Image Classification Using Artificial Neural Networks [1]

by. Dian Ade Kurnia, M.Kom

STMIK IKMI Cirebon

Introduction

Before applying artificial neural networks over a set of images for classification, let's first examine what neural networks are. The structure of a neuron is presented in Figure

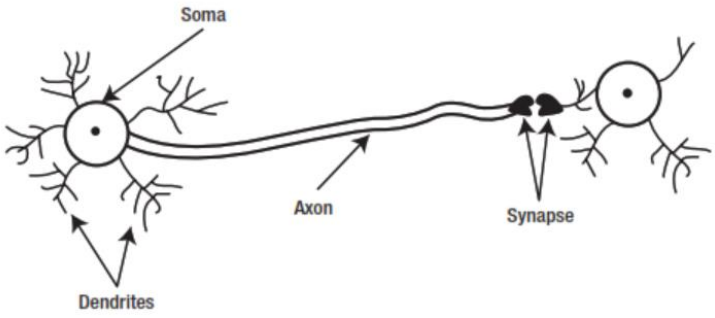

Figure 1 : Biological Neuron

Artificial neural networks use the same analogy, and process information using artificial neurons. Information is transferred from one artificial neuron to another, which finally leads to an activation function, which acts like a brain and makes a decision. The structure of a simple artificial neural network is shown in Figure, below :

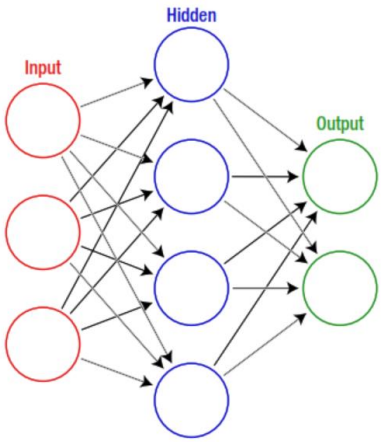

Figure 2 : Simple Artificial Neural Network

Methodology & Datasets

The Modified National Institute of Standards and Technology (MNIST) database includes a dataset that contains approximately 60,000 training images and 10,000 testing images of handwritten digits. We'll use the training dataset* to train our neural network, and then we'll use the test dataset* to look at its accuracy. Finally, you can give your own handwritten digit to check the predictions by our trained model. First, let us look at a flowchart of how to proceed with neural networks

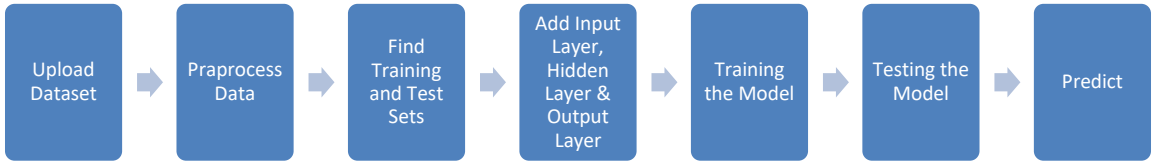

Figure 3 : Flowchart of Process Artificial Neural Network

First, we have to download the train and test datasets from the MNIST database. We can download them from the kaggle web site (https://www.kaggle.com/c/digit-recognizer/data#).

Implementation with Python

Now let's look at our training set. It consists of 785 columns. Each image has 28 × 28 resolution. Therefore, 784 columns contain the pixel values of each digit

```
import keras
from keras.models import Sequential
from keras.layers import Dense
```

```
[ ] import pandas as pd
    input_data=pd.read_csv('/content/drive/My Drive/dataset/tabel/digit-recognizer/train.csv')
    input_data.head(10)
```

	label	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel11	pixel12	pixel13
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0

We must create two data frames in Python. One will store all the pixel values X; the other will store the actual number y

```
[ ]  y=input_data['label']
     input_data.drop('label',axis=1,inplace=True)
     X=input_data
```

Now we convert the labels present in y into dummies (see important terms).

```
[ ]  y = pd.get_dummies(y)
```

Now that we have our data in X and y, we can start with our neural network. Let's create four hidden layers, one input layer, and one output layer using Keras

```
[ ]  classifier = Sequential()
     classifier.add(Dense(units = 600, kernel_initializer = 'uniform', activation = 'relu', input_dim = 784))
     classifier.add(Dense(units = 400, kernel_initializer = 'uniform', activation = 'relu'))
     classifier.add(Dense(units = 200, kernel_initializer = 'uniform', activation = 'relu'))
     classifier.add(Dense(units = 10, kernel_initializer = 'uniform', activation = 'sigmoid'))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:66: The name tf.get_de

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:541: The name tf.place

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:4432: The name tf.ranc
```

Now we need to compute the stochastic gradient descent algorithm* to minimize the loss:

```
[ ]  classifier.compile(optimizer='sgd', loss='mean_squared_error',metrics=['accuracy'])
```

Finally, we start the training by giving a batch size* of ten and epochs* of ten:

```
[ ]  classifier.fit(X,y,batch_size=10, epochs=10)
```

```
42000/42000 [==============================] - 29s 681us/step - loss: 0.0206 - acc: 0.8728
Epoch 2/10
42000/42000 [==============================] - 28s 668us/step - loss: 0.0092 - acc: 0.9486
Epoch 3/10
42000/42000 [==============================] - 28s 667us/step - loss: 0.0068 - acc: 0.9635
Epoch 4/10
42000/42000 [==============================] - 28s 665us/step - loss: 0.0055 - acc: 0.9706
Epoch 5/10
42000/42000 [==============================] - 28s 670us/step - loss: 0.0045 - acc: 0.9762
Epoch 6/10
42000/42000 [==============================] - 28s 669us/step - loss: 0.0038 - acc: 0.9800
Epoch 7/10
42000/42000 [==============================] - 29s 689us/step - loss: 0.0032 - acc: 0.9829
Epoch 8/10
42000/42000 [==============================] - 28s 669us/step - loss: 0.0028 - acc: 0.9855
Epoch 9/10
42000/42000 [==============================] - 28s 669us/step - loss: 0.0025 - acc: 0.9871
Epoch 10/10
42000/42000 [==============================] - 28s 665us/step - loss: 0.0022 - acc: 0.9886
<keras.callbacks.History at 0x7fbba1cb9080>
```

Finally, we predict on the test dataset. First we upload it, then we do the predictions

```
[16] test_data=pd.read_csv("/content/drive/My Drive/dataset/tabel/digit-recognizer/test.csv")
     y_pred=classifier.predict(test_data)
```

Reference

[1] F. F. Recognition, O. Detection, P. Recognition, U. Python, and H. Singh, *Practical Machine Learning and Image Processing*. .