2020/2021

MINI PROJECT

Submitted in fulfillment of the requirements for the

MINI PROJECT GRADE FROM THE LEBANESE UNIVERSITY FACULTY OF ENGINEERING – BRANCH III

Major: Electrical and Telecommunication Engineering

Prepared By:

Dani Zeineddine 5485

Wafik Zahwa 5558

MNIST Digits Classification using Neural Networks

Supervised by:

Dr. Mohamad Awde

Defended on **19 July** 2021.

Table of contents

Abstract

Traditional systems of handwriting recognition have relied on handcrafted features and a large amount of prior knowledge. Research in the handwriting recognition field is focused around deep learning techniques and has achieved breakthrough performance in the last few years. Still, the rapid growth in the amount of handwritten data and the availability of massive processing power demands improvement in recognition accuracy and deserves further investigation.

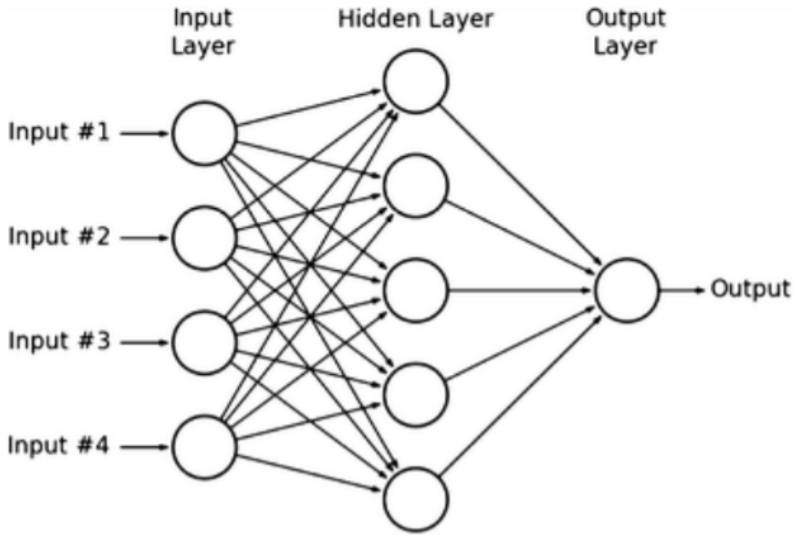
Here, our objective is to write a full python code using Tensorflow library to build a neural network and benefit from it to predict handwritten digits with a high accuracy .

Introduction

Artificial neural networks (ANNs) are inspired by biological central systems in brains and have been utilized in a variety of applications ranging from modeling, classification, pattern recognition, and multivariate data analysis. And in practical ANN application, Back-propagation neural network is widely applied.

A recurrent neural network (RNN) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Derived from feedforward neural networks, RNNs can use their internal state (memory) to process variable length sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition.

This mini project uses MNIST handwritten digit database on Artificial Neural Network (ANN). It has become a standard for fast-testing theories of pattern recognition and machine learning algorithms. The MNIST database was constructed out of the original NIST database; hence, modified NIST or MNIST. It contains 60,000 handwritten digit images for the classifier training and 10,000 handwritten digit images for the classifier testing, both drawn from the same distribution. All these black and white digits are size normalized, and centered in a fixed-size image where the center of the intensity lies at the center of the image with 28 × 28 pixels. The dimensionality of each image sample vector is 28 * 28 = 784, where each element is binary.
In this project, we use Back-propagation neural networks to achieve the classification problem. Back- propagation neural networks are supervised multi-layer feed forward neural networks, commonly consisting of an input layer, an output layer, and one or several hidden layers.



Back-propagation neural network setting up

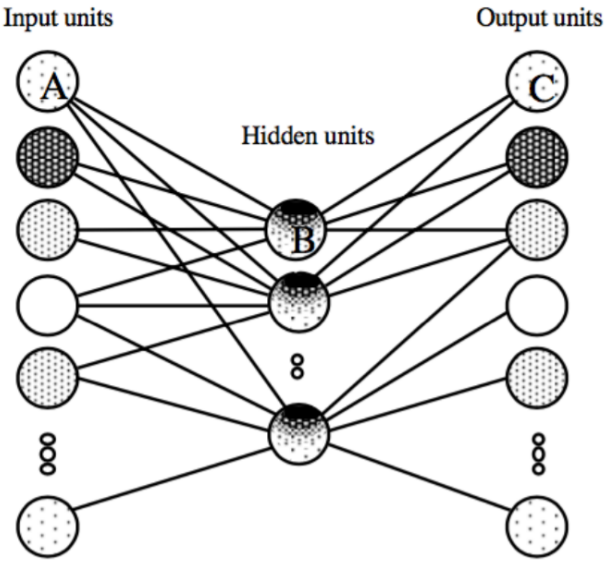
MNIST handwritten digit database in which the handwritten digits have been preprocessed including segmentation and normalization, contains 60,000 training images and 10,000 test images, and the dimensionality of each image sample vector is 28 * 28 = 784, where each element is binary.

Back-propagation neural networks are supervised multi-layer feed forward neural networks, commonly consisting of an input layer, an output layer, and one or several hidden layers. Here we build a neural network with only one hidden layer. Input layer contains 28 * 28 = 784 neurons, representing the features; Hidden layer contains 300 neurons, using Sigmoid as activation function; And output layer contains 10 neurons representing the digits from 0 to 9. Besides, we can use a mathematical error function as a network loss function and of course an optimizer.

After defining the neural network, we train the model. In order to set the appropriate parameters, we conducted several tests to evaluate. There should be a balance among these parameters, contributing to the performance of the neural network together.

In order to determine the performance of the neural network and predictions and report the results produced by our neural network, we can calculate the training accuracy and testing accuracy.

All these steps can be applied in an easy way using the tensorflow python library.



Implementation

In order to train a digit detector, we need to break our project into two distinct phases.

Step one: Training:

Here we'll focus on loading our MNIST dataset from disk, training a model (using Keras/TensorFlow) on this dataset, and then serializing the digit detector to disk.

Step two: Deployment

Once the digit classifier is trained, we can then move on to loading it, and then performing digit classification by classifying each digit as what number it represents.

Our implementation was done using Google Colab Research, python script is "Digit Prediction.ipynb" .

1)Importing libraries:

```
import tensorflow as tf
from matplotlib import pyplot as plt
import numpy as np
```

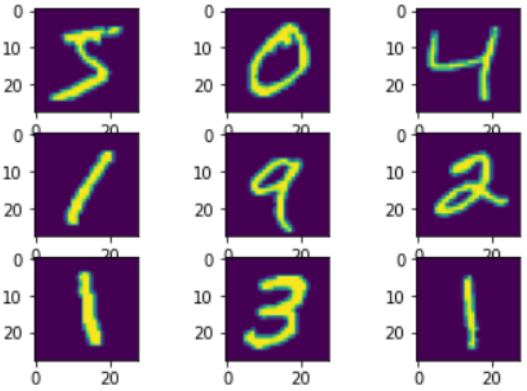
2)Importing the MNIST dataset:

```
objects =  tf.keras.datasets.mnist
(training_images, training_labels), (test_images, test_labels) = objects.load_data()
```

Importing MNIST dataset is done using keras library and stored in "objects" variable. And next it is splitted into into two sections ,the training section and the testing section and each section it is splitted into images and labels.

3)Plotting our training images:

```
for i in range(9):
  # define subplot
  plt.subplot(330 + 1 + i)
  # plot raw pixel data
  plt.imshow(training_images[i])
```



4)Printing the 28*28 pixels matrix of an image

```
print(training_images.shape)
print(training_images[0])
```

5)Normalizing the 28*28 pixels matrix of all dataset images:

```
training_images  = training_images / 255.0
test_images = test_images / 255.0
```

6)Defining the model,setting parameters and optimizer:

```
model = tf.keras.models.Sequential([tf.keras.layers.Flatten(input_shape=(28,28)),
                                    tf.keras.layers.Dense(128, activation='relu'),
                                    tf.keras.layers.Dense(10, activation=tf.nn.softmax)])
```

```
model.compile(optimizer = 'adam',
              loss = 'sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

7)Start training with 5 iterations and calculation accuracy in each time:

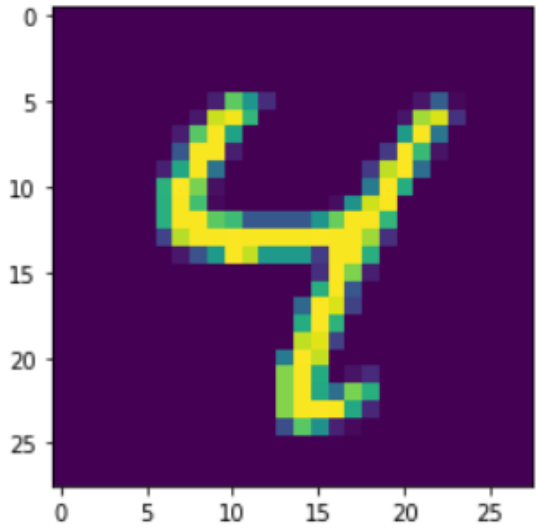
```
model.fit(training_images, training_labels, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 5s 2ms/step - loss: 0.2552 - accuracy: 0.9278
Epoch 2/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1158 - accuracy: 0.9653
Epoch 3/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0798 - accuracy: 0.9758
Epoch 4/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0596 - accuracy: 0.9819
Epoch 5/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0464 - accuracy: 0.9859
<tensorflow.python.keras.callbacks.History at 0x7f1cbc867f90>
```

8)Verifying a digit classification:

```
plt.imshow(test_images[6])
prediction=model.predict(test_images)
print(np.argmax(prediction[6]))
```

```
4
```



We can verify the classification of all our dataset by simply changing the index i of `print(np.argmax(prediction[i]))`.

Difference Between CNN and RNN

Convolutional Neural Networks	Recurrent Neural Networks
In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery.	A recurrent neural network (RNN) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence.
It is suitable for spatial data like images.	RNN is used for temporal data, also called sequential data.
CNN is a type of feed-forward artificial neural network with variations of multilayer perceptron's designed to use minimal amounts of preprocessing.	RNN, unlike feed-forward neural networks- can use their internal memory to process arbitrary sequences of inputs.
CNN is considered to be more powerful than RNN.	RNN includes less feature compatibility when compared to CNN.
This CNN takes inputs of fixed sizes and generates fixed size outputs.	RNN can handle arbitrary input/output lengths.
CNN's are ideal for images and video processing.	RNNs are ideal for text and speech analysis.
Applications include Image Recognition, Image Classification, Medical Image Analysis, Face Detection and Computer Vision.	Applications include Text Translation, Natural Language Processing, Language Translation, Sentiment Analysis and Speech Analysis.

As a conclusion from this table, CNN would be better than RNN in classifying handwritten digits. But in the case of the MNIST dataset which is not a complex dataset and each image does not contain any noise (includes only a handwritten digit) , using RRN would not affect the accuracy of the classifier and all the tested results will be right.

Conclusion

In this work , we have defined what a neural network is , how it works and what its aim is.
We conducted an experiment implementing Back-propagation Neural Network to achieve the classification of the MNIST handwritten digit database. In the experimental model, 28 * 28 = 784 pixels are regarded as input and 10 different classes of digits from 0 to 9 as output. Besides, we use classification accuracy to determine the performance of the neural network.
And by benefiting from the tensorflow python library , we built our full program to classify digits from the MNIST database.

References

1. Basheer, I.A., and Hajmeer, M., 2000. Artificial neural networks: fundamentals, computing, design, and application. Journal of microbiological methods.

2. Ciresan, D.C., Meier, U., Gambardella, L.M. and Schmidhuber, J., 2011, September. Convolutional neural network committees for handwritten character classification. In Document Analysis and Recognition (ICDAR), 2011 International Conference. IEEE.

Websites:

Google COLAB Research.