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:

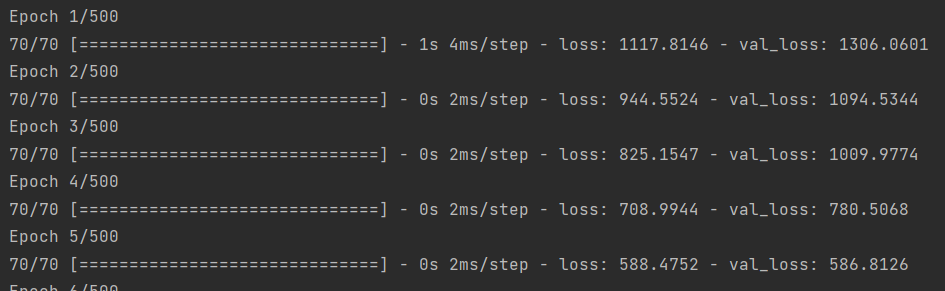
import tensorflow as tf  
from keras import Model  
from keras.layers import Dense, Input, Concatenate, SimpleRNN  
  
  
class NeuralNetworkModel:  
 def \_\_init\_\_(self, name: str, model: Model):  
 self.nn\_model\_name = name  
 self.model = model  
  
  
class FeedForwardBackprop(NeuralNetworkModel):  
 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)

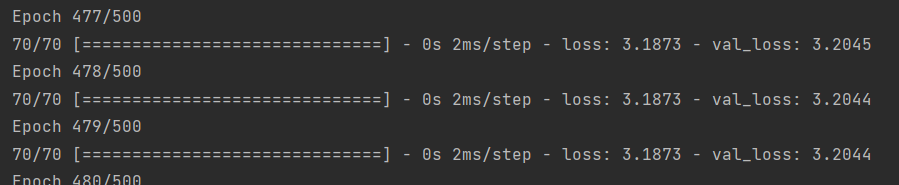
Компіляція, тренування та оцінювання моделі:

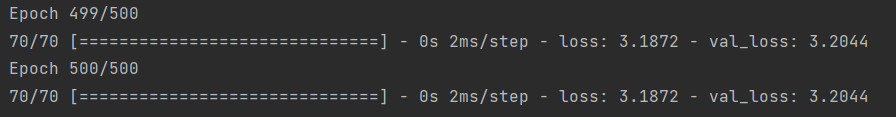
import tensorflow as tf  
import numpy as np  
import matplotlib.pyplot as plt  
from neural\_network\_types import FeedForwardBackprop, CascadeForwardBackprop, ElmanBackprop, NeuralNetworkModel  
from training\_data\_generation import data\_split, generate\_data  
  
  
def get\_learning\_rate(epochs, 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)

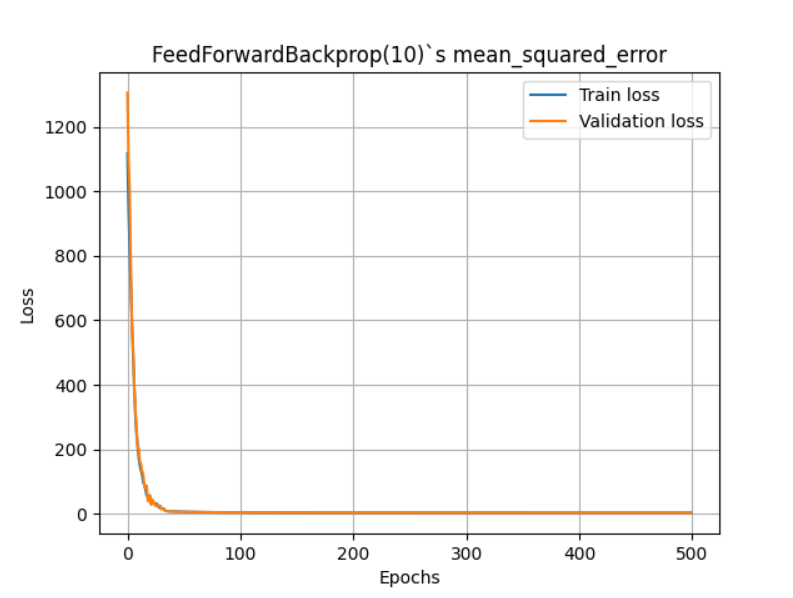
Результати тренування:

FeedForwardBackprop(10):

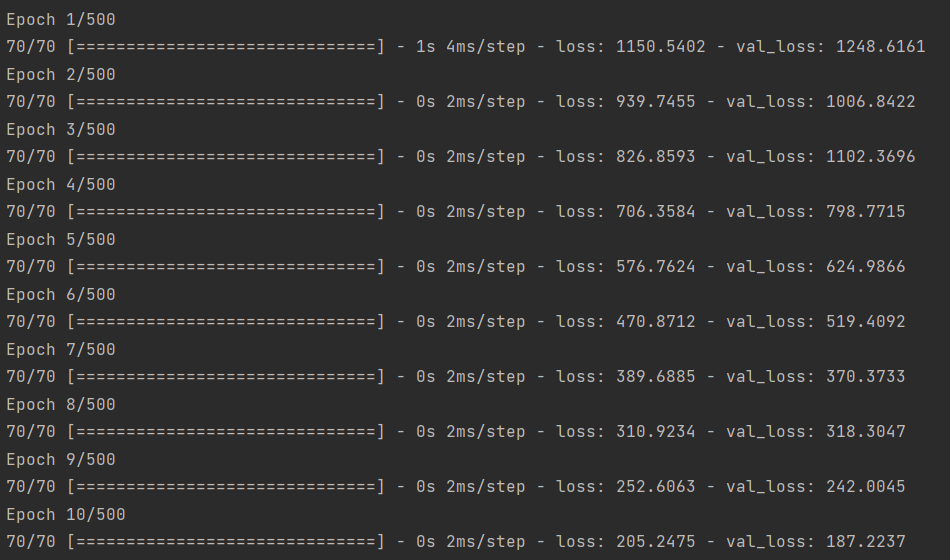


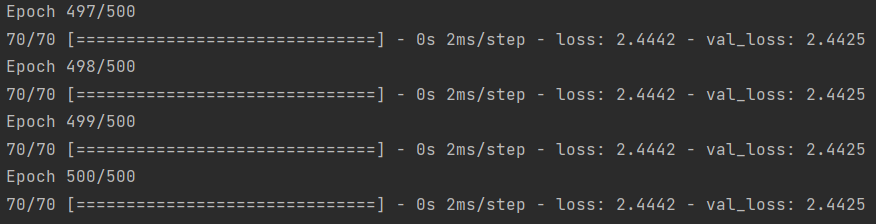


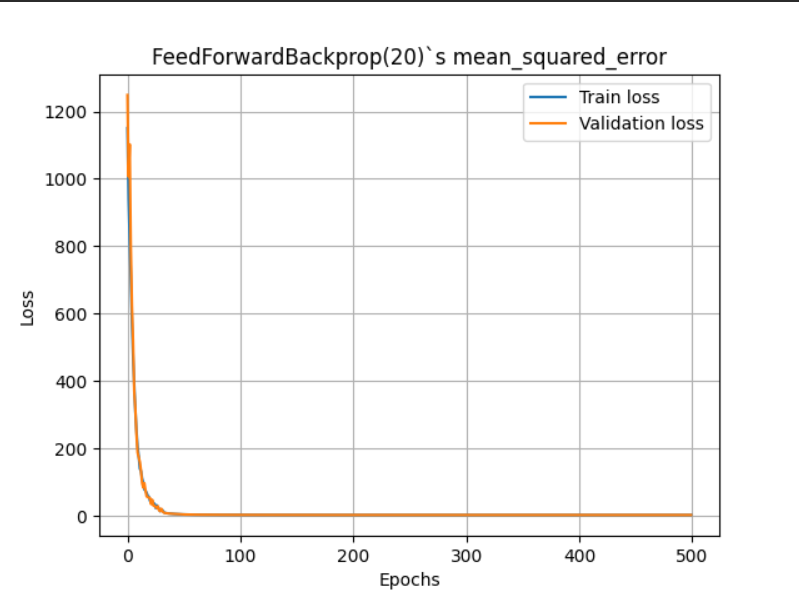




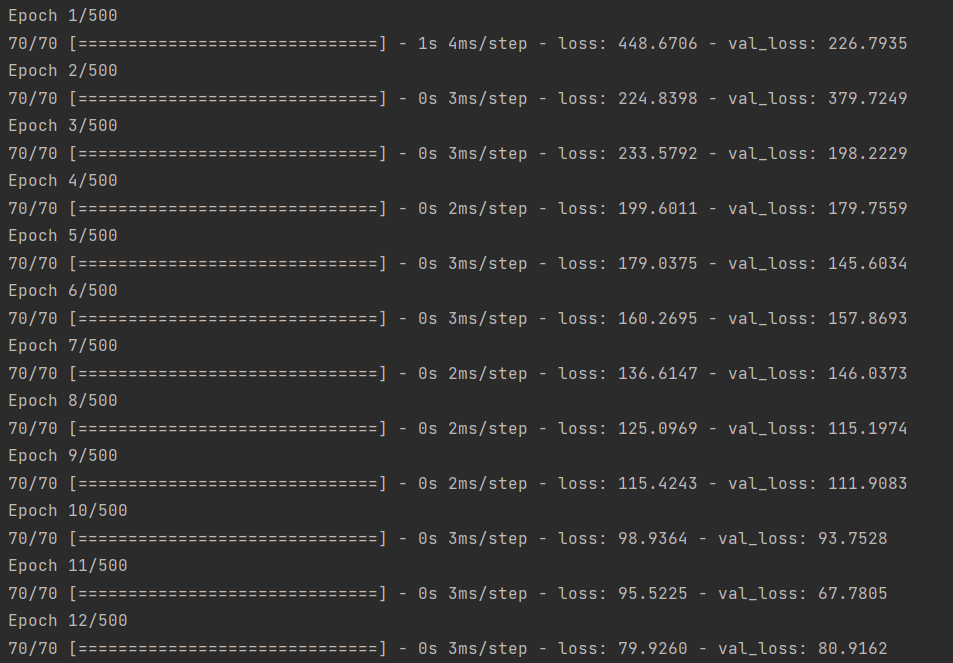
FeedForwardBackprop(10, 10):

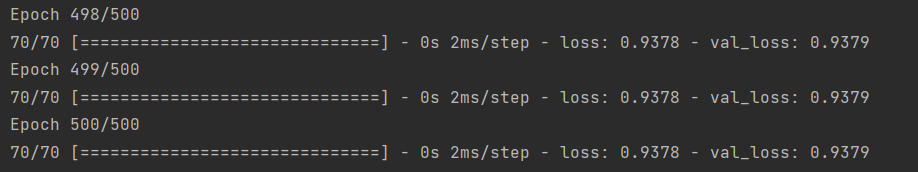


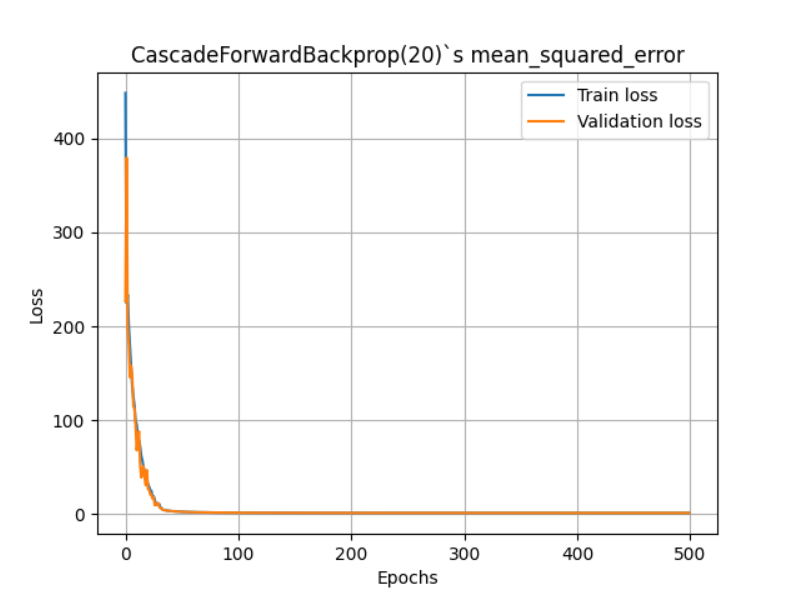




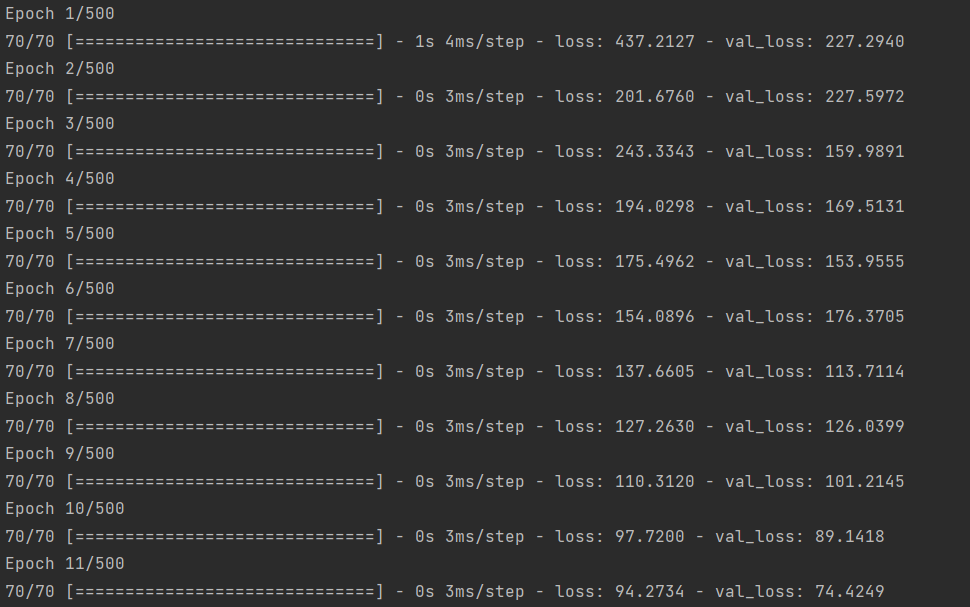
CascadeForwardBackprop(20):

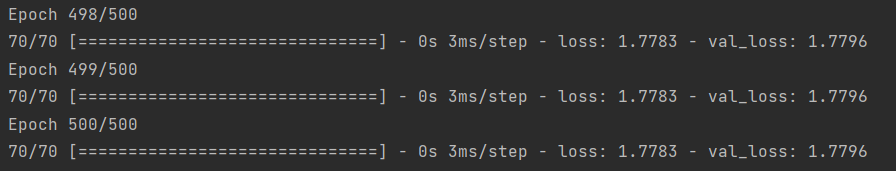


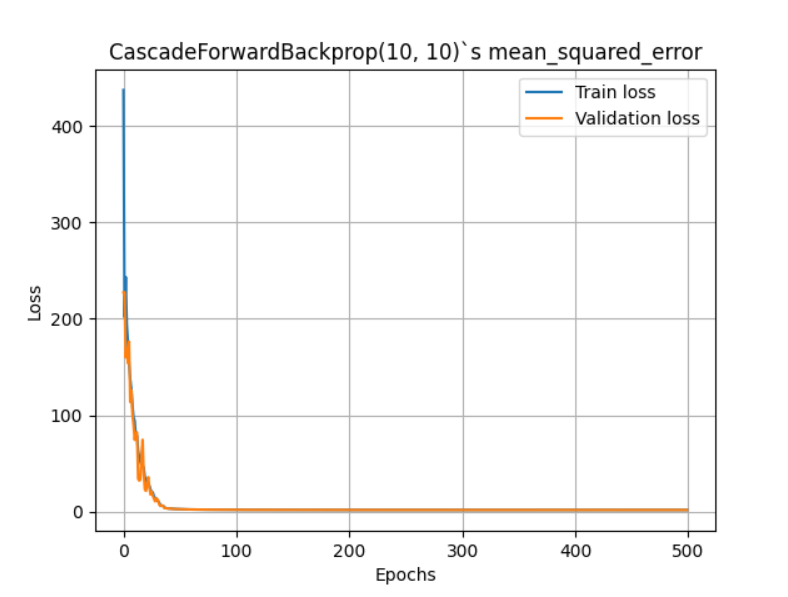




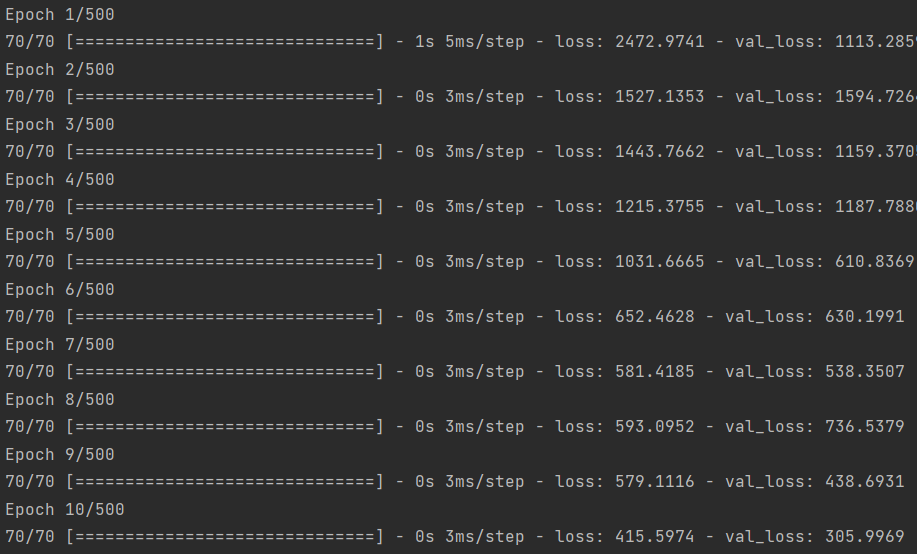
CascadeForwardBackprop(20):

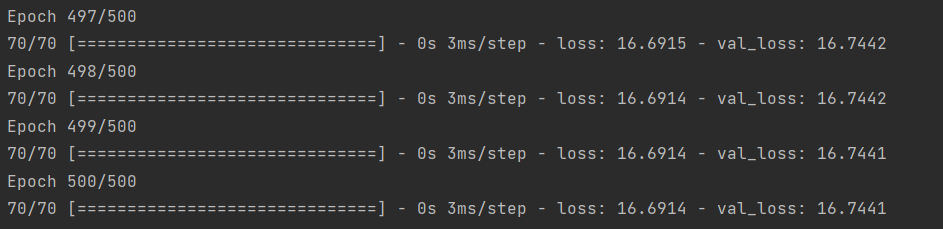


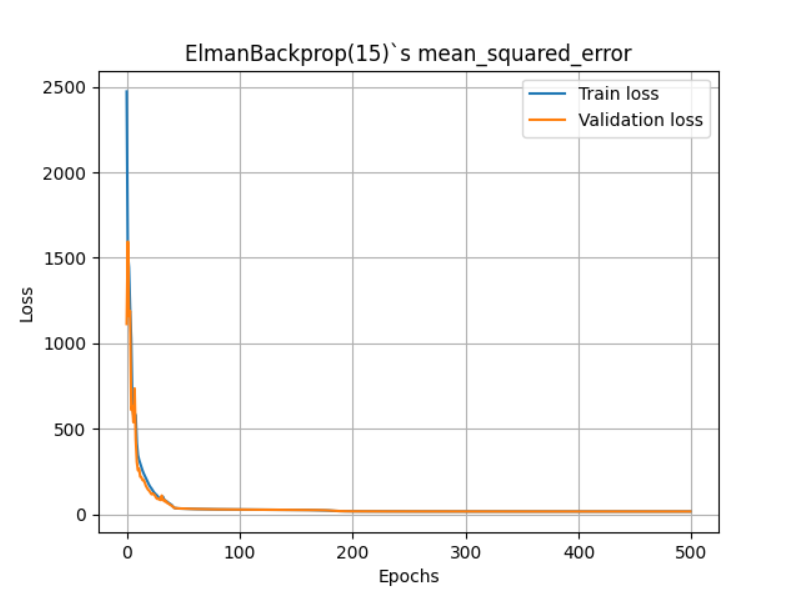




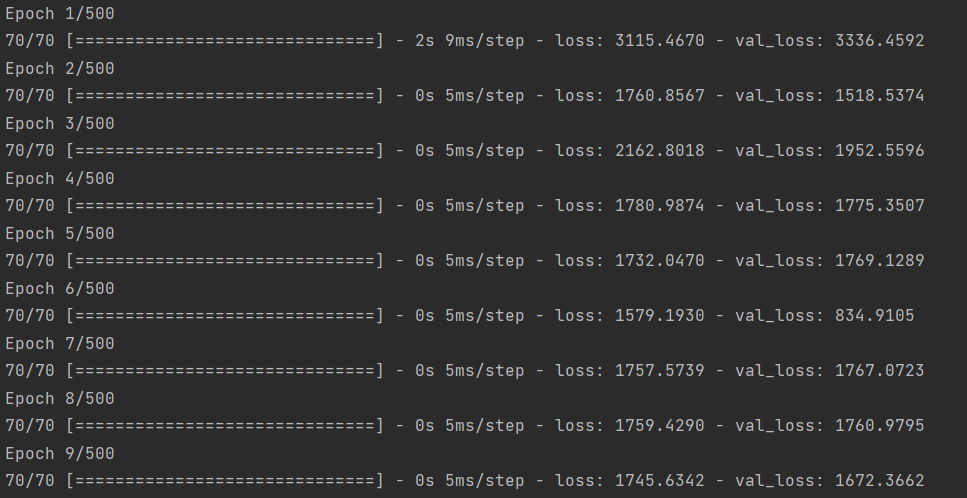
ElmanBackprop(15):

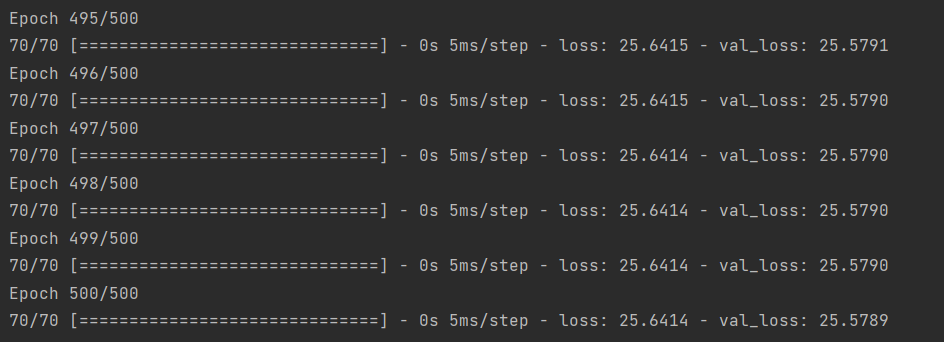


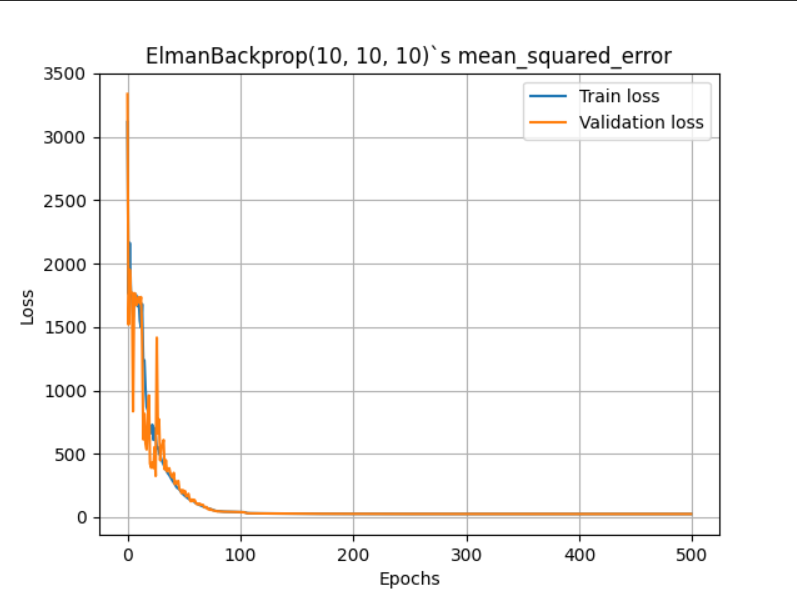




ElmanBackprop(10, 10, 10):







**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) = x2+y2. 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.