НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”
КАФЕДРА ІНФОРМАТИКИ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Проектування та дослідження систем із штучним інтелектом

ЗВІТ

до лабораторної роботи №2

Виконав ІП-41мн Пашковський Євгеній Сергійович

Київ 2024

Завдання

Написати програму, що реалізує нейронні мережі для моделювання функції двох змінних. Функцію двох змінних обрати самостійно. Промоделювати на невеликому відрізку, скажімо від 0 до 10.

Хід роботи

У якості функції для моделювання було обрано:

f(x, y) = x * x + x * y * np.sin(y) - 2 * np.sqrt(y)

Код програми

```
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
```

```
def func(v):
  x = v[..., 0]
  y = v[..., 1]
  return x * x + x * y * np.sin(y) - 2 * np.sqrt(y)
```

```
x = np.linspace(0, 20, 1000)
y = np.linspace(0, 20, 1000)
X, Y = np.meshgrid(x, y)

V = np.stack((X, Y), axis=-1)
W = func(V)
```

```
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(X, Y, W, cmap='viridis')

# Labels
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('f(X, Y)')
ax.set_title('3D Plot of f(x, y)')

plt.show()
```

```
    v = np.random.random(size=(1000, 2)) * 20
    w = func(v)
```

```
    fig = plt.figure(figsize=(8, 6))
    ax = fig.add_subplot(111, projection='3d')
    ax.scatter(v[..., 0], v[..., 1], w, c=w, cmap='viridis', s=10)

    # Labels
    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('f(X, Y)')
    ax.set_title('3D Plot of f(x, y)')

    plt.show()
```

```
    def getFeedForwardModel(hiddenLayers):
      model = keras.Sequential()
      model.add(keras.Input(shape=(2,)))

      for units in hiddenLayers:
        model.add(keras.layers.Dense(units, activation='relu'))

      model.add(keras.layers.Dense(1, activation="linear"))
      model.compile(optimizer="adam", loss="mse", metrics=["mae"])
      return model

    def getCascadeForwardModel(hiddenLayers):
      inputLayer = keras.Input(shape=(2,))
      prevOutputLayer = inputLayer
      for units in hiddenLayers:
                                outputLayer       =       keras.layers.Dense(units,
activation='relu')(prevOutputLayer)
        prevOutputLayer = keras.layers.Concatenate()([inputLayer, outputLayer])
      outputLayer = keras.layers.Dense(1)(prevOutputLayer)
      model = keras.Model(inputLayer, outputLayer)
      model.compile(optimizer="adam", loss="mse", metrics=["mae"])
      return model

    def getElmanModel(hiddenLayers):
      inputLayer = keras.Input(shape=(2,))
```

```
                    hiddenState       =       keras.layers.Dense(hiddenLayers[0],
activation="relu")(inputLayer)
        for units in hiddenLayers[1:]:
          hiddenState = keras.layers.Dense(units, activation="relu")(hiddenState)
        outputLayer = keras.layers.Dense(1)(hiddenState)
        model = keras.Model(inputLayer, outputLayer)
        model.compile(optimizer="adam", loss="mse", metrics=["mae"])
        return model

      def getMRE(model, x, y, epsilon=1e-10):
        predictions = model.predict(x)
        relativeErrors = np.abs((y - predictions) / (y + epsilon))
        mre = np.mean(relativeErrors) * 100
        return mre
```

```
      feedForwardModelA = getFeedForwardModel([10])
      feedForwardModelB = getFeedForwardModel([20])

      cascadeForwardModelA = getCascadeForwardModel([20])
      cascadeForwardModelB = getCascadeForwardModel([10, 10])

      elmanModelA = getElmanModel([15])
      elmanModelB = getElmanModel([5, 5, 5])


      names = ["Feed Forward A", "Feed Forward B", "Cascade Forward A", "Cascade
Forward B", "Elman A", "Elman B"]
      models   =   [feedForwardModelA,   feedForwardModelB,   cascadeForwardModelA,
cascadeForwardModelB, elmanModelA, elmanModelB]

      histories = []
      evaluations = []

      V = V.reshape(1000000, 2)

      for model in models:
        histories.append(model.fit(v, w, epochs=1000, verbose=1))
        evaluations.append(getMRE(model, V, W.reshape(1000000, 1)))
```

```
      print(evaluations)
```

```
for i in range(len(histories)):
  plt.plot(histories[i].history['loss'], label='Train Loss')
  plt.xlabel('Епохи')
  plt.ylabel('Функція втрат')
  plt.title(names[i])
  plt.legend()
  plt.grid()
  plt.show()
  print(f'{names[i]} MRE: {evaluations[i]}')
```

Отриманий результат

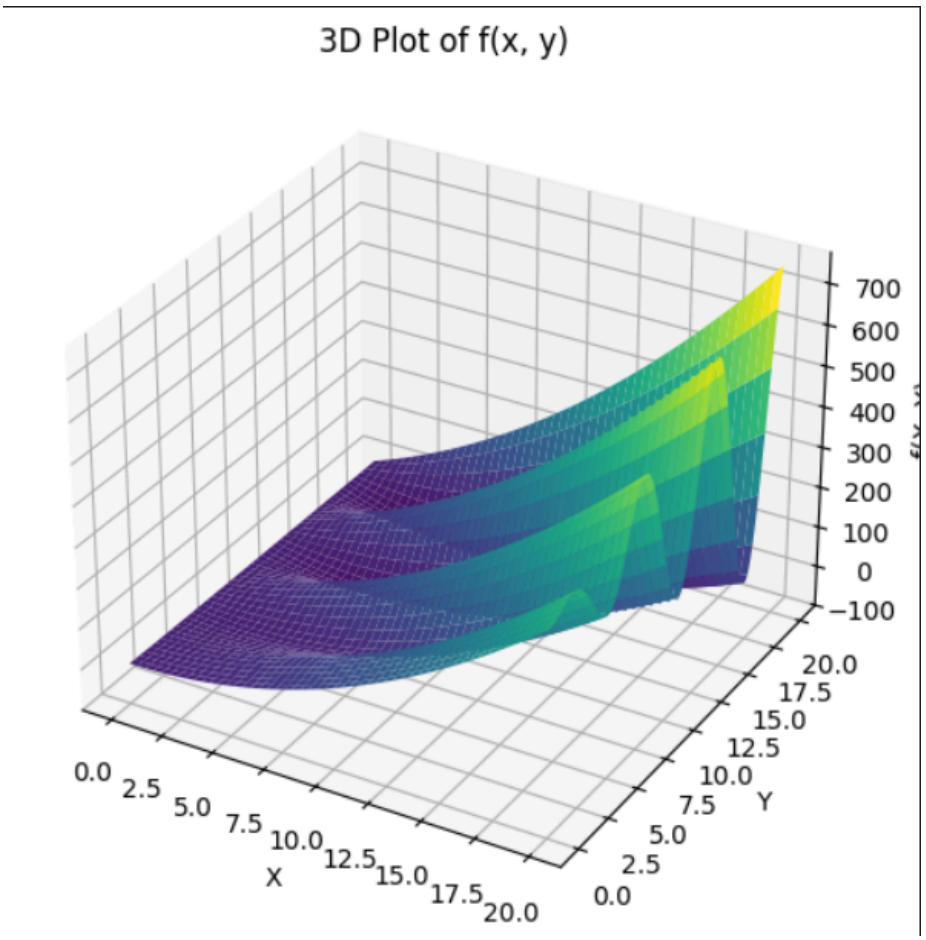

Рисунок 1 – Функція двох зміних, що моделюється

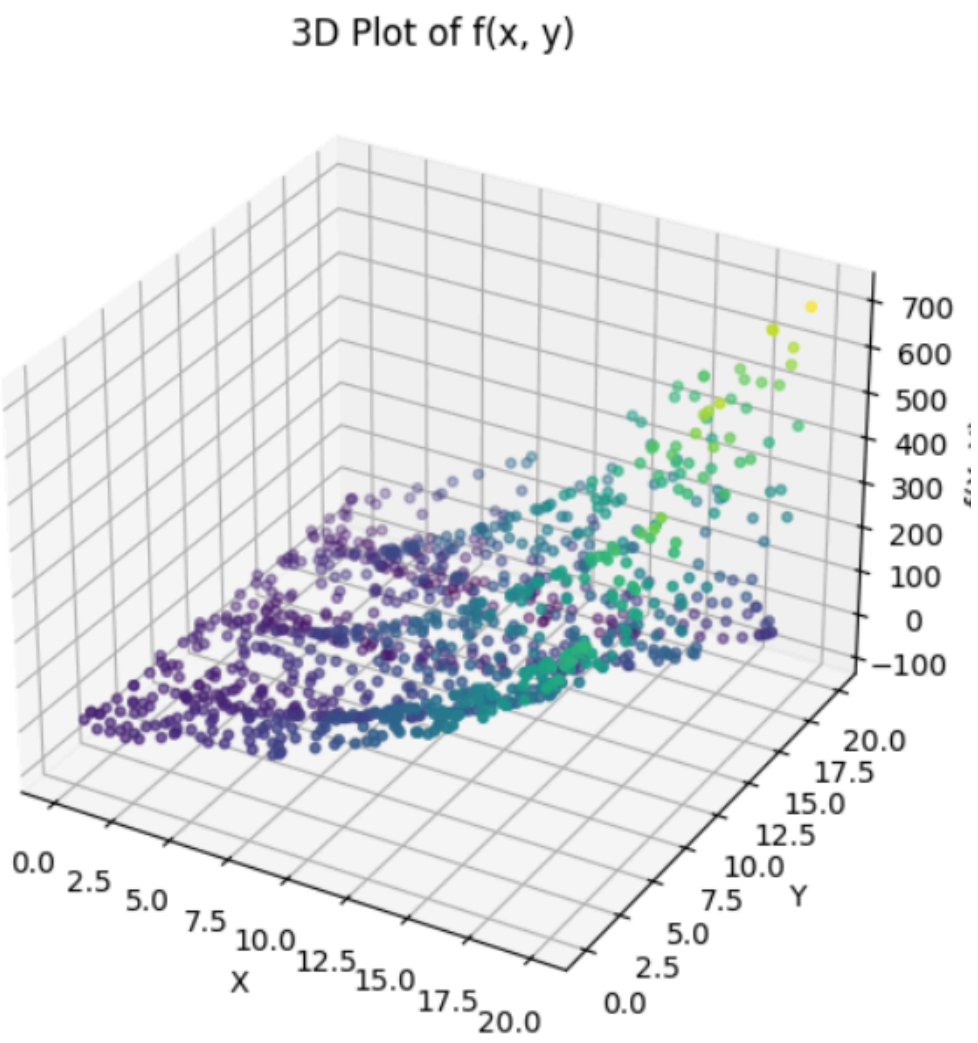

Рисунок 2 – Візуалізація тренувального датасету

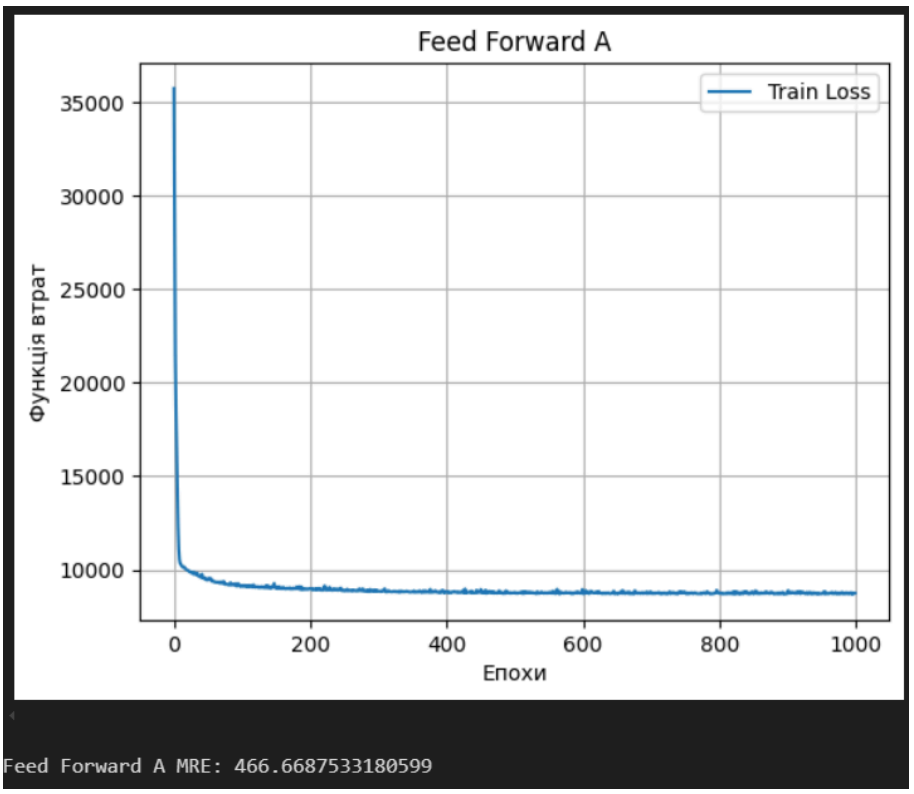

Рисунок 3 – Результат моделі Feed Forward A (1 внутрішній шар з 10 нейронами)

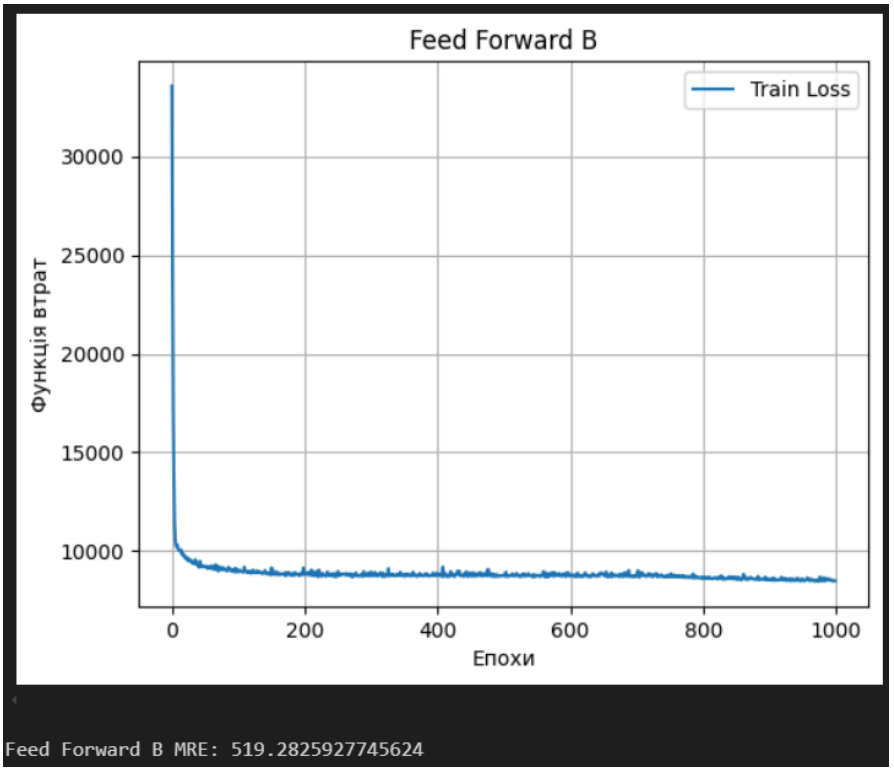

Рисунок 4 – Результат моделі Feed Forward B (1 внутрішній шар з 20 нейронами)

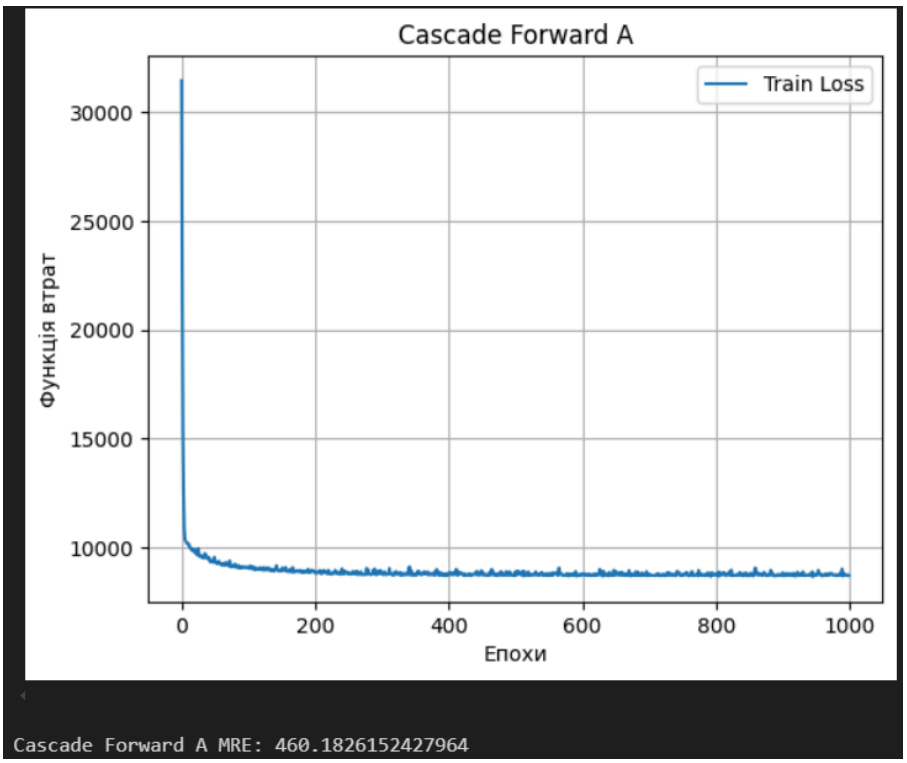

Рисунок 5 – Результат моделі Cascade Forward A (1 внутрішній шар з 20 нейронами)

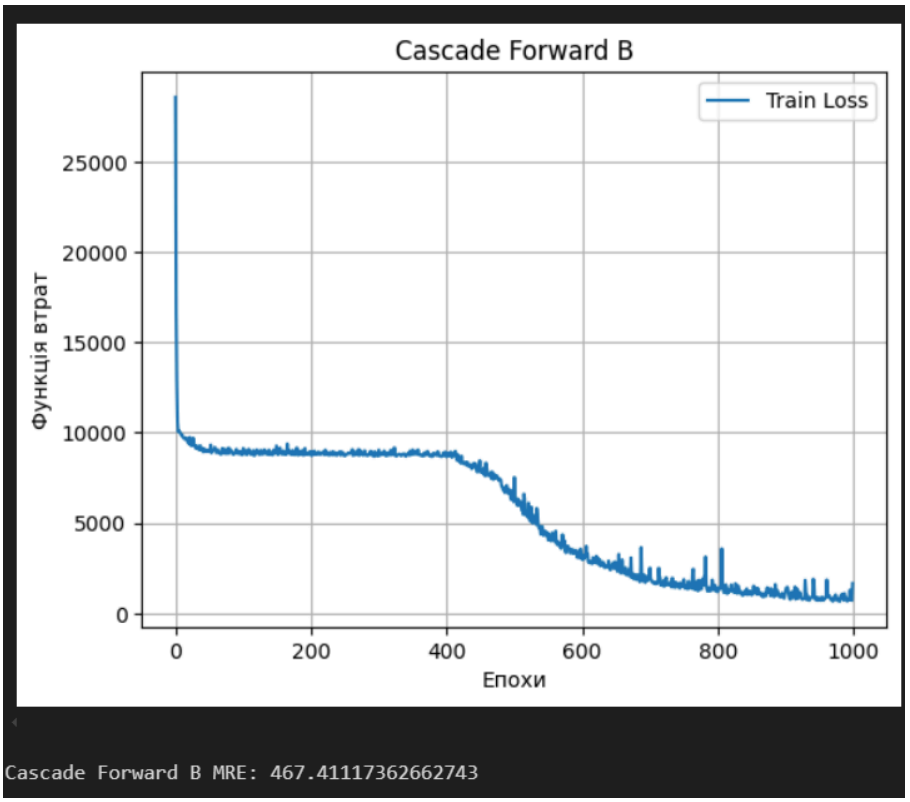

Рисунок 6 – Результат моделі Cascade Forward B (2 внутрішніх шари по 10 нейронів у кожному)

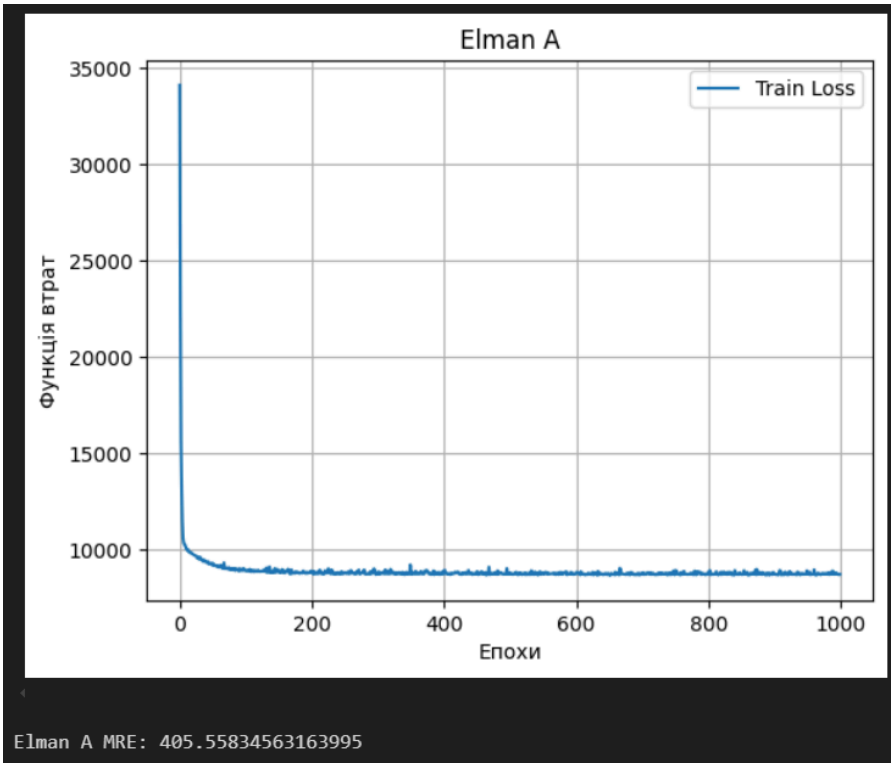

Рисунок 7 – Результат моделі Elman Backprop A (1 внутрішній шар з 15 нейронами)

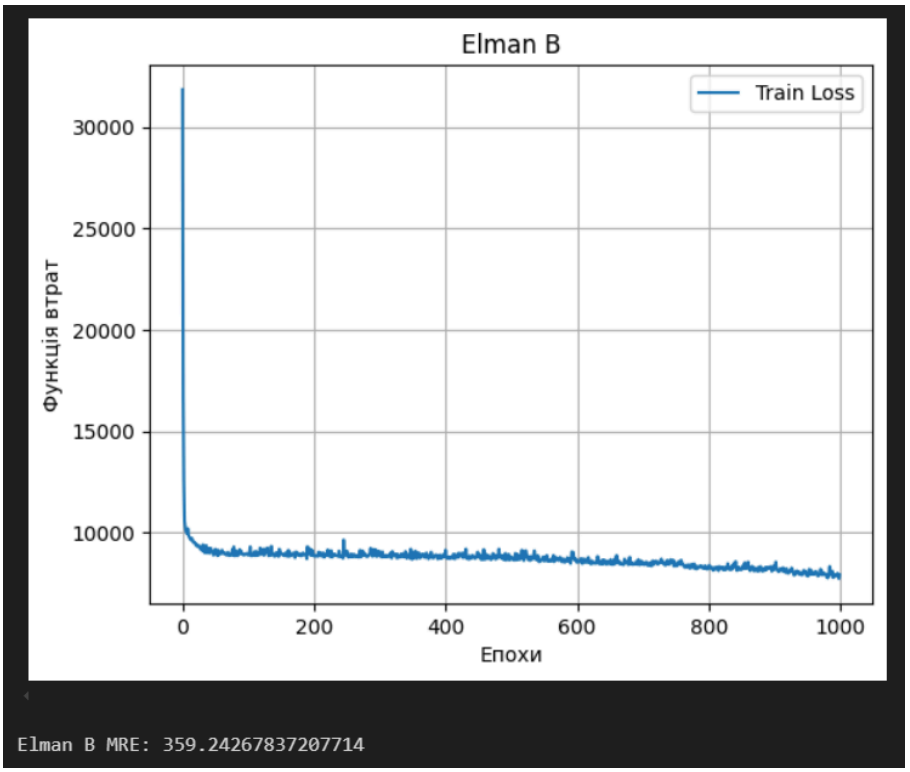

Рисунок 8 – Результат моделі Elman Backprop B (3 внутрішніх шари по 5 нейронів у кожному)

Висновки

В рамках цієї лабораторної роботи було побудовано 6 моделей нейронних мереж з різними архітектурами (Feed Forward, Cascade Forward, Elman Backprop), по 2 на кожну з різними параметрами (а саме, кількістю шарів та нейронах в них). Кращий результат показали моделі з архітектурою Elman Backprop, що вказує на переваги багатошарової рекурентності, яка в цій архітектурі використовується. Найгірший результат отримали моделі на основі архітектури Feed Forward. Це можна пояснити занизькою кількістю шарів та відсутністю рекурентності, що не дозволило достатньо точно змоделювати функцію. Варто уточнити, що загалом всі моделі показали недостатній результат точності моделювання вихідної функції, що пов’язано з її високою нелінійністю, що вимагає значно більшої кількості шарів та нейронів в них, ніж було вказано в завданні. Проте, це не заважає порівняти архітектури цих моделей, враховуючи, що моделі з рекурентними зв’язками показали відчутно кращий результат.