Lab Session 6

«More practice on Prediction, Classification and Neural network»

---

1. Introduction

In this lab session you will put into test the methods and techniques that you've acquired during the previous lab sessions.

2. Learning outcome

On successful completion of this lab session you will be able to :

- Choose you prediction model
- Optimize the accuracy
- build a neural network from scratch

+ Code + Texte

3. Ressources

Libraries Documentation

- Python : https://docs.python.org/3/
- NumPy : https://numpy.org/doc/
- SciPy : https://docs.scipy.org/doc/scipy/
- Matplotlib : https://matplotlib.org/3.5.1/
- Panda : https://pandas.pydata.org/docs/
- mglearn : https://libraries.io/pypi/mglearn
- sklearn : https://scikit-learn.org/stable/
- PyTorch : https://pytorch.org/docs/stable/index.html

**to proceed with the lab session, refer to this section*

4. Setup

to install pyTorch in UTBM Desktops :

1. Create and activate your conda environement (refers to moodle)
2. Install pytorch packages in conda environment using the following instruction

```
# !pip3 install torch torchvision torchaudio
```

5. Prediction, Classification and Accuracy optimization

Exercice 1

In this exercise, you will work on CDC's indicators of Heart Disease. Use a model of your choicee to Predict with minimum 80% accuracy.

Test your model on the following persons :

1. [20, Yes, No No, 10, 10, Yes, Male, 50-54, White, No, No, Fair, 4.0, Yes, No, Yes]
2. [35, No, No, No, 10, 30, Yes, Male, 55-59, Black, Yes, Yes, Fair, 10.0, No, Yes, No]

What are the least decisive parameters, which parameters can we eliminate, how does it affect our prediction model ?

URL : https://www.kaggle.com/datasets/kamilpytlak/personal-key-indicators-of-heart-disease

Exercice 2

the following dataset includes features of multiple grilled mushrooms configurations, to determine whether they are edible or not. Build a classification model to obtain the highest possible accuracy ? what is most impacting feature ?

URL : https://www.kaggle.com/datasets/uciml/mushroom-classification

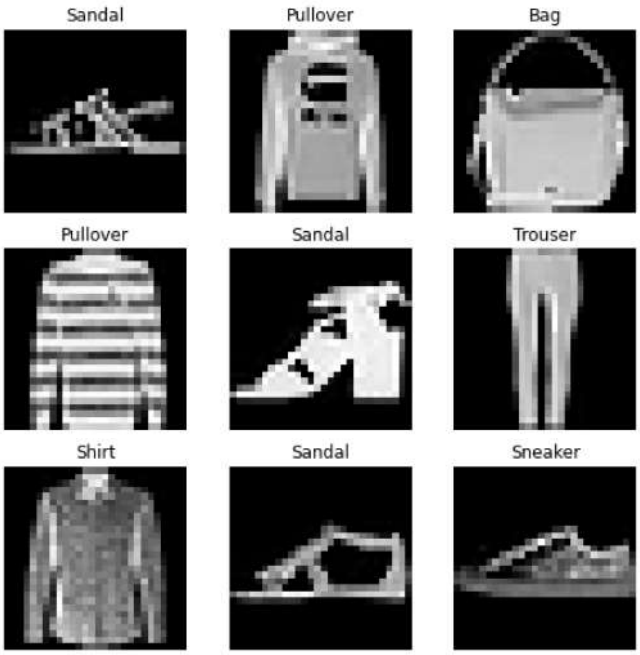
6. Neural Network on the Fashion MNIST dataset

Exercice 3 Create a neural network for classification on the Fashion MNIST dataset.
To do so, you can load the dataset by doing the following :
(Choose the right values for the different variables)

```
import torch
import torchvision # torch package for vision related things
import torch.nn.functional as F  # Parameterless functions, like (some) activation functions
import torchvision.datasets as datasets  # Standard datasets
import torchvision.transforms as transforms  # Transformations we can perform on our dataset for augmentation
from torchvision.transforms import ToTensor, Lambda # Transform PIL images to torch.FloatTensor
from torch import nn # For the neural network
from torch import optim  # For optimizers like SGD, Adam, etc.
from torch import nn  # All neural network modules
from torch.utils.data import DataLoader  # Gives easier dataset managment by creating mini batches etc.
from tqdm import tqdm  # For nicer progress bar
import matplotlib.pyplot as plt # visualize the data

## SOLUTION
input_size = ???     ## Size of the initial input
num_classes = ???    ## How many class do we have ?
learning_rate = ???  ## Size of the step between each iteration
batch_size = ???     ## Number of samples processed before the model is updated

training_data = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor()
)

test_data = datasets.FashionMNIST(
    root="data",
    train=False,
    download=True,
    transform=ToTensor()
)

train_dataloader = DataLoader(training_data, batch_size=64, shuffle=True)
test_dataloader = DataLoader(test_data, batch_size=64, shuffle=True)
```

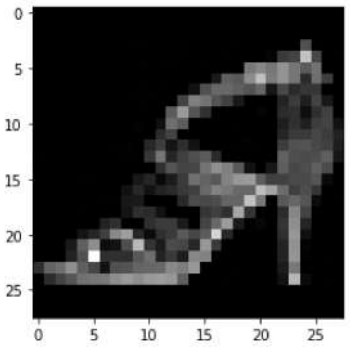
Fashion-MNIST is a dataset of Zalando's article images—consisting of a training set of 60,000 examples and a test set of 10,000 examples.
Each example is a 28x28 grayscale image, associated with a label from 10 classes.
You can see the data by doing the following code :

```
#Visualize the data
labels_map = {
    0: "T-Shirt",
    1: "Trouser",
    2: "Pullover",
    3: "Dress",
    4: "Coat",
    5: "Sandal",
    6: "Shirt",
    7: "Sneaker",
    8: "Bag",
    9: "Ankle Boot",
}

figure = plt.figure(figsize=(8, 8))
cols, rows = 3, 3
for i in range(1, cols * rows + 1):
    sample_idx = torch.randint(len(training_data), size=(1,)).item()
    img, label = training_data[sample_idx]
    figure.add_subplot(rows, cols, i)
    plt.title(labels_map[label])
    plt.axis("off")
    plt.imshow(img.squeeze(), cmap="gray")
plt.show()
```




```
# Display image and label.
train_features, train_labels = next(iter(train_dataloader))
print(f"Feature batch shape: {train_features.size()}")
print(f"Labels batch shape: {train_labels.size()}")
img = train_features[0].squeeze()
label = train_labels[0]
plt.imshow(img, cmap="gray")
plt.show()
print(f"Label: {label}")
```

```
Feature batch shape: torch.Size([64, 1, 28, 28])
Labels batch shape: torch.Size([64])
```




```
Label: 5
```

```
#Transfrom all the data
ds = datasets.FashionMNIST(
    root="data",
    train=True,
    download=True,
    transform=ToTensor(),
    target_transform=Lambda(lambda y: torch.zeros(10, dtype=torch.float).scatter_(0, torch.tensor(y), value=1))
)

target_transform = Lambda(lambda y: torch.zeros(
    10, dtype=torch.float).scatter_(dim=0, index=torch.tensor(y), value=1))

device = 'cuda' if torch.cuda.is_available() else 'cpu'
print('Using {} device'.format(device))
```

```
Using cpu device
```

Define the neural network as you think will make the more sense for the dataset.

The goal is an accuracy above 80%.

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        #Define the neural network
```

```
    def forward(self, x):
        #define the forward method
        return res
```

```
# Initialize network
model = NeuralNetwork().to(device)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# Print the model
print(model)
```

```
    NeuralNetwork(
      (flatten): Flatten(start_dim=1, end_dim=-1)
      (linear_relu_stack): Sequential(
        (0): Linear(in_features=784, out_features=512, bias=True)
        (1): ReLU()
        (2): Linear(in_features=512, out_features=512, bias=True)
        (3): ReLU()
        (4): Linear(in_features=512, out_features=10, bias=True)
        (5): ReLU()
      )
    )
```