Tensor Processor Units
Laboratory

Goal

- Get to know tensors in TensorFlow
- Evaluate the performance of TPU on a simple classification and compare it to GPU and CPU.
- Familiarise with convolutions in TensorFlow

Material

In this laboratory, we will make use of Google Colab at `https://colab.research.google.com/`.

You need to log with your Google account. A free plan is sufficient to answer all questions of this laboratory.

1 Tensors

The aim of this section is to get to know tensors in TensorFlow.

TensorFlow is based on the concept of dataflow (= computation) graphs. Nodes represent data or a mathematical operation and edges stand for flow of data between nodes. These data are represented by tensors, i.e. multidimensional arrays of any dimension.

To clarify some vocabulary[1], we call:

- *Shape*: length, i.e. number of elements, of each of the axes of a tensor
- *Rank*: number of tensor axes (A scalar has rank 0, a vector has rank 1, a matrix has rank 2.)
- *Axis* = *Dimension*: particular dimension of a tensor
- *Size*: total number of items in the tensor.

As you will progressively discover new functions used in TensorFlow, do not forget to consult their documentation in order to use them properly. You can use for example the following function:

```
help(functionName)
```

[1] `https://www.tensorflow.org/guide/tensor`

1.1 Importing TensorFlow

To use TensorFlow, we import the library as follows:

```
import tensorflow as tf
```

1.2 Tensors

To define a constant, we use the *constant* function as follows:

```
tf.constant(value, name='constant')
```

The function creates a constant tensor, i.e. a scalar or rank-0 tensor, with a given value.

Question 1: Define two constants a and b. Their values are respectively 8 and 5.

Using the function *print*, verify that constant tensors were created. What information can you get?

Similarly, we can create any tensor using the *constant* function. A vector has one axis and is represented as a list of values. A matrix has two axis and is represented as a list of lists of values.

Question 2: Define two vectors X and Y, such as

$$X = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \text{ and } Y = \begin{pmatrix} 4 \\ 5 \\ 6 \end{pmatrix}$$

Define two matrices A and B, such as

$$A = \begin{pmatrix} 1 & 2 \\ 4 & 5 \end{pmatrix} \text{ and } B = \begin{pmatrix} 3 & 4 \\ 6 & 7 \end{pmatrix}$$

Define 3-dimensional tensor with the shape 3x2x5.

Use the *shape* and *ndim* functions to verify the shape of the data structure and the number of axes.

Define 4-dimensional tensor with the shape 4x3x6x3 containing only 0. You can use the *zeros* function.

1.3 Basic mathematical operations

TensorFlow allows you to carry out basic mathematical operations, such as addition (*add*), subtraction (*subtract*), element-wise multiplication (*multiply*), and matrix multiplication (*matmul*).

Question 3: Using the tensors already defined, carry out several mathematical operations. Verify that results are correct.

1.4 Reshapping

It is also possible to reshape a tensor into a new shape, using the *reshape* function.

You can use -1 parameter in the *reshape* function to let TensorFlow choose the correct value, which fits.

Question 4: Reshape your 3-dimensional tensor. Consider for example shapes 10x3 and 2x15.

Try to type the shape 6x(-1). What do you observe?

1.5 Variables

As tensor objects are immutable, TensorFlow uses variables to share and persist some statistics that are manipulated by the program, such as model weights. Variables are created and tracked via the `tf.Variable` class. To create a variable, we type

```
tf.variable(tensor)
```

The created variable will have the same data type as the initialisation value.

To assign a variable to the tensor, we use the *assign* function.

Question 5: Create one boolean variable and one complex variable.

1.6 Your own function

If you want to create your own function, you need to use the *tf.function* function in order to separate your pure-TensorFlow code from Python. The following example shows a user function adding two constants a and b.

```
@tf.function
def my_function(a,b):
    c = ... # To be completed in Question 6
    return c
```

Question 6: Complete the above mentioned function in order to add two constants a and b and return the result c.

2 TPU Performance

The aim of this section is to evaluate the performance of TPU on a simple classification and compare them to GPU and CPU.

The exercise makes use of MNIST dataset of handwritten digits, and it trains a model on a chosen resource (TPU, T4 GPU or CPU) to make predictions later.

To carry out this exercise, we will make use of educational example proposed by TensorFlow. This example can be directly executed in the Google Colab environment. Note, that we will carry out only the first five learning objectives.

Go to `https://colab.research.google.com/github/tensorflow/tpu/blob/master/tools/colab/keras_mnist_tpu.ipynb?hl=fr` and make a copy of the code to your Google Drive. There is a button that you can use directly.

Question 1: Get to know the code.
Identify the code section related to detection of available resource (TPU, T4 GPU or CPU)

Question 2: What are the parameters used to train the model? In particular, how is the variable "BATCH_SIZE" defined?

Question 3: Find the model definition. What is the model structure? How many parameters does it contain?

Question 4: Identify the sections in the code related to model training and evaluation.

Question 5: For each resource (TPU, T4 GPU or CPU),

- Measure the time required for training and test.
- Evaluate the model performance (loss and accuracy).
- How many epochs are considered? Why this number varies?
- What can you conclude?

Do not hesitate to plot graphics to visualize the data to answer the previous questions.

Question 6: Try to change the size of the variable "BATCH_SIZE" using the TPU. What do you observe?

3 Principle of the convolution

The aim of this section is to manipulate convolutions in TensorFlow.

3.1 Convolution in 1D

In this exercise, we consider an input x and a kernel h. We compute a one-dimensional convolution, which is defined as follows:

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot h[n-k] \tag{1}$$

Question 1: Let assume that

```
h = [4, 3, 1]
x = [3, 2, 5]
```

Manually compute the convolution between x and h.

Question 2: In Python, the convolution operation is implemented by the `numpy.convolve` function. Verify that your results in Question 1 are correct.

The `numpy.convolve` function allows user to choose one of the three modes: *full*, *same*, or *valid*. Look at the documentation of this function to see the difference among these modes.

Question 3: Compute three convolution operations using three different modes. What do you observe?

3.2 Convolution in 2D

Now, we consider an input I and a kernel h and we compute a two-dimensional convolution, which is defined as follows:

$$y[m,n] = x[m,n] * h[m,n] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} x[i,j] \cdot h[m-i, n-j] \tag{2}$$

Question 4: Let assume that

$$X = \begin{pmatrix} 155 & 111 & 204 \\ 142 & 78 & 3 \\ 183 & 146 & 140 \end{pmatrix} \text{ and } H = \begin{pmatrix} 1 & 2 \\ 2 & -1 \end{pmatrix}$$

Compute the convolution between X and H, using the `signal.convolve` function available in the *scipy* library. Observe the difference among three modes: *full*, *same*, or *valid*.

3.3 Convolution in TensorFlow

We will carry out the two-dimensional convolution in TensorFlow. We consider that we have 10x10 image, i.e. 4D tensor (batch size, width, height, number of channels), and 3x3 filter, which is also a 4D tensor (width, height, number of channels, number of filters).

Question 5: Define two variables X and H that are randomly sampled from a normal distribution as follows:

```
X = tf.Variable(tf.random.normal([1, 10, 10, 1]))
H = tf.Variable(tf.random.normal([3, 3, 1, 1]))
```

Question 6: Consult the documentation of the `tf.nn.conv2d` function.

We consider that a stride equals $[1, 1, 1, 1]$ and that the type of padding is set respectively at *same* first and then at *valid*. Compute the convolution of X and H.

3.4 Application of the convolution

In this part, we will apply the convolution to images.

Question 7: Choose an image and upload it to your code:

```
from PIL import Image
image_colour = Image.open('imageName.jpg')
```

If you encounter difficulties to access your image, try the following lines:

```
from google.colab import drive
drive.mount('/content/drive')
```

Then, type the following code to (1) convert colour images into black and white, (2) convert image to a matrix with values ranging from 0 to 255, and (3) plot the image.

```
image_BW = image_colour.convert("L")
image_array = np.asarray(image_BW)
image_plot = plt.imshow(image_array)
image_plot.set_cmap('gray')
plt.show(image_plot)
```

Question 8: Let consider a filter

$$X = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Compute the two-dimensional convolutional using the `signal.convolve2d` function with *mode* set at *same* and *boundary* fixed at *symm*.

Plot the new image. What do you observe?

Question 9: To add a bias, we consider the following code:

```
image_biases = np.absolute(image_after_convolution) + 100
image_biases[image_biases > 255] = 255
```

Plot the new image. What do you observe?

Question 10: Choose an image of a handwritten digit, for example on `https://ibm.box.com/shared/static/vvm1b63uvuxq88vbw9znpwu5ol380mco.jpg` and repeat Questions from 7 to 9.

What can you conclude?