Introduction to Keras

February 06 , 2023

Outline

- ❑ Introduction
- ❑ Architecture of Keras
- ❑ Building a Simple Deep Learning Network using Keras

Introduction

Complete Neural Network Pipeline include :

- Problem (Application)
- Dataset
- Preprocessing
- Training and Testing Dataset
- Type of Model
- No of layers
- No of Nodes
- Activation Function
- Batch size
- Epoch
- Optimization Function
- Initialization of Weights and Bias
- Evaluation Metrics

Introduction

- **Implementing complete pipeline and Experimenting with it is a complex task**

K Keras

- High-level deep learning API
- Written in python
- Use TensorFlow or Theano for its backend
- Support almost all deep learning models
- Runs smoothly on CPU and GPU

Introduction

Why Keras

- Easy to use and enable fast experimentation
- Support distributed training
- Modular in nature
- Models are described in Python, which make it easy to debug and explore.

Introduction

Installation

Important libraries :

- Python
- Numpy
- Scipy
- h5py
- Matplotlib
- TensorFlow

Tools:

- Google Colab
- Anaconda

- Visit the Keras page to install and explore the API : https://keras.io

Architecture of Keras

Models

- Sequential API
- Functional API

Layers

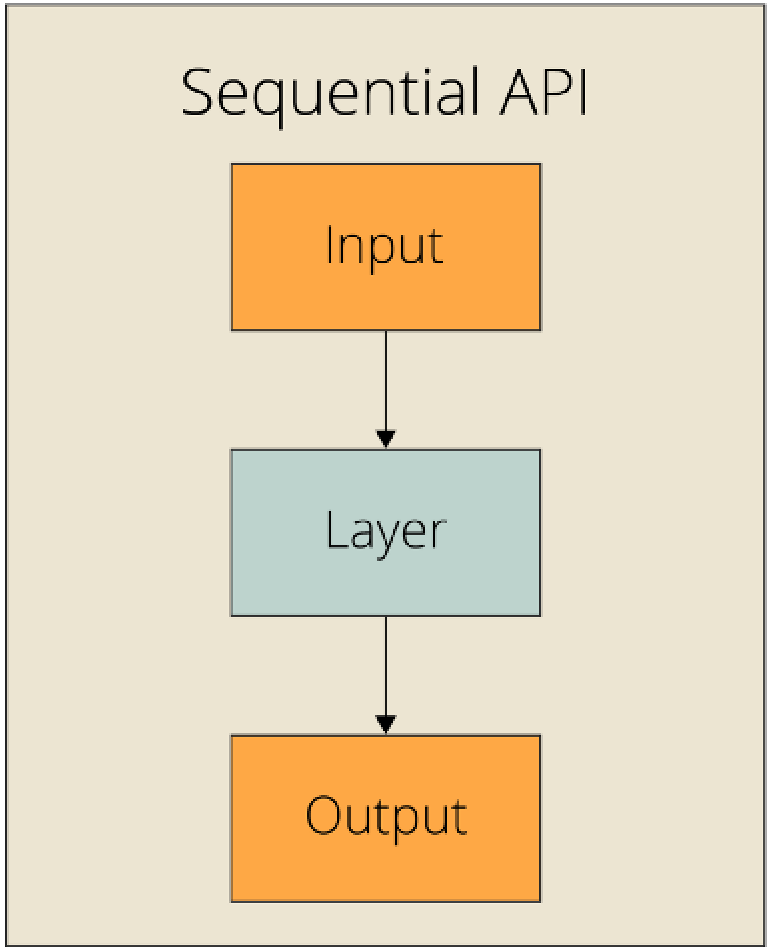
- Core Layers
  - Dense
  - Flatten
  - Reshape
  - many more..
- Convolution Layers
- Pooling Layers
- Recurrent Layers
- many more….

Other Modules

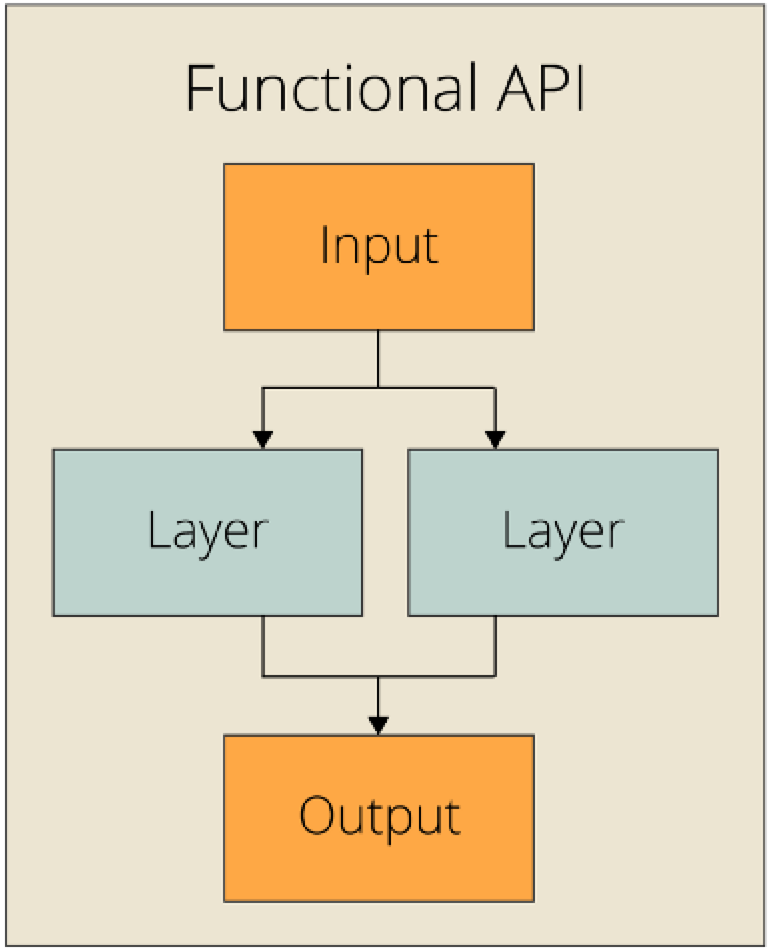
- Data Loading
- Datasets
- Applications
- Utilities
- Keras Tuner

Models

Sequential API



Functional API



Keras Provides

Optimizer

- Algorithm used to update weights while we train our model
- such as sgd (Stochastic gradient descent optimizer)

Objective Function

- Used by the optimizer to navigate the space of weights
- such as mse (mean squared error)

Metrics

- Used to judge the performance of your model such as accuracy

Building a Simple Deep Learning Network using Keras

Steps

- Import libraries and modules
- Load data
- Pre-process data
- Define model architecture
- Compile model
- Fit and evaluate

Building a Simple Deep Learning Network using Keras

Problem :

- Digit recognition from image data

Dataset :

- Keras provides in-build many datasets such as MNIST , CIFAR10 and many more.

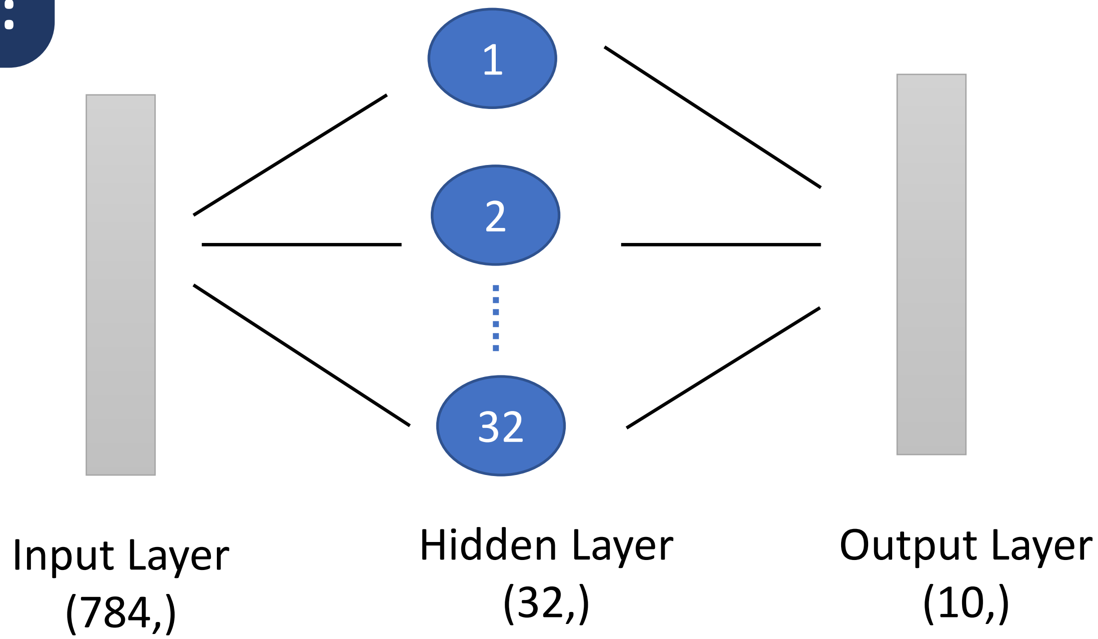
- **MNIST**
  - Dataset contains grayscale images of handwritten single digits between 0 and 9
  - 28×28 pixels
  - Training set of 60,000 examples, and a test set of 10,000 examples
  - Keras provides method to load MNIST data set

Building a Simple Deep Learning Network using Keras

Data Preprocessing :

- Reshaping
- Convert data type
- Change the labels from integer to categorical data

Model Architecture :



Building a Simple Deep Learning Network using Keras

Model Architecture :

- Use sequential model
- A sequential model is defined as

  model = Sequential()
- Add layers
  - First layer in a Sequential model needs to receive information about its input shape
  - Dense(32, input dim=784) specifies that
    - Input dimension is 784
    - It is first hidden layer
    - output dimension is 32
    - If no activation function specified, no activation is applied (i.e. ”linear” activation: a(x) = x).

Building a Simple Deep Learning Network using Keras

Model Architecture :

- There are many other initializations available in Keras
- Rectifier (ReLU) activation function is used for the neurons in the hidden layer
- Softmax activation function is used on the output layer

Building a Simple Deep Learning Network using Keras

Compile Model :

- Before training, use compile() method to build network. It uses three arguments:
  - Optimizer : Adam
  - Loss function : Logarithmic loss
  - list of metrics : Accuracy

Train Model :

- Use fit() function

Evaluate Model on test data:

- Use evaluate () function

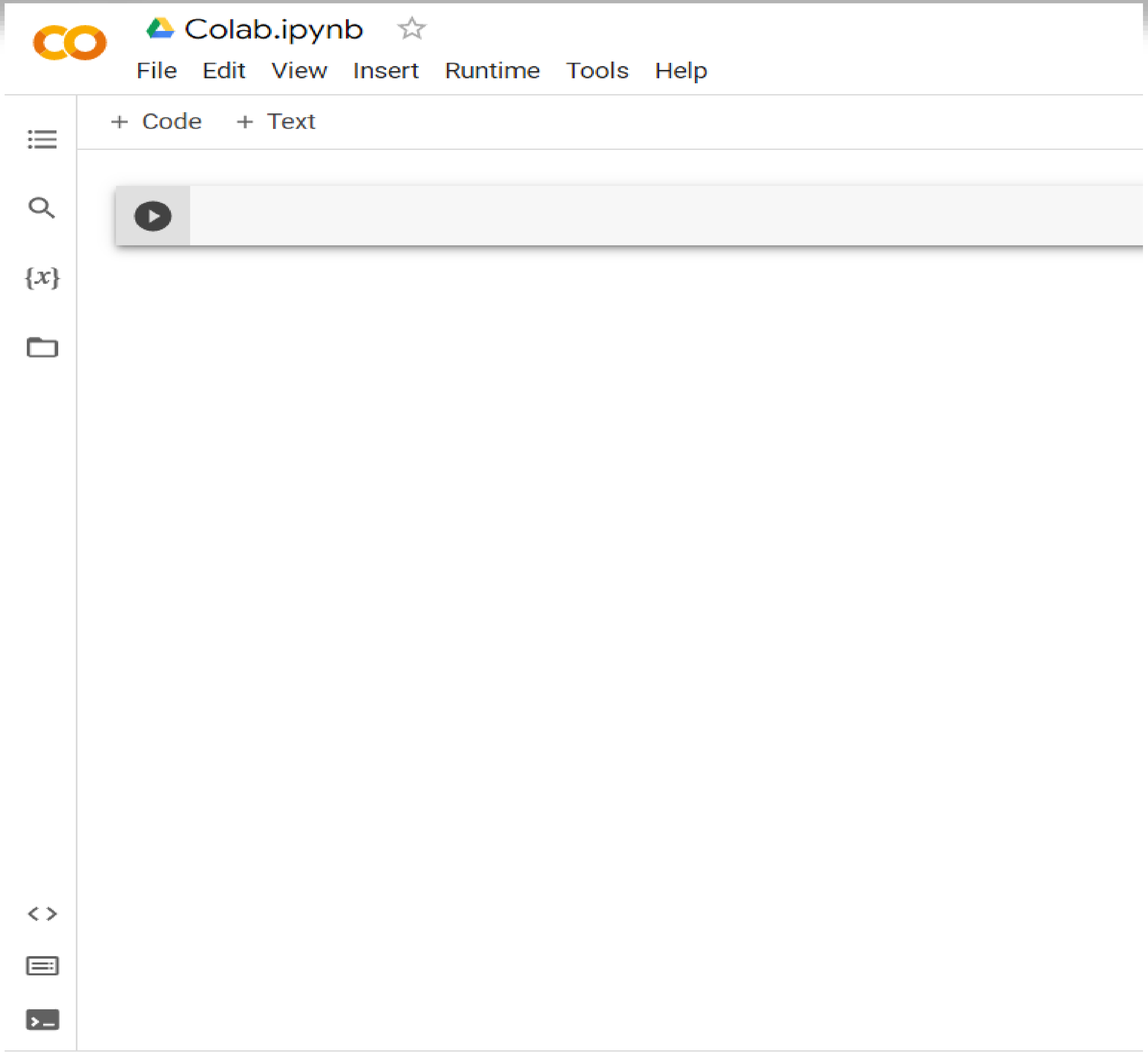
Building a Simple Deep Learning Network using Keras

Options to explore:

- Different learning rate for optimizer
- Number of neurons in hidden layer
- Batch size
- Additional hidden layers
- With dropout
- Different optimizers
- Increases number of epochs

Introduction to Google Colab

- Free Google service
- Free GPU
- Pre-installed libraries
- Built on top of Jupyter Notebook



Examples

- Classification using feed forward network

A simple feed forward network for MNIST image classification

```
# Import the required packages
from keras.models import Sequential
from keras.layers import Dense
from keras.datasets import mnist
from keras.utils.np_utils import to_categorical
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
```

Get the training data

```
# Get the training data
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
print('Training data shape : ', train_images.shape, train_labels.shape)

print('Testing data shape : ', test_images.shape, test_labels.shape)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 2s 0us/step
11501568/11490434 [==============================] - 2s 0us/step
Training data shape :  (60000, 28, 28) (60000,)
Testing data shape :  (10000, 28, 28) (10000,)
```

```
# Find the unique numbers from the train labels
classes = np.unique(train_labels)
classes_num = len(classes)
print('Unique output classes : ', classes)
print('Total number of outputs : ', classes_num)
```

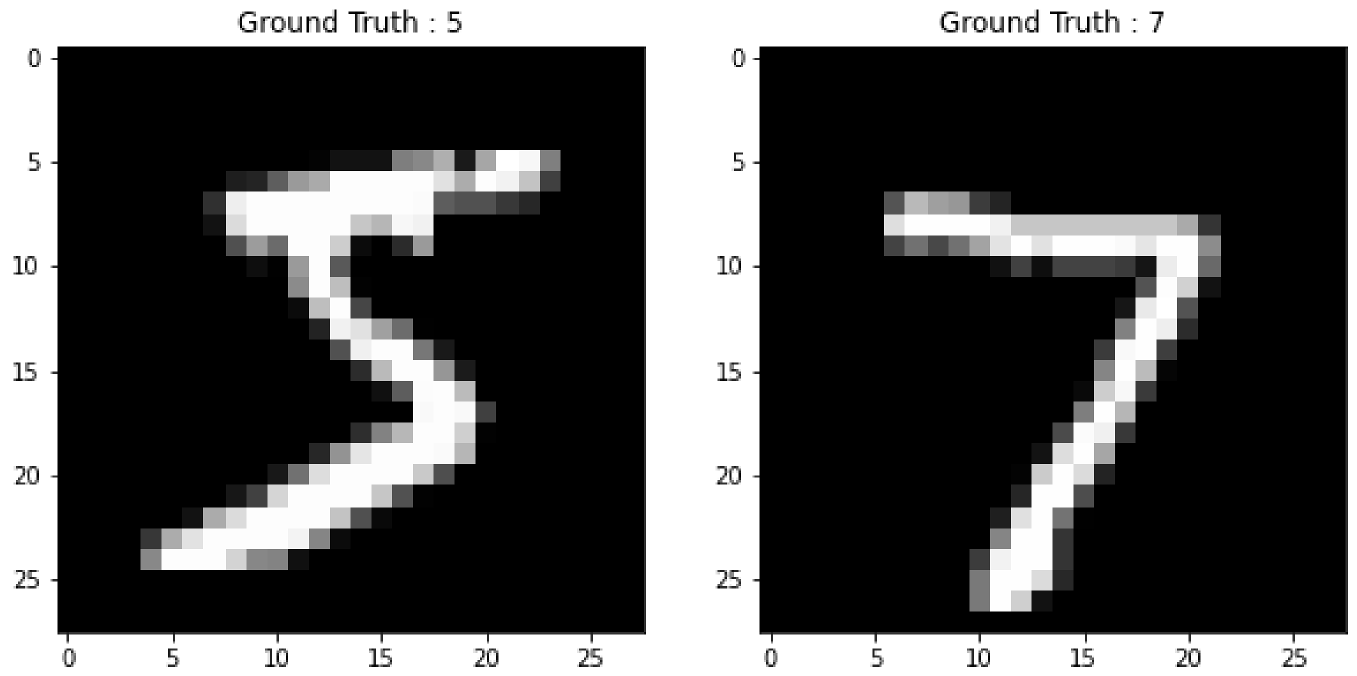
```
Unique output classes :  [0 1 2 3 4 5 6 7 8 9]
Total number of outputs :  10
```

```
# let's see some sample images in the dataset

# Define the plot size
plt.figure(figsize=[10,5])

# Display the first image in training data
plt.subplot(121)
plt.imshow(train_images[0,:,:], cmap='gray')
plt.title("Ground Truth : {}".format(train_labels[0]))

# Display the first image in testing data
plt.subplot(122)
plt.imshow(test_images[0,:,:], cmap='gray')
plt.title("Ground Truth : {}".format(test_labels[0]))
plt.show()
```



```python
# Change the image format from 2D array of size 28x28 to 1D arrya of size 784
print(train_images.shape)
```

```
(60000, 28, 28)
```

```python
# Get the size of required 1D array

dim_data = np.prod(train_images.shape[1:])
print(dim_data)
```

```
784
```

```python
# Now reshape the 2D array to 1D array

train_data = train_images.reshape(train_images.shape[0], dim_data)
test_data = test_images.reshape(test_images.shape[0], dim_data)
```

```python
# Change to float datatype

train_data = train_data.astype('float32')
test_data = test_data.astype('float32')
```

```python
# Change the labels from integer to categorical data

train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)

# Check how the one hot encoded labels look like
print(test_labels_one_hot[0:10])
```

```
[[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 1. 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. 1.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]]
```

```python
# Define the model

model = Sequential() # type of model

# Define the model layers

model.add(Dense(32, activation='relu', input_shape=(dim_data,)))
model.add(Dense(classes_num, activation='softmax'))

# Compile the model

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

```

```python
# Let's see how the model looks and check the parameters

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 32)                25120

 dense_1 (Dense)             (None, 10)                330

=================================================================
Total params: 25,450
Trainable params: 25,450
Non-trainable params: 0
```

Train the model

```
# Now we can start the training

history = model.fit(train_data, train_labels_one_hot, batch_size=256, epochs=2, verbose =1,
                    validation_data=(test_data, test_labels_one_hot))
```

```
Epoch 1/2
235/235 [==============================] - 2s 6ms/step - loss: 4.5760 - accuracy: 0.3473 - val_loss: 1.5965 - val_accuracy: 0.4
251
Epoch 2/2
235/235 [==============================] - 1s 5ms/step - loss: 1.4386 - accuracy: 0.4916 - val_loss: 1.3346 - val_accuracy: 0.5
221
```

Evaluate the model

```
# let's run the trained model on test data and see how it performs

[test_loss, test_acc] = model.evaluate(test_data, test_labels_one_hot)
print("Evaluation result on Test Data : Loss = {}, accuracy = {}".format(test_loss, test_acc))
```

```
313/313 [==============================] - 0s 2ms/step - loss: 1.3346 - accuracy: 0.5221
Evaluation result on Test Data : Loss = 1.3345507383346558, accuracy = 0.5220999717712402
```

Run the sample predictions

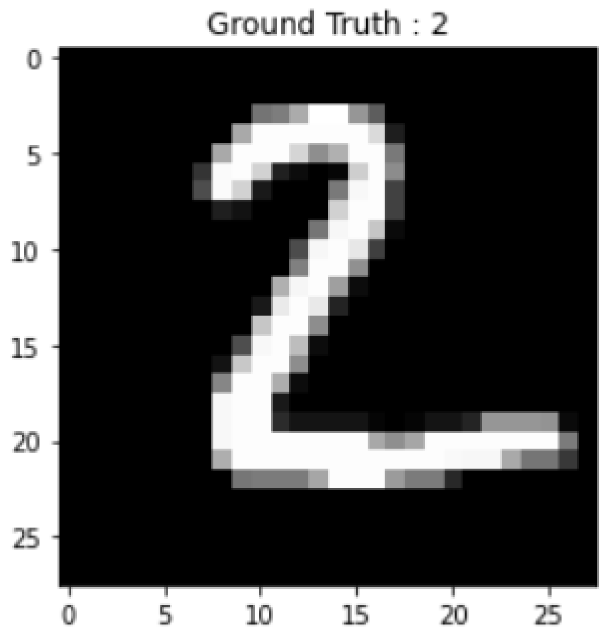
```
# Predict the most likely class

print("Probability of all the classes: {}".format(model.predict(test_data[[1],:])))

print("Model prediction: {}".format(np.argmax(model.predict(test_data[[1],:])),axis=1))

# Display the predicted image
plt.imshow(test_images[1], cmap='gray')
plt.title("Ground Truth : {}".format(test_labels[1]))
plt.show()
```

```
Probability of all the classes: [[1.9178690e-01 3.6784500e-10 7.9985172e-01 8.0762245e-03 1.5730684e-08
  3.3894274e-09 1.0777729e-07 1.5123810e-08 2.8397932e-04 1.1393026e-06]]
Model prediction: 2
```



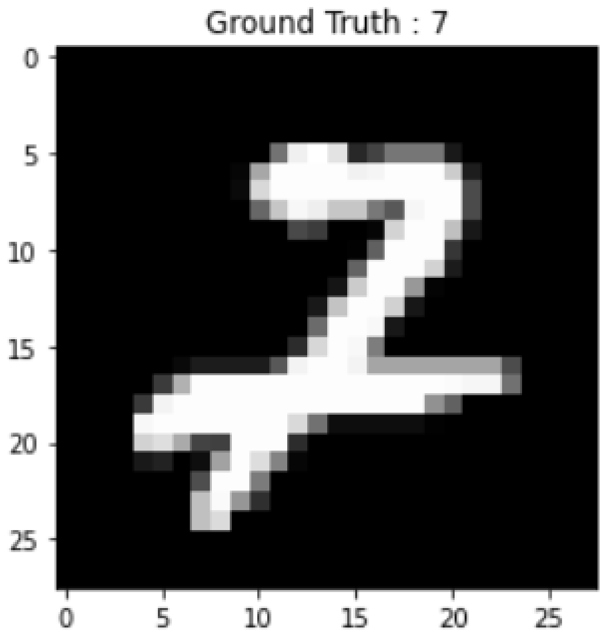
```python
# Predict the most likely class

print("Probability of all the classes: {}".format(model.predict(test_data[[9009]])))

print("Model prediction: {}".format(np.argmax(model.predict(test_data[[9009]]),axis=1)))

# Display the predicted image
plt.imshow(test_images[9009], cmap='gray')
plt.title("Ground Truth : {}".format(test_labels[9009]))
plt.show()
```

```
Probability of all the classes: [[0.13197137 0.06553078 0.11186828 0.13483736 0.0777123  0.12524498
  0.08340577 0.06744915 0.13503858 0.06694139]]
Model prediction: 8
```



```python
DIGITS = {
    0: '0',
    1: '1',
    2: '2',
    3: '3',
    4: '4',
    5: '5',
    6: '6',
    7: '7',
    8: '8',
    9: '9',
}
```

```python
def confusion_matrix(Y_true, Y_pred):
    Y_true = pd.Series([DIGITS[y] for y in np.argmax(Y_true, axis=1)])
    Y_pred = pd.Series([DIGITS[y] for y in np.argmax(Y_pred, axis=1)])

    return pd.crosstab(Y_true, Y_pred, rownames=['True'], colnames=['Pred'])
```

```python
print(confusion_matrix(test_labels_one_hot, model.predict(test_data)))
```

```
Pred   0     1    2   3    4   5    6    7    8    9
True
0      6     0   23   3    2   6    0    5  935    0
1      1  1071    1  10    0   6    2    0   44    0
2     16     1  611   7    8   3    4    4  378    0
3      2     5   15   5   10   2    2   16  948    5
4      0     2    1   3  816   1    7    3  132   17
5      1     9    3   2   25  40    5   58  745    4
6      0     3    2   8    4   4  672    0  265    0
7      4    14    1   4   29  31    0  792   93   60
8      1     1    0   1   32  11    2    5  921    0
9      0     5    0   2  609   2    2   20   82  287
```

Improve performance of feed forward network for MNIST image classification

```python

model = Sequential() # type of model

# Define the model layers

model.add(Dense(256, activation='relu', input_shape=(dim_data,)))
model.add(Dense(128, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(classes_num, activation='softmax'))

# Compile the model

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

```

```python
# Let's see how the model looks and check the parameters

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 256)               200960

 dense_1 (Dense)             (None, 128)               32896

 dense_2 (Dense)             (None, 32)                4128

 dense_3 (Dense)             (None, 10)                330

=================================================================
Total params: 238,314
Trainable params: 238,314
Non-trainable params: 0
```

Evaluate the model

```
# let's run the trained model on test data and see how it performs

[test_loss, test_acc] = model.evaluate(test_data, test_labels_one_hot)
print("Evaluation result on Test Data : Loss = {}, accuracy = {}".format(test_loss, test_acc))
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.2774 - accuracy: 0.9585
Evaluation result on Test Data : Loss = 0.2773880660533905, accuracy = 0.9585000276565552
```

```
print(confusion_matrix(test_labels_one_hot, model.predict(test_data)))
```

```
Pred    0     1    2    3    4    5    6    7    8    9
True
0     970     0    0    3    1    1    3    0    2    0
1       1  1119    3    5    0    1    2    2    2    0
2       4     5  991    8    1    0    3    5   13    2
3       1     0   12  971    0    8    0    5   12    1
4       8     5    5    0  908    1    8    5    3   39
5       7     0    1   24    1  841    9    1    7    1
6      11    10    0    1    4    8  921    0    3    0
7       1     6   20    7    1    1    1  981    2    8
8       4     0   10   12    3    4    2    3  931    5
9       4     7    2   11   11    4    0    9    9  952
```

Implementation of AND gate

```python
import keras
import numpy as np
from matplotlib import pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
```

```python
# Training set 1
# This will be used to train the network

# Input to the gate

x_train = np.array([[0,0],[0,1],[1,0],[1,1]], "uint8")

# Ouput of the gate, the truth value

y_train = np.array([[0],[0],[0],[1]], "uint8")
```

```python
# Training set 2
# Let us use real nos. instead of binary int values

# Input to the gate
x_train = np.random.uniform(low = 0, high = 1,size=400)
x_train = np.reshape(x_train, [-1,2])
print(x_train.shape)

# Output or truth values for inputs
y_train = np.zeros([x_train.shape[0]])
print(len(y_train))
for i in range(x_train.shape[0]):
    if x_train[i,0]>=0.5 and x_train[i,1]>=0.5:
        y_train[i]=1

# let's check the values in training set
print(x_train[:10])
print(y_train[:10])
```

```
(200, 2)
200
[[0.43793899 0.46888843]
 [0.3708003  0.52206144]
 [0.88984203 0.40711617]
 [0.11018836 0.47267797]
 [0.33346626 0.90531951]
 [0.08060852 0.70609599]
 [0.38756715 0.36927417]
 [0.91520471 0.13676701]
 [0.76839496 0.62005915]
 [0.64423126 0.02879003]]
[0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
```

```
# Create neural network model
# A sequential model is one where layers are stacked one
# after another and there is not skipping, feedback and
# distributed connection

model = Sequential()

# Add a single dense layer to the model
model.add(Dense(1, activation='sigmoid', input_dim=2))      # first hidden layer
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 1)                 3

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

```
# Let's do some prediction

test = np.array([[.9,.8]])
model.predict(test,batch_size=1)
```

```
array([[0.58591753]], dtype=float32)
```

```
test = np.array([[.3,.8]])
model.predict(test,batch_size=1)
```

```
array([[0.2722925]], dtype=float32)
```

Questions?