Simple Machine Learning Tutorial

Mark Harvey

MNIST Fashion Tutorial

This is a quick "Hello World" type of machine learning project and is a modified version of the Tensorflow tutorial found here: https://www.tensorflow.org/tutorials/keras/basic_classification - I found it useful to test my Anaconda installation and virtual environment.

I ran this on Win 10 without a GPU but it should work the same under Linux.

To run this tutorial, you will need to have installed Anaconda and created a virtual environment – see https://github.com/foolmarks/tf_setup

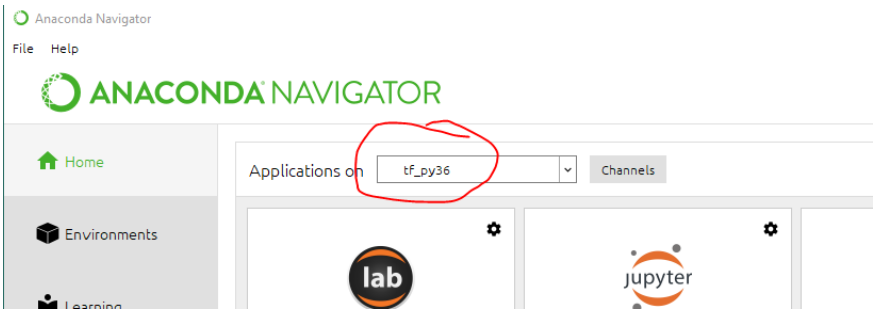
The implemented neural network will classify images from the MNIST Fashion dataset into one of 10 classes.

Preparation

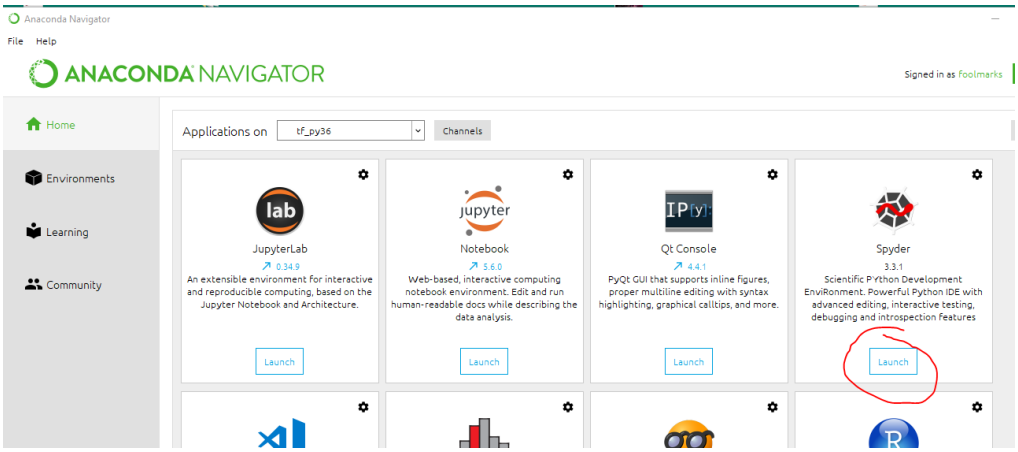
Download the demo from https://github.com/foolmarks/mnist_fashion. Once you have unzipped the archive or cloned it with git, you should have this document and a python source code file called fashion.py.

Running the demo

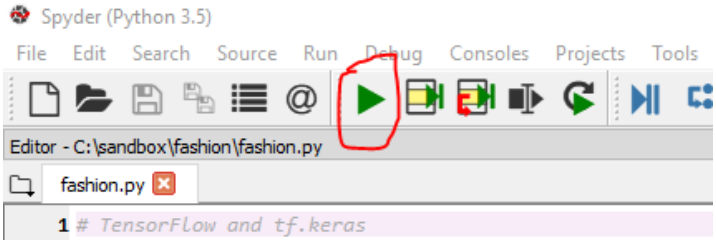
1. Open the Anaconda Navigator.
2. Click the Anaconda "Home" icon (top left) and make sure the "Applications on" field is set to the name of your virtual environment that contains Tensorflow. If you followed the previous tutorial to the letter, it will be called tf_py36:



3. Open Spyder by clicking on its “Launch” button (..the first time round, you may need “Install” then “Launch”):



4. From the menu: File -> Open and the browse to where you saved the fashion.py file and select it. The fashion.py python script should open in the text editor window. Run by clicking on the big green “Run” button:



5. If everything has gone well, you should see the following output in the Spyder console:

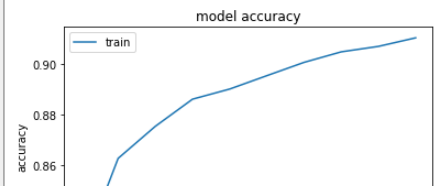
IPython console

Console 1/A

```
Python 3.6.6 |Anaconda, Inc.| (default, Jun 28 2018, 11:27:44) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 6.5.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/ml/mnist_fashion/code/fashion.py', wdir='C:/ml/mnist_fashion/code')
Training set images shape: (60000, 28, 28)
Training set labels shape: (60000,)
Test set images shape: (10000, 28, 28)
Test set labels shape: (10000,)
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 128)               100480
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/10
60000/60000 [==============================] - 2s 41us/step - loss: 0.5013 - acc: 0.8244
Epoch 2/10
60000/60000 [==============================] - 2s 37us/step - loss: 0.3774 - acc: 0.8628
Epoch 3/10
60000/60000 [==============================] - 2s 37us/step - loss: 0.3396 - acc: 0.8755
Epoch 4/10
60000/60000 [==============================] - 2s 41us/step - loss: 0.3134 - acc: 0.8862
Epoch 5/10
60000/60000 [==============================] - 2s 38us/step - loss: 0.2981 - acc: 0.8902
Epoch 6/10
60000/60000 [==============================] - 2s 36us/step - loss: 0.2813 - acc: 0.8955
Epoch 7/10
60000/60000 [==============================] - 2s 37us/step - loss: 0.2675 - acc: 0.9008
Epoch 8/10
60000/60000 [==============================] - 2s 37us/step - loss: 0.2554 - acc: 0.9050
Epoch 9/10
60000/60000 [==============================] - 2s 37us/step - loss: 0.2476 - acc: 0.9072
Epoch 10/10
60000/60000 [==============================] - 2s 36us/step - loss: 0.2392 - acc: 0.9105
```



What's in the code?

As is usual for any python source code, the first few lines of fashion.py import some external modules and gives them some short-hand aliases for convenience. Note how keras is imported from tensorflow – keras has been included in tensorflow for several releases now. Numpy is needed for a math function used near the end of the code. Matplotlib is a 2D plotting library – the pyplot API makes a series of MATLAB-like commands available which we will use the plot some figures.

```
# Import modules
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
```

The datasets API from keras is used to load the fashion_mnist dataset. Note that there is no reference to an external URL or file – the dataset is included with keras. The fashion mnist dataset contains 70,00 images – each image is 28x28 pixels and each pixel is 8-bit grayscale with values from 0 to 255. The complete datset is split into 60,000 images for training and 10,000 images for testing. Each image can fall into one of ten categories – so our dataset has ten 'labels'.

```
# use the Fashion dataset provided by Keras
# Dataset of 60,000 28x28 grayscale images of 10 fashion categories, along with a
test set of 10,000 images.

fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

# create a python list of the 10 fashion categories (class labels)
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal',
'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

Just for reference, we print out some information about the dataset:

```
# Shapes of training set
print("Training set images shape: {shape}".format(shape=train_images.shape))
print("Training set labels shape: {shape}".format(shape=train_labels.shape))


# Shapes of test set
print("Test set images shape: {shape}".format(shape=test_images.shape))
print("Test set labels shape: {shape}".format(shape=test_labels.shape))
```

Then we do our first serious operation on the datasets – every image is 'normalized', that means that we scale the pixel values such that they lie between 0 and 1 rather than 0 and 255. This is done by simply dividing by 255:

```
# pixel values are 0:255, normalize to range of 0:1
train_images = train_images / 255.0
test_images = test_images / 255.0
```

Now we move onto the definition of the neural network itself. This is a simple network with 3 layers and is defined using the Keras Sequential model. The input layer 'flattens' the incoming 28x28 images into a 784-bit vector.

The second, hidden layer is a 128-node Dense (i.e. fully-connected) layer with a ReLU activation layer to introduce a non-linearity. Each node will calculate the dot product of its inputs and the weights matrix and add a bias. Then the activation function will be applied. Each dense layer has weights that are initialized using a Xavier/Glorot uniform initializer (the default) which selects weights from a uniform distribution.

The third, output layer is a 10-node Dense layer with softmax activation – 10 nodes because we have 10 classes in our dataset, so we need 10 outputs.

```
# 1st layer flattens 28x28 array into 784 pixel vector
# 2nd layer is fully-connected 128 node layer with ReLu activation
# output layer is fully-connected 10-node softmax layer
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])

# print a summary of the model
print(model.summary())
```

Now we configure the training/learning step by defining the optimizer and loss functions.

```
# Adam optimizer to change weights & biases
# Loss function is sparse categorical crossentropy
model.compile(optimizer=tf.train.AdamOptimizer(),
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

The model is trained using the Keras fit function. The batch size defines the number of samples that are processed during training before an update of the model is done. The number of epochs defines the number of times the training dataset will be passed through the model.

We then use the matplotlib.pyplot module (aliased as plt) to plot the history object that is retuned by the fit method.

```
# the keras fit function trains the model and returns a
# history object whose .history attribute is a python
# dictionary of training and loss metrics

# train the model
history = model.fit(train_images, train_labels, batch_size=32, epochs=5)


# plot history for accuracy
plt.plot(history.history['acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

# plot history for loss
plt.plot(history.history['loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

We then use the matplotlib.pyplot module (aliased as plt) to plot the history object that is retuned by the fit method.

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)
```

The finally we can make some actual predictions using our trained model. The .predict() method returns a numpy array – in our case, 'predictions' will be set to an array of 10 elements. Each array element is probability that the input sample belongs to one of the 10 classes. We need to use the numpy argmax function to obtain the highest value of the array, which is our prediction:

```
print("\nLet's make some predictions with the trained model..")
predictions = model.predict(test_images)

# each prediction is an array of 10 values
# the max of the 10 values is the model's
# highest "confidence" classification

# use numpy argmax function to get highest of the set of 10

print("Predict 1st sample in the test set is: {pred}".format(pred=class_names[np.argmax(predictions[0])]))
print("The 1st sample in test set actually is: {actual}".format(actual=class_names[test_labels[0]]))

print("Predict 5th sample in the test set is: {pred}".format(pred=class_names[np.argmax(predictions[4])]))
print("The 5th sample in test set actually is: {actual}".format(actual=class_names[test_labels[4]]))
```