- Implement a multi-layer perceptron (MLP)
- Tune related hyper-parameters
- Evaluate on classification tasks

This practice session requires the packages Tensorflow Keras. You may require to install Tensorflow and Keras along with their dependencies in a dedicated virtual environment.

1 Installing Tensorflow and Keras

You may refer to TF install Keras install for the install guidelines. For a proper install of both packages, let create a dedicated virtual environment.

1. Create a virtual environment using a terminal

```
# create a folder for your project
mkdir .myproject
cd myproject
# create a virtual environment with Python
python3 -m venv tf-keras-venv
source tf-keras-venv/bin/activate
# Type deactivate at the prompt and press Enter to exit the environment
# Upgrade pip
pip install --upgrade pip
# Installing TensorFlow
pip install --upgrade tensorflow
```

2. To test your environment, you can launch python and test the following code

```
python
# test tensorflow
import tensorflow as tf
print(tf.__version__)
# test keras
import keras
keras.__version__
```

3. You need to add the kernel if you want to have the virtual environment in the Jupyter Notebook. Exit from Python and then run the code in your terminal

```
pip install --user ipykernel
python -m ipykernel install --user --name=tf-keras-venv
#Now your are ready to launch jupyter-notebook
jupyter-notebook
```

When your jupyter-notebook is active, change the kernel to your virtual environment `tf-keras-venv`

2 Getting started on synthetic 2D data

We will use the datasets `mixtureexampleTrain.csv`, `mixtureexampleTest.csv` and the script script `utils_intro_deep.py`, available on Moodle.

1. Load the dataset `mixtureexampleTrain.csv` and check its dimensions.

```
import numpy as np
import pandas as pd
datatrain = pd.read_csv("mixtureexampleTrain.csv", delimiter = "\t", header=
    None)
Ytrain = datatrain[2].values
Ytrain[Ytrain < 0] = 0
Xtrain = datatrain.drop(2, axis=1).values
```

Which type of problem is to be solved? Proceed similarly to load the test set `mixtureexampleTest.csv`.

2. Create a simple MLP with one hidden layer of dimension 64 and the `Relu` activation function followed by an output layer with a `sigmoid` activation function.

```
from keras.models import Sequential
from keras.layers import Dense, Activation, InputLayer
nhu = 64
input_shape=(2, )
model = Sequential([
  InputLayer(shape=input_shape),
  Dense(units=nhu, activation="relu"),
  Dense(units=1, activation="sigmoid")
])
model.summary()
```

How many trainable parameters does the model have? Justify the answer.

3. Now let set the optimizer, and then compile the model.

```
from keras.optimizers import SGD
#set stochastic gradient (with momentum and weigth decay) as optimizer
sgd = SGD(learning_rate=0.1, weight_decay=1e-6, momentum=0.9)
# compile the model with the optimizer, the loss function and the evaluation
    metric
model.compile(optimizer="sgd", loss="binary_crossentropy", metrics=["accuracy"
    ])
```

4. Train the model and visualize the decision frontier on the training data

```
hist = model.fit(Xtrain, Ytrain, epochs=10, batch_size = 32)
from utils-intro-deep import plot_decision_2d
plot_decision_2d(Xtrain, Ytrain, model, resolution=0.02, titre="RNN")
```

What do the parameters `batch_size` and `epochs` represent?

5. Let plot the evolution of the training loss across epochs

```
  plt.plot(range(1, len(hist.history["loss"])+1), hist.history["loss"], color="
      blue")
  plt.title("Train Loss")
```

Complete the program to also plot the evolution of the metric (accuracy) over the epochs.

6. Finally, we evaluate the generalization performance of the model

```
score_test = model.evaluate(Xt, Yt)
print("\nLoss (test): %.3f" % score_test[0])
print("Accuracy (test): %.3f" % score_test[1])
```

7. Analyze the results obtained by testing different numbers of epochs, batch_size, and optimizer types (see the Keras doc).

3 MLP for MNIST

The goal is to train an MLP to classify the images Mnist.

1. Load the dataset.

```
  from keras.datasets import mnist
  (x_train, y_train), (x_test, y_test) = mnist.load_data()
```

How many samples do we have in both datasets? How many outputs are to predict?
Visualize some of the digits (recall that each input is a 28×28 gray scale image).

2. For the purpose of the MLP training, we scale all inputs to the [0, 1] range and flatten each input to a vector o f size 784. Also each output is transformed using one-hot encoding.

```
# flatten 28*28 images to a 784 vector for each image and conversion to
    float32 for normalization sake
nb_pixels = x_train.shape[1] * x_train.shape[2]
x_train = x_train.reshape(x_train.shape[0], nb_pixels).astype('float32')
x_test = x_test.reshape(x_test.shape[0], nb_pixels).astype('float32')
# one-hot class encoding
from keras.utils import to_categorical
nbclasses = len(np.unique(y_train))
y_train = to_categorical(y_train, nbclasses)
y_test = to_categorical(y_test, nbclasses)
```

3. Create and MLP `model_mnist` with one hidden layer of 800 neurons and a `"relu"` activation function followed by an output layer of size 10 with a `"softmax"` acctivation function . How many parameters do we have?

4. Compile the model with `"categorical_crossentropy"` loss, the `"accuracy"` metric.

```
model_mlp.compile(loss="categorical_crossentropy", optimizer=..., metrics=["
    accuracy"])
```

5. Train the model over a few epochs and evaluate its performance on the test set. Comments the results relatively to the best performances obtained on MNIST.

6. If your computer allows it, implement a neural network with two hidden layers and evaluate it. Compare to the previous question.

4 To go further

4.1 Model saving

1. The trained model can be saved and used afterwards.

```
## Save the model
model_mlp.save("model_mlp.keras")

## Relaod back the model
from keras.models import load_model
reconstructed_model_mlp = load_model("model_mlp.keras")
```

2. Propose a way to check that the original model and the reloaded model are similar and yield to the same performance

```
# Let's check:
check = np.testing.assert_allclose(model_mlp.predict(x_test),
    reconstructed_model_mlp.predict(x_test))
print(check)
```

4.2 Monitoring validation error, callback

1. It can be useful to monitor the loss and the metric on the training data, but also on the validation set. For this, split the training set to generate validation data to be used during the model training. Let `x_val, y_val` the validation set.

```
nbepochs = 5
hist = model_mlp.fit(x_train, y_train, validation_data=(x_val, y_val), epochs=
    nbepochs, batch_size=32)
# Instead of that we can specify the proportion of the original training set
    to be used as validation data
# hist = model.fit(x_train, y_train, validation_split=0.25, epochs=nbepochs,
    batch_size=32 )
```

Plot the training loss and the metric for respectively the training and validation sets. Could we have stopped earlier the training?

```
plt.figure(), plt.subplot(1,2,1)
plt.plot(range(1, nbepochs+1), hist.history["loss"], color="blue",
label="Train loss")
plt.plot(range(1, nbepochs+1), hist.history["val_loss"], color="red", label="
    Val loss")
plt.xlabel("Epochs"), plt.title("Loss function"), plt.legend(loc="best")
plt.subplot(1,2,2)
plt.plot(range(1, nbepochs+1), hist.history["accuracy"], color="blue",
label="Train acc")
plt.plot(range(1, nbepochs+1), hist.history["val_accuracy"], color="red",
label="Val acc")
plt.xlabel("Epochs"), plt.title("Accuracy"), plt.legend(loc="best")
```

2. Keras allows to monitor how the neural network performs across epochs via *callbacks*. For instance, one may perform an earlystopping based on the evolution of the validation error. An example implementing this mechanism is shown below. Adapt it to your problem.

```
from keras.callbacks import EarlyStopping
# monitor the validation accuray
early_stop = EarlyStopping(monitor="val_loss", min_delta=0, patience=2,
    verbose=0)
# min_delta : minimum change in the monitored quantity to qualify as an
    improvement
# patience : number of epochs with no improvement after which training will be
     stopped
# training of the model with early stopping
nbepochs = 10
callbacks = EarlyStopping(monitor="val_loss", patience=2)
hist = model_mlp.fit(x_train, y_train, validation_split=0.25, epochs=nbepochs,
     batch_size=32, shuffle=True, callbacks=[early_stop])
```

3. By consulting the Keras documentation, extend the callback principle to save the model after each epoch if necessary (see callback `ModelCheckpoint`)