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**

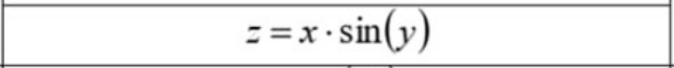
**Тема:** Реалізація базових архітектур нейронних мереж.

**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.



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

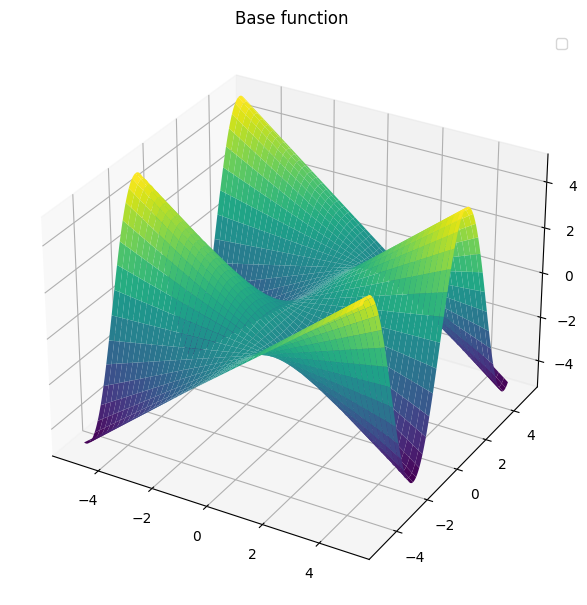
fig = plt.figure(figsize=(16, 8))

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

def feedforward(layers, neurons, shape = (2,)):

model = Sequential(name = f'Feedforward\_{layers}\_layers\_{neurons}\_neurons')

model.add(Dense(neurons, activation = 'relu', input\_shape=shape))

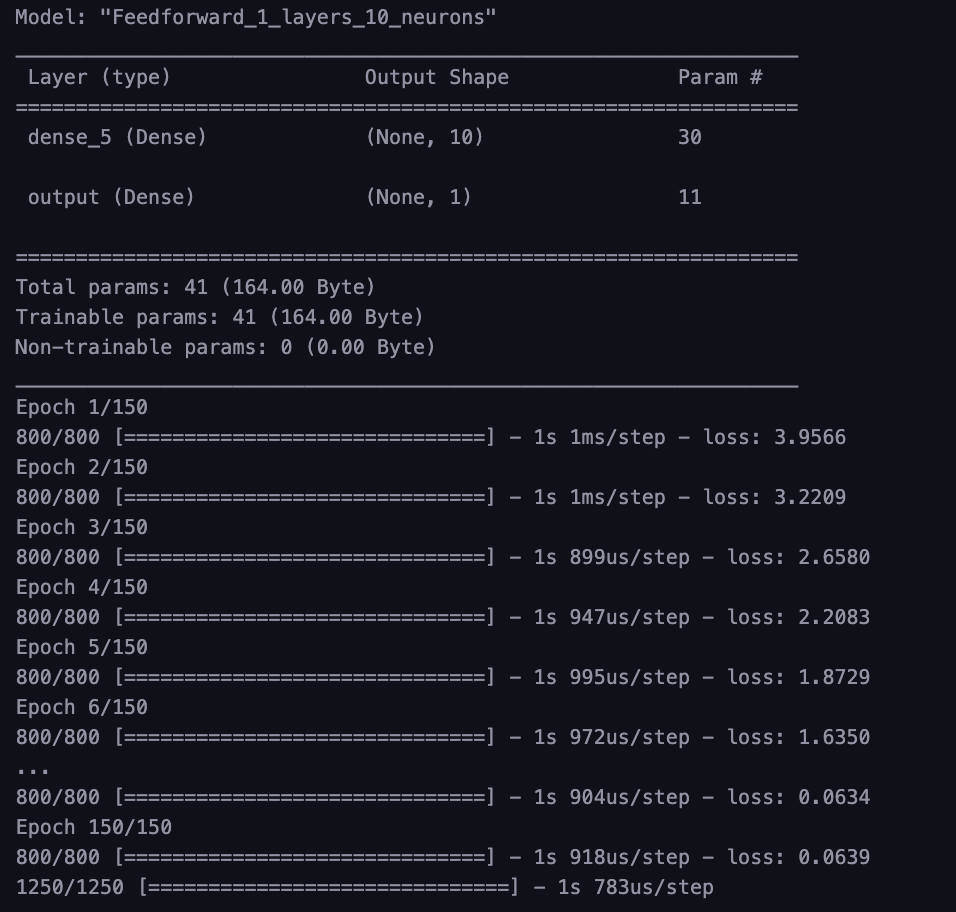
*for* i *in* range(layers - 1):

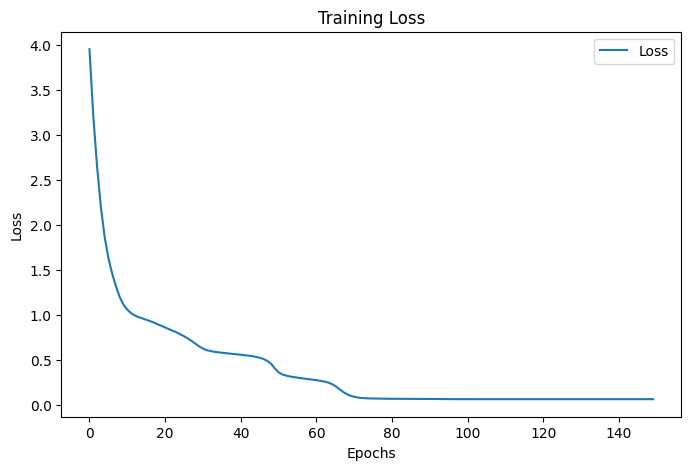
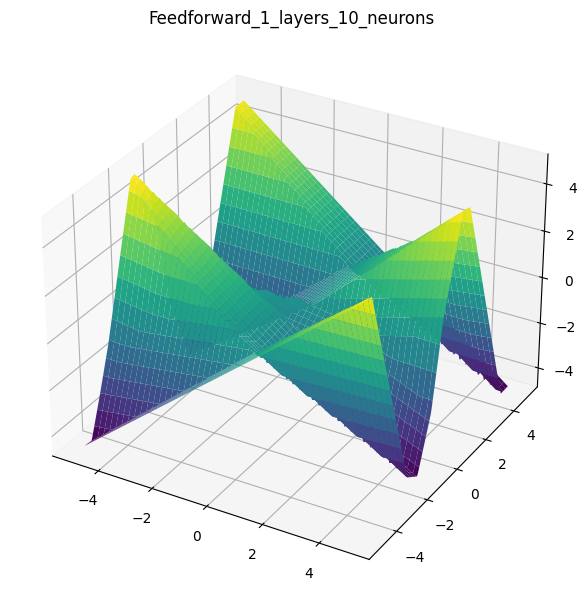
model.add(Dense(neurons, activation = 'relu'))

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

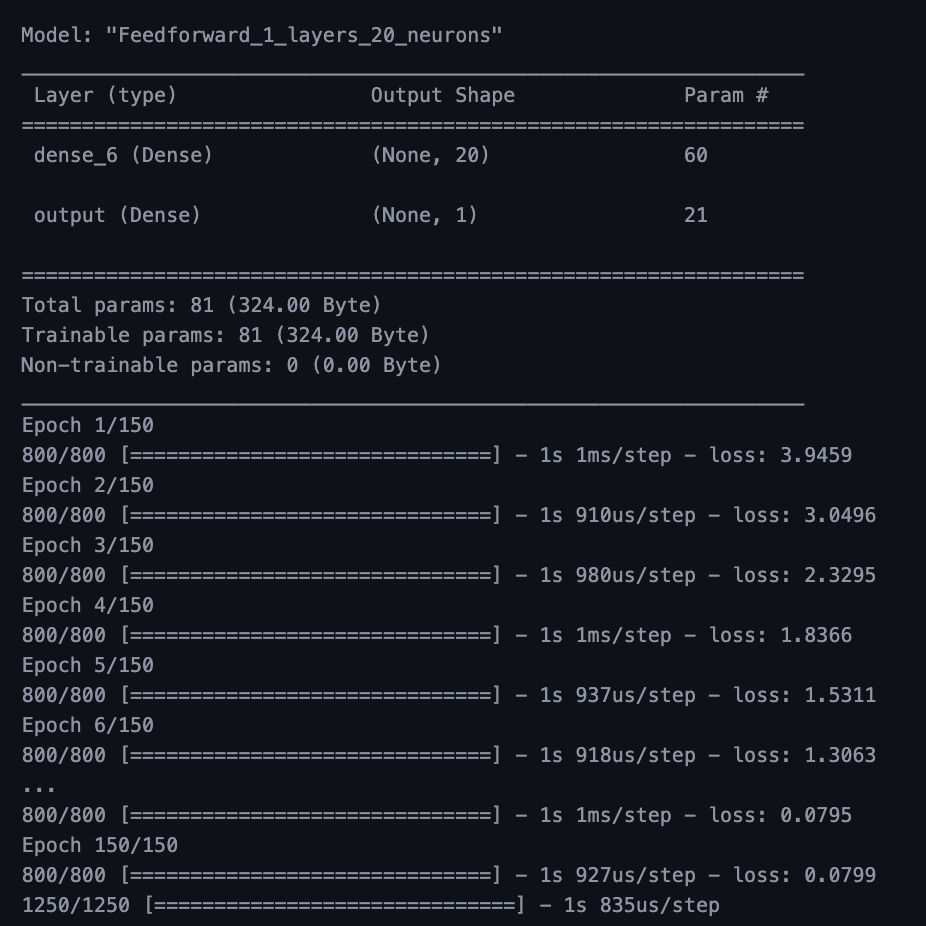
*return* model

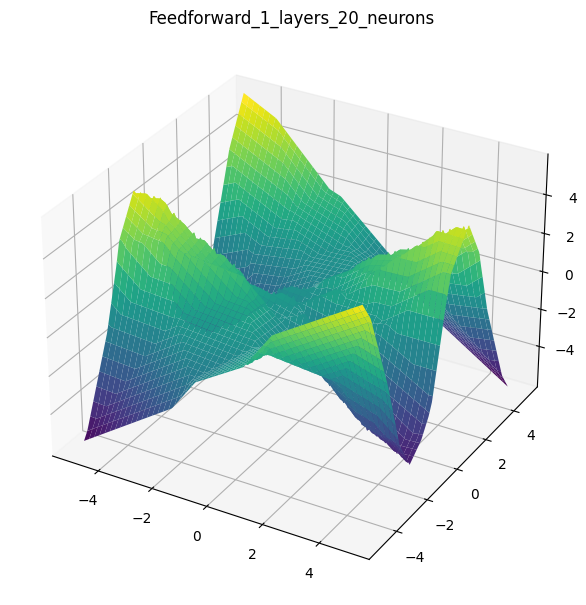
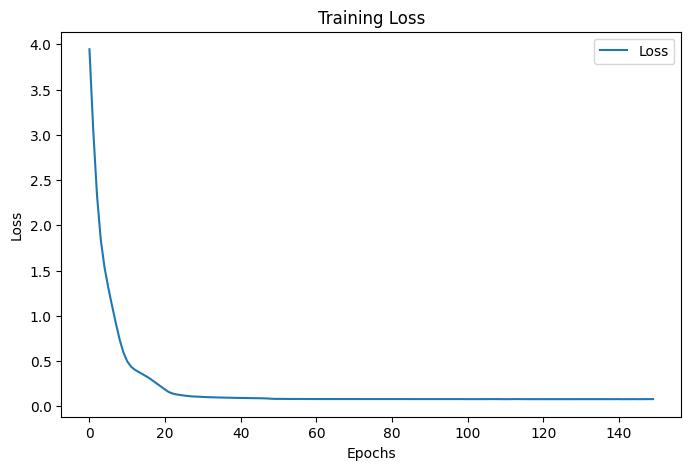
a) 1 внутрішній шар з 10 нейронами;





b) 1 внутрішній шар з 20 нейронами;



1. 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])

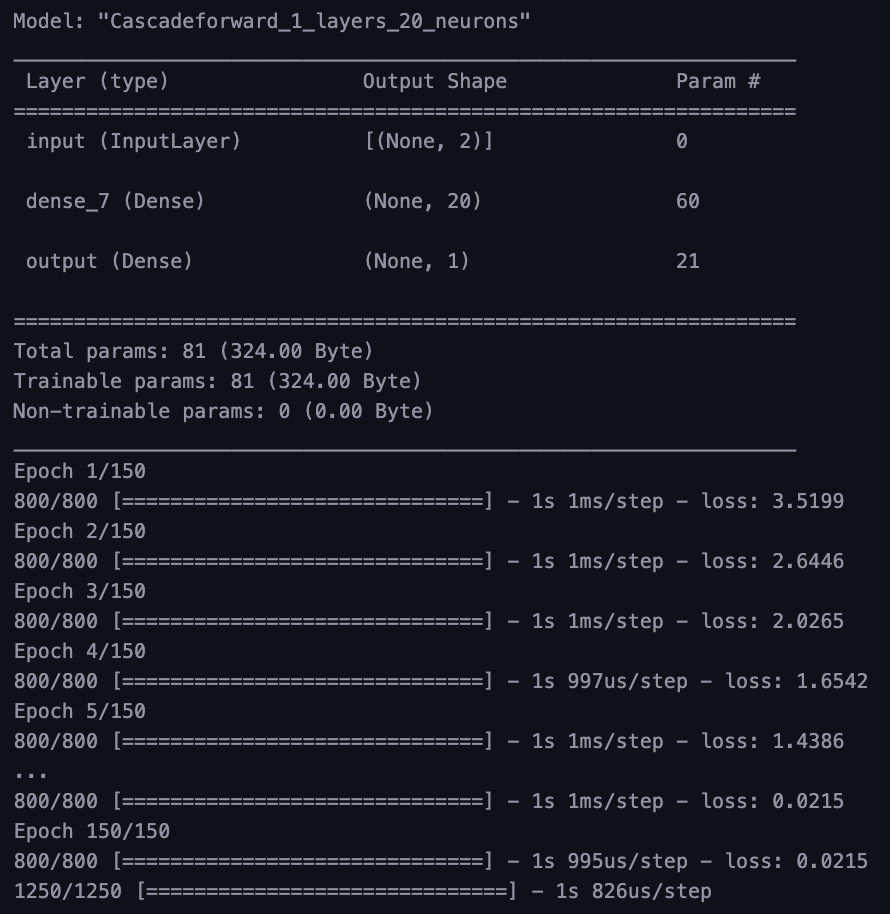
current\_layer = Dense(neurons, activation = 'relu', input\_shape=shape)(concatenated\_layer)

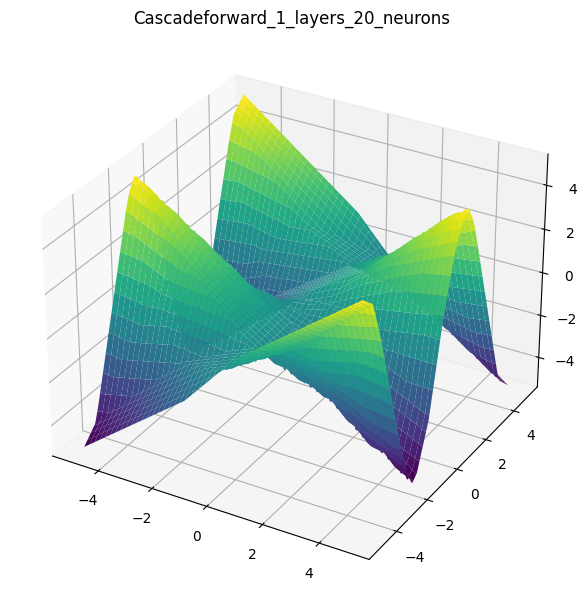
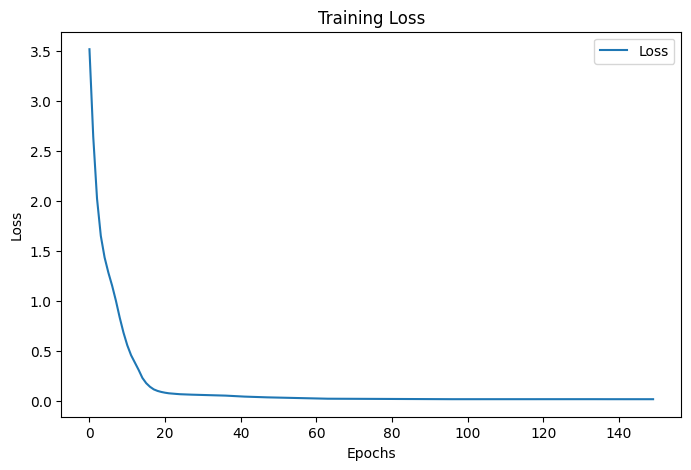
output\_layer = Dense(1, name = 'output')(current\_layer)

model = Model(inputs = input\_layer, outputs = output\_layer, name = f'Cascadeforward\_{layers}\_layers\_{neurons}\_neurons')

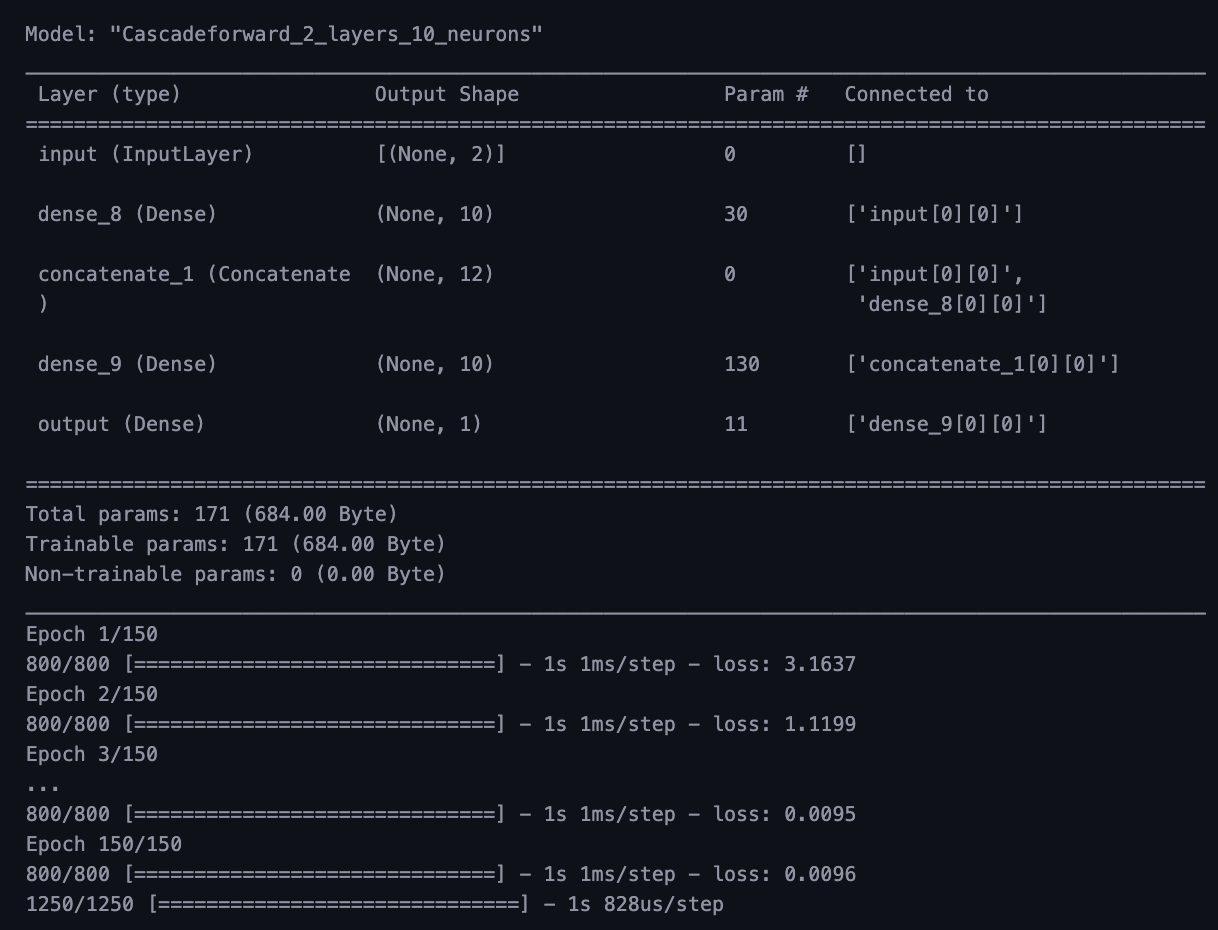
*return* model

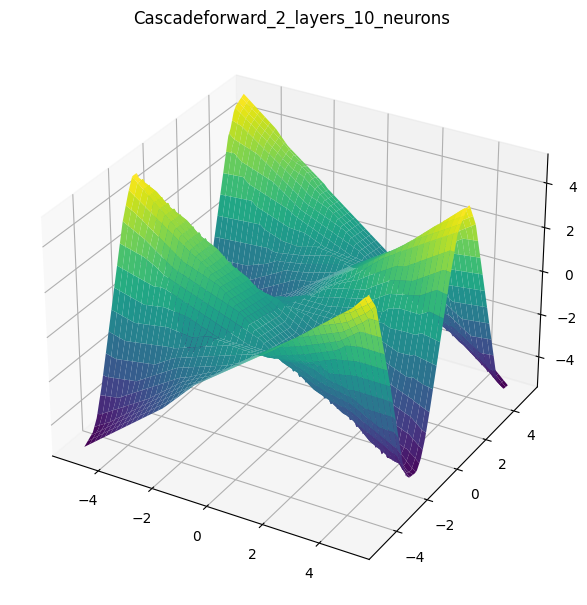
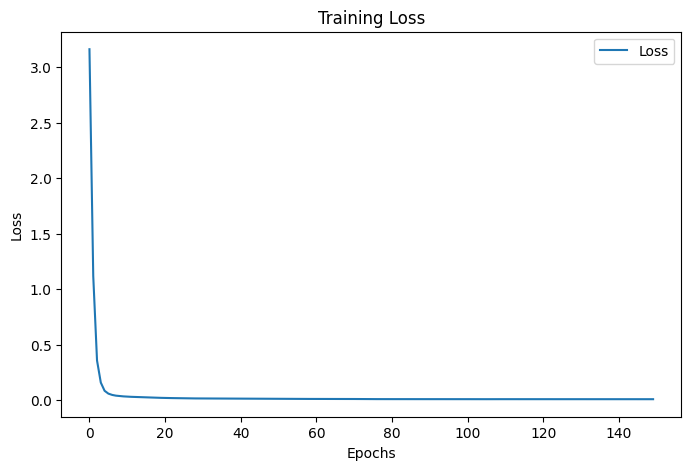
1. 1 внутрішній шар з 20 нейронами;



1. 2 внутрішніх шари по 10 нейронів у кожному;



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):

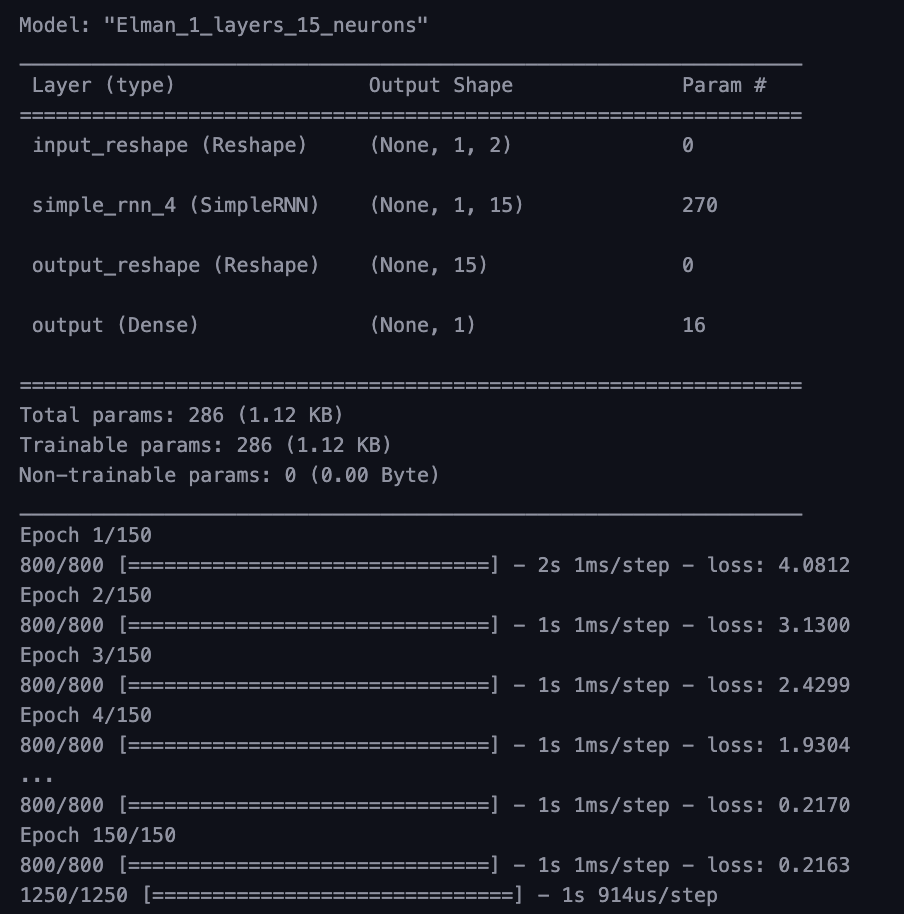
model.add(SimpleRNN(neurons, return\_sequences=True, activation = 'relu'))

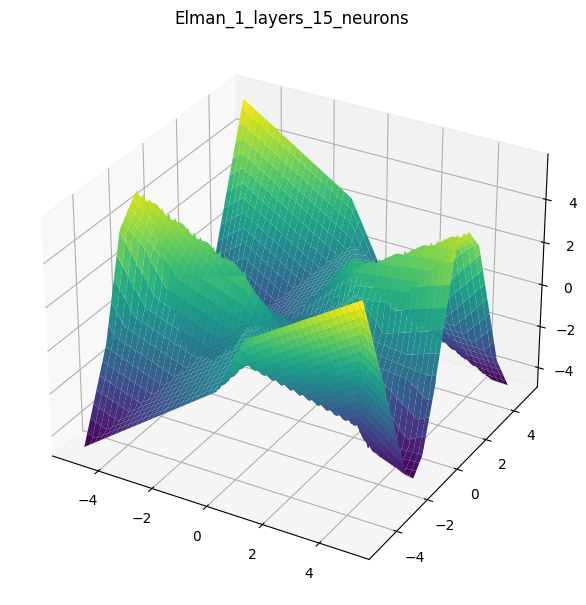
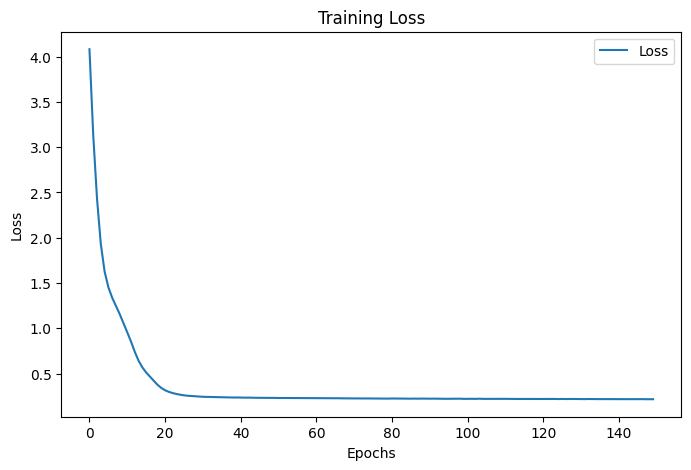
model.add(Reshape((neurons, ), name = 'output\_reshape'))

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

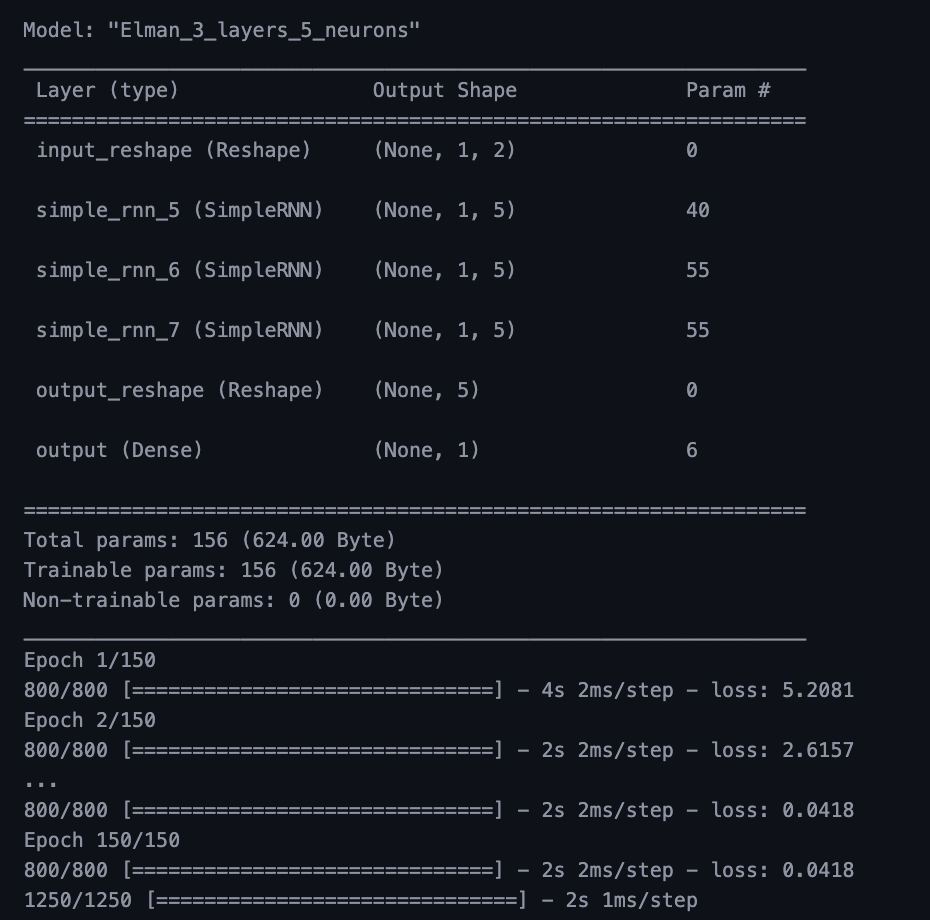
*return* model

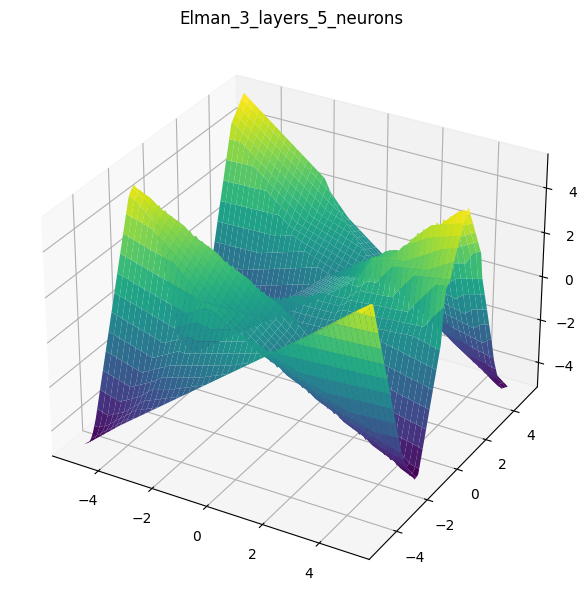
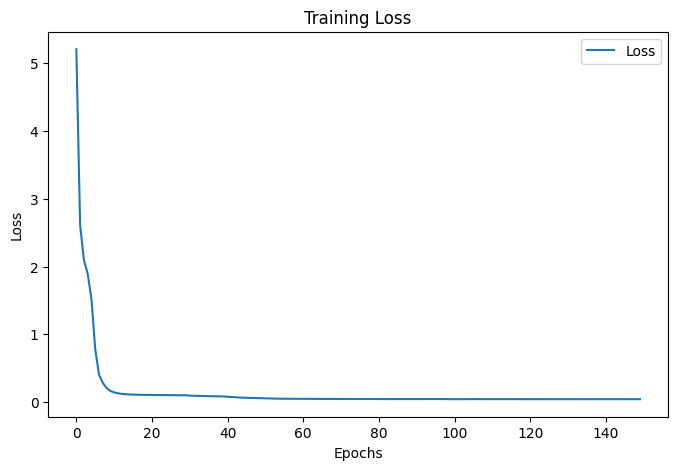
1. 1 внутрішній шар з 15 нейронами;



b) 3 внутрішніх шари по 5 нейронів у кожному;



**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.