**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР**

отчет

**по лабораторной работе №1**

**по дисциплине «Основы искусственного интеллекта»**

Тема: Нейронная сеть Хопфилда

|  |  |  |
| --- | --- | --- |
| Студенты гр. 9302 |  | Ширнин К.В. |
|  |  | Квитко Д.В. |
| Преподаватель |  | Новакова Н.Е. |

Санкт-Петербург

2022

**Содержание**

[Цель работы 3](#_Toc120489069)

[Теоретические сведения 3](#_Toc120489070)

[Архитектура сети 3](#_Toc120489071)

[Обучение сети 4](#_Toc120489072)

[Особенности реализации алгоритма 6](#_Toc120489073)

[UML-диаграмма 6](#_Toc120489074)

[Ход работы 7](#_Toc120489075)

[Пример работы программы 9](#_Toc120489076)

[Вывод 12](#_Toc120489077)

[Список используемой литературы 12](#_Toc120489078)

[Листинг 13](#_Toc120489079)

# Цель работы

Приобретение и закрепление знаний, а также получение практических навыков работы с простейшими нейронными сетями, для обучения которых используется алгоритм Хопфилда

# Теоретические сведения

## Архитектура сети

Нейронная сеть Хопфилда устроена так, что её отклик на запомненные m эталонных «образов» составляют сами эти образы, а если образ немного исказить и подать на вход, он будет восстановлен и в виде отклика будет получен оригинальный образ. Таким образом, сеть Хопфилда осуществляет коррекцию ошибок и помех.

Сеть Хопфилда однослойная и состоит из N [искусственных нейронов](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D0%BA%D1%83%D1%81%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BD%D0%B5%D0%B9%D1%80%D0%BE%D0%BD). Каждый нейрон системы может принимать на входе и на выходе одно из двух состояний (что аналогично выходу нейрона с пороговой [функцией активации](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_%D0%B0%D0%BA%D1%82%D0%B8%D0%B2%D0%B0%D1%86%D0%B8%D0%B8)):

Из-за их биполярной природы нейронные сети Хопфилда иногда называют [спинами](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D0%BD).

Каждый нейрон связан со всеми остальными нейронами. Взаимодействие нейронов сети описывается выражением:

Где — элемент матрицы взаимодействий W, которая состоит из весовых коэффициентов связей между нейронами. В процессе обучения формируется выходная матрица W, которая запоминает m эталонных «образов» — N-мерных бинарных векторов:  , эти образы во время эксплуатации сети будут выражать отклик системы на входные сигналы, или иначе - окончательные значения выходов  после серии итераций.

В сети Хопфилда матрица связей является симметричной ,а диагональные элементы матрицы полагаются равными нулю , что исключает эффект воздействия нейрона на самого себя и является необходимым для сети Хопфилда, но не достаточным условием устойчивости в процессе работы сети. Достаточным является асинхронный режим работы сети. Подобные свойства определяют тесную связь с реальными физическими веществами, называемыми [спиновыми стёклами](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D0%BD%D0%BE%D0%B2%D1%8B%D0%B5_%D1%81%D1%82%D1%91%D0%BA%D0%BB%D0%B0).

Матрица взаимодействий хранится на самих нейронах в виде весов при связях нейронов с другими нейронами.

Так например, если входной сигнал определяется 10 параметрами, то нейронная сеть Хопфилда формируется из одного уровня с 10 нейронами. Каждый нейрон связывается со всеми остальными 9-ю нейронами, таким образом в сети образуется 90 (10 x 9) связей. Для каждой связи определяется весовой коэффициент .Все веса связей и образуют матрицу взаимодействий, которая заполняется в процессе обучения

## Обучение сети

Обучение сети заключается в том, что находятся веса матрицы взаимодействий так, чтобы запомнить m векторов (эталонных образов, составляющих "память" системы).

Вычисление коэффициентов основано на следующем правиле: для всех запомненных образов  матрица связи должна удовлетворять уравнению

поскольку именно при этом условии состояния сети  будут устойчивы — попав в такое состояние, сеть в нём и останется.

Запоминаемые векторы должны иметь бинарный вид. Расчёт весовых коэффициентов проводится по следующей формуле:

где N — размерность векторов, m — число запоминаемых выходных векторов, d— номер запоминаемого выходного вектора,  — i-я компонента запоминаемого выходного j-го вектора.

Это выражение может стать более ясным, если заметить, что весовая матрица W может быть найдена вычислением внешнего произведения каждого запоминаемого вектора с самим собой и суммированием матриц, полученных таким образом. Это может быть записано в виде

где  — i-й запоминаемый вектор-столбец.

Расчёт этих весовых коэффициентов и называется обучением сети, которое проводится только за одну эпоху.

# Особенности реализации алгоритма

Для реализации алгоритма Хопфилда ПО было разделено на несколько составляющих. Была встроен собственный класс матриц, класс алгоритма и формы. Программный комплекс был написан в рамках правил ООП и без особых трудностей может использоваться как отдельный виджет в других ПО или как библиотека.

# UML-диаграмма

На рисунке ниже представлена UML-диаграмма структуры программы (Рисунок 1)

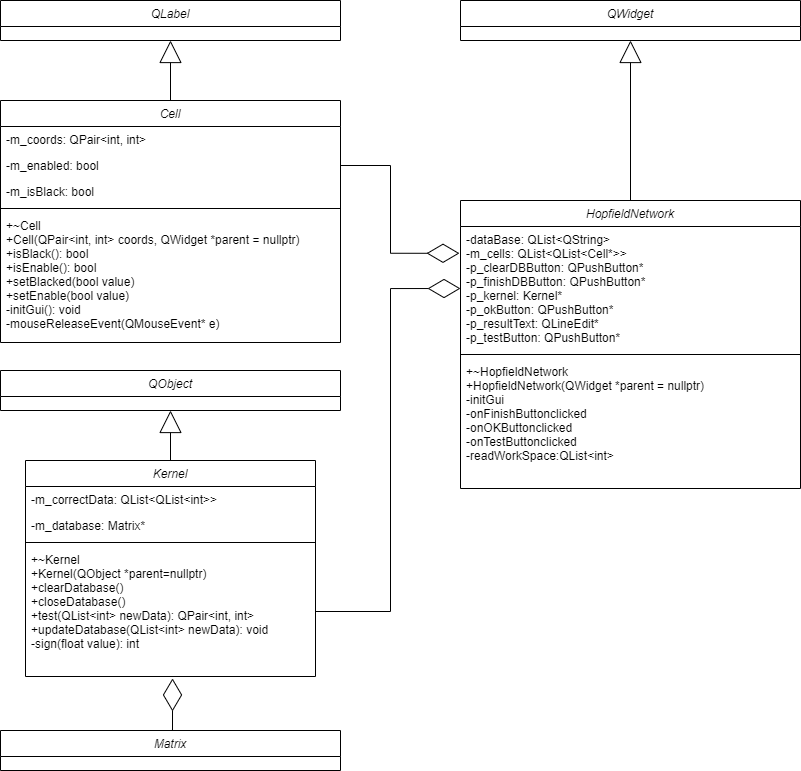


Рисунок 1 - UML-диаграмма

Таблица 1. Методы и их описание

|  |  |
| --- | --- |
| Метод | Описание |
| Kernel(QObject \*parent=nullptr) | Конструктор класса Kernel |
| ~Kernel() | Деструктор класса Kernel |
| void updateDatabase(QList<int> newData) | Обновляет базу данных для обучения |
| void closeDatabase() | Сформировывает конечную матрицу для обучения |
| void clearDatabase() | Очищает матрицу “обученной” сети |
| QPair<int, int> test(QList<int> newData) | Вызов алгоритма |
| int sign(float value) | Функция активации |

# Ход работы

Для выполнения данной лабораторной работы был выбран язык C++. Пример главного окна представлен на рисунке ниже (Рисунок 2)

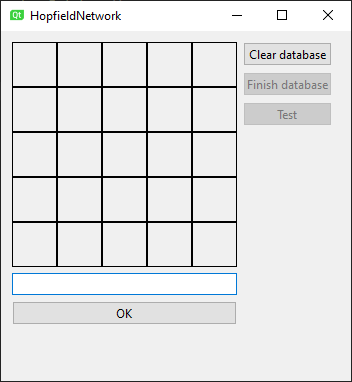


Рисунок 2 - Главное окно приложения

Для нейронной сети Хопфилда нам потребуется работа с матрицами. Для этого воспользуемся реализованным ранее классом Matrix. Для возможности рисования букв создадим поле, состоящее из собственно созданных ячеек Cell. Cell – разработанный нами виджет, который унаследован от встроенного класса QLabel. В класс виджета Cell был добавлен функционал смены цвета при нажатии на него. Сам алгоритм Хопфилда был вынесен в отдельный файл, что позволяет использовать его как независимую библиотеку в других программах и проектах. (Для этого потребуется скомпилировать его как dll библиотеку). Реализация GUI вынесена также в отдельный файл и спроектирована так, чтобы разработку можно было использовать в качестве внешней независимой библиотеки виджетов фреймворка Qt.

# Пример работы программы

Главное окно представлен на рисунке ниже (Рисунок 2)

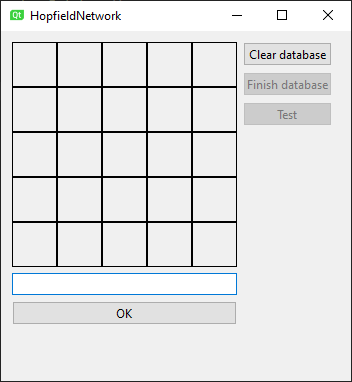


Рисунок 3 - Главное окно приложения

Пример обучения нейронной сети представлен на рисунке ниже

(Рисунок 4)

|  |  |
| --- | --- |
|  |  |
|  |  |

Рисунок 4 – Пример обучения нейронной сети

Пример работы алгоритма представлен на рисунке ниже (Рисунок 5)

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

Рисунок 5 – Пример работы алгоритма

# Вывод

В ходе выполнения работы стало ясно, что алгоритм Хопфилда не всегда выдает ожидаемые ответы. Но не смотря на подобные недостатки, подобный подход является очень простым и “дешевым” с точки зрения времени выполнения вычислений, что является существенным плюсом для разработок, где высокая скорость распознавания букв/образов является более важной задачей, чем точность. Алгоритм был успешно реализован на языке C++ на фреймворке Qt. В лабораторной работе были достигнуты все цели.

# Список используемой литературы

1. Нейронная сеть Хопфилда// Хабр URL: https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%B9%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F\_%D1%81%D0%B5%D1%82%D1%8C\_%D0%A5%D0%BE%D0%BF%D1%84%D0%B8%D0%BB%D0%B4%D0%B0 (дата обращения: 21.10.2022).

# Листинг

#include "kernel.h"

Kernel::Kernel(QObject \*parent)

: QObject(parent)

{

m\_database = new MathLogic::Matrix(25, 25);

}

Kernel::~Kernel()

{

delete m\_database;

}

void Kernel::updateDatabase(QList<int> newData)

{

m\_correctData.append(newData);

MathLogic::Matrix a(25, 1);

MathLogic::Matrix b(1, 25);

for (int i = 0; i < 25; ++i)

{

a.valueOf(i, 0) = newData[i];

b.valueOf(0, i) = newData[i];

}

\*m\_database = \*m\_database + (a \* b);

}

void Kernel::closeDatabase()

{

\*m\_database = \*m\_database / 25;

for (int i = 0; i < 25; ++i)

{

m\_database->valueOf(i, i) = 0;

}

}

void Kernel::clearDatabase()

{

m\_correctData.clear();

m\_database->clear();

}

QPair<int, int> Kernel::test(QList<int> newData)

{

MathLogic::Matrix a(25, 1);

MathLogic::Matrix b(25, 1);

for (int i = 0; i < 25; ++i)

a.valueOf(i, 0) = newData[i];

b = \*m\_database \* a;

for (int i = 0; i < 25; ++i)

a.valueOf(i, 0) = sign(a.valueOf(i, 0));

int coincidence = 0;

int theBestData = -1;

int theBestCoincidence = 0;

for (int i = 0; i < m\_correctData.count(); ++i)

{

for (int j = 0; j < m\_correctData[i].count(); ++j)

{

if (a.valueOf(j, 0) == m\_correctData[i][j])

++coincidence;

}

if (coincidence > theBestCoincidence)

{

theBestCoincidence = coincidence;

theBestData = i;

}

coincidence = 0;

}

if (theBestCoincidence < 18)

return QPair<int, int>(2, -1);

if (theBestCoincidence < 25)

return QPair<int, int>(1, theBestData);

return QPair<int, int>(0, theBestData);

}

int Kernel::sign(float value)

{

return value > 0 ? 1 : -1;

}