Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

Лабораторна робота №4

Методи і технології штучного інтелекту

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

Виконала:

студентка групи ІА-34

Мартинюк Т.В.

Перевірила:

Польшакова О. М.

Київ – 2025

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

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

Хід роботи:

1. Імпортуємо усі необхідні модулі, для x обираємо значення від -5 до 5, обраховуємо y та z за формулами відповідно до завдання.

```
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential, Model
from keras.layers import Input, Dense, Concatenate, SimpleRNN, Reshape

x = np.linspace(-5, stop: 5, num: 1000)
y = x * np.cos(x) + np.sin(x)
z = np.sin(x) + np.cos(y / 2)
```

2. Оскільки будемо тестувати 6 різних нейронних мереж, то створюємо функцію з моделлю як параметром для навчання, тестування та візуалізації результату. Навчаємо модель на 20 епохах з розміром сегменту 100.

```
def modelTesting(model):
    model.compile(optimizer='adam', loss='mse')
    model.fit(x, z, epochs=20, batch_size=100)
    z_pred = model.predict(x)

    plt.plot( *args: x, z, label='Actual')
    plt.plot( *args: x, z_pred, label='Predicted')
    plt.legend()
    plt.show()
```

3. Створюємо мережу feedforward backprop. Відповідно маємо sequential модель з кількома щільними шарами Dense, кількість шарів задається. Вихідний шар також Dense з 1 нейроном.

```
def feedforwardCreation(layers, neurons):
    model = Sequential()
    model.add(Dense(neurons, activation='relu', input_shape=(1,)))
    for i in range(layers - 1):
        model.add(Dense(neurons, activation='relu'))
    model.add(Dense( units: 1, name='output'))
    return model
```

4. Наступною створюємо cascadeforward backprop. Відповідно створюємо вхідний шар, за ним створюю Dense, пов'язаний з вхідним. І далі відповідно до кількості шарів створюю наступні шари, які за допомогою Concatenate пов'язані з усіма попередніми.

```
2 usages
def cascadeforwardCreation(layers, neurons):
    inputLayer = Input(shape=(1,), name='input')
    current = Dense(neurons, activation='relu', input_shape=(1,))(inputLayer)
    for i in range(layers - 1):
        concatenatedLayer = Concatenate()([inputLayer, current])
        current = Dense(neurons, activation='relu',
                        input_shape=(1,))(concatenatedLayer)
    outputLayer = Dense( units: 1, name='output')(current)
    model = Model(inputs=inputLayer, outputs=outputLayer)
    return model
```

5. Останньою мережею створюємо Elman backprop. Мережа Elman схожа на feedforward, але з додаванням контекстного шару. Контекстний шар отримує дані від прихованого шару і, у свою чергу, передає свій вихід назад у прихований шар. Ця зворотна петля надає мережі форму пам'яті.

```
2 usages
def elmanCreation(layers, neurons):
    model = Sequential()
    model.add(Reshape( target_shape: (1, 1), input_shape=(1,), name="input_reshape"))
    model.add(SimpleRNN(neurons, return_sequences=True, activation='relu',
                        input_shape=(1,)))
    for i in range(layers - 1):
        model.add(SimpleRNN(neurons, return_sequences=True, activation='relu'))
    model.add(Dense( units: 1, name='output'))
    model.add(Reshape( target_shape: (1,), input_shape=(1, 1), name='output_reshape'))
    return model
```

6. Далі створюємо власні мережі відповідно до завдання. У кожній мережі перший параметр – кількість шарів, другий – кількість нейронів у шарі.

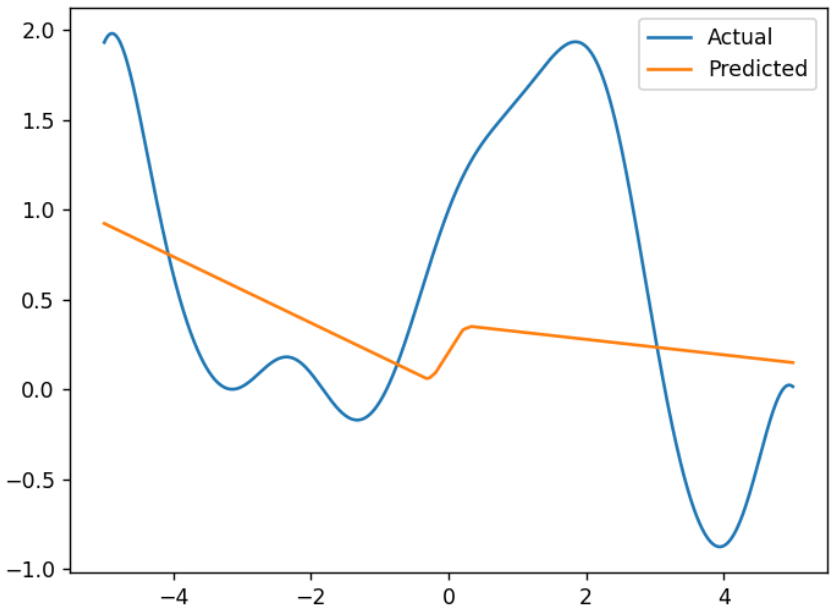
```
if __name__ == "__main__":
    f1 = feedforwardCreation( layers: 1,  neurons: 10)
    f2 = feedforwardCreation( layers: 1,  neurons: 20)

    c1 = cascadeforwardCreation( layers: 1,  neurons: 20)
    c2 = cascadeforwardCreation( layers: 2,  neurons: 10)

    e1 = elmanCreation( layers: 1,  neurons: 15)
    e2 = elmanCreation( layers: 3,  neurons: 5)

    modelTesting(f1)
    modelTesting(f2)

    modelTesting(c1)
    modelTesting(c2)

    modelTesting(e1)
    modelTesting(e2)
```

7. Отримуємо такі результати:

1. Feedforward NN:

a. 1 layer, 10 neurons:

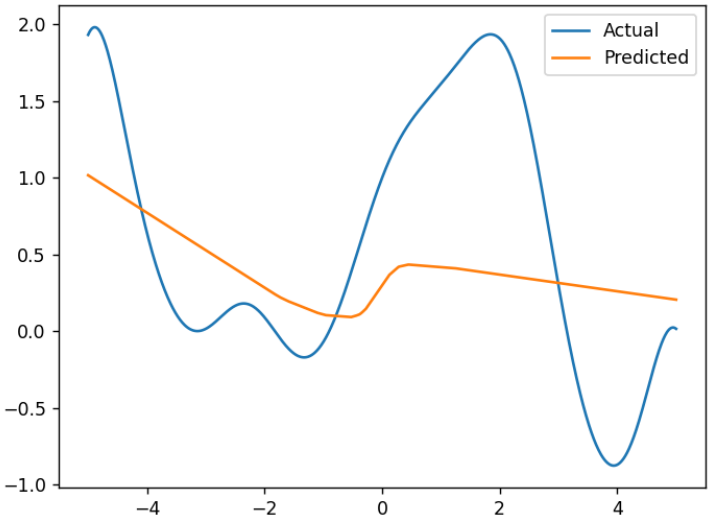
```
Epoch 20/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.7032
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```



Можемо побачити, що результат досить хороший, навіть для такої нескладної нейронної мережі.

b. 1 layer, 20 neurons:

```
Epoch 20/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.6382
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
```

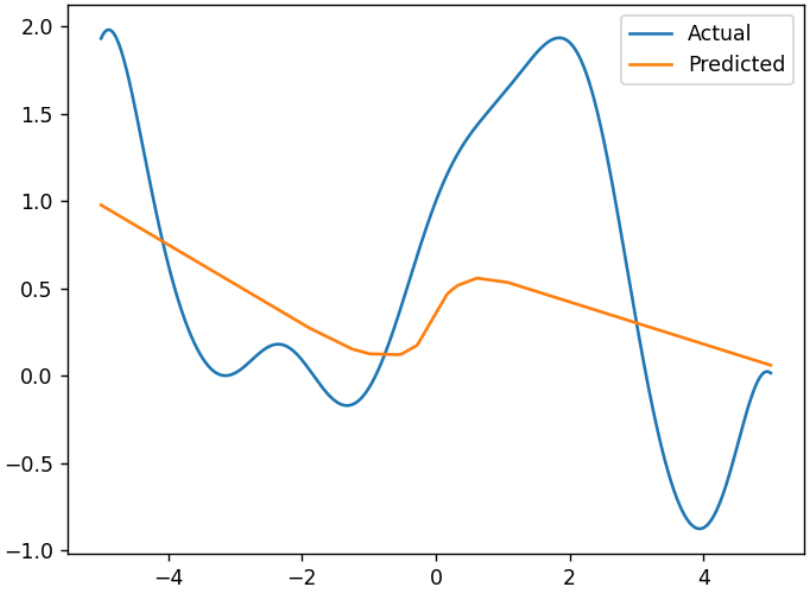


Бачимо, що збільшення кількості нейронів покращує якість моделювання та зменшує похибку.

2. Cascade forward NN:

a. 1 layer, 20 neurons:

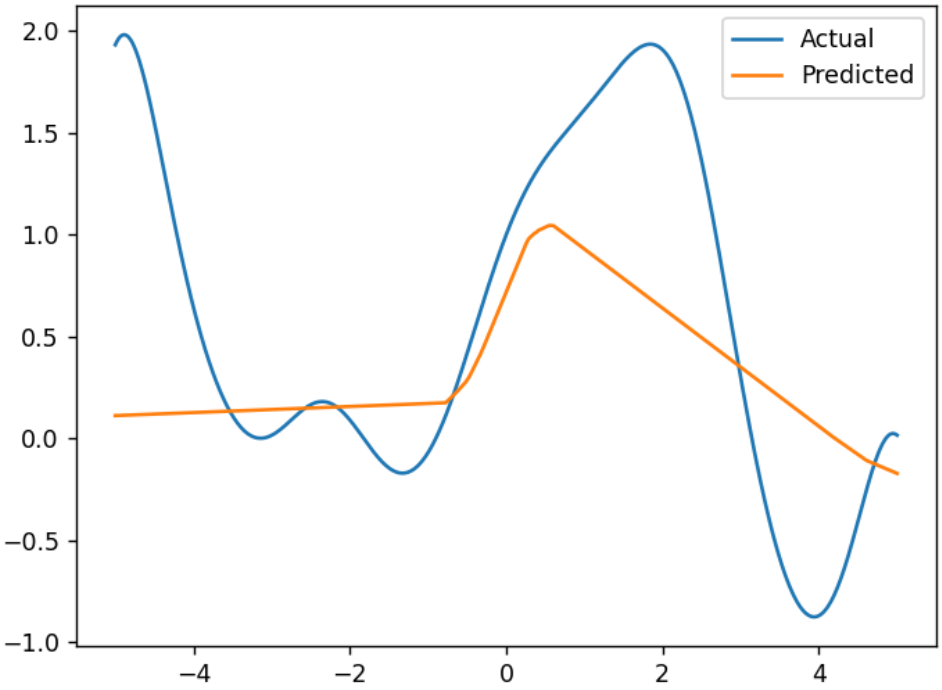
```
Epoch 20/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.5758
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```



Каскадна нейронна мережа показує ще менше похибку через те, що кожен шар отримує також вхід напряму від усіх попередніх шарів та створює “каскадну” структуру – додаткові шляхи для передачі інформації.

b. 2 layers, 10 neurons:

```
Epoch 20/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.5221
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```



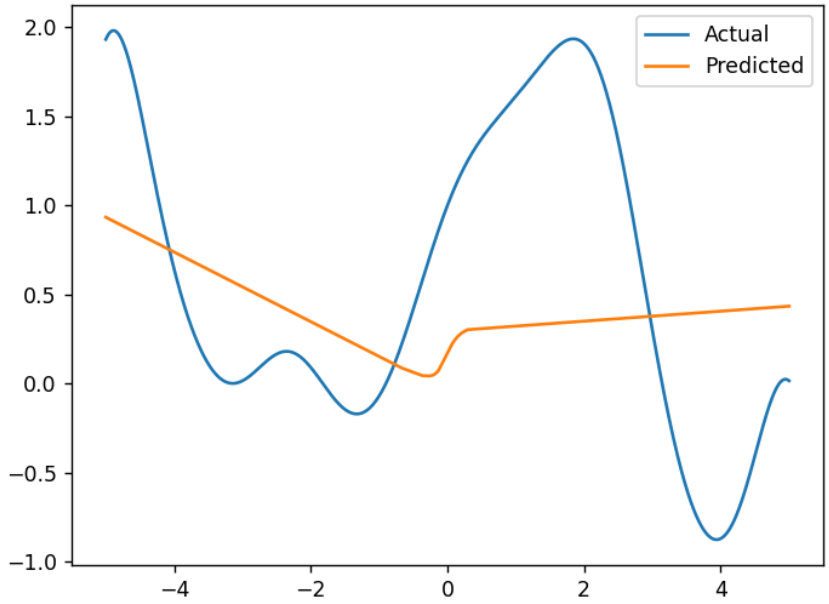
З отриманих результатів можемо побачити що, збільшення кількості шарів нейронів значно покращує якість моделювання та зменшує похибку.

Elman NN:

a. 1 layer, 15 neurons:

```
Epoch 20/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 0.7451
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
```

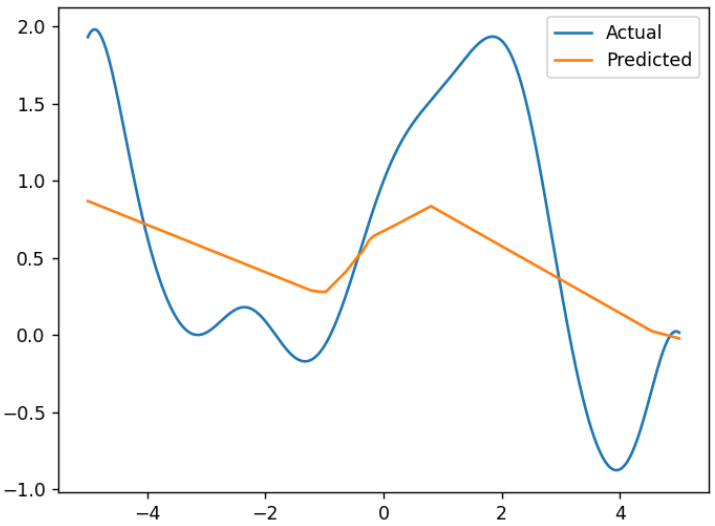


Можемо побачити, що рекурентна нейронна мережа Ельмана з малою кількістю шарів дає найгірший результат та найбільшу похибку.

b. 3 layers, 5 neurons:

```
Epoch 20/20
10/10 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 0.4820
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step
```



З отриманих результатів можемо зробити висновок, що рекурента нейронна мережа Ельмана зі збільшенням кількості шарів дає найкращий результат та

найменшу похибку, адже вона має зворотні зв’язки – тобто частина виходу одного кроку зберігається як контекст для наступного

Висновок: Проведене моделювання показало, що архітектура нейронної мережі та кількість її параметрів суттєво впливають на якість апроксимації. Звичайна Feedforward Neural Network навіть із невеликою кількістю нейронів здатна відтворювати залежності у даних, однак збільшення кількості нейронів у прихованому шарі підвищує точність завдяки більшій здатності мережі узагальнювати складніші нелінійні функції.

Cascade Forward Neural Network продемонструвала ще кращі результати, оскільки кожен шар отримує не лише виходи попередніх, а й безпосередньо вхідні сигнали. Така “каскадна” структура забезпечує додаткові шляхи передачі інформації, покращуючи збіжність навчання та зменшуючи похибку.

Натомість Elman Recurrent Neural Network у найпростішій конфігурації (з одним шаром) дала гірший результат через обмежену пам’ять і складність у навчанні коротких послідовностей. Проте зі збільшенням кількості шарів та нейронів вона продемонструвала значне покращення, оскільки рекурентні зв’язки дозволяють моделі враховувати часові залежності та контекст попередніх станів, що робить її більш гнучкою у моделюванні динамічних процесів.

Отже, чим складніша архітектура та глибша мережа, тим краща її здатність моделювати складні залежності. Водночас надмірне ускладнення може призводити до перенавчання, тому для кожного завдання необхідно знаходити баланс між складністю моделі та якістю узагальнення.