РЕФЕРАТ

Общий объем 44 страницы, 16 рисунков, 5 источников.

Ключевые слова: искусственные нейронные сети, сеть Кохена, многослойный персептрон, алгоритм самообучения, алгоритм обратного распространения ошибки.

Объектом исследования данной выпускной квалификационной работы является анализ методов моделирования искусственных нейронных сетей, построение алгоритма выполнения программы, позволяющей построить искусственную нейронную сеть, а также разработка самой программы.

Цель работы – изучение возможности применения искусственной нейронной сети для выполнения идентификации прибора, на основе его характерной битовой последовательности.

В процессе работы проанализированы существующие способы классификации объектов, а так рассмотрены популярные и эффективные методы кластеризации данных.

В результате работы была создана система идентификации приборов по их показаниям, основанная на использовании искусственной нейронной сети.

Программная реализация выполнена в среде программирования «PyCharm 3.4.1». Программа написана на языке «Python».

СОДЕРЖАНИЕ

[РЕФЕРАТ 2](#_Toc466484898)

[СОДЕРЖАНИЕ 3](#_Toc466484899)

[ВВЕДЕНИЕ 4](#_Toc466484900)

[1 Анализ построения искусственной нейронной сети 5](#_Toc466484901)

[1.1 Этапы решения задач 8](#_Toc466484902)

[1.1.1 Сбор данных для обучения 9](#_Toc466484903)

[1.1.2 Выбор топологии сети 10](#_Toc466484904)

[1.1.3 Экспериментальный подбор характеристик сети 11](#_Toc466484905)

[1.1.4 Экспериментальный подбор параметров обучения 12](#_Toc466484906)

[1.1.5 Обучение сети 12](#_Toc466484907)

[1.1.6 Проверка адекватности обучения 14](#_Toc466484908)

[2 Разработка алгоритма решения задачи, процедур и функций 15](#_Toc466484909)

[2.1 Описание основного алгоритма программы 15](#_Toc466484910)

[2.2 Описание алгоритма обучение сети 16](#_Toc466484911)

[2.3 Описание алгоритма функции тренировки сети 17](#_Toc466484912)

[2.4 Описание алгоритма получения и обработки данных 18](#_Toc466484913)

[3 Разработка программы формирования нейронной сети 20](#_Toc466484914)

[3.1 Постановка задачи 20](#_Toc466484916)

[3.2 Общие сведения о программе 20](#_Toc466484917)

[3.3 Описание программы 20](#_Toc466484918)

[4 Контрольный пример 21](#_Toc466484919)

[5 Отладка и тестирование 25](#_Toc466484920)

[ЗАКЛЮЧЕНИЕ 30](#_Toc466484921)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 32](#_Toc466484922)

[ПРИЛОЖЕНИЕ А –ТЕКСТ ПРОГРАММЫ НА ИСХОДНОМ ЯЗЫКЕ 33](#_Toc466484923)

ВВЕДЕНИЕ

Актуальность …

Искусственные нейронные сети используются для решения таких задач, как распознавание образов и классификация, принятие решений и управление, кластеризация, прогнозирование, аппроксимация, сжатие данных и ассоциативная память.

В этой выпускной квалификационной работе ставятся задача идентификации микросхем, для выполнения которой необходимо:

* провести анализ методов построения искусственных нейронных сетей;
* построить алгоритм построения, обучения искусственной нейронной сети;
* разработать реализацию алгоритма построения, обучения, искусственной нейронной сети, позволяющую проводить идентификацию устройств;
* провести тестирование программы для идентификации приборов;
* провести качественную оценку полученных результатов и их характеристик.

Таким образом, целью данной научно-практической работы является анализ методов построения искусственных нейронных сетей, построение алгоритма идентификации микросхем, разработка программы, для реализации такого алгоритма.

1 Анализ построения искусственной нейронной сети

Искусственные нейронные сети строятся по принципам организации и функционирования их биологических аналогов. Такие сети предназначены для решения широкого круга задач: распознавание образов, идентификации, прогнозирования, оптимизации, управление сложными объектами. Дальнейшее повышение производительности компьютеров все в большей мере связывают с искусственными нейронными сетями, в частности, с нейрокомпьютерами, основу которых составляет искусственная нейронная сеть. Основные проблемы, решаемые искусственными нейронными сетями: классификация образов, кластеризация, аппроксимация, предсказание-прогноз, оптимизация.

В рамках поставленной задачи, следует рассмотреть задачу классификации образов. Задача состоит в указании принадлежности входного образа, представленного вектором признаков, одному или нескольким предварительно определенным классам.

Искусственная нейронная сеть представляют собой систему соединённых и взаимодействующих между собой простых процессоров (искусственных нейронов). Каждый искусственный нейрон характеризуется своим текущим состоянием по аналогии с нервными клетками мозга, которые могут быть возбуждены или заторможены. Искусственный нейрон обладает группой синапсов – однонаправленных входных связей, соединенных с выходами других нейронов, а также имеет аксон – выходную связь данного нейрона, с которой сигнал поступает на синапсы следующих нейронов. Общий вид искусственного нейрона представлен на рисунке 1.

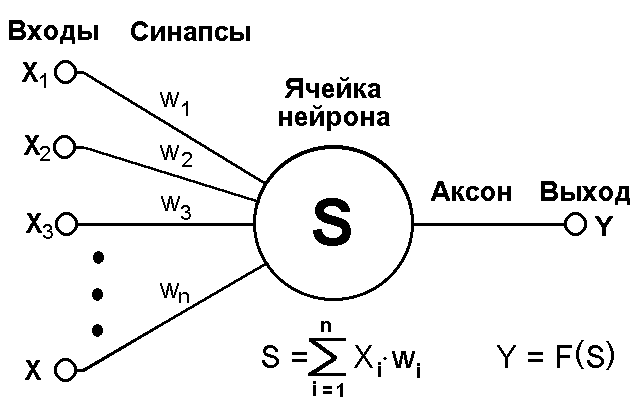


Рисунок 1 – Модель искусственного нейрона.

Здесь множество входных сигналов, обозначенных x1, x2, … xn, поступает на искусственный нейрон. Эти входные сигналы, в совокупности обозначенные вектором X, соответствуют сигналам, приходящим в синапсы биологического нейрона. Каждый синапс характеризуется величиной синапсической связи или ее весом wi. Каждый сигнал умножается на соответствующий вес w1, w2, … wn, и поступает на суммирующий блок. Каждый вес соответствует «силе» одной биологической синапсической связи. Множество весов в совокупности обозначается вектором W. Суммирующий блок, соответствующий телу биологического элемента, складывает взвешенные входы алгебраически, создавая величину S. Таким образом, текущее состояние нейрона определяется как взвешенная сумма его входов:

(1)

Выход нейрона есть функция его состояния:

, (2)

где f – активационная функция, более точно моделирующая нелинейную передаточную характеристику биологического нейрона и предоставляющая нейронной сети большие возможности. Наиболее распространенными являются пороговая:

(3)

и сигмоидальная:

, (4)

активационные функции.

Логистическая функция непрерывно заполняет своими значениями диапазон от нуля до единицы; параметр a всегда положителен. При уменьшении параметра a график становится более пологим, в пределе при a равном нулю вырождаясь в горизонтальную линию на уровне одной второй, при увеличении параметра a график сигмоида приближается к виду функции единичного скачка с порогом 0. Следует отметить, что сигмоидальная функция дифференцируема на всей оси абсцисс, что используется в некоторых алгоритмах обучения. Кроме того, она обладает свойством усиливать слабые сигналы и предотвращает насыщение от больших сигналов, так как они соответствуют тем областям аргументов, где сигмоид имеет пологий наклон.

Хотя один нейрон и способен выполнять простейшие процедуры распознавания, сила нейронных вычислений проистекает от соединений нейронов в сетях. Простейшая сеть состоит из группы нейронов, образующих слой, как представлено на рисунке 2.

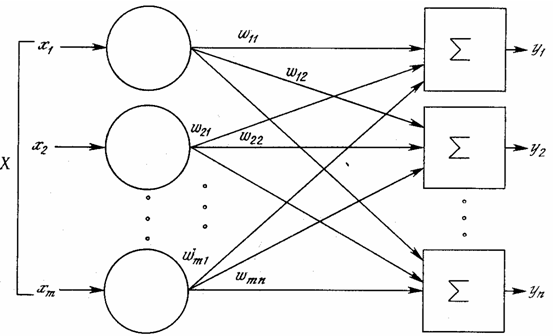


Рисунок 2 – Простейшая однослойная нейронная сеть.

Стоит отметить, что вершины-круги слева служат лишь для распределения входных сигналов. Они не выполняют каких-либо вычислений, и поэтому не будут считаться слоем. По этой причине они обозначены кругами, чтобы отличать их от вычисляющих нейронов, обозначенных квадратами. Каждый элемент из множества входов X соединён с каждым искусственным нейроном отдельной связью, которой приписан вес. А каждый нейрон выдает взвешенную сумму входов в сеть. В искусственных и биологических сетях многие соединения могут отсутствовать, на рисунке все соединения показаны в целях общности. Могут иметь место также соединения между выходами и входами элементов в слое. Удобно считать веса элементами матрицы W. Матрица имеет m строк и n столбцов, где m – число входов, а n – число нейронов. Например, w23 – это вес связи второго входа с третьим нейроном. Таким образом, вычисление выходного вектора Y, компонентами которого являются выходы yi нейронов, сводится к матричному умножению Y = X\*W.

Нейронные сети не программируются в привычном смысле этого слова, они обучаются. Возможность обучения – одно из главных преимуществ нейронных сетей перед традиционными алгоритмами. Технически обучение заключается в нахождении коэффициентов связей между нейронами. В процессе обучения нейронная сеть способна выявлять сложные зависимости между входными данными и выходными, а также выполнять обобщение. Это значит, что в случае успешного обучения сеть сможет вернуть верный результат на основании данных, которые отсутствовали в обучающей выборке, а также неполных и/или «зашумленных», частично искаженных данных.

* 1. Этапы решения задач

Задача классификации с использованием искусственной нейронной сети состоит из следующих этапов:

* сбор данных для обучения;
* подготовка и нормализация данных;
* выбор топологии сети;
* экспериментальный подбор характеристик сети;
* экспериментальный подбор параметров обучения;
* обучение сети на основе исходных данных;
* проверка адекватности обучения;
* корректировка параметров, окончательное обучение;
* вербализация сети с целью дальнейшего использования.

После выполнения данных действий осуществляется разработка алгоритма идентификации.

* + 1. Сбор данных для обучения

Выбор данных для обучения сети и их обработка является самым сложным этапом решения задачи. Набор данных для обучения должен удовлетворять нескольким критериям:репрезентативность — данные должны иллюстрировать истинное положение вещей в предметной области;непротиворечивость — противоречивые данные в обучающей выборке приведут к плохому качеству обучения сети. Исходные данные преобразуются к виду, в котором их можно подать на входы сети. Каждая запись в файле данных называется обучающей парой или обучающим вектором. Обучающий вектор содержит по одному значению на каждый вход сети и, в зависимости от типа обучения (с учителем или без), по одному значению для каждого выхода сети. Обучение сети на «сыром» наборе, как правило, не даёт качественных результатов. Существует ряд способов улучшить «восприятие» сети.

Нормировка выполняется, когда на различные входы подаются данные разной размерности. Например, на первый вход сети подаются величины со значениями от нуля до единицы, а на второй — от ста до тысячи. При отсутствии нормировки значения на втором входе будут всегда оказывать существенно большее влияние на выход сети, чем значения на первом входе. При нормировке размерности всех входных и выходных данных сводятся воедино.

Квантование выполняется над непрерывными величинами, для которых выделяется конечный набор дискретных значений. Например, квантование используют для задания частот звуковых сигналов при распознавании речи.

Фильтрация выполняется для «зашумленных» данных.

Кроме того, большую роль играет само представление как входных, так и выходных данных. Предположим, сеть обучается распознаванию букв на изображениях и имеет один числовой выход — номер буквы в алфавите. В этом случае сеть получит ложное представление о том, что буквы с номерами 1 и 2 более похожи, чем буквы с номерами 1 и 3, что, в общем, неверно. Для того, чтобы избежать такой ситуации, используют топологию сети с большим числом выходов, когда каждый выход имеет свой смысл. Чем больше выходов в сети, тем большее расстояние между классами и тем сложнее их спутать.

* + 1. Выбор топологии сети

Выбирать тип сети следует, исходя из постановки задачи и имеющихся данных для обучения. Для обучения с учителем требуется наличие для каждого элемента выборки «экспертной» оценки. Иногда получение такой оценки для большого массива данных просто невозможно. В этих случаях естественным выбором является сеть, обучающаяся без учителя (например, самоорганизующаяся карта Кохонена[2] или нейронная сеть Хопфилда). При решении других задач (таких, как прогнозирование временных рядов) экспертная оценка уже содержится в исходных данных и может быть выделена при их обработке. В этом случае можно использовать многослойный перцептрон или сеть Ворда.

Для решения поставленной задачи классификации следует использовать совмещенные подходы сети Кохонена и сети Ворда.

Сети Кохонена эффективно решают задачу классификации по принципу «победитель забирает все». Применение целесообразно при обучении без учителя на основе самоорганизации, также при неизвестных векторах-прототипов. В нашем случае, в рамках поставленной задачи можно задать жесткое соответствие между входными (вектор-признаками) и выходными (вектор-прототипами) значениями. Поэтому следует использовать лишь часть подхода при построении сети, а именно архитектуру сети Кохонена. Сеть состоит из одного слоя нейронов. Число входов каждого нейрона n равно размерности векторам параметров объекта. Количество нейронов m совпадает с требуемым числом классов, на которые нужно разбить объекты (меняя число нейронов, можно динамически менять число классов).

Так как нам заранее известно какими должны быть вектора на выходе сети, то целесообразно будет использовать не алгоритм обучения сети Кохонена, а алгоритм обучения сети обратного распространения.

* + 1. Экспериментальный подбор характеристик сети

После выбора общей структуры нужно экспериментально подобрать параметры сети. Для сетей, подобных перцептрону, это будет число слоев, число блоков в скрытых слоях (для сетей Ворда), наличие или отсутствие обходных соединений, передаточные функции нейронов. При выборе количества слоев и нейронов в них следует исходить из того, что способности сети к обобщению тем выше, чем больше суммарное число связей между нейронами. С другой стороны, число связей ограничено сверху количеством записей в обучающих данных.

Настоящая сеть будет иметь вход длиной в 128 бит, выход сети будет определяться динамически, в зависимости от числа требуемых для определения приборов, длина которого будет равна количеству бит представленного в бинарном виде числа идентифицируемых объектов; так, для определения 100 устройств необходим выход длиной в 7 бит.

* + 1. Экспериментальный подбор параметров обучения

После выбора конкретной топологии необходимо выбрать параметры обучения нейронной сети. Этот этап особенно важен для сетей, обучающихся с учителем. От правильного выбора параметров зависит не только то, насколько быстро ответы сети будут сходиться к правильным ответам. Например, выбор низкой скорости обучения увеличит время схождения, однако иногда позволяет избежать паралича сети. Увеличение момента обучения может привести как к увеличению, так и к уменьшению времени сходимости, в зависимости от формы поверхности ошибки. Исходя из такого противоречивого влияния параметров, можно сделать вывод, что их значения нужно выбирать экспериментально, руководствуясь при этом критерием завершения обучения (например, минимизация ошибки или ограничение по времени обучения), по средствам проведения анализа статистических результатов. В качестве критерия выступает установленный порог ошибки E, либо максимальное количество итераций обучения сети.

* + 1. Обучение сети

Нейронные сети обратного распространения – это современный инструмент поиска закономерностей, прогнозирования, качественного анализа. Такое название – сети обратного распространения – они получили из-за используемого алгоритма обучения, в котором ошибка распространяется от выходного слоя к входному, т.е. в направлении, противоположенном направлению распространения сигнала при нормальном функционировании сети.

Нейронная сеть обратного распространения состоит из нескольких слоев, однако будет использоваться один слой нейронов. Матрицу весовых коэффициентов от входов к выходному слою обозначим W. Для индексов примем следующие обозначения: входы будем нумеровать только индексом i, выходы – индексом j. Число входов сети равно n, число нейронов в выходном слое – m. Пусть сеть обучается на выборке (Xt,Dt), где t = 1,2, …, T – количество объектов идентификации, X – вектор-признаки, D – вектор-прототипы.

При обучении нейронной сети ставится задача минимизации целевой функции ошибки, которая находится по методу наименьших квадратов:

, (5)

Следует отметить, что возведение разности в квадрат можно заменить вычислением модуля разности. Разница лишь в том, что при нахождении модуля потребуется установить большее значение порога допустимой ошибки, в итоге функция примет вид:

(6)

где yj – полученный результат на выходе сети для сравнения с вектор-прототипом dj.

Значение порога допустимой ошибки eps, при обучении исчисляется следующей формулой:

. (7)

Вследствие динамического формирования выходов сети, величина допустимой ошибки прямо пропорциональна количеству нейронов на выходе сети.

Рассмотрим теперь полный алгоритм обучения нейросети:

Шаг 1. Инициализация сети. Весовым коэффициентам присваиваются случайные значения в диапазоне (-0,3;0,3); задаются eps – параметр точности обучения, Nmax – максимальное число итераций при обучении.

Шаг 2. Вычисление текущего выходного сигнала. На вход сети подается один из образов обучающей выборки и определяются значения выходов всех нейронов сети.

Шаг 3. Настройка синапсичских весов. Текущие значения весов изменяются на случайную величину из диапазона (-0,015;0,015).

Шаг 4. Шаги 2-3 повторяются для всех обучающих векторов. Обучение завершается по достижении для каждого из обучающих образов значения функции ошибки, не превосходящего eps или после максимального допустимого числа итераций.

* + 1. Проверка адекватности обучения

Даже в случае успешного, на первый взгляд, обучения сеть не всегда обучается именно тому, чего от неё хотел создатель. Известен случай, когда сеть обучалась распознаванию изображений танков по фотографиям, однако позднее выяснилось, что все танки были сфотографированы на одном и том же фоне. В результате сеть «научилась» распознавать этот тип ландшафта, вместо того, чтобы «научиться» распознавать танки. Таким образом, сеть «понимает» не то, что от неё требовалось, а то, что проще всего обобщить.

Тестирование качества обучения нейросети необходимо проводить на примерах, которые не участвовали в её обучении. При этом число тестовых примеров должно быть тем больше, чем выше качество обучения. Если ошибки нейронной сети имеют вероятность близкую к одной миллиардной, то и для подтверждения этой вероятности нужен миллиард тестовых примеров. Получается, что тестирование хорошо обученных нейронных сетей становится очень трудной задачей.

При обучении сети, на вход подавались моделированные вектора из множества обучающей выборки X, сгенерированной на основе векторов вероятностей приборов. При тестировании моделируются 100 новых векторов тестируемой выборки X’, на основе тех же векторов вероятностей приборов, для которых проводилось обучение. Количество правильно идентифицируемых приборов отображает эффективность работы сети в процентном соотношении. Что отображает качественную оценку функционирования сети.

1. Разработка алгоритма решения задачи, процедур и функций

Идентификация микросхемы производится следующим образом, представленном на рисунке :

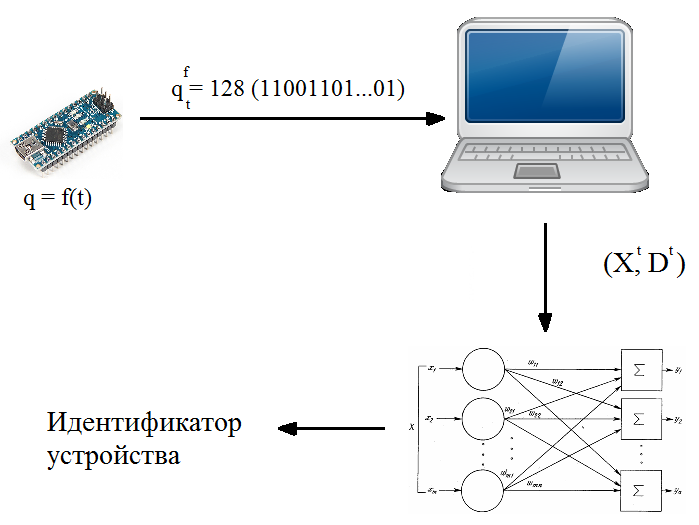


Рисунок – Общая схема процесса идентификации

Показания микросхемы представлены в виде:

|  |  |
| --- | --- |
| , | () |

где f – искажение сигнала устройства воздействием внешних факторов t (например, температура окружающей среды, влажность воздуха). Данные с микросхем поступают в компьютер в виде последовательности 128 бит (например, 11001101…01); затем компьютер формирует обучающую выборку из X – вектор-признаков и D – вектор-прототипов, после чего эта выборка используется искусственной нейронной сетью для обучения и последующего использования, с целью идентификации объектов; по выполнению описанного выше получается идентификатор устройства (при непосредственном использовании для идентификации).

Программа идентификации реализует следующие алгоритмы.

* 1. Описание общего алгоритма программы

Разработанный алгоритм, позволяющий формировать и обучать искусственную нейронную сеть, представлен на рисунке 3:

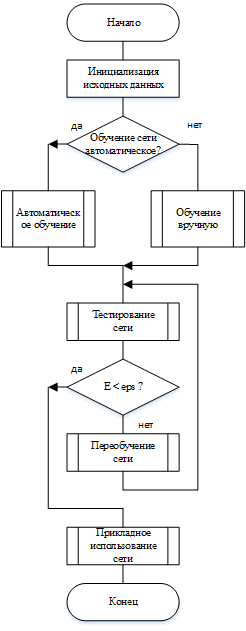


Рисунок 3 – Блок-схема алгоритма построения искусственной нейронной сети

В качестве входных данных представленного алгоритма выступает множество последовательностей q (на основе которых формируются данные для обучения сети), полученных от микросхем. На стадии прикладного использования искусственная нейронная сеть готова к применению идентификации. В качестве выходных данных выступает массив индексов микросхем.

Обучение вручную означает ввод с клавиатуры всех параметров для конфигурации сети, а также всех входных данных. Автоматическое обучение определяет часть параметров динамически и позволяет вводить данные микросхем путём считывания с COM-порта.

Тестирование сети заключается в вычислении выходного вектора сети, для его сравнения с вектор-прототипом, и последующим сравнением порога ошибки eps с вычисленной ошибкой для проверки сети.

Проверка сети представляет собой вычисление функции ошибки:

, (8)

где di и yi вектор-прототип и вектор, получившийся на выходе сети, соответственно. Однако при обучении будет учитываться не сумма из i элементов вектора, а максимальное значение ошибки среди элементов вектора.

Переобучение сети состоит в: модификации весовых коэффициентов; расчёт нового значения yi`; вычисление нового значения ошибки e`; сравнение e и e`.

* 1. Описание алгоритма обучение сети

Для обучения сети разработан алгоритм функции обучения сети для заданных вектор-признаков и соответствующих им вектор-прототипов, представленный на рисунке 4:

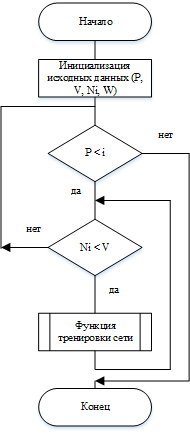


Рисунок 4 – Блок-схема алгоритма обучения сети

На представленном рисунке: P – количество повторов обучения (для закрепления обучения); Ni – количество вектор-признаков от каждого прибора; V – общее число вектор-признаков, участвующих в обучении, W – матрица весовых коэффициентов.

Для того чтобы обучить сеть для всех имеющихся вектор-признаков, необходимо применить функцию тренировки сети для каждого вектор-признака.

* 1. Описание алгоритма функции тренировки сети

Для того чтобы тренировать сеть для каждого вектора, был построен алгоритм функции тренировки вектора (который соответствует одной «эпохе обучения»), представленный на рисунке 6.

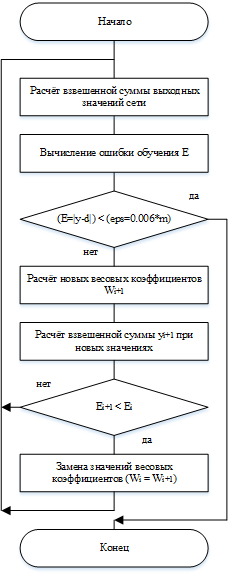


Рисунок 6 – блок-схема алгоритма тренировки сети для вектор-признака

В представленном алгоритме: E – значение вычисленной ошибки обучения; eps – значение допустимого параметра E; Ei и Ei+1 – значения вычисленных ошибок на текущей и следующей итерациях, соответственно. Замена значений весовых коэффициентов означает приравнивание Wi к Wi+1 и производится при условии, что значение ошибки обучения в текущей итерации меньше, чем на новой итерации, т.е. при условии успешной итерации обучения. Эпоха обучения – один полный проход по обучающей выборке.

* 1. Описание алгоритма получения и обработки данных

Прежде всего, необходимо инициализировать исходные данные программы. Существуют два способа: искусственное моделирование для тестирования сети и ведения статистики; загрузка параметров устройств из внешнего источника, с последующим прикладным использованием. Каждой последовательности из 128 бит присваивается уникальный индекс.

В рамках поставленной задачи требуется идентифицировать приборы по их показаниям. Объект выдает показания q+e, где q – уникальные показания прибора, а e – случайная ошибка. При обучении и функционировании сети q неизвестно. Показания прибора представляют собой бинарную последовательность 128 бит. Для моделирования показаний приборов был разработан следующий алгоритм.

Шаг 1. Генерируется последовательность случайных вещественных чисел, длиной 128, в диапазоне от 0 до 1. Каждая такая последовательность соответствует уникальному устройству прибора.

Шаг 2. К каждому биту последовательности применяется функция нормального распределения вероятностей, параметр среднеквадратичного отклонения составляет 0,1. В результате получается последовательность вещественных чисел в диапазоне от 0 до 1, эта новая последовательность соответствует только одному прибору.

Шаг 3. Над полученной последовательностью производится нормировка – округление до ближайшего целого. В итоге получена смоделированная бинарная последовательность длиной 128, которая соответствует показаниям настоящих приборов.

Так получаются смоделированные вектор-признаки для обучения сети. После накопления достаточного количества векторов обучающей выборки (их число равно количеству устройств, умноженному на пять) задействуется этап обучения сети, алгоритм обучения уже описан выше.

Поскольку входной вектор будет состоять из единиц и нулей, то выходные вектора также будут бинарными, а передаточная, или активационная функция сигмоидальная, так как сглаживание суммы происходит в диапазоне от 0 до 1, что в свою очередь довольно просто нормировать в бинарный вектор. На выходе сети получается бинарный вектор, после формализации которого, определяется индекс идентифицированного устройства, что и требовалось получить.

При прикладном использовании на первом шаге происходит загрузка векторов вероятностей из внешнего источника. После обучения для идентификации необходимых приборов, система готова к использованию, доступен ручной ввод вектора, а также загрузка показаний приборов из внешнего источника. Если это файлы, то они должны содержать 128\*n строк, где n – количество устройств для идентификации, в каждой строке по одному вещественному числу.

3 Разработка программы формирования нейронной сети

1. 1. Постановка задачи

Разработать программу ~~систему эффективной~~ идентификации приборов по их показаниям. В основе программы – искусственная нейронная сеть.

Требуется определить класс I устройства dev:

, (9)

где f – функция идентификации прибора, и I принадлежит некоторому множеству таких классов:

, (10)

тогда принадлежность прибора к группе однозначно определяется:

, (11)

* 1. Общие сведения о программе

Программа выполнена на языке Python. Для ~~её~~ запуска ~~по~~требуется интерпретатор Python версии 3.4.3, а также модули PyQt4, random, math, sys, time.

* 1. Описание программы

В качестве исходных данных выступает набор векторов P1, P2, P3, … Pn, (состоящих из 128 бит единиц и нулей) в совокупности представляющих собой вероятности перехода бита в состояние единицы (при моделировании приборов) или 128-ми битные показания приборов (считываются с COM-порта), результатом работы программы является индексы приборов.

При запуске программы открывается диалоговое меню, в сопровождении с краткими инструкциями, взаимодействуя с которыми пользователь сможет достичь желаемого результата – произвести обучение искусственной нейронной сети для требуемого числа приборов, идентификацию приборов на основе их показаний, а также сохранение сети для последующего использования, без необходимости обучать новую сеть.

4 Контрольный пример

В качестве набора входных данных выступает множество D последовательностей d из 128 бит:

, (12)

Выходной набор данных представлен множеством Y индексов устройств y (например, yi = 5):

, (13)

где n – количество приборов, участвующих в обучении.

Критерием проверки соответствия входных и выходных данных является формула (8).

Рассмотрим моделирование приборов.

После запуска программа пользователю предоставляется графический интерфейс, взаимодействие с которым, в сопровождении инструкций, позволит выполнить поставленную задачу – создание и обучение искусственной нейронной сети для идентификации приборов, с последующим прикладным использованием.

Вид интерфейса представлен на рисунке 7.

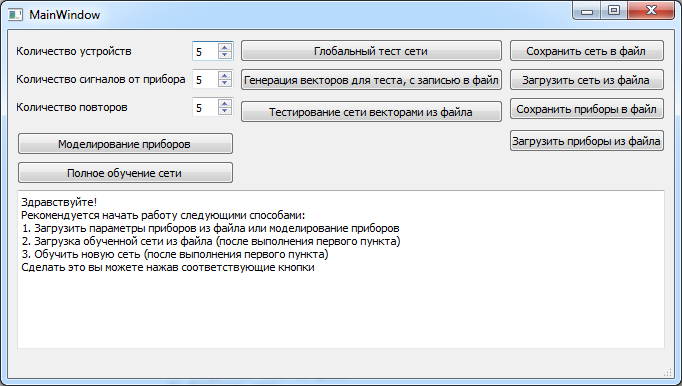


Рисунок 7 – интерфейс пользователя

Следуя рекомендациям, нажимая на кнопки, вы будете получать информацию о работе системы.

Проанализируем проверку всех функций программы на рисунке 8.

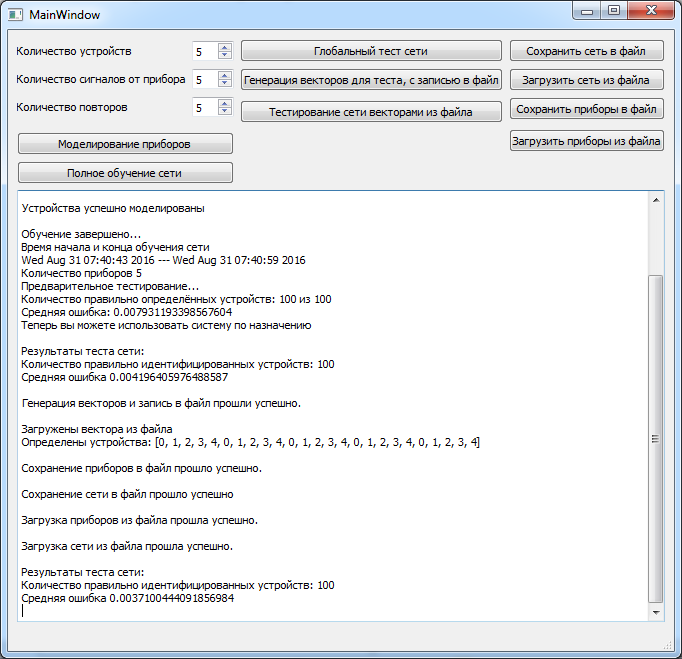


Рисунок 8 – результат выполнения всех функций системы

Рассмотрим прикладное использование.

В данном опыте будут участвовать 3 устройства. Искусственная нейронная сеть будет обучена определять лишь два из них, третье же устройство будет использовано для проверки способности отличать устройства, которые не участвовали при обучении. Эта проверка будет осуществляться за счёт вычисления ошибки между новыми входными данными и примерами из обучающей выборки

Аналогично, запускаем пользовательский интерфейс и формируем исходные данные для подготовки к работе, как показано на рисунке 9.

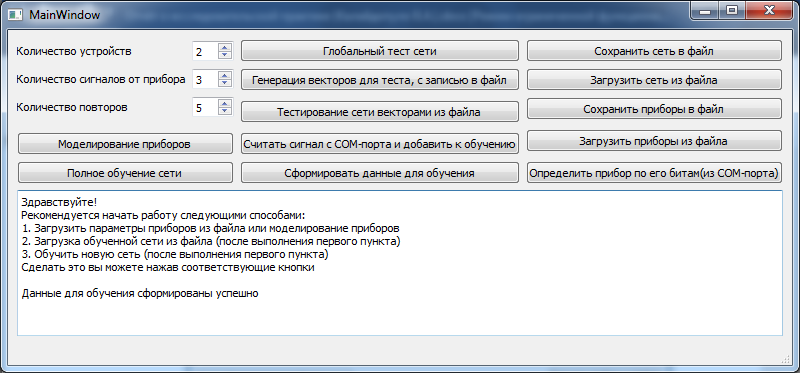


Рисунок 9 – прикладное использование

Затем считываем 6 показаний устройств, по 3 от каждого из двух. После чего выполним обучение сети.

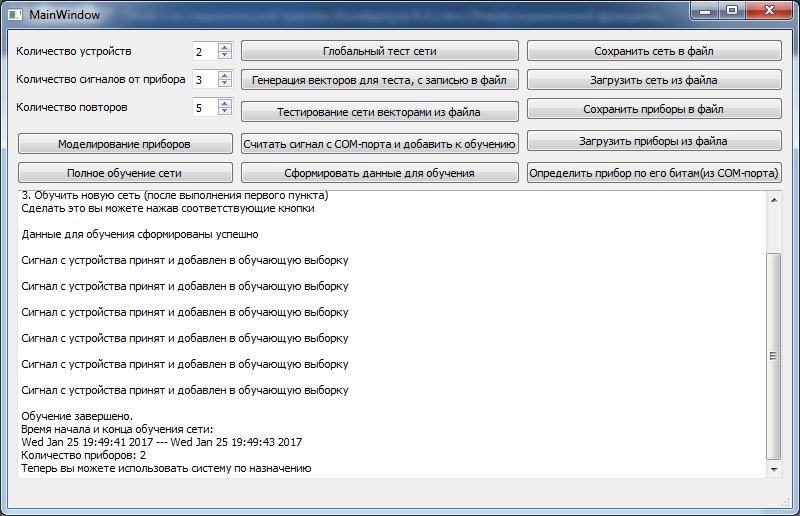


Рисунок 10 – обучение сети распознаванию приборов

Далее предоставляем сети тестовые биты устройств для проверки, убеждаемся в корректной идентификации устройств сетью.

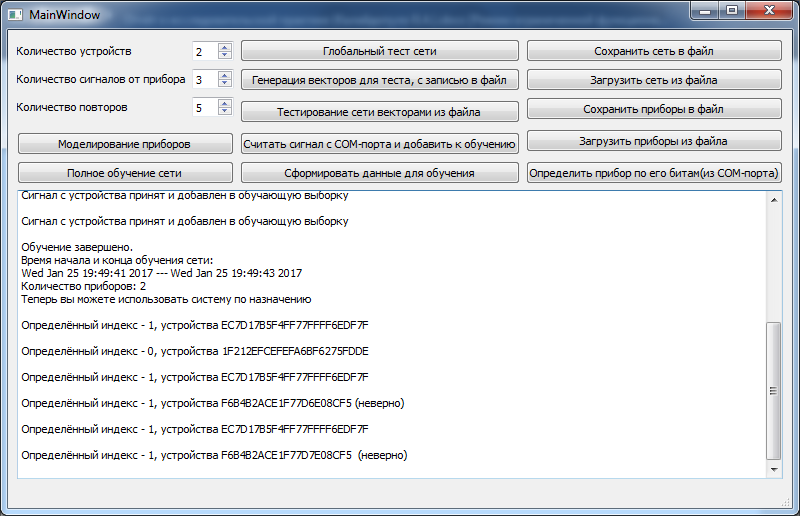


Рисунок 11 – проверка сети

Полученные данные свидетельствуют об успешном и корректном обучении разработанной сети.

Выводы: в результате проведённого контрольного примера установлено, что входные данные соответствуют выходным данным, так же, данное соответствие не противоречит критерию проверки; программа корректно выполняет идентификацию устройств.

5 Отладка и тестирование

Для выполнения поставленной задачи была написана программа, результат работы которой рассмотрен на тестовом примере.

~~Исходные данные – искусственно смоделированные вероятностные вектора устройств.~~

Для тестирования сети был разработан следующий алгоритм:

Шаг 1. Моделирование данных для тестирования. На основе моделированных или загруженных параметров устройств генерируется множество тестовой выборки вектор-признаков по следующему циклу: генерируется случайное число rnd в диапазоне от 0 до n-1, где n – число устройств; далее из набора приборов выбирается устройство с индексом rnd; затем от текущего прибора генерируется вектор-признак; все генерируемые для теста параметры записываются в соответствующий список. Данный цикл повторяется 100 раз.

Шаг 2. Проверка сети новыми вектор-признаками. Запускается следующий цикл: на вход сети поочередно подаются новые данные, неизвестные системе; на выходе сети определяется индекс устройства, соответствующему входной информации; затем производится сравнение настоящего индекса устройства (которые записаны в списке) и индекса, определенного сетью. Производится 100 итераций цикла.

Результат тестирования отображен на рисунке 12.

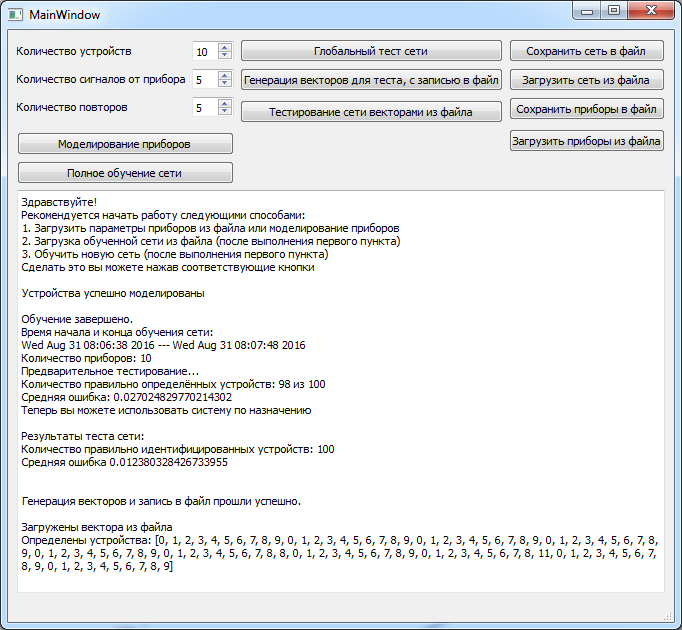


Рисунок 12 – результат тестирования сети

Тестирование сети разным количеством параметров обучения выявило зависимости, представленные на графиках.

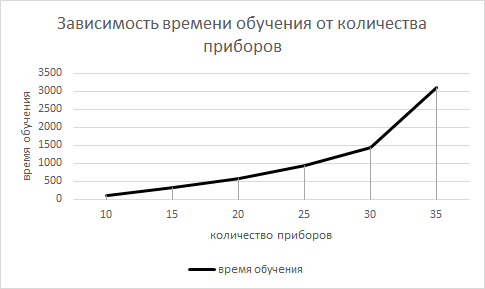


Рисунок 13 – график зависимости времени обучения от числа приборов

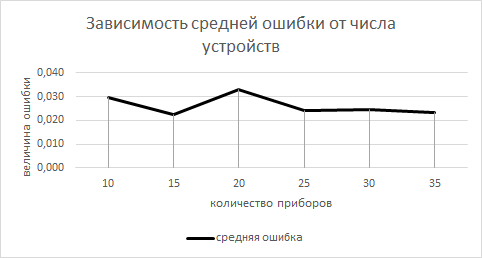


Рисунок 14 – график зависимости средней ошибки от числа устройств

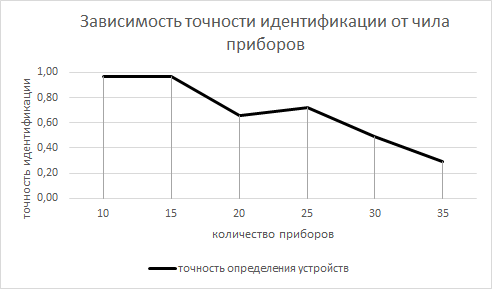


Рисунок 15 – график изменения точности определения для числа приборов

Для оценки точности идентификации была проведена серия экспериментов. В каждом эксперименте были моделированы сто устройств в виде наборов векторов P1, P2, P3, … Pn, (состоящих из 128 бит единиц и нулей) в совокупности представляющих собой вероятности перехода бита в состояние единицы (при моделировании приборов). Таким образом, каждый такой набор векторов P соответствует одному устройству, с помощью которого генерируются последовательности из 128 бит, соответствующих сигналам приборов.

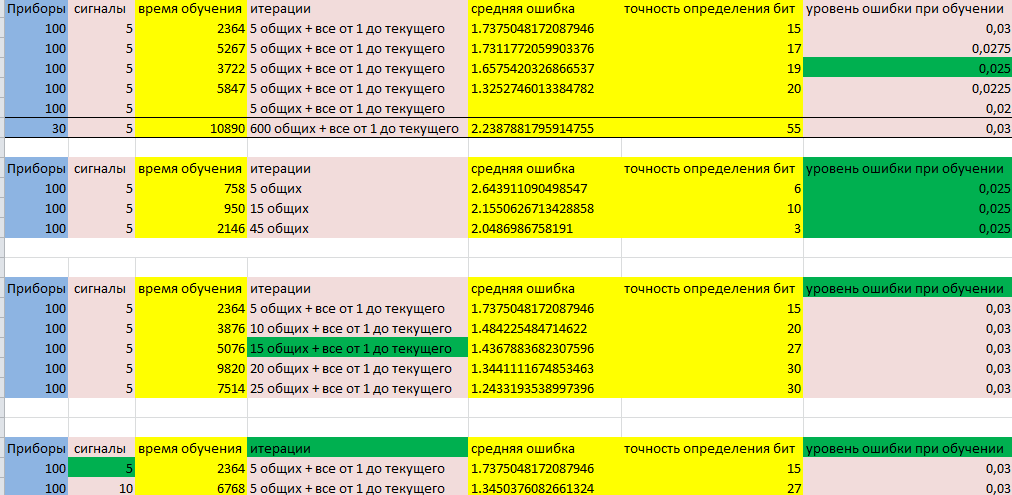


Рисунок 16 – общая статистика тестирования сети

Синим, обозначены исходные данные; светло-красным – различные значения параметров сети; желтым – результаты работы системы (в зависимости от соответствующих параметров); зеленым – оптимальные значения параметров сети, ориентированных на время обучения и точность определения.

Статистические результаты дают возможность сделать следующий вывод – при увеличении количества исходных данных (а именно: количество приборов, количество сигналов от каждого устройства, число итераций обучения) снижается эффективность разработанного алгоритма, ввиду увеличения ресурсов на обработку данных. При анализе графиков, представленных выше, легко видеть зависимость точности определения устройств от количества идентифицируемых приборов, с помощью данной системы; точность определения обратно пропорциональна количеству приборов.

Эта проблема имеет возможное решение. В случае, когда требуемая точность определения соответствует гораздо меньшему числу приборов, чем необходимое количество приборов для идентификации (например, необходимая точность 0.66, но она обеспечивается при 20 устройствах, когда требуется определить 100 или больше приборов), следует разделить устройства на равные группы и определять по очереди.

Рассмотрим недостатки при данном подходе. Первый – увеличение затраты ресурсов и времени для выполнения идентификаций; тогда как система разработана с целью сокращения затрат. Второй – после автоматизации «разбиения» большого множества приборов на группы (с последующим обучением, т.е. создание множества искусственных сетей, распознающих свою часть от общего числа приборов), остается неопределенность – когда обучение завершено, при использовании программы по назначению, в качестве данных на вход могут подаваться очень большие объемы данных (например 100000 наборов бит от неизвестных устройств, которые требуется идентифицировать); проблема заключается в том, что неизвестно на какую из нескольких сетей подавать данные, ведь они все на выходе определят индекс устройства, но достоверно установить какая из них определила верно, крайне затруднительно.

ЗАКЛЮЧЕНИЕ

В ходе выполнения научно-практической работы были проанализированы алгоритмы построения и обучения искусственной нейронной сети, в результате чего для решения поставленной задачи был выбран алгоритм обучения сети обратного распространения, как наиболее удобный для программирования, с математической точки зрения, а именно многопараметрической задачи нелинейной оптимизации. Основная идея этого метода состоит в распространении сигналов ошибки от выходов сети к её входам, в направлении, обратном прямому распространению сигналов в обычном режиме работы. Наиболее часто в качестве функций активации используется Функция Ферми (экспоненциальная сигмоида). Основным задачей являлось решение вопроса «как модифицировать веса». Так как в процессе разработки программы не удалось реализовать «стохастический градиентный спуск», был выбран метод случайного изменения весов. По этому алгоритму разработана программа. В перспективе возможно изменение алгоритма для увеличения быстродействия, а главное оптимизации определения ошибки и предсказание для изменения весов нейронов сети.

В результате тестирования программы было выявлено, что она работает корректно со всеми входными данными.

Статистические результаты дают возможность сделать следующий вывод – при увеличении количества исходных данных (а именно: количество приборов, количество сигналов от каждого устройства, число итераций обучения) снижается эффективность разработанного алгоритма, ввиду увеличения ресурсов на обработку данных. При анализе графиков, представленных выше, легко видеть зависимость точности определения устройств от количества идентифицируемых приборов, с помощью данной системы; точность определения обратно пропорциональна количеству приборов.

Эта проблема имеет возможное решение. В случае, когда требуемая точность определения соответствует гораздо меньшему числу приборов, чем необходимое количество приборов для идентификации (например, необходимая точность 0.66, но она обеспечивается при 20 устройствах, когда требуется определить 100 или больше приборов), следует разделить устройства на равные группы и определять по очереди.

Таким образом, поставленная задача успешно решена. Все цели научно-практической работы успешно достигнуты.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Системы искусственного интеллекта. Практический курс : учебное пособие / [В. А. Чулюков, И. Ф. Астахова, А. С. Потапов и др. ; под ред. И. Ф. Астаховой]. — М. : БИНОМ. Лаборатория знаний, 2008. — 292 с. : ил. — (Адаптивные и интеллектуальные системы). – Дата обращения 12.07.2016.

2. Искусственная нейронная сеть – https://ru.wikipedia.org/wiki/Искусственная\_нейронная\_сеть   
[Электронный ресурс] – Дата обращения 13.07.2016.

3. Машинное обучение – https://ru.wikipedia.org/wiki/Машинное\_ обучение. [Электронный ресурс] – Дата обращения 17.0071.2016.

4. Самоорганизующаяся карта Кохонена – https://ru.wikipedia.org/wiki/Самоорганизующаяся\_карта\_Кохонена [Электронный ресурс] – Дата обращения 18.07.2016.

5. Оверфиттинг – http://ru.math.wikia.com/wiki/Оверфиттинг [Электронный ресурс] – Дата обращения 18.07.2016.

ПРИЛОЖЕНИЕ А – ТЕКСТ ПРОГРАММЫ НА ИСХОДНОМ ЯЗЫКЕ

DetectionNetwork.py

|  |
| --- |
| # -\*- coding: utf-8 -\*-  \_\_author\_\_ = 'Mody'  from PyQt4 import QtCore, QtGui, uic  from PyQt4.QtGui import QMainWindow  import random  from math import \*  from random import normalvariate, uniform  import sys  import serial  import time  def inform(how\_many\_device, how\_many\_signals\_from\_each\_of\_them, device):  for\_length\_of\_d = len(bin(how\_many\_device))-2  d\_arr = []  empty = []  for f in range(how\_many\_device):  part\_d = empty[:]  length\_of\_bin\_f = len(bin(f))-2  missing\_piece = for\_length\_of\_d - length\_of\_bin\_f  part\_d = increase\_vector(missing\_piece, 0)  part\_d.extend(vector\_in\_int(list(bin(f)[2:length\_of\_bin\_f+2])))  d\_arr.append(part\_d)  x = []  for k in range(how\_many\_device):  for j in range(how\_many\_signals\_from\_each\_of\_them):  x.append(p\_to\_bin(device[k])[2:130])  return x, d\_arr  def short\_inform(how\_many\_device):  for\_length\_of\_d = len(bin(how\_many\_device))-2  d\_arr = []  empty = []  for f in range(how\_many\_device):  part\_d = empty[:]  length\_of\_bin\_f = len(bin(f))-2  missing\_piece = for\_length\_of\_d - length\_of\_bin\_f  part\_d = increase\_vector(missing\_piece, 0)  part\_d.extend(vector\_in\_int(list(bin(f)[2:length\_of\_bin\_f+2])))  d\_arr.append(part\_d)  return d\_arr  def modeling\_device(how\_many\_device):  device = []  for i in range(how\_many\_device):  device.append(generate\_p())  return device  def normalize(v, alpha=0.5):  if v >= alpha:  return 1  return 0  def generate\_p(size=128):  p\_arr = []  for i in range(size):  p\_arr.append(uniform(0, 1))  return p\_arr  def p\_to\_bin(p\_arr, sigma=0.1):  s = '0b'  for p in p\_arr:  if normalize(normalvariate(p, sigma)):  s += '1'  else:  s += '0'  return s  def formalization\_vector\_in\_number\_of\_device(vector\_arr\_of\_float):  vector\_in\_byte = increase\_vector(len(vector\_arr\_of\_float), 0)  for i in range(len(vector\_arr\_of\_float)):  if vector\_arr\_of\_float[i] > 0.5:  vector\_in\_byte[i] = 1  else:  vector\_in\_byte[i] = 0  s = '0b'  for j in vector\_in\_byte:  if j:  s += '1'  else:  s += '0'  number\_of\_device = int(s,2)  return number\_of\_device  def increase\_vector(len\_vector, x):  vector = []  while len(vector) < len\_vector:  vector.append(x)  return vector  def the\_init\_web(length\_x, length\_out):  w = [[random.uniform(-0.3, 0.3) for g in range(length\_out)] for h in range(length\_x)]  return w  def change\_of\_scales(w, length\_x, length\_out):  w = [[w[j][k] + random.uniform(-0.015, 0.015) for k in range(length\_out)] for j in range(length\_x)]  return w  def sigma(vector\_for\_sigma):  for j in range(len(vector\_for\_sigma)):  vector\_for\_sigma[j] = 1 / (1 + exp(-vector\_for\_sigma[j]/2))  return vector\_for\_sigma  def summation(x, w, length\_vector):  summ = increase\_vector(length\_vector, 0)  for j in range(len(summ)):  for i in range(len(x)):  summ[j] = summ[j] + w[i][j] \* x[i]  summ\_after\_sigma = sigma(summ)  return summ\_after\_sigma  def error(d, y):  maximum = abs(d[0]-y[0])  for i in range(len(d)):  if abs(d[i]-y[i]) > maximum:  maximum = abs(d[i]-y[i])  e = maximum  return e  def training(x\_ni, d\_ni, w, length\_out):  e = 5  o = 0  while e > 0.003:  yj = summation(x\_ni, w, length\_out)  e = error(d\_ni, yj)  w\_change = change\_of\_scales(w, len(x\_ni), length\_out)  yj = summation(x\_ni, w\_change, length\_out)  e\_change = error(d\_ni, yj)  if e\_change < e:  w = w\_change[:]  e = e\_change  o += 1  if o > 20000:  return w  return w  def test\_net(test\_vector, w, length\_out, d):  yj = summation(test\_vector, w, length\_out)  number\_of\_device = formalization\_vector\_in\_number\_of\_device(yj)  print("Последовательность соответствует объекту №: %s" % number\_of\_device)  the\_error = error(d[number\_of\_device], yj)  print("Ошибка определения прибора №: %s" % the\_error)  return number\_of\_device  def global\_test(device\_arr, w, length\_out, d):  test\_vectors\_arr = []  index\_device\_of\_vector = []  e\_summ = 0  index\_determinate\_device = 0  how\_many\_device\_determinate\_right = 0  for i in range(100):  rnd = random.randint(0,len(device\_arr)-1)  int\_vectors\_for\_test = vector\_in\_int(list(p\_to\_bin(device\_arr[rnd])[2:130]))  test\_vectors\_arr.append(int\_vectors\_for\_test)  index\_device\_of\_vector.append(rnd)  for j in range(100):  yj = summation(test\_vectors\_arr[j], w, length\_out)  e\_summ += error(d[index\_device\_of\_vector[j]], yj)  index\_determinate\_device = formalization\_vector\_in\_number\_of\_device(yj)  if (index\_determinate\_device == index\_device\_of\_vector[j]):  how\_many\_device\_determinate\_right += 1  else:  print(" Неверно определён вектор:")  print(" Определение сетью: ", yj)  print(" Вектор-протоип: ", d[index\_device\_of\_vector[j]])  print(" Индекс прибора, определённый сетью: ", index\_determinate\_device)  print(" Настоящий индекс прибора", index\_device\_of\_vector[j])  e\_summ = e\_summ / 100  return how\_many\_device\_determinate\_right, e\_summ  def split\_input\_vector(input\_vector):  split\_vector = []  for i in range(128):  j = int(input\_vector[i])  split\_vector.append(j)  return split\_vector  def save\_device(device\_arr):  file\_for\_save\_device = open("device2016\_mark6.txt", "w")  for i in range(len(device\_arr)):  for j in range(len(device\_arr[0])):  file\_for\_save\_device.write("%s\n" % device\_arr[i][j])  file\_for\_save\_device.close()  def load\_device():  file\_for\_load\_device = open("device2016\_mark6.txt", "r")  empty = []  device\_arr = []  part\_of\_device\_arr = empty[:]  while True:  line\_for\_device = file\_for\_load\_device.readline()  if not line\_for\_device:  break  part\_of\_device\_arr.append(float(line\_for\_device))  if len(part\_of\_device\_arr) == 128:  device\_arr.append(part\_of\_device\_arr)  part\_of\_device\_arr = empty[:]  file\_for\_load\_device.close()  return device\_arr  def save\_net(w, length\_out):  file\_for\_save\_net = open("net2016\_mark6.txt", "w")  for i in range(128):  for j in range(length\_out):  file\_for\_save\_net.write("%s\n" % w[i][j])  file\_for\_save\_net.close()  def load\_net(length\_out):  file\_for\_load\_net = open("net2016\_mark6.txt", "r")  empty = []  w1 = []  part\_of\_w = empty[:]  while True:  line\_for\_net = file\_for\_load\_net.readline()  if not line\_for\_net:  break  part\_of\_w.append(float(line\_for\_net))  if len(part\_of\_w) == length\_out:  w1.append(part\_of\_w)  part\_of\_w = empty[:]  file\_for\_load\_net.close()  return w1  def generation\_vector\_for\_generation\_in\_file(number\_of\_vectors, device\_arr):  file\_for\_generation = open("file\_for\_definition\_device.txt", "w")  for i in range(number\_of\_vectors):  for j in range(len(device\_arr)):  file\_for\_generation.write("%s\n" % p\_to\_bin(device\_arr[j]))  file\_for\_generation.close()  def definition\_device\_from\_file(w, length\_out):  file\_for\_definition = open("file\_for\_definition\_device.txt", "r")  index\_device\_arr = []  while True:  x\_input = file\_for\_definition.readline()  if not x\_input:  break  if x\_input[1] == 'b':  x\_input = x\_input[2:130]  x\_split\_vector = split\_input\_vector(x\_input)  index\_device = test\_net(x\_split\_vector, w, length\_out)  index\_device\_arr.append(index\_device)  file\_for\_definition.close()  return index\_device\_arr  def vector\_in\_int(vector\_not\_int):  vector\_yes\_int = increase\_vector(len(vector\_not\_int), 0)  for i in range(len(vector\_not\_int)):  vector\_yes\_int[i] = int(vector\_not\_int[i])  return vector\_yes\_int  def connection\_with\_port():  number\_of\_ports = 30  open\_port = ''  i = 0  k = 0  while k == 0:  #print("i= ", i)  name\_port = 'COM' + str(i)  try:  ser = serial.Serial(name\_port, 9600, timeout=1)  ser.close()  k = 1  #print("try ")  except serial.serialutil.SerialException:  k = 0  #print("except ")  if k == 1:  open\_port = name\_port  if i > number\_of\_ports:  open\_port = 'there are no available ports'  break  i += 1  return open\_port  def get\_vector\_from\_port():  port\_name = connection\_with\_port()  try:  ser = serial.Serial(port\_name, 9600, timeout=1)  time.sleep(2)  ser.write(b"GET FNF\r\n")  bytes\_from\_com\_port = ser.read(128)  ser.close()  except:  w.textEdit.append("something is wrong, it is impossible to open a port!")  array\_for\_net = 0  return array\_for\_net  print("bytes\_from\_com\_port= ", bytes\_from\_com\_port)  bytes\_to\_binary\_str = bin(int(str(bytes\_from\_com\_port)[2:len(str(bytes\_from\_com\_port))-5],16))  bytes\_to\_array\_str = list(bytes\_to\_binary\_str[2:len(bytes\_to\_binary\_str)])  for i in range(128-len(bytes\_to\_array\_str)):  bytes\_to\_array\_str.insert(0, '0')  array\_for\_net = vector\_in\_int(bytes\_to\_array\_str)  return array\_for\_net  def identifyDeviceFromCOMport(w, length\_out, d):  vector\_for\_test = get\_vector\_from\_port()  number\_of\_device = test\_net(vector\_for\_test, w, length\_out, d)  return number\_of\_device  class MainWindow(QMainWindow):  def \_\_init\_\_(self, \*args):  QMainWindow.\_\_init\_\_(self, \*args)  uic.loadUi("BitDetection.ui", self)  self.connect(self.pushButtonFullChargeLern, QtCore.SIGNAL("clicked()"), self.fullChargeLerning)  self.connect(self.pushButtonSaveNet, QtCore.SIGNAL("clicked()"), self.saveNet)  self.connect(self.pushButtonLoadNet, QtCore.SIGNAL("clicked()"), self.loadNet)  self.connect(self.pushButtonSaveDevice, QtCore.SIGNAL("clicked()"), self.saveDevice)  self.connect(self.pushButtonLoadDevice, QtCore.SIGNAL("clicked()"), self.loadDevice)  self.connect(self.pushButtonGlobalTest, QtCore.SIGNAL("clicked()"), self.globalTest)  self.connect(self.pushButtonGenerationVectorForTestInFile, QtCore.SIGNAL("clicked()"), self.generationVector)  self.connect(self.pushButtonTestNetWithVectorFromFile, QtCore.SIGNAL("clicked()"), self.definitionDevice)  self.connect(self.pushButtonModelForLern, QtCore.SIGNAL("clicked()"), self.modelingForLearn)  self.connect(self.pushButtonAddSignalDeviceForTraining, QtCore.SIGNAL("clicked()"), self.appendSignalArr)  self.connect(self.pushButtonNewLearn, QtCore.SIGNAL("clicked()"), self.newModelingForLearn)  self.connect(self.pushButtonForTestFromCOMPort, QtCore.SIGNAL("clicked()"), self.identifyDevice)  self.device = []  self.x\_arr = []  self.textEdit.append("Здравствуйте!")  self.textEdit.append("Рекомендуется начать работу следующими способами:")  self.textEdit.append("1. Загрузить параметры приборов из файла или моделирование приборов ")  self.textEdit.append("2. Загрузка обученной сети из файла (после выполнения первого пункта) ")  self.textEdit.append("3. Обучить новую сеть (после выполнения первого пункта)")  self.textEdit.append("Сделать это вы можете нажав соответствующие кнопки\n")  def identifyDevice(self):  number\_of\_device = identifyDeviceFromCOMport(self.w, self.length\_out, self.d)  self.textEdit.append("Определён индекс устройства %s \n" % number\_of\_device)  def modelingForLearn(self):  self.how\_many\_device = int(self.spinBoxHowManyDevice.text())  self.device = modeling\_device(self.how\_many\_device)  self.how\_many\_signal = int(self.spinBoxSignalsFromDevice.text())  self.x, self.d = inform(self.how\_many\_device, self.how\_many\_signal, self.device)  self.length\_out = len(self.d[0])  self.textEdit.append("Устройства успешно моделированы \n")  def appendSignalArr(self):  self.x\_arr.append(get\_vector\_from\_port())  self.textEdit.append("Сигнал с устройства принят и добавлен в обучающую выборку \n")  def newModelingForLearn(self):  self.how\_many\_device = int(self.spinBoxHowManyDevice.text())  self.how\_many\_signal = int(self.spinBoxSignalsFromDevice.text())  self.x = self.x\_arr  self.d = short\_inform(self.how\_many\_device)  self.length\_out = len(self.d[0])  self.textEdit.append("Данные для обучения сформированы успешно \n")  def fullChargeLerning(self):  self.w = the\_init\_web(len(self.x[0]), self.length\_out)  p = 0  print("Идёт процесс обучения сети, пожалуйста, подождите...")  print("И ничего не нажимайте...")  starting = time.ctime()  repeat\_training = int(self.spinBoxRepeat.text())  while p < repeat\_training:  ni = 0  i = 0  empty = []  d\_arr = []  x\_arr = []  while ni < len(self.x):  vector\_x\_for\_training = vector\_in\_int(list(self.x[ni]))[:]  vector\_d\_for\_training = self.d[i][:]  d\_arr.append(vector\_d\_for\_training)  x\_arr.append(vector\_x\_for\_training)  for l in range(len(x\_arr)):  w\_training = training(x\_arr[l], d\_arr[l], self.w, self.length\_out)  self.w = w\_training[:]  ni = ni + 1  if ni % self.how\_many\_signal == 0:  i += 1  d\_arr = empty[:]  x\_arr = empty[:]  p = p + 1  print("Сеть производит повтор обучения № %s " % p)  ending = time.ctime()  print("Обучение завершено.")  self.textEdit.append("Обучение завершено.")  self.textEdit.append("Время начала и конца обучения сети:")  self.textEdit.append("%s --- %s" % (starting, ending))  self.textEdit.append("Количество приборов: %s" % self.how\_many\_device)  self.textEdit.append("Теперь вы можете использовать систему по назначению \n")  def saveNet(self,):  save\_net(self.w, self.length\_out)  self.textEdit.append("Сохранение сети в файл прошло успешно\n")  def loadNet(self):  w\_new = load\_net(self.length\_out)  self.w = w\_new[:]  self.x, self.d = inform(self.how\_many\_device, int(self.spinBoxSignalsFromDevice.text()), self.device)  self.textEdit.append("Загрузка сети из файла прошла успешно.\n")  def saveDevice(self):  save\_device(self.device)  self.textEdit.append("Сохранение приборов в файл прошло успешно.\n")  def loadDevice(self):  self.device = load\_device()  self.length\_out = len(bin(len(self.device)))-2  self.how\_many\_device = len(self.device)  self.x, self.d = inform(self.how\_many\_device, int(self.spinBoxSignalsFromDevice.text()), self.device)  self.textEdit.append("Загрузка приборов из файла прошла успешно.\n")  def globalTest(self):  self.textEdit.append("Результаты теста сети:")  how\_many\_device\_determinate\_right, e\_summ = global\_test(self.device, self.w, self.length\_out, self.d)  self.textEdit.append("Количество правильно идентифицированных устройств: %s" % how\_many\_device\_determinate\_right)  self.textEdit.append("Средняя ошибка %s \n" % e\_summ)  def generationVector(self):  generation\_vector\_for\_generation\_in\_file(self.how\_many\_device, self.device)  self.textEdit.append("Генерация векторов и запись в файл прошли успешно.\n")  def definitionDevice(self):  index\_device\_arr = definition\_device\_from\_file(self.w, self.length\_out)  self.textEdit.append("Загружены вектора из файла")  self.textEdit.append("Определены устройства: %s \n" % index\_device\_arr)  if \_\_name\_\_ == "\_\_main\_\_":  app = QtGui.QApplication(sys.argv)  w = MainWindow()  w.show()  sys.exit(app.exec\_()) |