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:
студент групи ІП-93
Домінський Валентин Олексійович

Перевірив:
Шимкович Володимир Миколайович

Київ 2022

Зміст:

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

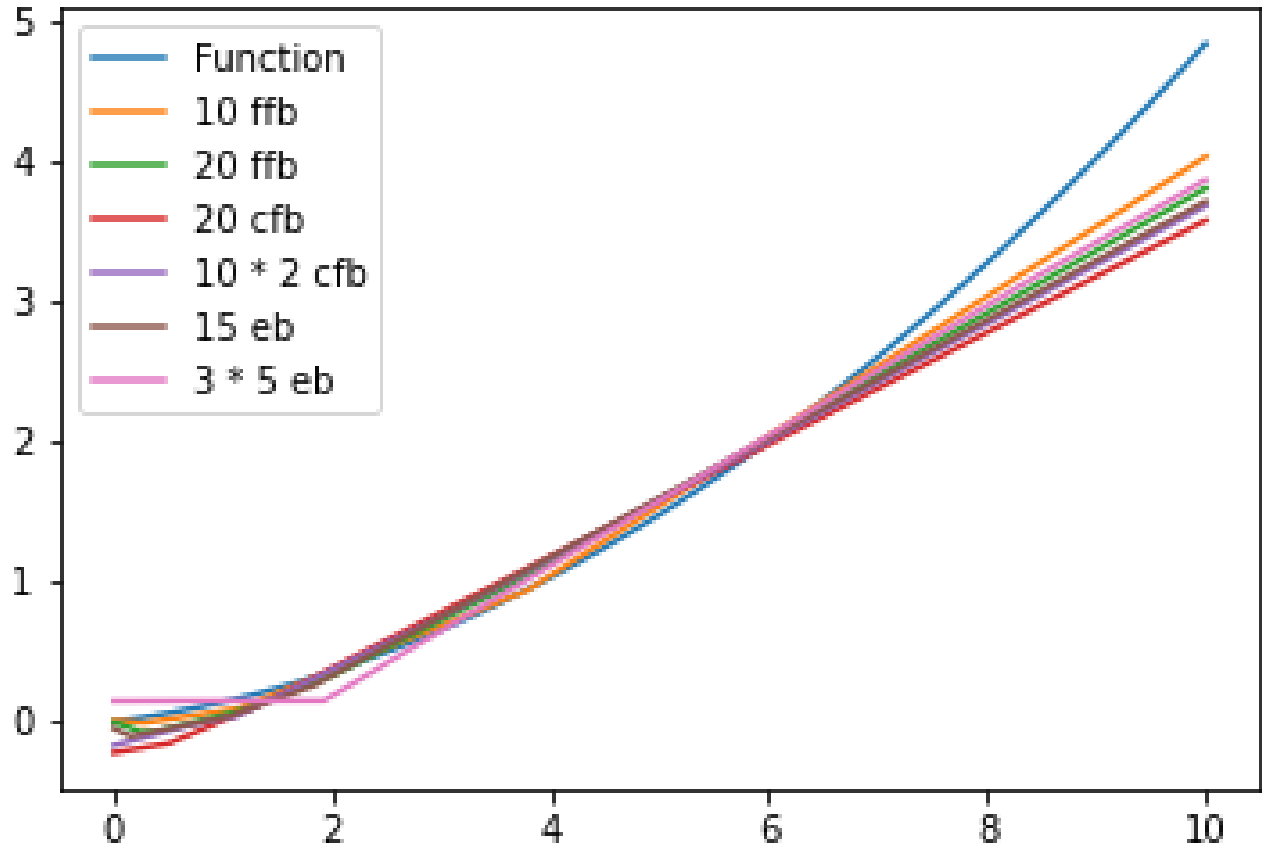
```
import tensorflow as tf

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

# PIPSCґC‚CЂC–C€PSC–P‰ C€PºCЂ P· 10 PSPµP‰CЂPsPSPºPjPë

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)

# PIPSCґC‚CЂC–C€PSC–P‰ C€PºCЂ P· 20 PSPµP‰CЂPsPSPºPjPë

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.

# PIPSCґC,CЂC-C€PSC-P℠ C€P°CЂ P· 20 PSPµP℠CЂPsPSP°PjPë

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 PIPSCґC,CЂC-C€PSC-C… C€P°CЂPë PïPs 10 PSPµP℠CЂPsPSC-PI Cґ PєPsP¶PSPsPjCґ

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 PIPSCґC,CЂC-C€PSC-P℠ C€P°CЂ P· 15 PSPµP℠CЂPsPSP°PjPë

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 PIPSCґC,CЂC–C€PSC–C… C€PºCЂPë PïPs 5 PSPµP℁CЂPsPSC–PI Cґ PєPsP¶PSPsPjCґ

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