МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи №4 з дисципліни

«Методи та технології штучного інтелекту»

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

Перевірив:
Шимкович В.М.

Виконав:
студент 3 курсу
групи ІП-11 ФІОТ
Прищепа В.С.

Київ-2023

Лабораторна робота №4

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

Мета роботи: 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:
    a) 1 внутрішній шар з 20 нейронами;
    b) 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.

Хід роботи:

Варіант 74 = 14

14.	$y = \cos(x/2) + \sin(x/3)$ $z = 0.5\sin(x+y)\cdot\cos(y)$	14.	17.	3.

Для тренування та тестування ми візьмемо проміжки від 0 до 20 і від 20 до 25 відповідно, розбивши їх на 80 і 20 значень відповідно. Далі виконаємо моделювання і тестування відповідно до вказаних варіантів нейронних мереж і зробимо порівняння.

Лістинг:

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

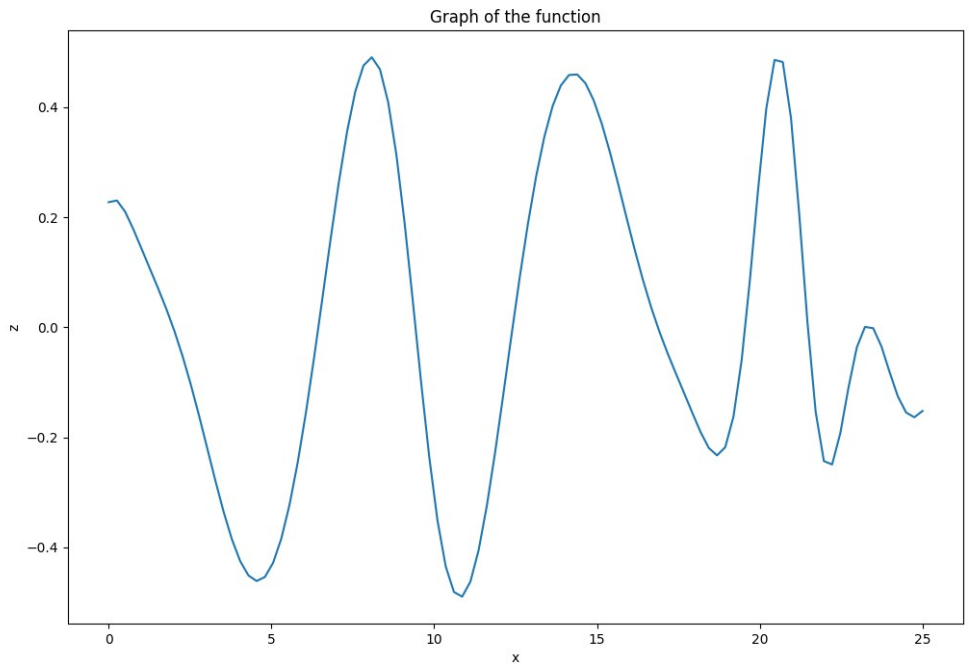
```
from tensorflow.keras.layers import Input, Dense, Add
from tensorflow.keras.models import Model
from tensorflow.keras.layers import SimpleRNN
from sklearn.metrics import mean_absolute_error
import matplotlib.pyplot as plt

x_trn = np.linspace(0, 20, 80)
x_tst = np.linspace(20, 25, 20)
y = lambda x: np.cos(x / 2) + np.sin(x / 3)
z = lambda x: 0.5 * np.sin(x + y(x)) * np.cos(y(x))
y_trn = y(x_trn)
y_tst = y(x_tst)
z_trn = z(x_trn)
z_tst = z(x_tst)
X_train = np.vstack((x_trn, y_trn)).T
X_test = np.vstack((x_tst, y_tst)).T
Y_train = z_trn
Y_test = z_tst
def plot(Y_pred, label):
    plt.figure(figsize=(12, 8))
    plt.plot(x_tst, Y_test, label='Real data')
    plt.plot(x_tst, Y_pred, label=f'{label}')
    plt.xlabel('x')
    plt.ylabel('z')
    plt.title('Model`s result vs real data')
    plt.legend()
    plt.show()

plt.figure(figsize=(12, 8))
plt.plot(np.linspace(0, 25, 100), z(np.linspace(0, 25, 100)))
plt.xlabel('x')
plt.ylabel('z')
plt.title('Graph of the function')
plt.show()

def train_model(layers, neurons):
    model = Sequential()
    model.add(Dense(neurons, input_dim=2, activation='relu'))
    for _ in range(1, layers):
      model.add(Dense(neurons, activation='relu'))
    model.add(Dense(1, activation='linear'))

    model.compile(optimizer='adam', loss='mean_squared_error')
    history = model.fit(X_train, Y_train, epochs=100, batch_size=10, verbose=0,
validation_split=0.2)
```

```
    Y_pred = model.predict(X_test)
    error = mean_absolute_error(Y_test, Y_pred)
    return history, error, Y_pred

results = {}
configurations = [(1, 10), (1, 20)]
for layers, neurons in configurations:
    history, error, pred = train_model(layers, neurons)
    results[f'Feed Forward {layers} Layer(s), {neurons} Neurons'] = error
    plot(pred, f'Feed Forward {layers} Layer(s), {neurons} Neurons')
for configuration, result in results.items():
    print(f"{configuration}: {result}")

def train_cascade_model(layers, neurons):
    inputs = Input(shape=(2,))
    resized_input = Dense(neurons, activation='relu')(inputs)
    x = Dense(neurons, activation='relu')(resized_input)
    for _ in range(1, layers):
      x = Dense(neurons, activation='relu')(x)
      x = Add()([x, resized_input])
    outputs = Dense(1, activation='linear')(x)
    model = Model(inputs=inputs, outputs=outputs)
    model.compile(optimizer='adam', loss='mean_squared_error')
    history = model.fit(X_train, Y_train, epochs=100, batch_size=10, verbose=0,
validation_split=0.2)

    Y_pred = model.predict(X_test)
    error = mean_absolute_error(Y_test, Y_pred)
    return history, error, Y_pred

cascade_results = {}
cascade_configurations = [(1, 20), (2, 10)]
for layers, neurons in cascade_configurations:
    history, error, pred = train_cascade_model(layers, neurons)
    cascade_results[f'Cascade Forward {layers} Layer(s), {neurons} Neurons'] = error
    plot(pred, f'Cascade Forward {layers} Layer(s), {neurons} Neurons')
for configuration, result in cascade_results.items():
    print(f"{configuration}: {result}")

def train_elman_model(layers, neurons):
    model = Sequential()
    model.add(SimpleRNN(neurons, input_shape=(1, 2), return_sequences=True if
layers > 1 else False))
    for _ in range(1, layers):
```

```
        model.add(SimpleRNN(neurons, return_sequences=True if _ < layers - 1 else
False))
    model.add(Dense(1, activation='linear'))
    model.compile(optimizer='adam', loss='mean_squared_error')
    X_train_expanded = np.expand_dims(X_train, 1)
    X_test_expanded = np.expand_dims(X_test, 1)
    history = model.fit(X_train_expanded, Y_train, epochs=100, batch_size=10,
verbose=0, validation_split=0.2)
    Y_pred = model.predict(X_test_expanded)
    error = mean_absolute_error(Y_test, Y_pred)
    return history, error, Y_pred

elman_results = {}
elman_configurations = [(1, 15), (3, 5)]
for layers, neurons in elman_configurations:
    history, error, pred = train_elman_model(layers, neurons)
    elman_results[f'Elman {layers} Layer(s), {neurons} Neurons'] = error
    plot(pred, f'Elman {layers} Layer(s), {neurons} Neurons')
for configuration, result in elman_results.items():
    print(f"{configuration}: {result}")
```

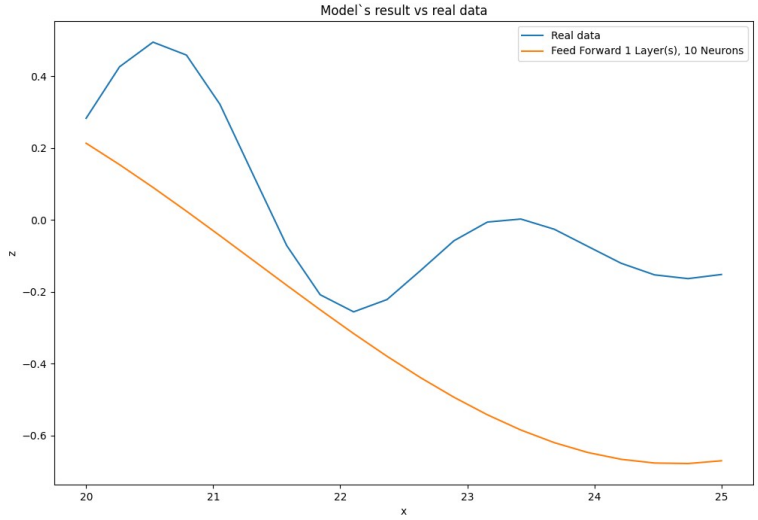
Результат виконання програми:

Графік функції Z від X на проміжку від x = 0 до x = 25:

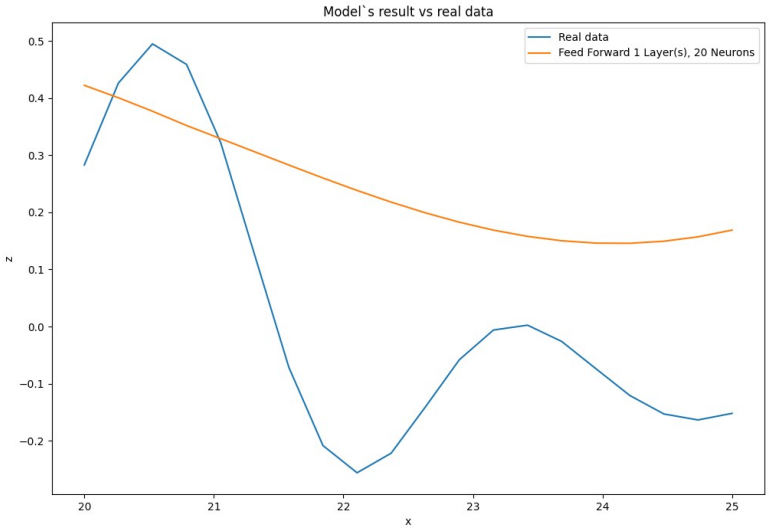


1) Тип мережі: feed forward backprop:

a) 1 внутрішній шар з 10 нейронами;



b) 1 внутрішній шар з 20 нейронами;

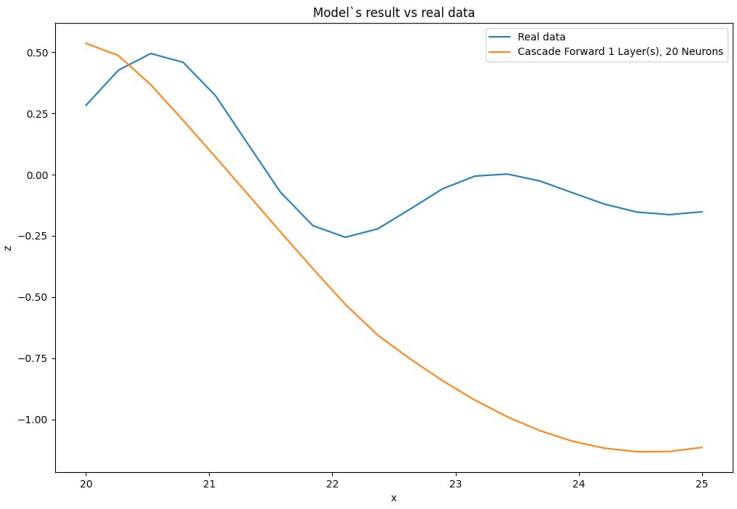


Абсолютні середні похибки:

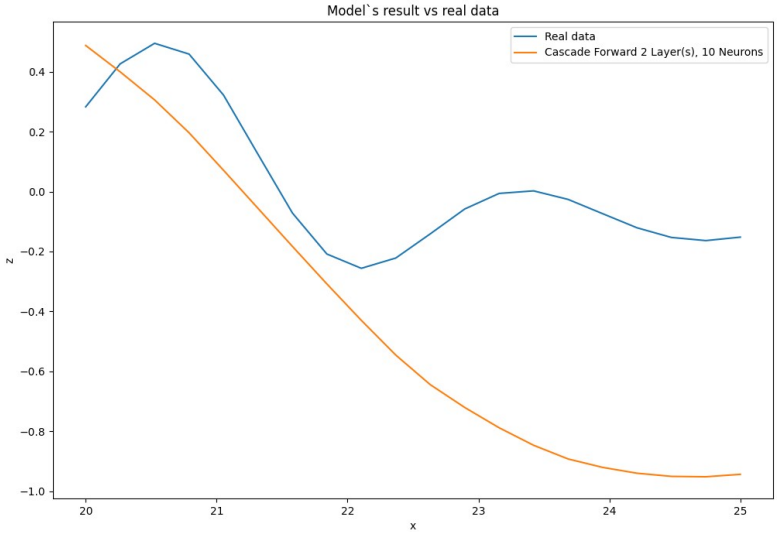
```
Feed Forward 1 Layer(s), 10 Neurons: 0.364286235498754
Feed Forward 1 Layer(s), 20 Neurons: 0.2426899197918196
```

2) Тип мережі: cascade - forward backprop:

a) 1 внутрішній шар з 20 нейронами;



b) 2 внутрішніх шари по 10 нейронів у кожному;

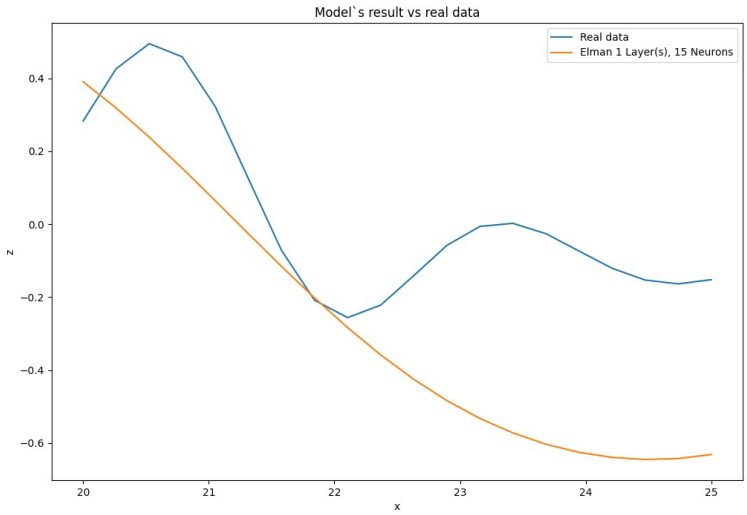


Абсолютні середні похибки:

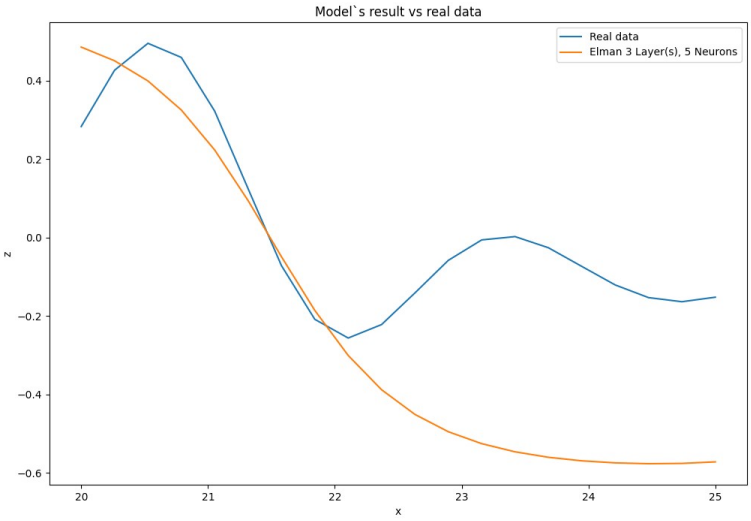
```
Cascade Forward 1 Layer(s), 20 Neurons: 0.5715260573707205
Cascade Forward 2 Layer(s), 10 Neurons: 0.476553550914819
```

3) Тип мережі: elman backprop:

a) 1 внутрішній шар з 15 нейронами;



b) 3 внутрішніх шари по 5 нейронів у кожному;



Абсолютні середні похибки:

```
Elman 1 Layer(s), 15 Neurons: 0.3154476806252477
Elman 3 Layer(s), 5 Neurons: 0.26966225533727517
```

Висновок:

Під час виконання лабораторної роботи я дослідив вплив кількості внутрішніх шарів та кількості нейронів на середню абсолютну помилку моделювання для різних типів мереж (feed forward backprop, cascade forward backprop, elman backprop). Я помітив наступне: мережі з більшою кількістю шарів та нейронів показали кращий результат. Найкращими виявилися feed forward backprop з 1 внутрішнім шаром та 20 нейронами та elman backprop з 3 внутрішніми шарами по 5 нейронів у кожному.