SLIIT
Discover Your Future

BSc in Information Technology
Specialising in Data Science

Semester 2

Lab Exercise 3

IT3071 – Machine Learning and Optimization Methods 2023

Lab 3: Introduction to Tensorflow

Objective:

- To obtain a knowledge in Deep Learning libraries Tensorflow and Keras

Content:

- TensorFlow 2.0 Introduction
- Neural Networks with Keras

Reduce the code size.
Important in deep learning.
Faster than tensorflow.
When using large data sets, it is better to use Keras on the top of tensorflow.

Introduction and Setup

We will discuss the following topics within the TensorFlow module:

- TensorFlow Install and Setup
- Representing Tensors
- Tensor Shape and Rank
- Types of Tensors

If you'd like to you can install TensorFlow on your machine or you can use Google Collaboratory (colab). Collaboratory is a free Jupyter notebook environment that requires no setup and runs entirely in the cloud.

Installing TensorFlow (only need if you decide to work in your own machine rather than using colab)

To install TensorFlow on your local machine you can use pip:

```
! pip install tensorflow
$ pip install tensorflow
```

If you have a CUDA enabled GPU you can install the GPU version of TensorFlow. You will also need to install some other software which can be found here: https://www.tensorflow.org/install/gpu

pip install tensorflow-gpu

Importing TensorFlow

```
%tensorflow_version 2.x  # Check if your version of tensorflow is 2.0. Not required unless you are in a notebook

import tensorflow as tf  # now import the tensorflow module
print(tf.version)        # make sure the version is 2.x
```

Can store data in more than 2 dimensions.

Tensor

"A tensor is a generalization of vectors and matrices to potentially higher dimensions. Internally, TensorFlow represents tensors as n-dimensional arrays of base data types." (https://www.tensorflow.org/guide/tensor)

Tensors are the main objects that are passed around and manipulated throughout the program. Each tensor represents a partialy defined computation that will eventually produce a value. TensorFlow programs work by building a graph of Tensor objects that details how tensors are related. Running different parts of the graph allow results to be generated.

Note that Graphs are used by tf.functions to represent the function's computations. Each graph contains a set of tf.Operation objects, which represent units of computation; and tf.Tensor objects, which represent the units of data that flow between operations. Graphs do not evaluate the function, but represent the computations that were defined.

Each tensor has a data type and a shape.

Data Types Include: float32, int32, string and others.

Shape: Represents the dimension of data.

Just like vectors and matrices tensors can have operations applied to them like addition, subtraction, dot product, cross product etc.

Creating Tensors

Below is an example of how to create some different tensors.

Simply define the value of the tensor and the data type. And usually we deal with tensors of numeric data, it is quite rare to see string tensors.

```
string = tf.Variable("this is a string", tf.string)
number = tf.Variable(324, tf.int16)
floating = tf.Variable(3.567, tf.float64)
```

For a full list of data types please refer to the following guide:
https://www.tensorflow.org/api_docs/python/tf/dtypes/DType?version=stable

Rank/Degree of Tensors

Rank/ degree mean the number of dimensions involved in the tensor. What we created above is a tensor of rank 0, also known as a scalar.

Now we'll create some tensors of higher degrees/ranks.

```
rank1_tensor = tf.Variable(["Test"], tf.string)
rank2_tensor = tf.Variable([["1,2], [3, 4]], tf.int16)
```

To determine the rank of a tensor we can call the following method.

```
tf.rank(rank2_tensor)
```

The rank of a tensor is directly related to the deepest level of nested lists. You can see in the first example ["Test"] is a rank 1 tensor as the deepest level of nesting is 1. Where in the second example [[1,2], [3,4]] is a rank 2 tensor as the deepest level of nesting is 2.

Shape of Tensors

The shape of a tensor is simply the number of elements that exist in each dimension. TensorFlow will try to determine the shape of a tensor but sometimes it may be unknown.

To get the shape of a tensor we use the shape attribute.

```
rank2_tensor.shape
```

Changing Shape

The number of elements of a tensor is the product of the sizes of all its shapes. There are often many shapes that have the same number of elements, making it convenient to be able to change the shape of a tensor.

The example below shows how to change the shape of a tensor.

```
tensor1 = tf.ones([1,2,3])                  # tf.ones() creates a shape [1,2,3] tensor full of ones
tensor2 = tf.reshape(tensor1, [2,3,1])      # reshape existing data to shape [2,3,1]
tensor3 = tf.reshape(tensor2, [3, -1])      # -1 tells the tensor to calculate the size of the dimension in that
                                            place and this will reshape the tensor to [3,3]
```

The number of elements in the reshaped tensor MUST match the number in the original

Now let's have a look at our different tensors.

```
print(tensor1)
print(tensor2)
print(tensor3)
```

Slicing Tensors

The slice operator can be used on tensors to select specific axes or elements.

When we slice or select elements from a tensor, we can use comma separated. Each subsequent value references a different dimension of the tensor.

Ex: tensor[dim1, dim2, dim3]

```
# Creating a 2D tensor
matrix = [[1,2,3,4,5],
          [6,7,8,9,10],
          [11,12,13,14,15],
          [16,17,18,19,20]]

tensor = tf.Variable(matrix, dtype=tf.int32)
print(tf.rank(tensor))
print(tensor.shape)
```

let's select some different rows and columns from our tensor

```
three = tensor[0,2] # selects the 3rd element from the 1st row
print(three)  # -> 3

row1 = tensor[0] # selects the first row
print(row1)

column1 = tensor[:, 0] # selects the first column
print(column1)
```

: -> represent each and every list inside the list

```
column_1_in_row_2_and_3 = tensor[1:3, 0]
print(column_1_in_row_2_and_3)
```

Types of Tensors

There are different types of tensors. These are the most used.

- Variable
- Constant
- Placeholder
- SparseTensor

With the exception of Variable all these tensors are immutable, meaning their value may not change during execution.

Creating a Neural Network

This guide is based off of the following TensorFlow tutorial.
https://www.tensorflow.org/tutorials/keras/classification

Imports

```
# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt
```

Dataset

We will use the MNIST Fashion Dataset. This is a dataset that is included in keras.
This dataset includes 60,000 images for training and 10,000 images for validation/testing.

```
fashion_mnist = keras.datasets.fashion_mnist      # load dataset

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data() # split into tetsing and training
```

Let's have a look at this data to see what we are working with.

```
train_images.shape
```

So we've got 60,000 images that are made up of 28x28 pixels (784 in total).

```
train_images[0,23,23]  # look at one pixel
```

Our pixel values are between 0 and 255, 0 being black and 255 being white.

```
train_labels[:10]  # look at the first 10 training labels
```

Let's create an array of label names to indicate which is which.

```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat','Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

let's look at what some of these images look like.

```
plt.figure()
plt.imshow(train_images[1])
plt.colorbar()
plt.grid(False)
plt.show()
```

Data Preprocessing

The last step before creating our model is to preprocess our data. This simply means applying some prior transformations to our data before feeding it the model. In this case we will simply scale all our greyscale pixel values (0-255) to be between 0 and 1. We can do this by dividing each value in the training and testing sets by 255.0. We do this because smaller values will make it easier for the model to process our values.

```
train_images = train_images / 255.0
test_images = test_images / 255.0
```

Building the Model

Creating the architecture of the neural network.

We are going to use a keras sequential model with three different layers. This model represents a feed-forward neural network (one that passes values from left to right).

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),      # input layer (1)
    keras.layers.Dense(128, activation='relu'),      # hidden layer (2)
    keras.layers.Dense(10, activation='softmax')     # output layer (3)
])
```

28 x 28 neurones should be there in the input layer.

Layer 1: This is our input layer and it will consist of 784 neurons. We use the flatten layer with an input shape of (28,28) to denote that our input should come in in that shape. The flatten means that our layer will reshape the shape (28,28) array into a vector of 784 neurons so that each pixel will be associated with one neuron.

We are flatting the 2d neurone into 1d. 1 pixel for 1 neurone.

Use more neurones to make it faster.

Layer 2: This is our first and only hidden layer. The dense denotes that this layer will be fully connected and each neuron from the previous layer connects to each neuron of this layer. It has 128 neurons and uses the rectify linear unit activation function.

Layer 3: This is our output later and is also a dense layer. It has 10 neurons that we will look at to determine our models output. Each neuron represents the probability of a given image being one of the 10 different classes. The activation function softmax is used on this layer to calculate a probability distribution for each class. This means the value of any neuron in this layer will be between 0 and 1, where 1 represents a high probability of the image being that class.

Compile the Model

The last step in building the model is to define the loss function, optimizer and metrics we would like to track.

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

Training the Model

```
model.fit(train_images, train_labels, epochs=10) # we pass the data, labels and epochs
```

--> we are running the model for 10 times (forward and back) With epoches, accuracy will be increased.

Evaluating the Model

We can evaluate easily using another built-in method from keras.

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

Making Predictions

To make predictions we simply need to pass an array of data in the form we've specified in the input layer to .predict() method.

```
predictions = model.predict(test_images)
```

This method returns to us an array of predictions for each image we passed it. Let's have a look at the predictions for image 1

```
predictions[0]
```

If we wan't to get the value with the highest score we can use a useful function from numpy called argmax(). This simply returns the index of the maximium value from a numpy array.

```
np.argmax(predictions[0])
```

And we can check if this is correct by looking at the value of the cooresponding test label.

```
test_labels[0]
```