|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 |  | Косенков Александр Александрович |
| Группа |  | РК6-12М |
| Тип задания |  | Лабораторная работа №3 |
| Тема лабораторной работы |  | Программирование искусственного нейрона |

Студент **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_ Косенков А.А.\_**

*подпись, дата фамилия, и.о.*

Преподаватель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_ Федорук В.Г.\_**

*подпись, дата фамилия, и.о.*

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*Москва, 2021 г.*

**Оглавление**

[Задание на лабораторную работу 3](#_Toc91793962)

[Описание сигмоидального нейрона 3](#_Toc91793963)

[Программная реализация 6](#_Toc91793964)

[Обучение нейрона 7](#_Toc91793965)

[Результаты работы программы 7](#_Toc91793966)

[Текст программы 9](#_Toc91793967)

# Задание на лабораторную работу

**Вариант 2-7**

Разработать программу, моделирующую поведение искусственного трехвходового сигмоидального нейрона и обеспечивающую его обучение для решения задачи классификации. Рекомендуется, в тех ситуациях, когда это возможно, использовать режим обучения «оффлайн». Обучить разработанный нейрон на предложенном варианте двухмерных данных (рис. 1) и проверить его работу на ряде контрольных точек.

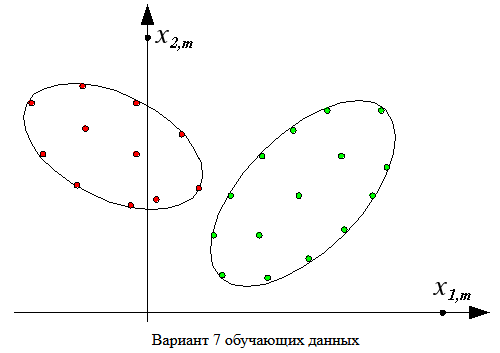


Рис. . Обучающая выборка данных.

Метод

# Описание сигмоидального нейрона

Нейрон данного типа устраняет основной недостаток персептрона (модель искусственного нейрона, предложенная в 1943 г.) – разрывность функции активации . Структурная схема сигмоидального нейрона представлена ниже:

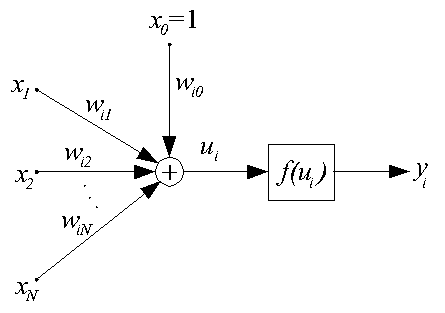


Рис. . Структурная схема сигмоидального нейрона.

В качестве функции активации выступает сигмоидальная функция (рис. 3). На практике используются как униполярные, так и биполярные функции активации.

Пусть – взвешенная сумма входных сигналов для -го нейрона в сети (в рамках текущей задачи рассматривается модель только одного нейрона, поэтому индекс опускается), которую можно записать как:

где – количество входов нейрона, – номер связи. Тогда униполярная функция активации сигмоидального нейрона записывается как:

где – коэффициент, определяющий «крутизну» функции (в рамках задачи полагается равным ).

В свою очередь производная униполярной функции активации выражается как:

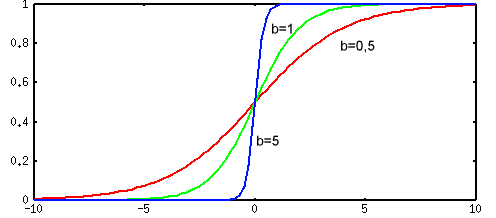


Рис. . Униполярная функция активации сигмоидального нейрона (b = ).

Для обучения сигмоидального нейрона используется стратегия «с учителем», которая подразумевает поиск минимума целевой функции, которая в случае режима обучения «оффлайн» (учитываются все обучающие выборки сразу) имеет вид:

где – количество обучающих выборок, – выходной сигнал нейрона для -й выборки, – ожидаемое значение выходного сигнала для -й выборки.

В отличие от персептрона, для поиска минимума целевой функции (4) используются методы поисковой оптимизации первого порядка, в которых целенаправленное изменение весовых коэффициентов осуществляется в направлении отрицательного градиента

Так, -й компонент вектора градиента целевой функции имеет вид:

В таком случае обучение производится посредством коррекции весов нейрона методом градиента по формуле:

где – коэффициент обучения из метода градиентного спуска.

Основными проблемами в реализации метода градиента являются:

* выбор стратегии определения коэффициента обучения ;
* выбор критерия окончания вычислительного процесса.

В простейшем варианте метода градиента коэффициент выбирается постоянной величиной из диапазона , а процесс поиска минимума завершается при выполнении условия

где – константа, определяющая точность отыскания минимума.

Однако, для эффективного обучения нейрона необходимы адаптивные стратегии подбора коэффициента обучения в ходе поиска экстремума.

Хорошие результаты демонстрирует следующая стратегия адаптации коэффициента :

1. В текущем -ом цикле обучения вычисляется новое значение вектора входных весов нейрона по стандартной формуле (6) и сравниваются значения целевой функции для и .
2. Если , то и счетчик числа подряд идущих удачных применений величины увеличивается на 1. Если этот счетчик превышает наперед заданную величину (например, 2), то значение удваивается, а счетчик сбрасывается в 0. Далее переход на следующую итерацию цикла (п. 1)
3. Если , то значение коэффициента уменьшаешься в 2 раза, а счетчик цикла подряд идущих удачных применений величины сбрасывается в 0. Далее переход к п.1
4. Процесс обучения завершается при условии , где – наперед заданная константа (например, 1E-3).

В разработанной программной реализации сигмоидального нейрона применен описанный выше алгоритм адаптивного выбора коэффициента обучения .

# Программная реализация

В рамках лабораторной работы был программно реализован трехвходовой сигмоидальный нейрон (один из входов является входом поляризации). Программный код представлен в Листингах 4 и 5 и представляет собой класс с публичными методами learn и test, производящими обучение нейрона в режиме оффлайн и тестирование нейрона соответственно.

Для настройки коэффициентов обучения был разработан механизм конфигурации с помощью текстового файла config.cfg (Листинг 7) без необходимости перекомпиляции исходного кода.

Так, значение устанавливается параметром LEARNING\_COEF\_EPS, а точность обучения устанавливается параметром TARGET\_FUNC\_EPS. В качестве значения было выбрано 1E-3.

# Обучение нейрона

Для генерации выборок в соответствии с заданием (рис. 1) были разработаны классы EllipseParams (Листинг 2) и EllipseDataGenerator (Листинги 3 и 6), позволяющие генерировать точки в кластерах эллипсов, параметры которых также задаются в конфигурационном файле config.cfg.

Так, результат генерации точек по рис. 1 представлен на рис. 4.

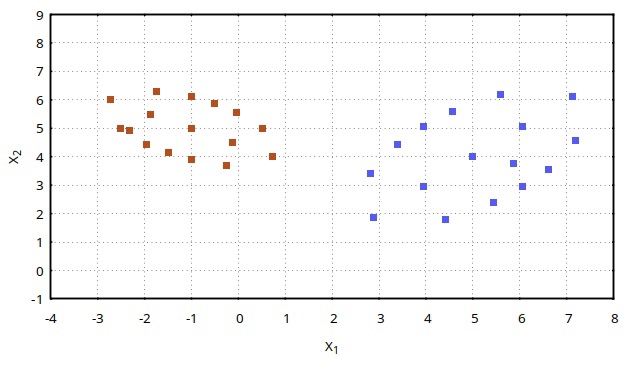


Рис. . Сгенерированные точки для обучающей выборки.

При этом начальные значения весов задаются случайным образом в интервале .

# Результаты работы программы

Для обучающей выборки, продемонстрированной на рис. 4 было проведено обучение в режиме «оффлайн» и получено разбиение на 2 кластера за 3 итерации. Демонстрация обучения с линией весов представлена на рис. 5.

Помимо этого, также была сформирована контрольная выборка точек внутри эллипсов и проведено тестирование, результаты которого для определенного количества точек представлено на рис. 6

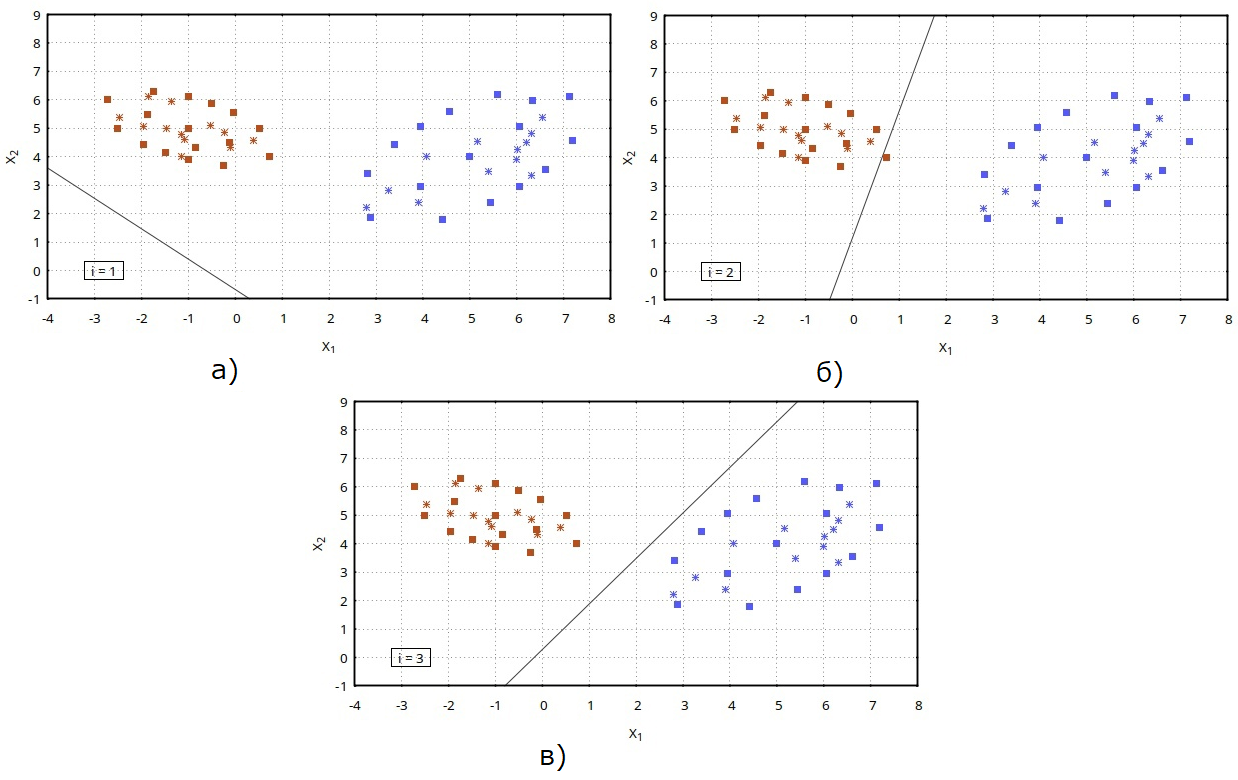


Рисунок . Процесс обучения нейрона на тестовой выборке. Черная линия – линия весов, демонстрирующая разделение данных на 2 кластера. а) 1-я итерация б) 2-я итерация в) 3-я, конечная итерация.

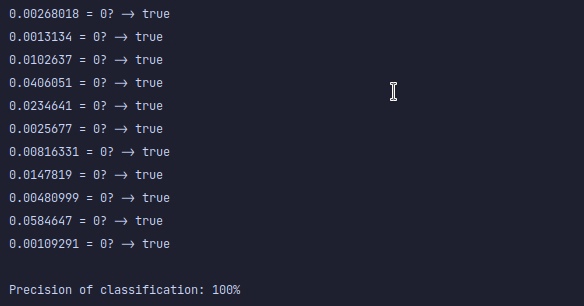


Рис. . Результаты тестирования обученного нейрона.

Сравнение значений весов для начального состояния нейрона и обученного представлено в таблице 1.

Таблица 1. Сравнение значений весов нейрона

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| Начальное значение весов | 0.55099 | 0.875552 | 0.820256 | 1 |
| Итоговое значение весов | 0.213148 | 1.20636 | -0.753643 | 3 |

# Текст программы

Листинг 1. Файл main.cpp

#include <vector>

#include <cmath>

#include <iostream>

#include "config.hpp"

#include "SigmoidalNeuron.hpp"

#include "EllipseDataGenerator.hpp"

#include "EllipseParams.hpp"

// Метод для конкатенации двух векторов

template<typename T>

std::vector<T> operator+(const std::vector<T>& v1, const std::vector<T>& v2){

std::vector<T> vr(std::begin(v1), std::end(v1));

vr.insert(std::end(vr), std::begin(v2), std::end(v2));

return vr;

}

int main(int argc, const char\*\* argv) {

ConfigurationSingleton& configuration = ConfigurationSingleton::getInstance();

EllipseParams ellipse1(configuration, "1");

EllipseParams ellipse2(configuration, "2");

// Генерация данных в эллипсах для обучения

EllipseDataGenerator ellipse1Generator(ellipse1);

EllipseDataGenerator ellipse2Generator(ellipse2);

int ellipse1PointsLearn = configuration.getVariableInt("ELLIPSE\_1\_LEARN\_POINTS");

int ellipse2PointsLearn = configuration.getVariableInt("ELLIPSE\_2\_LEARN\_POINTS");

int ellipse1PointsTest = configuration.getVariableInt("ELLIPSE\_1\_TEST\_POINTS");

int ellipse2PointsTest = configuration.getVariableInt("ELLIPSE\_2\_TEST\_POINTS");

std::vector<std::vector<double>> XLearn1 = ellipse1Generator.generateLearnData(ellipse1PointsLearn, "first\_ellipse");

std::vector<std::vector<double>> XLearn2 = ellipse2Generator.generateLearnData(ellipse2PointsLearn, "second\_ellipse");

std::vector<double> DLearn1(XLearn1.size(), 1.0);

std::vector<double> DLearn2(XLearn2.size(), 0.0);

std::vector<std::vector<double>> XLearn = XLearn1 + XLearn2;

std::vector<double> DLearn = DLearn1 + DLearn2;

EllipseDataGenerator::writeToFile(configuration.getVariable("LEARN\_DATA\_FILENAME"), XLearn, DLearn);

// Генерация данных в эллипсе для тестирования

std::vector<std::vector<double>> XTest1 = ellipse1Generator.generateRandomData(ellipse1PointsTest);

std::vector<std::vector<double>> XTest2 = ellipse2Generator.generateRandomData(ellipse2PointsTest);

std::vector<double> DTest1(XTest1.size(), 1.0);

std::vector<double> DTest2(XTest2.size(), 0.0);

std::vector<std::vector<double>> XTest = XTest1 + XTest2;

std::vector<double> DTest = DTest1 + DTest2;

EllipseDataGenerator::writeToFile(configuration.getVariable("TEST\_DATA\_FILENAME"), XTest, DTest);

// Создание сигмоидального нейрона, обучение, тестирование и переобучение по необходимости

SigmoidalNeuron neuron(3, configuration);

neuron.learn(XLearn, DLearn);

neuron.test(XTest, DTest);

return 0;

}

Листинг 2. Файл EllipseParams.hpp

#ifndef ELLIPSEPARAMS\_H

#define ELLIPSEPARAMS\_H

#include <string>

#include "config.hpp"

struct EllipseParams {

// Полуоси эллипса

double a;

double b;

// Координаты точки центра

double x0;

double y0;

// Угол поворота осей эллипса

double rotate;

EllipseParams() = default;

explicit EllipseParams(ConfigurationSingleton& configuration, std::string ellipseName) {

this->a = configuration.getVariableDouble("ELLIPSE\_" + ellipseName + "\_A");

this->b = configuration.getVariableDouble("ELLIPSE\_" + ellipseName + "\_B");

this->x0 = configuration.getVariableDouble("ELLIPSE\_" + ellipseName + "\_X0");

this->y0 = configuration.getVariableDouble("ELLIPSE\_" + ellipseName + "\_Y0");

this->rotate = configuration.getVariableDouble("ELLIPSE\_" + ellipseName + "\_ROTATE");

}

};

#endif // ELLIPSEPARAMS\_H

Листинг 3. Файл EllipseDataGenerator.hpp

#ifndef GENERATOR\_H

#define GENERATOR\_H

#include <string>

#include <vector>

#include <cstdlib>

#include <ctime>

#include <cmath>

#ifndef M\_PI

#define M\_PI 3.14159265358979323846

#endif

#include "EllipseParams.hpp"

class EllipseDataGenerator {

private:

EllipseParams ellipse;

// Генерация точек внутри эллипса

std::vector<std::vector<double>> generatePointsInsideEllipse(size\_t points\_num) const;

// Генерация точек по периметру эллипса

std::vector<std::vector<double>> generatePointsOnEllipse(size\_t points\_num) const;

// Получение точки эллипса в абсолютных координатах (сдвиг + поворот осей)

std::vector<double> pointToAbsoluteCoord(double xLocal, double yLocal) const;

public:

explicit EllipseDataGenerator(EllipseParams& \_ellipse) : ellipse(\_ellipse) {

srand(time(nullptr));

this->ellipse.rotate = this->ellipse.rotate \* M\_PI / 180.0;

};

// Сгенерерировать случайное число в интервале [l, r)

static double generateRandNumInRange(double l, double r);

static void writeToFile(const std::string& filename, const std::vector<std::vector<double>>& res);

static void writeToFile(const std::string& filename, const std::vector<std::vector<double>>& res, const std::vector<double>& D\_res);

static void plotDots(

std::vector<std::vector<double>>& data,

std::vector<double>& DLearn,

std::string dataFilename,

std::string gnuplotFilename

);

// Сгенерировать точки в эллипсе для обучения нейрона

std::vector<std::vector<double>> generateLearnData(size\_t points\_num, std::string filename) const;

std::vector<std::vector<double>> generateRandomData(size\_t points\_num) const;

~EllipseDataGenerator() = default;

};

#endif // GENERATOR\_H

Листинг 4. Файл SigmoidalNeuron.hpp

#ifndef NEURON\_H

#define NEURON\_H

#include <vector>

#include <fstream>

#include "config.hpp"

class SigmoidalNeuron {

private:

// Количество входов нейрона

size\_t inputsNumber;

double sigma;

std::vector<double> weights;

std::ofstream weightsFile;

ConfigurationSingleton& configuration;

double neuronOutput(std::vector<double>& X, std::vector<double>& extWeights);

double getUSum(std::vector<double>& X, std::vector<double>& extWeights);

double getDerivative(std::vector<double>& X);

double targetFunction(std::vector<double>& D, std::vector<double>& Y);

void appendWeightsToFile();

public:

explicit SigmoidalNeuron(int \_inputs\_num, ConfigurationSingleton& \_configuration);

void learn(

std::vector<std::vector<double>>& XLearn,

std::vector<double>& DLearn

);

bool test(std::vector<std::vector<double>>& XTest, std::vector<double>& DTest);

std::vector<double> getWeights() { return this->weights; };

~SigmoidalNeuron() = default;

};

#endif // NEURON\_H

Листинг 5. Файл SigmoidalNeuron.cpp

#include <string>

#include <iostream>

#include <chrono>

#include "SigmoidalNeuron.hpp"

#include "EllipseDataGenerator.hpp"

SigmoidalNeuron::SigmoidalNeuron(int \_inputsNumber, ConfigurationSingleton& \_configuration)

: inputsNumber(\_inputsNumber), configuration(\_configuration) {

this->sigma = configuration.getVariableDouble("SIGMA");

std::string weightsFilename = configuration.getVariable("WEIGHTS\_FILENAME");

this->weights.resize(inputsNumber);

this->weightsFile.open(weightsFilename);

if (!this->weightsFile.is\_open()) {

std::cerr << "Error: can't open file " << weightsFilename << std::endl;

throw std::runtime\_error(std::string("Error: can't open file ") + weightsFilename);

}

for (size\_t i = 0; i < inputsNumber; ++i) {

weights.at(i) = EllipseDataGenerator::generateRandNumInRange(0, 1);

}

std::cout << "Initial weights:" << std::endl;

this->appendWeightsToFile();

}

double SigmoidalNeuron::neuronOutput(std::vector<double>& X, std::vector<double>& extWeights) {

// Функция активации

return 1 / ( 1 + std::exp(- this->sigma \* this->getUSum(X, extWeights)) );

}

double SigmoidalNeuron::getUSum(std::vector<double>& X, std::vector<double>& extWeights) {

double u = 0;

// X0 - вход поляризации = 1

u += extWeights.at(0) \* 1.0;

for (size\_t i = 1; i < this->inputsNumber; ++i) {

u += X.at(i - 1) \* extWeights.at(i);

}

return u;

}

double SigmoidalNeuron::getDerivative(std::vector<double>& X) {

double output = this->neuronOutput(X, this->weights);

return this->sigma \* output \* (1 - output);

}

double SigmoidalNeuron::targetFunction(std::vector<double>& D, std::vector<double>& Y) {

double E = 0.0;

for (size\_t k = 0; k < D.size(); ++k) {

E += std::pow(Y.at(k) - D.at(k), 2.0);

}

return 0.5 \* E;

}

void SigmoidalNeuron::learn(

std::vector<std::vector<double>>& XLearn,

std::vector<double>& DLearn

) {

std::cout << "learning..." << std::endl;

std::vector<double> correctedWeights(this->weights.size());

double learningCoef = this->configuration.getVariableDouble("LEARNING\_COEF\_INIT");

size\_t successIterations = 0;

double currentTargetFunc = 1.0;

double learningCoefEPS = this->configuration.getVariableDouble("LEARNING\_COEF\_EPS");

double targetFuncEPS = this->configuration.getVariableDouble("TARGET\_FUNC\_EPS");

while (learningCoef > learningCoefEPS && currentTargetFunc > targetFuncEPS) {

for (size\_t j = 0; j < this->inputsNumber; ++j) {

double derivative = 0.0;

for (size\_t k = 0; k < XLearn.size(); ++k) {

double neuronOut = this->neuronOutput(XLearn.at(k), this->weights);

double xj = 1.0;

if (j > 0) {

xj = XLearn.at(k).at(j - 1);

}

// Вычисление компоненты градиента

derivative += (neuronOut - DLearn.at(k)) \* this->getDerivative(XLearn.at(k)) \* xj;

}

// коорекция веса

correctedWeights.at(j) = this->weights.at(j) - learningCoef \* derivative;

}

std::vector<double> prevE(DLearn.size());

std::vector<double> currE(DLearn.size());

for (size\_t k = 0; k < XLearn.size(); ++k) {

prevE.at(k) = this->neuronOutput(XLearn.at(k), this->weights);

currE.at(k) = this->neuronOutput(XLearn.at(k), correctedWeights);

}

double targetFuncCurr = this->targetFunction(DLearn, currE);

double targetFuncPrev = this->targetFunction(DLearn, prevE);

if (targetFuncCurr < targetFuncPrev) {

std::cout << "Successed iteration" << std::endl;

if (successIterations > 2) {

learningCoef \*= 2.0;

successIterations = 0;

} else {

++successIterations;

}

this->weights = correctedWeights;

currentTargetFunc = targetFuncCurr;

std::cout << "Corrected weights: " << std::endl;

this->appendWeightsToFile();

std::cout << "Current learning coefficient: " << learningCoef << std::endl;

std::cout << "Current target function value: " << currentTargetFunc << std::endl;

} else {

std::cout << "Failed iteration" << std::endl;

learningCoef /= 2.0;

successIterations = 0;

}

}

}

void SigmoidalNeuron::appendWeightsToFile() {

std::cout << "W = [ ";

for (size\_t i = 0; i < this->inputsNumber; ++i) {

this->weightsFile << this->weights.at(i) << " ";

std::cout << this->weights.at(i) << " ";

}

this->weightsFile << std::endl;

std::cout << "]" << std::endl << std::endl;

}

bool SigmoidalNeuron::test(std::vector<std::vector<double>>& XTest, std::vector<double>& DTest) {

std::cout << "testing..." << std::endl;

double testDiffEPS = this->configuration.getVariableDouble("TEST\_DIFF\_EPS");

size\_t matchesCount = 0;

for (size\_t i = 0; i < DTest.size(); ++i) {

double result = this->neuronOutput(XTest.at(i), this->weights);

std::cout << result << " = " << DTest.at(i) << "? -> ";

if (std::fabs(result - DTest.at(i)) < testDiffEPS) {

std::cout << "true";

++matchesCount;

} else {

std::cout << "false";

}

std::cout << std::endl;

}

std::cout << std::endl;

std::cout << "Precision of classification: " << (double) matchesCount / XTest.size() \* 100. << "%" << std::endl;

if (matchesCount != XTest.size()) {

return false;

} else {

return true;

}

}

Листинг 6. Файл EllipseDataGenerator.cpp

#include <fstream>

#include <string>

#include "EllipseDataGenerator.hpp"

#include "EllipseParams.hpp"

#include <cmath>

// Сгенерерировать случайное число в интервале [l, r)

double EllipseDataGenerator::generateRandNumInRange(double l, double r) {

double tmp = (double) std::rand() / (RAND\_MAX / (r - l));

return l + tmp;

}

std::vector<double> EllipseDataGenerator::pointToAbsoluteCoord(double xLocal, double yLocal) const {

EllipseParams ellipse = this->ellipse;

double x = ellipse.x0 + (xLocal \* std::cos(ellipse.rotate) - yLocal \* std::sin(ellipse.rotate));

double y = ellipse.y0 + (xLocal \* std::sin(ellipse.rotate) + yLocal \* std::cos(ellipse.rotate));

return std::vector<double>{x, y};

}

// Сгенерировать точки внутри эллипса

std::vector<std::vector<double>> EllipseDataGenerator::generatePointsInsideEllipse(size\_t points\_num) const {

EllipseParams ellipse = this->ellipse;

std::vector<std::vector<double>> res;

for (size\_t i = 0; i < points\_num; ++i) {

double rad = std::sqrt(generateRandNumInRange(0, 1));

double phi = 2 \* M\_PI \* generateRandNumInRange(0, 1);

double x\_e = rad \* std::cos(phi) \* ellipse.a;

double y\_e = rad \* std::sin(phi) \* ellipse.b;

res.push\_back(this->pointToAbsoluteCoord(x\_e, y\_e));

}

return res;

}

// Сгенерировать точки на эллипсе

std::vector<std::vector<double>> EllipseDataGenerator::generatePointsOnEllipse(size\_t points\_num) const {

EllipseParams ellipse = this->ellipse;

std::vector<std::vector<double>> result;

double stepAngle = 360.0 / points\_num;

for (size\_t i = 0; i < points\_num; ++i) {

double phi = i \* stepAngle \* M\_PI / 180.0;

// Переход от угла эллипса к углу образующих окружностей

double phiAbs = std::atan2((ellipse.a \* std::sin(phi)), (ellipse.b \* std::cos(phi)));

// Получение точки эллипса в локальных координатах

double xLocal = ellipse.a \* std::cos(phiAbs);

double yLocal = ellipse.b \* std::sin(phiAbs);

result.push\_back(this->pointToAbsoluteCoord(xLocal, yLocal));

}

return result;

}

// Генерация точек в области эллипса

std::vector<std::vector<double>> EllipseDataGenerator::generateLearnData(size\_t points\_num, std::string filename) const {

// Генерация точек по периметру эллипса

std::vector<std::vector<double>> points = generatePointsOnEllipse(points\_num);

// Генерация точек на главной оси эллипса (полуэллипс содержит главную ось)

std::vector<double> aBegin = points.at(0); // Начало полуэллипса (начало оси)

std::vector<double> aEnd = points.at(points\_num / 2); // Конец полуэллипса (конец оси)

// Центральная точка

points.push\_back({ (aBegin.at(0) + aEnd.at(0)) / 2, (aBegin.at(1) + aEnd.at(1)) / 2});

auto p3 = points.at(12);

// Точки по бокам от центра

double x\_1 = aBegin.at(0) + p3.at(0);

double y\_1 = aBegin.at(1) + p3.at(1);

double x\_2 = p3.at(0) + aEnd.at(0);

double y\_2 = p3.at(1) + aEnd.at(1);

points.push\_back({x\_1 / 2, y\_1 / 2});

points.push\_back({x\_2 / 2, y\_2 / 2});

// Добавить несколько точек внутри эллипса

std::vector<std::vector<double>> pointsInside = this->generatePointsInsideEllipse(1);

points.insert(points.end(), pointsInside.begin(), pointsInside.end());

std::string resultFilename = std::string(filename + "\_" + std::to\_string(this->ellipse.a) + "\_" + std::to\_string(this->ellipse.b) + ".dat");

EllipseDataGenerator::writeToFile(resultFilename, points);

return points;

}

std::vector<std::vector<double>> EllipseDataGenerator::generateRandomData(size\_t points\_num) const {

return this->generatePointsInsideEllipse(points\_num);

}

void EllipseDataGenerator::writeToFile(const std::string& filename, const std::vector<std::vector<double>>& res) {

std::ofstream file(filename);

for (auto & row : res) {

for (auto & el : row) {

file << el << " ";

}

file << std::endl;

}

}

void EllipseDataGenerator::writeToFile(const std::string& filename,

const std::vector<std::vector<double>>& res,

const std::vector<double>& D\_res) {

std::ofstream file(filename);

for (size\_t i = 0; i < res.size(); i++) {

for (auto & el : res.at(i)) {

file << el << " ";

}

file << D\_res.at(i) << std::endl;

}

}

Листинг 7. Файл config.cfg

ELLIPSE\_1\_A = 3.0

ELLIPSE\_1\_B = 1.5

ELLIPSE\_1\_X0 = 5.0

ELLIPSE\_1\_Y0 = 4.0

ELLIPSE\_1\_ROTATE = 45.0

ELLIPSE\_2\_A = 2.0

ELLIPSE\_2\_B = 1.0

ELLIPSE\_2\_X0 = -1.0

ELLIPSE\_2\_Y0 = 5.0

ELLIPSE\_2\_ROTATE = 150.0

ELLIPSE\_1\_LEARN\_POINTS = 12

ELLIPSE\_2\_LEARN\_POINTS = 12

ELLIPSE\_1\_TEST\_POINTS = 12

ELLIPSE\_2\_TEST\_POINTS = 12

SIGMA = 1

LEARNING\_COEF\_INIT = 0.5

LEARNING\_COEF\_EPS = 1e-3

TARGET\_FUNC\_EPS = 1e-2

TEST\_DIFF\_EPS = 1e-1

WEIGHTS\_FILENAME = weights.dat

LEARN\_DATA\_FILENAME = learn\_data.dat

TEST\_DATA\_FILENAME = test\_data.dat

Листинг 8. Файл gnuplot\_window.gp

reset

**set** wxt

color\_set1 = '#b54f1e'

color\_set2 = '#565af5'

color\_line = '#3e3c3d'

**set** border linewidth 1.5

**set** pointsize 1

**set** style line 1 lc rgb color\_line pt 7

**set** palette defined **(**0 color\_set1, 1 color\_set2**)**

**set** grid

**unset** key

**unset** colorbox

**set** tics scale 0.1

**set** xtics 1

**set** ytics 1

**set** yrange**[**-1:9**]**

**set** xrange**[**-4:8**]**

**set** xlabel 'X\_1'

**set** ylabel 'X\_2'

N = system**(**"wc -l weights.dat"**)**

**do** **for** **[**i=1:N**]** **{**

w\_0 = system**(**"sed -n " . i . "p weights.dat | awk '{print $1}'"**)**

w\_1 = system**(**"sed -n " . i . "p weights.dat | awk '{print $2}'"**)**

w\_2 = system**(**"sed -n " . i . "p weights.dat | awk '{print $3}'"**)**

LABEL = "i = " . i

**set** obj 10 rect at graph 0.1,0.1 **size** char strlen**(**LABEL**)**, char 1

**set** obj 10 fillstyle empty border -1 front

**set** label 10 at graph 0.1,0.1 LABEL front center

f**(**x**)** = -1**/(**w\_2**)** **\*** **(**w\_1 **\*** x + w\_0**)**

plot f**(**x**)** **w** l **ls** 1, 'learn\_data.dat' using 1:2:3 **w** p pt 5 lc palette z, 'test\_data.dat' using 1:2:3 **w** p pt 3 lc palette z

**}**