Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота №2**

з дисципліни «Програмні засоби проектування та реалізаціїї нейромережевих систем»

Тема: " Реалізація базових архітектур нейронних мереж"

Виконав: Перевірив:

студент групи ІП-93 Шимкович Володимир

Домінський Валентин Миколайович

Олексійович

Київ 2022

Зміст:

[Мета: 3](#_Toc103010390)

[Вихідний код 3](#_Toc103010391)

[Результат роботи: 3](#_Toc103010392)

[Висновки: 7](#_Toc103010393)

## Мета:

Написати програму, що реалізує нейронні мережі для моделювання функції двох змінних. Функцію 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.

## Вихідний код

import tensorflow as tf

import tensorflow as tf

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

from tensorflow.keras import Sequential, Input, Model

from tensorflow.keras.layers import Dense, SimpleRNN

from keras import layers, models

start\_pos = 0

end\_pos = 10

epochs = 15

validation\_split = 0.25

elman\_step = 1

new\_x\_train = []

new\_y\_train = []

def example\_function(x):

return np.sin(x / 10) + 0.04 \* x \*\* 2

train\_x = np.arange(start\_pos, end\_pos, 0.001)

train\_y = example\_function(train\_x)

# Feed forward backprop

# It iteratively learns a set of weights for prediction

# of the class label of tuples. A multilayer feed-forward

# neural network consists of an input layer,

# one or more hidden layers, and an output layer

# РІРЅСѓС‚СЂС–С€РЅС–Р№ С€Р°СЂ Р· 10 РЅРµР№СЂРѕРЅР°РјРё

ffb\_10\_model = models.Sequential()

ffb\_10\_model.add(Input(shape=(1)))

# A type of activation function that transforms the

# value results of a neuron. The transformation imposed

# by ReLU on values from a neuron is represented by

# the formula y=max(0,x). The ReLU activation function

# clamps down any negative values from the neuron to 0,

# and positive values remain unchanged. The result of this

# mathematical transformation is utilized as the output

# of the current layer and used as input to

# a consecutive layer within a neural network

ffb\_10\_model.add(Dense(10, activation='relu'))

ffb\_10\_model.add(Dense(1))

# Mean squared error

ffb\_10\_model.compile(loss='mse', optimizer='adam', metrics=['accuracy'])

ffb\_10\_model.fit(train\_x, train\_y, epochs = epochs, validation\_split = validation\_split)

# РІРЅСѓС‚СЂС–С€РЅС–Р№ С€Р°СЂ Р· 20 РЅРµР№СЂРѕРЅР°РјРё

ffb\_20\_model = models.Sequential()

ffb\_20\_model.add(Input(shape=(1)))

ffb\_20\_model.add(Dense(20, activation='relu'))

ffb\_20\_model.add(Dense(1))

ffb\_20\_model.compile(loss='mse', optimizer='adam', metrics=['accuracy'])

ffb\_20\_model.fit(train\_x, train\_y, epochs = epochs, validation\_split = validation\_split)

plt.plot(train\_x, train\_y, label='Function')

plt.plot(train\_x, ffb\_10\_model.predict(train\_x), label='10')

plt.plot(train\_x, ffb\_20\_model.predict(train\_x), label='20')

plt.legend()

"""# Cascade-forward backprop"""

# cascade - forward backprop

# Cascade-forward neural network is a class of neural network

# which is similar to feed-forward networks, but include a

# connection from the input and every previous layer to

# following layers. In a network which has three layers,

# the output layer is also connected directly with

# the input layer beside with hidden layer.

# РІРЅСѓС‚СЂС–С€РЅС–Р№ С€Р°СЂ Р· 20 РЅРµР№СЂРѕРЅР°РјРё

input = Input(shape=(1))

first\_hidden = layers.Dense(20, activation='relu')(input)

first\_output = layers.add([first\_hidden, input])

output = layers.Dense(1)(first\_output)

second\_output = layers.add([output, input])

cfb\_20\_model = Model(input, second\_output)

cfb\_20\_model.compile(loss='mse', optimizer='adam', metrics=['accuracy'])

cfb\_20\_model.fit(train\_x, train\_y, epochs = epochs, validation\_split = validation\_split)

cfb\_20\_model.summary()

# 2 РІРЅСѓС‚СЂС–С€РЅС–С… С€Р°СЂРё РїРѕ 10 РЅРµР№СЂРѕРЅС–РІ Сѓ РєРѕР¶РЅРѕРјСѓ

input = Input(shape=(1))

first\_hidden = layers.Dense(10, activation='relu')(input)

first\_output = layers.add([first\_hidden, input])

second\_hidden = layers.Dense(10, activation='relu')(first\_output)

second\_output = layers.add([second\_hidden, input])

output = layers.Dense(1)(second\_output)

third\_output = layers.add([output, input])

cfb\_10\_model = Model(input, third\_output)

cfb\_10\_model.compile(loss='mse', optimizer='adam', metrics=['accuracy'])

cfb\_10\_model.fit(train\_x, train\_y, epochs = epochs, validation\_split = validation\_split)

plt.plot(train\_x, train\_y, label='Function')

plt.plot(train\_x, cfb\_20\_model.predict(train\_x), label='20')

plt.plot(train\_x, cfb\_10\_model.predict(train\_x), label='10 \* 2')

plt.legend()

# Elman backprop

# Goes forward, if smth bad happens, then goes back

# 1 РІРЅСѓС‚СЂС–С€РЅС–Р№ С€Р°СЂ Р· 15 РЅРµР№СЂРѕРЅР°РјРё

for i in range(len(train\_x) - elman\_step):

point = i + elman\_step

new\_x\_train.append(train\_x[i:point,])

new\_y\_train.append(train\_y[point,])

new\_x\_train = np.array(new\_x\_train)

new\_y\_train = np.array(new\_y\_train)

new\_x\_train = np.reshape(new\_x\_train, (new\_x\_train.shape[0], 1, new\_x\_train.shape[1]))

eb\_15\_model = models.Sequential()

# Fully-connected RNN where the output is to be fed back to input.

# inputs: A 3D tensor, with shape [batch, timesteps, feature].

eb\_15\_model.add(SimpleRNN(15, input\_shape=(1, elman\_step), activation="relu"))

eb\_15\_model.add(Dense(1))

eb\_15\_model.compile(loss = 'mse', optimizer='adam', metrics=['accuracy'],)

eb\_15\_model.fit(new\_x\_train, new\_y\_train, epochs = epochs, validation\_split = validation\_split)

# 3 РІРЅСѓС‚СЂС–С€РЅС–С… С€Р°СЂРё РїРѕ 5 РЅРµР№СЂРѕРЅС–РІ Сѓ РєРѕР¶РЅРѕРјСѓ

eb\_5\_model = models.Sequential()

eb\_5\_model.add(SimpleRNN(5, input\_shape=(1, elman\_step), activation="relu", return\_sequences=True))

eb\_5\_model.add(SimpleRNN(5, input\_shape=(1, elman\_step), activation="relu", return\_sequences=True))

eb\_5\_model.add(SimpleRNN(5, input\_shape=(1, elman\_step), activation="relu"))

eb\_5\_model.add(Dense(1))

eb\_5\_model.compile(loss = 'mse', optimizer='adam', metrics=['accuracy'],)

eb\_5\_model.fit(new\_x\_train, new\_y\_train, epochs = epochs, validation\_split = validation\_split)

plt.plot(train\_x, train\_y, label='Function')

plt.plot(train\_x[:-elman\_step], eb\_15\_model.predict(new\_x\_train), label='15')

plt.plot(train\_x[:-elman\_step], eb\_5\_model.predict(new\_x\_train), label='3 \* 5')

plt.legend()

plt.plot(train\_x, train\_y, label='Function')

plt.plot(train\_x, ffb\_10\_model.predict(train\_x), label='10 ffb')

plt.plot(train\_x, ffb\_20\_model.predict(train\_x), label='20 ffb')

plt.plot(train\_x, cfb\_20\_model.predict(train\_x), label='20 cfb')

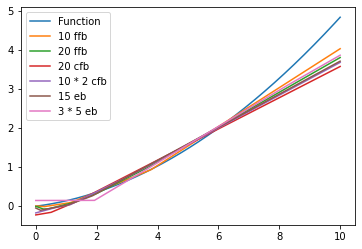
plt.plot(train\_x, cfb\_10\_model.predict(train\_x), label='10 \* 2 cfb')

plt.plot(train\_x[:-elman\_step], eb\_15\_model.predict(new\_x\_train), label='15 eb')

plt.plot(train\_x[:-elman\_step], eb\_5\_model.predict(new\_x\_train), label='3 \* 5 eb')

plt.legend()

## Результат роботи:



## Висновки:

Я дізнався більше інформації про базові типи нейронних мереж, чим вони відрізняються так як їх створити