|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт информационных технологий |
| Кафедра инструментального и прикладного программного обеспечения (ИППО) |
|  |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1** | |
| **по дисциплине** | |
| **«**Интеллектуальные информационные системы**»** | |
| по теме: «Программирование искусственного нейрона» | |
| Выполнил студент группы ИКБО-16-17 | Акжигитов Р. Р. |
| Принял ассистент | Рачков А. В. |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «6» апреля 2020 г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_\_\_2020 г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Москва 2020

**Оглавление**

[Цель лабораторной работы 3](#_Toc37075765)

[Задание 3](#_Toc37075766)

[Описание реализованной модели нейрона и процедуры его обучения 3](#_Toc37075767)

[Распределение в пространстве обучающих данных 5](#_Toc37075768)

[Начальное состояние, ход обучения и его результат 6](#_Toc37075769)

[Графическое представление результатов обучения нейрона 7](#_Toc37075770)

[Исходный текст программы 8](#_Toc37075771)

[Заключение 11](#_Toc37075772)

[Список литературных источников 12](#_Toc37075773)

# Цель лабораторной работы

Целью данной лабораторной работы является создание программы, реализующей искусственный нейрон; разработка процедуры обучения нейрона; использование полученных результатов для решения тестовых задач классификации и аппроксимации.

# Задание

Реализовать сигмоидный нейрон (вариант 2), способный классифицировать точку в двумерном пространстве.

# Описание реализованной модели нейрона и процедуры его обучения

Функция активации – сигмоид, так как необходимо реализовать именно сигмоидный нейрон. Обучение «оффлайн» с учителем (изменяем коэффициенты с зависимости от правильного ответа и ответа нейросети). Коэффициенты эпсилон (остановить обучение сети при достижении такой ошибки) или кол-во элементов в наборе данных для обучения, бета (крутизна сигмоидной функции) и N (скорость обучаемости сети) выставляются вручную до запуска для конфигурирования. Для создания автономной и более эффективной сети необходимо применять стратегии изменения коэффициента обучаемости, так как если он будет слишком велик, то есть вероятность «перескочить» ожидаемые веса и такими скачками не дойти до «точки схождения» весов, слишком маленький коэффициент может слишком долго искать ответ.

Можно изначально сгенерировать массив для обучения (кол-во итераций обучения) и полностью обучить на нем нейрон, либо задать эпсилон и использовать бесконечный цикл до достижения нейроном среднего значения ошибки (например, среднее из 100 последних значений) до определенного значения эпсилон.

Входы нашего нейрона: значения двух координат и константа смещения (1). Для возможности смещения линии аппроксимации, а не только через (0; 0). Соответственно для каждого входа свой вес.

Выход нейрона – число, вероятность нахождения введенной точки выше графика введенной функции.

Ниже представлен код основной процедуры обучения (вариант: обход заданного кол-ва точек).

Алгоритм обучения:

1. Вычисляем целевую функцию посредством вычисления сигмоидной функции от матричного умножения координат на веса,
2. Находим ошибку: разница правильного ответа на данные координаты и полученного результата
3. Корректируем веса: к весам прибавляем изменение веса (dw), умноженный на коэффициент обучаемости (dw вычисляется как матричное умножения транспонированных координат на производную значения целевой функции, умноженной на ошибку).

for x in x\_total:

u\_i = y(x, w)

e = correct\_answer(x, solved) - u\_i

w = correct\_weights(w, dw(x, e, u\_i))

error.append(np.average(np.abs(e)) if len(e) > 1 else np.abs(e)[0])

if len(error) > n\_error and np.average(error[-n\_error:]) < eps:

break

Приблизительная диаграмма процесса обучения представлена на Рисунок 1.

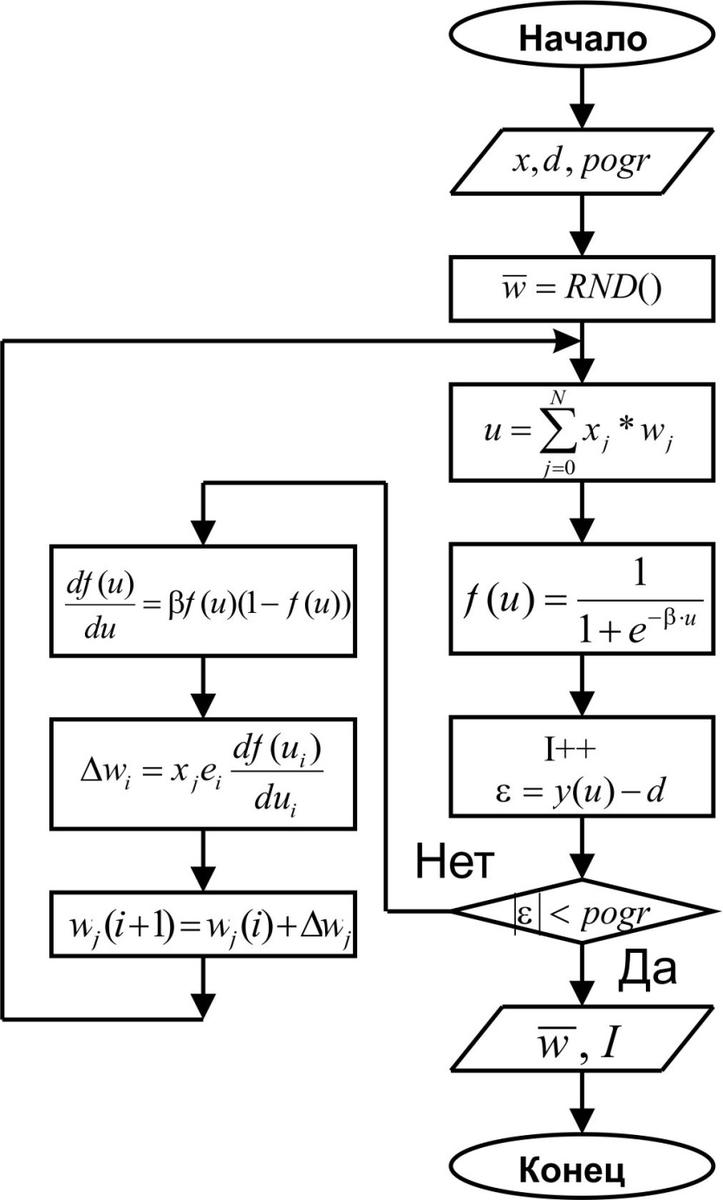


Рисунок – диаграмма алгоритма обучения

# Распределение в пространстве обучающих данных

Для универсализации рассмотрим любое возможное деление двумерного пространства на две части (то есть граница между результатами 0 и 1 – это любая линейная функция). Тогда для обучения нам необходимо сгенерировать большое кол-во точек в любом месте двумерного пространства (случайно), после этого проверить выше они или ниже введенного графика (правильный ответ).

iters = 10000

x\_total = np.array([ np.array([[ np.random.uniform(\*x\_lim), np.random.uniform(\*y\_lim), 1 ]]) for i in range(iters) ])

# Начальное состояние, ход обучения и его результат

Начальные значения входных весов неважны, поэтому они генерируются случайно в отрезке от 0 до 1. В идеальном случае веса должны стремиться к такой же пропорции как у коэффициентов при прямой.

Величина коэффициента обучения 1, количество циклов 100 000 (при достижении малой ошибки прерывание).

Ход обучения может охарактеризовать график уменьшения ошибки, представленный на Рисунок 2 (значения усреднены для того, чтобы график всегда помещался в 100 точек).

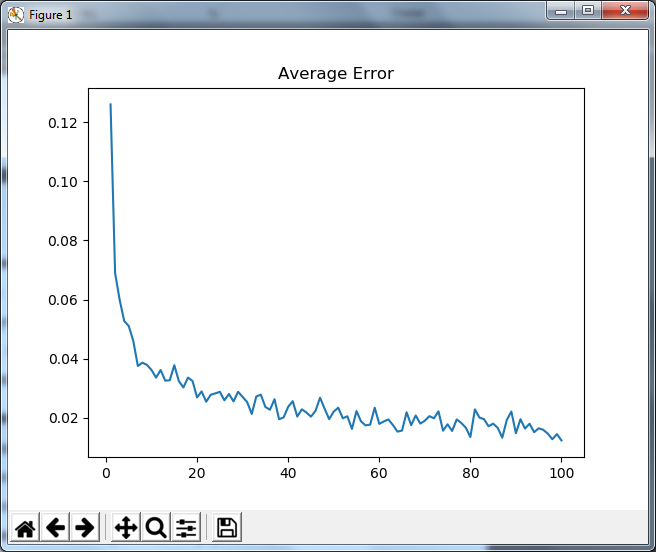


Рисунок – график ошибки нейрона

На Рисунок 3 представлен способ взаимодействия с программой, сначала необходимо ввести функцию (первой степени), далее происходит процесс обучения, после этого программа выдает набор весов (если нормализовать значения, то они будут похожи на нормализованные коэффициенты введенной прямой). Далее идут кол-во произведенных итераций, последнее значение ошибки. Далее спрашиваются координаты точек для проверки и показывается вероятность, что точка выше графика (ближе к 1), ниже или на графике (ближе к 0).

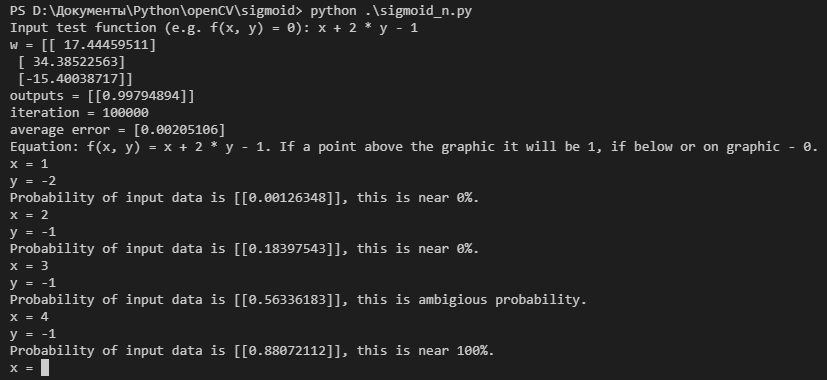


Рисунок – работа с программой, ввод координат для проверки

# Графическое представление результатов обучения нейрона

График зависимости выходного сигнала нейрона от входных данных представлен на Рисунок 4.

Замечено, что без использования активационной функции, график выходного сигнала представляет собой наклоненную плоскость, применение функции искривляет плоскость под форму сигмоиды, причем, чем меньше коэффициент при функции (к 0), тем более пологая становится искривленная плоскость. Однако увеличение кол-ва итераций влечет за собой тот же эффект, что и увеличение коэффициента при активационной функции, плоскость становится более крутой на подъем.

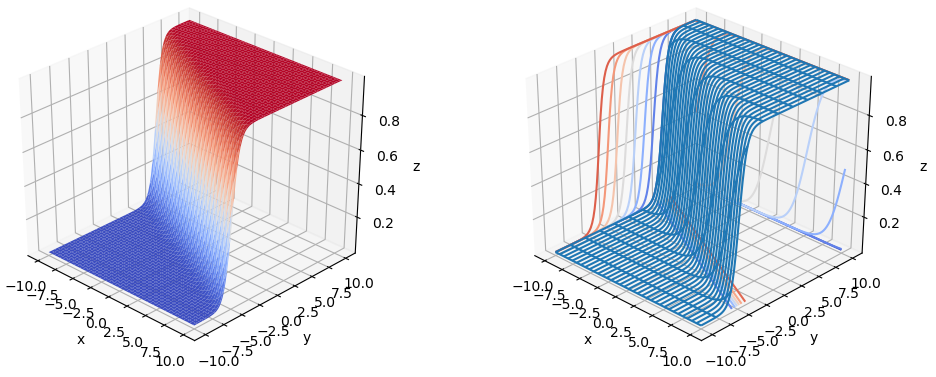


Рисунок – зависимость выходного сигнала нейрона от входных данных

# Исходный текст программы

Исходный текст программы представлен ниже.

import itertools

import numpy as np

import pylab as plt

from mpl\_toolkits.mplot3d import Axes3D

from sympy.solvers import solve

from matplotlib import cm

beta = 0.1

eps = 0.001

n\_error = 100

iters = 100000

n\_points\_per\_grid = 100

x\_lim, y\_lim = [-10, 10], [-10, 10]

w = np.random.rand(3, 1)

x\_total = np.array([ np.array([[ np.random.uniform(\*x\_lim), np.random.uniform(\*y\_lim), 1 ]]) for i in range(iters) ])

f = lambda u: 1 / (1 + np.exp(-beta \* u))

der\_f = lambda u: beta \* f(u) \* (1 - f(u))

dw = lambda x, e, u: np.dot(x.T, e \* der\_f(u))

y = lambda x, w: f(np.dot(x, w))

correct\_weights = lambda w, dw, n=1: w + n \* dw

def correct\_answer(x, solved\_func):

# возвращение статического бинарного ответа

# return np.array([[1, 1, 0, 0]]).T

# return np.array([[int(x[0] < x[1]), int(x[0] >= x[1])]])

# функция y = x

# return int(x[0, 0] \*\* 2 < x[0, 1])

# вычисление введенной функции через библиотеку sympy

# return int(bool(solved\_func.subs('x', x[0, 0]) < x[0, 1]))

# вычисление введенной через eval

# print(x)

return int(eval(str(solved\_func).replace('x', str(x[0, 0]))) < x[0, 1])

def get\_answer(input\_data, w):

prob = y(np.array([input\_data]), w)

estimation = ''

if 0.5 - 0.05 < prob < 0.5 + 0.05:

estimation = 'near 50%'

elif prob > 0.8:

estimation = 'near 100%'

elif prob < 0.2:

estimation = 'near 0%'

else:

estimation = 'ambigious probability'

print('Probability of input data is {}, this is {}.'.format(prob, estimation))

def input\_test\_value(func):

print('Equation: f(x, y) = {}. If a point above the graphic it will be 1, if below or on graphic - 0.'.format(func))

while True:

x\_inp, y\_inp = float(input('x = ')), float(input('y = '))

get\_answer(np.array([x\_inp, y\_inp, 1]), w)

error = []

func = input('Input test function (e.g. f(x, y) = 0): ').replace('= 0', '').replace('=0', '')

solved = solve(func, 'y')[0]

# for i in range(iters):

for x in x\_total:

# while True:

# x = np.array([[ np.random.uniform(\*x\_lim), np.random.uniform(\*y\_lim), 1 ]])

u\_i = y(x, w)

e = correct\_answer(x, solved) - u\_i

w = correct\_weights(w, dw(x, e, u\_i))

error.append(np.average(np.abs(e)) if len(e) > 1 else np.abs(e)[0])

if len(error) > n\_error and np.average(error[-n\_error:]) < eps:

break

print('w =', w)

print('outputs =', u\_i)

print('iteration =', len(error))

print('average error =', error[-1])

plt.title('Average Error')

# Отображение всех точек

# plt.plot(range(1, len(error) + 1), error)

# Отображение 100 точек (по-среднему)

plt.plot(range(1, 100 + 1), [np.average(arr) for arr in np.array\_split(error, 100)])

# plt.plot(range(1, len(error) // (iters // 100) + 1), [np.average(arr) for arr in np.array\_split(error, 100)])

plt.show()

# Зависимость результата сигмоида от входных координат

x\_side = np.linspace(x\_lim[0], x\_lim[1], n\_points\_per\_grid)

y\_side = np.linspace(y\_lim[0], y\_lim[1], n\_points\_per\_grid)

X, Y = np.meshgrid(x\_side, y\_side)

# вычисление каждой точки графика

# Z = []

# cartesian = np.array([ [[i[0], i[1], 1]] for i in itertools.product(x\_side, y\_side) ])

# for c in cartesian:

# Z.append([c[0, 0], c[0, 1], y(c, w)])

# Z = np.array(Z)

# способ отрисовать с помощью reshape

# ax.plot\_surface(X, Y, Z[:, 2].reshape(n\_points\_per\_grid, n\_points\_per\_grid))

# отрисовка каждой точки

# for z in np.vsplit(Z, n\_points\_per\_grid):

# ax.plot3D(z[:, 0], z[:, 1], z[:, 2])

Z = f(X \* w[0] + Y \* w[1] + 1 \* w[2])

fig = plt.figure(figsize=plt.figaspect(1 / 2.5))

ax1 = fig.add\_subplot('121', projection='3d')

ax1.plot\_surface(X, Y, Z, cmap=cm.coolwarm)

ax1.set\_xlabel('x'), ax1.set\_ylabel('y'), ax1.set\_zlabel('z')

ax2 = fig.add\_subplot('122', projection='3d')

ax2.plot\_wireframe(X, Y, Z, rstride=10, cstride=3)

ax2.contour(X, Y, Z, zdir='z', offset=0, cmap=cm.coolwarm)

ax2.contour(X, Y, Z, zdir='x', offset=x\_lim[0], cmap=cm.coolwarm)

ax2.contour(X, Y, Z, zdir='y', offset=y\_lim[1], cmap=cm.coolwarm)

ax2.set\_xlabel('x'), ax2.set\_ylabel('y'), ax2.set\_zlabel('z')

def on\_move(event):

if event.inaxes == ax1:

if ax1.button\_pressed in ax1.\_rotate\_btn:

ax2.view\_init(elev=ax1.elev, azim=ax1.azim)

elif ax1.button\_pressed in ax1.\_zoom\_btn:

ax2.set\_xlim3d(ax1.get\_xlim3d())

ax2.set\_ylim3d(ax1.get\_ylim3d())

ax2.set\_zlim3d(ax1.get\_zlim3d())

elif event.inaxes == ax2:

if ax2.button\_pressed in ax2.\_rotate\_btn:

ax1.view\_init(elev=ax2.elev, azim=ax2.azim)

elif ax2.button\_pressed in ax2.\_zoom\_btn:

ax1.set\_xlim3d(ax2.get\_xlim3d())

ax1.set\_ylim3d(ax2.get\_ylim3d())

ax1.set\_zlim3d(ax2.get\_zlim3d())

else:

return

fig.canvas.draw\_idle()

c1 = fig.canvas.mpl\_connect('motion\_notify\_event', on\_move)

plt.show()

input\_test\_value(func.strip())

# Заключение

В данной лабораторной работе была изучена теория создания простых нейросетей (термины: весовые коэффициенты, целевая функция, функция активации, входные данные), а точнее сигмоидного нейрона, алгоритмы его обучения («онлайн», «оффлайн», стратегии обучения), были визуализированы график падения ошибки и зависимость результата нейрона от входных координат.

# Список литературных источников

1. Гудфеллоу Я., Бенджио И., Курвилль А. Глубокое обучение = Deep Learning. — М.: ДМК-Пресс, 2017. — 652 с. — ISBN 978-5-97060-554-7.
2. Еремин Д. М., Гарцеев И. Б. Искусственные нейронные сети в интеллектуальных системах управления. — М.: МИРЭА, 2004. — 75 с. — ISBN 5-7339-0423-2.
3. Каллан Р. Основные концепции нейронных сетей = The Essence of Neural Networks First Edition. — М.: Вильямс, 2001. — 288 с. — ISBN 5-8459-0210-X.
4. Круглов В. В., Борисов В. В. Искусственные нейронные сети. Теория и практика. — М.: Горячая линия - Телеком, 2001. — 382 с. — ISBN 5-93517-031-0.
5. Миркес Е. М. Нейрокомпьютер. Проект стандарта. — Новосибирск: Наука, 1999. — 337 с. — ISBN 5-02-031409-9. Другие копии онлайн: Нейрокомпьютер. Проект стандарта.
6. Николенко С., Кадурин А., Архангельская Е. Глубокое обучение. — СПб.: Питер, 2018. — 480 с. — ISBN 978-5-496-02536-2.
7. Осовский Станислав. Нейронные сети для обработки информации = Sieci neuronowe do przetwarzania informacji (польск.) / Перевод И. Д. Рудинского. — М.: Финансы и статистика, 2004. — 344 с. — ISBN 5-279-02567-4.
8. Савельев А. В. На пути к общей теории нейросетей. К вопросу о сложности // Нейрокомпьютеры: разработка, применение. — 2006. — № 4—5. — С. 4—14. Архивировано 11 сентября 2016 года.
9. Сигеру Омату, Марзуки Халид, Рубия Юсоф. Нейроуправление и его приложения = Neuro-Control and its Applications. 2-е изд. — М.: ИПРЖР, 2000. — 272 с. — ISBN 5-93108-006-6.