**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

**Лабораторные работы 1-8**

по курсу «Нейроинформатика».

Студент: Бахарев А.Т.

Группа: М8О-406Б

Вариант: 3

Оценка:

Москва, 2019

**Содержание**

*Лабораторная работа №1.*

Персептроны. Процедура обучения Розенблатта . . . . . . . . . . . . . . . 3

**Постановка задачи.**

Целью работы является исследование свойств персептрона Розенблатта и его применение для решения задачи распознавания образов.

**Основные этапы работы.**

1. Для первой обучающей выборки построить и обучить сеть, которая будет правильно относить точки к двум классам. Отобразить дискриминантную линию и проверить качество обучения.

2. Изменить обучающее множество так, чтобы классы стали линейно неразделимыми. Проверить возможности обучения по правилу Розенблатта.

3. Для второй обучающей выборки построить и обучить сеть, которая будет правильно относить точки к четырем классам. Отобразить дискриминантную линию и проверить качество обучения.

**Описание алгоритма.**

Принцип обучения перцептрона по Розенблатту следующий: Инициализируем веса случайными значениями. Затем начинаем процесс обучения. Берем точку и получаем предсказанный лейбл. Сравниваем его с тем, который нам нужен получить. В данной лабораторной работе функция активации — функция Хевисайда. Если ref\_label == pred\_label, то продолжаем обучение со следующей точкой. Если нет, то смотрим, какой именно лейбл вернула активационная функция. Если 0, а должно быть 1, то увеличиваем веса перцептрона пропорционально координатам точки и скорости обучения, которая задается пользователем. Если получили 1, а должно быть 1, то вычитаем. После прогона всей обучающей выборки будем иметь веса, которые разделяют классы. Будем повторять обучение по всей выборке заданное количество раз(эпох) или пока сеть не начнет выдавать правильные результаты

**Ход работы.**

Для начала надо реализовать класс нейрона. Он будет предсказывать принадлежность к тому или иному классу. Затем реализуем класс нейронной сети, которая будет содержать несколько перцептронов. Процесс обучения следующий – каждому перцептрону передается точка и соответствующий ему класс. Будет происходить смена весов. Во время предсказания класса нужно, чтобы каждый перцептрон предсказал принадлежность к своему классу. Затем все предсказанные лейблы(это последовательность 0 и 1) переводится в десятичную систему счисления и возвращается пользователю. Это будет предсказанный класс.

**Исходный код.**

1. **import** math
2. **import** random
3. **import** matplotlib.pyplot as plt
5. **class** Neuron(object):
6. **def** \_\_init\_\_(self, feature\_num):
7. self.weights = [(random.randrange(1, 10)) / 100 **for** \_ **in** range(feature\_num + 1)]
9. **def** ActivationFunction(self, feature):
10. **if** len(feature) + 1 != len(self.weights):
11. exit(-1)
13. res = float(self.weights[0])
14. **for** i, w **in** zip(feature, self.weights[1:]):
15. res += i \* w
16. **return** 1 **if** res > 0 **else** 0
18. **def** Learn(self, point, label, learn\_speed):
19. pred\_label = self.ActivationFunction(point)
20. **if** pred\_label == label:
21. **return**
22. **else**:
23. **if** pred\_label > label: # 1 0
24. self.weights[0] -= learn\_speed
25. **for** i **in** range(1, len(self.weights)):
26. self.weights[i] -= learn\_speed \* point[i - 1]
27. **else**: # 0 1
28. self.weights[0] += learn\_speed
29. **for** i **in** range(1, len(self.weights)):
30. self.weights[i] += learn\_speed \* point[i - 1]
32. **def** Predict(self, feature):
33. **if** len(feature) + 1 != len(self.weights):
34. **print**("Incorrect feature input")
35. exit(-1)
36. **return** self.ActivationFunction(feature)

39. **def** ShowPlot(self, points, labels):
40. x\_value = []
41. y\_value = []
43. plt.plot([-10, 10], [0, 0]) #x axis
45. plt.plot([0, 0], [-50, 50]) # y axis
47. i = -5
48. **while** i <=5: # draw line between classes
49. x\_value.append(i)
50. y\_value.append(-((self.weights[0] + self.weights[1] \* i) / self.weights[2]))
51. i += 0.01
52. plt.plot(x\_value, y\_value)
54. #draw classes
55. plt.scatter([p[0] **for** p **in** points], [p[1] **for** p **in** points], c=[c **for** c **in** labels])
57. plt.show()
59. **class** NeuralNetwork(object):
60. **def** \_\_init\_\_(self, neuron\_count, feature\_num, learn\_speed):
61. self.neurons = [Neuron(feature\_num) **for** i **in** range(neuron\_count)]
62. self.learn\_speed = learn\_speed
64. **def** Train(self, points, labels, epochs=50):
65. **for** \_ **in** range(epochs):
66. **for** point, label **in** zip(points, labels):
67. **for** neuron, label\_val **in** zip(self.neurons, label):
68. neuron.Learn(point, label\_val, self.learn\_speed)
70. **def** Predict(self, point):
71. result = []
72. **for** neuron **in** self.neurons:
73. result.append(neuron.Predict(point))
74. **return** result
76. **def** ShowPlot(self, points, labels):
77. plt.plot([-10, 10], [0, 0]) #x axis
78. plt.plot([0, 0], [-50, 50]) # y axis
80. **for** neuron **in** self.neurons:
81. x\_value = []
82. y\_value = []
83. i = -5
84. **while** i <=5: # draw line between classes
85. x\_value.append(i)
86. y\_value.append(-((neuron.weights[0] + neuron.weights[1] \* i) / neuron.weights[2]))
87. i += 0.01
88. plt.plot(x\_value, y\_value)
90. #draw classes
91. **def** get\_color(label):
92. color = 0
93. **for** i **in** range(0, len(label)):
94. color += (2 \*\* i) \* label[i]
95. **return** color
97. plt.scatter([p[0] **for** p **in** points], [p[1] **for** p **in** points], c=[get\_color(l) **for** l **in** labels])
99. plt.show()
101. **def** ReadPoints(file\_name):
102. points = []
103. with open(file\_name, encoding='utf-8') as input\_file:
104. X = input\_file.readline().strip("\n").split(" ")
105. Y = input\_file.readline().strip("\n").split(" ")
107. **if** len(X) != len(Y):
108. **print**("Incorrect input data. Exit")
109. exit(-1)
110. **for** i **in** range(len(X)):
111. points.append([float(X[i]), float(Y[i])])
112. **return** points
114. **def** ReadLabels(file\_name):
115. with open(file\_name, encoding='utf-8') as input\_class\_file:
116. data = input\_class\_file.readlines()
117. **for** i **in** range(len(data)):
118. data[i] = data[i].strip("\n").split(" ")
120. labels\_num = len(data[0])
121. labels = [[] **for** \_ **in** range(labels\_num)]
123. **for** i **in** range(len(data)):
124. **for** j **in** range(labels\_num):
125. labels[j].append(int(data[i][j]))
126. **return** labels
128. **def** GenRandomPoints(num):
129. points = []
130. **for** \_ **in** range(num):
131. random.seed()
132. x = random.randrange(-5, 5)
133. y = random.randrange(-5, 5)
134. points.append([x, y])
135. **return** points

138. **def** FirstTask():
139. points = ReadPoints("input\_1.txt")
140. classes = ReadLabels("classes\_1.txt")
142. nn = NeuralNetwork(1, 2, 0.01)
143. nn.Train(points, classes, epochs=50)
144. **for** point, label **in** zip(points, classes):
145. res = nn.Predict(point)
146. **print**("point = {}, label = {}, pred\_label = {}".format(point, label, res))
148. # nn.ShowPlot(points, classes)
150. **print**("----------------------------")
152. validation\_points = GenRandomPoints(3)
153. val\_labels = []
154. **for** val\_point **in** validation\_points:
155. res = nn.Predict(val\_point)
156. **print**("point = {}, pred\_label = {}".format(val\_point, res))
157. val\_labels.append(res)
159. # nn.ShowPlot(validation\_points, val\_labels)
161. **print**("----------------------------")
163. random.shuffle(points)
164. random.shuffle(classes)
165. pred\_labels = []
166. **for** point, label **in** zip(points, classes):
167. res = nn.Predict(point)
168. **print**("point = {}, label = {}, pred\_label = {}".format(point, label, res))
169. pred\_labels.append(res)
170. nn.ShowPlot(points, classes)
172. **def** SecondTask():
173. points = ReadPoints("input\_2.txt")
174. classes = ReadLabels("classes\_2.txt")
176. nn = NeuralNetwork(2, 2, 0.01)
177. nn.Train(points, classes, epochs=50)
178. **for** point, label **in** zip(points, classes):
179. res = nn.Predict(point)
180. **print**("point = {}, label = {}, pred\_label = {}".format(point, label, res))
181. nn.ShowPlot(points, classes)
183. **print**("----------------------------")
185. validation\_points = GenRandomPoints(5)
186. val\_labels = []
187. **for** val\_point **in** validation\_points:
188. res = nn.Predict(val\_point)
189. **print**("point = {}, pred\_label = {}".format(val\_point, res))
190. val\_labels.append(res)
191. nn.ShowPlot(validation\_points, val\_labels)

194. **def** main():
196. FirstTask()
197. SecondTask()

200. **if** \_\_name\_\_ == "\_\_main\_\_":
201. main()

**Входные данные и результаты.**

Вариант 3.

|  |  |
| --- | --- |
| **Обучение** | **Предсказание** |
| **1.**  -2.8 4 3.4 0.8 1.6 1.2  -0.1 0.7 2.3 -2.6 -4.2 1.6  1 0 0 1 1 1  **2.**  3 3 2  2 -4 -4  0 1 1  **3.**  0.8 1.2 1.6 3.4 -2.8 4.0  -2.6 1.6 -4.2 2.3 -0.1 0.7  1 0 1 1 1 0 | **1.**  **2.**    **3.** |
| **1.**  -1.8 2.1 2.2 1.7 -0.7 3.1 -2.6 -1.3  -0.5 3.8 -4.9 -0.7 -3.9 -1.8 -1.6 0.4  0 1 1 1 0 1 0 0  1 0 0 0 1 0 1 0  **2.**  -4 4 1 -1 -4  0 0 1 -3 -3  0 1 1 0 0  1 0 0 1 1 | **1.**  2. |

**Выводы:**

Данная лабораторная работа познакомила меня с моделью перцептрона и простейшим алгоритмом его обучения. Я разобрался в устройстве однослойной нейронной сети и научился ее обучать и получать предсказанные значения.