Yahya Qteishat, EECS @ Diablo Valley College

MNIST Handwritten Digit Classification

Summary Abstract: This past winter break, I decided to explore machine learning (ML). After completing a ML w/ Python bootcamp from Udemy, I was prepared to undertake my first technical project with Tensorflow: Implementing an artificial neural network to classify handwritten digits from the MNIST data set. This is common for beginners to ML. **A final accuracy of 96% was reported**.

Background: In a nutshell, artificial neural networks (ANNs) are a collection of interconnected perceptrons, (mathematical representations of biological neurons). These perceptrons take in an input, multiply it by weight and add it to a bias. An activation function takes in that product and sets boundaries on the output of that perceptron. Which then becomes the input for the next perceptron(s).

The output of the ANN is then evaluated with a loss function, which compares the model's predictions to the correct labels of the training data. The more incorrect predictions, the higher the loss. The goal is to minimize loss, so an ANN "learns" by going back and adjusting the weights and biases in the ANN. This is back propagation. The model would then traverse the training data & make new predictions.
The mathematics involved in minimizing loss is known as gradient descent.

Implementation & Thought Process:

```
import numpy as np # for data manipulation
import pandas as pd # for data manipulation

import matplotlib.pyplot as plt # for data visualization
%matplotlib inline # Allows us to print in Jupyter.

import tensorflow as tf # machine learning library.
```

Tensorflow (& the accompanying Keras API) greatly abstract the process behind implementing an ANN in python.

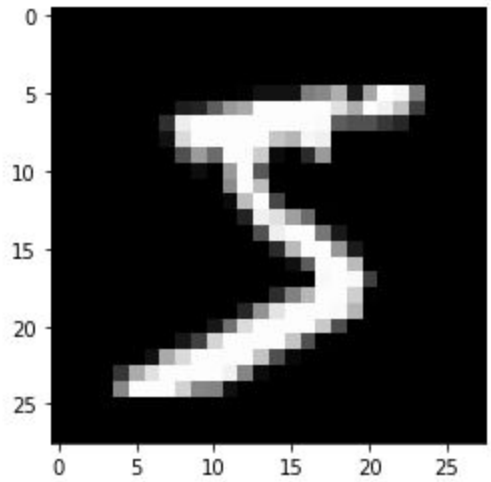
```
MNIST = tf.keras.datasets.mnist # import our data

(X_train, y_train), (x_test, y_test) = MNIST.load_data()
''' We split our data into a training set to fit the model
onto, and a testing set to evaluate the model with. '''
```

X is the image.
Y is the number represented.

```
plt.imshow(X_train[0],cmap='gist_gray')
```

```
<matplotlib.image.AxesImage at 0x15ff8e1f0>
```



The MNIST data set is a collection of 28x28 pixel images of handwritten digits ranging from 0 to 9. Each image is represented by a large array of numbers. Each value in said array represents how luminous an individual pixel is, values range from (0-255).

Here is the 1st image in the training set visualized using matplotlib.

```
''' Normalizing our data: pixel values in each array
will now range from 0-1. This is a common practice.'''
X_train = tf.keras.utils.normalize(X_train, axis=1)
x_test = tf.keras.utils.normalize(x_test, axis=1)
```

```
#Defining our model using Tensorflow / Keras:

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten

model = Sequential() # defining model object as a sequence of layers.

#input layer of ANN:
model.add(Flatten()) ''' The "Flatten" layer converts the 2D
array of pixel values (the "image") into a one dimensional series.'''

# hidden layers of ANN:
'''hidden layers: "Dense" meaning each neuron in one layer is
connected to every neuron in the next layer.
A recitified linear unit is a common activation function '''

model.add(Dense(64, activation='relu')) # 64 neurons
model.add(Dense(32, activation='relu')) # 32 neurons

#output layer of ANN:
''' We are preforming multi class classfication, therefore we
need an output neuron for each number an image could represent, 10.

Each class is mutually exclusive, therefore we use softmax as our
activation function. Each output neuron will represent the
probability that an image was the number ranging from 0-9. '''

model.add(Dense(10, activation='softmax')) # 10 neurons

'''We will optimize gradient descent with the adam optimizer.
Cross entropy is a common loss function for classfication.
We will keep track of the accuracy metric whilst parsing data'''

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

```
model.fit(X_train, y_train, epochs = 3)
```

```
Epoch 1/3
1875/1875 [==============================]
Epoch 2/3
1875/1875 [==============================]
Epoch 3/3
```

Here we "fit" the training data onto the model, passing in images & their labels. We then pass in unlabeled data for the model to make predictions on.

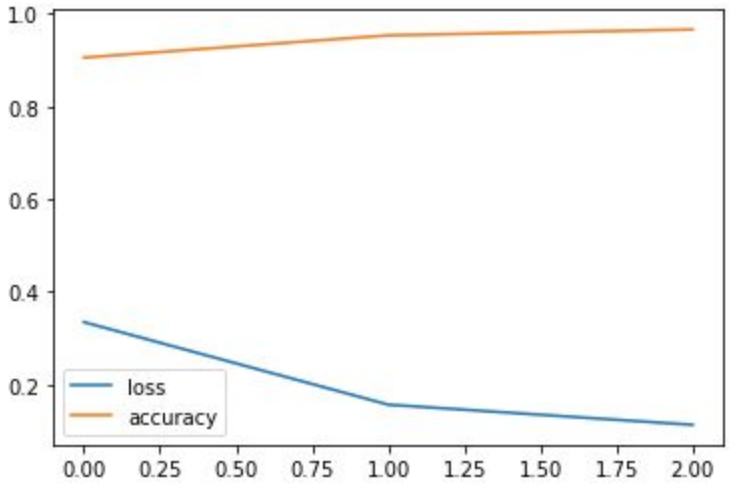
```
predictions = np.argmax(model.predict(x_test), axis=-1)
```

Yahya Qteishat, EECS @ Diablo Valley College

Results:

```
losses = pd.DataFrame(model.history.history)
losses.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1
```



We can see that as the model "learns" via back propagation & linear descent, loss decreases and accuracy increases.

One could definitely tinker with an ANN's architecture much further. The number of layers can vary.

Here we parsed through the data 3 times, however the higher the number of times we parse the data, we run the risk of loss increasing again. This is "overfitting" and can be combated by adding a dropout layer. Which deactivates neurons as soon as loss begins to increase again.

```
from sklearn.metrics import classification_report
print(classification_report(y_test,predictions))
```

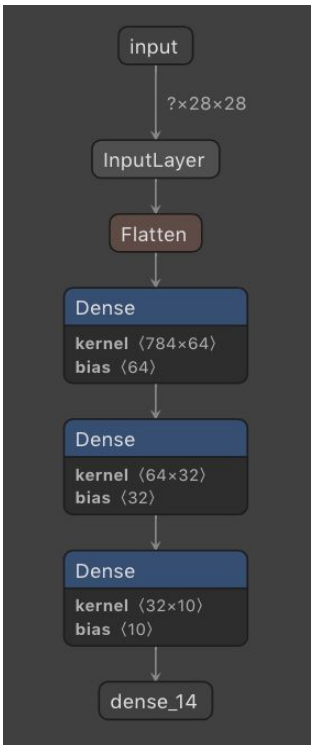
	precision	recall	f1-score	support
0	0.97	0.98	0.98	980
1	0.98	0.98	0.98	1135
2	0.96	0.97	0.96	1032
3	0.95	0.96	0.95	1010
4	0.96	0.96	0.96	982
5	0.96	0.96	0.96	892
6	0.96	0.97	0.97	958
7	0.97	0.97	0.97	1028
8	0.94	0.95	0.95	974
9	0.98	0.92	0.95	1009
accuracy			0.96	10000
macro avg	0.96	0.96	0.96	10000
weighted avg	0.96	0.96	0.96	10000

Although there are multiple metrics to gauge an ANN's performance at classification, accuracy is the most relevant to us in this context.

It is the percentage of correct predictions.

Visualization and Summary:

We can export our model as a .h5 file using "model.save()" and passing in a name for our model. I used *Netron*, an open source neural network visualization tool to describe our ANN's architecture.



Doing this project definitely piqued my interest in the field of Machine Learning. Thank you for reading!!