/10
6250/6250 [==============================] - 8s 1ms/step - loss: 0.1495 -
accuracy: 0.9605 - val_loss: 0.1451 - val_accuracy: 0.9610
Epoch 4/10
6250/6250 [==============================] - 8s 1ms/step - loss: 0.1409 -
accuracy: 0.9628 - val_loss: 0.1423 - val_accuracy: 0.9610
Epoch 5/10
6250/6250 [==============================] - 8s 1ms/step - loss: 0.1362 -
accuracy: 0.9640 - val_loss: 0.1439 - val_accuracy: 0.9622
Epoch 6/10
6250/6250 [==============================] - 9s 1ms/step - loss: 0.1329 -
accuracy: 0.9653 - val_loss: 0.1396 - val_accuracy: 0.9635
Epoch 7/10
6250/6250 [==============================] - 7s 1ms/step - loss: 0.1299 -
accuracy: 0.9664 - val_loss: 0.1356 - val_accuracy: 0.9643
Epoch 8/10
6250/6250 [==============================] - 7s 1ms/step - loss: 0.1280 -
accuracy: 0.9670 - val_loss: 0.1385 - val_accuracy: 0.9633
Epoch 9/10
6250/6250 [==============================] - 7s 1ms/step - loss: 0.1266 -
accuracy: 0.9671 - val_loss: 0.1351 - val_accuracy: 0.9642
Epoch 10/10
6250/6250 [==============================] - 9s 1ms/step - loss: 0.1258 -
accuracy: 0.9674 - val_loss: 0.1334 - val_accuracy: 0.9654
1250/1250 [==============================] - 1s 876us/step - loss: 0.1323 -
accuracy: 0.9665
```

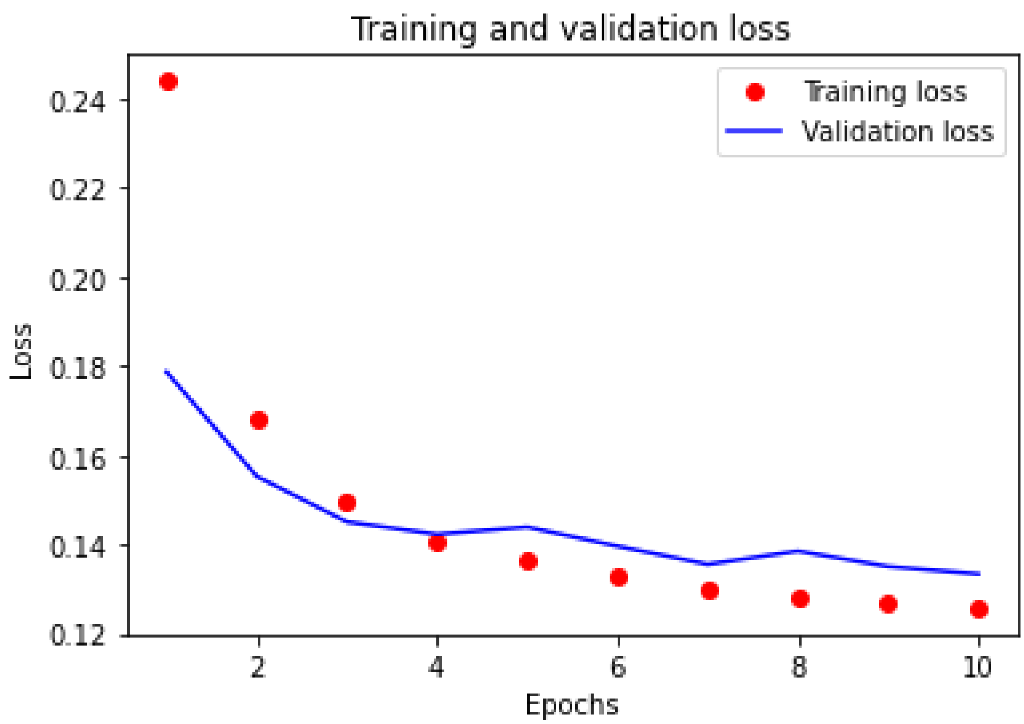
[15]:
```
# Model Accuracy
train_loss, train_acc = network.evaluate(x_train, y_train)
print('Train data accuracy: ', round(train_acc*100, 2), '%')
print('Total time to test train data: ', round(MLPTD, 2), 'Seconds')
```

```
6250/6250 [==============================] - 5s 857us/step - loss: 0.1173 -
accuracy: 0.9694
Train data accuracy:  96.94 %
```

```
Total time to test train data:  79.7 Seconds
```

[16]:
```python
# Graphing model
plt.plot(epochs, loss_values, 'ro', label = 'Training loss')
plt.plot(epochs, val_loss_values, 'b', label = 'Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.ylim
plt.show
```

[16]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



[17]:
```python
# Creating overfit model
network1 = models.Sequential()
network1.add(layers.Dense(32, activation = 'relu', input_shape = (28 * 28,)))
network1.add(layers.Dense(10.9, activation = 'softmax'))


network1.compile(optimizer = 'rmsprop',
                loss = 'categorical_crossentropy',
                metrics = ['accuracy'])
```

```
TrainingHistory1 = network1.fit(x_train, y_train, epochs = 10, batch_size = 32,␣
  ↪validation_data = (x_val, y_val), verbose = 1)
network1.evaluate(x_test, y_test)

history_dict = TrainingHistory1.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']

epochs = range(1, len(loss_values) + 1)
```

```
Epoch 1/10
6250/6250 [==============================] - 10s 1ms/step - loss: 0.1836 -
accuracy: 0.9490 - val_loss: 0.1216 - val_accuracy: 0.9659
Epoch 2/10
6250/6250 [==============================] - 9s 1ms/step - loss: 0.1069 -
accuracy: 0.9705 - val_loss: 0.1014 - val_accuracy: 0.9720
Epoch 3/10
6250/6250 [==============================] - 9s 1ms/step - loss: 0.0946 -
accuracy: 0.9743 - val_loss: 0.1012 - val_accuracy: 0.9729
Epoch 4/10
6250/6250 [==============================] - 9s 1ms/step - loss: 0.0896 -
accuracy: 0.9761 - val_loss: 0.0941 - val_accuracy: 0.9752
Epoch 5/10
6250/6250 [==============================] - 9s 1ms/step - loss: 0.0864 -
accuracy: 0.9772 - val_loss: 0.0982 - val_accuracy: 0.9746
Epoch 6/10
6250/6250 [==============================] - 9s 1ms/step - loss: 0.0851 -
accuracy: 0.9782 - val_loss: 0.0987 - val_accuracy: 0.9747
Epoch 7/10
6250/6250 [==============================] - 9s 1ms/step - loss: 0.0834 -
accuracy: 0.9786 - val_loss: 0.1007 - val_accuracy: 0.9749
Epoch 8/10
6250/6250 [==============================] - 9s 1ms/step - loss: 0.0825 -
accuracy: 0.9793 - val_loss: 0.0991 - val_accuracy: 0.9763
Epoch 9/10
6250/6250 [==============================] - 10s 2ms/step - loss: 0.0814 -
accuracy: 0.9798 - val_loss: 0.1018 - val_accuracy: 0.9766
Epoch 10/10
6250/6250 [==============================] - 11s 2ms/step - loss: 0.0815 -
accuracy: 0.9803 - val_loss: 0.1062 - val_accuracy: 0.9760
1250/1250 [==============================] - 1s 991us/step - loss: 0.0977 -
accuracy: 0.9769
```

[18]:
```
# Overfit model accuracy
over_train_loss, over_train_acc = network1.evaluate(x_train, y_train)
print('Overfit train data accuracy: ', round(train_acc*100, 2), '%')
```

```
6250/6250 [==============================] - 6s 955us/step - loss: 0.0772 -
accuracy: 0.9813
Overfit train data accuracy:  96.94 %
```

[19]:

```
# Modelling with test data (without overfit)
Test_tm = tm.time()
test_loss, test_acc = network.evaluate(x_test, y_test)
Test_tm2 = tm.time() - Test_tm
print('Total time to run model over test data: ', round(Test_tm2, 2), 'Seconds')
print('Test data accuracy: ', round(test_acc*100, 2), '%')
```

```
1250/1250 [==============================] - 1s 823us/step - loss: 0.1323 -
accuracy: 0.9665
Total time to run model over test data:  1.08 Seconds
Test data accuracy:  96.65 %
```