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) *ІП-01 Галько М.В.* _____

(шифр, прізвище, ім'я, по батькові)

Перевірив(ла) *Шимкович В. М.* _____

(прізвище, ім'я, по батькові)

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 $f(x+y) = x^2+y^2$, 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: $x^2 + xy$

Результати:

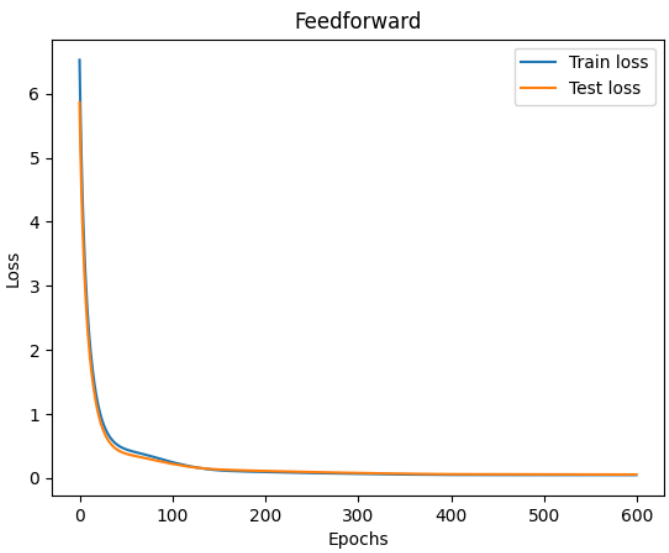
Дані:

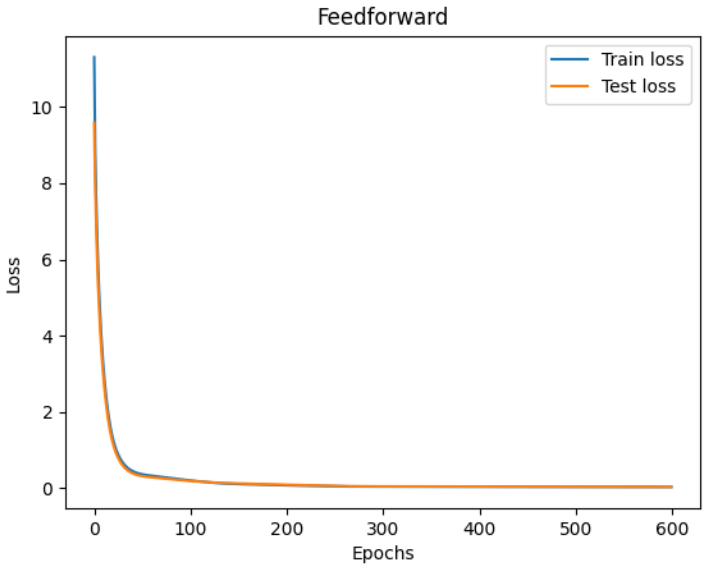
```
N = 2000
TRAIN_PERCENTAGE = 0.9
EPOCHS = 700
LOSS = "mean_squared_logarithmic_error"
ACTIVATION = "relu"
FUNCTION = x^2 + xy
```

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

```
Mila Halko
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")
```

```
Feedforward
Final Loss: 0.04797808453440666
Final Test Loss: 0.055815957486629486
Final Loss: 0.03653264790773392
Final Test Loss: 0.039865609258413315
```

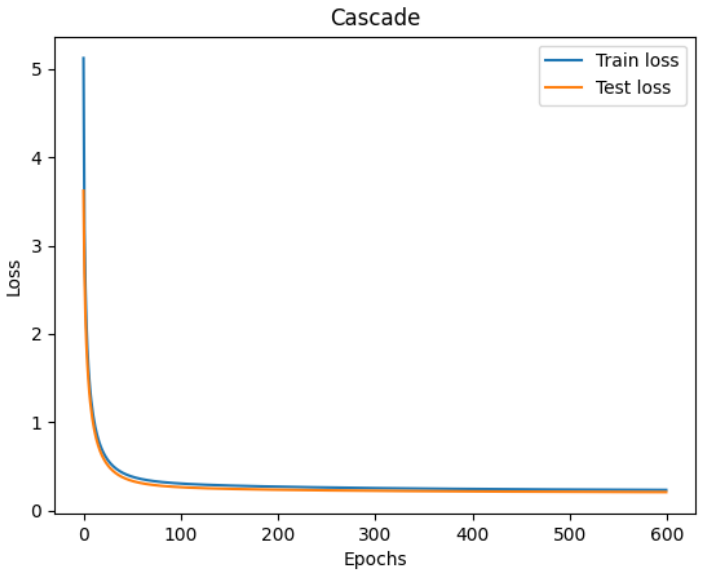


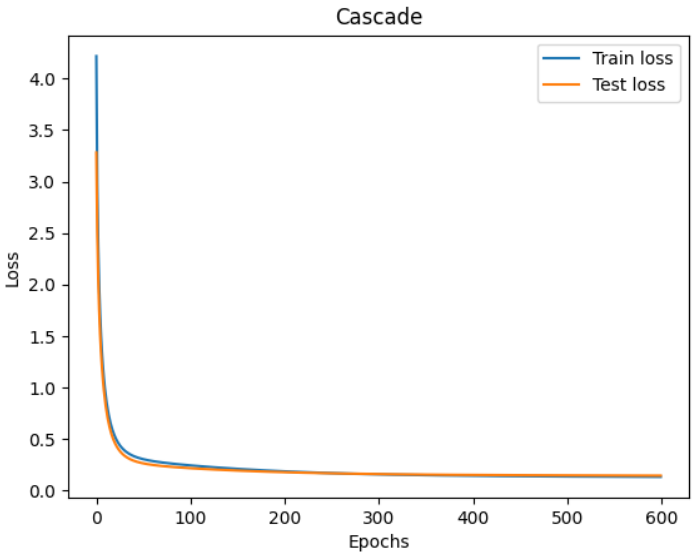


2. Тип мережі: cascade - forward backprop:
    a. 1 внутрішній шар з 20 нейронами;
    b. 2 внутрішніх шари по 10 нейронів у кожному;

```python
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")
```

```
Cascade
Final Loss: 0.23273897171020508
Final Test Loss: 0.20817218720912933
Final Loss: 0.13446581363677979
Final Test Loss: 0.14650289714336395
```

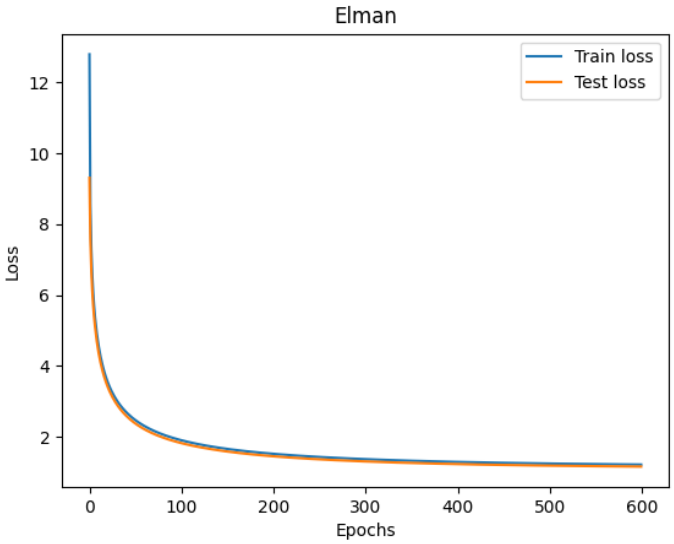


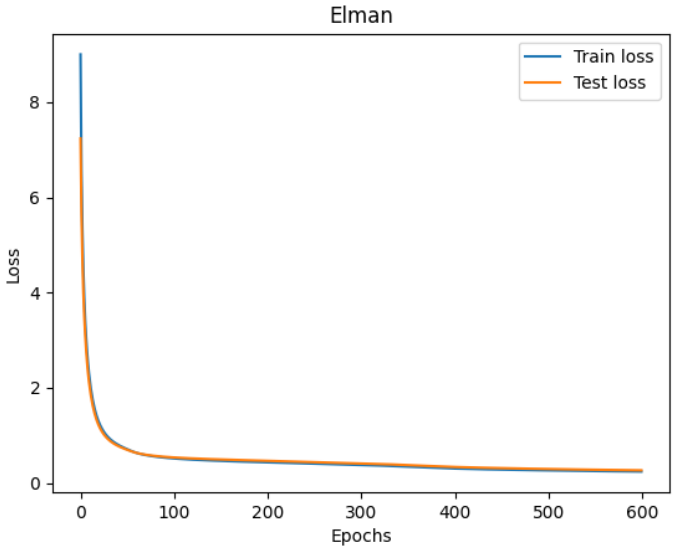


3. Тип мережі: elman backprop:
    a. 1 внутрішній шар з 15 нейронами;
    b. 3 внутрішніх шари по 5 нейронів у кожному;

```python
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")
```

```
Elman
Final Loss: 1.2199434041976929
Final Test Loss: 1.1633262634277344
Final Loss: 0.23964078724384308
Final Test Loss: 0.266126811504364
```





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')
```