ЛАБОРАТОРНАЯ РАБОТА № 1

ОЗНАКОМЛЕНИЕ С ВИЗУАЛЬНОЙ СРЕДОЙ ПРОГРАММИРОВАНИЯ. ПЕРСЕПТРОН. ОБУЧЕНИЕ ПЕРСЕПТРОНА. РЕШЕНИЕ ПРОБЛЕМЫ КЛАССИФИКАЦИИ ВЕКТОРОВ

1.1 Цель роботы

Ознакомление с визуальной средой программирования на языке Python (на примере Spyder). Освоение методики создания персептрона и процедуры его настройки при помощи библиотеки Neurolab на языке программирования Python. Приобретение навыков решения задачи классификации векторов.

1.2 Методические указания по самостоятельной работе студентов

План работы:

1. Установка среды и библиотек (SciPy, Neurolab). Проверка работоспособности модулей numpy, matplotlib, neurolab.

2. Изучение визуальной среды по предлагаемым и встроенным примерам.

3. Повторение теоретического материала об однослойном персептроне, выполнение численных экспериментов.

4. Создание однослойного персептрона, выполнение предложенных и встроенных экспериментов.

5. Проведение контрольных экспериментов по вариантам.

Каждый пункт должен быть отражён в отчёте.

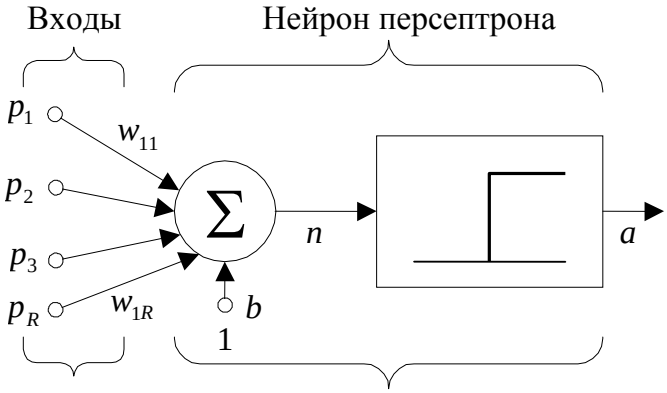
1.3. Создание нейронной сети

1.3.1 Архитектура персептрона

Простейшая из моделей персептронов – однослойный персептрон, - веса и сдвиг которого можно настроить так, чтоб решить задачу классификации входных векторов, что в дальнейшем позволяет решать сложные проблемы анализа коммуникационных соединений, распознавания образов и других задач классификации с относительно высоким быстродействием и гарантией правильного результата.

Нейрон персептрона. Нейрон, используемый в модели персептрона, имеет ступенчатую функцию активации HardLim с жёсткими ограничениями (рис. 2.1).

Каждый элемент входа персептрона взвешен соответствующим коэффициентом w_{ij}, а их сумма является входом функции активации. Нейрон персептрона возвращает 1, если вход функции активации $n > 0$, и 0, если $n < 0$.



$$a = \text{hardlim}\,(Wp + b)$$

Рисунок 2.1 – Нейрон персептрона

Функция активации с жёстким ограничением позволяет персептрону классифицировать входящие вектора, разделяя пространство входов на две области, как это показано на рис. 2.2 для персептрона с двумя входами и со сдвигом.

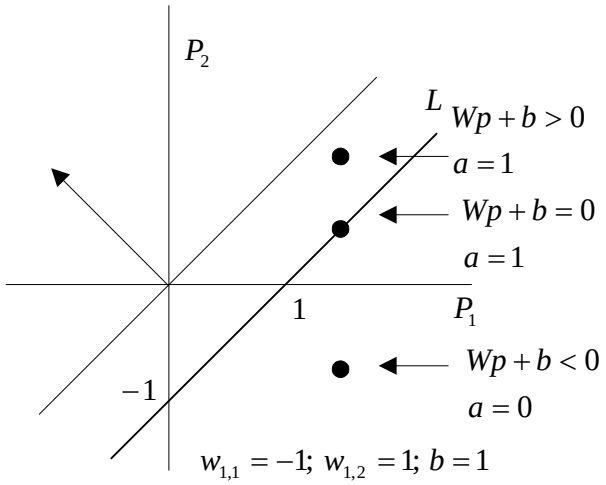


Рисунок 2.2 – Разделяющие поверхности

Пространство входов делится на две области разделяющей линией L, что для двухмерного случая задаётся уравнением

$$w^T p + b = 0. \qquad (2.1)$$

Эта линия перпендикулярна к вектору весов w и смещена на величину b. Векторы входа выше линии L соответствуют положительному потенциалу нейрона, а значит реакция нейрона для этих векторов будет равна 1; векторы же ниже линии L соответствуют выходу персептрона, равному 0. При изменении значения сдвига и весов линия L меняет своё положение. Персептрон без сдвига всегда формирует разделяющую линию, которая проходит через начало координат. В случае, если размерность входа превышает 2, границей является гиперплоскость.

Пример neurolab/examples/newp.py демонстрирует работающий персептрон.

Архитектура сети. Персептрон состоит из единственного слоя, включающего пять нейронов, как это показано на рис. 2.3. Веса w_{ij} – это коэффициенты передачи j-го входа к i-му нейрону. Уравнение однослойного персептрона имеет вид:

$$a = f(Wp + b). \tag{2.2}$$

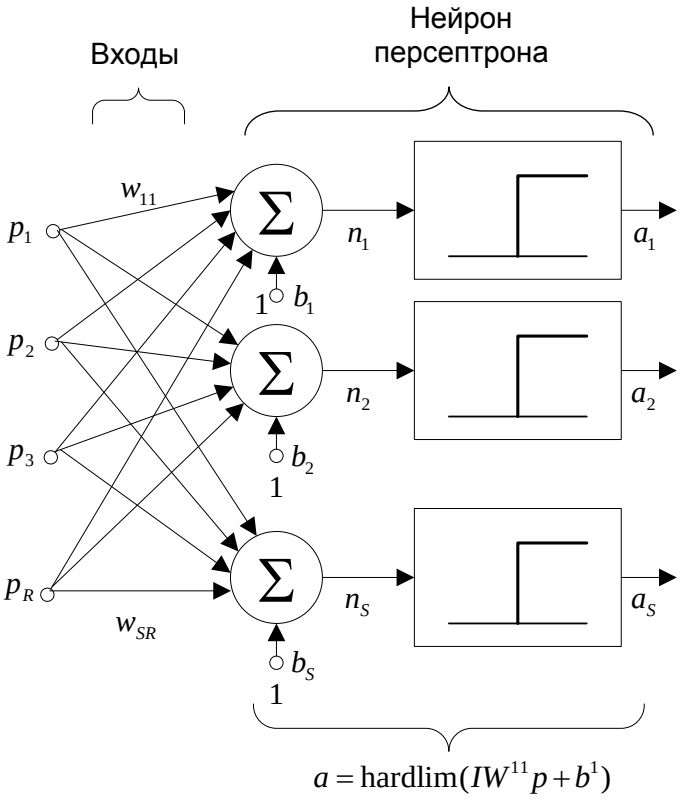


Рисунок 2.3 – Пример персептрона

1.3.2 Модель персептрона

Для создания однослойного персептрона предназначена функция newp(). Синтаксис функции:

`neurolab.net.`**`newp`***`(minmax, cn, transf = HardLim)`*

Её аргументы:

minmax – массив (list) размера ci*2 минимальных и максимальных значений для ci элементов входа;

cn – количество нейронов в слое,

transf – функция активации персептрона, по умолчанию HardLim.

Пример:

```
import neurolab as nl # импортируем библиотеку Neurolab под псевдонимом nl.
net = nl.net.newp([[0,2]],1) # создаём персептрон с одноэлементным входом и
одним нейроном. Диапазон значений входа - [0,2].
```

Ознакомимся с характеристиками созданной сети, установленными по умолчанию:

```
>>> net.__dict__
{'ci': 1,    #число входов
 'co': 1,    #число выходов
 'connect': [[-1], [ 0]], #схема соединения слоёв
 'errorf': <neurolab.error.SSE instance at 0x05C16918>, #функция критерия качества
 'inp': array([  0.]),    #текущее значение входов
 'inp_minmax': array([[  0., 2.]]),    #диапазон входных значений
 'layers': [<neurolab.layer.Perceptron object at 0x05D2FBB0>],  #перечисление слоёв
 'out': array([  0.]),    #текущее значение выходов
 'out_minmax': array([[  0., 1.]]),    #диапазон выходных значений
 'trainf': <neurolab.core.Trainer object at 0x057F2AD0>}  #функция активации
```

Моделирование персептрона

Рассмотрим однослойный персептрон с двухэлементным вектором входа, значения элементов которого изменяются в диапазоне от -2 до 2:

```
net = nl.net.newp([[-2,2],[-2,2]],1)    # создание персептрона net
```

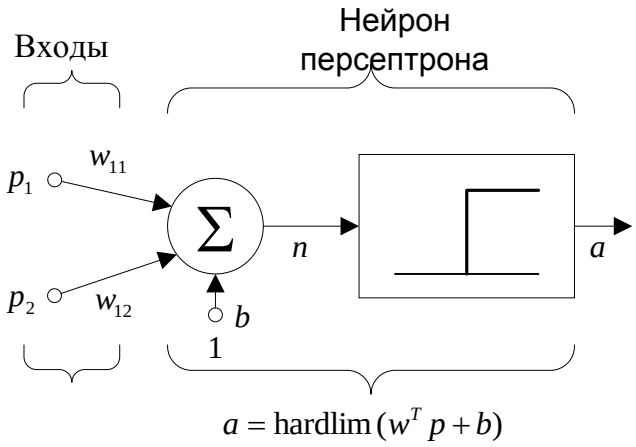


Рисунок 2.5 – Элементарный персептрон

Пакет Neurolab не предусматривает ручную настройку весов и сдвигов, только нормальное самообучение сети. Однако в образовательных целях попробуем узнать и изменить эти параметры вручную.

Введя в консоли «**nl.layer.Perceptron??**» мы выведем на экран весь код класса Perceptron. Из него можно понять, что веса и сдвиги хранятся в словаре (dict) формата {'w':(cn, ci), 'b': cn}. Сами же экземпляры класса Perceptron находятся в структуре сети в списке (list) layers. Значит, напрямую обратиться к весам первого (в нашем случае единственного) слоя можно по ключу **net.layers[0].np['w'],** а к сдвигу, соответственно, **net.layers[0].np['b'].** Действительно:

```
>>> print net.layers[0].np['w']
[[ 0.  0.]]
```

```
>>> print net.layers[0].np['b']
[ 0.]
```

Видно, что по умолчанию веса и сдвиг равны 0, чтобы установить желаемые значения необходимо использовать следующие операторы:

```
>>> net.layers[0].np['w'] = [[-1,1]]
>>> net.layers[0].np['b'] = [1]
```

В этом случае разделяющая линия имеет вид: $L: -p_1 + p_2 + 1 = 0$.

Структурная схема модели персептрона показана на рис. 2.4

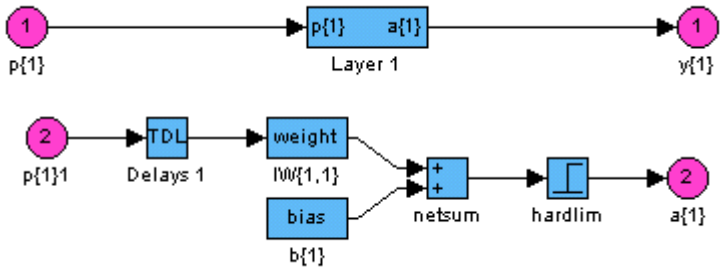


Рисунок 2.4 – Модель персептрона в нотации Simulink

Теперь посмотрим, как откликается сеть на входные векторы p_1 и p_2, расположенные по разные стороны разделяющей линии:

```
>>> p1 = [[1,1]]
>>> p2 = [[1,-1]]
>>> print net.sim(p1)   # симуляция сети с входным вектором p1
[[ 1.]]                 # ответ сети на вектор p1
>>> print net.sim(p2)   # симуляция сети с входным вектором p1
[[ 0.]]                 # ответ сети на вектор p2
```

Персептрон правильно классифицировал эти два вектора.

Заметим, что можно было бы ввести последовательность векторов в виде массива и получить результат тоже в виде массива:

```
 # Последовательность векторов в виде массива
>>> p3 = [[1,1],[1,-1]]
>>> print net.sim(p3)
[[ 1.]
 [ 0.]]
```

1.3.3 Процедура настройки параметров

Процесс обучения персептрона – это процедура настройки весов и сдвигов с целью уменьшить разницу между целевым и фактическим сигналами на его выходе, используя некоторое *правило настройки (обучения)*. Процедуры обучения делятся на два класса: *обучение с учителем* и *обучение без учителя.* Персептрон поддерживает только первый вариант.

При обучении с учителем задаётся множество примеров необходимого поведения сети, которое называется *обучающим множеством.*

$$\{p_1,t_1\},\{p_2,t_2\},\ldots,\{p_Q,t_Q\}.$$

Здесь $p_1, p_2, \ldots, p_Q$ – входы персептрона, а $t_1, t_2, \ldots, t_Q$ – необходимые выходы (реакции).

При подаче входов выходы персептрона сравниваются с целями. Правило обучения используется для настройки весов и сдвигов персептрона так, чтоб приблизить значение выхода к целевому значению. Алгоритмы, которые используют такие правила обучения, называются *алгоритмами обучения с учителем.*

При обучении без учителя веса и сдвиги меняются только в связи с изменениями входов сети. В этом случае целевые выходы в явном виде не задаются. Главное, что делает обучение без учителя привлекательным – это его самоорганизация, обусловленная, как правило, использованием обратных связей. Что же касается процесса настраивания параметров сети, то он организуется с использованием тех же процедур, что и при обучении с учителем. Большинство алгоритмов обучения без учителя используется при решении задач кластеризации данных, когда необходимо разделить входы на конечное количество заранее неизвестных классов.

Правила настройки

Настройка (обучение) персептрона происходит с использованием обучающего множества. Обозначим через p вектор входов персептрона, а через t – вектор соответствующих желаемых выходов. Цель обучения – уменьшить ошибку $e = a - t$, которая равна разнице между вектором выхода a и вектором цели t.

Целевой вектор t может включать только значения 0 и 1, т.к. персептрон с функцией активации HardLim может генерировать только такие значения.

При настройке параметров персептрону без сдвига и с одним нейроном возможны только три ситуации:

1) $e = a - t = 0$ - ответ сети правильный, веса изменять не нужно.

2) $e = t - 0 = 1$ - ответ 0, а должен быть 1. Значит, выход функции активации $w^T p$ отрицательный и его необходимо скорректировать в отрицательную сторону. Прибавим к вектору весов w вектор входа p, тогда выход изменится положительную величину, и после нескольких таких шагов вектор будет классифицирован правильно.

3) $e = t - a = -1$ - ответ 1, а должен быть 0. Ситуация обратна предыдущей, необходимо отнять вектор входа от вектора весов.

Эти ситуации можно связать с изменением вектора весов Δw следующим образом:

$$\Delta w = \begin{cases} 0, & \text{если } e = 0, \\ p, & \text{если } e = 1, \\ -p, & \text{если } e = -1. \end{cases} \qquad (2.3)$$

Все три случая можно описать одним соотношением:

$$\Delta w = (t - a)p = ep. \qquad (2.4)$$

Можно получить аналогичное выражения для изменения сдвига, если рассматривать сдвиг как вес единичного входа:

$$\Delta b = (t - a)1 = e. \tag{2.5}$$

В случае нескольких нейронов, эти соотношения обобщаются следующим образом:

$$\begin{cases} \Delta W = (t - a)p^T = ep^T, \\ \Delta b = (t - a) = e. \end{cases} \tag{2.6}$$

Тогда правило обучения персептрона можно записать в такой форме:

$$\begin{cases} W^{new} = W^{old} + ep^T, \\ b^{new} = b^{old} + e. \end{cases} \tag{2.7}$$

Описанные соотношения положены в основу алгоритма настройки персептрона, который реализован в Neurolab внутренней функцией layer.Perceptron._step(). Каждый раз при вызове этой функции происходит перенастройка параметров персептрона. Доказано, что если решение существует, то процесс обучения персептрона завершится за конечное число шагов.

Рассмотрим всё тот же пример персептрона с одним нейроном и двухэлементным вектором входа:

```
net = nl.net.newp([[-2,2],[-2,2]],1)
```

Выставим сдвиг b равным 0, а вектор весов w равным [1, -0.8]

```
net.layers[0].np['w'] = [[1,-0.8]]
net.layers[0].np['b'] = [0]
```

Обучающее множество зададим следующим образом:

```
p = [[1,2]]
t = [1]
```

Моделируя персептрон, рассчитаем выход и ошибку на первом шаге настройки:

```
>>> a = net.sim(p)
>>> print a
[[ 0.]]
>>> e = t - a
>>> print e
[[  1. ]]
```

Наконец, найдём необходимое изменение весов

```
>>> dw = e*p
>>> print dw
[[ 1.  2.]]
```

Значит новый вектор весов примет вид

```
>>> net.layers[0].np['w'] += dw
>>> print net.layers[0].np['w']
[[ 2.   1.2]]

>>> a = net.sim(p)
>>> print a
[[ 1.]]
```

Заметим, что описанные выше правило и алгоритм обучения персептрона гарантируют сходимость за конечное число шагов для любых задач, которые способен решить персептрон. Это в первую очередь задача классификации векторов, которые относятся к классу *линейно разделимых,* то есть таких, которые можно разделить некоторой прямой линией, в многомерном случае – гиперплоскостью.

Процедура адаптации

Многажды использовав функции sim и step для изменения весов и сдвигов персептрона, можно в конечном итоге построить разделяющую линию, которая решит задачу классификации при условии, что персептрон может её решить. Каждая реализация процесса настройки с использованием всего обучающего множества называется *эпохой* или *циклом*. Следует заметить, что один проход по обучающему множеству не гарантирует, что синтезированная сеть выполнит классификацию нового вектора входа. Возможно, понадобится несколько эпох обучения.

Чтобы пояснить процедуру адаптации, рассмотрим простой пример. Выберем персептрон с одним нейроном и двухэлементным вектором входа (рис. 2.5).

Эта сеть и задания, которые мы собираемся рассматривать, достаточно просты, чтоб выполнить все расчёты вручную.

Допустим, что нужно при помощи персептрона решить задачу классификации, если задано такое обучающее множество:

$$\left\{p_1 = \begin{bmatrix}2\\2\end{bmatrix}, t_1 = 1\right\}\left\{p_2 = \begin{bmatrix}1\\-2\end{bmatrix}, t_2 = 1\right\}\left\{p_3 = \begin{bmatrix}-2\\2\end{bmatrix}, t_3 = 0\right\}\left\{p_4 = \begin{bmatrix}-1\\1\end{bmatrix}, t_4 = 0\right\}.$$

Используем нулевые начальные веса и сдвиг. Для обозначения переменных на каждом шаге используется индекс в круглых скобках. Итак, начальные значения вектора весов $w^T(0)$ и сдвиги $b(0)$ равны соответственно $w^T(0) = [0 \quad 0]$ и $b(0) = 0$.

Посчитаем выход персептрона для первого вектора входа p_1:

$$\begin{aligned} a &= \text{hardlim}(w^T(0)\,p_1 + b(0)) = \\ &= \text{hardlim}\left(\left[\begin{bmatrix}0 & 0\end{bmatrix}\begin{bmatrix}2\\2\end{bmatrix}\right] + 0\right) = \text{hardlim}(0) = 0. \end{aligned} \qquad (2.9)$$

Выход не совпадает с целевым значением t_1 и необходимо использовать правило настройки, чтоб вычислить необходимые изменения весов и сдвигов:

$$\begin{cases} e = t_1 - a = 0 - 1 = -1, \\ \Delta w^T = e p_1^T = (-1)\begin{bmatrix}2 & 2\end{bmatrix} = \begin{bmatrix}-2 & -2\end{bmatrix}, \\ \Delta b = e = (-1) = -1. \end{cases} \qquad (2.10)$$

Посчитаем новые веса и сдвиг:

$$\begin{cases} w^{Tnew} = w^{Told} + \Delta w^T = \begin{bmatrix} 0 & 0 \end{bmatrix} + \begin{bmatrix} -2 & -2 \end{bmatrix} = \begin{bmatrix} -2 & -2 \end{bmatrix} = w^T(1), \\ b^{new} = b^{old} + \Delta b = 0 + (-1) = -1 = b(1). \end{cases} \tag{2.11}$$

Обратимся к новому вектору входа p_2, тогда

$$\begin{aligned} a &= hard\lim(w^T(1)p_2 + b(1)) = \\ &= hard\lim\left(\left[\begin{bmatrix} -2 & -2 \end{bmatrix}\begin{bmatrix} 1 \\ -2 \end{bmatrix}\right] + (-1)\right) = hard\lim(1) = 1. \end{aligned} \tag{2.12}$$

В этом случаях выход персептрона совпадает с целевым выходом, так что изменений весов не нужно. Значит:

$$\begin{cases} w^T(2) = w^T(1) = \begin{bmatrix} -2 & -2 \end{bmatrix}, \\ b(2) = b(1) = -1. \end{cases} \tag{2.13}$$

Продолжим процесс и убедимся, что после третьего шага настройки не изменились:

$$\begin{cases} w^T(3) = w^T(2) = \begin{bmatrix} -2 & -2 \end{bmatrix}, \\ b(3) = b(2) = -1, \end{cases} \tag{2.14}$$

а после четвёртого приняли значения

$$\begin{cases} w^T(4) = \begin{bmatrix} -3 & -1 \end{bmatrix}, \\ b(4) = 0. \end{cases} \tag{2.15}$$

Чтоб узнать, получено ли удовлетворительное решение, нужно сделать ещё один проход через все вектора входа и сравнить их с целевым множеством. Снова используем первый член обучающей последовательности и получим:

$$\begin{cases} w^T(5) = w^T(4) = \begin{bmatrix} -3 & -1 \end{bmatrix}, \\ b(5) = b(4) = 0. \end{cases} \tag{2.16}$$

Переходя ко второму члену, получим такой результат:

$$\begin{cases} w^T(6) = \begin{bmatrix} -2 & -3 \end{bmatrix}, \\ b(6) = 1. \end{cases} \tag{2.17}$$

На этом закончим ручные вычисления.

Теперь выполним аналогичные расчёты, используя функцию обучения train. Снова сформируем модель персептрона (рис. 2.5):

```
net = nl.net.newp([[-2,2],[-2,2]],1)
```

Введём первый элемент обучающего множества:

```
p = [[2, 2]]
t = [[1]]
```

Запустим процедуру train, установив параметр epochs = 1

```
e = net.train(input = p, target = t, epochs = 1)
e = [-1]
```

Скорректированные вектора весов равны

```
w = -2  -2
b = -1
```

Это совпадает с результатами ручного расчёта. Теперь можно ввести второй элемент множества и т. д. Но можно выполнить процедуру автомаически, задав всё обучающее множество и выполнив проход:

```
net = nl.net.newp([[-2,2],[-2,2]],1)
p = [[2,2],[1,-2],[-2,-2],[-1,1]]
e = net.train(input = p, target = t, epochs = 2, show = 1)
print net.sim(p).transpose()
     [[ 1.  1.  0.  0.]]
```

Скорректированные векторы весов и сдвиг равны:

```
net.layers[0].np['w']
     [[ 3, 0.]]
net.layers[0].np['b']
     [ 2. ]
```

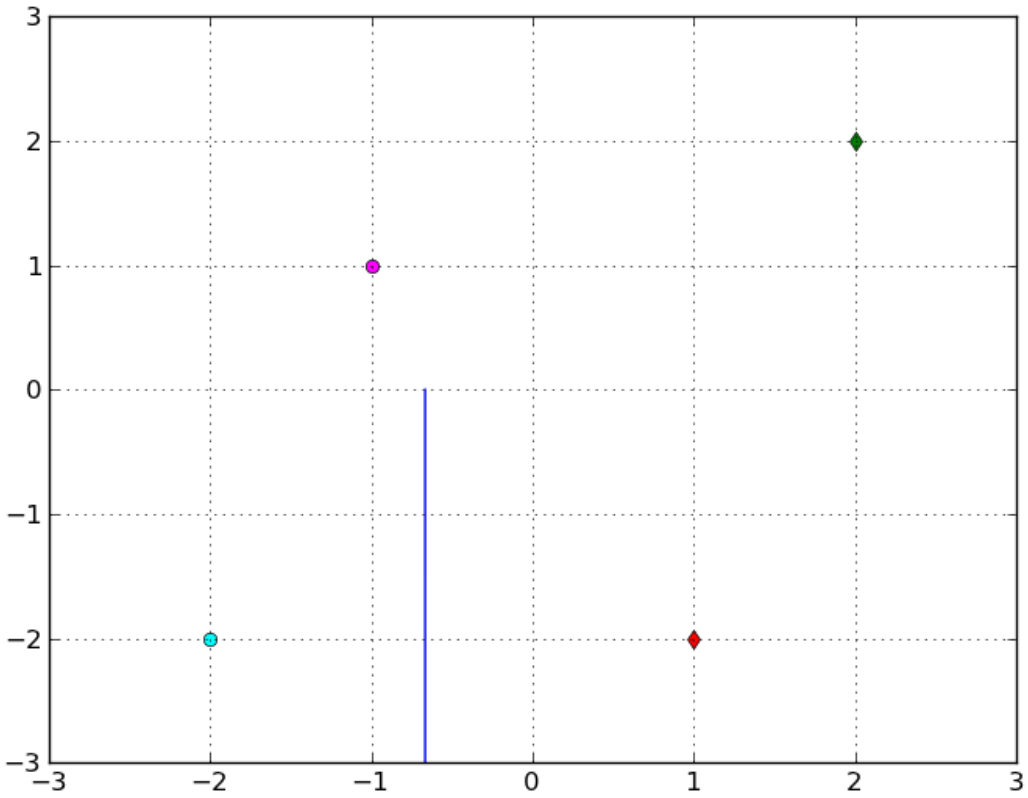


Рисунок 2.6 – Тестовые точки и классифицирующая линия, соответствующая параметрам обученного персептрона

Решение совпадает с целевыми выходами, хотя веса и сдвиг несколько изменились по сравнению с расчётами, поскольку в автоматическом режиме мы выполнили два полных прохода, а в ручном только полтора. Если бы рассчитанные выходы персептрона не совпали с целевыми значениями, то необходимо было бы выполнить ещё несколько циклов настройки.

Взгляните на визуализацию процесса обучения персептрона, например, по адресу http://lcn.epfl.ch/tutorial/english/perceptron/html/index.html

Нейронные сети на основе персептрона имеют ряд ограничений. Во-первых, выход персептрона может быть равен лишь 0 или 1. Во-вторых, персептроны могут решать задачу классификации только для линейно разделимых наборов векторов. Изменив в вышеприведённом примере t на [[1],[0],[1],[0]] и введя команду e = net.train(input = p, target = t, epochs = 100, show = 1) можно наблюдать бессмысленность попыток персептрона разделить линейно неразделимые вектора.

Для решения более сложных задач можно использовать сети с несколькими персептронами. Например, для классификации четырёх векторов на четыре группы можно построить сеть с двумя персептронами, чтоб сформулировать две разделяющие линии и таким образом приписать каждому вектору свою область.

Под конец необходимо ответить, что основное назначение персептронов – решать задачу классификации. Они хорошо справляются с ней. Длительность обучения чувствительна к выбросам, но и в этом случае решение можно гарантированно построить. Для решения задач классификации линейно неразделимых векторов можно использовать многослойный персептрон либо другие виды нейронных сетей, например сети с обратным распространением ошибки.

2.4 Порядок выполнения работы

1. Провести моделирование персептрона.
2. Провести настройку параметров персептрона.
3. Провести классификацию векторов (по варианту в таблице) при помощи персептрона.

Таблица 2.1 – Варианты заданий

Вариант	Координаты точек	Номера точек первого класса	Номера точек второго класса
1	(-2;1), (-1;0), (0;0), (0.5;-1), (0.5;1)	1,2	3,4,5
2	(-1;-1), (0.5;0), (0;0), (1;1)	1	2,3,4
3	(-0.5;-1), (-1;0), (-1.5;1.5), (0;0), (0.5;-1)	1,2,3	4,5
4	(1;-1), (-1;-1), (0.5;0), (-1.5;1)	1,3	2,4
5	(0;-2), (0;-1), (0.5;-0.5), (-1;-1), (-0.5;1)	1,2,3	4,5
6	(-0.5;-0.5), (-1;0), (0;0.5), (0.5;1)	1,2	3,4
7	(0;0.5), (-1;0), (0.5;-1), (-1.5;1.5), (0.5;0)	1,3,5	2,4
8	(0;0), (0.5;0.5), (-0.5;-0.5), (-1;-1)	1,2	3,4
9	(-1;-0.5), (-0.5;0), (0;0), (0.5;0.5), (1;1)	1,2,3	4,5
10	(0;-1), (-0.5;0), (0;1), (1;1)	1,2,3	4
11	(1;0.5), (0.5;1), (-0.5;0), (-0.5;1), (-1;0.5)	1,2	3,4,5
12	(0;0), (0.5;1), (-0.5;-1), (-1;-0.5)	1,2	3,4
13	(0.5;0), (0;1), (-0.5;-0.5), (-1;0), (-0.5;1)	1,2,3	4,5

14	(0;0.5), (0;-0.5), (-1;0.5), (-1;-0.5)	2,3	1,4
15	(0.5;0.5), (0.5;-1), (-0.5;0.5), (0;0), (0;1)	2,4	1,3,5

4. Внести результаты в отчёт.

5. При помощи приведённого ниже фрагмента сгенерировать классы векторов:

```
import neurolab as nl
n1 = 3          # количество векторов первого класса
n2 = 3          # количество векторов первого класса
x0 = 10
alpha = 15
y0 = 50
beta = 20
X = numpy.random.normal(x0,alpha,(n1,2))
Y = numpy.random.normal(y0,beta,(n2,2))
P = numpy.concatenate((X,Y)) # формирование входных векторов
# формирование выходных векторов
T = numpy.zeros((1, len(P)))
T[:n1] = numpy.ones(n1)
T = T.transpose()
min = numpy.min(P)
max = numpy.max(P)
net = neurolab.net.newp([[min,max],[min,max]],1)
```

6. Сделать классификацию векторов.

7. С помощью следующего кода сделать тестовый пример для классификации:

```
x1 = 45
y1 = 50
x1 = numpy.random.normal(x1,beta,1)
y1 = numpy.random.normal(y1,beta,1)
p = numpy.concatenate(x1,y1)
```

Сделать классификацию

8. Результат внести в отчёт

1.5 Содержание отчёта

Содержание отчёта должно соответствовать плану работы. Обязательные пункты:

1) Название работы
2) Цель работы
3) Краткое описание исследованной нейронной сети и её архитектуры
4) Краткое описание проведённых исследований в виде графиков и числовых данных
5) Листинг программы моделирования нейронной сети
6) Полученные результаты в виде графиков и числовых данных
7) Расширенные выводы из лабораторной работы

Дополнительные задания:

1) Провести персептроном классификацию на 4 класса.
2) Использовать персептрон для обработки реальных данных (своя задача или подготовленные выборки, например из репозитория UCI (http://archive.ics.uci.edu/ml/index.html), например http://archive.ics.uci.edu/ml/datasets/Iris)
3) Использовать персептрон для распознавания образов (база MNIST: http://code.google.com/p/vlabdownload/downloads/detail?name=MNIST.rar&can=2&q=#makechanges — 60 тысяч подготовленных изображений рукописных цифр от 0 до 9).

1.6 Контрольные вопросы и задания

1. Что такое персептрон? Какова его архитектура?
2. Какие алгоритмы обучения персептрона вы знаете?
3. Охарактеризуйте правила настройки параметров персептрона
4. Охарактеризуйте процедуру адаптации параметров персептрона
5. Каковы достоинства и недостатки персептрона?