**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**

Комп‘ютерного практикуму № 2 з дисципліни

«Програмні засоби проектування та реалізації нейромережевих систем»

**«Реалізація базових архітектур нейронних мереж»**

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-01 Галько М.В.*

**Перевірив(ла)**

(прізвище, ім'я, по батькові)

*Шимкович В. М.*

Київ 2022

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

**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;

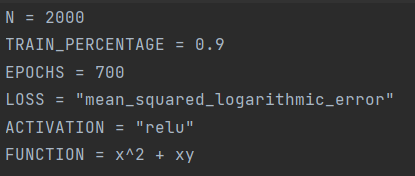
b) 3 внутрішніх шари по 5 нейронів у кожному;

4. Зробити висновки на основі отриманих даних.

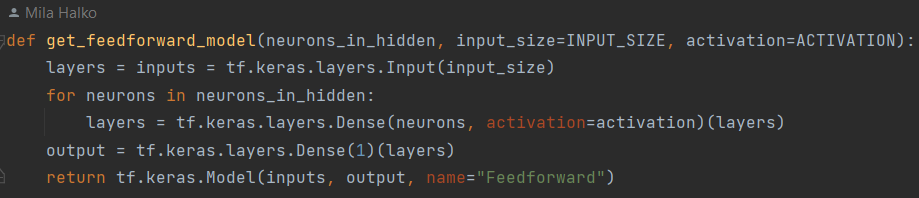
Функція: x2 + xy

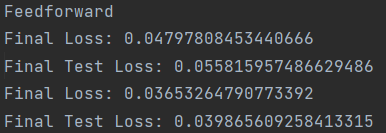
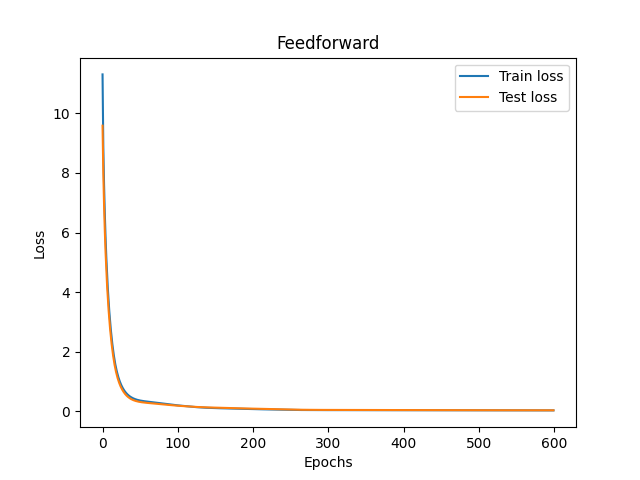
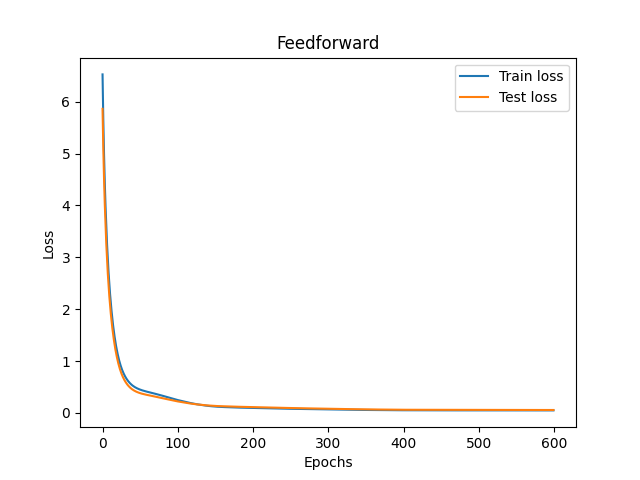
**Результати:**

Дані:

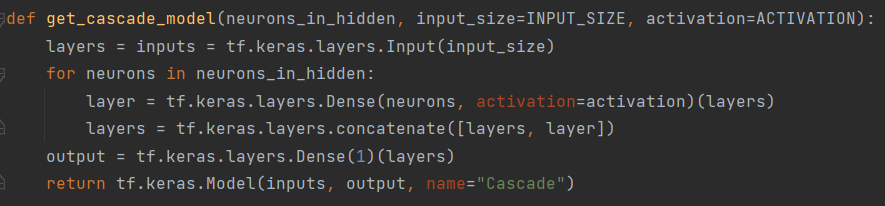


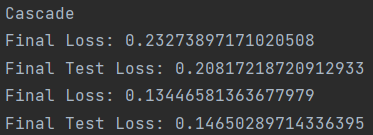
1. Тип мережі: feed forward backprop:
   1. 1 внутрішній шар з 10 нейронами;
   2. 1 внутрішній шар з 20 нейронами;

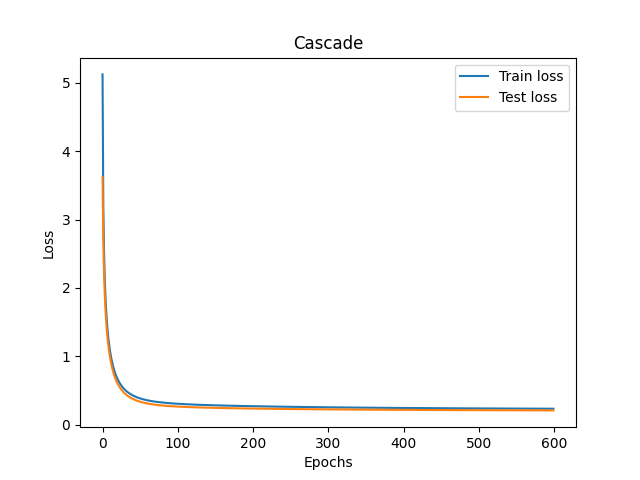
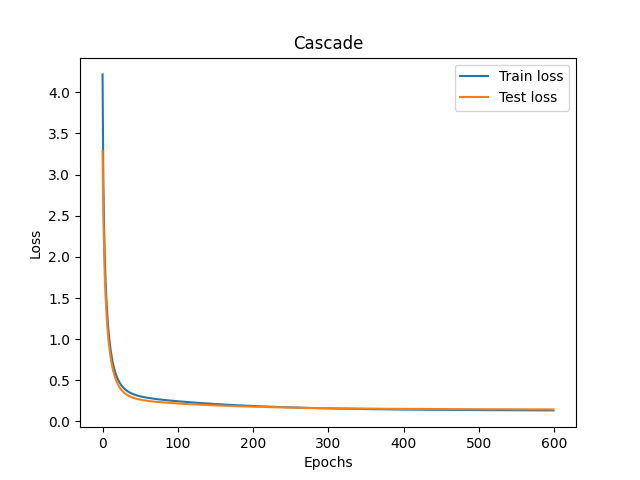




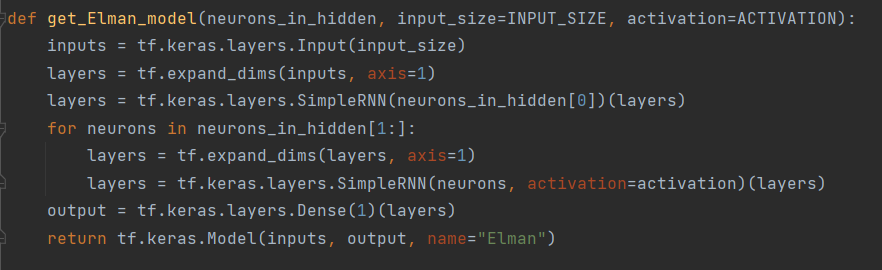
1. Тип мережі: cascade - forward backprop:
   1. 1 внутрішній шар з 20 нейронами;
   2. 2 внутрішніх шари по 10 нейронів у кожному;

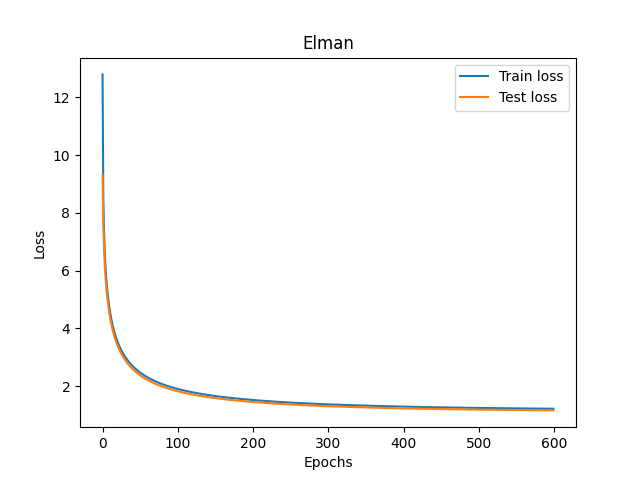
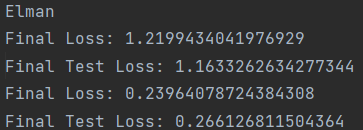


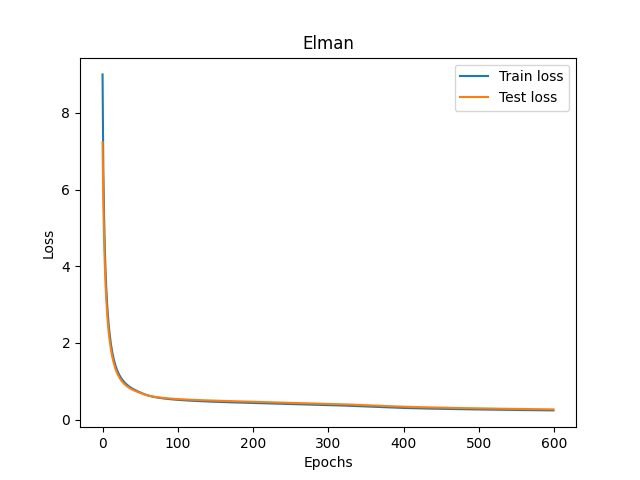




1. Тип мережі: elman backprop:
   1. 1 внутрішній шар з 15 нейронами;
   2. 3 внутрішніх шари по 5 нейронів у кожному;





**Висновок щодо результатів:**

Аналізуючи отримані результати, можна зробити такі висновки:

* Feedforward моделі з більшою кількістю нейронів в шарах (20 нейронів) показали кращу здатність апроксимувати задану функцію і мали нижчі значення втрати на тестових даних. Більша кількість нейронів дозволяє моделі збільшити свою складність і вивчити складніші залежності в даних.
* Cascade моделі не показали таку ж хорошу здатність апроксимувати задану функцію. Модель з меншою кількістю нейронів (10 нейронів) мала кращі результати, що може свідчити про те, що більш складна модель (20 нейронів) не змогла використати всі доступні дані ефективно.
* Elman моделі показали більшу втрату, що може свідчити про те, що ці моделі не змогли ефективно захопити динаміку вхідних послідовностей. Модель з меншою кількістю нейронів (15 нейронів) мала трохи кращі результати порівняно з моделлю з більшою кількістю нейронів (5 нейронів).

У цілому, для цієї конкретної задачі апроксимації функції x^2 + xy, модель Feedforward з більшою кількістю нейронів в шарах (20 нейронів) показала найкращі результати. Оскільки тестові дані кожен раз генеруються, то це спричиняє трохи різні результати.

**Код:**

import numpy as np  
  
from Lab2.NNmodels import \*  
from Lab2.output import \*  
  
# PARAMETERS  
N = 1500  
TRAIN\_PERCENTAGE = 0.8  
EPOCHS = 600  
LOSS = 'mean\_squared\_logarithmic\_error'  
LEARNING\_RATE = tf.keras.optimizers.schedules.ExponentialDecay(  
 initial\_learning\_rate=0.001,  
 decay\_steps=100,  
 decay\_rate=0.99,  
)  
  
# DATA  
inputs = np.random.uniform(10, size=(N, 2))  
z = np.array([x \*\* 2 + x \* y for x, y in inputs])  
inputs\_train = inputs[:int(N \* TRAIN\_PERCENTAGE)]  
z\_train = z[:int(N \* TRAIN\_PERCENTAGE)]  
inputs\_test = inputs[int(N \* TRAIN\_PERCENTAGE):]  
z\_test = z[int(N \* TRAIN\_PERCENTAGE):]  
  
  
def train\_model(model):  
 if model.name == 'Feedforward':  
 model.compile(loss=LOSS, optimizer=tf.keras.optimizers.Adamax(learning\_rate=LEARNING\_RATE))  
 else:  
 model.compile(loss=LOSS, optimizer=tf.keras.optimizers.SGD(learning\_rate=LEARNING\_RATE))  
 history = model.fit(inputs\_train, z\_train, epochs=EPOCHS, validation\_data=(inputs\_test, z\_test), verbose=0)  
 output(model.name, history)  
  
  
def main():  
 set\_txt\_name('final\_results3')  
 test\_start('N = 2000 \nTRAIN\_PERCENTAGE = 0.9 \nEPOCHS = 700 \n' +  
 'LOSS = "mean\_squared\_logarithmic\_error" \nACTIVATION = "relu" \nFUNCTION = x^2 + xy')  
  
 train\_model(get\_feedforward\_model([10]))  
 train\_model(get\_feedforward\_model([20]))  
 train\_model(get\_cascade\_model([20]))  
 train\_model(get\_cascade\_model([10, 10]))  
 train\_model(get\_Elman\_model([15]))  
 train\_model(get\_Elman\_model([5, 5, 5]))  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

import tensorflow as tf  
  
INPUT\_SIZE = 2  
ACTIVATION = 'relu'  
  
  
def get\_feedforward\_model(neurons\_in\_hidden, input\_size=INPUT\_SIZE, activation=ACTIVATION):  
 layers = inputs = tf.keras.layers.Input(input\_size)  
 for neurons in neurons\_in\_hidden:  
 layers = tf.keras.layers.Dense(neurons, activation=activation)(layers)  
 output = tf.keras.layers.Dense(1)(layers)  
 return tf.keras.Model(inputs, output, name="Feedforward")  
  
  
def get\_cascade\_model(neurons\_in\_hidden, input\_size=INPUT\_SIZE, activation=ACTIVATION):  
 layers = inputs = tf.keras.layers.Input(input\_size)  
 for neurons in neurons\_in\_hidden:  
 layer = tf.keras.layers.Dense(neurons, activation=activation)(layers)  
 layers = tf.keras.layers.concatenate([layers, layer])  
 output = tf.keras.layers.Dense(1)(layers)  
 return tf.keras.Model(inputs, output, name="Cascade")  
  
  
def get\_Elman\_model(neurons\_in\_hidden, input\_size=INPUT\_SIZE, activation=ACTIVATION):  
 inputs = tf.keras.layers.Input(input\_size)  
 layers = tf.expand\_dims(inputs, axis=1)  
 layers = tf.keras.layers.SimpleRNN(neurons\_in\_hidden[0])(layers)  
 for neurons in neurons\_in\_hidden[1:]:  
 layers = tf.expand\_dims(layers, axis=1)  
 layers = tf.keras.layers.SimpleRNN(neurons, activation=activation)(layers)  
 output = tf.keras.layers.Dense(1)(layers)  
 return tf.keras.Model(inputs, output, name="Elman")

from matplotlib import pyplot as plt  
  
TXT\_PATH = "Texts/output1.txt"  
  
  
def set\_txt\_name(name):  
 global TXT\_PATH  
 TXT\_PATH = "Texts/" + name + ".txt"  
  
  
def output(model\_name, history):  
 with open(TXT\_PATH, "a") as f:  
 f.write(model\_name + "\n")  
 f.write("Final Loss: " + str(history.history['loss'][-1]) + "\n")  
 f.write("Final Test Loss: " + str(history.history['val\_loss'][-1]) + "\n")  
 f.write("\n")  
 print(model\_name, "finished training")  
  
 plt.title(model\_name)  
 plt.xlabel('Epochs')  
 plt.ylabel('Loss')  
 plt.plot(history.history['loss'], label='Train loss')  
 plt.plot(history.history['val\_loss'], label='Test loss')  
 plt.legend()  
 plt.show()  
  
  
def test\_start(test\_object\_string):  
 with open(TXT\_PATH, "a") as f:  
 f.write('\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n')  
 f.write(test\_object\_string + '\n')  
 f.write('--------------------------------------------------------------------\n')