Fiche TP4: Boudelal Abderahmane et Nahoui Seif :

Exercice1 :

Lien vers le code d'exo1 :
https://colab.research.google.com/drive/1b3wYxZXQHECp6Y_ywAHTdVyB7YcEogGc?usp=sharing

Lien vers le code d'exo2 :

https://colab.research.google.com/drive/1ZdE-G3_u0c5GTvBzS2KrV8uF_NW_EJpc?usp=sharing

Ex01 :

1 /

Import necessary libraries : Les bibliothèques nécessaires importées, numpy pour les opérations sur les tableaux, scikit-learn pour le traitement des données et l'évaluation, et tensorflow.keras pour la construction et l'entraînement du modèle ANN.

```
# Import necessary libraries
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import accuracy_score
```

2/Data est chargé via load_iris(). Il contient des caractéristiques (longueur/largeur des pétales et sépales) et des cibles (espèces).

```
[2] # Load the Iris dataset
    iris = load_iris()
    X = iris.data  # Features: Petal length, petal width, sepal length, sepal width
    y = iris.target.reshape(-1, 1)  # Targets: Species
```

3/Les cibles (y) sont encodées en one-hot grâce à OneHotEncoder. Cela est indispensable pour la compatibilité avec la sortie softmax du réseau.

```
[3] # One-hot encode the target labels
    encoder = OneHotEncoder()
    y = encoder.fit_transform(y).toarray()
```

4/Split data into train and and test (0.8 and 0.2 test)

```
[4]  # Split the dataset into training and testing sets
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

5/standardisées Les caractéristiques (StandardScaler) pour assurer une meilleure convergence pendant l'entraînement du réseau.

```
# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

6/modèle séquentiel avec 4 couches denses : Trois couches cachées de 10 neurones utilisant l'activation ReLU. Une couche de sortie avec 3 neurones utilisant softmax pour produire des probabilités.

```
[6]  # Build the ANN model
     model = Sequential([
         Dense(10, activation='relu', input_shape=(X_train.shape[1],)),
         Dense(10, activation='relu'),
         Dense(10, activation='relu'),
         Dense(3, activation='softmax')  # 3 output neurons for 3 classes
     ])
```

7/modèle est compilé avec l'optimiseur Adam, une fonction de perte categorical_crossentropy (adaptée aux tâches de classification multiclasse) et l'accuracy comme métrique. L'entraînement est effectué sur 1000 époques avec une taille de lot de 16. accuracy of 0.97.

modèle est compilé avec l'optimiseur Adam, une fonction de perte categorical_crossentropy (adaptée aux tâches de classification multiclasse) et l'accuracy comme métrique. L'entraînement est effectué sur 1000 époques avec une taille de lot de 16. accuracy of 0.97.

```
[7]  # Compile the model
     model.compile(optimizer=Adam(learning_rate=0.01), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
[8]  # Train the model
     history = model.fit(X_train, y_train, epochs=1000, batch_size=16, verbose=0)
```

```
[9]  # Evaluate the model on test data
     test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
     print(f"Test Accuracy: {test_accuracy:.2f}")
```

8/ **Perte (Loss)** :

- La courbe de perte (en bleu) diminue rapidement dans les premières époques, indiquant une bonne optimisation du modèle au début.
- Après environ 200 époques, la perte se stabilise à une valeur proche de zéro, suggérant que le modèle a bien appris les données d'entraînement.
- On observe quelques pics dans la courbe de perte après 200 époques, probablement dus à des variations dans les mini-lots utilisés pendant l'entraînement (batch size). Cela peut être corrigé en ajustant l'optimiseur ou en augmentant la taille du lot.

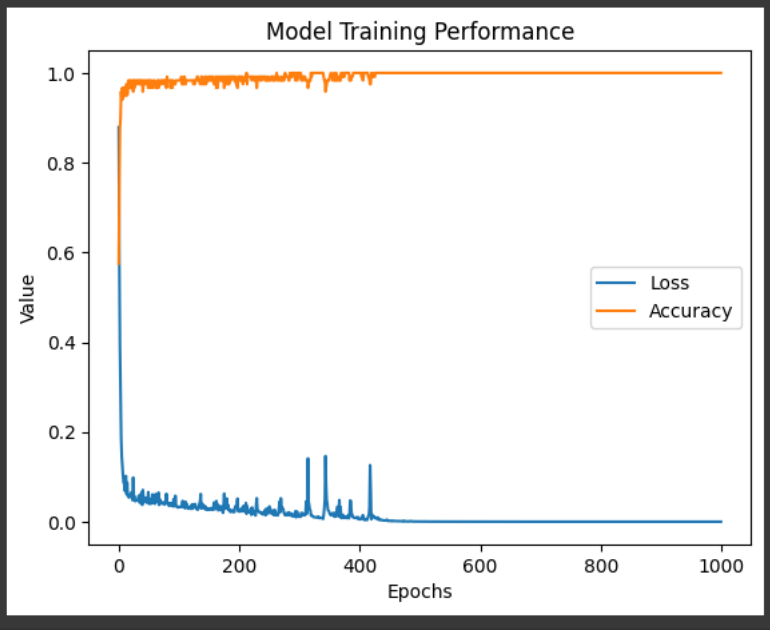
 Accuracy (Précision) :

- La précision (en orange) atteint presque 1.0 (100%) très rapidement (avant 100 époques) et reste stable sur les époques suivantes.
- Cette performance parfaite sur l'ensemble d'entraînement pourrait indiquer un **surapprentissage** (overfitting), surtout si la précision sur les données de test n'est pas similaire.

```
Accuracy on Test Set: 0.97
```

```python
# Plot training performance
import matplotlib.pyplot as plt

plt.plot(history.history['loss'], label='Loss')
plt.plot(history.history['accuracy'], label='Accuracy')
plt.title('Model Training Performance')
plt.xlabel('Epochs')
plt.ylabel('Value')
plt.legend()
plt.show()
```



7/

Exercice 02 :

Upload the Mnist data and print the shape and type of data

```python
fashion_mnist = keras.datasets.fashion_mnist
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist.load_data()

# Check dataset shape and data type
print("Training set shape:", X_train_full.shape)
print("Training set dtype:", X_train_full.dtype)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Training set shape: (60000, 28, 28)
Training set dtype: uint8
```

2/

Define neural netwirk with sequentienl en 4 couches Flatten(input_shape=[28, 28], name="InputLayer") : Cette couche aplatie une image d'entrée de dimension (28, 28) en un vecteur unidimensionnel de longueur 784 (28 x 28). and 2 hidden layers and output layer to produce the probabilities . loss='sparse_categorical_crossentropy' : Utilisée pour une tâche de classification multiclasse avec des étiquettes entières (non encodées one-hot). optimizer=SGD() : Stochastic Gradient Descent (SGD) est utilisé pour ajuster les poids. metrics=["accuracy"] : La précision (accuracy) est suivie pour évaluer les performances du modèle

```
[4] model = Sequential([
        Flatten(input_shape=[28, 28], name="InputLayer"),
        Dense(300, activation='relu', name="HiddenLayer1"),
        Dense(100, activation='relu', name="HiddenLayer2"),
        Dense(10, activation='softmax', name="OutputLayer")
    ])


    model.compile(
        loss='sparse_categorical_crossentropy',
        optimizer=SGD(),
        metrics=["accuracy"]
    )

    # Display model summary
    model.summary()
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_
  super().__init__(**kwargs)
Model: "sequential"
```

Layer (type)	Output Shape	Param #
InputLayer (Flatten)	(None, 784)	0
HiddenLayer1 (Dense)	(None, 300)	235,500
HiddenLayer2 (Dense)	(None, 100)	30,100
OutputLayer (Dense)	(None, 10)	1,010

3/

history, qui stocke des informations sur l'évolution des métriques pour chaque époque, à la fois pour l'ensemble d'entraînement et l'ensemble de validation. Le modèle sera entraîné pendant 20 itérations sur l'ensemble complet des données d'entraînement.Les données de validation sont utilisées à la fin de chaque époque pour évaluer les performances du modèle sur des données jamais vues.verbose=1 Contrôle le niveau de détail affiché pendant l'entraînement.

```
history = model.fit(
    X_train, y_train,
    epochs=20,
    validation_data=(X_valid, y_valid),
    verbose=1
)
```

4/ Graphique de gauche : "Training and Validation Loss" (Perte d'entraînement et de validation) Axe des X :

Représente les époques (le nombre d'itérations complètes sur les données d'entraînement). Axe des Y :

Représente la valeur de la perte (loss), une mesure de l'erreur entre les prédictions du modèle et les étiquettes réelles. Courbe bleue (Training Loss):

La perte sur l'ensemble d'entraînement diminue progressivement, ce qui montre que le modèle apprend bien les données d'entraînement. Cette diminution constante est attendue : au fur et à mesure de l'entraînement, le modèle s'améliore à prédire les sorties correctes sur les données d'entraînement. Courbe orange (Validation Loss):

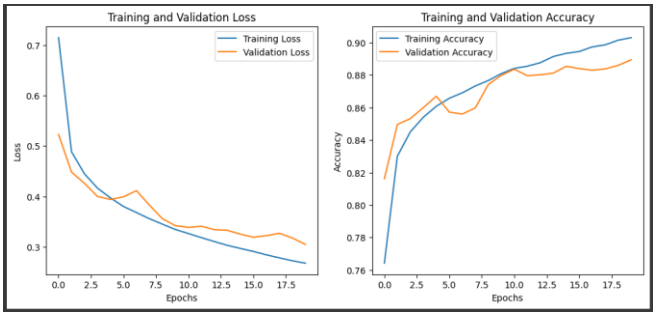
La perte de validation diminue initialement, indiquant que le modèle généralise bien sur des données qu'il n'a pas vues. La stabilité relative des deux courbes suggère qu'il n'y a pas d'overfitting majeur après 20 époques (l'écart entre la perte d'entraînement et de validation reste modéré). Cependant, la légère hausse et irrégularité au début des époques (entre 5 et 10) pourraient signaler des fluctuations dues à des variations aléatoires ou au batch sampling. Graphique de droite : "Training and Validation Accuracy" (Précision d'entraînement et de validation) Axe des X :

Comme pour le graphique de gauche, représente les époques. Axe des Y :

Représente la précision (accuracy), une métrique qui mesure le pourcentage de prédictions correctes. Courbe bleue (Training Accuracy):

La précision sur l'ensemble d'entraînement augmente régulièrement avec les époques. Cela montre que le modèle devient plus performant à prédire correctement les labels sur les données d'entraînement. Courbe orange (Validation Accuracy):

La précision de validation suit une tendance similaire à celle de l'entraînement, atteignant un plateau autour de 88%. Cela suggère que le modèle généralise relativement bien sur les données de validation. Une légère oscillation est visible, ce qui est normal pour un petit nombre d'époques et peut être amélioré avec davantage d'échantillons de validation ou des techniques de régularisation.



5/

e modèle prédit les probabilités pour chaque classe pour les trois premières images du jeu de test X_test. Le résultat, predictions, est une matrice où chaque ligne représente les probabilités associées aux classes pour une image.a fonction np.argmax sélectionne l'indice de la probabilité maximale (la classe prédite) pour chaque ligne de predictions.

```
[6]  predictions = model.predict(X_test[:3])
     predicted_classes = np.argmax(predictions, axis=1)
     print(f"Predicted Classes: {predicted_classes}")
     print(f"True Classes: {y_test[:3]}")

     # Visualize predictions
     for i in range(3):
         plt.imshow(X_test[i], cmap="binary")
         plt.title(f"Predicted: {predicted_classes[i]}, True: {y_test[i]}")
         plt.axis('off')
         plt.show()
```