**НИТУ «МИСиС»**

Институт ИТКН

**Кафедра инженерной кибернетики**

Направление подготовки: 01.03.04 прикладная математика Квалификация (степень): бакалавр

**КУРСОВАЯ РАБОТА**

**«Искусственные нейронные сети»**

на тему «Однослойный перцептрон. Теорема о сходимости перцептрона»

**VII семестр 2020 – 2021 у.г.**

**Учащийся: *Дементьева Ю. Д.***

**Группа: БПМ-17-1**

**Проверила: Кондыбаева А. Б.**

**Москва 2020**

Содержание

[**1.** **Введение** 3](#_Toc57808796)

[**2.** **Актуальность выбранной темы** 4](#_Toc57808797)

[**3.** **Цель работы** 4](#_Toc57808798)

[**4.** **Однослойный перцептрон** 5](#_Toc57808799)

[**5.** **Теорема о сходимости перцептрона** 8](#_Toc57808800)

[**5.1.** **Необходимые обозначения** 8](#_Toc57808801)

[**5.2.** **Формулировка теоремы** 9](#_Toc57808802)

[**5.3.** **Абсолютная процедура адаптации однослойного перцептрона на основе коррекции ошибок** 9](#_Toc57808803)

[**Список литературы** 12](#_Toc57808804)

[**ПРИЛОЖЕНИЕ А. Программная реализация однослойного перцептрона** 13](#_Toc57808805)

1. **Введение**

Нейронные сети были разработаны с целью имитации нервной системы человека для решения задач машинного обучения на основе вычислительных элементов, работа которых напоминала бы действие человеческих нейронов. Концептуальной задачей, стоящей перед разработчиками нейронных сетей, является создание искусственного интеллекта (ИИ) на основе машин, архитектура которых была бы способна имитировать процессы, происходящие в нервной системе человека. Решить подобную задачу не так просто, поскольку вычислительная мощность даже самых быстрых современных компьютеров составляет лишь ничтожную долю возможностей человеческого мозга. Первые упоминания о нейронных сетях появились после появления компьютеров. Алгоритм перцептрона Розенблатта вызвало сенсацию относительно перспектив искусственного интеллекта. Но алгоритм требовал завышенные запросы в отношении необходимых объемов данных и вычислительных ресурсов нейронных сетей, что считалось основной помехой на пути их дальнейшего развития. В конечном счете на рубеже тысячелетия повышение доступности данных и увеличение вычислительных мощностей обеспечили дальнейший прогресс нейронных сетей, и эта область возродилась под названием "глубокое обучение".

# **Актуальность выбранной темы**

Потенциальными областями применения искусственных нейронных сетей являются те, где человеческий интеллект малоэффективен, а традиционные вычисления трудоёмки или не отражают реальные физические процессы и объекты. Актуальность применения нейронных сетей многократно возрастает, когда появляется необходимость решения плохо формализованных задач. Основные области применения нейронных сетей: автоматизация процесса классификации, автоматизация прогнозирования, автоматизация процесса распознавания и т.д. Тем самым разобраться в том, как устроен перцептрон, являющийся основной составляющей любой нейронной сети, одна из главных задач в изучении области «Нейронные сети и глубокое обучение».

# **Цель работы**

Изучить принципы работы однослойного перцептрона, а также теорему о его сходимости.

# **Однослойный перцептрон**

Перцептрон представляет собой простейшую форму нейронной сети, предназначенную для классификации линейно-разделимых сигналов, у которых образы можно разделить некоторой гиперплоскостью. Перцептрон состоит из одного нейрона с настраиваемыми синаптическими весами и порогами. Первый алгоритм настройки свободных параметров для такой нейронной сети был создан Розенблаттом. Перцептрон строится для модели Мак-Каллоки-Питца, которая представляет нелинейный нейтрон. Такая нейронная модель состоит из линейного сумматора и ограничителя, реализованного в виде пороговой функции вычисления знака. Суммирующий узел этой нейронной модели вычисляет линейную комбинацию входных сигналов, поступающих на синапсы с учетом внешнего возмущения(порога). Полученная сумма (индуцированное локальное поле) передается на узел ограничителя. Таким образом, выход нейрона принимает значение +1, если сигнал на выходе сумматора положителен, и -1, если отрицателен.

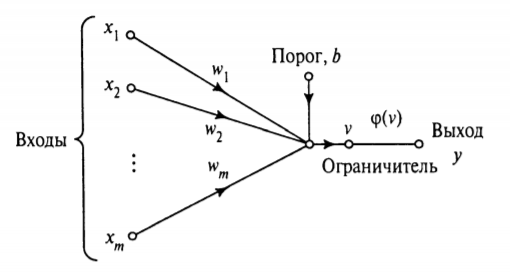


Рисунок 1 — Граф передачи сигнала для перцептрона

На Рисунке 1 синаптические веса перцептрона обозначены сигналы, поступающие на вход перцептрона, обозначены а пороговое значение – b. Согласно структуре модели, можно заключить, что входной сигнал ограничителя нейрона определяется выражением:

Целью перцептрона является корректное отнесение множества внешних стимулов к одному из двух классов: A или B. Решающее правило такой классификации заключается в том, что входной сигнал относится к классу А, если выход равен , и к классу B, если равен .

В том случае, когда в качестве классификатора выступает перцептрон, имеется всего две области решения разделенные гиперплоскостью, определяемые формулой:

На Рисунке 2 проиллюстрирован пример, когда для переменных разделяющая гиперплоскость вырождается в прямую. Пороговое значение определяет смещение разделяющей поверхности по отношению к началу координат.

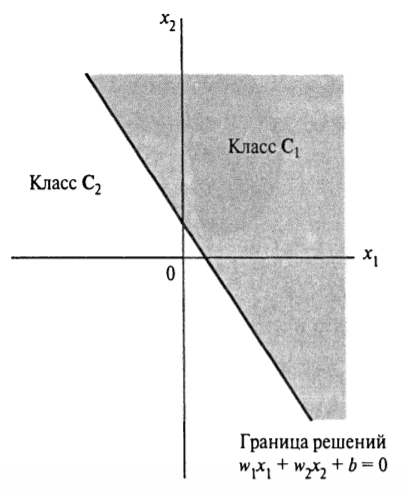


Рисунок 2 — Разделяющая поверхность в виде гиперплоскости для двумерной задачи классификации образов на две задачи

Синаптические веса перцептрона можно адаптировать итеративным методом. В частности, для настройки весовых коэффициентов можно использовать алгоритм сходимости перцептрона.

# **Теорема о сходимости перцептрона**

# **Необходимые обозначения**

Введем необходимые обозначения и описания к элементам перцептрона, которые потребуются для алгоритма обучения перцептрона, основанного на коррекции ошибок. На Рисунке 3 порог рассматривается как синаптический вес связи с фиксированным входным сигналом +1. Для входного вектора размерности (m+1) запишем как:

Где n – номер итерации алгоритма. Аналогично можно определить (m+1)-мерный вектор весовых коэффициентов:

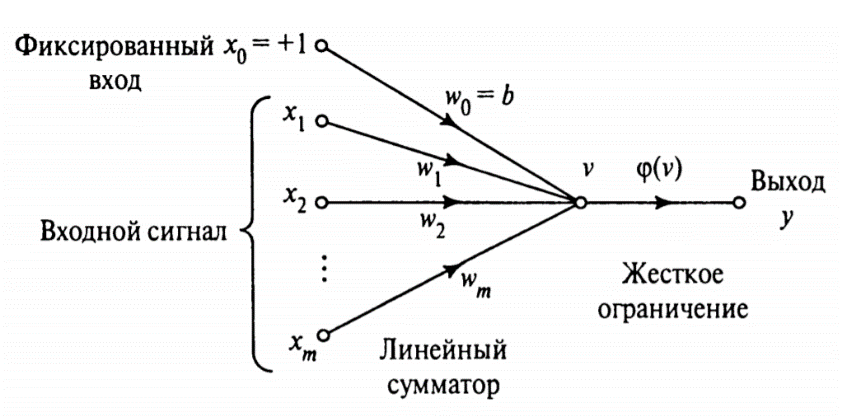


Рисунок 3 — Граф передачи сигнала для перцептрона

Таким образом, выход линейного сумматора можно записать в более компактной форме

Где – пороговое значение b(n). При фиксированном значении n уравнение = 0 в m-мерном пространстве с координатами определяет гиперплоскость, которая является поверхностью решений для двух различных классов входных сигналов.

Для того, чтобы перцептрон работал корректно, два класса А и В должны быть линейно-разделимы. Другими словами, для правильной классификации образы должны быть значительно отдалены друг от друга, чтобы поверхность решений могла представлять собой гиперплоскость.

# **Формулировка теоремы**

Теорема сходимости для алгоритма обучения перцептрона с фиксированным приращением для перцептрона сформулирована так: «Пусть подмножество векторов обучения и линейно-разделимы и входные сигналы поступают перцептрону только из этих подмножеств. Тогда алгоритм обучения перцептрона сходится после некоторого числа итераций в том смысле, что

является вектором решения для»

# **Абсолютная процедура адаптации однослойного перцептрона на основе коррекции ошибок**

Рассмотрим абсолютную процедуру адаптации однослойного перцептрона на основе коррекции ошибок, в которой Пусть – наименьшее целое число, для которого выполняется соотношение:

Согласно этой процедуре, если скалярное произведение на шаге n имеет неверный знак, то на итерации n+1 будет иметь правильный знак. Предполагается, что если знак произведения некорректен, то можно изменить последовательность обучения для итерации n+1, приняв Другими словами, каждый из образов представляется перцептрону до тех пор, пока он не будет классифицирован корректно.

В Таблице 1 представлен алгоритм сходимости перцептрона. Символ на третьем шаге алгоритма означает функцию вычисления знака (сигнум).

Таблица 1. Алгоритм сходимости перцептрона

|  |
| --- |
| Переменные и параметры  – вектор-строка размерности m+1;  – вектор-строка размерности m+1;  b(n) – порог;  y(n) – фактический отклик;  d(n) – желаемый отклик; |
| 1. Инициализация   Пусть w(0) = 0. Последующие вычисления выполняются для шагов n = 1,2, … |
| 1. Активация   На шаге n активируем перцептрон, используя вектор x(n) с вещественными компонентами и желаемый отклик d(n) |
| 1. Вычисление фактического ответа   Вычисляем фактический отклик перцептрона: |
| 1. Адаптация вектора весов   Изменяем вектор весов перцептрона:  *),* |
| 1. Продолжение   Увеличиваем номер итерации n на единицу и возвращаемся к п. 2 алгоритма. |

Алгоритм адаптации вектора весовых коэффициентов w(n) соответствует правилу обучения на основе коррекции ошибок:

*),*

Где 𝜼 – параметр скорости обучения, а разность сигнал ошибки. Параметр скорости является положительной константой, принадлежащей интервалу . Выбирая значение параметра скорости обучения из этого диапазона, следует учитывать два взаимоисключающий требования:

1. Усреднение предыдущих входных сигналов, обеспечивающее устойчивость оценки вектора весов, требует малых значений .
2. Быстрая адаптация к реальным изменениям распределения процесса, отвечающего за формирование векторов сходного сигнала x, требует больших значений .

# **Список литературы**

1. Ф. Розенблатт, «Перцептрон: вероятностная модель для хранения и организации информации в мозге» Психологический обзор, 1958
2. М. Мори и А. Ростамизаде, «Перцептрон ошибочные границы»Arxiv, 2013
3. Николенко С. И. – «Глубокое обучение», Библиотека программиста, 2018

# **ПРИЛОЖЕНИЕ А. Программная реализация однослойного перцептрона**

Результат работы модели 𝑎(𝑀)⊂{𝐎-𝐜𝐥𝐚𝐬𝐬,𝐗-𝐜𝐥𝐚𝐬𝐬} - распознанный класс "крестик" или "нолик".

# импорт основных классов

import numpy as np

import json

from pip.\_internal import main

main(['install', 'termcolor','random'])

import random

import math

from termcolor import colored, cprint

#создаем класс, который генерит крестики и нолики

class createXO(object):

def \_\_init\_\_(self):

self.matrix = [[0]\*7 for i in range(7)]

pass

def createX(self):

# задаем начальную точку для рисования крестика

x = random.randint(2,4)

y = random.randint(2,4)

self.matrix[x][y] = 1

# задаем радиус рисования крестика

rad = min(6-x,6-y, x, y)

# заполняем крестик по диагоналям

for i in range(1, rad):

self.matrix[x-i][y-i] = 1

self.matrix[x-i][y+i] = 1

self.matrix[x+i][y-i] = 1

self.matrix[x+i][y+i] = 1

# оставшиеся клетки заполняем случайно 0/1

self.matrix[x+rad][y-rad] = random.randint(0,1)

self.matrix[x-rad][y+rad] = random.randint(0,1)

self.matrix[x+rad][y+rad] = random.randint(0,1)

self.matrix[x-rad][y-rad] = random.randint(0,1)

return self.matrix

def createO(self):

x = random.randint(2,4)

y = random.randint(2,4)

rad = min(6-x,6-y, x, y)

for i in range(-rad+1, rad):

self.matrix[x+i][y-rad] = 1

self.matrix[x+i][y+rad] = 1

self.matrix[x+rad][y+i] = 1

self.matrix[x-rad][y+i] = 1

self.matrix[x+rad][y-rad] = random.randint(0,1)

self.matrix[x-rad][y+rad] = random.randint(0,1)

self.matrix[x+rad][y+rad] = random.randint(0,1)

self.matrix[x-rad][y-rad] = random.randint(0,1)

return self.matrix

def lenCreate(self):

return len(self.matrix)

def getIndex(self,i,j):

return self.matrix[i][j]

# размер данных

data\_value = 1000

# размер элемента данных

matrix\_len=7

# заполняем нулями матрицу для данных

data = np.zeros(((data\_value+1),(matrix\_len\*matrix\_len+1)))

number = 0

def saveObject(row, matrix\_len):

dist={}

numberRow = 0

for i in range(0, len(row)-1, matrix\_len ):

st=[]

for j in range(i,i+matrix\_len):

index=row[j]

st.append(index)

dist.update({numberRow:st})

numberRow+=1

dist.update({'answer':'X' if row[len(row)-1] == 1 else '0'})

return dist

# заполняем матрицу данных элементами 0/1

for h in range(data\_value):

a = createXO()

# выбираем элемнт случайным образом

choose = random.randint(0,1)

if choose == 0:

m=a.createO()

# в матрицу данных 0/1 кладем по строкам -> одной строкой один элемент

for l in range(matrix\_len):

for p in range(matrix\_len):

data[h][p + matrix\_len\*l] = m[l][p]

data[h][matrix\_len\*matrix\_len]=choose

if choose == 1:

m=a.createX()

for l in range(matrix\_len):

for p in range(matrix\_len):

data[h][p + matrix\_len\*l] = m[l][p]

data[h][matrix\_len\*matrix\_len]=choose

import random

def printXO(data):

for j in range(0, matrix\_len):

for x in data[j \* matrix\_len: (j + 1) \* matrix\_len]:

print(colored(str(int(x)), 'red'), end=' ') if x == 1 else print(int(x), end=' ')

print()

i = random.randint(0, len(data))

printXO(data[i])

# предсказываем результат определения элемента, на вход элемент и используемые веса

def predict(row, weights):

activation = weights[0]

for i in range(len(row)-1):

# перемножая веса на вход, получаем выход

activation += weights[i + 1] \* row[i]

return 1.0 if activation >= 0.0 else 0.0

# Функция активации

def activation(x):

return round((1 + math.exp(-x))\*\*(-1))

# row-данные по крестику/нолику

# предсказываем результат определения элемента, на вход элемент и используемые веса

def a\_x(row, weights):

sum\_wx = 0

for i in range(1,len(weights)):

sum\_wx += weights[i]\*row[i]

return activation(sum\_wx - weights[0])

def z\_elem(a,y):

return (a - y) \*\* 2

def sum\_z\_elem(train, weights):

Q=0

for i in range(len(train)):

y\_i=train[i][matrix\_len\*matrix\_len]

a = a\_x(train[i], weights)

Q += z\_elem(a, y\_i)

return Q

def calc\_phi\_der(x):

return math.exp(-x)/((1 + math.exp(-x))\*\*2)

def answerPrint(answer):

if(answer == 1):

return "Крестик"

return "Нолик"

# обучение

def train\_weights(train, l\_rate, n\_epoch):

# иициализируем веса в начале нулями

weights = [0.0 for i in range(len(train[0]))]

# Инициализация функции качества

Q= sum\_z\_elem(train, weights)

Q1=0

l=len(train)

for i in range(data\_value):

Q+=predict(train[i], weights)

n\_current = 0

# while(abs(Q-Q1) > 1e-10 and n\_current < n\_epoch):

while( Q!=Q1 and n\_current < n\_epoch):

Q1=Q

n\_current +=1

obj=random.randint(0, len(train)-1)

a = a\_x(train[obj], weights)

eps = z\_elem(a, train[obj][matrix\_len\*matrix\_len])

scalar\_x\_w = 0

for i in range(len(weights)):

scalar\_x\_w += train[obj][i]\*weights[i]

m = 2\*(a - train[obj][matrix\_len\*matrix\_len])\*calc\_phi\_der(scalar\_x\_w)

for j in range(len(weights)):

weights[j] -= l\_rate\*m\*train[obj][j]

Q= Q\*(l-1)/l +eps\*eps/l

return weights

ready\_weights= train\_weights(data, 0.5, 500)

print("Весы: ",ready\_weights)

# Проверка полноты

print("---Проверка полноты---")

# Количество объектов, на которых проверяется правильность настроенных весов

quantity = 1000;

trueAnswer=0

lenObj=len(data[0])

for i in range(0, quantity):

if(a\_x(data[i], ready\_weights)==data[i][lenObj-1]):

trueAnswer+=1

print('Точность', trueAnswer\*100/quantity, "%")

print("---Генерация объекта и ответ обучающей модели---")

genObject = random.randint(0, len(data))

printXO(data[genObject])

print("Ответ:", answerPrint(a\_x(data[genObject], ready\_weights)))