جامعة عبد المالك السعدي
Université Abdelmalek Essaâdi

كلية العلوم و التقنيات بطنجة
Faculté des Sciences et Techniques de Tanger

# Atelier 1

Realise par :

**Ziad Ben Saada**

Encadre par :

**Lotfi El Aachak**

# Part one regression:

## Part 1: Exploratory Data Analysis (EDA)

```
import pandas as pd

# Load the dataset

fundamentals = pd.read_csv("fundamentals.csv")

prices = pd.read_csv("prices.csv")

prices_split_adjusted = pd.read_csv("prices-split-adjusted.csv")

securities = pd.read_csv("securities.csv")

# Perform exploratory data analysis

print("Fundamentals dataset:")

print(fundamentals.head())
```

## Part 2: Deep Neural Network Architecture for Regression

```
import torch

import torch.nn as nn

import torch.optim as optim



# Define your neural network architecture

class RegressionModel(nn.Module):

    def __init__(self, input_size, hidden_size, output_size):

        super(RegressionModel, self).__init__()
```

```
        self.fc1 = nn.Linear(input_size, hidden_size)

        self.relu = nn.ReLU()  # Indentation corrected here

        self.fc2 = nn.Linear(hidden_size, output_size)


    def forward(self, x):

        out = self.fc1(x)

        out = self.relu(out)

        out = self.fc2(out)

        return out


# Define your hyperparameters

input_size = 100 # Define input size

hidden_size = 64 # Define hidden layer size

output_size = 1  # Since it's regression


# Initialize the model

model = RegressionModel(input_size, hidden_size, output_size)


# Define loss function and optimizer

criterion = nn.MSELoss()

optimizer = optim.SGD(model.parameters(), lr=0.001)  # Example learning rate
self.relu = nn.ReLU()
```

## Part 3: Hyperparameter Tuning with GridSearch

```
from sklearn.model_selection import GridSearchCV

# Define the hyperparameters grid

param_grid = {

    'lr': [0.001, 0.01, 0.1],

    'optimizer': [optim.SGD, optim.Adam],

    # Add more hyperparameters to search...

}

# Initialize the GridSearchCV

grid_search = GridSearchCV(estimator=model, param_grid=param_grid,
scoring='neg_mean_squared_error', cv=5)

grid_search.fit(X_train, y_train)  # Assuming you have X_train and y_train

# Get the best parameters

best_params = grid_search.best_params_

print("Best hyperparameters:", best_params)
```

## Part 4: Visualization of Loss and Accuracy

```
import matplotlib.pyplot as plt

# Plot Loss / Epochs and Accuracy / Epochs for both training and test data

plt.plot(train_losses, label='Train Loss')

plt.plot(test_losses, label='Test Loss')

plt.xlabel('Epochs')
```

```
plt.ylabel('Loss')

plt.legend()

plt.show()

plt.plot(train_accuracies, label='Train Accuracy')

plt.plot(test_accuracies, label='Test Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()
```

## Part two multi class classification:

### Part 1: Data Preprocessing

```
import pandas as pd

from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Load the dataset

data = pd.read_csv("predictive_maintenance.csv")

# Data cleaning - handle missing values if any

data.dropna(inplace=True)

# Standardization/Normalization

scaler = StandardScaler()  # or MinMaxScaler for normalization

data_scaled = scaler.fit_transform(data)
```

## Part 2: Exploratory Data Analysis (EDA)

```
import seaborn as sns

import matplotlib.pyplot as plt

# Perform exploratory data analysis

# Example: visualizing distribution of features

sns.pairplot(data)

plt.show()
```

## Part 3: Data Augmentation Techniques

```
from sklearn.utils import resample

# Apply data augmentation techniques such as oversampling or undersampling

# Example: Oversampling the minority class

# Assuming 'label' is the column representing class labels

majority_class = data[data['label'] == 0]

minority_class = data[data['label'] == 1]

minority_upsampled = resample(minority_class, replace=True,
n_samples=len(majority_class))

# Combine majority class with upsampled minority class

data_balanced = pd.concat([majority_class, minority_upsampled])

# Now 'data_balanced' contains balanced classes
```

## Part 4: Deep Neural Network Architecture for Multi-class Classification

```
import torch

import torch.nn as nn

import torch.optim as optim


# Define your neural network architecture

class ClassificationModel(nn.Module):

    def __init__(self, input_size, hidden_size, output_size):

        super(ClassificationModel, self).__init__()

        self.fc1 = nn.Linear(input_size, hidden_size)

        self.relu = nn.ReLU()

        self.fc2 = nn.Linear(hidden_size, output_size)

    def forward(self, x):

        out = self.fc1(x)

        out = self.relu(out)

        out = self.fc2(out)

        return out

# Define your hyperparameters

input_size =  # Define input size

hidden_size =  # Define hidden layer size

output_size =  # Define output size (number of classes)

# Initialize the model
```

```
model = ClassificationModel(input_size, hidden_size, output_size)

# Define loss function and optimizer

criterion = nn.CrossEntropyLoss()

optimizer = optim.SGD(model.parameters(), lr=0.001)  # Example learning rate
```

## Part 5: Hyperparameter Tuning with GridSearch

```
from sklearn.model_selection import GridSearchCV

# Define the hyperparameters grid

param_grid = {

    'lr': [0.001, 0.01, 0.1],

    'optimizer': [optim.SGD, optim.Adam],

    # Add more hyperparameters to search...

}

# Initialize the GridSearchCV

grid_search = GridSearchCV(estimator=model, param_grid=param_grid,
scoring='accuracy', cv=5)

grid_search.fit(X_train, y_train)  # Assuming you have X_train and y_train

# Get the best parameters

best_params = grid_search.best_params_

print("Best hyperparameters:", best_params)
```

## Part 6: Visualization of Loss and Accuracy

```
import matplotlib.pyplot as plt

# Plot Loss / Epochs and Accuracy / Epochs for both training and test data

plt.plot(train_losses, label='Train Loss')

plt.plot(test_losses, label='Test Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.legend()

plt.show()

plt.plot(train_accuracies, label='Train Accuracy')

plt.plot(test_accuracies, label='Test Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend()

plt.show()
```

## Part 7: Metrics Calculation

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Calculate metrics on training dataset

train_predictions = model.predict(X_train)

train_accuracy = accuracy_score(y_train, train_predictions)

train_precision = precision_score(y_train, train_predictions)
```

```
train_recall = recall_score(y_train, train_predictions)

train_f1 = f1_score(y_train, train_predictions)
```

## Synthesis

Throughout this lab, I have learned various essential concepts and techniques in machine learning and deep learning:

1. Data preprocessing: Cleaning, standardization, and normalization are crucial steps to prepare data for modeling.
2. Exploratory Data Analysis (EDA): Visualizing data distributions, correlations, and patterns helps in understanding the dataset better.
3. Data Augmentation: Techniques like oversampling and undersampling can help address class imbalance issues.
4. Deep Neural Network Architecture: Designing neural network architectures for classification tasks involves defining layers, activation functions, and output layers.
5. Hyperparameter Tuning: Using tools like GridSearchCV helps in finding the best hyperparameters for the model.
6. Visualization: Plotting loss and accuracy curves helps in understanding model training and performance.
7. Metrics Calculation: Evaluating model performance using metrics like accuracy, precision, recall, and F1-score provides insights into its effectiveness.
8. Regularization Techniques: Applying techniques like dropout and weight decay helps prevent overfitting and improve model generalization.

Overall, this lab has provided a comprehensive understanding of various aspects of machine learning and deep learning, equipping me with valuable skills for tackling real-world problems in predictive maintenance and other domains.

Lien Github:

https://github.com/ziadbensaada/Atelier1_DNN-MLP/