Національний технічний університет України «КПІ ім. Ігоря Сікорського»
Факультет Інформатики та Обчислювальної Техніки

Кафедра інформаційних систем та технологій

Лабораторна робота №6

з дисципліни «Методи та технології штучного інтелекту»

на тему

«Нейро-нечітке моделювання»

Виконала:
студентка групи ІС-12
Павлова Софія

Викладач:
Шимкович В. М.

Київ – 2023

1. Постановка задачі

Мета: Отримання і закріплення знань про методи моделювання та принципи функціонування нейронечітких систем, а також формування практичних навичок з конструювання нейронечітких мереж.

Завдання:

1. Сформулювати завдання в галузі обчислювальної техніки, для вирішення якої було б обґрунтовано застосування гібридної нейронечіткої мережі.
2. Сформувати вибірку для навчання гібридної нейронної мережі.
3. Згенерувати і візуалізувати структуру гібридної нейронної мережі.
4. Навчити гібридну нейронну мережу, при цьому задати і обґрунтувати параметри її навчання.
5. Виконати перевірку адекватності побудованої нечіткої моделі гібридної мережі.

2. Виконання

Постановка задачі:

Сформулюємо задачу в галузі машинного навчання наступним чином:

Здача – Прогнозування курсу валюти.

Необхідно прогнозувати курс валюти (EUR до USD) за допомогою гібридної нейронної мережі.

Задача полягає у створенні та тренуванні моделі, яка може адекватно визначати залежність між показниками відкриття (*'BO'* або *'Open'*), високої (*'BH'* або *'High'*), низької (*'BL'* або *'Low'*) ціни та кінцевою (*'BC'* або *'Close'*) ціною валютного обміну.

Набір даних:

Для нашої задачі використаємо публічний датасет EUR USD Forex Pair Historical Data (2002 - 2020).

Date	Time	BO	BH	BL	BC
2005-05-02	00:00	1.2852	1.2852	1.284	1.2844
2005-05-02	01:00	1.2844	1.2848	1.2839	1.2842
2005-05-02	02:00	1.2843	1.2854	1.2841	1.2851
2005-05-02	03:00	1.2851	1.2859	1.285	1.2851
2005-05-02	04:00	1.2852	1.2859	1.2849	1.2855
2005-05-02	05:00	1.2854	1.2858	1.2853	1.2854
2005-05-02	06:00	1.2854	1.286	1.2852	1.28585

Рисунок 1 – Вигляд датасету

Датасет містить наступні стопці:

Date – **Дата**:

Містить дані з *2005-05-02* (2 травня 2005) по *2020-04-29* (29 квітня 2020).

Time – **Час**:

Година, в якій була виміряна ціна. Містить значення з *00:00* по *23:00*.

BO – **Open**:

Ціна початкової пропозиції.

BH – **High**:

Найвища ціна пропозиції за годину.

BL – **Low**:

Найнижча ціна пропозиції за годину.

BC – **Close**:

Ціна закриття пропозиції.

Створення гібридної нейронної мережі:

Завантаження даних

Оскільки набір даних містить *93 084* зразка – що є завеликою кількістю для навчання нейромережі, використаємо перші *3794* зразка, завантаживши дані до *2006-02-01* (1 лютого 2006).

Завантажимо наш датасет та розділимо його на тренувальний і тестовий у співвідношенні 80:20. Відобразимо датасет у вигляді графіка та таблиць DataFrame.

За ознаки візьмемо стовпці **BO (Open)**, **BH (High)**, **BL (Low)**, а за мітки – **BC (Close)**.

Лістинг:

```
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import warnings
```

```
# Вимкнення варнінгів PyTorch
warnings.filterwarnings("ignore", category=UserWarning)

# Функція завантаження даних
def load_data(file_name, end_date):
    # Завантаження даних з CSV-файлу
    dataset = pd.read_csv(file_name)
    dataset['Date'] = pd.to_datetime(dataset['Date'])
    dataset = dataset[dataset['Date'] <= end_date]

    # Розділення даних на ознаки та мітки
    features = dataset[['BO', 'BH', 'BL']]
    labels = dataset['BC']

    # Конвертація у тензори PyTorch
    X = torch.tensor(features.values).float()
    y = torch.tensor(labels.values).float()

    # Розділення даних на тренувальні та тестувальні набори
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
shuffle=False)

    # Візуалізація даних на графіках
    plot_data(dataset)

    return X_train, X_test, y_train, y_test

# Функція візуалізації датасету
def plot_data(dataset):
    plt.plot(dataset.index, dataset['BC'])
    plt.title('Курс EUR до USD')
    plt.xlabel('Екземпляр')
    plt.ylabel('Курс')
    plt.grid(True)
    plt.show()

# Головні виклики
if __name__ == "__main__":
    file_name = 'eur_usd_hour.csv'
    end_date = '2006-02-01'

    # Завантаження даних

    X_train, X_test, y_train, y_test = load_data(file_name, end_date)

    # Візуалізація вхідних даних

    # Конвертація train_data у DataFrame
    df_X_train = pd.DataFrame(X_train.numpy(), columns=['Open', 'High', 'Low'])
    df_y_train = pd.DataFrame(y_train.numpy(), columns=['Close'])
    # Об'єднання DataFrames
    df_train = pd.concat([df_X_train, df_y_train], axis=1)
    print('\nТренувальні дані:')
    print(df_train)

    # Конвертація test_data у DataFrame
    df_X_test = pd.DataFrame(X_test.numpy(), columns=['Open', 'High', 'Low'])
    df_y_test = pd.DataFrame(y_test.numpy(), columns=['Close'])
    # Об'єднання DataFrames
    df_test = pd.concat([df_X_test, df_y_test], axis=1)
    print('\nТестові дані:')
    print(df_test)
```

Результат:

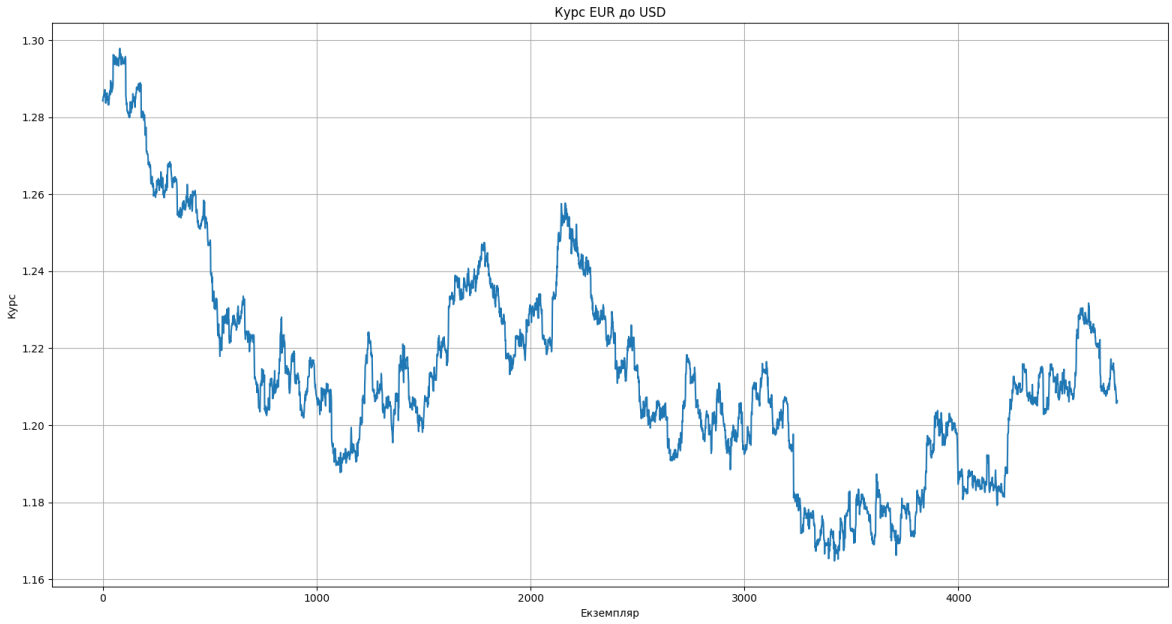

Рисунок 2 – Візуалізація датасету

Тренувальні дані:

	Open	High	Low	Close
0	1.28520	1.28520	1.28400	1.28440
1	1.28440	1.28480	1.28390	1.28420
2	1.28430	1.28540	1.28410	1.28510
3	1.28510	1.28590	1.28500	1.28510
4	1.28520	1.28590	1.28490	1.28550
...	...	...	...	...
3790	1.17133	1.17173	1.17113	1.17158
3791	1.17168	1.17361	1.17163	1.17263
3792	1.17253	1.17273	1.17073	1.17103
3793	1.17093	1.17173	1.17043	1.17123
3794	1.17123	1.17148	1.17088	1.17123

[3795 rows x 4 columns]

Тестові дані:

	Open	High	Low	Close
0	1.17113	1.17233	1.17113	1.17183
1	1.17183	1.17213	1.17153	1.17208
2	1.17203	1.17257	1.17113	1.17186
3	1.17196	1.17440	1.17156	1.17393
4	1.17393	1.17785	1.17393	1.17640
..	...	...	...	...
944	1.20778	1.20783	1.20503	1.20573
945	1.20583	1.20643	1.20553	1.20610
946	1.20610	1.20683	1.20470	1.20643
947	1.20643	1.20673	1.20553	1.20583
948	1.20563	1.20673	1.20553	1.20633

[949 rows x 4 columns]

Рисунок 3 – Розбиття датасету на тренувальний та тестовий

Архітектура нейромережі

Створимо нейромережу, яка має **3 шари**: *вхідний* (3 нейрони), *прихований* (64 нейрони) та *вихідний* (1 нейрон).

1. **Перший шар (3, 64)**:

   Функція активації: ReLU (Rectified Linear Unit).

   Кількість вхідних нейронів: 3 (відповідає кількості ознак, у цьому випадку – *відкриття*, *висока* та *низька* ціни).

   Кількість вихідних нейронів: 64.

   Лінійний шар (Linear).

2. **Другий шар (64, 32)**:

   Функція активації: ReLU (Rectified Linear Unit).

   Кількість вхідних нейронів: 64.

   Кількість вихідних нейронів: 32.

   Лінійний шар (Linear).

3. **Третій шар (32, 1)**:

   Кількість вхідних нейронів: 32.

   Кількість вихідних нейронів: 1 (відповідає прогнозованій кінцевій ціні).

   Лінійний шар (Linear).

Лістинг:

```
# Клас гібридної нейронної мережі
class HybridNN(nn.Module):
    def __init__(self):
        super(HybridNN, self).__init__()
        self.train_loss = []
        self.test_loss = []
        self.layers = nn.Sequential(
            nn.Linear(3, 64),
            nn.ReLU(),
            nn.Linear(64, 32),
            nn.ReLU(),
            nn.Linear(32, 1)
        )

# Головні виклики
if __name__ == "__main__":

    [...]
```

```
    # Створення нейромережі

    model = HybridNN()
    print('\nАрхітектура гібридної нейромережі:')
    print(model)
```

Результат:

```
Архітектура гібридної нейромережі:
HybridNN(
  (layers): Sequential(
    (0): Linear(in_features=3, out_features=64, bias=True)
    (1): ReLU()
    (2): Linear(in_features=64, out_features=32, bias=True)
    (3): ReLU()
    (4): Linear(in_features=32, out_features=1, bias=True)
  )
)
```

Рисунок 4 – Архітектура нейромережі

Навчання нейромережі:

Навчимо нейромережу на **100 епохах**. За критерій оцінки точності мережі оберемо **MSE (Mean Squared Error)** – середньоквадратичну помилку. Відобразимо процес навчання та функцію втрат.

Лістинг:

```
# Клас гібридної нейронної мережі
class HybridNN(nn.Module):
    [...]

    # Функція прямого проходу
    def forward(self, x):
        x = self.layers(x)
        return x

    # Функція тренування моделі
    def fit(self, X_train, y_train, X_test, y_test, epochs, learning_rate):
        criteria = nn.MSELoss()
        optimizer = optim.Adam(self.parameters(), lr=learning_rate)

        history = {'train_loss': [], 'test_loss': []}

        print('\nНавчання нейромережі:')
        for epoch in range(epochs):
            self.train()
            optimizer.zero_grad()
            train_loss = self.epoch_train(X_train, y_train, criteria)
```

```
            optimizer.step()
            self.eval()

            test_loss = self.epoch_test(X_test, y_test, criteria)

            history['train_loss'].append(train_loss.item())
            history['test_loss'].append(test_loss.item())

            print(f'Epoch {epoch + 1}/{epochs},\tTrain Loss: {train_loss.item()}, \tTest
Loss: {test_loss.item()}')

        return self, history

    # Функція оновлення історії втрат
    def update_history(self, train_loss, test_loss):
        self.train_loss.append(train_loss.item())
        self.test_loss.append(test_loss.item())

    # Функція отримання тренувальних втрат
    def epoch_train(self, X_train, y_train, criteria):
        train_output = self(X_train)
        train_loss = criteria(train_output, y_train)
        train_loss.backward()

        return train_loss

    # Функція отримання тестових втрат
    def epoch_test(self, X_test, y_test, criteria):
        with torch.no_grad():
            test_output = self(X_test)
            test_loss = criteria(test_output, y_test)

        return test_loss

    # Функція отримання загальних втрат
    def get_loss(self, X_test, y_test):
        self.eval()
        with torch.no_grad():
            y_pred = self(X_test)
            test_loss = nn.MSELoss()(y_pred, y_test.unsqueeze(1))

        return y_pred, test_loss.item()

# Функція відображення процесу навчання та графіка функції втрат
def plot_training_process(train_losses, test_losses):
    plt.plot(train_losses, label='Тренувальні втрати')
    plt.plot(test_losses, label='Тестові втрати')
    plt.title('Втрати навчання та тестування протягом епох')
    plt.xlabel('Епохи')
    plt.ylabel('Втрати')
    plt.legend()
    plt.show()

# Головні виклики
if __name__ == "__main__":
    [...]

    # Навчання нейромережі

    model = HybridNN()
    [...]

    model, history = model.fit(
```

```
        X_train=X_train,
        y_train=y_train,
        X_test=X_test,
        y_test=y_test,
        learning_rate=0.001,
        epochs=100
    )

    # Візуалізація процесу навчання та графіка функції втрат

    plot_training_process(history['train_loss'], history['test_loss'])
```

Результат:

```
Навчання нейромережі:
Epoch 1/100,    Train Loss: 1.6380259990692139,     Test Loss: 1.5171842575073242
Epoch 2/100,    Train Loss: 1.5506634712219238,     Test Loss: 1.4398598670959473
Epoch 3/100,    Train Loss: 1.4719997644424438,     Test Loss: 1.3655850887298584
Epoch 98/100,   Train Loss: 0.001697571249678731,   Test Loss: 0.0004180703544989228
Epoch 99/100,   Train Loss: 0.0016767005436122417,  Test Loss: 0.00040311479824595153
Epoch 100/100,  Train Loss: 0.0016616604989394546,  Test Loss: 0.0003933497646357864
```

Рисунок 5 – Навчання нейромережі

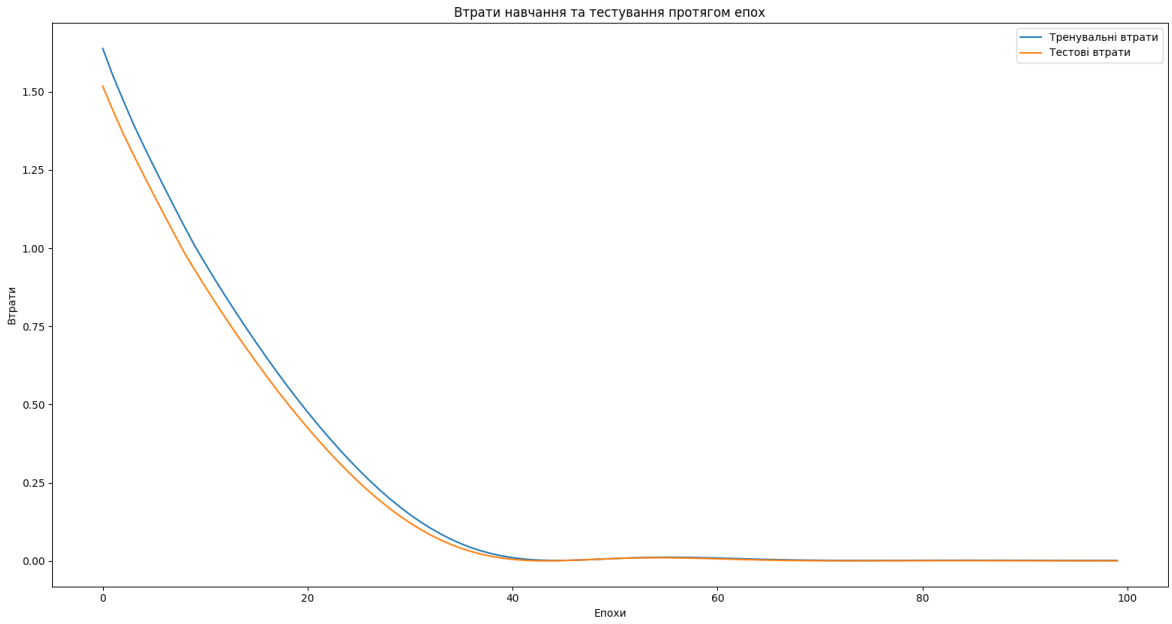

Рисунок 6 – Функція втрат

Тестування:

Перевіримо працездатність навченої нейромережі на тестовому датасеті. Для кращого порівняння результатів, виведемо їх точні значення та зобразимо їх на графіку.

Лістинг:

```
# Функція візуалізації реальних та передбачених значень
def plot_pred_compare(y_test, y_pred):
    plt.plot(y_test, label='Реальна ціна')
    plt.plot(y_pred, label='Передбачена ціна')
    plt.title('Прогнозований курс VS Реальний')
    plt.xlabel('Екземпляр')
    plt.ylabel('Курс')
    plt.grid(True)
    plt.show()

# Головні виклики
if __name__ == "__main__":
    [...]

    # Порівняння результатів

    y_pred, _ = model.get_loss(X_test, y_test)
    y_real = y_test.squeeze()
    # Конвертація у DataFrame
    df_real = pd.DataFrame(y_real.numpy(), columns=['Real'])
    df_pred = pd.DataFrame(y_pred.numpy(), columns=['Predicted'])
    # Об'єднання DataFrames
    df_combined = pd.concat([df_real, df_pred], axis=1)

    print('\nРезультати тестування:')
    print(df_combined)

    plot_pred_compare(y_real.numpy(), y_pred.detach().numpy())
```

Результат:

```
Результати тестування:
        Real  Predicted
0    1.17183   1.169667
1    1.17208   1.169985
2    1.17186   1.170111
3    1.17393   1.170772
4    1.17640   1.173251
..       ...        ...
944  1.20573   1.204157
945  1.20610   1.203032
946  1.20643   1.203071
947  1.20583   1.203374
948  1.20633   1.203047
```

Рисунок 7 – Порівняння прогнозованих та реальних значень

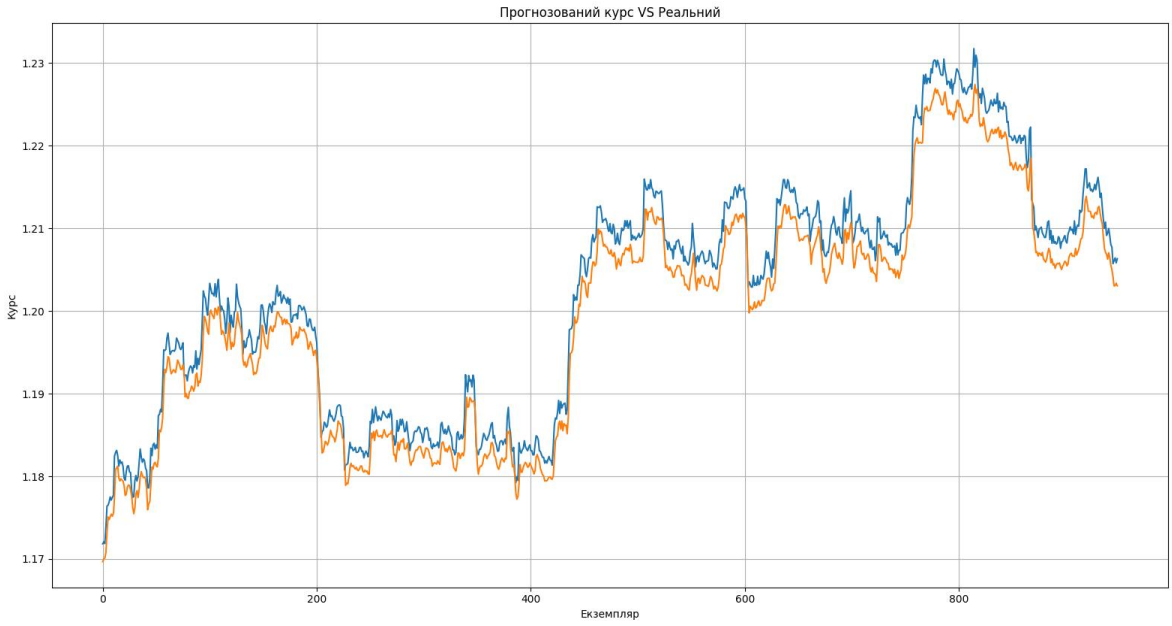

Рисунок 8 – Графік порівняння прогнозованих та реальних значень

Бачимо, що значення і графіки курсу EUR до USD доволі подібні. Отже нейромережа справляється з поставленою задачею.

Висновок:

Під час виконання лабораторної роботи було отримано теоретичні знання та практичні навички.

Досліджено застосування методів моделювання та принципів функціонування нейронечітких систем. Сформульовано задачу, яка являє собою розроблення гібридної нейронної мережі, що вирішує завдання прогнозування курсу валют.

Обрано датасет з даними курсів EUR до USD.

Створено та навчено гібридну нейронну мережу, що виконує поставлену задачу.

У результаті, створена нейромережа показала гарні результати в прогнозуванні значень показавши невелику похибку. Результати прогнозування було відображено аналітично – шляхом виведення точних значень прогнозів та реальних даних та графічно – шляхом побудови графіків порівняння.