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

Кафедра інформаційних систем та технологій

Лабораторна робота №4

з дисципліни «Методи та технології штучного інтелекту»

на тему

«Моделювання функції двох змінних з двома входами і одним виходом на основі нейронних мереж»

Виконала:
студентка групи ІС-12
Павлова Софія

Викладач:
Шимкович В. М.

Київ – 2023

1. Постановка задачі

Мета: Дослідити структуру та принцип роботи нейронної мережі. За допомогою нейронної мережі змоделювати функцію двох змінних.

Завдання:

За допомогою програмних засобів моделювання або мови програмування високого рівня створити та дослідити вплив кількості внутрішніх шарів та кількості нейронів на середню відносну помилку моделювання для різних типів мереж (feed forward backprop, cascade - forward backprop, elman backprop):

1. Тип мережі: **feed forward backprop**:
    a) 1 внутрішній шар з 10 нейронами;
    b) 1 внутрішній шар з 20 нейронами;
2. Тип мережі: **cascade-forward backprop**:
    c) 1 внутрішній шар з 20 нейронами;
    d) 2 внутрішніх шари по 10 нейронів у кожному;
3. Тип мережі: **elman backprop**:
    a) 1 внутрішній шар з 15 нейронами;
    b) 3 внутрішніх шари по 5 нейронів у кожному;
4. Зробити висновки на основі отриманих даних.

9.	$y = 0.2 \cdot \sin(3x) \cdot x^2$ $z = \sin\lvert x\rvert \cdot \sin(x + y)$	9.	22.	13.

Рисунок 1 – завдання за варіантом № 22

2. Виконання

Вхідні дані:

Обчислимо значення функції Z. У якості тренувальних даних візьмемо проміжок, де функція має декілька піків. Розіб’ємо цей проміжок на 80 значень.

Лістинг:

```
import numpy as np
import torch
from matplotlib import pyplot as plt

# Визначення функцій
def get_y(x):
    return 0.2 * np.sin(3 * x) * x**2

def get_z(x, y):
    return np.sin(np.abs(x)) * np.sin(x + y)

# Вхідні дані
def load_data():
    x_values_train = np.linspace(7.05, 7.65, 80)
    y_values_train = get_y(x_values_train)
    z_values_train = get_z(x_values_train, y_values_train)
    [...]
```

Результат:

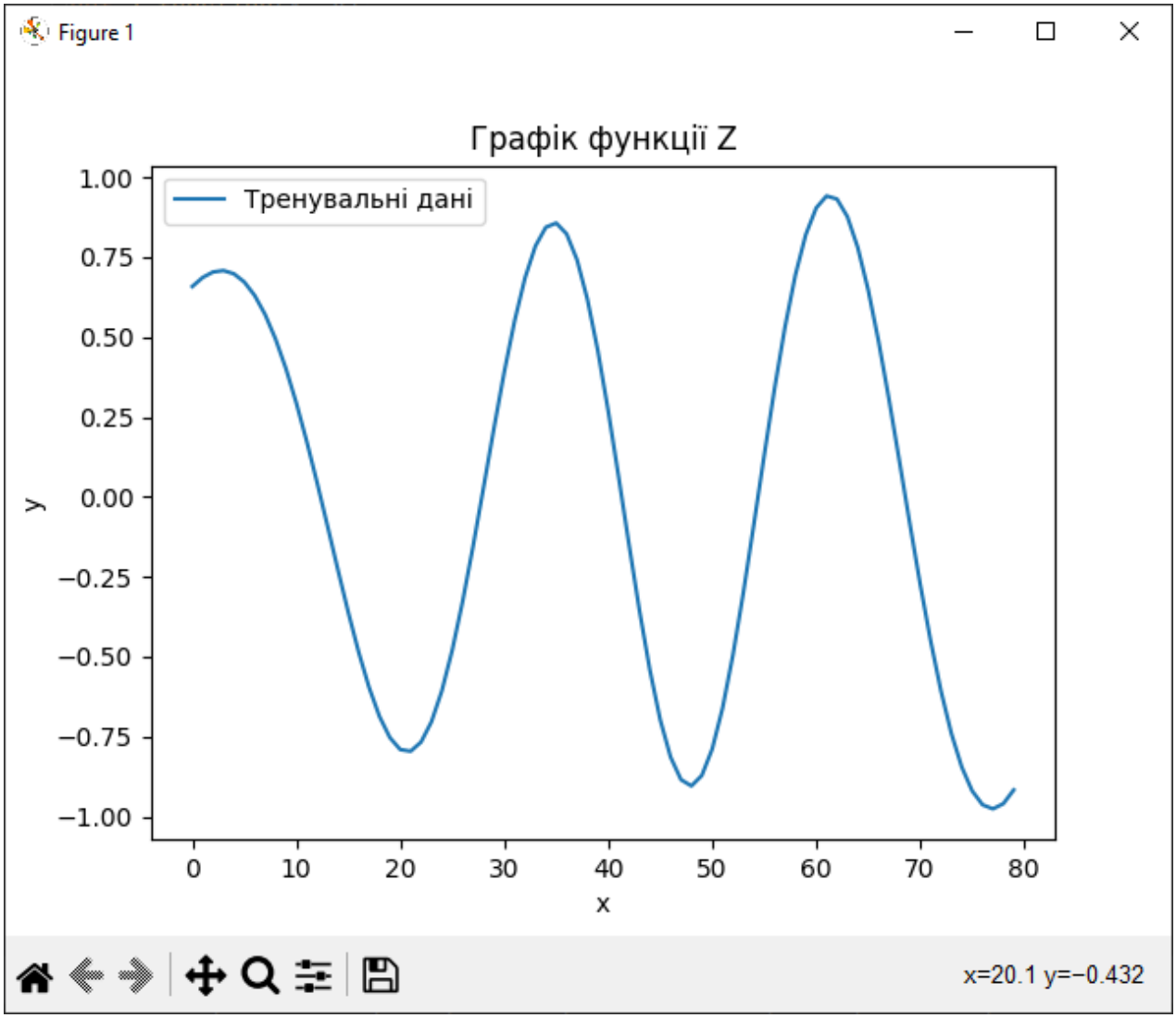

Рисунок 2 – Графік функції Z для тренування моделі

У якості тестувальних даних візьмемо проміжок, де функція має 1 пік. Розіб'ємо цей проміжок на 20 значень.

Таким чином співвідношення тренувальних до тесту вальних даних буде 80:20.

Лістинг:

```
# Вхідні дані
def load_data():
    [...]
    x_values_test = np.linspace(8.15, 8.25, 20)
    y_values_test = get_y(x_values_test)
    z_values_test = get_z(x_values_test, y_values_test)
    return np.vstack((x_values_train, y_values_train)).T, np.vstack((x_values_test,
y_values_test)).T, z_values_train, z_values_test
```

Результат:

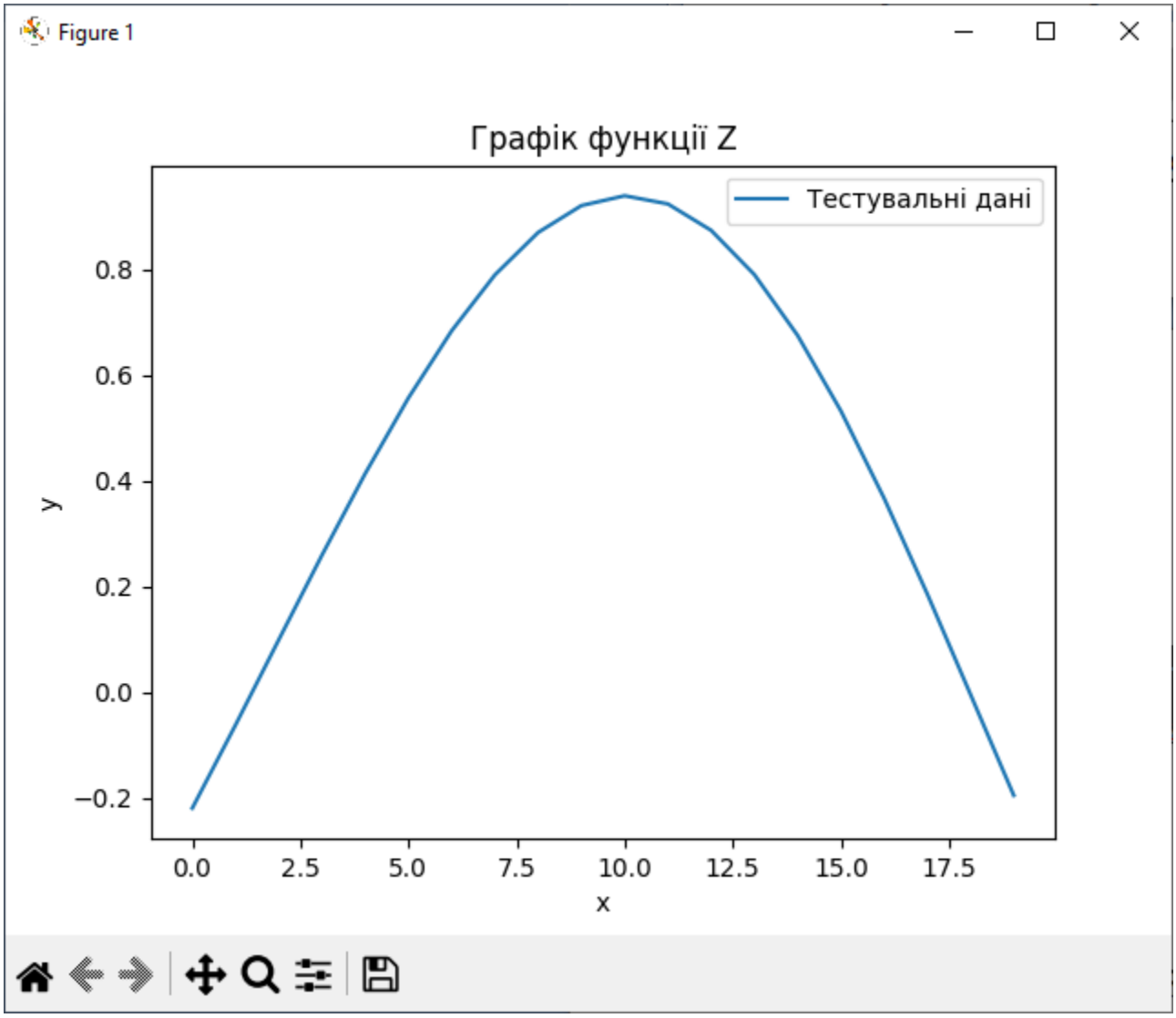

Рисунок 3 – Графік функції Z для тестування моделі

Зберемо вхідні дані до купи й переконаємося в їх правильній розмірності. Під “*x*” надалі матиметься на увазі вхідні дані *x* та *y*, а під “*y*” – вихідна функція *z*.

Лістинг:

```
# Вхідні дані
X_train, X_test, y_train, y_test = load_data()
print('Розмір вхідних даних:')
print('X_train shape:', X_train.shape, 'y_train shape:', y_train.shape)
print('X_test shape:', X_test.shape, 'y_test shape:', y_test.shape)
input_size = X_train.shape[1]
output_size = 1
```

Результат:

```
Розмір вхідних даних:
X_train shape: (80, 2) y_train shape: (80,)
X_test shape: (20, 2) y_test shape: (20,)
```

Рисунок 4 – Розмірності вхідних даних

Feed Forward Backprop:

Feed Forward Backprop – нейронна мережа з **прямим проходом** і **зворотнім поширенням помилки**. Створимо та навчимо таку нейронну мережу з **1 шаром з 10 нейронів**.

Лістинг:

```
# Функція навчання моделі
def train_model(num_epochs, inputs, targets, model, optimizer, criterion):
    for epoch in range(num_epochs):
        # Прохід вперед
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, targets)
        # Прохід назад
        loss.backward()
        # Оптимізація вагових коефіцієнтів
        optimizer.step()
        if (epoch + 1) % 1000 == 0:
            print('Епоха [' + str(epoch + 1) + '/' + str(num_epochs) + '], Втрати: ' +
str(loss.item()))
    return

# Нейромережа типу 'Feedforward'
class FeedForwardNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(FeedForwardNN, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, output_size)
    # Прохід вперед через мережу
    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        return x

# Використання 'Feedforward' з 1 шаром, N нейронами
def try_model_FeedForward(hidden_size):
    # Архітектура нейронної мережі
    print('\n---------------------------------')
    print('Модель - Feedforward')
    print('Шарів - 1')
```

```
    print('Нейронів у шарі -', hidden_size)
    print('---------------------------------')
    model = FeedForwardNN(input_size, hidden_size, output_size)
    optimizer = optim.RMSprop(model.parameters(), lr=0.001)
    criterion = nn.MSELoss()
    # Навчання моделі
    print('Навчання моделі:')
    num_epochs = 10000
    inputs = torch.Tensor(X_train)
    targets = torch.Tensor(y_train).view(-1, 1)
    functions.train_model(num_epochs, inputs, targets, model, optimizer, criterion)
    [...]

# Навчання та оцінка для моделі з 10 нейронами
try_model_FeedForward(10)
```

Результат:

```
---------------------------------
Модель - Feedforward
Шарів - 1
Нейронів у шарі - 10
---------------------------------
Навчання моделі:
Епоха [1000/10000], Втрати: 0.055051468312740326
Епоха [2000/10000], Втрати: 0.019388588145375252
Епоха [3000/10000], Втрати: 0.013504447415471077
Епоха [4000/10000], Втрати: 0.012282565236091614
Епоха [5000/10000], Втрати: 0.012004098854959011
Епоха [6000/10000], Втрати: 0.011926214210689068
Епоха [7000/10000], Втрати: 0.011902540922164917
Епоха [8000/10000], Втрати: 0.011892402544617653
Епоха [9000/10000], Втрати: 0.01188664697110653
Епоха [10000/10000], Втрати: 0.011881195940077305
```

Рисунок 5 – Навчання моделі на 10000 епохах

Зробимо передбачення значень тестувальних даних і оцінимо якість моделі, розрахувавши середню відносну помилку моделювання.

Лістинг:

```
# Функція оцінки якості нейронної мережі за критерієм - середня відносна помилка
моделювання
def eval_model(test_inputs, test_targets, model):
    # Передбачення на тестовому наборі
    predictions = model(test_inputs)
    # Розрахунок Relative Error
    abs_relative_error = torch.abs((test_targets - predictions) / test_targets)
    # Розрахунок Mean Relative Error
    mean_relative_error = torch.mean(abs_relative_error) * 100
    return mean_relative_error

# Функція виведення графіків
def plot(X_test, y_test, y_pred, title):
    plt.plot(X_test, y_test, label='Справжні значення')
    plt.plot(X_test, y_pred, label='Прогнозовані значення')
    plt.title(title)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend()
    plt.show()

# Використання 'Feedforward' з 1 шаром, N нейронами
def try_model_FeedForward(hidden_size):
    [...]
    # Оцінка якості моделі
    print('\nОцінка якості моделі:')
    model.eval()
    test_inputs = torch.Tensor(X_test)
    test_targets = torch.Tensor(y_test).view(-1, 1)
    mean_relative_error = functions.eval_model(test_inputs, test_targets, model).item()
    print('Mean Relative Percentage Error = ' + str(round(mean_relative_error, 2)) + '%')
    # Графіки
    functions.plot(X_test[:, 0], y_test, model(test_inputs).detach().numpy(),
'Feedforward 1 шар,' + str(hidden_size) + ' нейронів у шарі')

# Навчання та оцінка для моделі з 10 нейронами
try_model_FeedForward(10)
```

Результат:

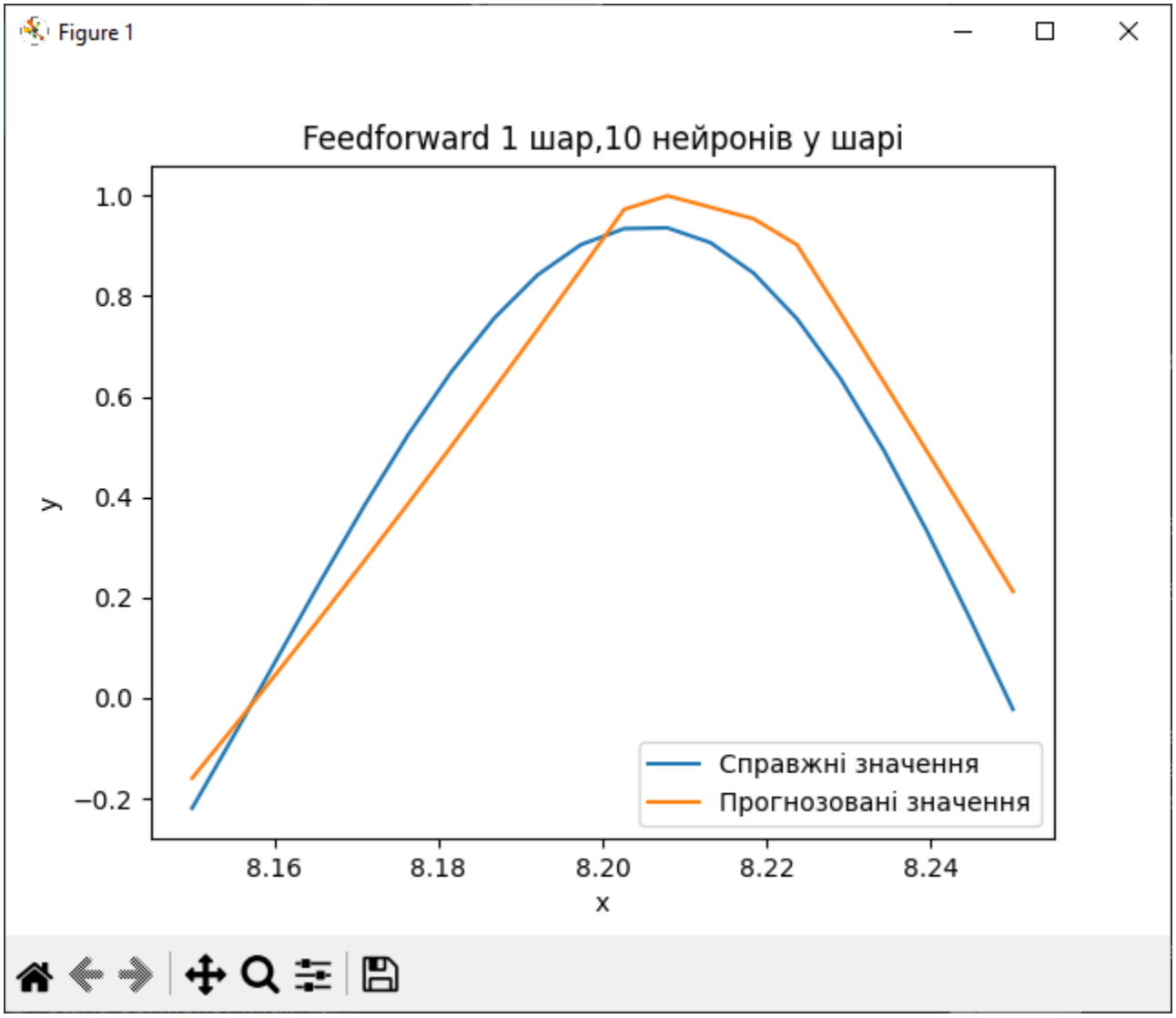

Рисунок 6 – Графік Feedforward з 1 шаром з 10 нейронів

```
Оцінка якості моделі:
Mean Relative Percentage Error = 76.69%
```

Рисунок 7 – Середня відносна помилка моделювання Feedforward з 1 шаром з 10 нейронів

Створимо, навчимо та протестуємо таку нейронну мережу з **1 шаром з 20 нейронів**.

Лістинг:

```
# Навчання та оцінка для моделі з 20 нейронами
try_model_FeedForward(20)
```

Результат:

```
--------------------------------
Модель - Feedforward
Шарів - 1
Нейронів у шарі - 20
--------------------------------
Навчання моделі:
Епоха [1000/10000], Втрати: 0.09748004376888275
Епоха [2000/10000], Втрати: 0.04172038286924362
Епоха [3000/10000], Втрати: 0.023782741278409958
Епоха [4000/10000], Втрати: 0.017711840569972992
Епоха [5000/10000], Втрати: 0.015515027567744255
Епоха [6000/10000], Втрати: 0.014569577760994434
Епоха [7000/10000], Втрати: 0.014196326956152916
Епоха [8000/10000], Втрати: 0.014125521294772625
Епоха [9000/10000], Втрати: 0.014104614034295082
Епоха [10000/10000], Втрати: 0.013954770751297474
```

Рисунок 8 – Навчання моделі на 10000 епохах

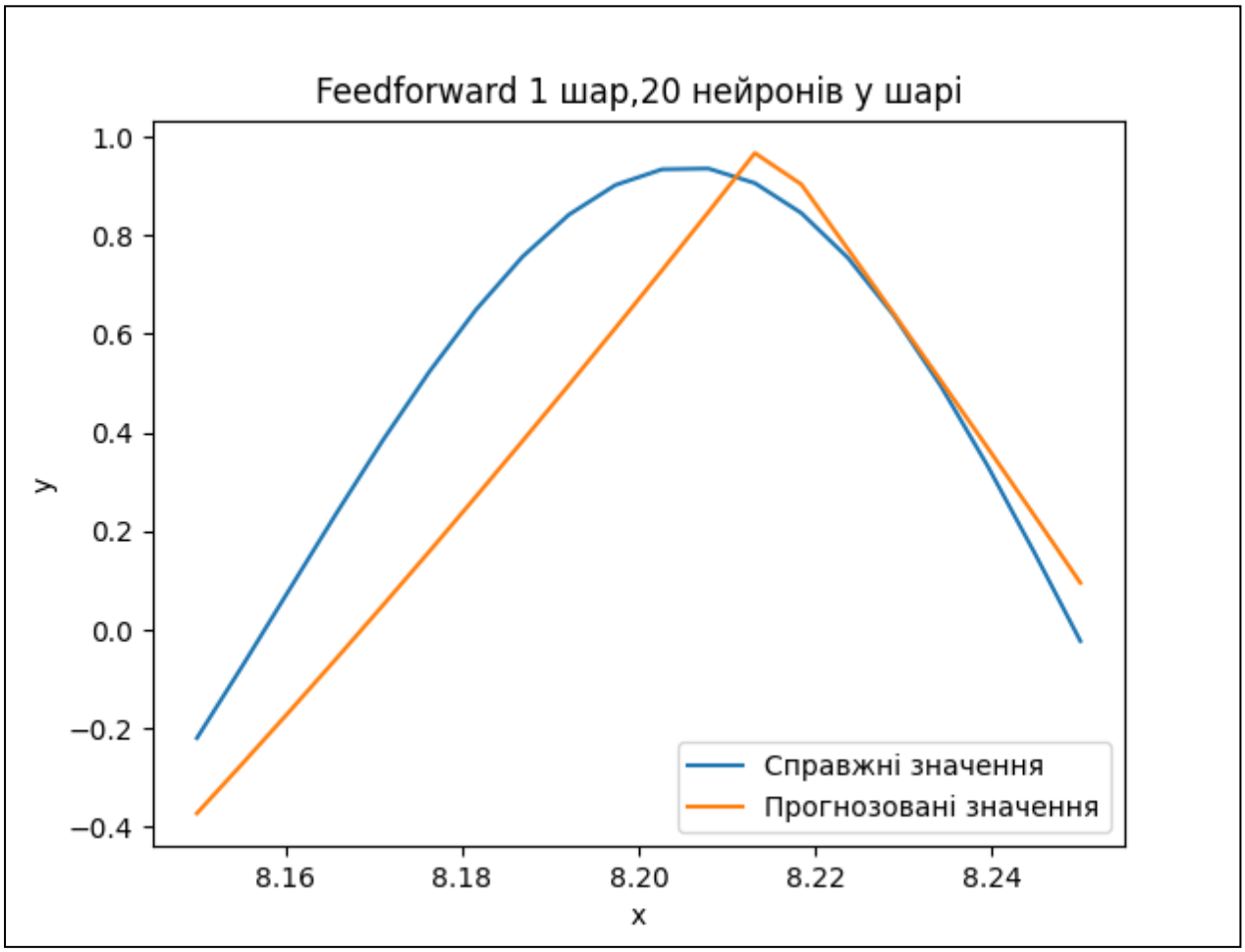

Рисунок 9 – Графік Feedforward з 1 шаром з 20 нейронів

```
Оцінка якості моделі:
Mean Relative Percentage Error = 87.13%
```

Рисунок 10 – Середня відносна помилка моделювання Feedforward з 1 шаром з 20 нейронів

Модель такої нейронної мережі доволі проста і непогано справляється з поставленою задачею як графічно, так і по визначених показниках ефективності.

Cascade-Forward Backprop:

Основна ідея Cascade-Forward Backprop полягає в тому, що мережа починається з невеликої кількості нейронів і, по мірі тренування, **автоматично розширюється** за рахунок додавання нових нейронів та шарів відповідно до потреб завдання. Створимо, навчимо та протестуємо таку нейронну мережу з **1 шаром з 20 нейронів**.

Лістинг:

```
# Нейромережа типу 'Cascadeforward'
class CascadeForwardNN(nn.Module):
    def __init__(self, input_size, hidden_sizes, output_size):
        super(CascadeForwardNN, self).__init__()
        self.layers = nn.ModuleList()
        in_features = input_size
        # Каскадне додавання шарів
        for hidden_size in hidden_sizes:
            # Лінійний шар
            self.layers.append(nn.Linear(in_features, hidden_size))
            # Активаційний шар
            self.layers.append(nn.ReLU())
            in_features = hidden_size
        # Вихідний шар
        self.layers.append(nn.Linear(in_features, output_size))
    # Прохід вперед
    def forward(self, x):
        for layer in self.layers:
            x = layer(x)
        return x

# Використання 'Cascadeforward' з N шарами по N нейронів
def try_model_CascadeForward(hidden_size):
    # Архітектура нейронної мережі
    print('\n---------------------------------')
    print('Модель - Cascadeforward')
    print('Шарів -', len(hidden_size))
    print('Нейронів у шарі -', hidden_size[0])
    print('---------------------------------')
```

```
    model = CascadeForwardNN(input_size, hidden_size, output_size)
    optimizer = optim.RMSprop(model.parameters(), lr=0.001)
    criterion = nn.MSELoss()
    # Навчання моделі
    print('Навчання моделі:')
    num_epochs = 10000
    inputs = torch.Tensor(X_train)
    targets = torch.Tensor(y_train).view(-1, 1)
    functions.train_model(num_epochs, inputs, targets, model, optimizer, criterion)
    # Оцінка якості моделі
    print('\nОцінка якості моделі:')
    model.eval()
    test_inputs = torch.Tensor(X_test)
    test_targets = torch.Tensor(y_test).view(-1, 1)
    mean_relative_error = functions.eval_model(test_inputs, test_targets, model).item()
    print('Mean Relative Percentage Error = ' + str(round(mean_relative_error, 2)) + '%')
    # Графіки
    functions.plot(X_test[:, 0], y_test, model(test_inputs).detach().numpy(),
'Cascadeforward ' + str(len(hidden_size)) + ' шарів, ' + str(hidden_size[0]) + ' нейронів
у шарі')

# Навчання та оцінка для моделі з 20 нейронами
try_model_CascadeForward([20])
```

Результат:

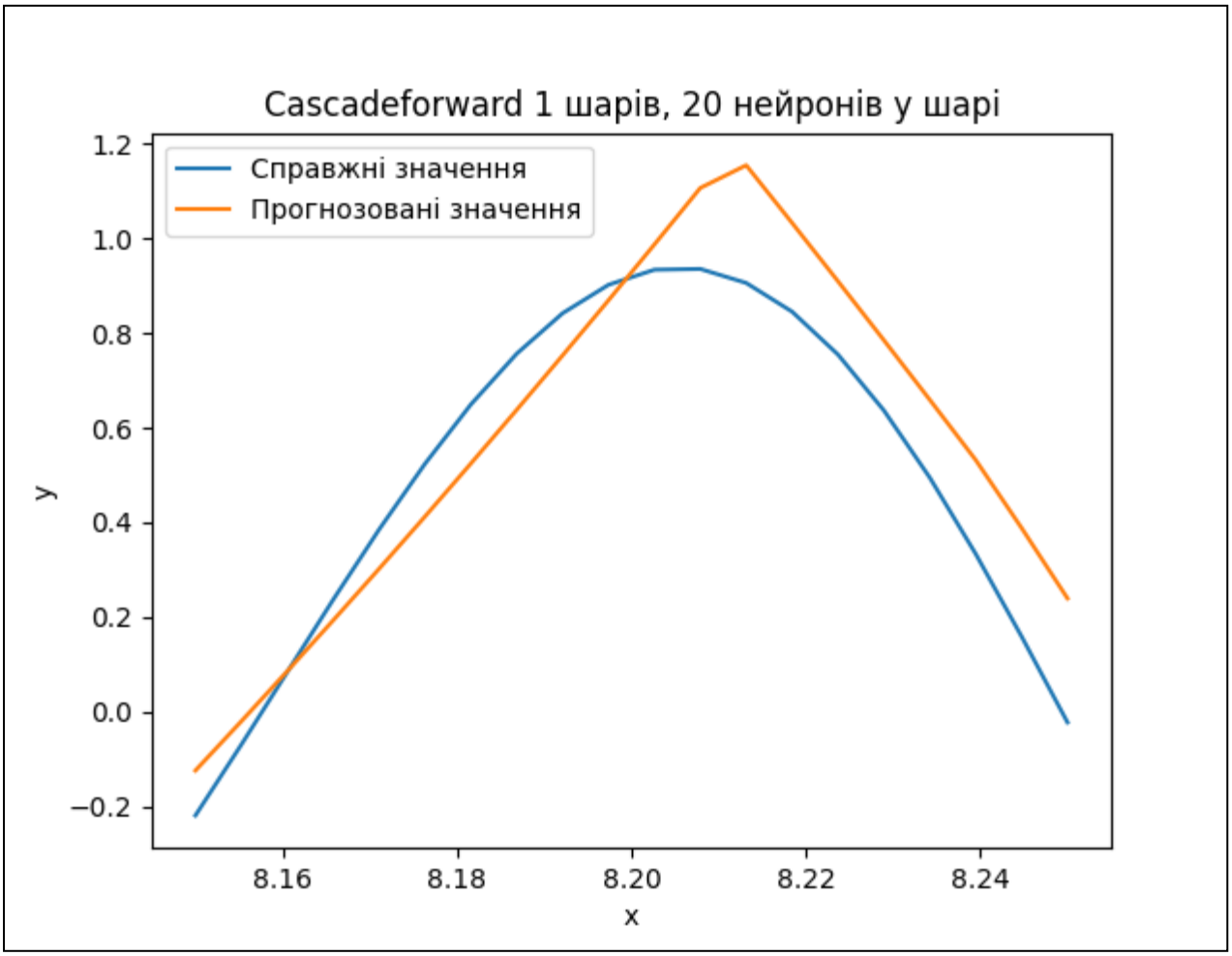

Рисунок 11 – Графік Cascadeforward з 1 шаром з 20 нейронів

```
Оцінка якості моделі:
Mean Relative Percentage Error = 86.78%
```

Рисунок 12 – Середня відносна помилка моделювання Cascadeforward з 1 шаром з 20 нейронів

Створимо, навчимо та протестуємо таку нейронну мережу з **2 шарами по 10 нейронів**.

Лістинг:

```
# Навчання та оцінка для моделі з 2 шарами по 10 нейронів
try_model_CascadeForward([10, 10])
```

Результат:

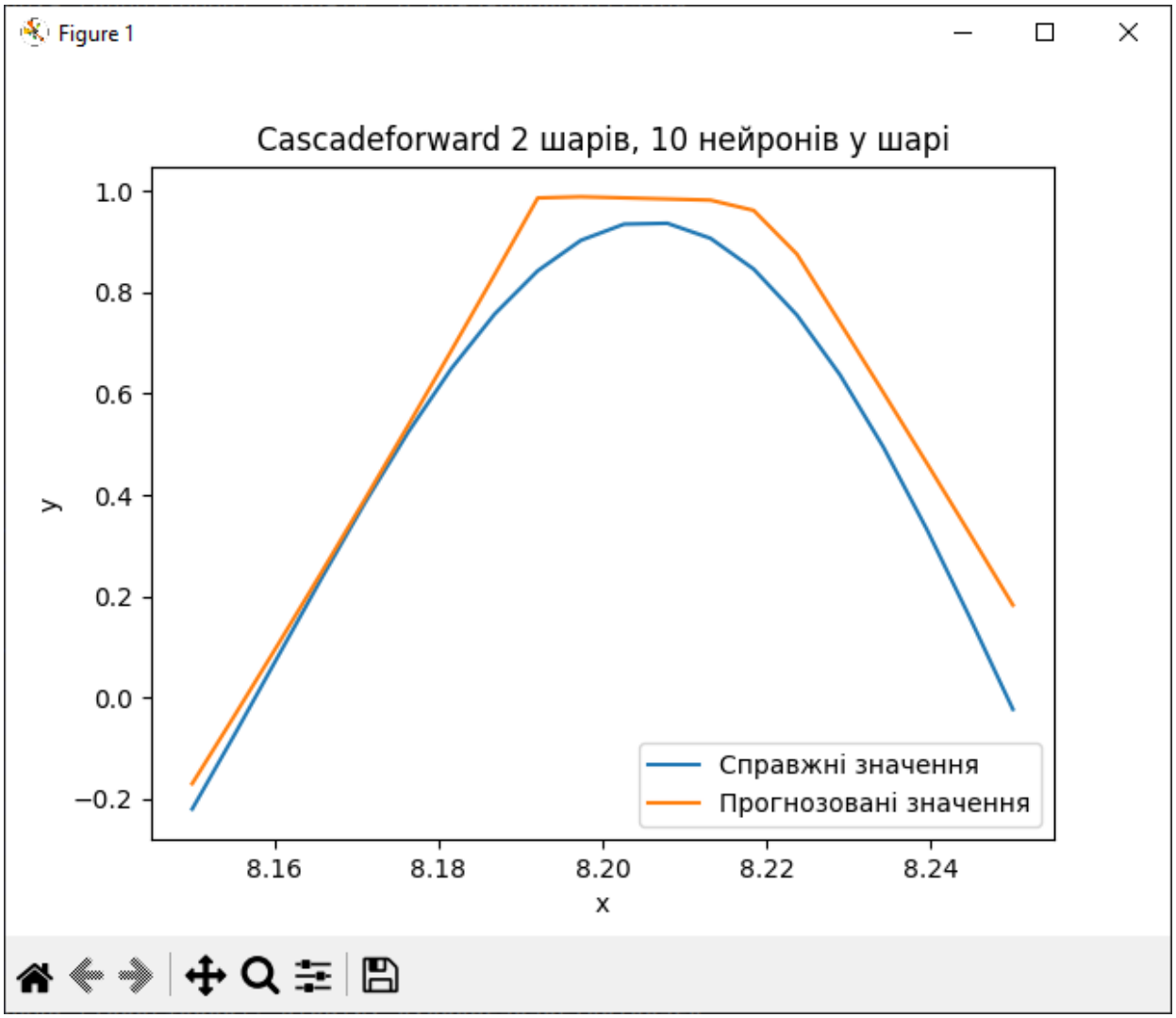

Рисунок 13 – Графік Cascadeforward з 2 шарами по 10 нейронів

```
Оцінка якості моделі:
Mean Relative Percentage Error = 64.52%
```

Рисунок 14 – Середня відносна помилка моделювання Cascadeforward з 2 шарами по 10 нейронів

Модель такої нейронної мережі трохи складніша за попередню і непогано справляється з поставленою задачею як графічно, так і по визначених показниках ефективності.

Elman Backprop:

Elman Backprop або Elman RNN (рекурентна нейронна мережа) – це вид рекурентної нейронної мережі, яка має внутрішній шар, що дозволяє **зберігати попередні стани** і використовувати їх для обробки подальших вхідних даних. Створимо, навчимо та протестуємо таку нейронну мережу з **1 шаром з 15 нейронів**.

Лістинг:

```
# Нейромережа типу 'Elman'
class ElmanNN(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, output_size):
        super(ElmanNN, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.elman_layers = nn.ModuleList()
        self.elman_layers.append(nn.RNN(input_size, hidden_size, batch_first=True))
        # RNN шари
        for i in range(num_layers - 1):
            self.elman_layers.append(nn.RNN(hidden_size, hidden_size, batch_first=True))
# Additional hidden RNN layers
        # Вихідний шар
        self.fc = nn.Linear(hidden_size, output_size)
    # Прямий прохід
    def forward(self, x):
        batch_size = x.size(0)
        # Приховані стани
        h = [torch.zeros(1, batch_size, self.hidden_size).to(x.device) for _ in
range(self.num_layers)]
        out = x
        # Прямий прохід через шари RNN, оновлення прихованих станів
        for i in range(self.num_layers):
            out, h[i] = self.elman_layers[i](out, h[i])
        out = self.fc(out[:, -1, :])
        return out
```

```
# Використання 'Elman' з N шарами по N нейронів
def try_model_Elman(num_layers, hidden_size):
    # Архітектура нейронної мережі
    print('\n---------------------------------')
    print('Модель - Elman')
    print('Шарів -', num_layers)
    print('Нейронів у шарі -', hidden_size)
    print('---------------------------------')
    model = ElmanNN(input_size, hidden_size, num_layers, output_size)
    optimizer = optim.RMSprop(model.parameters(), lr=0.001)
    criterion = nn.MSELoss()
    # Навчання моделі
    print('Навчання моделі:')
    num_epochs = 10000
    inputs = torch.Tensor(X_train).unsqueeze(1)
    targets = torch.Tensor(y_train).view(-1, 1)
    functions.train_model(num_epochs, inputs, targets, model, optimizer, criterion)
    # Оцінка якості моделі
    print('\nОцінка якості моделі:')
    model.eval()
    test_inputs = torch.Tensor(X_test).unsqueeze(1)
    test_targets = torch.Tensor(y_test).view(-1, 1)
    mean_relative_error = functions.eval_model(test_inputs, test_targets, model).item()
    print('Mean Relative Percentage Error = ' + str(round(mean_relative_error, 2)) + '%')
    # Графіки
    functions.plot(X_test[:, 0], y_test, model(test_inputs).detach().numpy(), 'Elman ' +
str(num_layers) + ' шарів, ' + str(hidden_size) + ' нейронів у шарі')

# Навчання та оцінка для моделі з 15 нейронами
try_model_Elman(1, 15)
```

Результат:

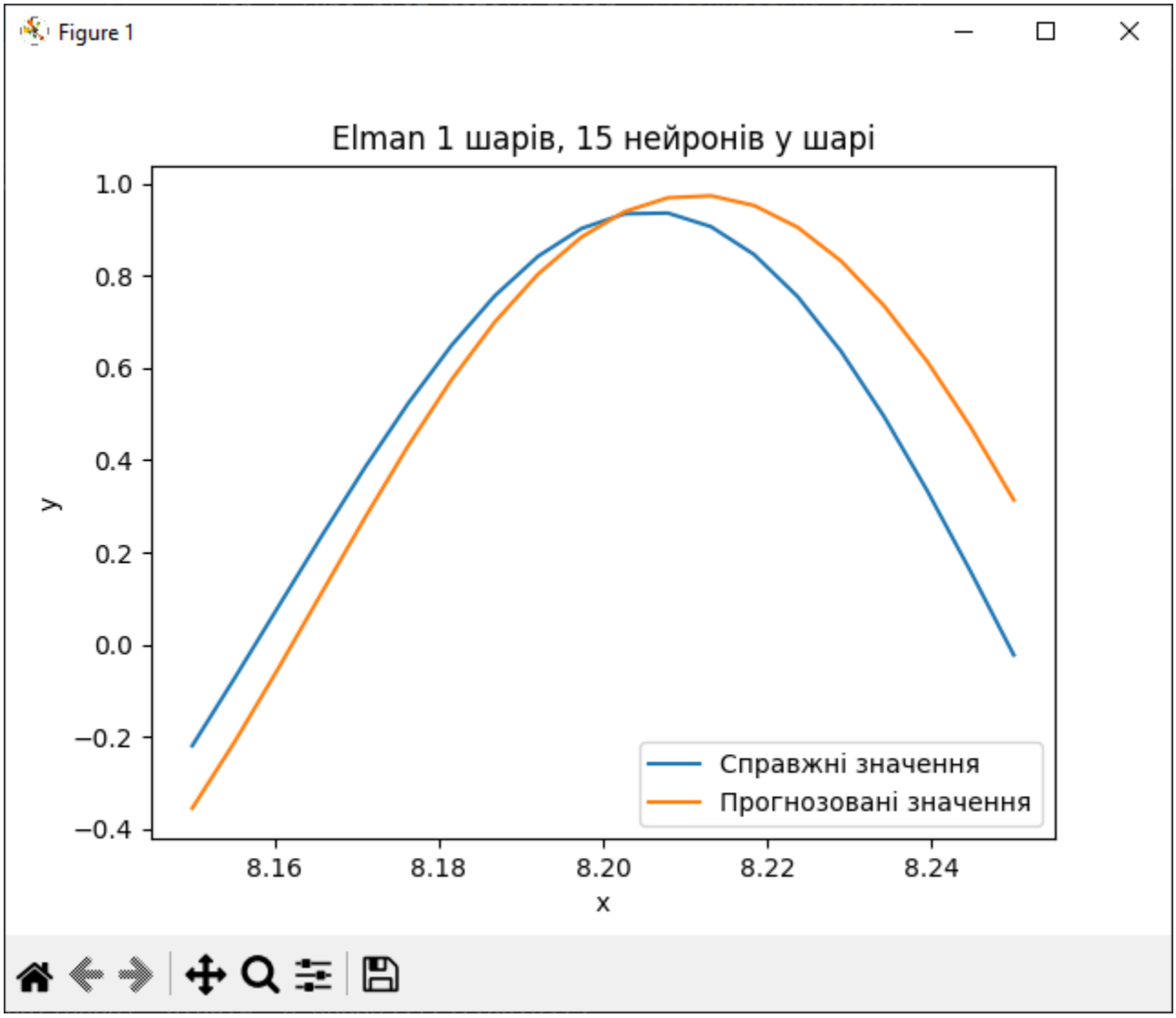

Рисунок 15 – Графік Elman з 1 шаром з 15 нейронів

```
Оцінка якості моделі:
Mean Relative Percentage Error = 121.4%
```

Рисунок 16 – Середня відносна помилка моделювання Elman з 1 шаром з 15 нейронів

Створимо, навчимо та протестуємо таку нейронну мережу з **3 шарами по 5 нейронів**.

Лістинг:

```
# Навчання та оцінка для моделі з 3 шарами по 5 нейронів
try_model_Elman(3, 5)
```

Результат:

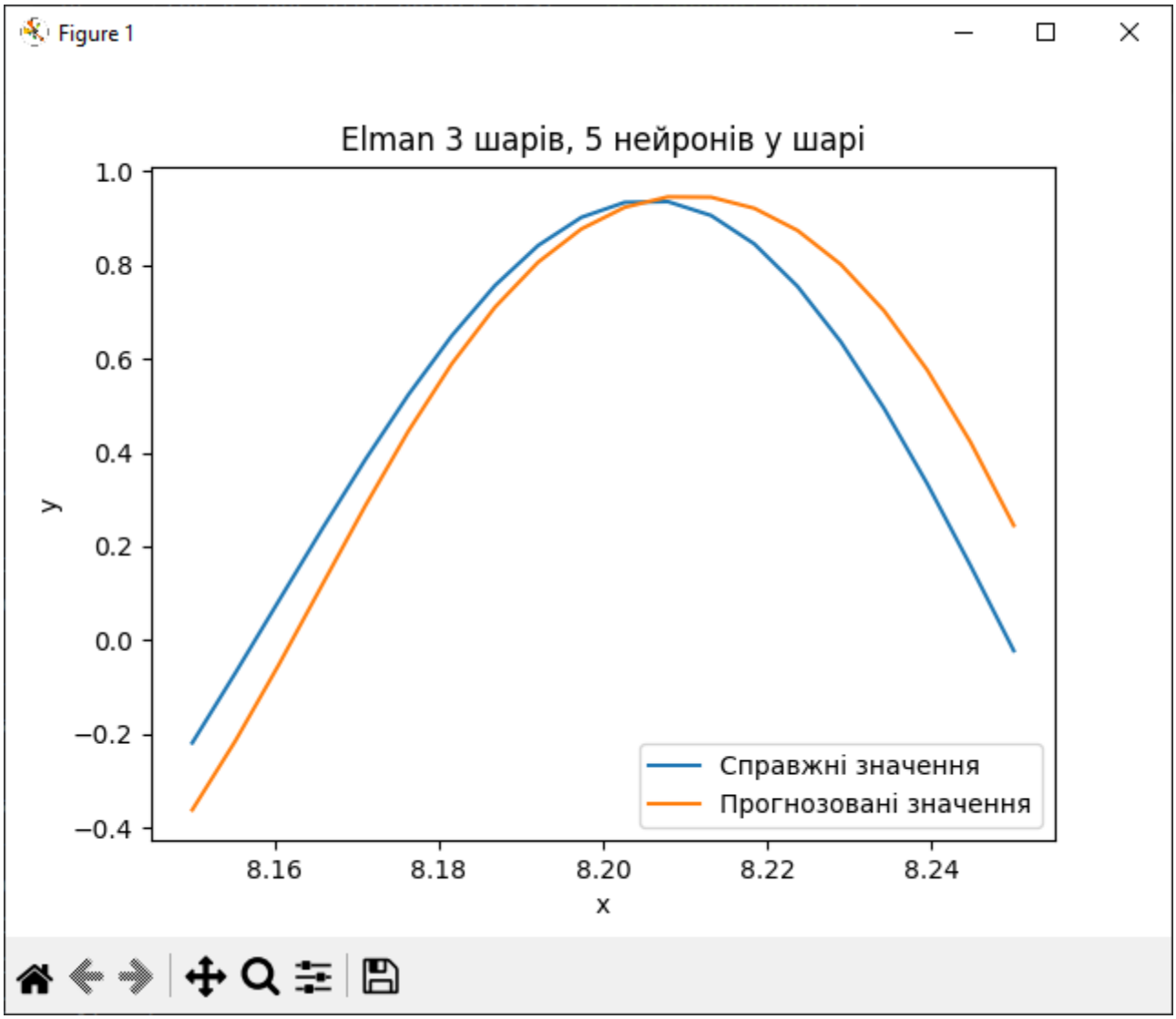

Рисунок 17 – Графік Elman з 3 шарами по 5 нейронів

```
Оцінка якості моделі:
Mean Relative Percentage Error = 103.24%
```

Рисунок 18 – Середня відносна помилка моделювання Elman з 3 шарами по 5 нейронів

Модель такої нейронної мережі не така складна як попередня і просто чудово справляється з поставленою задачею як графічно, утім по значенню середньої відносної помилки відстає від попередніх.

Аналіз отриманих результатів:

У результаті дослідження отримано наступні результати.

Таблиця 1 – Середня відносна помилка моделювання

	Feed forward		Cascade forward		Elman	
	1 шар з 10 нейронів	1 шар з 20 нейронів	1 шар з 20 нейронів	2 шари по 10 нейронів	1 шар з 15 нейронів	3 шари по 5 нейронів
MRE	76,69%	87,13%	86,78%	64,52%	121,4%	103,24%

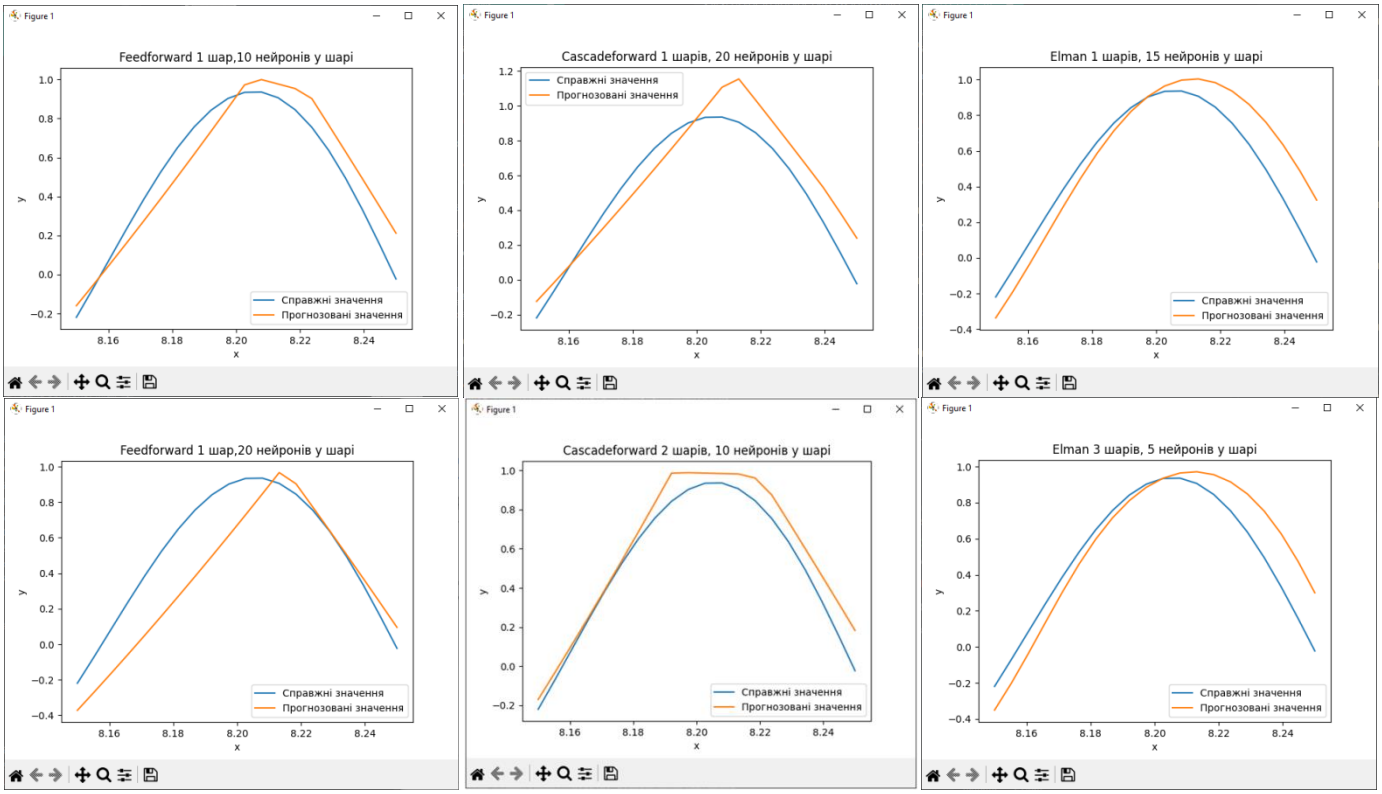

Рисунок 19 – Графіки відповідності реальних даних прогнозованим значенням

За загальними відомостями, збільшення кількості прихованих шарів у нейромережах усіх трьох розглянутих типів призведе до покращення результату. На покращення якості моделі також позитивно впливає й збільшення кількості нейронів у кожному шарі, але воно має менший вплив за кількість прихованих шарів. У обох випадках слід уникати перенавчання.

За отриманими результатами по критерію ефективності – відносній середній оцінці моделювання найкраще себе показала на даному наборі даних нейромережа типу **Cascade forward backprop** з **2 шарами по 10 нейронів**.

По відтворенню початкової структури даних найкращий результат дала нейромережа типу **Elman**, так як вона при навчанні зберігає в пам'яті попередні значення функції. Утім по графіках добре видно, що прогнозована крива трохи зміщена вбік, що призводить до великого значення середньої відносної помилки.

Висновок:

У даній лабораторній роботі я дослідила структуру та принцип роботи нейронних мереж типу Feed forward backprop, Cascade forward backprop та Elman backprop.

Розробила 6 нейромереж 3 різних типів з різною архітектурою за допомогою бібліотек мови програмування python. Змоделювала функцію двох змінних за варіантом індивідуального завдання та оцінила середню відносну похибку моделювання для кожної з них.

Провела аналіз отриманих результатів, у ході якого визначила, що збільшення як шарів так і нейронів у моделях призводить до покращення результатів. Найкращою нейромережею для заданої задачі виявилась **Cascade forward backprop** з **2 шарами по 10 нейронів**.