Introduction to Neural Networks
PW1: Neural network based classification with Pytorch Keras

Requirements for the work: the exercise 7 should be uploaded on the DVO repository before the deadline. You should create your colab file correctly named.

1 Introduction

We are interesting to apply a neural network model to a supervised machine learning problem and we need five elements:

1. an annotated dataset and its related problem
2. an architecture well adapted to the dataset
3. a loss function
4. a learning algorithm allowing us to minimize the loss function
5. a programming environment dedicated to easily implement the algorithms

Exercice 1: quick strat with GoogleColab Environment

Colab (or "Colaboratory") allows you to write and execute Python code in your browser with 1)No configuration requirement; 2)Free access to GPUs; 3)Easy sharing the code.

Step1: Create your own colab

Open a new colab in GoogleColab for Python3 by starting a new environment from `https://colab.research.google.com`. You can name your file as your $Name\_LastName\_TP01.ipynb$

Step2: Import APIs

Two concurrent high level API exist allowing us to built, train and test different models based on TensorFlow or Pytorch. Let discover these two APIs by doing. The first step that we need is ***Working with data***, let we download the dataset with PyTorch and TensorFlow then we compare the source code in between.

Pytorch API

PyTorch provides the designed modules and classes as **torch.nn**, **torch.optim**, **Dataset**, and **DataLoader** to help us creating and training neural networks.

Import API Pythorch

```
import torch
from torch import nn
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision.transforms import ToTensor
```

TensorFlow API and Keras package

TensorFlow is one of the most prominent machine learning packages. Knowing which version is on the system is vital as different builds have different options. In our case, we use tensorflow version 2.x and your can execute the following code to activate a recent version greater than 2.0.X

```
try:
  #%tensorflow_version 2.0 provide a method for printing the TensorFlow version...
  #method: tf.version.VERSION
  %tensorflow_version 2.x
except Exception:
  pass
```

After tensorflow activation, we have to import the package tensorflow as following

```
import tensorflow as tf
print(tf.version.VERSION)
```

The result of the print is the tensorflow version that you have in your system. As an example, colab returns the version 2.9.2. We can import additional packages like Keras numpy and matplotlib. here the entire code to execute:

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
print(tf.version.VERSION)
```

Step 3: download the Dataset

Pytorch Datasets

In particular, PyTorch has two primitives for data manipulation: ***torch.utils.data.DataLoader*** and ***torch.utils.data.Dataset***. ***Dataset*** stores the samples (records) and their corresponding labels (annotations), and ***DataLoader*** wraps an iterable around the Dataset.

More image datasets are available!

It is worth to note that ***torchvision.datasets*** module contains Dataset objects for many real-world vision data like MNIST, CIFAR, COCO, etc. A full list of open images dataset could be found here

Here below, an example to download MNIST open images training dataset.

```
# Download training data from open datasets.
training_data = datasets.MNIST(
    root="data", #root is the path where the data is stored,
    train=True, #specifies the training dataset,
    download=True, #downloads the data from the internet if it is not available at root
    transform=ToTensor(), #A function that takes in an PIL image and returns a transformed version
)
```

Notes!

- Fashion-MNIST is a dataset of Zalando's article images consisting of 60,000 training examples and 10,000 test examples. Each example comprises a 28×28 grayscale image and an associated label from one of 10 classes;
- PIL Image : PIL is the Python Imaging Library which provides the python interpreter with image editing capabilities;
- Converts a PIL Image $(H \times W \times C)$ to a Tensor of shape $(C \times H \times W)$.

After executing the code, you obtained the following answers from Colab:

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to data/M

100%
9912422/9912422 [00:00<00:00, 12340839.34it/s]

Extracting data/MNIST/raw/train-images-idx3-ubyte.gz to data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to data/M

100%
28881/28881 [00:00<00:00, 1328344.21it/s]

Extracting data/MNIST/raw/train-labels-idx1-ubyte.gz to data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to data/MN
```

```
100%
1648877/1648877 [00:00<00:00, 13681384.45it/s]

Extracting data/MNIST/raw/t10k-images-idx3-ubyte.gz to data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to data/MN

100%
4542/4542 [00:00<00:00, 199006.86it/s]

Extracting data/MNIST/raw/t10k-labels-idx1-ubyte.gz to data/MNIST/raw
```

TensorFlow Datasets

TensorFlow Datasets is a collection of ready-to-use datasets, with TensorFlow or other Python ML framework. All datasets are exposed as **tf.data.Datasets**, enabling easy-to-use, high-performance input pipelines. To benefit of the TensorFlow collection, we should import the $tensorflow\_datasets$ then we call the **.load()** method or $.load\_data()$:

```
#import the tensorflow datasets
import tensorflow_datasets as tfds
# Constract the MNIST dataset.
mnist_data=tfds.load('mnist', split='train', shuffle_files=True)
```

or

```
import tensorflow_datasets as tfds
MNIST_dataset = keras.datasets.mnist
mnist_data = MNIST_dataset.load_data()
```

We can remark the difference between the two methods. However, with the load method we should constract by ourself the separeted data for training and for test.

load() Vs. load_data() in Keras

- tfds.load('mnist', split='train', $shuffle\_files$=True): we should specify the name as a string of the dataset, we should precise if we download the train set or the set set. the parameter $shuffle\_files$ at True allows to shuffle the dataset before use;
- $load\_data()$ is a method operates on the dataset downloded before and allows to download the train and test data.

We can check the dataset size for train and for test by using the method ***len***. Hence we have 60,000 samples for train and 10,000 for test.

Exercise 2: Data preparing

For the remainder of the present Practical Work we will use Keras and tensorflow. In this exercise we will see how we download the train and test datasets with their corresponding labels vectors. To do this, your will complete the last code with the following: First you execute again the downloading code for all the MNIST dataset:

```
import tensorflow_datasets as tfds
MNIST_dataset = keras.datasets.mnist
mnist_data = MNIST_dataset.load_data()
```

then we load separately the train and the test datasets with their respective labels using the following code source

```
(train_images, train_labels),(test_images, test_labels) = mnist_data
```

1. You can check the data length of the train and test data by using the method len

```
len_train=len(mnist_data_train)
len_test=len(mnist_data_test)
print("number of train images",len_train)
print("number of test images",len_test)
```

2. You can check the labels values by transforming fisrt the label array to a simple list

```
labels = train_labels.tolist()
print("labels values",labels)
```

3. Check the length of the train and the test label's arrays
4. train and test datasets are splited to train your model and to test its performance and its capacity to generalize.
5. print the pixels image values from any sample:

```
print("image pixel values \n",train_images[1], " \n label of the image: ",
    train_labels[1])
```

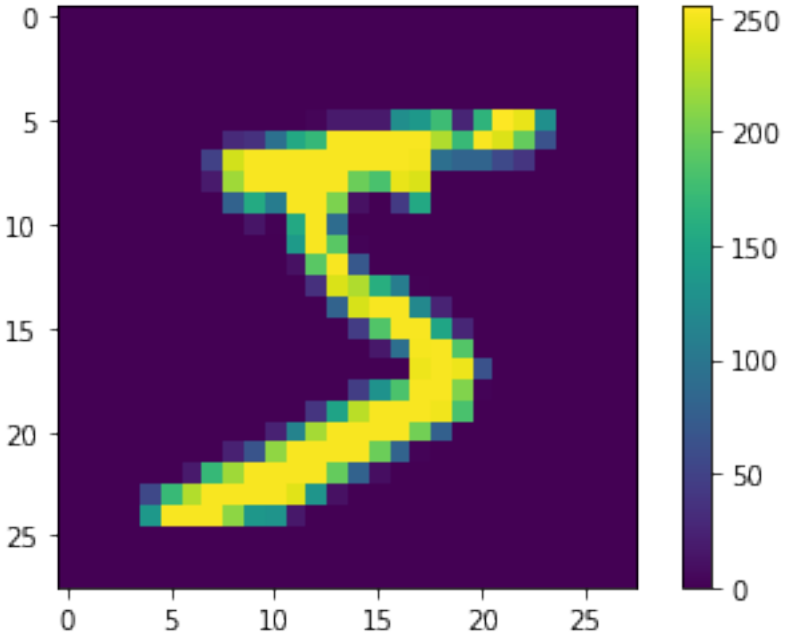
6. print an image resolution by using the method shape on an image:

```
print(" image resolution:", train_images[0].shape)
```

2 Exercise 3: Image plotting

Before starting the modelling step we can visualize the dataset of images thanks to the mathplotlib package. Here a simple example to display, or to plot the image:

```
import matplotlib.pyplot as plt
plt.figure()
plt.imshow ( train_images [0])
plt.colorbar ()
plt.grid ( False )
plt.show ()
```



We can easily check the content of the image at the index 0

1. You can write a simple function named displayImages allowing you to plot the image pixels and to display its label value as an integer value.
2. You can write a simple function named displayGridImages allowing to plot a grid of $n \times n$ image and their associated labels at the X-axis
3. What is the range values of the pixels of any image from train or test dataset?
4. Normalize the pixels values to be ranged between 0 and 1

Just an indication...!

```
def displayImages(nbImages,imageDataset, labels):
   i= 0
   for image in imageDataset:
          plt.figure()
          plt.imshow (imageDataset[i])
          plt.colorbar()
          plt.grid ( False )
          plt.xticks([])
          plt.yticks([])
          plt.xlabel(labels[i])
          plt.show()
          i = i + 1
          if i == nbImages:
             break
displayImages(2,train_images, train_labels)
```

Exercise 3: A simple Neural network model to image classification

In this exercise, we will propose to test and to understand two neural network architectures. Before starting with the architecture we should prepare our data to be correctly treated by the neural network.

Data preparation or data preprocessing

As it is detected in the last exercise, the shape of our input data is a 28x28x1 matrix, where each pixel is defined on one byte (a simple 2D image). Therefore, we fix the parameter $input\_shape = (28, 28, 1)$. The number of labels are of 10 varying from 0 to 9, hence we fix $num\_classes = 10$. Then, we normalize the input data to scale them in the $[0, 1]$ interval. To do this, we devide the pixel values by 255 after transforming the integer values to a float as follows:

```
#number of labels=number of classes
num_classes = 10
#define the shape of input data
input_shape = (28, 28, 1)
# Make sure images have shape (28, 28, 1)
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
print("x_train shape:", x_train.shape)
print(x_train.shape[0], "train samples")
print(x_test.shape[0], "test samples")
# Normalization:  Scale images to the [0, 1] range
x_train = train_images.astype("float32") / 255
x_test = test_images.astype("float32") / 255
```

Now let we see the output array which it should be a binary array having a dimension equal to 10. If the class of the input image is 5, we must have an output array equal to 1 at 5^{th} index and 0 everywhere else $[0., 0., 0., 0., 0., 1., 0., 0., 0., 0.]$. This conversion could be reached by executing the following code:

```
# convert class arrays to binary class matrices
y_train = keras.utils.to_categorical(train_labels, num_classes)
y_test = keras.utils.to_categorical(test_labels, num_classes)
```

Question: You can execute the hole code to prepare you data. What does mean the following message $x_trainshape$: $(60000, 28, 28, 1)$?

Model Architecture: model1

We propose a sequential layers composed by a flatten layer and two dense layers. The code of the model is:

```
model1= keras.Sequential([
    keras.layers.Flatten(input_shape=input_shape),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10, activation='softmax')
])
model1.summary()
```

- The Flatten (applatir) layer is used to reduce the input dimension and in general to transform the 3D Tensor on a reduced 1D tensor

- a dense layer with a ***relu*** activation function is the basic layer in Deep Learning. It simply takes an input, and applies a basic transformation with its activation function. The dense layer is essentially used to modify the dimensions of the tensor. It contains 128 fully connected neurons and we will see later how the activation function is important to data distillation to reach the best problem solution.
- The last dense layer contains 10 neurons and uses the softmax activation function allowing to associate 10 probability scores to each array class (tensor). Each node contains a score that indicates the probability that the current image belongs to one of the 10 classes. We remind the sum of scores at each output tensor is equal to 1.

Model Architecture: model2

For defining a network compatible with our data, we should define an input layer with the same size as the input data and an output corresponding the output data. The second architecture is a gold standard architecture for mnist dataset. Let excute the model2 defined as following:

```
from keras import layers
model2 = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)

model2.summary()
```

Question: Compare the detail of the parameters needed by each architecture on printing the summary() of the model.

The model2 uses a 2D convolutional layers and maxPooling layers. The important thing about convolution layers is the pattern they learn. The model2 could be considered as a ***convnet*** architecture that can also learn the spatial hierarchy of these patterns. A first convolution layer will learn small patterns, a second convolution layer will learn larger patterns consisting of features from the first layers, and so on. This allows convnets architectures to efficiently learn increasingly complex and abstract concepts. When using a convolution layer we will be interested in some aspects of these patterns:

- the number of neurons, with each neuron in a layer applying a different convolution to the entire image
- the size of the pattern, the number of pixels it will contain
- the distance between the patterns (it can be zero)

The difference between a dense layer and a convolution layer is the following: the dense layer is a global feature's learning while the convolution layer has a filtering action on pixels and allows to learn (to characterize) the local

features. We remark that in our model2, the mask size of the filter is 3×3. The first conv-layer computes 32 filters while the second conv-layer computes 64 filters, the double of the first conv-layer. We will see that the depth of the output feature map increases to capture more significant local features. Even the dimension increases with the increase of features, we apply a Flatten-layer to reduce the dimension. The Dropout-layer is a technique used to counter the problem of Overfitting, which is a frequent problem when training a Deep Learning model. It exists a large number algorithms to resolve this problem by using the $L2$ Regularization methods. We will see this fundamental notion later. With the Dropout-layer, we temporarily disable some random neurons in the network, as well as all its incoming and outgoing connections.

Exercise 4: Model's training step

In this exercise, we are looking for training the two models described in the last section. Before training the model, we need a few more settings added during the model's compile step:

Compiling step

- a loss function: is a measure needed by the model to evaluate its performance at the training step. Loss function is a kind of target trajectory followed by the training model to converge.
- an optimizer mechanism allowing the model to update its parameters according to the data learned and to the loss function return.
- a performance metrics: used by the model to monitor its response in the training and test steps.

Here the code that you can use as it is in the compile step:

```
model1.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
model2.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])
```

Training Step

Now we can start the training step by calling the ***fit*** function:

```
batch_size = 128
epochs = 15
model1.fit(x_train,y_train,batch_size=batch_size,epochs=epochs)
```

the function fit will be trained on the input data x_train with their tensor classes y_train using a number of iterations (15 iterations) or epochs for all the training data. The input data is injected to the model by a slot or mini-batches of 128 samples. The overfitting of the training model could be influenced by these two parameters. The higher the epoch number, the higher the computation time of the training model. You can check the time needed by each epoch during the model fitting. You can observe the difference between the fit function on the model1 and the following fit function on the model2:

```
model2.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
```

The difference between them is the use of a validation parameter. Use this fit function on the model1 and compare the results with those obtained with the first fit function. What do you observe?

model.fit() return value

the model.fit() returns a history object. This object is the history element considered as a history dictionary contains the information about the training period. We can access to this content by :

```
    history_dict = history.history
    history_dict.keys()

```

the history is composed of 4 parameters, one for each monitored metric. these information could be used to plot the evolution of the loss function over the epoch number and to plot the accuracy over the epoch number. These plots coukd be used to observe the fitting quality of the model (problem of overfitting), underfitting).

Exercise 5: Model Evaluation

It is mandatory to check the performance of the model to classify correctly or not a new image. To do this, we call the function model.evaluate() using the test dataset:

```
score = model2.evaluate(x_test, y_test, verbose=2)
print("Test loss:", score[0])
print("Test accuracy:", score[1])

```

You can observe the performance of the model2 against to the model1 after using the same fit function.

Exercise 6: Model Prediction

Based on the trained model, we can predict the class for some images and we can observe the model answer by executing the model.predict(). This function takes as a parameter a single or a set of images as an input and returns the predicted class. We can compare the predicted class with the real and observed class of the input image:

```
    predictions_1 = model1.predict(x_test)
    predictions_2 = model2.predict(x_test)
    print('observed class= ',y_test[0], 'predicted class=',np.argmax(predictions_1[0]))

```

Exercise 7: Application to another dataset

As it is said in the exercise 1, many datasets are available on TensorFlow. You will select another dataset using a classification probel and you should create your own colab with the different steps explained in this PW. Your colab should be uploaded online on the DVO repository before the deadline indicated.