МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ЗВІТ

з лабораторної роботи №2

з навчальної дисципліни «Проектування та реалізація програмних систем з нейронними мережами»

Виконав:
студент групи ІП-15
Мєшков Андрій Ігорович

Перевірив:
Шимкович В.М.

Київ 2024

ЛАБОРАТОРНА РОБОТА №2

Тема: Реалізація базових архітектур нейронних мереж.

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

Варіант 6

Індивідуальні завдання

Завдання – Написати програму, що реалізує нейронні мережі для моделювання функції двох змінних. Функцію двох змінних, типу f(x+y) = x2+y2, обрати самостійно. Промоделювати на невеликому відрізку, скажімо від 0 до 10.

$$z = x \cdot \sin(y)$$

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

- Генеруємо дані

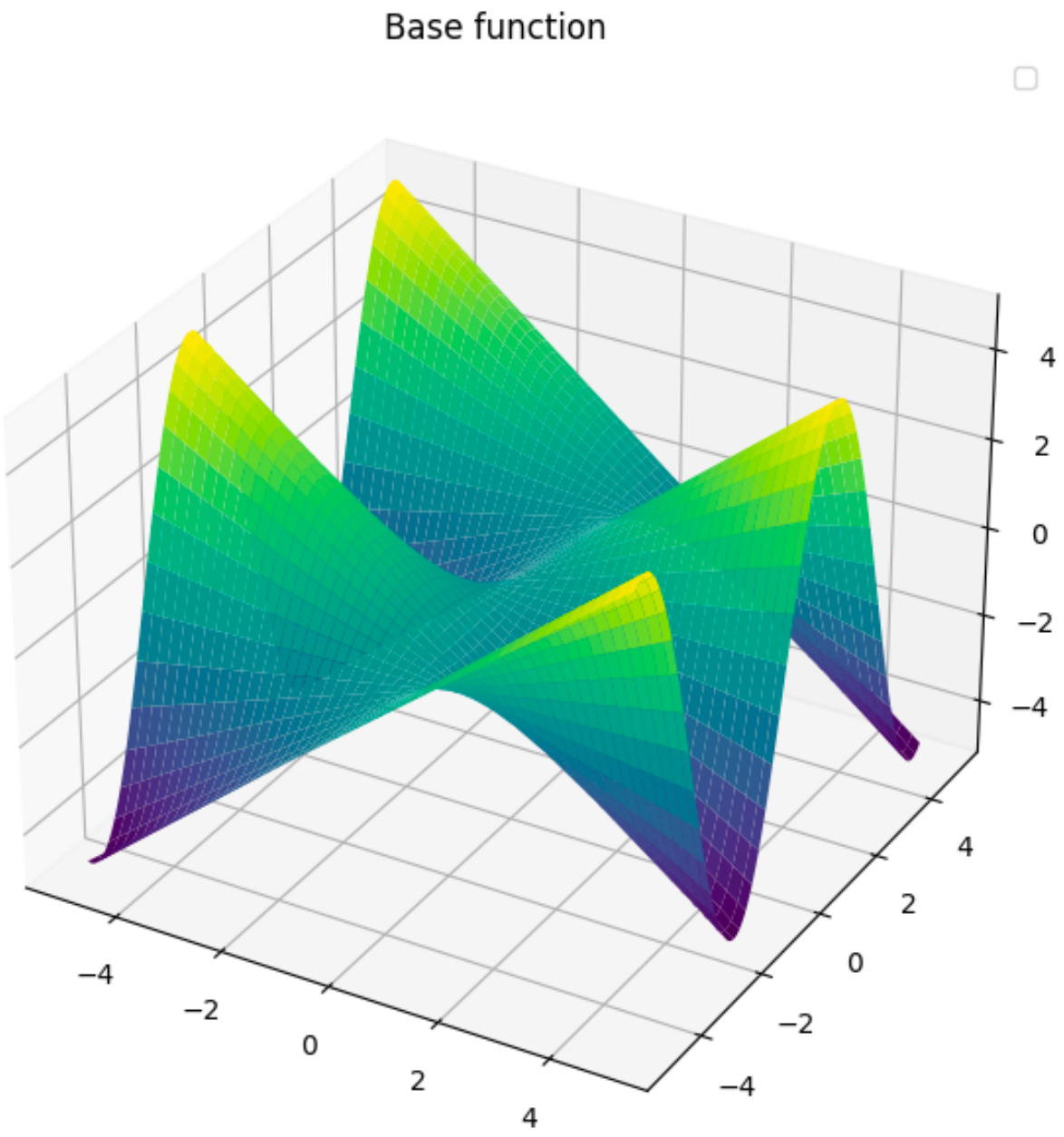
```
x_range = 5
y_range = x_range
density = 200

X = np.linspace(-x_range, x_range, density)
Y = np.linspace(-y_range, y_range, density)

X, Y = np.meshgrid(X, Y)

z_grid = X * np.sin(Y)

z = z_grid.flatten()
XY = np.array([X.flatten(), Y. flatten()]).T
```

- Створюємо функцію тренування та тестування моделей

```
def train_and_test(model, epochs = 150, batch_size = 50):
    model.summary()
    model.compile(optimizer='adam', loss='mse')
    history = model.fit(XY, z, epochs=epochs, batch_size=batch_size, verbose=1)

    z_pred = model.predict(XY)
    z_pred = np.reshape(z_pred, (density, density))

    fig = plt.figure(figsize=(16, 8))
    ax = fig.add_subplot(1, 2, 2, projection='3d')
    ax.plot_surface(X, Y, z_pred, cmap='viridis')
    plt.title(model.name)

    plt.figure(figsize=(8, 5))
    plt.plot(history.history['loss'], label='Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.title('Training Loss')
    plt.legend()
    plt.show()
```

- Візуалізуємо базову функцію

```
fig = plt.figure(figsize=(16, 8))
ax = fig.add_subplot(1, 2, 2, projection='3d')
ax.plot_surface(X, Y, z_grid, cmap='viridis')
plt.title("Base function")
plt.legend()
```



- Візуалізуємо кожну модель

```
train_and_test(feedforward(1, 10))
train_and_test(feedforward(1, 20))
train_and_test(cascadeforward(1, 20))
train_and_test(cascadeforward(2,10))
train_and_test(elman(1, 15))
train_and_test(elman(3, 5))
```

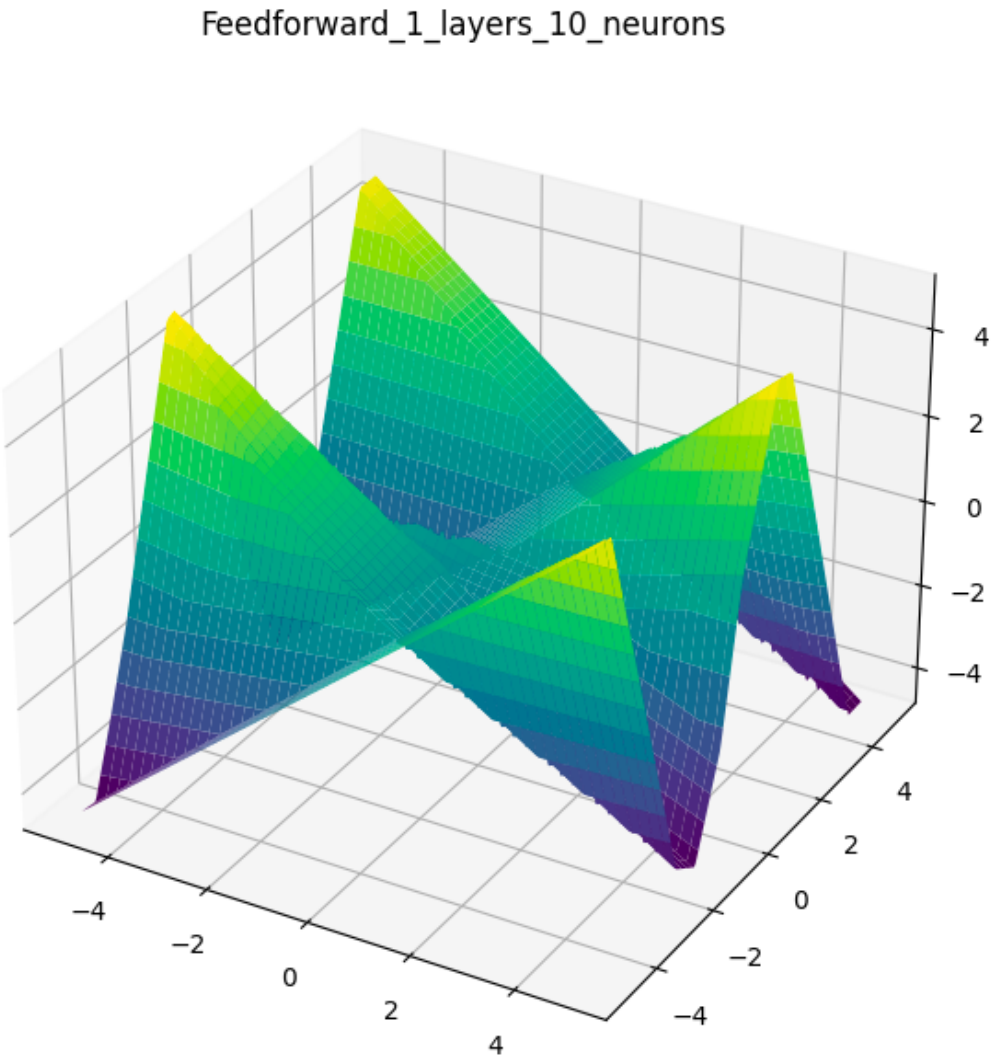
1. Тип мережі: feed forward backprop.

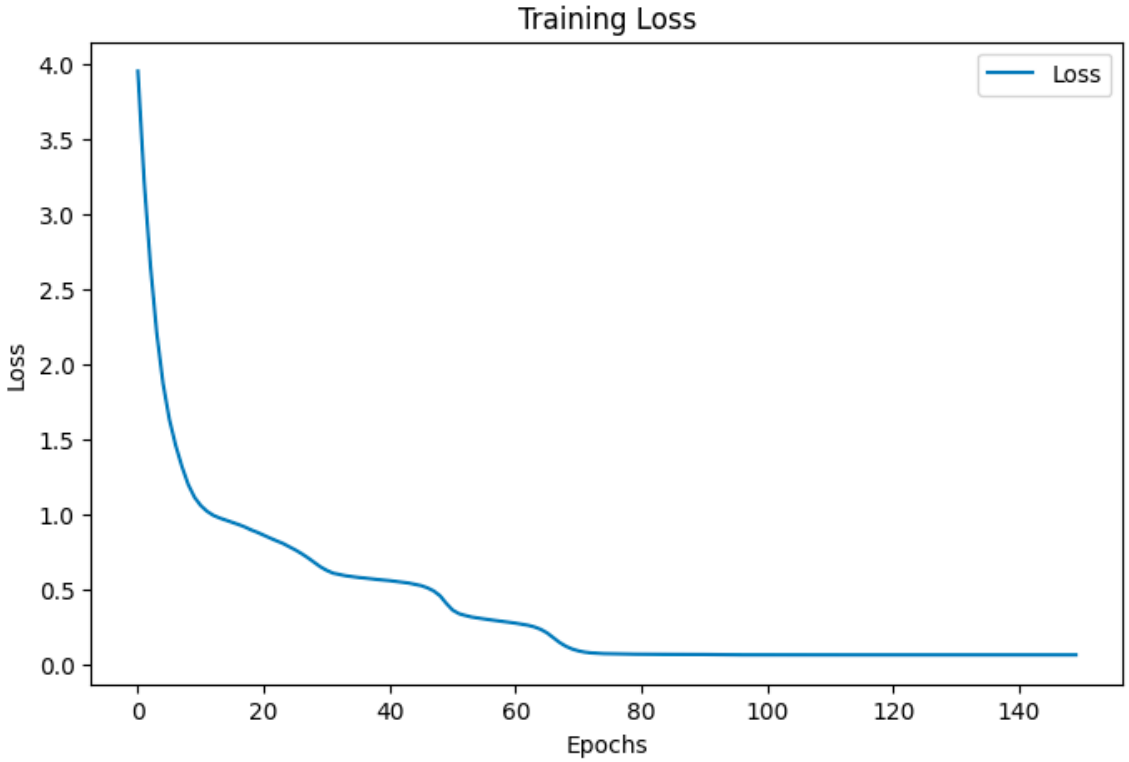
Основна відмінність моделі feed forward від інших архітектур полягає в її прямому перенапрямленні даних без зворотного зв'язку або циклічних зв'язків між шарами. У цій архітектурі кожен шар передає свої виходи безпосередньо наступному шару, без конкатенації з вхідними даними чи зворотного зв'язку. Це робить feedforward простим та ефективним для використання в багатьох задачах, зокрема в розпізнаванні зображень, натренованні та класифікації даних. Було використано Keras.Sequential

```python
def feedforward(layers, neurons, shape = (2,)):
    model = Sequential(name = f'Feedforward_{layers}_layers_{neurons}_neurons')

    model.add(Dense(neurons, activation = 'relu', input_shape=shape))

    for i in range(layers - 1):
        model.add(Dense(neurons, activation = 'relu'))

    model.add(Dense(1, name = 'output'))

    return model
```

а) 1 внутрішній шар з 10 нейронами;

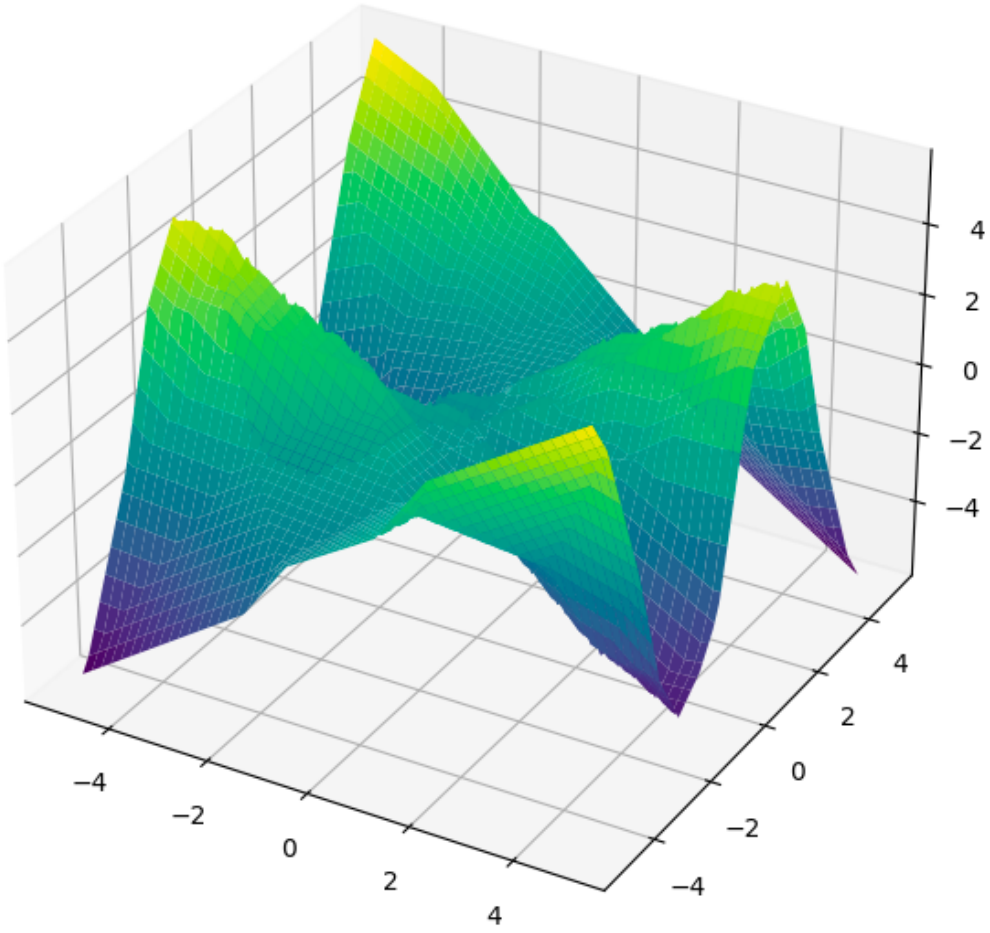
```
Model: "Feedforward_1_layers_10_neurons"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_5 (Dense)             (None, 10)                30

 output (Dense)              (None, 1)                 11

=================================================================
Total params: 41 (164.00 Byte)
Trainable params: 41 (164.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/150
800/800 [==============================] - 1s 1ms/step - loss: 3.9566
Epoch 2/150
800/800 [==============================] - 1s 1ms/step - loss: 3.2209
Epoch 3/150
800/800 [==============================] - 1s 899us/step - loss: 2.6580
Epoch 4/150
800/800 [==============================] - 1s 947us/step - loss: 2.2083
Epoch 5/150
800/800 [==============================] - 1s 995us/step - loss: 1.8729
Epoch 6/150
800/800 [==============================] - 1s 972us/step - loss: 1.6350
...
800/800 [==============================] - 1s 904us/step - loss: 0.0634
Epoch 150/150
800/800 [==============================] - 1s 918us/step - loss: 0.0639
1250/1250 [==============================] - 1s 783us/step
```

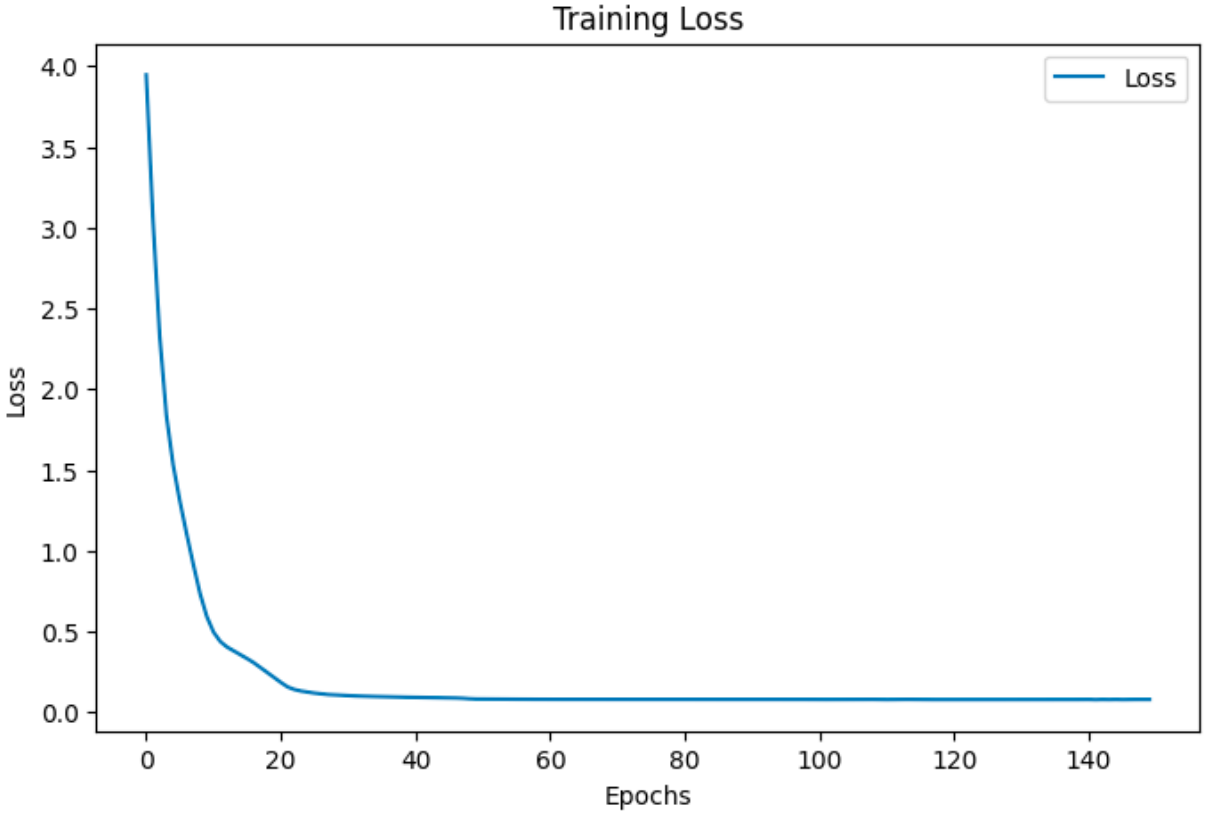

Feedforward_1_layers_10_neurons
4
2
0
−2
−4
−4
−2
0
2
4
4
2
0
−2
−4


Training Loss
Loss
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Loss
0
20
40
60
80
100
120
140
Epochs

b) 1 внутрішній шар з 20 нейронами;

```
Model: "Feedforward_1_layers_20_neurons"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 20)                60

 output (Dense)              (None, 1)                 21

=================================================================
Total params: 81 (324.00 Byte)
Trainable params: 81 (324.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/150
800/800 [==============================] - 1s 1ms/step - loss: 3.9459
Epoch 2/150
800/800 [==============================] - 1s 910us/step - loss: 3.0496
Epoch 3/150
800/800 [==============================] - 1s 980us/step - loss: 2.3295
Epoch 4/150
800/800 [==============================] - 1s 1ms/step - loss: 1.8366
Epoch 5/150
800/800 [==============================] - 1s 937us/step - loss: 1.5311
Epoch 6/150
800/800 [==============================] - 1s 918us/step - loss: 1.3063
...
800/800 [==============================] - 1s 1ms/step - loss: 0.0795
Epoch 150/150
800/800 [==============================] - 1s 927us/step - loss: 0.0799
1250/1250 [==============================] - 1s 835us/step
```


Feedforward_1_layers_20_neurons
4
2
0
−2
−4
−4
−2
0
2
4
4
2
0
−2
−4


Training Loss
Loss
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
20
40
60
80
100
120
140
Epochs

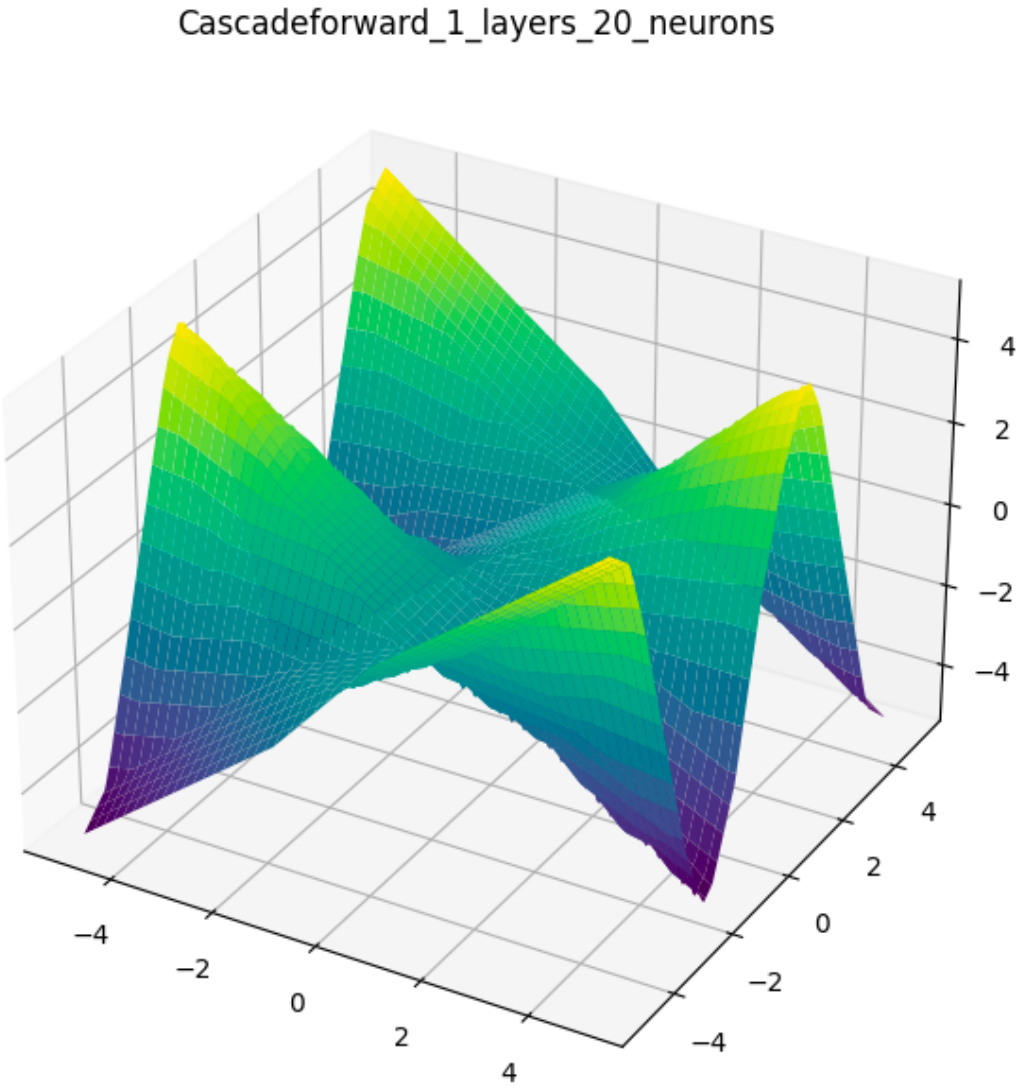
2. Тип мережі: cascade - forward backprop.

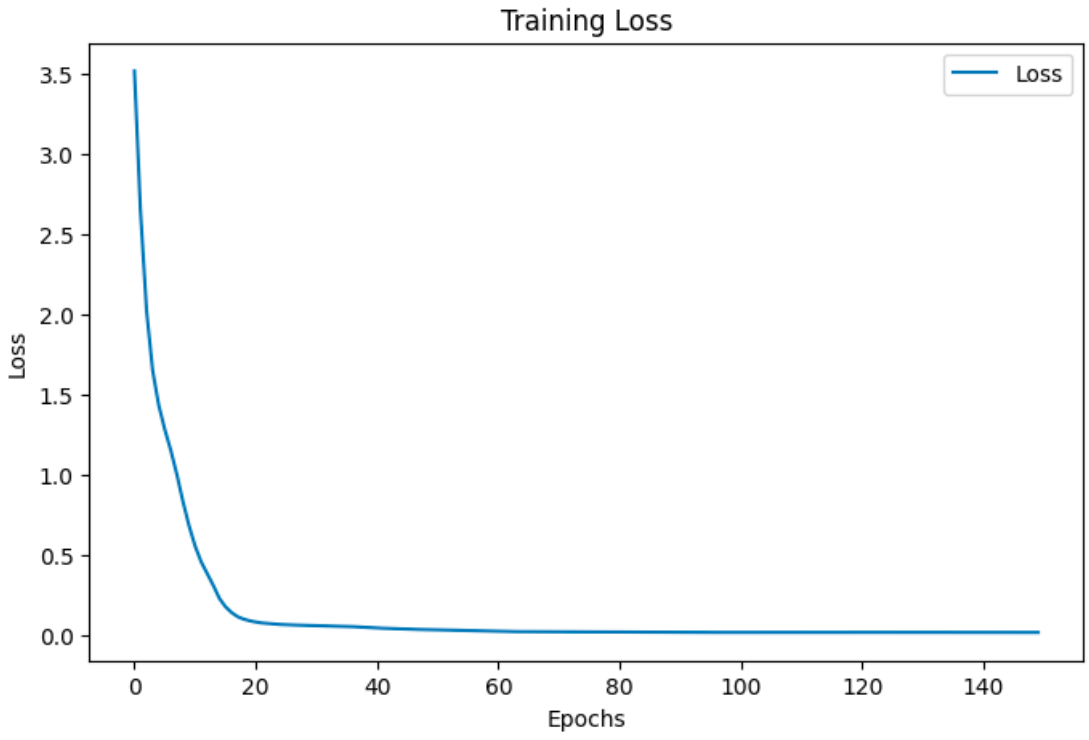
Головна відмінність каскадного перенаправлення від інших архітектур полягає у тому, що кожен наступний шар приймає на вхід не лише вихід поточного шару, але і вхідні дані, що конкатенуються з виходами попередніх шарів. Це дозволяє кожному шару в архітектурі мати доступ до інформації, що проходить через всю мережу, підвищуючи потенційну репрезентативність інформації на кожному рівні. Така архітектура дозволяє краще враховувати взаємозв'язки між вхідними даними та може покращити ефективність моделі, особливо в задачах зі складними взаємодіями між ознаками.

```
def cascadeforward(layers, neurons, shape = (2,)):
    input_layer = Input(shape = shape, name = 'input')

    current_layer = Dense(neurons, activation = 'relu', input_shape=shape)(input_layer)

    for i in range(layers-1):
        concatenated_layer = Concatenate()([input_layer, current_layer])
        current_layer = Dense(neurons, activation = 'relu',
input_shape=shape)(concatenated_layer)

    output_layer = Dense(1, name = 'output')(current_layer)

    model = Model(inputs = input_layer, outputs = output_layer, name =
f'Cascadeforward_{layers}_layers_{neurons}_neurons')

    return model
```

а) 1 внутрішній шар з 20 нейронами;

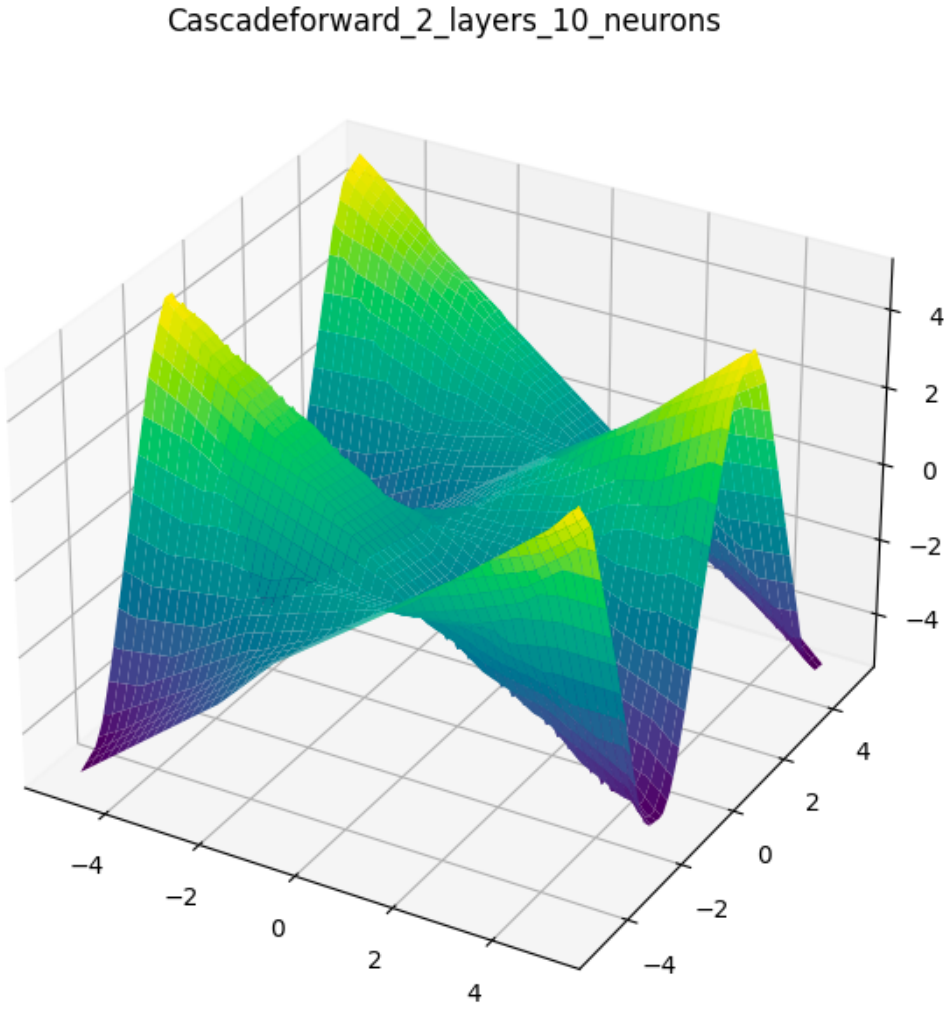
```
Model: "Cascadeforward_1_layers_20_neurons"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input (InputLayer)          [(None, 2)]               0

 dense_7 (Dense)             (None, 20)                60

 output (Dense)              (None, 1)                 21

=================================================================
Total params: 81 (324.00 Byte)
Trainable params: 81 (324.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/150
800/800 [==============================] - 1s 1ms/step - loss: 3.5199
Epoch 2/150
800/800 [==============================] - 1s 1ms/step - loss: 2.6446
Epoch 3/150
800/800 [==============================] - 1s 1ms/step - loss: 2.0265
Epoch 4/150
800/800 [==============================] - 1s 997us/step - loss: 1.6542
Epoch 5/150
800/800 [==============================] - 1s 1ms/step - loss: 1.4386
...
800/800 [==============================] - 1s 1ms/step - loss: 0.0215
Epoch 150/150
800/800 [==============================] - 1s 995us/step - loss: 0.0215
1250/1250 [==============================] - 1s 826us/step
```

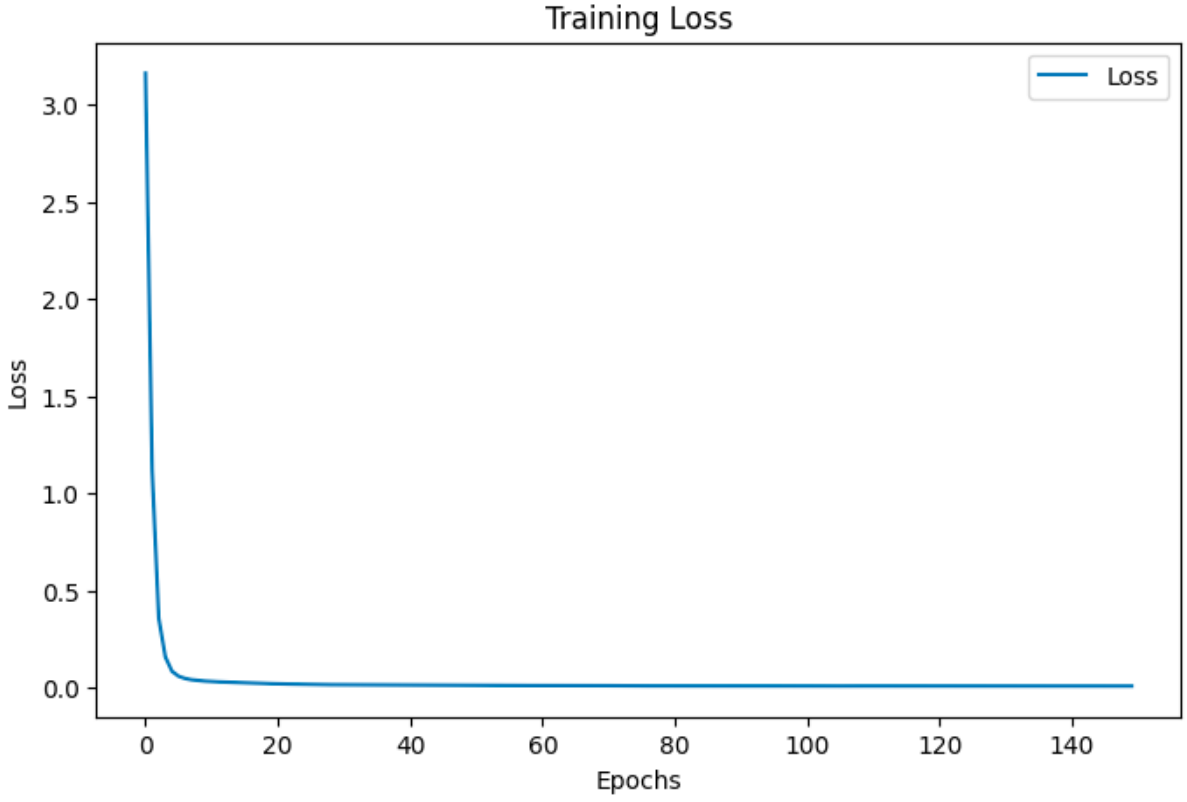

Cascadeforward_1_layers_20_neurons
4
2
0
−2
−4
−4
−2
0
2
4
4
2
0
−2
−4


Training Loss
Loss
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Loss
0
20
40
60
80
100
120
140
Epochs

b) 2 внутрішніх шари по 10 нейронів у кожному;

```
Model: "Cascadeforward_2_layers_10_neurons"
_________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
=================================================================================================
 input (InputLayer)          [(None, 2)]                  0         []

 dense_8 (Dense)             (None, 10)                   30        ['input[0][0]']

 concatenate_1 (Concatenate  (None, 12)                   0         ['input[0][0]',
 )                                                                   'dense_8[0][0]']

 dense_9 (Dense)             (None, 10)                   130       ['concatenate_1[0][0]']

 output (Dense)              (None, 1)                    11        ['dense_9[0][0]']

=================================================================================================
Total params: 171 (684.00 Byte)
Trainable params: 171 (684.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________________________________________
Epoch 1/150
800/800 [==============================] - 1s 1ms/step - loss: 3.1637
Epoch 2/150
800/800 [==============================] - 1s 1ms/step - loss: 1.1199
Epoch 3/150
...
800/800 [==============================] - 1s 1ms/step - loss: 0.0095
Epoch 150/150
800/800 [==============================] - 1s 1ms/step - loss: 0.0096
1250/1250 [==============================] - 1s 828us/step
```


Cascadeforward_2_layers_10_neurons
4
2
0
−2
−4
−4
−2
0
2
4
4
2
0
−2
−4


Training Loss
Loss
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Loss
0
20
40
60
80
100
120
140
Epochs

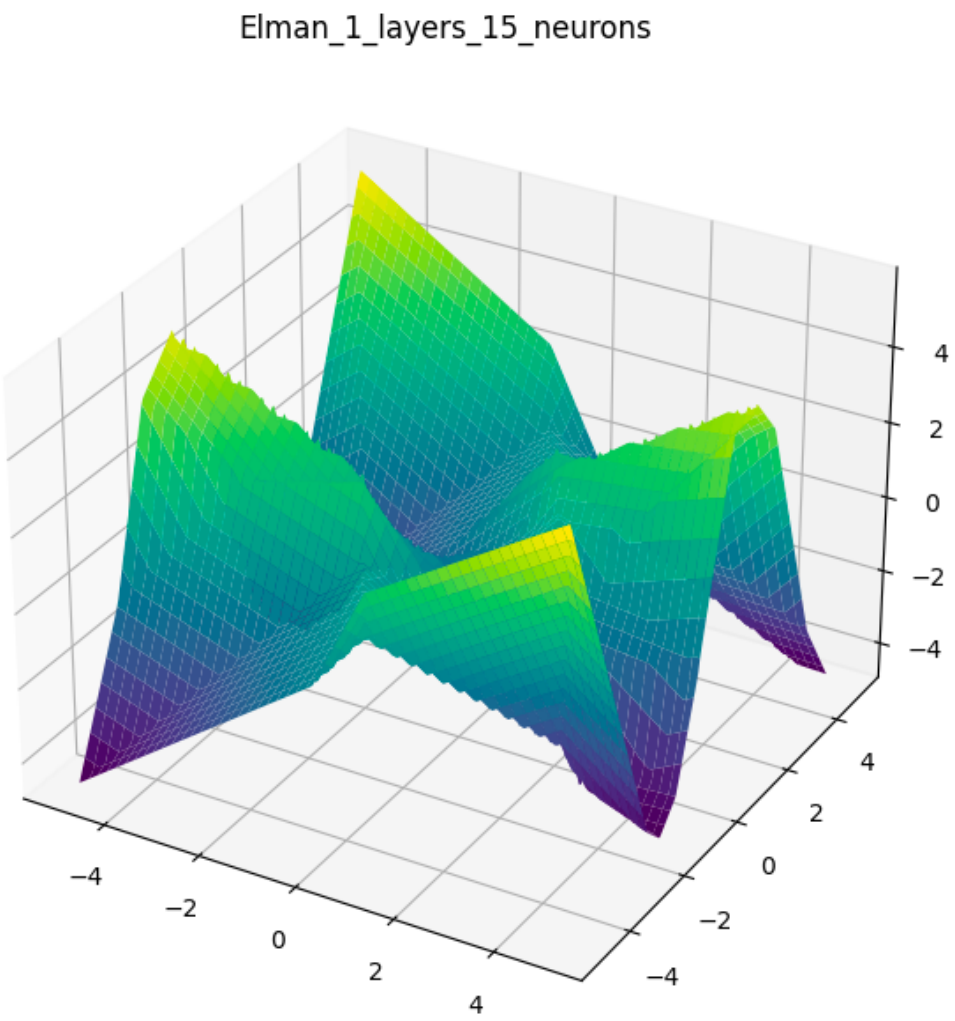
3.Тип мережі: elman backprop.

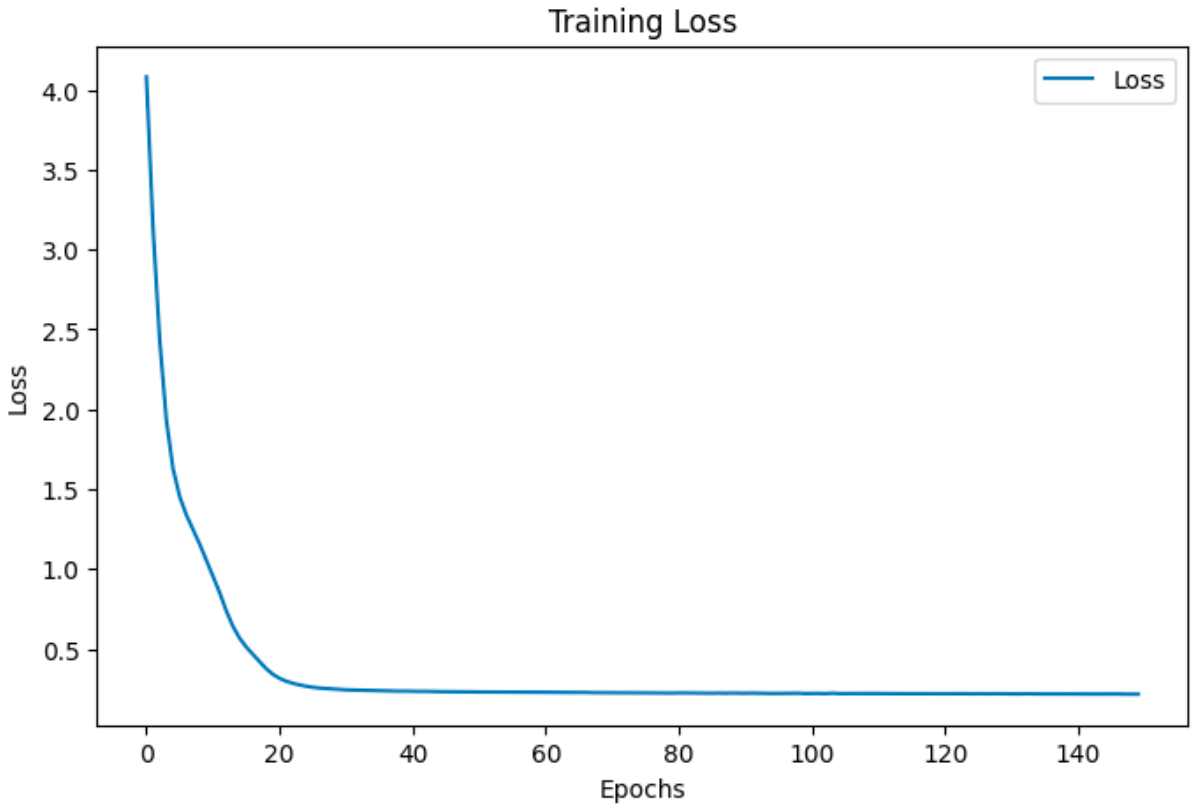
Архітектура Elman використовує рекурентні шари, особливо SimpleRNN, для збереження стану попередніх часових кроків. Під час навчання модель отримує як вхід дані та інформацію з попередніх часових кроків, що дозволяє враховувати контекст та залежності в часі. Основна відмінність Elman від інших моделей, як feedforward чи каскадний, полягає в здатності моделі до роботи з послідовними даними, наприклад, в часових рядках чи текстах, враховуючи їхню структуру та послідовність.

```
def elman(layers, neurons, shape = (2,)):
    model = Sequential(name = f'Elman_{layers}_layers_{neurons}_neurons')

    model.add(Reshape((1, shape[0]), input_shape = shape, name = 'input_reshape'))
    model.add(SimpleRNN(neurons, return_sequences=True, activation = 'relu',
input_shape=shape))

    for i in range(layers - 1):
        model.add(SimpleRNN(neurons, return_sequences=True, activation = 'relu'))

    model.add(Reshape((neurons, ), name = 'output_reshape'))
    model.add(Dense(1, name = 'output'))

    return model
```

а) 1 внутрішній шар з 15 нейронами;

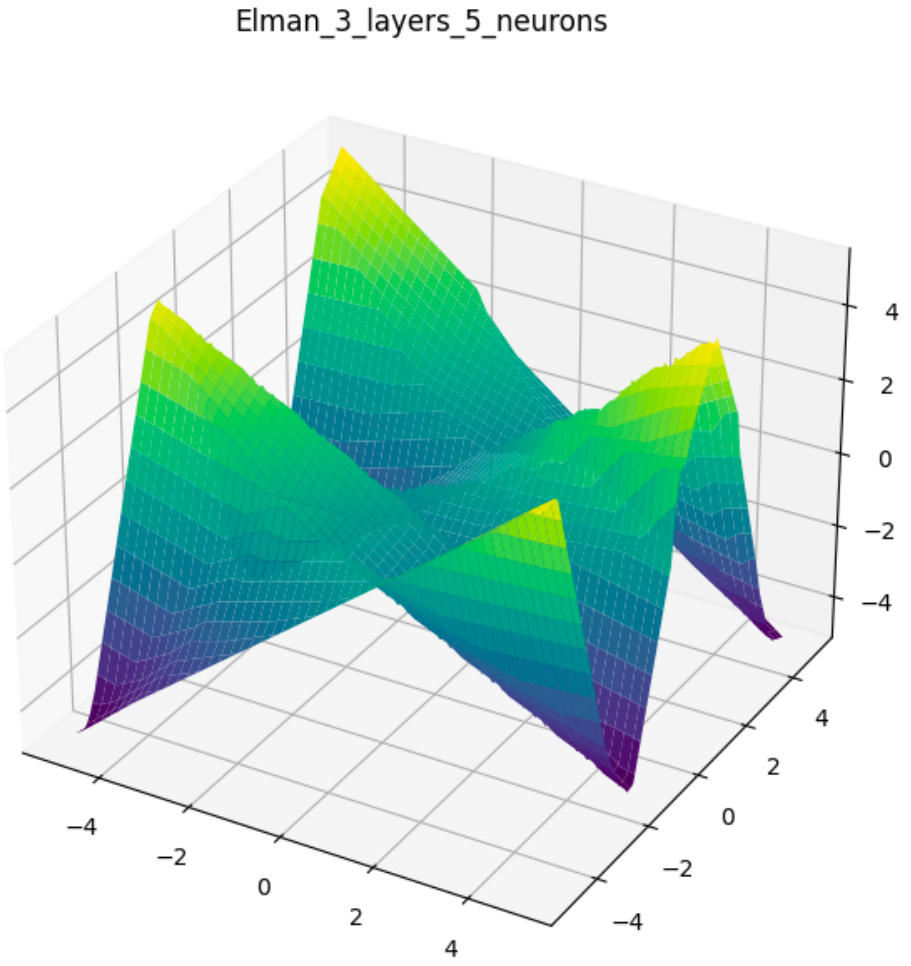
```
Model: "Elman_1_layers_15_neurons"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_reshape (Reshape)     (None, 1, 2)              0

 simple_rnn_4 (SimpleRNN)    (None, 1, 15)             270

 output_reshape (Reshape)    (None, 15)                0

 output (Dense)              (None, 1)                 16

=================================================================
Total params: 286 (1.12 KB)
Trainable params: 286 (1.12 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/150
800/800 [==============================] - 2s 1ms/step - loss: 4.0812
Epoch 2/150
800/800 [==============================] - 1s 1ms/step - loss: 3.1300
Epoch 3/150
800/800 [==============================] - 1s 1ms/step - loss: 2.4299
Epoch 4/150
800/800 [==============================] - 1s 1ms/step - loss: 1.9304
...
800/800 [==============================] - 1s 1ms/step - loss: 0.2170
Epoch 150/150
800/800 [==============================] - 1s 1ms/step - loss: 0.2163
1250/1250 [==============================] - 1s 914us/step
```

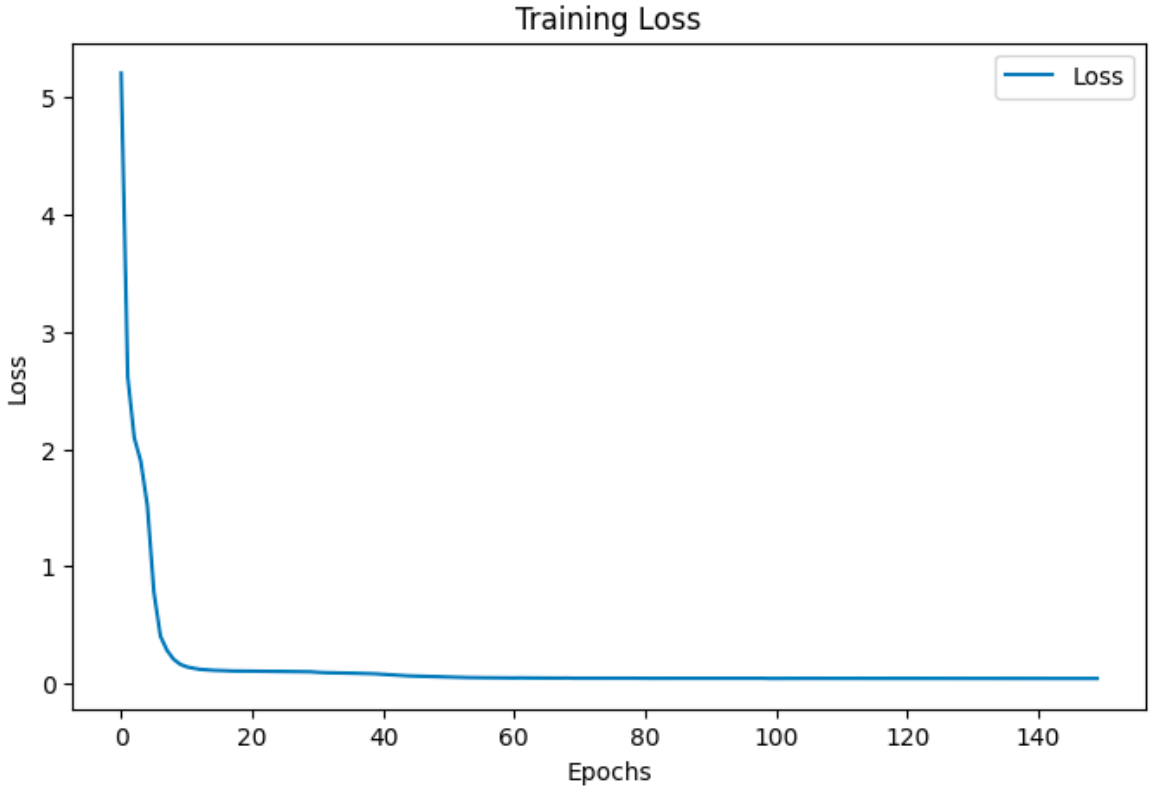

Elman_1_layers_15_neurons
4
2
0
−2
−4
−4
−2
0
2
4
4
2
0
−2
−4


Training Loss
Loss
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
Loss
0
20
40
60
80
100
120
140
Epochs

b) 3 внутрішніх шари по 5 нейронів у кожному;

```
Model: "Elman_3_layers_5_neurons"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_reshape (Reshape)     (None, 1, 2)              0

 simple_rnn_5 (SimpleRNN)    (None, 1, 5)              40

 simple_rnn_6 (SimpleRNN)    (None, 1, 5)              55

 simple_rnn_7 (SimpleRNN)    (None, 1, 5)              55

 output_reshape (Reshape)    (None, 5)                 0

 output (Dense)              (None, 1)                 6

=================================================================
Total params: 156 (624.00 Byte)
Trainable params: 156 (624.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/150
800/800 [==============================] - 4s 2ms/step - loss: 5.2081
Epoch 2/150
800/800 [==============================] - 2s 2ms/step - loss: 2.6157
...
800/800 [==============================] - 2s 2ms/step - loss: 0.0418
Epoch 150/150
800/800 [==============================] - 2s 2ms/step - loss: 0.0418
1250/1250 [==============================] - 2s 1ms/step
```


Elman_3_layers_5_neurons
4
2
0
−2
−4
−4
−2
0
2
4
4
2
0
−2
−4


Training Loss
Loss
Loss
5
4
3
2
1
0
0
20
40
60
80
100
120
140
Epochs

Висновки:

У цьому дослідженні я вивчав проектування нейронних мереж для апроксимації функцій з двома змінними. Я використав три різні типи нейронних мереж з двома параметрами для кожного. В результаті виявилося, що модель Cascade Forward показала кращі результати порівняно з моделями Feed Forward та Elman. Додатково, в усіх моделях спостерігався високий Bias, який можна виправити шляхом збільшення кількості шарів та нейронів.