Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ *Робототехники и комплексной автоматизации*

КАФЕДРА *Системы автоматизированного проектирования (РК-6)*

ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ

по дисциплине:

Введение в искусственный интеллект

Студент	Абидоков Рашид Ширамбиевич
Группа	РК6-11М
Вариант	1-2
Тема лабораторной работы	Программирование искусственного нейрона

Студент _____ **Абидоков Р. Ш.**
подпись, дата *фамилия, и.о.*

Преподаватель _____ **Федорук В. Г.**
подпись, дата *фамилия, и.о.*

Оценка _____

Москва, 2020 г.

Оглавление

Задание на лабораторную работу

Цель лабораторной работы – создание программы, реализующей искусственный нейрон; разработка процедуры обучения нейрона; использование полученных результатов для решения тестовых задач классификации и аппроксимации

Тип нейрона – персептрон

Вариант обучающих данных на Рис.1

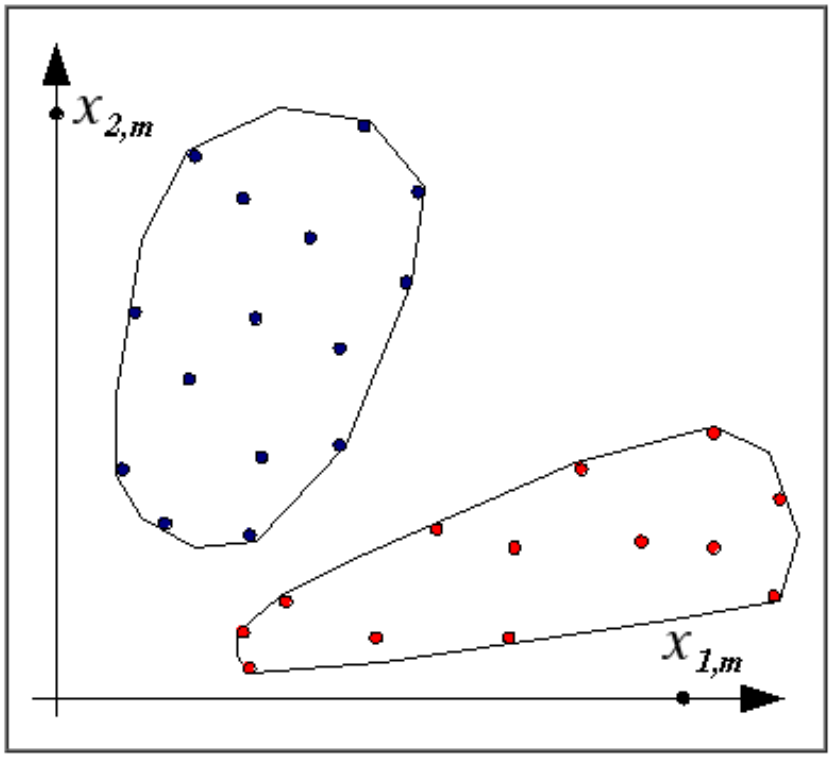

Рис 1.

Описание входных данных

Были взяты точки, распределенные на плоскости в соответствии с Рис. 1. Координаты точек и их класс приведены в таблице ниже. Из них был сформирован входной файл points.txt

Табл. 1

i	x_i	y_i	класс
1	7	2	0
2	3	10	1

3	12	3	0
4	6	5	1
5	14	0	0
6	6	9	1
7	17	-2	0
8	8	13	1
9	16	10	1

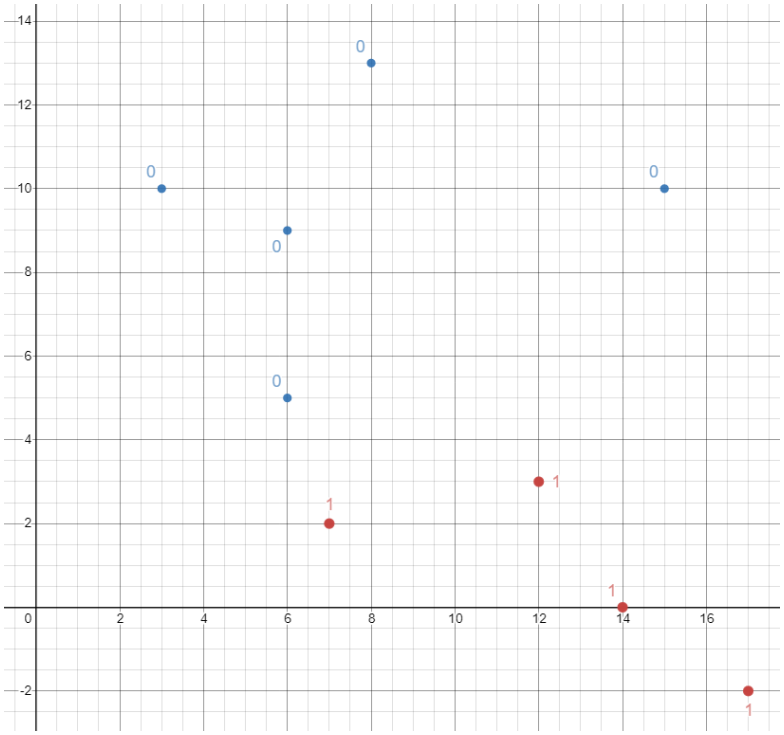

Рис 2. Принятое распределение точек

Описание используемой модели

Перцептрон – это нейрон со ступенчатой функцией активации, т.е.

$$y_i = f(u_i) = \begin{cases} 1, & u_i \geq 0 \\ 0, & u_i < 0 \end{cases}$$

Где $u_i = \sum_{j=1}^{m} w_j x_{ij}$ – взвешенная сумма входных сигналов, i – номер объекта обучающей выборки, j – номер параметра, m – количество параметров

Обучение нейрона сводится к подбору весов w_j путем минимизации среднеквадратичной ошибки на обучающей выборке

$$Q = \frac{1}{2}\sum_{i=1}^{n}(y_i - d_i)^2$$

Где $n -$ количество объектов обучающей выборки, $d_i -$ класс i-го объекта

В силу дискретности функции активации и, как следствие, невозможности брать производные функции ошибки, невозможно использование методов оптимизации выше нулевого порядка, таких как, например, градиентные методы. Поэтому обучение производилось с помощью правила персептрона:

1. Выбираются начальные значения весов $w_j = w_{j0}$
2. Для каждой обучающей пары $(x_{i1}, x_{i2}), d_i$ выполняется ряд циклов уточнений входных весов:

   Вычисляется y_i

   Если $(d_i - y_i) = 1$, то $w_j(t+1) = w_j(t) + \eta x_{ij}$

   Если $(d_i - y_i) = -1$, то $w_j(t+1) = w_j(t) - \eta x_{ij}$

   Если $(d_i - y_i) = 0$ или $t = t_{max}$, то цикл прекращается
3. Повторяем циклические проходы по обучающей выборке до тех пор, пока решение не сойдется либо пока не будет превышено максимальное количество итераций

Описание программной реализации

Программная реализация выполнена на языке C++ с использованием компилятора gcc. Считываются координаты и классы точек из входного txt-файла, задается начальное приближение весов, нейрон обучается по алгоритму, описанному выше, после чего выводятся веса после обучения и метки классов d_i и y_i

Листинг 1. Функции обучения класса нейрона

```
unsigned perceptron::fit_one(const std::vector<std::vector<double> >& X_train,
                             const std::vector<bool>& y_train, size_t idx, unsigned max_iter) {
  unsigned count = 0;
  double f_coeff = 0.5;          // Коэффициент обучения
  double y_true = y_train[idx];
  while (count < max_iter) {
    double y_pred = this->out(X_train[idx]);
    if (y_true == y_pred) {
      break;
    }
    // Отдельно поляризацию
    weights_[0] += f_coeff * (y_true - y_pred);
    // И остальное
    for (size_t i = 1; i < N_inputs_; i++) {
      weights_[i] += f_coeff * (y_true - y_pred) * X_train[idx][i - 1];
    }
    count++;
  }
  return count;
};

unsigned perceptron::fit(const std::vector<std::vector<double> >& X_train,
                         const std::vector<bool>& y_train, unsigned max_iter) {
  unsigned total_count = 0;
  for (unsigned iter = 0; iter < max_iter; iter++) {
    for (size_t i = 0; i < X_train.size(); i++) {
      total_count += this->fit_one(X_train, y_train, i, 10);
    }
  }
  return total_count;
}
```

Результаты обучения

Обучение производилось с двумя различными начальными значениями весов. В первом случае все веса принимались нулевыми – при таком начальном приближении решение сошлось за два шага

Листинг 2. Вывод программы при нулевом начальном приближении

```
C:\Study\Introduction-to-AI\lab3>bin\main.out
================
weights0:
---------
w0 : 0
w1 : 0
w2 : 0
================
weights after fit:
---------
w0 : 0
w1 : -2
w2 : 4
================
n_iters: 2
================
p0: y_true0| y_pred0
p1: y_true1| y_pred1
p2: y_true0| y_pred0
p3: y_true1| y_pred1
p4: y_true0| y_pred0
p5: y_true1| y_pred1
p6: y_true0| y_pred0
p7: y_true1| y_pred1
p8: y_true1| y_pred1
```

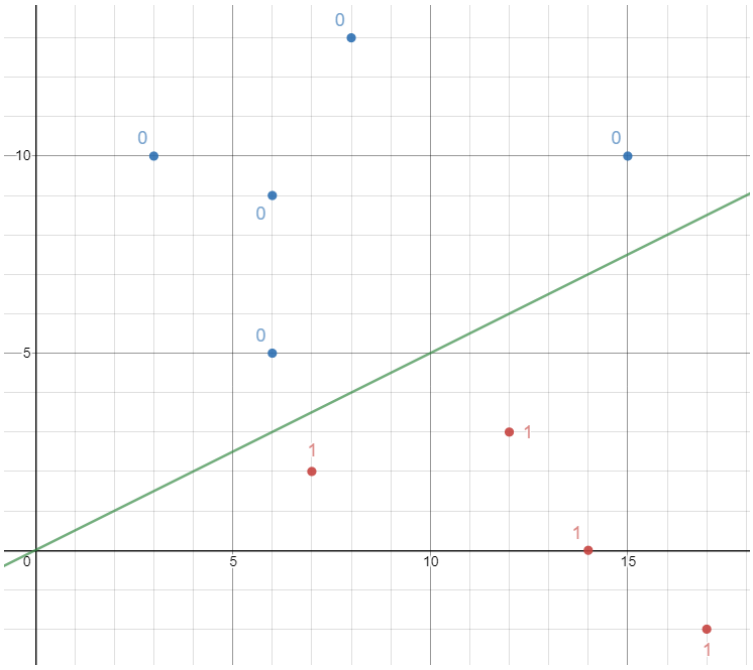

Рис 3. Прямая, соответствующая полученным весам

Табл. 2

N шага	0	1	2
w_0	0	-0.5	0
w_1	0	-3.5	-2
w_2	0	1	4

Во втором случае в качестве начальных весов были приняты значения (-60, -9.5, -3) – при таких весах разделяющая прямая выглядит так, как показано на Рис. 4. Т.е. приближение можно считать плохим

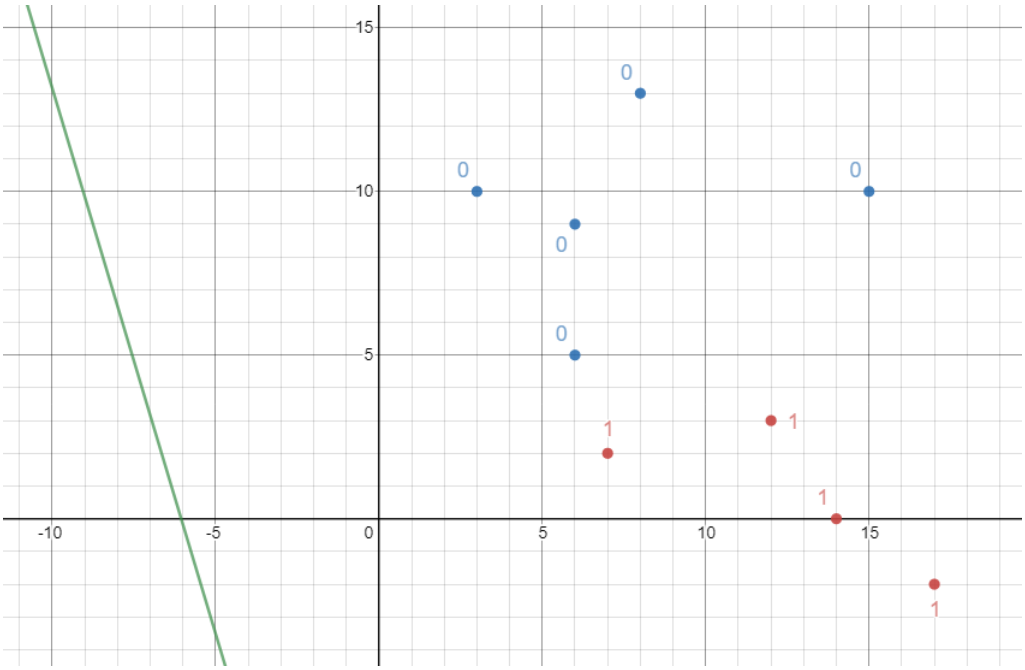

Рис 4. Прямая, соответствующая "плохому" нач. приближению

В этом случае решение сошлось за 4 итерации

Листинг 3. Вывод программы при плохом начальном приближении

```
C:\Study\Introduction-to-AI\lab3>bin\main.out
================
weights0:
---------
w0 : -60
w1 : -9.5
w2 : -3
================
weights after fit:
---------
w0 : -58
w1 : -2
w2 : 14.5
================
n_iters: 4
================
p0: y_true0| y_pred0
p1: y_true1| y_pred1
p2: y_true0| y_pred0
p3: y_true1| y_pred1
p4: y_true0| y_pred0
p5: y_true1| y_pred1
p6: y_true0| y_pred0
p7: y_true1| y_pred1
p8: y_true1| y_pred1
```

Табл. 3

N шага	0	1	2	3	4
w_0	-60	-59.5	-59	-58.5	-58
w_1	-9.5	-8	-6.5	-5	-2
w_2	-3	2	7	12	14.5

Итоговая разделяющая прямая выглядит аналогично Рис. 3