Forward School

Program Code: J620-002-4:2020

Program Name: FRONT-END SOFTWARE DEVELOPMENT

Title : Exe31 - MNIST Handwriting Exercise

Name: Ooi Caaron

IC Number: 990701-07-5837

Date :2/8/23

Introduction : Learning MNIST digit classification

Conclusion :

The Problem: MNIST digit classification

```
We’re going to tackle a classic machine learning problem: MNIST handwritten digit
classification. It’s simple: given an image, classify it as a digit.

![image.png](attachment:image.png)


Each image in the MNIST dataset is 28x28 and contains a centered, grayscale digit.
We’ll flatten each 28x28 into a 784 dimensional vector, which we’ll use is input to our
neural network. Our output will be one of 10 possible classes: one for each digit.

Please check that you have the following packages installed (via conda or pip)
    keras tensorflow numpy mnist
```

1. Setup

In [2]:

```
!pip install mnist
```

```
Collecting mnist
  Downloading mnist-0.2.2-py2.py3-none-any.whl (3.5 kB)
Requirement already satisfied: numpy in c:\users\user\anaconda3\envs\pytho
n-dscourse\lib\site-packages (from mnist) (1.24.3)
Installing collected packages: mnist
Successfully installed mnist-0.2.2
```

In [3]:

```
#import all the required libraries
import numpy as np
import mnist
import keras

# The first time you run this might be a bit slow, since the
# mnist package has to download and cache the data.
train_images = mnist.train_images()
train_labels = mnist.train_labels()
test_images = mnist.test_images()
test_labels = mnist.test_labels()
```

Q: What's the dimension of the images data?

In [4]:

```
train_images.shape
```

Out[4]:

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

In [6]:

```
test_images.shape
```

Out[6]:

```
(10000, 28, 28)
```

Q: What's the dimension of the label data?

In [7]:

```
train_labels.size
```

Out[7]:

```
60000
```

In [8]:

```
test_labels.size
```

Out[8]:

```
10000
```

Note: Curious about the dataset? try the following code. You can play around with the image_index value.

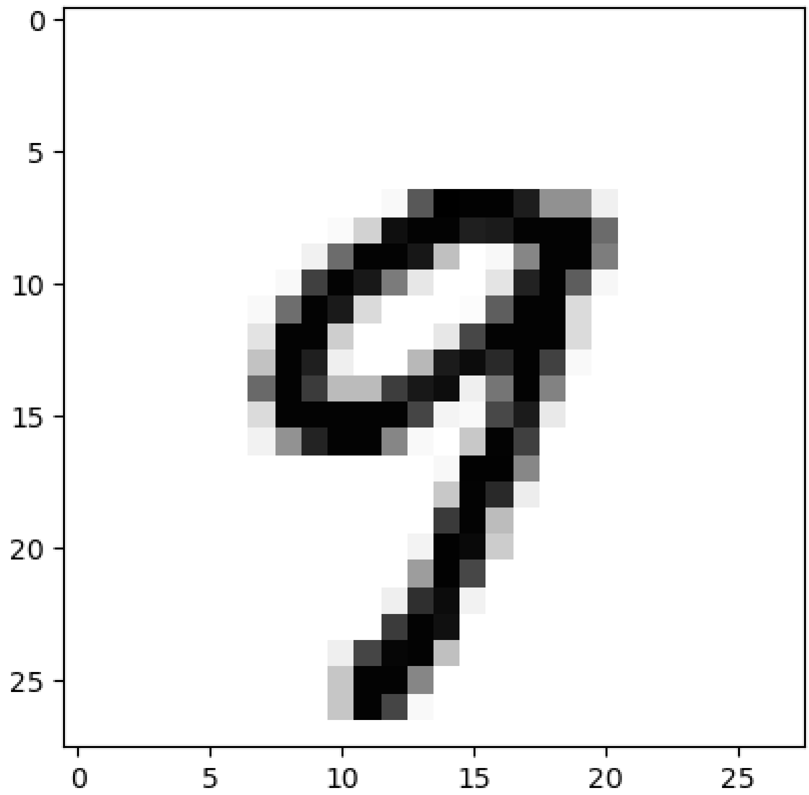
In [10]:

```
import matplotlib.pyplot as plt

image_index = 45 # You may select anything up to 60,000

print(train_labels[image_index]) # The label is 8
print(train_images[image_index])

plt.imshow(train_images[image_index], cmap='Greys')
```

```
9
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  11 185 255 253 253 230
  132 132  31   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   9  71 242 252 252 228 231 252
  252 252 167   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  28 166 252 252 235  92   0  14 142
  252 252 150   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  12 204 252 234 152  44   0   0  48 225
  252 180  16   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0  12 164 252 232  61   0   0   0   6 179 252
  252  60   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0  49 252 252  76   0   0   0  44 199 252 252
  252  60   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0  89 252 228  32   0   0  99 231 244 220 252
  203  12   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0 169 252 207  97  97 206 234 243  32 157 252
  145   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0  60 252 252 252 252 252 200  22  11 198 231
   41   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0  26 131 224 252 252 142  11   0  82 252 204
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  13 253 253 141
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  82 252 220  36
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 208 252  96   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  24 253 247  78   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0 121 253 199   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  32 216 244  26   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0 207 252 241   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  33 200 249 252  92   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  85 252 252 142   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  85 252 200  11   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]]
```

Out[10]:

```
<matplotlib.image.AxesImage at 0x1bf63a2d4d0>
```



2. Preparing the Data

As mentioned earlier, we need to flatten each image before we can pass it into our neural network. We'll also normalize the pixel values from [0, 255] to [-0.5, 0.5] to make our network easier to train (using smaller, centered values is often better).

In [16]:

```
# Normalize the images.
train_images = (train_images / 255) - 0.5
test_images = (test_images / 255) - 0.5

# Flatten the images.
train_images = train_images.reshape((-1, 784))
test_images = test_images.reshape((-1, 784))
```

Q: What's the dimension of the training and test images data?

In [11]:

```
train_images.shape
```

Out[11]:

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

In [12]:

```
test_images.shape
```

Out[12]:

```
(10000, 28, 28)
```

3. Building the Model

Every Keras model is either built using the Sequential class, which represents a linear stack of layers, or the functional Model class, which is more customizeable. We'll be using the simpler Sequential model, since our network is indeed a linear stack of layers.

Step: Start by instantiating a Sequential model.

- The first two layers have 64 nodes each and use the ReLU activation function.
- The last layer is a Softmax output layer with 10 nodes, one for each class.

Q: what's the correct input shape for your input layer?

In [17]:

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

# Define the model
model = Sequential([
  Dense(64, activation='relu'),
  Dense(64, activation='relu'),
  Dense(10, activation='softmax'),
])
```

4. Compiling the Model

Before we can begin training, we need to configure the training process. We decide 3 key factors during the compilation step:

- The optimizer. We'll stick with a pretty good default: the Adam gradient-based optimizer. Keras has many other optimizers you can look into as well.
- The loss function. Since we're using a Softmax output layer, we'll use the Cross-Entropy loss. Keras distinguishes between binary_crossentropy (2 classes) and categorical_crossentropy (>2 classes), so we'll use the latter
- A list of metrics. Since this is a classification problem, we'll just have Keras report on the accuracy metric.

Step: Compile the model using the above options - adam, categorical_crossentropy, accuracy as metrics

In [18]:

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

5. Training the Model

Training a model in Keras literally consists only of calling fit() and specifying some parameters. There are a lot of possible parameters, but we'll only manually supply a few:

- The training data (images and labels), commonly known as X and Y, respectively.
- The number of epochs (iterations over the entire dataset) to train for.
- The batch size (number of samples per gradient update) to use when training.

Step: set epochs to a suitable number, and batch_size = 32

In [19]:

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

# Train the model.
model.fit(
  train_images,
  to_categorical(train_labels),
  epochs=5,
  batch_size=32,
)
```

```
Epoch 1/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3797 -
accuracy: 0.8857
Epoch 2/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2054 -
accuracy: 0.9366
Epoch 3/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1571 -
accuracy: 0.9527
Epoch 4/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1322 -
accuracy: 0.9593
Epoch 5/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1174 -
accuracy: 0.9633
```

Out[19]:

```
<keras.src.callbacks.History at 0x1bf63b64090>
```

Q: Do you run into any problem? Why?

In [ ]:

```

```

Q: what's your achieved accuracy?

6. Testing the Model

Step: Evaluating the model by testing against the test data

In [20]:

```
model.evaluate(
  test_images,
  to_categorical(test_labels)
)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.1403 - ac
curacy: 0.9573
```

Out[20]:

```
[0.14030496776103973, 0.9573000073432922]
```

7. Using the Model

Now that we have a working, trained model, let’s put it to use. The first thing we’ll do is save it to disk so we can load it back up anytime.

Step: save the model using the save_weights function

In [16]:

```
model.save_weights('model.h5')
```

In [17]:

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

# Build the model.
model = Sequential([
  Dense(64, activation='relu', input_shape=(784,)),
  Dense(64, activation='relu'),
  Dense(10, activation='softmax'),
])

# Load the model's saved weights.
model.load_weights('model.h5')
```

8. Predict

Using the trained model to make predictions is easy: we pass an array of inputs to predict() and it returns an array of outputs. Keep in mind that the output of our network is 10 probabilities (because of softmax), so we'll use np.argmax() to turn those into actual digits.

In [18]:

```
# Predict on the first 5 test images.
predictions = model.predict(test_images[:5])

# Print our model's predictions.
print(np.argmax(predictions, axis=1)) # [7, 2, 1, 0, 4]

# Check our predictions against the ground truths.
print(test_labels[:5]) # [7, 2, 1, 0, 4]
```

```
[7 2 1 0 4]
[7 2 1 0 4]
```

Note: What's the difference between model.save_weights and model.save? - [https://stackoverflow.com/questions/42621864/difference-between-keras-model-save-and-model-save-weights#:~:text=save()%20saves%20the%20weights,to%20HDF5%20and%20nothing%20else (https://stackoverflow.com/questions/42621864/difference-between-keras-model-save-and-model-save-weights#:~:text=save()%20saves%20the%20weights,to%20HDF5%20and%20nothing%20else)](https://stackoverflow.com/questions/42621864/difference-between-keras-model-save-and-model-save-weights#:~:text=save()%20saves%20the%20weights,to%20HDF5%20and%20nothing%20else).

This exercise is adapted from https://victorzhou.com/blog/keras-neural-network-tutorial/ (https://victorzhou.com/blog/keras-neural-network-tutorial/)

Challenge 1:

Retrain your model by using different network depths - what will you conclude?

In [ ]:

```

```

Challenge 2:

Retrain your model by using different activation (other than ReLU) - what differences does it make?

In [21]:

```
model = Sequential([
    Dense(64, activation='sigmoid', input_shape=(784,)),
    Dense(64, activation='sigmoid'),
    Dense(10, activation='softmax'),
])
```

Challenge 3:

Fit your model using validation_data option - What differences will that bring?

In [22]:

```
from tensorflow.keras.optimizers import Adam

model.compile(
    optimizer=Adam(lr=0.005),
    loss='categorical_crossentropy',
    metrics=['accuracy'],
)

model.fit(
    train_images,
    to_categorical(train_labels),
    epochs=5,
    batch_size=32,
    validation_data=(test_images, to_categorical(test_labels))
)
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_r
ate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.

Epoch 1/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.5660 -
accuracy: 0.8584 - val_loss: 0.2610 - val_accuracy: 0.9253
Epoch 2/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.2313 -
accuracy: 0.9316 - val_loss: 0.1967 - val_accuracy: 0.9418
Epoch 3/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1713 -
accuracy: 0.9491 - val_loss: 0.1673 - val_accuracy: 0.9494
Epoch 4/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1349 -
accuracy: 0.9599 - val_loss: 0.1424 - val_accuracy: 0.9576
Epoch 5/5
1875/1875 [==============================] - 5s 3ms/step - loss: 0.1141 -
accuracy: 0.9652 - val_loss: 0.1301 - val_accuracy: 0.9597
```

Out[22]:

```
<keras.src.callbacks.History at 0x1bf62ebacd0>
```

Challenge 4:

How will you load your saved weights to use it in a separate code? Upload your saved model/weights, and compare your model/weights with a model/weights from one of your classmate's.

In [ ]:

```
model.save_weights('model.h5')
```

Challenge 5:

How can you load any image from the data set and let your model (or your classmate's) to predict the image?

In [23]:

```
predictions = model.predict(test_images[:5])
print(np.argmax(predictions, axis = 1))
print(test_labels[:5])
```

```
1/1 [==============================] - 0s 124ms/step
[7 2 1 0 4]
[7 2 1 0 4]
```

In [ ]:

```

```