Exercise 5

5.1 Problem Statement:

Build a neural network that will read the image of a digit and correctly identify the number.

5.2 Description of Machine Learning Algorithm:

Backpropagation is one of the important concepts of a neural network. Our task is to classify our data best. For this, we have to update the weights of parameter and bias, but how can we do that in a deep neural network? In the linear regression model, we use gradient descent to optimize the parameter. Similarly here we also use gradient descent algorithm using Backpropagation.

Classification using Multilayer Perceptron Networks and Backpropagation Algorithm

A simple neural network, with three layers is being constructured

(1) An input layer, with 64 nodes, one node per pixel in the input images. Nodes are neurons that actually do nothing. They just take their input value and send it to the neurons of the next layer.

(2) Two hidden layers with 128 and 64 neurons respectively. We could choose a different number, and also add more hidden layers with different numbers of neurons.

(3) An output layer with 10 neurons corresponding to our 10 classes of digits, from 0 to 9. This is a fully connected neural network, which means that each node in each layer is connected to all nodes in the previous and next layers.

To apply a classifier on this data, we need to flatten the images, turning each 2-D array of grayscale values from shape (8, 8) into shape (64,). Subsequently, the entire dataset will be of shape (n_samples, n_features), where n_samples is the number of images and n_features is the total number of pixels in each image.

5.3 Description of Data Set:

Title of the data set: Digits Dataset

The digits dataset consists of 8x8 pixel images of digits. The images attribute of the dataset stores 8x8 arrays of grayscale values for each image. The target attribute of the dataset stores the digit each image represents.

5.4 Data Preprocessing and Exploratory Data Analysis:

Data preprocessing is the process of transforming raw data into an understandable format. It is also an important step in data mining as we cannot work with raw data. The quality of the data should be checked before applying machine learning or data mining algorithms.

Major Tasks in Data Preprocessing:

1. Data cleaning
2. Data integration
3. Data reduction
4. Data transformation

Exploratory data analysis (EDA) is a technique that data professionals can use to understand a dataset before they start to model it. Some people refer to EDA as data exploration. The goal of conducting EDA is to determine the characteristics of the dataset. Conducting EDA can help data analysts make predictions and assumptions about data. Often, EDA involves data visualization, including creating graphs like histograms, scatter plots and box plots.

Major Tasks in EDA:

1. Observe your dataset
2. Find any missing values
3. Categorize your values
4. Find the shape of your dataset
5. Identify relationships in your dataset
6. Locate any outliers in your dataset

5.5 Machine Learning Package Used for Model building:

For classification of model we use scikit-learn

Scikit-learn is an open source Machine Learning Python package that offers functionality supporting supervised and unsupervised learning. Additionally, it provides tools for model development, selection and evaluation as well as many other utilities including data pre-processing functionality.

More specifically, scikit-learn's main functionality includes classification, regression, clustering, dimensionality reduction, model selection and pre-processing. sThe library is very simple to use and most importantly efficient as it is built on **NumPy**, **SciPy** and **matplotlib.**

Neural networks are a machine learning method inspired by how the human brain works. They are particularly good at doing pattern recognition and classification tasks, often using images as inputs. They're a well established machine learning technique that has been around since the 1950s but have gone through several iterations since that have overcome fundamental limitations of the previous one. The current state of the art neural networks are often referred to as deep learning.

5.6 Implementation:

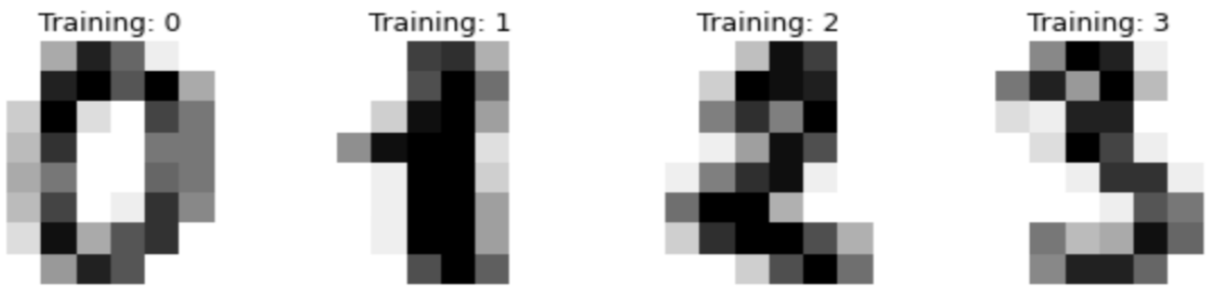
Recognizing hand-written digits -- The activity is to learn from training dataset of images of hand-written digits from 0-9.

```
# Importing datasets
from sklearn import datasets

# loading digit image dataset
digits = datasets.load_digits()
# The dir() function returns all properties and methods
#                          of the specified object, without the values.
dir(digits)

['DESCR', 'data', 'feature_names', 'frame', 'images', 'target', 'target_names']



# Let us describe the dataset
print(digits.DESCR)
.. _digits_dataset:
```

```
Optical recognition of handwritten digits dataset
--------------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 1797
    :Number of Attributes: 64
    :Attribute Information: 8x8 image of integer pixels in the range 0..16.
    :Missing Attribute Values: None
    :Creator: E. Alpaydin (alpaydin '@' boun.edu.tr)
    :Date: July; 1998

This is a copy of the test set of the UCI ML hand-written digits datasets
https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digi
ts

The data set contains images of hand-written digits: 10 classes where
each class refers to a digit.

Preprocessing programs made available by NIST were used to extract
normalized bitmaps of handwritten digits from a preprinted form. From a
total of 43 people, 30 contributed to the training set and different 13
to the test set. 32x32 bitmaps are divided into nonoverlapping blocks of
4x4 and the number of on pixels are counted in each block. This generates
an input matrix of 8x8 where each element is an integer in the range
0..16. This reduces dimensionality and gives invariance to small
distortions.

For info on NIST preprocessing routines, see M. D. Garris, J. L. Blue, G.
T. Candela, D. L. Dimmick, J. Geist, P. J. Grother, S. A. Janet, and C.
L. Wilson, NIST Form-Based Handprint Recognition System, NISTIR 5469,
1994.

.. topic:: References

  - C. Kaynak (1995) Methods of Combining Multiple Classifiers and Their
    Applications to Handwritten Digit Recognition, MSc Thesis, Institute of
    Graduate Studies in Science and Engineering, Bogazici University.
  - E. Alpaydin, C. Kaynak (1998) Cascading Classifiers, Kybernetika.
  - Ken Tang and Ponnuthurai N. Suganthan and Xi Yao and A. Kai Qin.
    Linear dimensionalityreduction using relevance weighted LDA. School of
    Electrical and Electronic Engineering Nanyang Technological University.
    2005.
  - Claudio Gentile. A New Approximate Maximal Margin Classification
    Algorithm. NIPS. 2000.
```

```python
# Let us check the feature names
print(digits.feature_names)
```

```
['pixel_0_0', 'pixel_0_1', 'pixel_0_2', 'pixel_0_3', 'pixel_0_4', 'pixel_0_5',
'pixel_0_6', 'pixel_0_7', 'pixel_1_0', 'pixel_1_1', 'pixel_1_2', 'pixel_1_3',
'pixel_1_4', 'pixel_1_5', 'pixel_1_6', 'pixel_1_7', 'pixel_2_0', 'pixel_2_1',
'pixel_2_2', 'pixel_2_3', 'pixel_2_4', 'pixel_2_5', 'pixel_2_6', 'pixel_2_7',
'pixel_3_0', 'pixel_3_1', 'pixel_3_2', 'pixel_3_3', 'pixel_3_4', 'pixel_3_5',
'pixel_3_6', 'pixel_3_7', 'pixel_4_0', 'pixel_4_1', 'pixel_4_2', 'pixel_4_3',
'pixel_4_4', 'pixel_4_5', 'pixel_4_6', 'pixel_4_7', 'pixel_5_0', 'pixel_5_1',
'pixel_5_2', 'pixel_5_3', 'pixel_5_4', 'pixel_5_5', 'pixel_5_6', 'pixel_5_7',
'pixel_6_0', 'pixel_6_1', 'pixel_6_2', 'pixel_6_3', 'pixel_6_4', 'pixel_6_5',
'pixel_6_6', 'pixel_6_7', 'pixel_7_0', 'pixel_7_1', 'pixel_7_2', 'pixel_7_3',
'pixel_7_4', 'pixel_7_5', 'pixel_7_6', 'pixel_7_7']


# type() function returns the datatype of the object.
type(digits.images)
numpy.ndarray

# Let us check the target names
print(digits.target_names)

[0 1 2 3 4 5 6 7 8 9]

# Let us check the shape of the dataset
digits.images.shape
(1797, 8, 8)
# Let us check how an image is represented
digits.images[0]
array([[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.],
       [ 0.,  0., 13., 15., 10., 15.,  5.,  0.],
       [ 0.,  3., 15.,  2.,  0., 11.,  8.,  0.],
       [ 0.,  4., 12.,  0.,  0.,  8.,  8.,  0.],
       [ 0.,  5.,  8.,  0.,  0.,  9.,  8.,  0.],
       [ 0.,  4., 11.,  0.,  1., 12.,  7.,  0.],
       [ 0.,  2., 14.,  5., 10., 12.,  0.,  0.],
       [ 0.,  0.,  6., 13., 10.,  0.,  0.,  0.]])
# Let us display first two images using matplotlib

import matplotlib.pyplot as plt
_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
for ax, image, label in zip(axes, digits.images, digits.target):
    ax.set_axis_off()
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation="nearest")
    ax.set_title("Training: %i" % label)
```



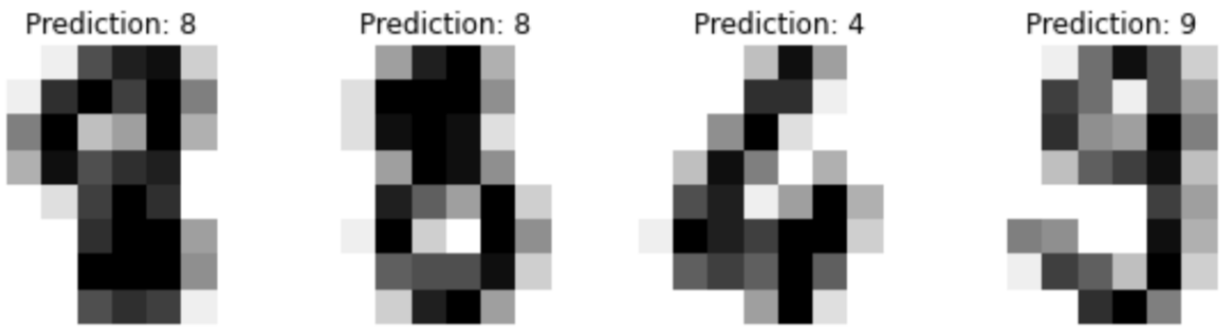
Further data can be split into train and test subsets and fit an MLP classifier on the training samples. The fitted classifier can subsequently be used to predict the value of the digit for the samples in the test subset.

```
# flattening the images
```

```
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))

from sklearn import metrics
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split

# Spliting data into 80% for training and 20% for testing
X_train, X_test, y_train, y_test = train_test_split(data,
                                                    digits.target,
                                                    test_size = 0.5,
                                                    shuffle = False
                                                    )

# Creating a classifier: an MLP classifier with 2 hidden layers
#                          - hidden layer 1 with 128 neurons
#                          - hidden layer 2 with 64 neurons

mlp = MLPClassifier(hidden_layer_sizes = (128 , 64), activation = 'logistic', a
lpha =  1e-4,
                    solver = 'sgd', tol = 1e-4, random_state = 1,
                    learning_rate_init = .1, verbose = True)

# Training the MLP with training data
mlp.fit(X_train, y_train)

# Predicting the value of the digit on the test data
y_predicted = mlp.predict(X_test)
```

```
Iteration 1, loss = 2.31806058
Iteration 2, loss = 2.23554298
Iteration 3, loss = 2.11152680
Iteration 4, loss = 1.93318102
Iteration 5, loss = 1.68340148
Iteration 6, loss = 1.34706543
Iteration 7, loss = 1.01047382
Iteration 8, loss = 0.73110347
Iteration 9, loss = 0.52564421
Iteration 10, loss = 0.38311327
Iteration 11, loss = 0.28745303
Iteration 12, loss = 0.22321755
Iteration 13, loss = 0.17980040
Iteration 14, loss = 0.14927177
Iteration 15, loss = 0.12598217
Iteration 16, loss = 0.10774715
Iteration 17, loss = 0.09467110
Iteration 18, loss = 0.08318579
Iteration 19, loss = 0.07544766
Iteration 20, loss = 0.06862140
Iteration 21, loss = 0.06272211
Iteration 22, loss = 0.05679316
Iteration 23, loss = 0.05173199
Iteration 24, loss = 0.04779437
Iteration 25, loss = 0.04500338
Iteration 26, loss = 0.04184104
Iteration 27, loss = 0.03904551
Iteration 28, loss = 0.03736341
Iteration 29, loss = 0.03486625
Iteration 30, loss = 0.03279666
Iteration 31, loss = 0.03104739
Iteration 32, loss = 0.02921414
Iteration 33, loss = 0.02809778
Iteration 34, loss = 0.02657393
Iteration 35, loss = 0.02545311
Iteration 36, loss = 0.02446084
Iteration 37, loss = 0.02329054
Iteration 38, loss = 0.02214299
Iteration 39, loss = 0.02126856
Iteration 40, loss = 0.02052622
Iteration 41, loss = 0.01975256
Iteration 42, loss = 0.01894544
Iteration 43, loss = 0.01845495
Iteration 44, loss = 0.01781872
Iteration 45, loss = 0.01701399
Iteration 46, loss = 0.01664254
Iteration 47, loss = 0.01585164
Iteration 48, loss = 0.01541949
Iteration 49, loss = 0.01493487
Iteration 50, loss = 0.01445705
Iteration 51, loss = 0.01395677
Iteration 52, loss = 0.01356621
Iteration 53, loss = 0.01313677
Iteration 54, loss = 0.01281921
Iteration 55, loss = 0.01243177
Iteration 56, loss = 0.01217347
Iteration 57, loss = 0.01179346
Iteration 58, loss = 0.01146299
Iteration 59, loss = 0.01118587
Iteration 60, loss = 0.01095729
Iteration 61, loss = 0.01066158
Iteration 62, loss = 0.01042367
Iteration 63, loss = 0.01020141
Iteration 64, loss = 0.01003254
Iteration 65, loss = 0.00970392
Iteration 66, loss = 0.00953260
Iteration 67, loss = 0.00931406
Iteration 68, loss = 0.00907894
Iteration 69, loss = 0.00893826
Iteration 70, loss = 0.00871577
Iteration 71, loss = 0.00856670
Iteration 72, loss = 0.00840610
Iteration 73, loss = 0.00823089
Iteration 74, loss = 0.00809737
Iteration 75, loss = 0.00792736
Iteration 76, loss = 0.00780340
Iteration 77, loss = 0.00764604
Iteration 78, loss = 0.00751589
Iteration 79, loss = 0.00738729
Iteration 80, loss = 0.00725937
Iteration 81, loss = 0.00714130
Iteration 82, loss = 0.00704057
Iteration 83, loss = 0.00690226
Iteration 84, loss = 0.00681652
Iteration 85, loss = 0.00670753
Iteration 86, loss = 0.00657451
Iteration 87, loss = 0.00648019
Iteration 88, loss = 0.00638242
Iteration 89, loss = 0.00629340
Iteration 90, loss = 0.00620439
Iteration 91, loss = 0.00612568
Iteration 92, loss = 0.00602496
Iteration 93, loss = 0.00595090
Iteration 94, loss = 0.00585264
Iteration 95, loss = 0.00575984
Iteration 96, loss = 0.00570582
Iteration 97, loss = 0.00561369
Iteration 98, loss = 0.00553325
Iteration 99, loss = 0.00546445
Iteration 100, loss = 0.00539742
Iteration 101, loss = 0.00533112
Iteration 102, loss = 0.00526372
Iteration 103, loss = 0.00520110
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
```

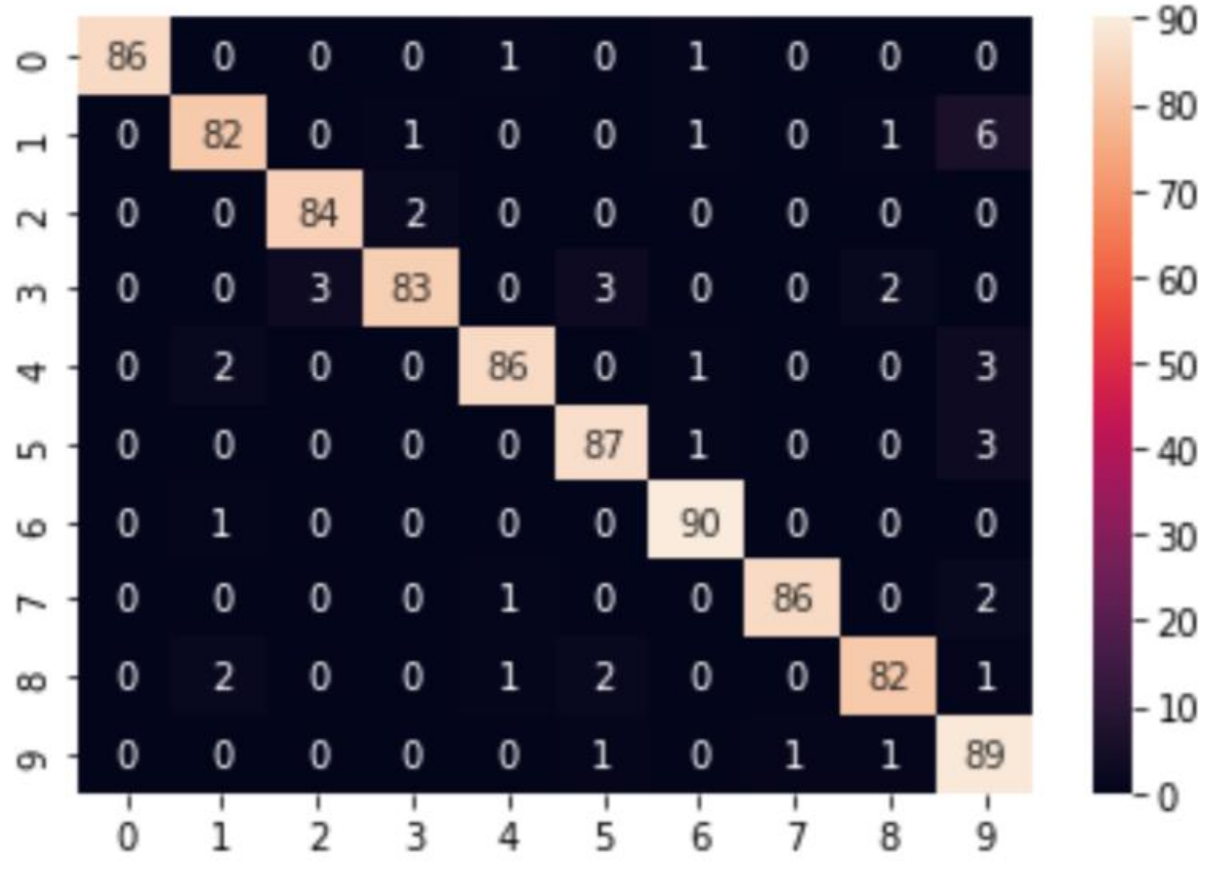
```python
# Below we visualize the first 4 test samples and show their predicted digit va
lue in the title.


_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
for ax, image, prediction in zip(axes, X_test, y_predicted):
    ax.set_axis_off()
    image = image.reshape(8, 8)
    ax.imshow(image, cmap = plt.cm.gray_r, interpolation = "nearest")
    ax.set_title(f"Prediction: {prediction}")
```



We can plot a `confusion matrix <confusion_matrix>` of the true digit values and the predicted digit values in the form of a heatmap.

```
from sklearn.metrics import confusion_matrix , accuracy_score
import seaborn as sns

# computing a confusion matrix
cm = confusion_matrix(y_test, y_predicted)
# displaing confusion matrix as a heatmap
sns.heatmap(cm, annot = True , fmt = 'd')

# computing and displaying the the accuracy of the KNN model
print('The % of Accuracy is :', accuracy_score(y_test , y_predicted)*100)
```

```
The % of Accuracy is : 95.10567296996662
```



```
# Visualizing weights of MLP

print("weights between input and first hidden layer:")
print(mlp.coefs_[0])

print("\nweights between first hidden and second hidden layer:")
print(mlp.coefs_[1])
```

```
weights between input and first hidden layer:
[[-0.01693263  0.04495979 -0.1020075  ...  0.00792393  0.01077894
   0.0697954 ]
 [-0.10832321 -0.05075487  0.01556038 ...  0.09515672  0.01825767
   0.05450046]
 [-0.06968199 -0.08100316 -0.02565275 ...  0.00246374  0.1093701
  -0.12351096]
 ...
 [ 0.0448075   0.10889104 -0.05133974 ...  0.03960139  0.00773755
   0.16546269]
 [ 0.13746521 -0.09864133  0.03776946 ...  0.09360828 -0.00222922
  -0.0966712 ]
 [ 0.02722291 -0.06329276 -0.08054808 ... -0.08376382  0.09110037
   0.02449374]]

weights between first hidden and second hidden layer:
[[-0.20563751  0.10088131  0.06349793 ...  0.08711656 -0.13983782
  -0.22816924]
 [ 0.14888228 -0.08867514  0.04615003 ... -0.14848029  0.01351488
   0.29328764]
 [-0.05430794  0.04978504 -0.09730345 ...  0.11565929 -0.02747184
   0.02509967]
 ...
 [-0.0275512   0.00269346 -0.07988297 ...  0.04356784 -0.10349571
   0.08501737]
 [-0.09235084 -0.05238574 -0.04346453 ...  0.08302493 -0.09771271
  -0.03994741]
 [-0.05631216 -0.05575696  0.05710291 ... -0.03214848 -0.09666771
   0.1159155 ]]
```

5.7 Results and Discussion:

The neural network that will read the image of a digit and correctly identify the number, built successfully with Maximum Accuracy and Minimum Loss.