НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра технічної кібернетики

Звіт до комп’ютерного практикуму 2 з дисципліни:
“**Програмні засоби проектування та реалізації нейромережевих систем**”

Виконав
ІП-01 Черпак А.В.

Перевірив:
Шимкович В.М.

Київ – 2023

Комп‘ютерний практикум 2

Завдання: Реалізація базових архітектур нейронних мереж

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+y, 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 10 нейронів у кожному;
4. Зробити висновки на основі отриманих даних

Виконання:

Генерація вхідних даних:

```
from pandas import DataFrame
import numpy as np


def arithmetic_operation(x: np.ndarray, y: np.ndarray) -> np.ndarray:
    x_square = x ** 2
    y_square = y ** 2
    return x_square + y_square


def generate_data(data_size: int, min_val=0, max_val=10) -> DataFrame:
    x = np.random.rand(data_size) * (max_val - min_val) + min_val
    y = np.random.rand(data_size) * (max_val - min_val) + min_val
    result = arithmetic_operation(x, y)
    d = {'x': x, 'y': y, 'result': result}
    df = DataFrame(d)
    return df


def data_split(data: DataFrame, train_percent: float) -> tuple[np.ndarray,
np.ndarray]:
    train_idx = int(train_percent * len(data))
    train_data = np.array(data[:train_idx])
    test_data = np.array(data[train_idx:])
    return train_data, test_data
```

Створення моделі:

```
import tensorflow as tf
from keras import Model
from keras.layers import Dense, Input, Concatenate, SimpleRNN


class NeuralNetworkModel:
    def __init__(self, name: str, model: Model):
        self.nn_model_name = name
        self.model = model


class FeedForwardBackprop(NeuralNetworkModel):
    def __init__(self, hidden_neuron_numbers: list[int]):
        neurons_str_repr = f"{hidden_neuron_numbers[0]}"
        for hnn in hidden_neuron_numbers[1:]:
            neurons_str_repr += f", {hnn}"
        model_name = f"FeedForwardBackprop({neurons_str_repr})"

        input_layer = Input(2)
        current_layer = input_layer
        for neurons in hidden_neuron_numbers:
            current_layer = Dense(neurons, activation='relu')(current_layer)
        output_layer = Dense(1, activation='relu')(current_layer)

        model = Model(input_layer, output_layer)
        super().__init__(model_name, model)


class CascadeForwardBackprop(NeuralNetworkModel):
    def __init__(self, hidden_neuron_numbers: list[int]):
        neurons_str_repr = f"{hidden_neuron_numbers[0]}"
        for hnn in hidden_neuron_numbers[1:]:
            neurons_str_repr += f", {hnn}"
        model_name = f"CascadeForwardBackprop({neurons_str_repr})"

        input_layer = Input(2)
        concatenated_layers = input_layer
        for neuron_number in hidden_neuron_numbers:
            hidden_layer = Dense(neuron_number,
activation='relu')(concatenated_layers)
            concatenated_layers = Concatenate(axis=-1)([concatenated_layers,
hidden_layer])
        output_layer = Dense(1, activation='relu')(concatenated_layers)

        model = Model(input_layer, output_layer)
        super().__init__(model_name, model)


class ElmanBackprop(NeuralNetworkModel):
    def __init__(self, hidden_neuron_numbers: list[int]):
        neurons_str_repr = f"{hidden_neuron_numbers[0]}"
        for hnn in hidden_neuron_numbers[1:]:
            neurons_str_repr += f", {hnn}"
        model_name = f"ElmanBackprop({neurons_str_repr})"

        input_layer = Input(2)
        current_layer = tf.expand_dims(input_layer, axis=1)
        current_layer = SimpleRNN(hidden_neuron_numbers[0])(current_layer)
        for neuron number in hidden neuron numbers[1:]:
            current_layer = tf.expand_dims(current_layer, axis=1)
            current_layer = SimpleRNN(neuron_number,
activation='relu')(current_layer)
        output_layer = Dense(1, activation='relu')(current_layer)
```

```
        model = Model(input_layer, output_layer)
        super().__init__(model_name, model)
```

Компіляція, тренування та оцінювання моделі:

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from neural_network_types import FeedForwardBackprop, CascadeForwardBackprop,
ElmanBackprop, NeuralNetworkModel
from training_data_generation import data_split, generate_data


def get_learning_rate(epochs, batch_size):
    initial_learning_rate = 10 ** (-3)
    final_learning_rate = 10 ** (-7)
    learning_rate_decay_factor = (final_learning_rate / initial_learning_rate) **
(1 / epochs)
    steps_per_epoch = int(len(train) / batch_size)
    return tf.keras.optimizers.schedules.ExponentialDecay(
        initial_learning_rate=initial_learning_rate,
        decay_steps=steps_per_epoch,
        decay_rate=learning_rate_decay_factor
    )


def train_model(model_type: type(NeuralNetworkModel), hidden_neurons, train_data,
test_data, epochs_num, batch_sz, learning_rate):
    model_t = model_type(hidden_neurons)
    model = model_t.model
    model.compile(loss='mean_squared_error',
optimizer=tf.keras.optimizers.SGD(learning_rate=learning_rate))
    model.summary()
    return model.fit(np.reshape(train_data[:, :2], (-1, 2)), train_data[:, 2],
epochs=epochs_num, batch_size=batch_sz,
                     validation_data=(np.reshape(test_data[:, :2], (-1, 2)),
test_data[:, 2]), verbose=1).history, model_t.nn_model_name


def graph(train_loss, val_loss, name):
    plt.title(name+'`s mean_squared_error')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.plot(train_loss, label='Train loss')
    plt.plot(val_loss, label='Validation loss')
    plt.grid()
    plt.legend()
    plt.show()


if __name__ == '__main__':
    data = generate_data(1000000)
    train, test = data_split(data, 0.7)
    epochs = 500
    batch_size = 10000
    lr = get_learning_rate(epochs, batch_size)
    cases = [(FeedForwardBackprop, [10]), (FeedForwardBackprop, [20]),
             (CascadeForwardBackprop, [20]), (CascadeForwardBackprop, [10, 10]),
             (ElmanBackprop, [15]), (ElmanBackprop, [10, 10, 10])]

    for (nn_model, hidden_neurons_number) in cases:
        log, name = train_model(nn_model, hidden_neurons_number, train, test,
epochs, batch_size, lr)
        graph(log['loss'], log['val_loss'], name)
```

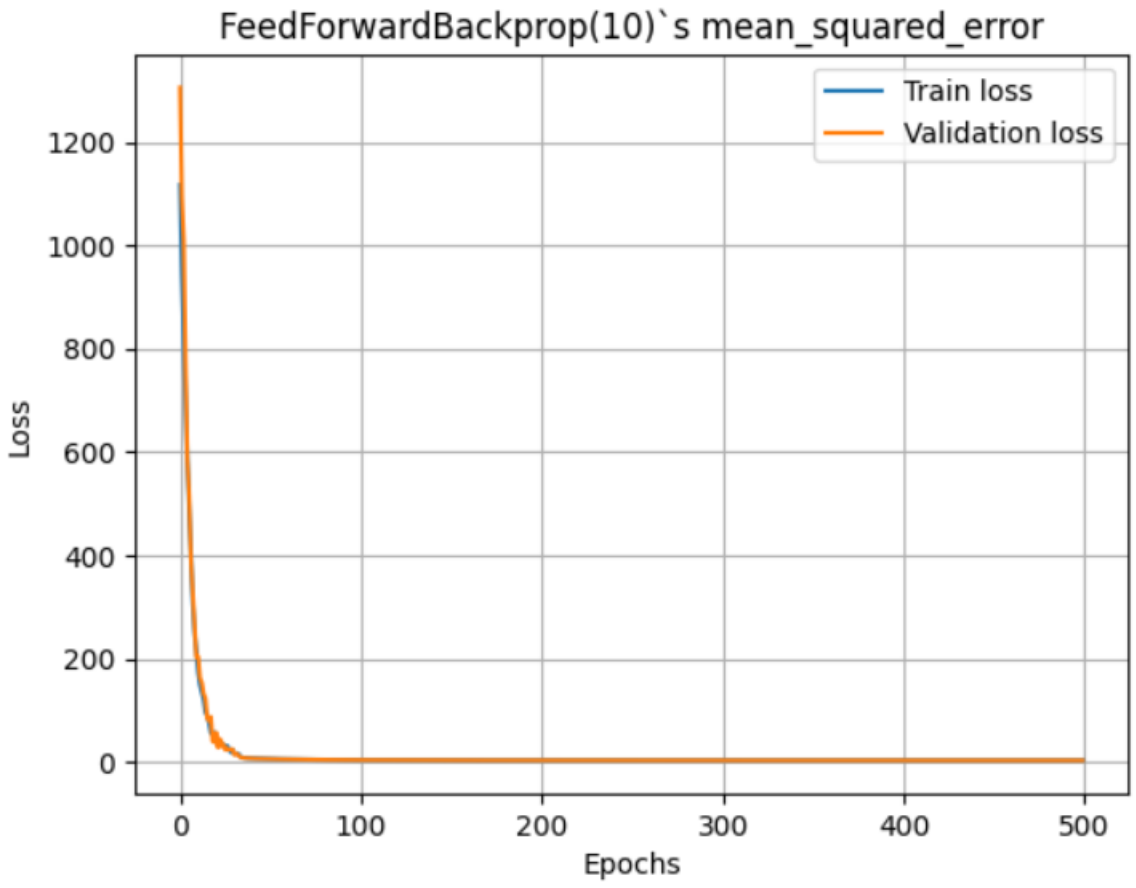
Результати тренування:

FeedForwardBackprop(10):

```
Epoch 1/500
70/70 [==============================] - 1s 4ms/step - loss: 1117.8146 - val_loss: 1306.0601
Epoch 2/500
70/70 [==============================] - 0s 2ms/step - loss: 944.5524 - val_loss: 1094.5344
Epoch 3/500
70/70 [==============================] - 0s 2ms/step - loss: 825.1547 - val_loss: 1009.9774
Epoch 4/500
70/70 [==============================] - 0s 2ms/step - loss: 708.9944 - val_loss: 780.5068
Epoch 5/500
70/70 [==============================] - 0s 2ms/step - loss: 588.4752 - val_loss: 586.8126
Epoch 6/500
```

```
Epoch 477/500
70/70 [==============================] - 0s 2ms/step - loss: 3.1873 - val_loss: 3.2045
Epoch 478/500
70/70 [==============================] - 0s 2ms/step - loss: 3.1873 - val_loss: 3.2044
Epoch 479/500
70/70 [==============================] - 0s 2ms/step - loss: 3.1873 - val_loss: 3.2044
Epoch 480/500
```

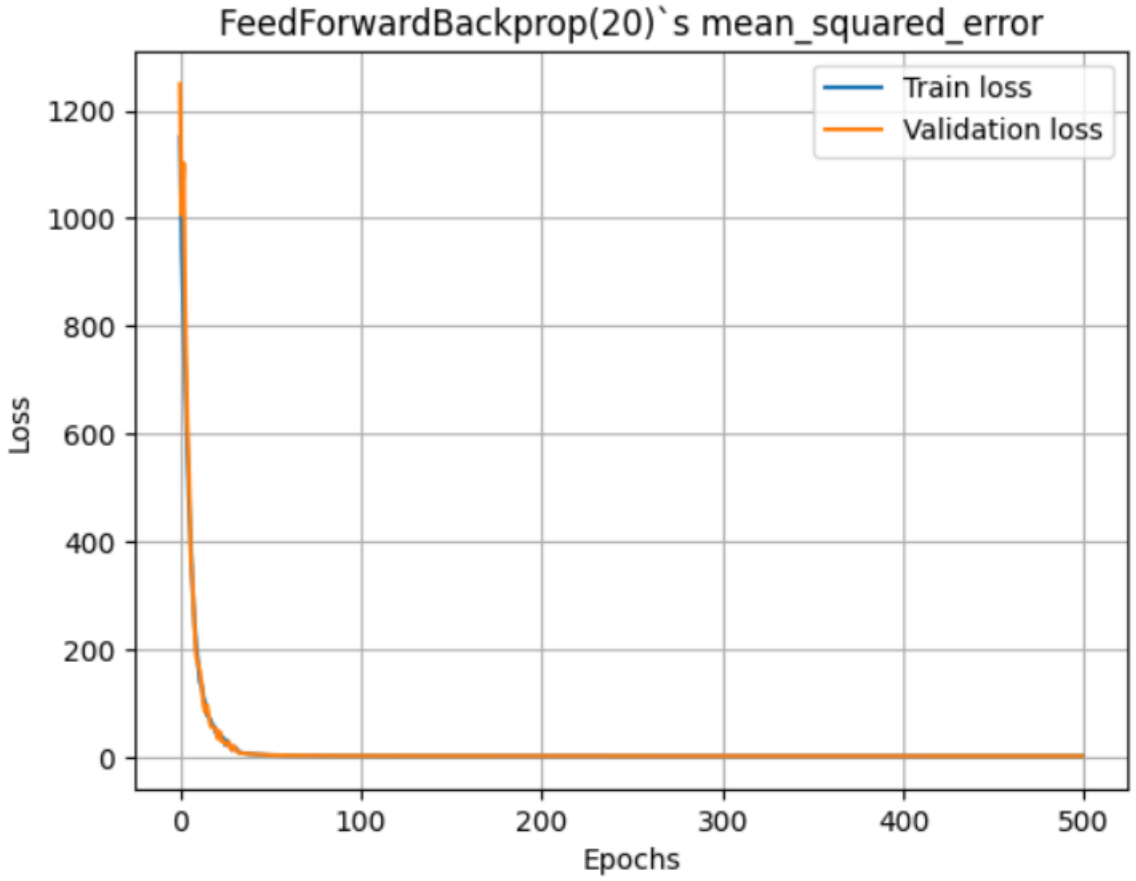
```
Epoch 499/500
70/70 [==============================] - 0s 2ms/step - loss: 3.1872 - val_loss: 3.2044
Epoch 500/500
70/70 [==============================] - 0s 2ms/step - loss: 3.1872 - val_loss: 3.2044
```



FeedForwardBackprop(10, 10):

```
Epoch 1/500
70/70 [==============================] - 1s 4ms/step - loss: 1150.5402 - val_loss: 1248.6161
Epoch 2/500
70/70 [==============================] - 0s 2ms/step - loss: 939.7455 - val_loss: 1006.8422
Epoch 3/500
70/70 [==============================] - 0s 2ms/step - loss: 826.8593 - val_loss: 1102.3696
Epoch 4/500
70/70 [==============================] - 0s 2ms/step - loss: 706.3584 - val_loss: 798.7715
Epoch 5/500
70/70 [==============================] - 0s 2ms/step - loss: 576.7624 - val_loss: 624.9866
Epoch 6/500
70/70 [==============================] - 0s 2ms/step - loss: 470.8712 - val_loss: 519.4092
Epoch 7/500
70/70 [==============================] - 0s 2ms/step - loss: 389.6885 - val_loss: 370.3733
Epoch 8/500
70/70 [==============================] - 0s 2ms/step - loss: 310.9234 - val_loss: 318.3047
Epoch 9/500
70/70 [==============================] - 0s 2ms/step - loss: 252.6063 - val_loss: 242.0045
Epoch 10/500
70/70 [==============================] - 0s 2ms/step - loss: 205.2475 - val_loss: 187.2237
```

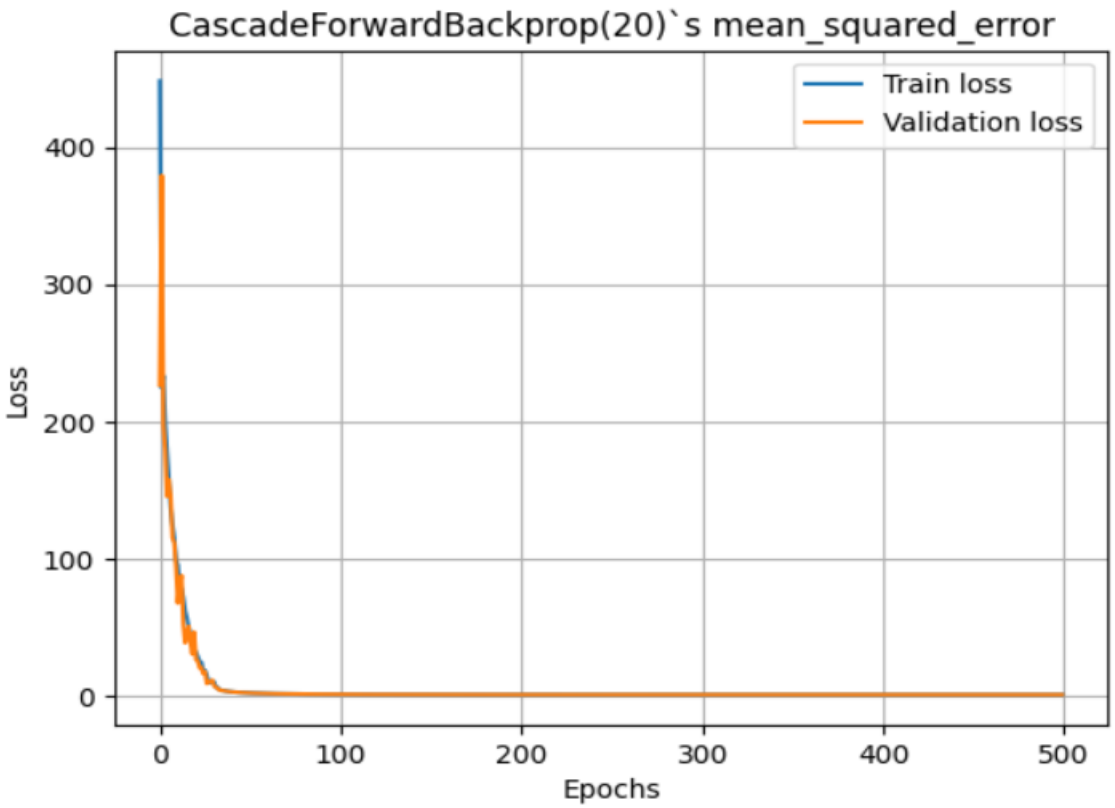
```
Epoch 497/500
70/70 [==============================] - 0s 2ms/step - loss: 2.4442 - val_loss: 2.4425
Epoch 498/500
70/70 [==============================] - 0s 2ms/step - loss: 2.4442 - val_loss: 2.4425
Epoch 499/500
70/70 [==============================] - 0s 2ms/step - loss: 2.4442 - val_loss: 2.4425
Epoch 500/500
70/70 [==============================] - 0s 2ms/step - loss: 2.4442 - val_loss: 2.4425
```



CascadeForwardBackprop(20):

```
Epoch 1/500
70/70 [==============================] - 1s 4ms/step - loss: 448.6706 - val_loss: 226.7935
Epoch 2/500
70/70 [==============================] - 0s 3ms/step - loss: 224.8398 - val_loss: 379.7249
Epoch 3/500
70/70 [==============================] - 0s 3ms/step - loss: 233.5792 - val_loss: 198.2229
Epoch 4/500
70/70 [==============================] - 0s 2ms/step - loss: 199.6011 - val_loss: 179.7559
Epoch 5/500
70/70 [==============================] - 0s 3ms/step - loss: 179.0375 - val_loss: 145.6034
Epoch 6/500
70/70 [==============================] - 0s 3ms/step - loss: 160.2695 - val_loss: 157.8693
Epoch 7/500
70/70 [==============================] - 0s 2ms/step - loss: 136.6147 - val_loss: 146.0373
Epoch 8/500
70/70 [==============================] - 0s 2ms/step - loss: 125.0969 - val_loss: 115.1974
Epoch 9/500
70/70 [==============================] - 0s 2ms/step - loss: 115.4243 - val_loss: 111.9083
Epoch 10/500
70/70 [==============================] - 0s 3ms/step - loss: 98.9364 - val_loss: 93.7528
Epoch 11/500
70/70 [==============================] - 0s 3ms/step - loss: 95.5225 - val_loss: 67.7805
Epoch 12/500
70/70 [==============================] - 0s 3ms/step - loss: 79.9260 - val_loss: 80.9162
```

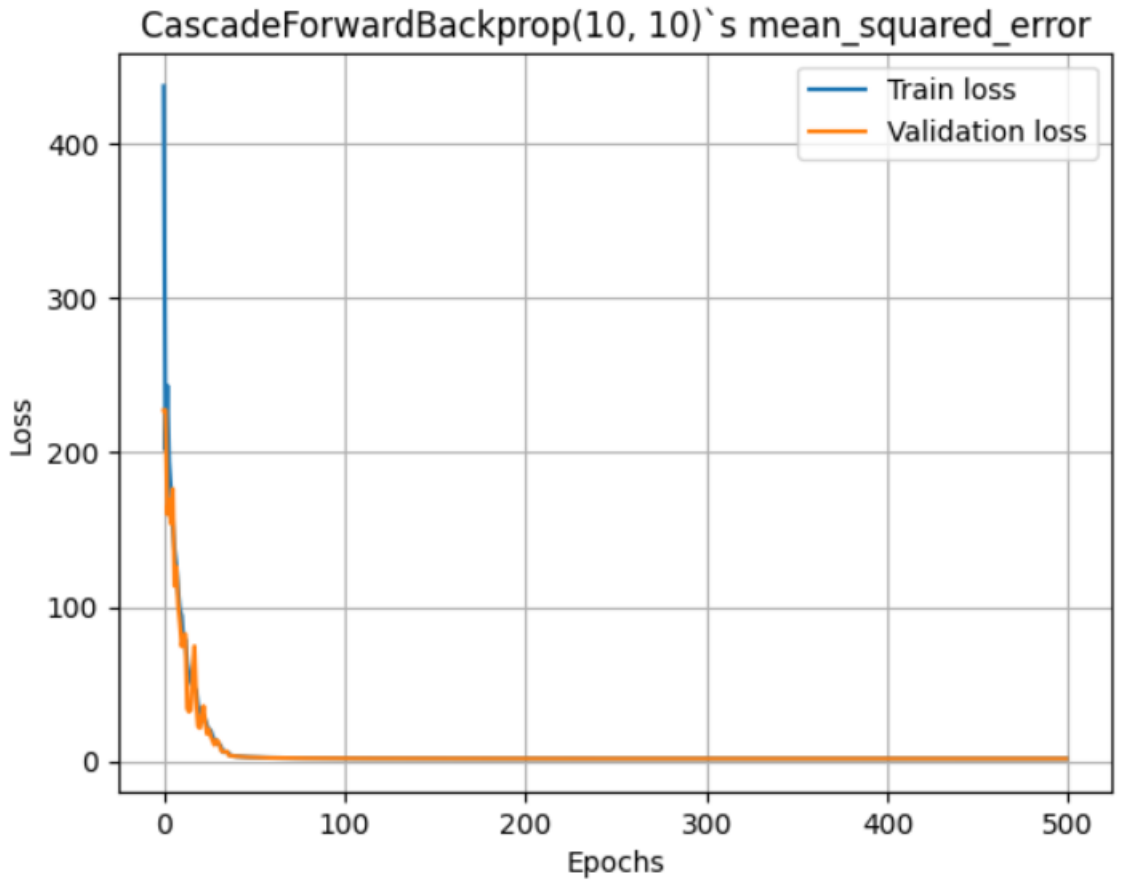
```
Epoch 498/500
70/70 [==============================] - 0s 2ms/step - loss: 0.9378 - val_loss: 0.9379
Epoch 499/500
70/70 [==============================] - 0s 2ms/step - loss: 0.9378 - val_loss: 0.9379
Epoch 500/500
70/70 [==============================] - 0s 2ms/step - loss: 0.9378 - val_loss: 0.9379
```



CascadeForwardBackprop(20):

```
Epoch 1/500
70/70 [==============================] - 1s 4ms/step - loss: 437.2127 - val_loss: 227.2940
Epoch 2/500
70/70 [==============================] - 0s 3ms/step - loss: 201.6760 - val_loss: 227.5972
Epoch 3/500
70/70 [==============================] - 0s 3ms/step - loss: 243.3343 - val_loss: 159.9891
Epoch 4/500
70/70 [==============================] - 0s 3ms/step - loss: 194.0298 - val_loss: 169.5131
Epoch 5/500
70/70 [==============================] - 0s 3ms/step - loss: 175.4962 - val_loss: 153.9555
Epoch 6/500
70/70 [==============================] - 0s 3ms/step - loss: 154.0896 - val_loss: 176.3705
Epoch 7/500
70/70 [==============================] - 0s 3ms/step - loss: 137.6605 - val_loss: 113.7114
Epoch 8/500
70/70 [==============================] - 0s 3ms/step - loss: 127.2630 - val_loss: 126.0399
Epoch 9/500
70/70 [==============================] - 0s 3ms/step - loss: 110.3120 - val_loss: 101.2145
Epoch 10/500
70/70 [==============================] - 0s 3ms/step - loss: 97.7200 - val_loss: 89.1418
Epoch 11/500
70/70 [==============================] - 0s 3ms/step - loss: 94.2734 - val_loss: 74.4249
```

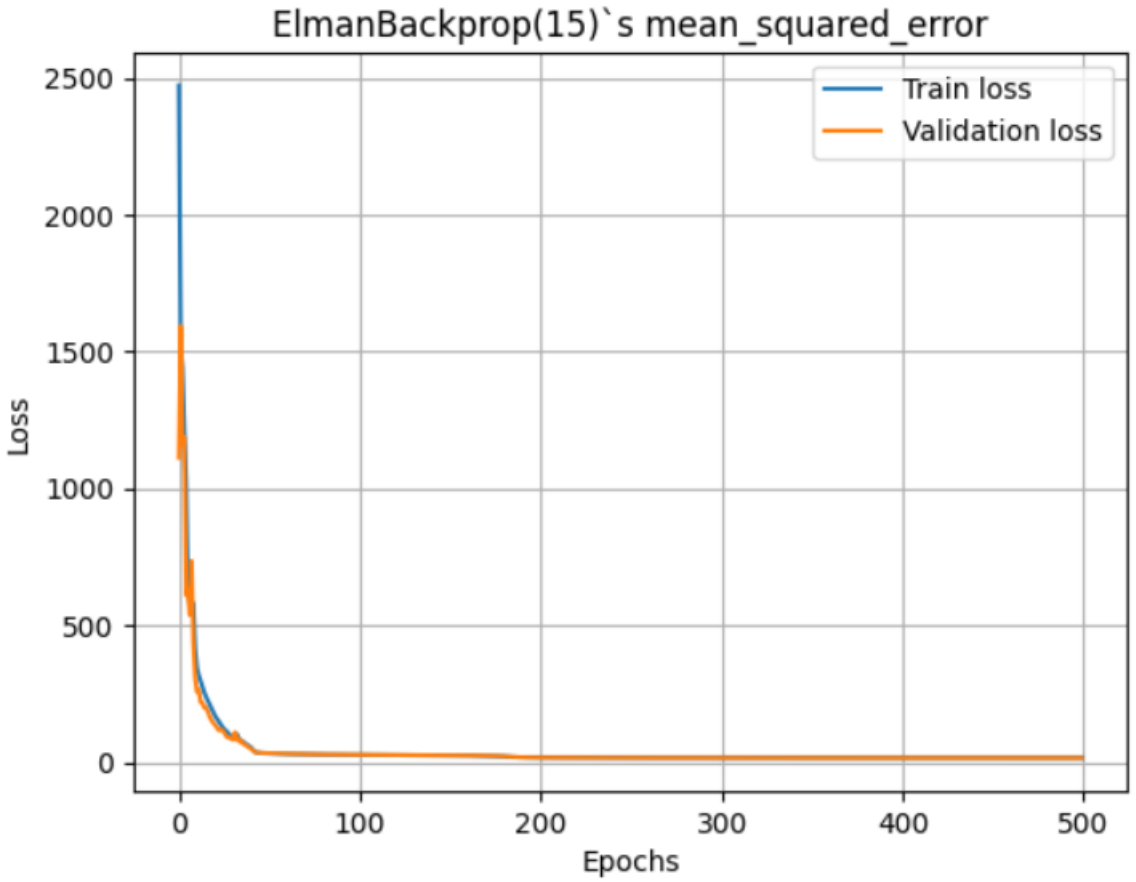
```
Epoch 498/500
70/70 [==============================] - 0s 3ms/step - loss: 1.7783 - val_loss: 1.7796
Epoch 499/500
70/70 [==============================] - 0s 3ms/step - loss: 1.7783 - val_loss: 1.7796
Epoch 500/500
70/70 [==============================] - 0s 3ms/step - loss: 1.7783 - val_loss: 1.7796
```



ElmanBackprop(15):

```
Epoch 1/500
70/70 [==============================] - 1s 5ms/step - loss: 2472.9741 - val_loss: 1113.285
Epoch 2/500
70/70 [==============================] - 0s 3ms/step - loss: 1527.1353 - val_loss: 1594.726
Epoch 3/500
70/70 [==============================] - 0s 3ms/step - loss: 1443.7662 - val_loss: 1159.370
Epoch 4/500
70/70 [==============================] - 0s 3ms/step - loss: 1215.3755 - val_loss: 1187.788
Epoch 5/500
70/70 [==============================] - 0s 3ms/step - loss: 1031.6665 - val_loss: 610.8369
Epoch 6/500
70/70 [==============================] - 0s 3ms/step - loss: 652.4628 - val_loss: 630.1991
Epoch 7/500
70/70 [==============================] - 0s 3ms/step - loss: 581.4185 - val_loss: 538.3507
Epoch 8/500
70/70 [==============================] - 0s 3ms/step - loss: 593.0952 - val_loss: 736.5379
Epoch 9/500
70/70 [==============================] - 0s 3ms/step - loss: 579.1116 - val_loss: 438.6931
Epoch 10/500
70/70 [==============================] - 0s 3ms/step - loss: 415.5974 - val_loss: 305.9969
```

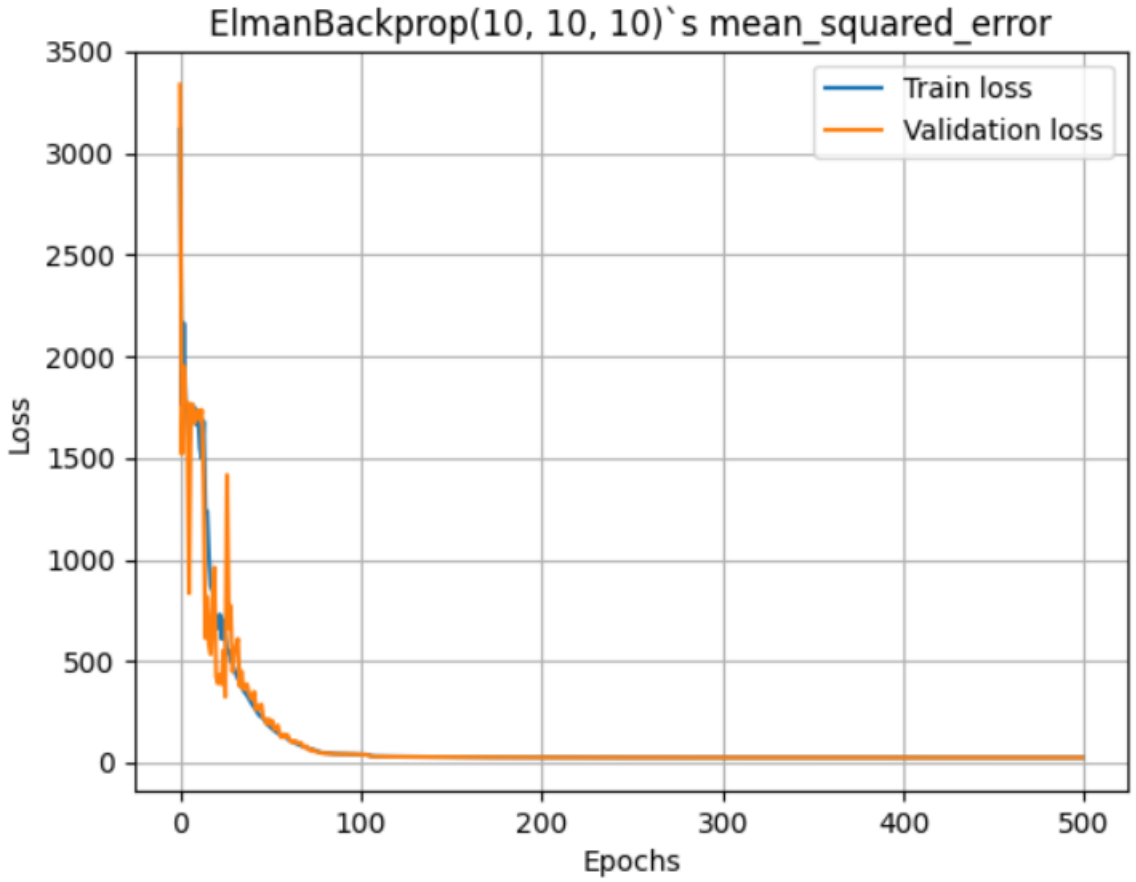
```
Epoch 497/500
70/70 [==============================] - 0s 3ms/step - loss: 16.6915 - val_loss: 16.7442
Epoch 498/500
70/70 [==============================] - 0s 3ms/step - loss: 16.6914 - val_loss: 16.7442
Epoch 499/500
70/70 [==============================] - 0s 3ms/step - loss: 16.6914 - val_loss: 16.7441
Epoch 500/500
70/70 [==============================] - 0s 3ms/step - loss: 16.6914 - val_loss: 16.7441
```



ElmanBackprop(10, 10, 10):

```
Epoch 1/500
70/70 [==============================] - 2s 9ms/step - loss: 3115.4670 - val_loss: 3336.4592
Epoch 2/500
70/70 [==============================] - 0s 5ms/step - loss: 1760.8567 - val_loss: 1518.5374
Epoch 3/500
70/70 [==============================] - 0s 5ms/step - loss: 2162.8018 - val_loss: 1952.5596
Epoch 4/500
70/70 [==============================] - 0s 5ms/step - loss: 1780.9874 - val_loss: 1775.3507
Epoch 5/500
70/70 [==============================] - 0s 5ms/step - loss: 1732.0470 - val_loss: 1769.1289
Epoch 6/500
70/70 [==============================] - 0s 5ms/step - loss: 1579.1930 - val_loss: 834.9105
Epoch 7/500
70/70 [==============================] - 0s 5ms/step - loss: 1757.5739 - val_loss: 1767.0723
Epoch 8/500
70/70 [==============================] - 0s 5ms/step - loss: 1759.4290 - val_loss: 1760.9795
Epoch 9/500
70/70 [==============================] - 0s 5ms/step - loss: 1745.6342 - val_loss: 1672.3662
```

```
Epoch 495/500
70/70 [==============================] - 0s 5ms/step - loss: 25.6415 - val_loss: 25.5791
Epoch 496/500
70/70 [==============================] - 0s 5ms/step - loss: 25.6415 - val_loss: 25.5790
Epoch 497/500
70/70 [==============================] - 0s 5ms/step - loss: 25.6414 - val_loss: 25.5790
Epoch 498/500
70/70 [==============================] - 0s 5ms/step - loss: 25.6414 - val_loss: 25.5790
Epoch 499/500
70/70 [==============================] - 0s 5ms/step - loss: 25.6414 - val_loss: 25.5790
Epoch 500/500
70/70 [==============================] - 0s 5ms/step - loss: 25.6414 - val_loss: 25.5789
```



Висновок:

Під час виконання комп'ютерного практикуму ми реалізували нейромережі трьох різних типів зі змінною кількістю шарів та нейронів у них. Потім ці нейромережі були протестовані шляхом моделювання функції двох змінних $f(x, y) = x^2+y^2$. Найкращі результати показала мережа cascade-forward-backprop з одним прихованим шаром у 20 нейронів. Втім, варто визнати, що з кожним запуском результати дуже відрізняються, а інколи певна мережа може взагалі відмовитися тренуватися.