# TP4MNIST

March 16, 2023

TP4:Tensorflow MNIST

Ce Notebook permet d'apprendre à développer un modèle de classification sur le dataset MNIST, en utilisant l'API Keras.

[24]:
```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow import keras
```

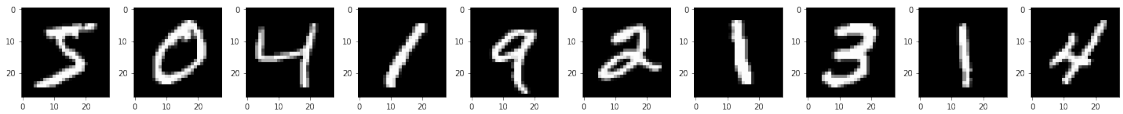
## 0.1 1. Chargement des données et Normalisation

[25]:
```
# Chargement des données MNIST
(X_train, y_train) , (X_test, y_test) = keras.datasets.mnist.load_data()
print('trainset:', X_train.shape) # 60,000 images
print('testset:', X_test.shape) # 10,000 images
# Normalisation des données
X_train = X_train / 255
X_test = X_test / 255
```

```
trainset: (60000, 28, 28)
testset: (10000, 28, 28)
```

## 0.2 2. Visualisation des données

[26]:
```
# visualisation de quelques images
fig, ax = plt.subplots(nrows=1, ncols=10, figsize=(20, 4))
for i in range(10):
    ax[i].imshow(X_train[i], cmap='gray')
plt.tight_layout()
plt.show()
```



### 0.3 3. Configuration des Couches du Réseau de Neurones

[27]:
```
# Configuration des couches du réseau
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(10)
])
```

### 0.4 4. Entrainement du Réseau de Neurones

[28]:
```
# Compilation du modele
#model.compile(optimizer='adamax', loss='MSE')
model.compile(optimizer='adam',
              loss= keras.losses.SparseCategoricalCrossentropy(from_logits=True),
               metrics=['accuracy'])

#Entrainement du modele
model.fit(X_train, y_train, epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 10s 3ms/step - loss: 0.2295 -
accuracy: 0.9327
Epoch 2/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0955 -
accuracy: 0.9704
Epoch 3/10
1875/1875 [==============================] - 5s 3ms/step - loss: 0.0697 -
accuracy: 0.9780
Epoch 4/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0524 -
accuracy: 0.9830
Epoch 5/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0421 -
accuracy: 0.9863
Epoch 6/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0349 -
accuracy: 0.9897
Epoch 7/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0277 -
accuracy: 0.9908
Epoch 8/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0236 -
accuracy: 0.9920
Epoch 9/10
1875/1875 [==============================] - 7s 3ms/step - loss: 0.0220 -
accuracy: 0.9925
```

```
Epoch 10/10
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0193 -
accuracy: 0.9935
```

[28]:
```
<keras.callbacks.History at 0x216b985f160>
```

### 0.5 5. Évaluation du réseau de neurone sur les données de Test

[29]:
```
# Evaluation du modele
test_loss, test_acc = model.evaluate(X_test,  y_test)
print('Test accuracy:', test_loss)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0884 -
accuracy: 0.9807
Test accuracy: 0.08844907581806183
```

### 0.6 6. Création d'un modele prédictif

[31]:
```
# modele prédictif (softmax)
prediction_model = keras.Sequential([model, keras.layers.Softmax()])
predictions = prediction_model.predict(X_test)
print('np.argmax(a, axis=1): {0}'.format(np.argmax(predictions, axis=1)))
print(y_test)
```

```
np.argmax(a, axis=1): [7 2 1 … 4 5 6]
[7 2 1 … 4 5 6]
```

[ ]:
```

```