Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №1

за 1 семестр

По дисциплине: «МиАПР»

Тема: «Линейная искусственная нейронная сеть. Правило Видроу-Хоффа»

Выполнил:

Студент 2 курса

Группы ПО-4(2)

Яковчик И.А.

Проверил:

Крощенко А.А.

2020

Лабораторная работа №1

Линейная искусственная нейронная сеть. Правило Видроу-Хоффа

Цель работы: изучить обучение и функционирование линейной ИНС при решении

задач прогнозирования.

Вариант 1

Задание:

Написать на любом ЯВУ программу моделирования прогнозирующей линейной

ИНС. Для тестирования использовать функцию

y = a\*sin(bx) + d

a = 1, b = 5 d = 0.1, кол-во входов ИНС = 3.

Обучение и прогнозирование производить на 30 и 15 значениях соответственно

табулируя функцию с шагом 0.1. Скорость обучения выбирается студентом

самостоятельно, для чего моделирование проводится несколько раз для разных .

Результаты оцениваются по двум критериям - скорости обучения и минимальной

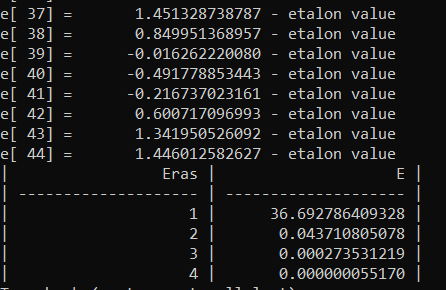
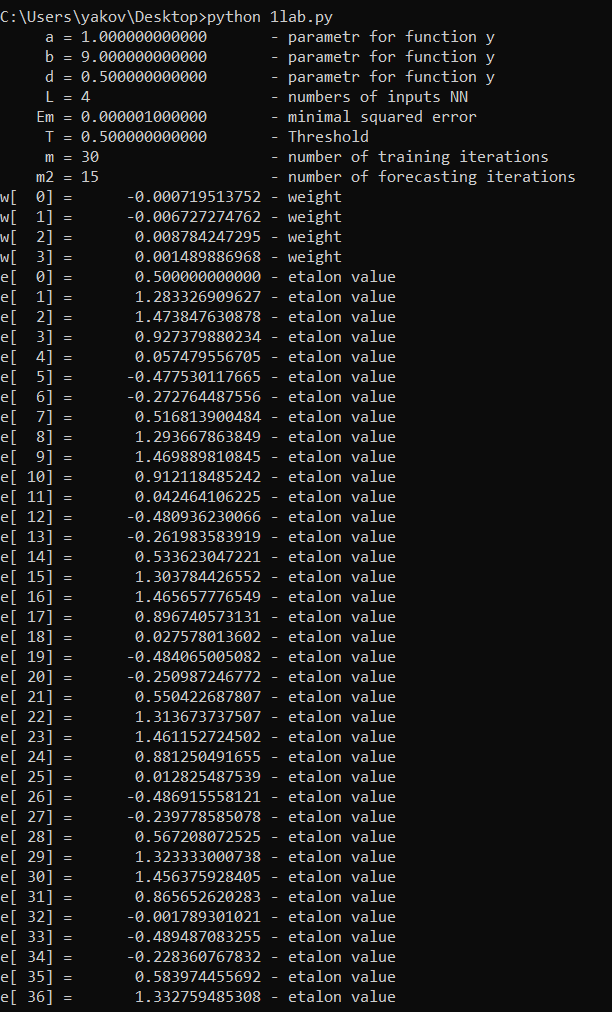
достигнутой ошибке. Необходимо заметить, что эти критерии в общем случае

являются взаимоисключающими, и оптимальные значения для каждого критерия

достигаются при разных a.

Код программы:

|  |
| --- |
| import random |
|  | import math |
|  | import matplotlib.pyplot as plt |
|  |  |
|  | class lab(): |
|  | def \_\_init\_\_(self, a, b, d, L, Em, T, m, m2): |
|  | self.a = a |
|  | print("%6s = %-20.12f - parametr for function y" % ("a", self.a)) |
|  |  |
|  | self.b = b |
|  | print("%6s = %-20.12f - parametr for function y" % ("b", self.b)) |
|  |  |
|  | self.d = d |
|  | print("%6s = %-20.12f - parametr for function y" % ("d", self.d)) |
|  |  |
|  | self.L = L |
|  | print("%6s = %-20d - numbers of inputs NN" % ("L", self.L)) |
|  |  |
|  | self.Em = Em |
|  | print("%6s = %-20.12f - minimal squared error" % ("Em", self.Em)) |
|  |  |
|  | self.T = T |
|  | print("%6s = %-20.12f - Threshold" % ("T", self.T)) |
|  |  |
|  | self.m = m |
|  | print("%6s = %-20d - number of training iterations" % ("m", self.m)) |
|  |  |
|  | self.m2 = m2 |
|  | print("%6s = %-20d - number of forecasting iterations" % ("m2", self.m2)) |
|  |  |
|  | def generate\_w(self, left\_point, right\_point): |
|  | self.w = [] |
|  | for i in range(self.L): |
|  | self.w.append(random.random() \* right\_point - left\_point) |
|  |  |
|  | def print\_w(self): |
|  | for i in range(self.L): |
|  | print("w[%3d] = %20.12f - weight" % (i, self.w[i])) |
|  |  |
|  | def generate\_e(self, step): |
|  | self.e = [] |
|  | for i in range(self.m + self.m2): |
|  | x = step \* i |
|  | result = self.a \* math.sin( self.b \* x ) + self.d |
|  | self.e.append(result) |
|  |  |
|  | def print\_e(self): |
|  | for i in range(self.m + self.m2): |
|  | print("e[%3d] = %20.12f - etalon value" % (i, self.e[i])) |
|  |  |
|  | def WidrowHoffAlgorithm\_constAlpha(self, alpha): |
|  | print("| %20s | %20s |" % ("Eras", "E")) |
|  | print("| %16s | %16s |" % ("--------------------", "--------------------")) |
|  |  |
|  | eras = 0 |
|  | valueXforGraph = [] |
|  | valueYforGraph = [] |
|  | while 1: |
|  | E = 0 |
|  | for i in range(self.m - self.L): |
|  | y1 = 0 |
|  | for j in range(self.L): |
|  | y1 += self.w[j] \* self.e[i + j] |
|  | y1 -= self.T |
|  | for j in range(self.L): |
|  | self.w[j] -= alpha \* (y1 - self.e[i + self.L]) \* self.e[i + j] |
|  | self.T += alpha \* (y1 - self.e[i + self.L]) |
|  | E += 0.5 \* math.pow( (y1 - self.e[i + self.L]), 2 ) |
|  |  |
|  | eras += 1 |
|  | print("| %20d | %20.12f |" % (eras, E)) |
|  | valueXforGraph.append(eras) |
|  | valueYforGraph.append(E) |
|  |  |
|  | if E < self.Em: |
|  | break |
|  | plt.plot(valueXforGraph, valueYforGraph, 'Db', label="Contantly alpha") |
|  |  |
|  | def printResult(self): |
|  | def print\_headTable(): |
|  | print("| %20s | %20s | %20s | %20s |" % ( |
|  | "y[]", |
|  | "Эталонное значение", |
|  | "Полученное значение", |
|  | "Отклонение" |
|  | )) |
|  | print("| %16s | %16s | %16s | %16s |" % ( |
|  | "--------------------", |
|  | "--------------------", |
|  | "--------------------", |
|  | "--------------------" |
|  | )) |
|  |  |
|  | trainingSample = [] |
|  | print("Result learning") |
|  | print\_headTable() |
|  | for i in range(self.m): |
|  | trainingSample.append(0) |
|  | for j in range(self.L): |
|  | trainingSample[i] += self.w[j] \* self.e[j + i] |
|  | trainingSample[i] -= self.T |
|  | print("| %20d | %20.12f | %20.12f | %20.12f |" % ( |
|  | i, |
|  | self.e[i + self.L], |
|  | trainingSample[i], |
|  | math.pow(self.e[i + self.L] - trainingSample[i], 2) |
|  | )) |
|  |  |
|  | print("Results forecasting") |
|  | print\_headTable() |
|  | for i in range(self.m2): |
|  | trainingSample.append(0) |
|  | for j in range(self.L): |
|  | trainingSample[i + self.m] += self.w[j] \* self.e[self.m - self.L + j + i] |
|  | trainingSample[i + self.m] -= self.T |
|  | print("| %20d | %20.12f | %20.12f | %20.12f |" % ( |
|  | i + self.m, |
|  | self.e[i + self.m], |
|  | trainingSample[i + self.m], |
|  | math.pow(self.e[i + self.m] - trainingSample[i + self.m], 2) |
|  | )) |
|  |  |
|  | """Main""" |
|  |  |
|  | x = lab( |
|  | 1, # a argument for function y |
|  | 9, # b argument for function y |
|  | 0.5, # d argument for function y |
|  | 4, # L number of inputs NN |
|  | 1e-6, # Em argument for algorithm |
|  | 0.5, # T argument for algorithm |
|  | 30, # m number of operations for training results |
|  | 15, # m2 numper of operation for forecasting results |
|  | ) |
|  |  |
|  | x.generate\_w(0.01, 0.02) # arguments (left\_point, right\_point) |
|  | x.print\_w() |
|  |  |
|  | x.generate\_e(0.1) # argument (step) for y |
|  | x.print\_e() |
|  |  |
|  | x.WidrowHoffAlgorithm\_constAlpha(0.5) # argument (alpha) |
|  | x.printResult() |
|  |  |
|  | plt.title("Error change graph") # Python write title in graph |
|  | plt.legend() # Python write legend in graph |
|  | plt.show() # Python open new windows and show graph |



Вывод: В ходе выполнения данной работы я спроектировал линейную искусственную нейронную сеть, используя правило обучения Видроу-Хоффа.