Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №3**

По дисциплине «Модели решения задач в интеллектуальных системах»

Тема: «Рекуррентные нейронные сети»

**Выполнил:**

Студент 3 курса

Группы ИИ-21

Карагодин Д. Л.

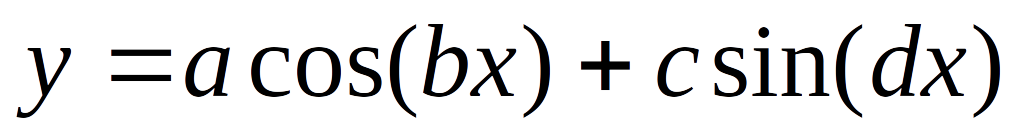
**Проверил:**

Туз И. С.

Брест 2024

**Цель:** необходимо реализовать и обучить рекуррентную нейронную сеть для прогнозирования периодической функции.

**Ход работы**



**Вариант 1**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № варианта | a | b | с | d |
| 1 | 0.1 | 0.1 | 0.05 | 0.1 |

**Код программы:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

def sigmoid(x):

    return 1 / (1 + np.exp(-x))

def get\_sequence\_data(df, seq\_len):

    X, Y = [], []

    nr\_records = len(df) - seq\_len

    sin\_wave = df

    for i in range(nr\_records - seq\_len):

        X.append(sin\_wave[i:i+seq\_len])

        Y.append(sin\_wave[i+seq\_len])

    return X, Y

def get\_test\_data(df, seq\_len, len\_test):

    X, Y = [], []

    nr\_records = len(df) - seq\_len

    sin\_wave = df

     for i in range(nr\_records - len\_test, nr\_records):

        X.append(sin\_wave[i:i+seq\_len])

        Y.append(sin\_wave[i+seq\_len])

    return X, Y

def list\_to\_array(X, Y):

    X = np.array(X)

    Y = np.array(Y)

    X = np.array(X)

    X = np.expand\_dims(X, axis=2)

    Y = np.array(Y)

    Y = np.expand\_dims(Y, axis=1)

    return X, Y

def forward(x, y, prev\_s):

    layers = []

    for t in range(T):

        new\_input = np.zeros(x.shape)

        new\_input[t] = x[t]

        m = np.dot(U, new\_input)

        n = np.dot(W, prev\_s)

        o = n + m

        s = sigmoid(o)

        p = np.dot(V, s)

        layers.append({'s':s, 'prev\_s':prev\_s})

        prev\_s = s

    return (m, n, o, s, p), layers

def clip\_min\_max(dU, dV, dW):

    if dU.max() > max\_clip\_value:

        dU[dU > max\_clip\_value] = max\_clip\_value

    if dV.max() > max\_clip\_value:

        dV[dV > max\_clip\_value] = max\_clip\_value

    if dW.max() > max\_clip\_value:

        dW[dW > max\_clip\_value] = max\_clip\_value

    if dU.min() < min\_clip\_value:

        dU[dU < min\_clip\_value] = min\_clip\_value

    if dV.min() < min\_clip\_value:

        dV[dV < min\_clip\_value] = min\_clip\_value

    if dW.min() < min\_clip\_value:

        dW[dW < min\_clip\_value] = min\_clip\_value

    return dU, dV, dW

def backward(alpha, y, layers):

    m, n, o, s, p = alpha

    dU = np.zeros(U.shape)

    dV = np.zeros(V.shape)

    dW = np.zeros(W.shape)

    dU\_t = np.zeros(U.shape)

    dV\_t = np.zeros(V.shape)

    dW\_t = np.zeros(W.shape)

    dU\_i = np.zeros(U.shape)

    dW\_i = np.zeros(W.shape)

    dp = (p - y)

    for t in range(T):

        dV\_t = np.dot(dp, np.transpose(layers[t]['s']))

        dsv = np.dot(np.transpose(V), dp)

        ds = dsv

        do = sigmoid(o) \* (1 - sigmoid(o)) \* ds

        dn = do \* np.ones\_like(n)

        dprev\_s = np.dot(np.transpose(W), dn)

        for j in range(t-1, max(-1, t-bptt\_truncate-1), -1):

            dV\_i = np.dot(dp, np.transpose(layers[j]['s']))

            ds = dsv + dprev\_s

            do = sigmoid(o) \* (1 - sigmoid(o)) \* ds

            dn = do \* np.ones\_like(n)

            dm = do \* np.ones\_like(m)

            dW\_i = np.dot(W, layers[t]['prev\_s'])

            dprev\_s = np.dot(np.transpose(W), dn)

            new\_input = np.zeros(x.shape)

            new\_input[t] = x[t]

            dU\_i = np.dot(U, new\_input)

            dx = np.dot(np.transpose(U), dm)

            dU\_t += dU\_i

            dV\_t += dV\_i

            dW\_t += dW\_i

        dU += dU\_t

        dV += dV\_t

        dW += dW\_t

    return clip\_min\_max(dU, dV, dW)

def optimize(alpha, grads):

    dU, dV, dW = grads

    U, V, W = alpha

    U -= learning\_rate \* dU

    V -= learning\_rate \* dV

    W -= learning\_rate \* dW

    return U, V, W

def val\_loss\_fn(alpha):

    m, n, o, s, p = alpha

    val\_loss = 0.0

    for i in range(y\_test.shape[0]):

        x, y = X\_test[i], y\_test[i]

        prev\_s = np.zeros((hidden\_dim, 1))

        alpha = forward(x, y, prev\_s)

        loss\_per\_record = (y - p)\*\*2 / 2

        val\_loss += loss\_per\_record

    return val\_loss / float(len\_data)

def loss\_fn(alpha, y):

    m, n, o, s, p = alpha

    return (y - p)\*\*2 / 2

df = pd.read\_csv('function\_values.csv', delimiter=',', nrows = 1000)

sin\_wave = (df.to\_numpy()).reshape(len(df))

seq\_len = T =  80

X\_train, y\_train = get\_sequence\_data(sin\_wave[:len(sin\_wave)], seq\_len)

X\_train, y\_train = list\_to\_array(X\_train, y\_train)

X\_test, y\_test = get\_test\_data(sin\_wave[:len(sin\_wave)], seq\_len, 400)

X\_test, y\_test = list\_to\_array(X\_test, y\_test)

len\_data = X\_train.shape[0]

learning\_rate = 0.0008

epochs = 10

bptt\_truncate = 6

min\_clip\_value = -0.4

max\_clip\_value = 2

hidden\_dim = 11

output\_dim = 1

U = np.random.uniform(0, 1, (hidden\_dim, T))

W = np.random.uniform(0, 1, (hidden\_dim, hidden\_dim))

V = np.random.uniform(0, 1, (output\_dim, hidden\_dim))

print("Training...")

for epoch in range(epochs):

    loss = 0.0

    for i in range(len\_data):

        # initialize

        x, y = X\_train[i], y\_train[i]

        prev\_s = np.zeros((hidden\_dim, 1))

        # forward pass

        alpha, layers = forward(x, y, prev\_s)

        # loss

        loss += loss\_fn(alpha, y)

        # backward pass

        grads = backward(alpha, y, layers)

        # update

        U, V, W = optimize((U, V, W), grads)

        if((len\_data - (i + 1))%100==0): print("data left: {}".format(len\_data - (i + 1)))

    # loss

    loss = loss / float(len\_data)

    # value loss

    val\_loss = val\_loss\_fn(alpha)

    print('Epoch:{:3d}, Loss:{:12.4f}, Val\_Loss:{:12.4f}'.format(epoch + 1, loss[0][0], val\_loss[0][0]))

#test

preds = []

for i in range(y\_test.shape[0]):

    x, y = X\_test[i], y\_test[i]

    prev\_s = np.zeros((hidden\_dim, 1))

    # Forward pass

    for t in range(T):

        mulu = np.dot(U, x)

        mulw = np.dot(W, prev\_s)

        add = mulw + mulu

        s = sigmoid(add)

        mulv = np.dot(V, s)

        prev\_s = s

    preds.append(mulv)

preds = np.concatenate(preds).squeeze()

plt.figure(figsize=(12,8))

plt.plot(preds, 'b-', label='Predicted')

plt.plot(y\_test.squeeze(), 'r-', label='Expected')

plt.legend(loc="upper left")

plt.show()

**Результат программы:**

Training...

data left: 800

data left: 700

data left: 600

data left: 500

data left: 400

data left: 300

data left: 200

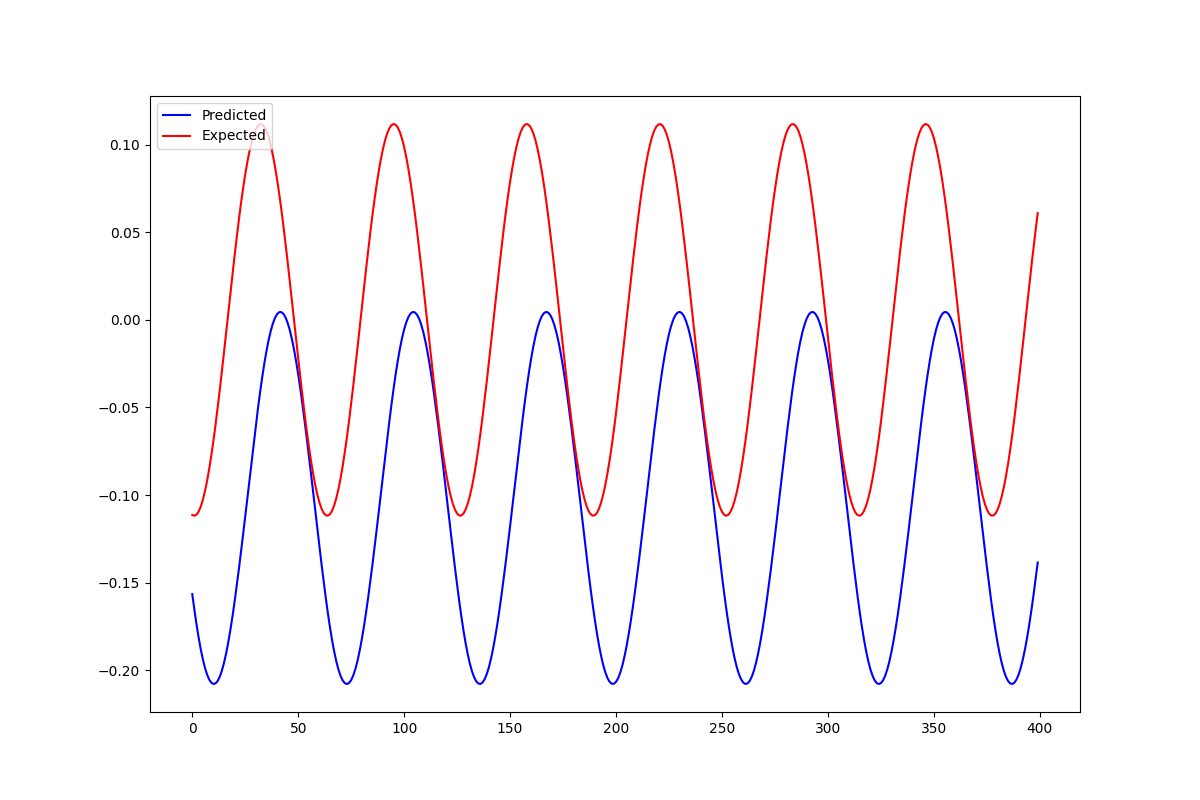
data left: 100

data left: 0

Epoch:  1, Loss:      4.3307, Val\_Loss:      0.0041

data left: 800

…



**Цель:** реализовал и обучил рекуррентную нейронную сеть для прогнозирования периодической функции.