МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование кафедры)

02.03.03 МАТЕМАТИЧЕСКОЕ ОБЕСПЕЧЕНИЕ И АДМИНИСТРИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ
(код и наименование направления подготовки, специальности)

ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ
(направленность (профиль)/специализация)

БАКАЛАВРСКАЯ РАБОТА

на тему «Применение радиально-базисных нейронных сетей в задачах управления технологическими процессами»

Студент В.И. Шмыров _____
(И.О. Фамилия) (личная подпись)

Руководитель В.С. Климов _____
(И.О. Фамилия) (личная подпись)

Допустить к защите
Заведующий кафедрой к.т.н., доцент, А.В.Очеповский _____
(ученая степень, звание, И.О. Фамилия) (личная подпись)

«_____» _____ 2018 г.

Тольятти 2018

АННОТАЦИЯ

Тема данной выпускной квалификационной работы: исследование виртуализации рабочих столов студентов в вычислительном кластере.

Целью данной ВКР является разработка искусственной нейронной сети радиально-базисных функций и исследование ее возможностей применительно к задачам управления.

Объектом исследования являются искусственные нейронные сети.

Предметом исследования являются искусственные нейронные сети в задачах управления.

Для достижения цели необходимо было решить следующие задачи:

- провести аналитический обзор наиболее популярных архитектур искусственных нейронных сетей;
- провести разбор предметной области;
- обосновать выбор наиболее оптимального решения;
- спроектировать искусственную нейронную сеть;
- реализовать искусственную нейронную сеть на одном из языков программирования;
- провести исследования эффективности разработанной искусственной нейронной сети.

Данные задачи были решены.

В ходе выполнения ВКР была разработана искусственная нейронная сеть радиально-базисных функций.

Выпускная квалификационная работа состоит из введения, трех глав и заключения.

Во введении описывается актуальность проводимого исследования, формируется цель, и ставятся задачи, которые необходимо решить для достижения цели.

В первой главе проводится анализ состояния вопроса, а именно анализ задач управления технологическими процессами. Также проводится

сравнительный анализ существующих архитектур искусственных нейронных сетей, в том числе нейронной сети радиально-базисных функций, многослойного персептрона и самоорганизующейся карты Кохонена. Также в первой главе рассматриваются существующие решения в нейросетевых вычислениях.

Во второй главе разрабатывается и описывается архитектура искусственной нейронной сети радиально-базисных функций, а также в общих чертах описывается реализация такой сети на языке программирования Java.

В третьей главе исследуется влияние значений гиперпараметров спроектированной и реализованной нейронной сети на вычисляемые ей значения.

В заключении приводятся основные выводы по работе, описываются достигнутые задачи в ходе выполнения ВКР.

Выпускная квалификационная работа содержит пояснительную записку объемом 40 страниц, включая 13 рисунков, 3 таблицы и список использованной литературы из 20 наименований.

Abstract

The topic of the given thesis is radial-basis function neural network in system control tasks.

The aim of the thesis is to develop neural network and research its possibilities in system control tasks.

The object of the thesis is neural network.

The subject of the thesis is neural networks in system control tasks.

We study the role of neural networks in modern tasks of system control. Then we examine currently existing neural networks architectures and compare them.

Also in a course of given thesis has been described the realization of radial basis function neural network and impact of hyperparametrs on the ouput of such network.

In the introduction author outlines relevance of the given topic and forms lists of tasks, that have to be solved.

In the first chapter of the thesis author analyzes currently existing architectures of artificial neural networks. Also author examines the system control tasks. After that, author analyzes currently existing solutions in the neural network calculations sphere.

In the second chapter of the thesis author models architecture of radial basis function neural network and gives details on backpropagation algorithm. Also in the second chapter author describes main algorithms of developed neural network

In the third chapter of the thesis impact of hyperparameters on the neural network output is being examined.

In conclusion author outlines main conclusions of the given thesis and solved in a process of given diploma paper tasks.

The thesis consists of 40 pages, containing 3 tables, 13 figures, a Power Point presentation containing 19 slides, a list of 20 references including foreign sources and 1 appendix.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 7

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 9

1.1 Задачи управления технологическими процессами 9

1.2 Общие сведения об искусственных нейронных сетях 11

1.3 Анализ существующих решений 19

1.3.1 MatLab 7.1- ANFIS-редактор 20

1.3.2 STATISTICA Automated Neural Networks 21

2 РАЗРАБОТКА ИНФОРМАЦИОННО-ЛОГИЧЕСКОГО ПРОЕКТА И ПРОГРАММНАЯ РЕАЛИЗАЦИЯ 23

2.1 Структура радиально-базисной нейронной сети 23

2.2 Применение алгоритма обратного распространения ошибки для радиально-базисных сетей. 25

2.3 Проектирование программной реализации нейронной сети 28

2.3.1 Диаграмма классов 29

2.3.2 Диаграмма деятельности 30

2.4 Обоснование средств реализации искусственной нейронной сети 32

3 ИССЛЕДОВАНИЕ ЭФФЕКТИВНОСТИ НЕЙРОННОЙ СЕТИ 37

3.1 Общие сведения о гиперпараметрах нейронной сети 37

3.2 Исследование зависимости эффективности алгоритма обучения от значения коэффициента обучения 39

3.3 Исследование зависимости погрешности прогнозирования от момента 41

3.4 Исследование зависимости эффективности алгоритма обучения от количества нейронов скрытого слоя 43

ЗАКЛЮЧЕНИЕ 45

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 47
Приложение А. 49

ВВЕДЕНИЕ

В последнее время можно наблюдать бум развития методов машинного обучения, и, в частности, искусственных нейронных сетей. Неудивительно, ведь математический аппарат нейронных сетей является одним из наиболее перспективных направлений машинного обучения. Искусственная нейронная сеть является, по своей сути, самообучающейся (чаще всего) системой, которая в процессе обучения устанавливает отношения между входными и выходными данными. Одним из важных свойств таких сетей является гибкость и масштабируемость для большого числа входящих параметров.

К тому же, искусственные нейронные сети являются адаптирующимися системами, ведь они могут перестраивать свою работу для соответствия данным не имея какой-либо информации об описываемой модели. Кроме этого, они способны аппроксимировать любую функцию с определенной точностью и являются нелинейными моделями, что позволяет им моделировать процессы окружающей среды.

Таким образом, применение нейронной сети позволит автоматизировать процесс решения задачи прогнозирования, при этом многократно сократив трудоемкость и время обработки данных.

Целью данной ВКР является разработка искусственной нейронной сети радиально-базисных функций и исследование ее возможностей применительно к задачам управления.

Объектом исследования являются искусственные нейронные сети.

Предметом исследования являются искусственные нейронные сети в задачах управления.

Для достижения цели ВКР необходимо решить следующие задачи:

- провести аналитический обзор наиболее популярных архитектур искусственных нейронных сетей;
- провести разбор предметной области;
- обосновать выбор наиболее оптимального решения;

- спроектировать искусственную нейронную сеть;
- реализовать искусственную нейронную сеть на одном из языков программирования;
- провести исследования эффективности разработанной искусственной нейронной сети.

Бакалаврская работа состоит из введения, трех глав, заключения, списка литературы и приложений.

Во введении описывается актуальность проводимого исследования, формируется цель, и ставятся задачи, которые необходимо решить для достижения цели.

В первой главе описывается анализ предметной области, а также разбор уже существующих решений в данной области.

Во второй главе описывается архитектура искусственной нейронной сети, описывается алгоритм ее обучения, и проектируется структура искусственной нейронной сети.

В третьей главе проводятся исследования эффективности искусственной нейронной сети с различными параметрами ее конфигурации.

В заключении приводятся основные выводы по работе, описываются достигнутые задачи в ходе выполнения ВКР.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1 Задачи управления технологическими процессами

Современные технологические процессы являют собой сложные системы, состоящие из большого числа различных технологических агрегатов и установок. Системы управления такими процессами, в свою очередь, являют собой сложные иерархические системы, в которых каждый уровень иерархии выполняет свои задачи.

Так, в последнее время выделяется три уровня управления технологическими процессами на предприятии [1]:

- верхний уровень управления предприятием – на данном уровне решаются стратегические задачи, решаются вопросы управления ресурсами в долгосрочной перспективе, и, помимо прочего, выполняется ряд функций поддержки производства, например, долгосрочное планирование в месячном или годовом масштабе;
- интеграционный уровень управления предприятием – на этом уровне решаются задачи оперативного управления технологическими процессами производства, а соответствующая автоматизированная система управления обеспечивает эффективное использование ресурсов, таких как сырье, персонал, прочие производственные средства. Помимо этого, на данном уровне происходит обеспечение оптимального выполнения плановых заданий на различных временных промежутках;
- нижний уровень решает классические задачи управления технологическими процессами.

Таким образом, имеется несколько уровней управления технологическими процессами, каждый со своей спецификой.

В связи с постоянным ростом производственных мощностей и сильно выраженным динамизмом производственных процессов возрастают и требования к эффективности управления технологическими процессами.

Так, основными целями автоматизации технологический процессов являются:

- повышение общей эффективности технологического процесса;
- повышение экологичности;
- повышение безопасности;
- повышение экономичности.

Стоит отметить, что наряду с усложнением производственных технологических процессов бурно развивались и информационные технологии. Поэтому современному этапу управления технологическими процессами характерно широкое применение средств вычислительной техники, в том числе таких, как:

- реализация алгоритмов оценки качества управления;
- реализация алгоритмов повышения эффективности потребления производственных ресурсов;
- сбор и оценка релевантных данных;
- реализация алгоритмов повышения эффективности проектирования;
- реализация диалога «человек-система управления».

Из всего вышеперечисленного следует, что современные системы управления технологическими процессами выполняют работу по сбору и анализу собранной информации, оценке этой информации, оптимизации производственных процессов исходя из полученных данных. Человек же в такой системе выполняет не более чем функцию принятия решений, основываясь на представленных ему системой данных. При этом чаще всего количество необходимых для принятия решений снижается – часть решений принимается алгоритмами. Такой вид систем управления называется экспертными системами.

Однако возникает проблема – существует ряд достаточно сложных нелинейных задач, которые либо превышают возможности общепринятых

алгоритмических методов, либо требуют больших материальных и временных затрат [2].

Для решения подобных задач использование методов машинного обучения подходит в большей степени, нежели использование классических методов алгоритмизации.

1.2 Общие сведения об искусственных нейронных сетях

В последние годы можно наблюдать бурное развитие технологий машинного обучения, и развитие нейросетевых технологий в частности. Это связано с рядом факторов. Технологии нейросетевого моделирования, имеют следующие достоинства:

- высокую эффективность работы с большими объемами данных;
- способность воспроизводить сложные нелинейные соотношения;
- отсутствие ограничений на функцию распределения данных;
- возможность обработки разнотипных экспериментальных данных;
- сохранение работоспособности при наличии пропусков в таблице обучающей выборки.

Основная идея, стоящая за нейронными сетями – человеческий головной мозг. Нейронные сети, по сути своей, являются приближенной моделью человеческого мозга и взаимодействия между естественными нейронами.

Естественный нейрон имеет отростки двух типов. Первый тип – дендриты. Дендритов может быть множество, и они выполняют функцию принятия информации, поступающей в виде электрических импульсов, от других нейронов. Также естественный нейрон имеет аксон, в единственном экземпляре. Аксон передает информацию другим нейронам.

Соединение между нейронами реализуется в синапсах. Принятые синапсом сигналы подводятся к телу нейрона. Здесь они суммируются, причем одни входы стремятся возбудить нейрон, другие воспрепятствовать возбуждению. Когда сумма возбуждающих сигналов в теле превышает

некоторый порог, нейрон возбуждается, посылая по аксону сигнал возбуждения другим нейронам. У этой функциональной схемы есть много обобщений, но, тем не менее, большинство моделей искусственного нейрона моделируют только эти простейшие свойства.

Искусственным нейроном называется узел искусственной нейронной сети, являющийся моделью биологического нейрона. Математически, искусственный нейрон представляют как некоторую нелинейную функцию от единственного аргумента – линейной комбинации всех входных сигналов. Эта функция может называться функцией активации, срабатывания или передаточной функции. Полученный из этой функции результат посылается на единственный выход искусственного нейрона.

Математическая модель искусственного нейрона была предложена У. Маккалоком и У. Питтсом и впервые практически реализована в 1958 году.

Формально искусственный нейрон можно представить следующим образом:

$$y=f(u), \text{ где } u=\sum_{i=1}^{n} w_i x_i + w_0 x_0 \qquad (1.1)$$

Где y - выходной сигнал нейрона; f (u) - функция выходного блока нейрона; w_i - синаптический коэффициент, он же вес i-го входа; x_i -i-й входной сигнал; w_0- начальное возбуждение или смещение нейрона; i, 1 - номер входа нейрона и n - число входов. Неформально искусственный нейрон можно охарактеризовать как математический сумматор.

Существует три типа нейронов – входные, промежуточные (или скрытые) и выходные. Входные принимают исходный вектор и передают его на выход, возможно, ослабив или усилив его. Далее сигнал следует в промежуточные нейроны, которые выполняют основные вычислительные операции, и оттуда полученные вычисления идут на выходные нейроны, являющие собой выход сети.

Возможные значения сигналов на входах нейрона считают заданными в промежутке [0;1], при этом они могут быть как аналоговыми, так и дискретными.

Передаточная функция чаще всего имеет область значений [-1;1] или [0;1], однако бывают и исключения, т.к. для некоторых нейронных сетей необходима непрерывная дифференцируемость на всей числовой оси.

Существует несколько видов передаточной функции. Остановимся лишь на нескольких из них.

Пороговая передаточная функция, по сути своей, является перепадом. До тех пор, пока взвешенный сигнал не достигает некоторого уровня T, выходной сигнал равен нулю. По достижении порога выходной сигнал равен единице. Таким образом, пороговую передаточную функцию можно формально описать как кусочно-заданную функцию:

$$f(x) = \begin{matrix} 0, & x < T \\ 1, & x \geq T \end{matrix} \quad (1.2)$$

Радиально-базисная функция передачи принимает в качестве аргумента расстояние между неким входным вектором и некоторым заданным наперед центром функции. Чем ближе входной вектор к центру, тем выше значение активационной функции.

Линейная функция связывает линейно сигнал выходного сигнала со взвешенной суммой сигналов на входе:

$$f(x) = tx \quad (1.3)$$

Так же архитектуру нейронных сетей принято различать по методу распространения сигнала, или же по характеру связей.

Одним из методов распространения сигнала является метод прямого распространения, в котором все связи направлены строго от входных нейронов к выходным.

Так же существуют рекуррентные нейронные сети, в которых сигнал с выходных нейронов частично передается назад на входные нейроны, таким образом осуществляя своеобразную фильтрацию данных.

Помимо этого, нейронные сети делятся по методу обучения – существуют как нейронные сети, обучающиеся с «учителем», так и самообучающиеся нейронные сети.

Обобщенная схема нейронной сети представлена на рисунке 1.1.

Для правильной работы нейронной сети необходимо ее «обучить». Для этого нужен массив промаркированных входных данных. В ходе обучения нейронной сети настраиваются веса нейронов скрытого слоя. По окончанию обучения получается нейронная сеть, которая, с некоторой вероятностью ошибки, получив входные данные будет выдавать выходные данные, которые можно использовать в дальнейшем для различных целей.

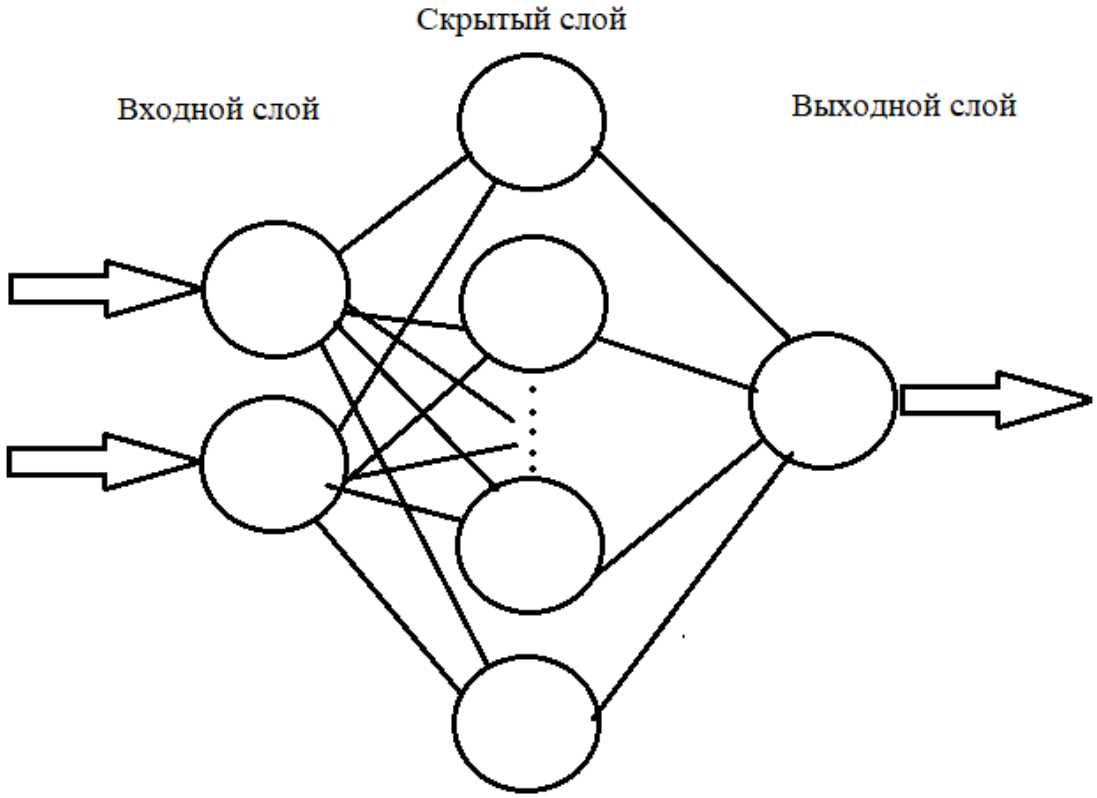

Рисунок 1.1 — Обобщенная схема однослойной нейронной сети

Искусственные нейронные сети могут использоваться для решения широчайшего круга задач, среди которых:

- задачи классификации;
- задачи принятия решений и управления;
- задачи кластеризации;

- задачи прогнозирования;
- задачи аппроксимации;
- задачи оптимизации.

Конкретно для задач управления технологическими процессами подходят те виды нейронных сетей, которые решают задачи принятия решений, задачи прогнозирования, задачи аппроксимации и задачи оптимизации.

Эти задачи решает несколько видов искусственных нейронных сетей. В их числе: персептрон, сеть радиально-базисных функций, и нейронные сети Кохонена, в частности, самоорганизующаяся карта Кохонена. Остановимся подробнее на каждой из этих архитектур.

Рассмотрим архитектуру персептрона.

По классификации нейронных сетей персептрон является искусственной нейронной сетью с одним (или несколькими) скрытым(и) слоем, пороговой передаточной функцией и прямым распространением сигнала. Персептрон с одним скрытым слоем называется однослойным, персептрон с несколькими скрытыми слоями, соответственно, многослойным.

Впервые принципы работы персептрона были описаны в книге «Персептроны» за авторством Марвина Мински и Сеймура Паперта в 1969 г.

Суть работы персептрона строится на принципе «стимул-реакция». В персептроне наличествуют ассоциативные элементы и реагирующие элементы. Сигнал сначала поступает на ассоциативный элемент, а далее на реагирующий элемент. Таким образом создается набор «ассоциаций» между входными стимулами и реакцией на выходе.

Лучше всего нейронные сети, построенные по архитектуре персептрона справляются с задачами классификации и задачами аппроксимации. Однако, в вышеупомянутой книге «Персептроны» было показано, что персептроны имеют ограничения в задачах с инвариантным представлением образов, не

имеют функционального преимущества над аналитическими методами, или могут потребовать нереально большого времени и памяти для решения некоторых задач, например, задач прогнозирования.

Рассмотрим архитектуру радиально-базисной нейронной сети.

Радиально-базисные нейронные сети – такие искусственные нейронные сети, которые в качестве функции активации используют радиально-базисные функции как функцию активации. Выходом таких сетей является линейная комбинация радиальных базисных функций и параметров нейрона. Такие сети имеют множество применений, в том числе их используют для решения задач распознавания и классификации, задач принятия решений и управления, задач прогнозирования и задач аппроксимации.

Формально радиально-базисная функция может быть представлена следующим образом:

$$f = \varphi(\frac{x^2}{\sigma^2}) \tag{1.4}$$

Где x - вектор входных сигналов нейрона, σ – ширина окна функции, φ – некая убывающая функция, чаще всего равная нулю вне определенного отрезка.

Впервые радиально-базисные нейронные сети были сформулированы в 1988 году Д.С. Брумхедом и Девидом Лоу.

Радиально-базисная сеть характеризуется тремя особенностями:

1. В ней имеется только один скрытый слой.
2. Только нейроны скрытого слоя имеют нелинейную активационную функцию.
3. Синаптические веса связей входного и скрытого слоев равны единице.

Радиально-базисные нейронные сети имеют ряд преимуществ над другими архитектурами сетей. Во-первых, как было упомянуто выше, они имеют всего один скрытый слой, что позволяет освободить разработчика от решения вопроса о количестве скрытых слоев и облегчает реализацию. Во-

вторых, полученные линейные комбинации можно оптимизировать с помощью известных методов линейной оптимизации. В связи с этим радиально-базисные нейронные сети быстро обучаются.

Однако, у радиально-базисных сетей есть и недостатки. Так, они получаются весьма громоздкими при большой размерности вектора входов. Также к недостаткам можно отнести тот факт, что заранее должно быть известно число эталонов.

Таким образом, для построения сети радиально-базисных функций необходимо выполнение ряда определенных условий.

В первую очередь, необходимо наличие эталонов, представленных в виде весовых векторов нейронов скрытого слоя. Данная задача на практике может решаться несколькими способами:

- взятие за эталонные вектора случайные вектора из обучающей выборки. Однако в таком случае сходимость нейронной сети может оказаться довольно долгой;
- предварительное нахождение центров кластеров с помощью одного из алгоритмов кластеризации. На практике чаще всего используется алгоритм k-means [3].

Во-вторых, необходимо наличие способа измерения расстояния входного вектора от эталона. Обычно это стандартное евклидово расстояние, но может быть и любое другое, например, расстояние Манхэттена.

В-третьих, необходимо определиться со специальной функцией активации нейронов скрытого слоя, задающей выбранный способ измерения расстояния. Чаще всего используется функция Гаусса, существенно усиливающая малую разницу между входным и эталонным векторами.

Рассмотрим архитектуру самоорганизующейся карты Кохонена.

Самоорганизующаяся карта Кохонена отличается тем, что это нейронная сеть с обучением без учителя. Чаще всего данная нейронная сеть выполняет задачи кластеризации или визуализации. Помимо этого часто

применяется для решения задач моделирования, прогнозирования, поиска закономерностей в больших массивах данных и т.д. Впервые была предложена финским ученым Теуво Кохоненом в 1989 году. По своей сути является методом проецирования многомерного пространства на пространство с более низкой размерностью.

Сеть Кохонена достаточно проста по своей структуре – она представляет всего два слоя: входной и выходной. Такая сеть называется картой. Элементы карты располагаются в n-мерном пространстве, чаще всего двумерном.

Сеть Кохонена обучается методом последовательных приближений. Как и в случае с любой искусственной нейронной сетью, в процессе обучения на вход подаются данные, но сеть при этом подстраивается не под эталонное значение выхода, а под закономерности во входных данных. Обучение начинается с выбранного случайным образом выходного расположения центров.

В процессе последовательной подачи на вход сети обучающих примеров определяется наиболее близкий нейрон (тот, у которого скалярное произведение весов и поданного на вход вектора минимально). Этот нейрон объявляется победителем и является центром при подстройке весов у соседних нейронов. Иначе говоря, такое правило обучения предполагает "соревновательное" обучение с учетом расстояния нейронов от "нейрона-победителя".

Обучение при этом заключается не в минимизации ошибки, как в случае с рядом других искусственных нейронных сетей, а в подстройке весов (внутренних параметров нейронной сети) для наибольшего совпадения с входными данными.

К преимуществам самоорганизующейся карты Кохонена можно отнести устойчивость к зашумленным данным, возможность упрощения многомерных входных данных с помощью визуализации и быстрое и неуправляемое обучение.

К недостаткам стоит отнести сильную зависимость от первоначальных настроек сети – специалисту, работающему с такой сетью, необходимо глубоко погрузится в объект исследования, и пройти серьезное обучение самому, перед тем как подвергнуть обучению непосредственно самообучающуюся карту Кохонена.

Так, изучив и проанализировав несколько самых подходящих архитектур искусственных нейронных сетей, можно сделать ряд выводов о том, какие из них наилучшим образом подходят для задач управления технологическими процессами.

Персептрон является простым по архитектуре решением, однако не имеет значительных функциональных преимуществ над классическими алгоритмическими методами. Кроме того, он мало приспособлен для решения задач управления.

Самообучающаяся карта Кохонена имеет ряд серьезных преимуществ, но требует серьезной подготовки внедряющего сеть специалиста. Кроме того, искусственная нейронная сеть данной архитектуры хорошо справляется с задачами кластеризации и визуализации, но не так хороша в задачах оптимизации.

Оптимальным выбором для решения задач управления технологических процессов будет радиально-базисная нейронная сеть. Данный вид нейронных сетей хорошо справляется с задачами управления, оптимизации и прогнозирования, а также относительно просты в реализации.

Так, определившись с оптимальной архитектурой искусственной нейронной сети, можно приступить к анализу уже существующих решений.

1.3 Анализ существующих решений

На данный момент существует несколько пакетных решений для создания, обучения и эксплуатации искусственных нейронных сетей. В целях оценки предоставляемой ими пользователям функциональности, а также выявления достоинств и недостатков для их последующего учета при разработке собственной системы остановимся на некоторых из них.

1.3.1 MatLab 7.1- ANFIS-редактор

ANFIS является аббревиатурой Adaptive Neuro-Fuzzy Inference System – (адаптивная нейро-нечеткая система). ANFIS-редактор позволяет автоматически синтезировать из экспериментальных данных нейро-нечеткие сети [4]. При этом функции принадлежности синтезированных систем сразу настроены (обучены) так, чтобы минимизировать отклонения между результатами нечеткого моделирования и экспериментальными данными.

Главное окно пакета ANFIS представлено на рисунке 1.2.

К сожалению, данный пакет распространяется по довольно дорогостоящей лицензии в составе программного пакета MatLAB.

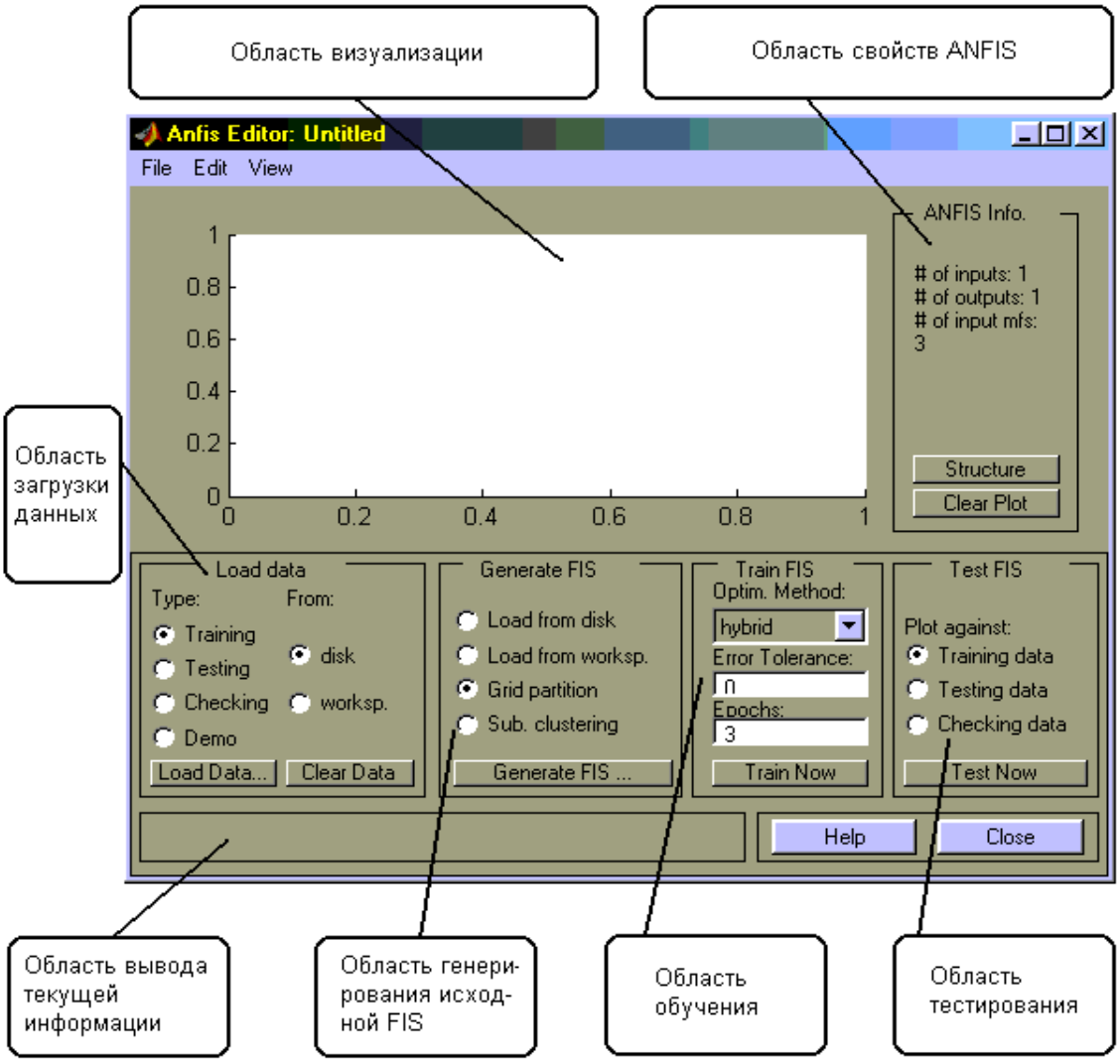

Рисунок 1.2 — Главное окно пакета ANFIS

1.3.2 STATISTICA Automated Neural Networks

Нейронную сеть для прогнозирования данных предоставляет пакет STATISTICA. В данном пакете имеется возможность настроить параметры обучения: тип нейронной сети (радиально-базисных функций или многослойный персептрон), функцию активации, минимальное и максимальное количество нейронов в скрытом слое, функцию ошибки и другие. STATISTICA Automated Neural Networks является одним из самых передовых и самых эффективных нейросетевых продуктов на рынке. Он предлагает множество уникальных преимуществ и богатых возможностей. Например, уникальные возможности инструмента автоматического нейросетевого поиска, Автоматизированная нейронная сеть (АНС), позволяют использовать систему не только экспертам по нейронным сетям, но и новичкам в области нейросетевых вычислений. Кроме того, присутствует возможность импортировать полученную нейронную сеть в различные языки программирования, такие как Java или C++.

Однако, богатый функционал данного ПО подразумевает несколько неудобств. Во-первых, это довольно высокая планка вхождения – для того, чтоб полноценно работать с STATISTICA необходимо потратить определенное время на то, чтоб разобраться в довольно запутанном интерфейсе, а также точно знать, какие цели необходимо достигнуть. Во-вторых, не всегда есть необходимость во всем богатом функционале, а платить приходится за пакет целиком.

Интерфейс пакета STATISTICA представлен на рисунке 1.3.

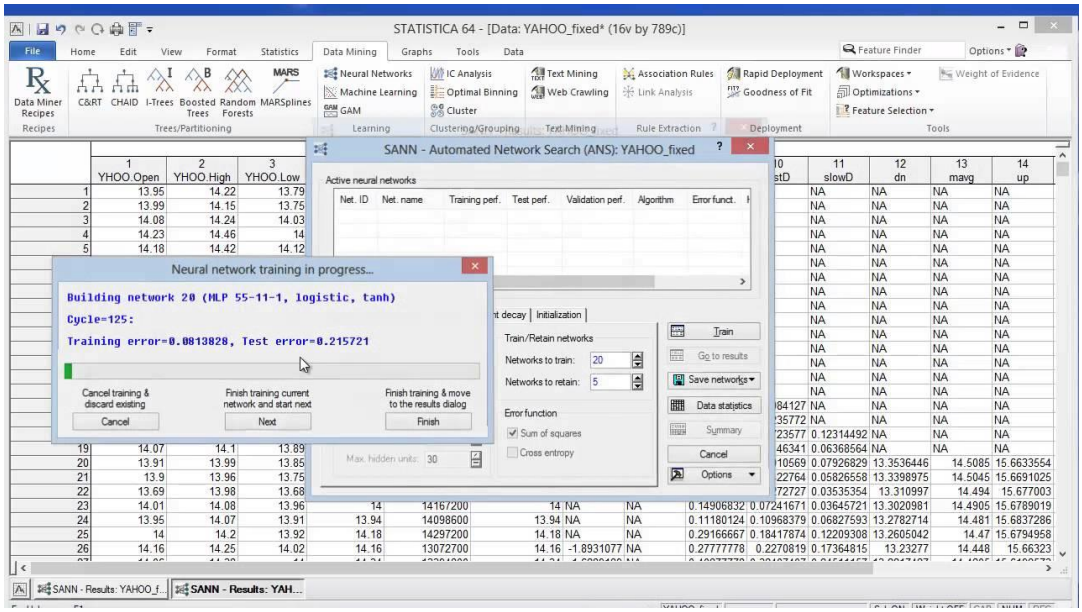

Рисунок 1.3 — Окно пакета STATISTICA

Так, в данной главе были описаны задачи управления технологическими процессами. В частности, были описаны классы решаемых задач при управлении технологическими процессами. Так же были описаны преимущества методов машинного обучения, и в частности нейронных сетей при решении таких задач.

Кроме того, были описаны основные архитектуры нейронных сетей, которые используются в задачах управления технологическими процессами. Были проанализированы недостатки и преимущества рассмотренных архитектур, рассмотрены возможности по применению различных архитектур применительно к задачам по управлению технологическими процессами и выбрана наиболее подходящая архитектура – искусственная нейронная сеть радиально-базисных функций.

Также были рассмотрены существующие на рынке решения, в частности пакет MatLAB ANFIS и пакет STATISTICA Automated Neural Networks. Были описаны их преимущества и недостатки.

Таким образом, учитывая все вышесказанное, можно приступать к проектированию и разработке искусственной нейронной сети.

2 РАЗРАБОТКА И ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ИСКУССТВЕННОЙ НЕЙРОННОЙ СЕТИ

2.1 Структура радиально-базисной нейронной сети

Радиально-базисная нейронная сеть состоит из лишь одного скрытого слоя и одного выходного слоя. В различной литературе входной слой сети радиально-базисных функций может как учитываться, так и опускаться. Это связано с тем, что входной слой такой сети, по сути, является вектором. Иными словами, входной слой можно рассматривать как набор нейронов, каждый из которых является одной из координат такого вектора, но точно так же можно опускать данное допущение.

Суть работы радиально-базисной нейронной сети заключается в следующем. Входной вектор поочередно «показывается» каждому из нейронов скрытого слоя, в которых заранее установлены эталонные вектора. Далее входной вектор сравнивается с эталонным. Для сравнения можно использовать любую метрику, например, расстояние Манхэттана, но чаще используется евклидово расстояние. Полученный результат сравнения обрабатывается активационной радиально-базисной функцией. Далее полученный результат умножается на вес синапса нейрона и передается выходному слою. Нейроны выходного слоя суммируют полученные сигналы и выдают ответ, на этом работу сети можно считать оконченной.

Общая структура радиально-базисной сети представлена на рисунке 2.1.

Использованы обозначения: n – количество элементов в первом слое; x_1, x_2, …, x_b – координаты входного вектора; b – количество нейронов во втором слое; c_{i1}, c_{i2}, ..., c_{ib} – координаты центра i-го элемента; σ_{ij} – ширина радиальной функции i-го элемента; θ_i – выходной сигнал i-го элемента; w_i – весовой коэффициент выходной связи i-го элемента; w_0-вес нейрона смещения; y – выходной сигнал сети.

Выходной сигнал каждого элемента определяется функцией Гаусса. Однако, в зависимости от решаемых искусственной нейронной сетью задач, функция Гаусса может незначительно изменяться. Так, в случае решения задач регрессии или прогнозирования функция Гаусса будет выглядеть следующим образом:

$$f\ x_1, x_2, \ldots x_m\ = g(w_0 + \ \sum_{i=1}^{b} w_i \exp(-\ \sum_{j=1}^{m} \frac{a_j^2\ x_j - c_{ij}\ ^2}{2\sigma_{ij}^2}\ ), \qquad (2.1)$$

где функция g является функцией идентификации.

В случае решения задач прогнозирования количество нейронов в выходном слое должно быть равно количеству классов, и основное отличие от задач регрессии состоит в том, что у каждого нейрона выхода будет свое значение веса нейрона смещения. Иными словами, формула (1) будет выглядеть следующим образом:

$$f\ x_1, x_2, \ldots x_m\ = g(w_{l0} + \ \sum_{i=1}^{b} w_{li} \exp(-\ \sum_{j=1}^{m} \frac{a_j^2\ x_j - c_{ij}\ ^2}{2\sigma_{ij}^2}\ ) \qquad (2.2)$$

Выходным сигналом нейронной сети будет являться взвешенная сумма сигналов элементов:

$$y = \ \sum_{i=1}^{m} w_i \cdot \theta_i. \qquad (2.3)$$

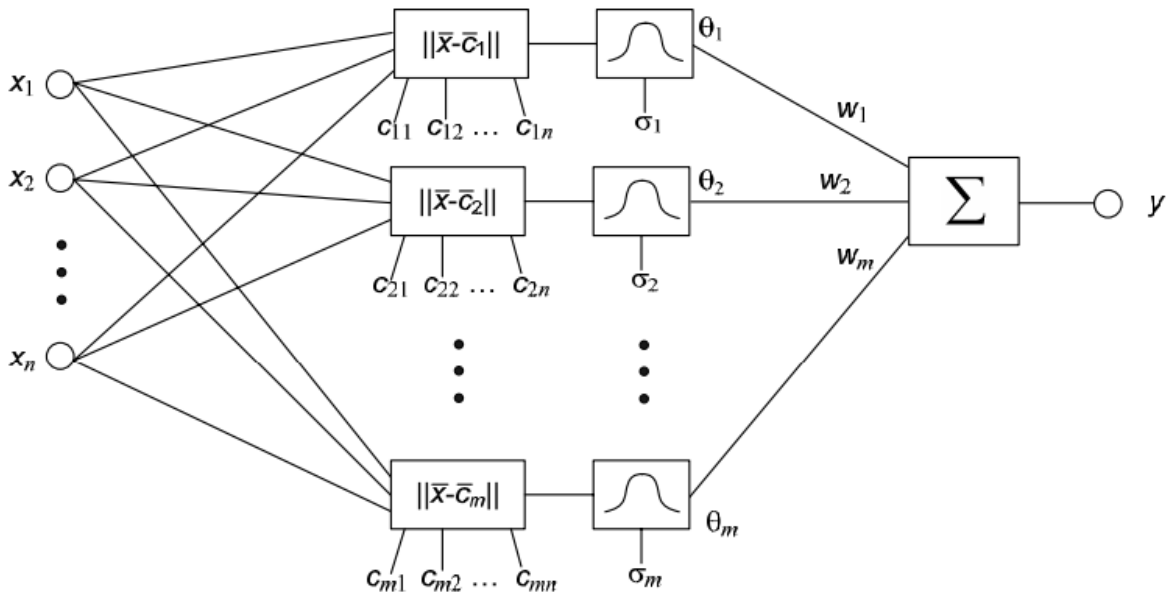

Рисунок 2.1 — Общая структура радиально-базисной сети

Отдельное внимание стоит уделить радиальному нейрону.

Нейроны радиального типа представляют собой естественное дополнение сигмоидальных нейронов. Сигмоидальный нейрон представляется в многомерном пространстве гиперплоскостью, которая разделяет это пространство на два класса, в которых выполняется одно из двух условий. Радиальный нейрон же представляет собой гиперсферу, которая осуществляет шаровое разделение пространства вокруг центральной точки. В случае круговой симметрии данных, такая структура позволяет заметно уменьшить количество нейронов, необходимых для разделения различных классов.

Для обучения радиально-базисной сети используется алгоритм обратного распространения ошибки, который основан на минимизации целевой функции ошибки сети.

Так, описав общую структуру радиально-базисной нейронной сети и модель радиального нейрона, можно приступить к описанию алгоритма обратного распространения ошибки.

2.2 Применение алгоритма обратного распространения ошибки для радиально-базисных сетей.

В случаях, когда имеется возможность оценить эффективность работы искусственной нейронной сети, обучение такой сети можно представить как задачу оптимизации.

Одним из способов такой оптимизации является алгоритм обратного распространения ошибки.

Обучение с помощью такого алгоритма происходит в два прохода по сети – прямого и обратного.

При прямом проходе входной вектор подается на каждый из нейронов нейронной сети, после чего генерируется набор выходных сигналов, иными словами – получается реакция нейронной сети на данный входной вектор.

Во время обратного прохода фактический результат нейронной сети, полученный во время прямого прохода, вычитается из желаемого и таким образом высчитывается ошибка нейронной сети. Далее сигнал этой ошибки

распространяется по сети в направлении обратном направлению синаптических связей и по определенным правилам пересчитываются различные параметры нейронной сети.

В процессе оптимизации работы искусственной нейронной сети посредством алгоритма обратного распространения ошибки можно модифицировать веса нейронов скрытого слоя, координаты эталонных векторов каждого из нейронов скрытого слоя, и ширину радиальной функции σ.

Отдельным классом алгоритмов обучения радиально-базисных сетей являются градиентные алгоритмы. Их основу составляет целевая функция, которая для одного обучающего примера имеет вид:

$$E = \frac{1}{2}\left[\sum_{i=1}^{K} w_i \phi_i\ x\ - d\right]^2 \tag{2.4}$$

Предположим, что применяется гауссовская радиальная функция вида

$$\begin{gathered}\phi_i\ x\ t\ = \exp\ -\frac{1}{2} u_i\ t\ \text{, где} \\ u_i(t) = \sum_{j=1}^{N} \frac{x_j(t) - c_{ij}(t)^2}{\sigma_{ij}^2(t)}\text{, а}\end{gathered} \tag{2.5}$$

i- индекс нейрона скрытого слоя, *j*-индекс компонента входного вектора, *t*- индекс обучающего примера в выборке.

Существует два способа градиентного спуска в реализации алгоритма обратного распространения ошибки:

1. Стохастическая реализация – в случае этого варианта реализации оптимизируемые элементы изменяются сразу же после расчета ошибки, «на месте».

2. Пакетная реализация – для этого варианта реализации функция потерь вычисляется по окончанию эпохи (однократному проходу по обучающему набору данных) для всех элементов вместе взятых, и только после вносятся поправки в оптимизируемые элементы.

У каждой из этих реализаций существуют как достоинства, так и недостатки.

Так, для стохастической реализации характерно присутствие вносимых шумов в процессе обучения. Кроме того, этот метод принято считать более медленным. Однако, стохастическая реализация имеет меньше затруднений в случае попадания в локальные минимумы функции.

Пакетная же реализация немного более стабильна, не вносит шумов и более быстрая, но при этом имеет определенные трудности в случае столкновения с локальными минимумами. Кроме того, такая реализация считается более сложной в плане внедрения.

В ходе данной работы рассматривается стохастическая реализация алгоритма обратного распространения ошибки.

Обучение сети с использованием алгоритма обратного распространения ошибки проводится в два этапа. На первом этапе предъявляется обучающий пример, а затем рассчитываются значения сигналов выходных нейронов сети и значение целевой функции, заданной выражением (2.4). На втором этапе минимизируется значение этой функции.

Подбор значений параметров можно осуществлять, используя градиентные методы оптимизации независимо от объекта обучения – будь то вес или центр. Независимо от выбираемого метода градиентной оптимизации, необходимо, прежде всего, получить вектор градиента целевой функции относительно всех параметров сети. Очевидно, что для градиентного спуска по функции, эта функция должна быть дифференцируема. Продифференцировав целевую функцию получим:

$$\frac{\partial E(t)}{\partial w_0(t)} = y\ t\ - d(t); \tag{2.6}$$

$$\frac{\partial \mathrm{E(t)}}{\partial \mathrm{w_i(t)}} = \exp\ -\frac{1}{2}\mathrm{u_i}\ \mathrm{t}\quad (\mathrm{y}\ \mathrm{t}\ - \mathrm{d(t)}; \tag{2.7}$$

$$\frac{\partial E(t)}{\partial c_{ij}(t)} = \ y\ t\ - d\ t\ \ w_i\ t\ \exp\ -\frac{1}{2}u_i(t)\ \ \frac{x_j(t)-c_{ij}(t)}{\sigma_{ij}^2(t)}; \tag{2.8}$$

$$\frac{\partial E(t)}{\partial \sigma_{ij}(t)} = \ y\ t\ - d\ t\ \ w_i\ t\ \exp\ -\frac{1}{2}u_i(t)\ \ \frac{x_j(t)-c_{ij}(t)}{\sigma_{ij}^3(t)}; \tag{2.9}$$

Формулы (2.6) – (2.9) принято называть дельтами.

При использовании метода наискорейшего спуска формулы для корректировки параметров радиально-базисной сети примут следующий вид для весов:

$$w_i\ t+1\ = w_i\ t\ - \eta \frac{\partial E(t)}{\partial w_i(t)}, \tag{2.10}$$

для координат эталонного вектора:

$$c_i\ t+1\ = c_{ij}\ t\ - \eta \frac{\partial E(t)}{\partial c_{ij}(t)}, \tag{2.11}$$

для ширины радиальной функции каждого нейрона:

$$\sigma_{ij}\ t+1\ = \sigma_{ij}\ t\ - \eta \frac{\partial E(t)}{\partial \sigma_{ij}(t)}. \tag{2.12}$$

Как можно видеть, формулы (2.10) - (2.12) являют собой разность между текущим значением оптимизируемого значения и дельтами, т.е. формулами (2.7) – (2.9), умноженными на коэффициент обучения η.

Так, определившись с математической составляющей нейронной сети и ее обобщенной структурой, можно приступать к ее проектированию и реализации.

2.3 Проектирование программной реализации нейронной сети

Язык UML (Unified Modeling Language) - унифицированный язык визуального моделирования, разработанный для визуализации, создания спецификации, документирования и конструирования бизнесс-процессов и компонентов программного обеспечения. Язык UML может быть использован для построения логических и концептуальных моделей сложных систем самого различного целевого назначения. [5].

В рамках языка UML все представления о модели сложной системы представляются в виде специальных графических конструкций, называемых диаграммами. В совокупности построенных диаграмм содержится вся информация, необходимая для реализации проекта сложной системы. Остановимся на основных видах диаграмм.

2.3.1 Диаграмма классов

Диаграмма классов служит для представления статической структурной модели системы в терминологии классов объектно-ориентированного проектирования. Диаграмма классов может отражать, в частности, различные взаимосвязи между отдельными сущностями предметной области, а также описывает их внутреннюю структуру и типы отношений. При этом на данной диаграмме не указывается информация о временных аспектах функционирования системы.

Диаграмма классов по сути является графом, вершины которого - элементы типа «классификатор», связанные различными типами структурных отношений, среди которых выделяют следующие:

- ассоциация – соответствует наличию некоторого отношения между классами;
- агрегация – частный случай ассоциации, когда один из классов представляет собой некоторую сущность, включающую в себя в качестве составных частей другие сущности;
- композиция – частный случай агрегации, при котором выделяется специальная форма отношения «часть-целое», при которой составляющие части не могут существовать в отрыве от целого. Иными словами, с уничтожением целого уничтожаются и все его части;
- зависимость – указывает некоторое семантическое отношение между двумя элементами модели или двумя множествами таких элементов, выраженное в том, что некоторое изменение одного элемента модели может потребовать изменения другого зависимого от него элемента модели.
- обобщение – отношение между более общим элементом (родителем или предком) и более частным и специальным элементом (потомком или дочерним);

На рисунке 2.2 изображена диаграмма классов системы. В результате анализа предметной области выделены следующие сущности: нейронная сеть, нейрон, класс обучения нейронной сети.

Рисунок 2.2 — Диаграмма классов нейронной сети

Так, описав диаграмму классов реализуемой нейронной сети, можно приступить к описанию диаграммы деятельности, описывающей функционирование реализуемой нейронной сети.

2.3.2 Диаграмма деятельности

Диаграммы деятельности позволяют описать особенности процедурного и синхронного управления, обусловленного выполнением внутренних деятельностей и действий (элементарных операций). Диаграмма деятельности является специальным видом диаграммы состояний, на которой отсутствует сигнатура событий, а для отображения действий введены специальные состояния.

При этом на этой диаграмме допускается использование и обычных состояний. Основным направлением использования диаграмм деятельности является визуализация особенностей реализации методов классов, когда необходимо представить алгоритмы их выполнения. При этом каждое состояние может являться выполнением операции некоторого класса либо ее части.

Для описания условных переходов используются символы ветвления. В частности, символы разделения и слияния (concurrent join) используются для разделения и слияния параллельных вычислений или потоков управления соответственно.

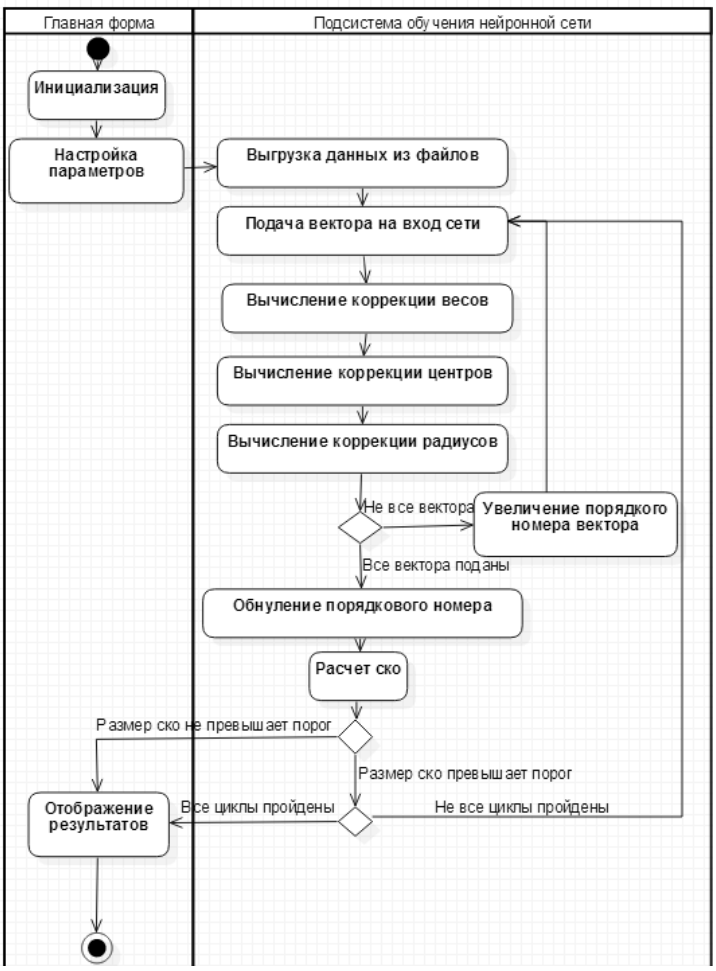

Рисунок 2.3 — Диаграмма деятельности

Дорожки позволяют разделять выполнение процессов между объектами или классами, а также, в случае описания бизнес-процесса с помощью диаграммы деятельности, и между подразделениями подразделениями.

Диаграммы деятельности состоят из ограниченного количества фигур, соединённых стрелками. Основные фигуры, используемые в диаграммах действий описывают следующие явления:

- прямоугольники с закруглениями — действия;
- ромбы — решения;

- широкие полосы — начало (разветвление) и окончание (схождение) ветвления действий;
- чёрный круг — начало процесса (начальный узел);
- чёрный круг с обводкой - окончание процесса (финальный узел);

Стрелки в диаграмме деятельности идут от начала к концу процесса и показывают потоки управления или потоки объектов (данных).

Диаграмма деятельности, описывающая алгоритм обучения нейронной сети, представлена на рисунке 2.3.

Так, определившись с общей архитектурой проектируемой нейронной сети и описав диаграмму ее деятельности, следует приступить к выбору средств реализации данной нейронной сети.

2.4 Обоснование средств реализации искусственной нейронной сети

Среди множества существующих на данный момент языков программирования одним из самых популярных является язык программирования Java. История создания языка Java начинается в июне 1991 года, когда Джеймс Гослинг создал проект для использования в одном из своих многочисленных проектов, но официальной датой создания языка Java считается 23 мая 1995 года, после выпуска компанией Sun первой реализации Java 1.0. С тех пор популярность Java только росла и на данный момент Java является одним из самых востребованных языков на рынке труда [6]. Такая популярность связана с рядом преимуществ данного языка. Ниже перечислены основные из них:

- объектно-ориентированность: все в Java является объектом. В случае, если программист ознакомлен с основными принципами ООП, работа с Java не должна составлять для него труда;
- кроссплатформенность: весь код, написанный на Java, транслируется в байт-код и передается виртуальной машине Java – JVM. Таким образом, избегается привязка к конкретной операционной системе –

если на компьютере установлена реализация JVM, то абсолютно неважно, под управлением какой операционной системы он находится, будь то ОС семейства Linux или Windows;

- открытая и обширная документация: на сайте разработчика достаточно подробно расписаны классы и их методы стандартных библиотек Java. Если у программиста возникают сложности всегда можно свериться с официальной документацией языка и найти ответы на все интересующие вопросы.

Так, определившись с архитектурой искусственной нейронной сети и средствами ее реализации, можно приступить к описанию данной реализации.

2.5 Описание основных алгоритмов реализованной искусственной нейронной сети

При запуске программы пользователь вводит параметры конфигурации искусственной сети, а также название файла, в котором находится обучающая выборка. Обучающая выборка представляет собой набор строк, каждая из которых, в свою очередь, являет собой набор разделенных табуляциями числовых значений.

Помимо вышеперечисленного, пользователь вводит количество циклов обучения нейронной сети и желаемое максимальное значение среднеквадратичной ошибки.

Количество циклов обучения необходимо в том случае, если среднеквадратичная ошибка нейронной сети по каким-то причинам не сходится к желаемой пользователем. В таком случае нейронная сеть может обучаться неограниченное количество времени и навсегда «застрять» внутри цикла обучения. Для того, чтоб избежать такой ситуации, и нужно максимальное количество циклов обучения – даже если среднеквадратичная ошибка сети не сошлась к желаемой, то программа все равно завершит работу.

После этого происходит инициализация нейронной сети, в частности инициализируются нейроны скрытого слоя, если конкретнее, присваиваются случайные значения сигм и весов. Помимо этого, каждому нейрону присваивается случайный вектор из обучающей выборки.

Так же инициализируется объект класса BackPropagation, иными словами инициализируется объект, отвечающий за обучение нейронной сети. В процессе инициализации данного объекта загружается файл обучающей выборки.

После этого запускается непосредственно алгоритм обучения искусственной нейронной сети, а именно алгоритм обратного распространения ошибки.

Как было описано выше, в ходе этого алгоритма сначала высчитывается выходное значение сети на обучающей выборке, полученное значение сравнивается с желаемым, после чего происходит переобучение сети, а именно изменение значений весов нейронов скрытого слоя, сигм, и координат эталонных векторов.

Для того, чтобы найти выходное значение сети, необходимо воспользоваться формулой (2.3). Иными словами, необходимо рассчитать значение функции Гаусса для каждого из нейронов, после чего умножить на значение веса соответствующего нейрона и просуммировать полученные от всех нейронов результаты.

За расчет значения функции Гаусса отвечают два метода – eucklidNorm(), отвечающий за расчет евклидова расстояния и деление полученного значения на квадрат сигмы и calculateGauss(), непосредственно рассчитывающий значение функции Гаусса из полученного в ходе выполнения eucklidNorm() значения.

Листинг методов eucklidNorm() и calculateGauss() представлены на рисунке 2.4.

После того, как найден выходной сигнал сети, начинается обратный ход алгоритма обратного распространения ошибки.

В процессе обратного хода высчитываются изменения весов, координат эталонных векторов и сигм для каждого нейрона сети.

```
public double calculateGauss(double[] actual, int index) {
    return Math.exp(-0.5 * (this.eucklidNorm(actual,
            arrayofneurons[index].c,arrayofneurons[index].sigma)));
}

public double eucklidNorm(double[] actual, double[] ci, double[] sigma) {
    double res = 0;
    for (int i = 0; i < actual.length; i++) {
        res += ((Math.pow(actual[i] - ci[i], 2)) / Math.pow(sigma[i], 2));
    }
    return res;
}
```

Рисунок 2.4 — Листинг методов eucklidNorm() и calculateGauss()

В частности, для расчета значений формул (2.8) и (2.9), используются методы gradientCenters () и gradientSigmas () соответственно.

Листинг методов gradientCenters() и gradientSigmas() представлен на рисунке 2.5.

```
public double gradientCenters(double[] actual, int indexI, int indexJ) {
    return lastneuron.getWeight(indexI)
            * ((actual[indexJ] - arrayofneurons[indexI].c[indexJ])
            / Math.pow(arrayofneurons[indexI].sigma[indexJ], 2))
            * (calculateGauss(actual, indexI));
}

public double gradientSigmas(double[] actual, int indexI, int indexJ) {
    return lastneuron.getWeight(indexI)
            * ((actual[indexJ] - arrayofneurons[indexI].c[indexJ])
            / Math.pow(arrayofneurons[indexI].sigma[indexJ], 3))
            * (calculateGauss(actual, indexI));
}
```

Рисунок 2.5 — Листинг методов gradientCenters() и gradientSigmas().

После проведения перерасчета сигм, весов и координат эталонных векторов, рассчитывается новый выходной сигнал нейронной сети и среднеквадратичная ошибка. Если полученная среднеквадратичная ошибка меньше желаемой пользователем, обучение можно считать законченным. В противном случае обучение продолжается до тех пор, пока не выполнится одно из двух условий:

- значение среднеквадратичной ошибки меньше желаемого пользователем;
- не пройдены все циклы обучения.

По завершению работы программа выводит в консоль значение среднеквадратичной ошибки, а также построчно записывает в файл полученные в ходе обучения нейронной сети значения.

Из недостатков данной реализации можно отметить отсутствие возможности сохранить обученную нейронную сеть в файл, для дальнейшей загрузки и работы с ней. Также к недостаткам можно отнести случайный выбор эталонных векторов, а не предварительная кластеризация посредством, например, алгоритма k-means.

Полный код реализованной программы приведен в приложении А.

Так, в данной главе была описана структура искусственной нейронной сети радиально-базисных функций, так же описана модель нейрона такой сети и различие такой модели от сигмоидального нейрона, используемого в, например, многослойном персептроне.

Так же был описан алгоритм обратного распространения ошибки, описаны реализации градиентного спуска в таком алгоритме, сравнены преимущества и недостатки этих реализаций.

Кроме того, была спроектирована диаграмма классов и диаграмма деятельности реализуемой искусственной нейронной сети радиально-базисных функций, а так же обоснованы средства реализации такой сети.

Помимо этого в общих чертах была описана реализация искусственной нейронной сети радиально-базисных функций.

Так, спроектировав искусственную нейронную сеть радиально-базисных функций, определившись с инструментами для ее реализации, и описав эту реализацию, можно приступать непосредственно к исследовательской части, а именно исследовать эффективность обучения реализованной сети.

3 ИССЛЕДОВАНИЕ ЭФФЕКТИВНОСТИ НЕЙРОННОЙ СЕТИ

3.1 Общие сведения о гиперпараметрах нейронной сети

Важным элементом конфигурирования искусственной нейронной сети является подбор так называемых гиперпараметров. Гиперпараметры искусственной нейронной сети в общем случае можно разделить на две группы: глобальные и локальные (узловые).

К глобальным гиперпараметрам относятся количество скрытых слоев, количество нейронов в каждом слое, уровень обучения и момент, инициализация весов нейронов.

Локальные гиперпараметры — тип слоя, функция активации и другие параметры регуляризации.

Так как большая часть локальных гиперпараметров, по сути, определено выбранной структурой нейронной сети, а именно сети радиально-базисных функций, то в ходе данной главы будут рассматриваться глобальные гиперпараметры, а именно:

- коэффициент обучения η;
- момент обучения m;
- количество нейронов в скрытом слое искусственной нейронной сети.

Для того, чтобы подобрать перечисленные выше гиперпараметры, необходимо провести исследование их влияния на прогнозируемую ошибку искусственной нейронной сети.

Для того, чтобы оценить эффективность обучения используется множество различных методов. Одним из самых распространенных является RMSE – корень среднеквадратического отклонения. RMSE является показателем рассеивания значений случайной величины относительно её математического ожидания. Вычислить корень среднеквадратического отклонения можно по следующей формуле:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - x)^2} \qquad (3.1)$$

Не будет лишним отметить, что оценка отклонения, полученная при расчете RMSE, является смещенной. Однако, данную оценку все равно можно считать состоятельной [7].

Нейронная сеть обучалась на выборке следующего вида: набор строк из произвольного числа значений, разделенных табуляцией. Последнее значение каждой строки – ожидаемое значение.

Данные обучающей выборки получены с ООО «ТСК-Штамп» и являются показаниями различных счетчиков, используемых при контактной сварке.

Вид обучающей выборки представлен на рисунке 3.1

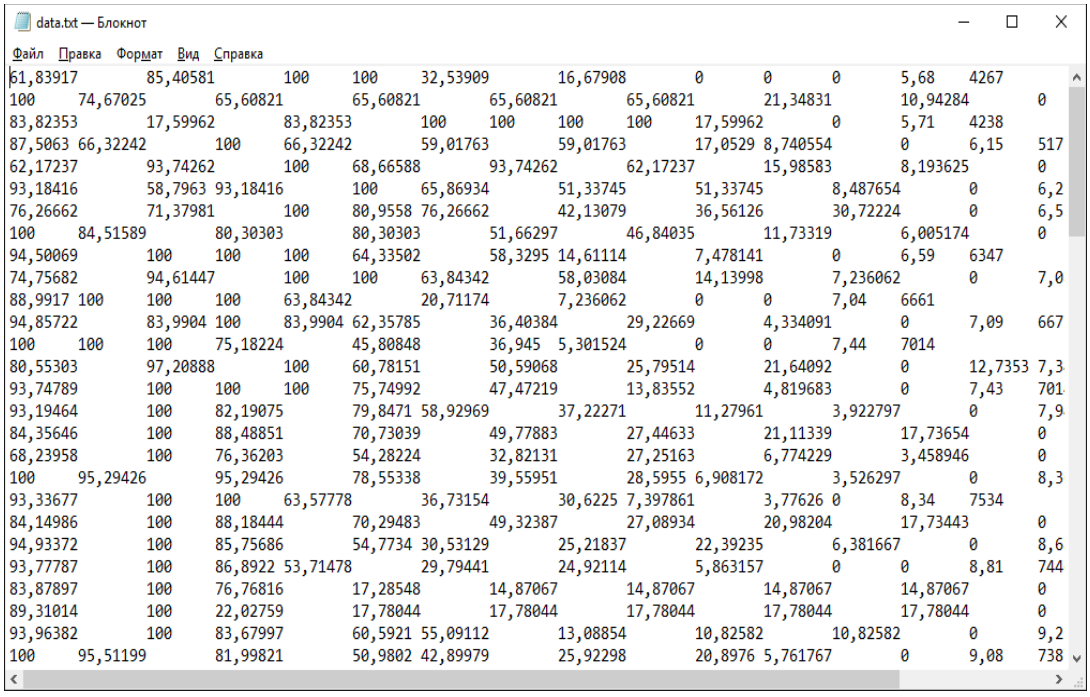

Рисунок 3.1 — Вид обучающей выборки

По завершению обучения реализованная программа выводит в консоль значение среднеквадратичной ошибки, а в файл выводит спрогнозированные значения, по одному значению на строку.

В связи со стохастичностью каждого переобучении нейронной сети в качестве результирующего значения RMSE бралось среднее за 20 переобучений.

Вывод программы в консоль и в файл представлен на рисунке 3.2.

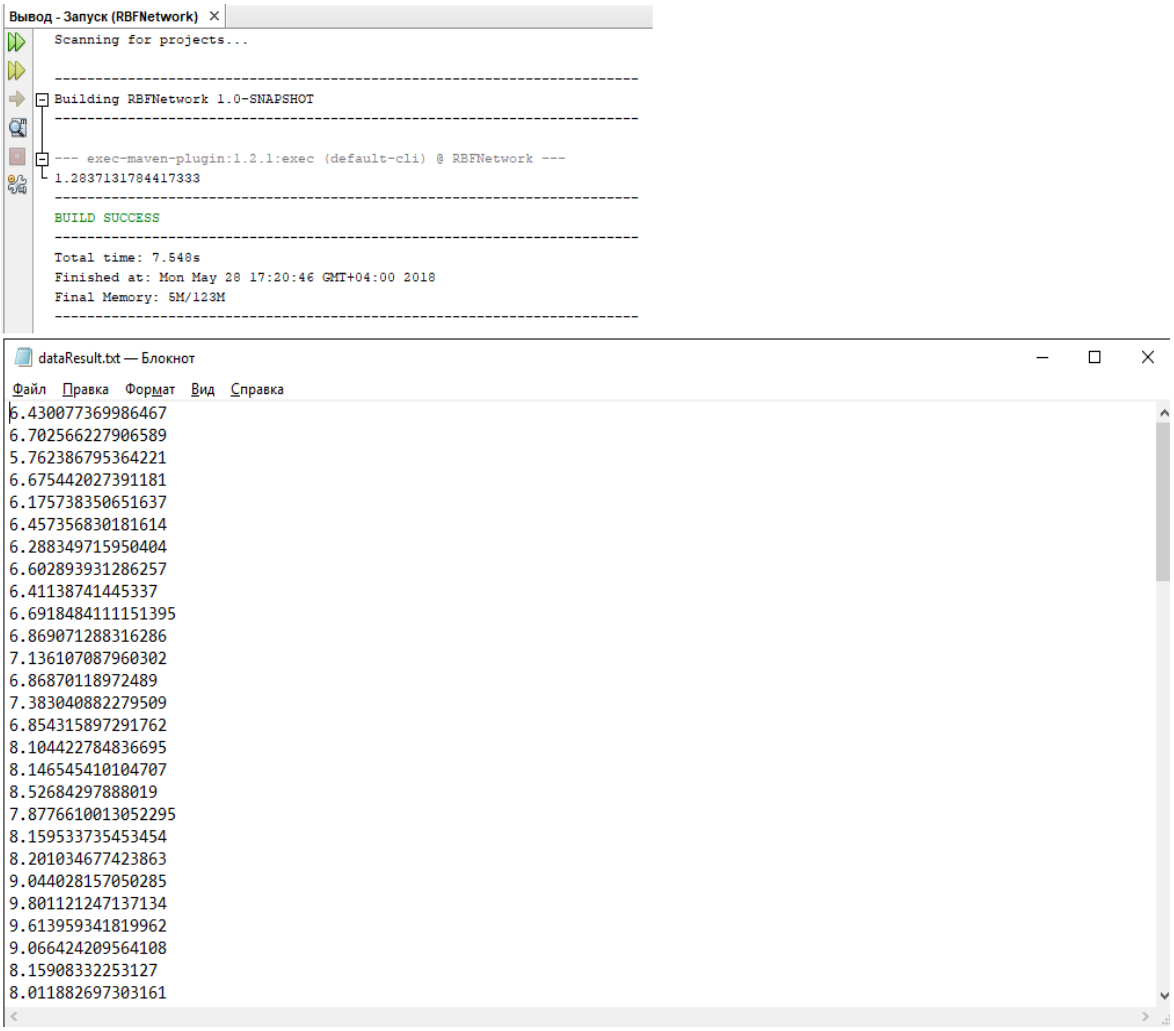

Рисунок 3.2 — Вывод программы в консоль и в файл

Так, описав формат входных и выходных данных, а также описав что из себя являет среднеквадратичная ошибка, можно приступить к исследованию влияния гиперпараметров обучения на эффективность обучения нейронной сети.

3.2 Исследование зависимости эффективности алгоритма обучения от значения коэффициента обучения

В процессе обучения нейронной сети коэффициент обучения учитывается при обновлении весов нейрона, координат эталонного вектора и ширины радиальной функции во время работы алгоритма обратного распространения ошибки с учетом моментов. Значение коэффициента обучения η эмпирически выбирают из интервала (0;1). Обновление весов определяется следующей формулой:

$$\Delta w_{ij}\ t+1\ = -\eta\delta_i x_i + \alpha\Delta w_{ij}(t) \quad (3.1)$$

где $\delta_i x_i$ - i-я составляющая градиента целевой функции, а $w_{ij}(t+1)$ и $w_{ij}(t)$ – веса на следующем и предыдущих этапах обучения соответственно.

Далее исследуем зависимость RMSE от коэффициента обучения η. Параметры обучения следующие:

- тестовая выборка – 200 векторов;
- число итераций обучения: 15000;
- момент m = 0.1;
- количество нейронов в скрытом слое n = 7.

Результаты исследования представлены в таблице 3.1

Таблица 3.1 — Зависимость RMSE от коэффициента обучения

Коэффициент обучения	RMSE
0.005	0.7264
0.01	0.7036
0.02	0.7226
0.03	0.6822
0.04	0.7071
0.05	0.7231
0.1	0.7636
0.2	0.8364
0.3	0.8647
0.4	0.8897
0.6	1.0384
0.8	1.0752

График зависимости RMSE от коэффициента обучения представлен на рисунке 3.3.

Как можно видеть из рисунка 3.3 и таблицы 3.1, с данной обучающей выборкой СКО растет с увеличением коэффициента обучения, а оптимальным значением коэффициента обучения будет η = 0.03.

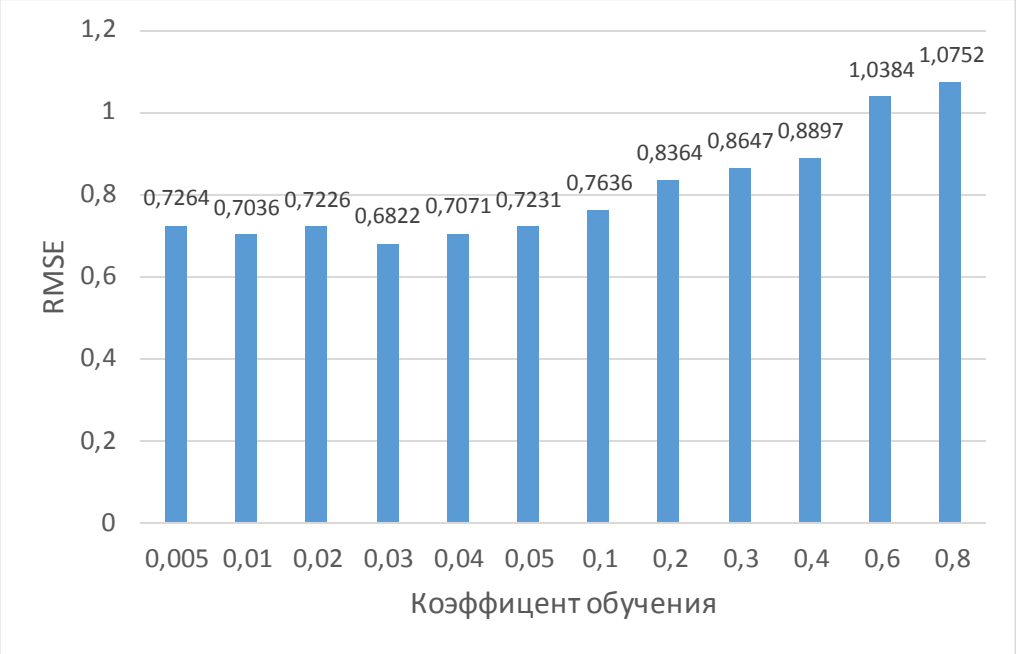

Рисунок 3.3 — Зависимость RMSE от коэффициента обучения

Так, рассмотрев зависимость погрешности прогнозирования от скорости обучения нейронной сети, можно приступить к исследованию зависимости погрешности прогнозирования от обучающего момента.

3.3 Исследование зависимости погрешности прогнозирования от момента

При обучении искусственной нейронной сети методом градиентного спуска может возникнуть ситуация, в которой обучение останавливается, находя локальный минимум оптимизируемой функции. Для того, чтобы избежать такой ситуации, в формулу изменения весов нейрона вводится дополнительный параметр – момент m. Так же, как и в случае с коэффициентом обучения η момент подбирается эмпирическим путем и его значение находится в диапазоне (0;1).

Далее исследуем зависимость СКО от момента m. Параметры обучения следующие:

- тестовая выборка – 200 векторов;
- число итераций обучения: 15000;
- коэффициент обучения η = 0.03;
- количество нейронов в скрытом слое n = 7.

Результаты исследования представлены в таблице 3.2.

График зависимости среднеквадратичного отклонения от момента обучения представлен на рисунке 3.4.

Таблица 3.2 — Зависимость RMSE от момента обучения

Момент обучения	RMSE
0.005	0.6906
0.01	0.6847
0.05	0.6604
0.1	0.7636
0.3	0.6585
0.5	0.6142
0.7	0.6768
0.9	0.8872

Как можно видеть из рисунка 3.1 и таблицы 3.1, оптимальным значением момента обучения будет m = 0.5.

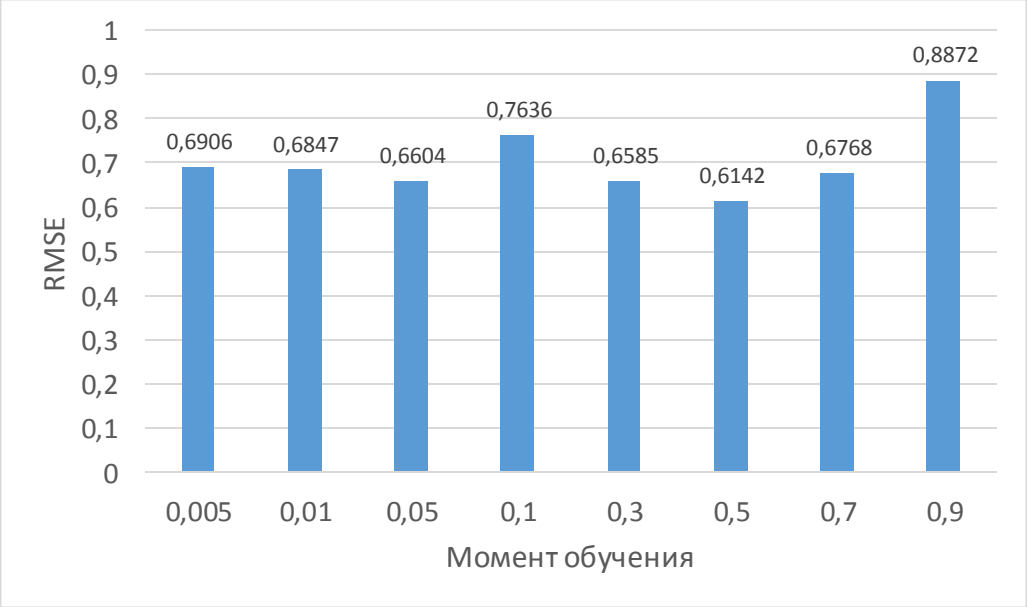

Рисунок 3.4 — Зависимость RMSE от момента обучения

Так, рассмотрев зависимость погрешности прогнозирования от момента обучения нейронной сети, можно приступить к исследованию зависимости погрешности прогнозирования от количества нейронов в скрытом слое нейронной сети.

3.4 Исследование зависимости эффективности алгоритма обучения от количества нейронов скрытого слоя

Количество нейронов в скрытом слое нейронной сети также является важным гиперпараметром при настройке конфигурации нейронной сети.

Так, при слишком низком количестве нейронов в скрытом слое нейронная сеть может быть подвержена большому разбросу выдаваемых данных, а при слишком низком количестве нейронов могут возникать шумы.

Исследуем зависимость RMSE тестирования от числа нейронов в скрытом слое при следующих параметрах обучения:

- тестовая выборка – 200 векторов;
- число итераций обучения: 15000;
- коэффициент обучения η = 0.03;
- момент обучения m = 0.5.

Результаты исследования представлены в таблице 3.3.

Таблица 3.3 — Зависимость RMSE от количества нейронов в скрытом слое

Количество нейронов	RMSE
3	1.200
5	0.8633
7	0.8132
9	0.7636
11	0.9192
13	0.9399
15	0.9811
20	1.003
40	1.26

График зависимости RMSE от количества нейронов в скрытом слое представлен на рисунке 3.5.

Как можно убедиться из рисунка 3.5 и таблицы 3.3 нейронная сеть при слишком маленьком количестве нейронов допускает значительные ошибки. При увеличении количества нейронов в скрытом слое ошибка падет, однако если продолжать увеличивать количество нейронов ошибка снова начинает расти. Оптимальным количеством нейронов будет являться n=9.

Рисунок 3.5 — Зависимость RMSE от количества нейронов в скрытом слое

Таким образом, в ходе данной главы были описаны основные гиперпараметры искусственной нейронной сети, дано формальное определение ошибке, выдаваемой нейронной сетью, а также исследовано влияние гиперпараметров на прогнозируемую ошибку искусственной нейронной сети и были подобраны оптимальные значения данных гиперпараметров.

Для данной в процессе написания данной работы выборки оптимальными значениями гиперпараметров являются следующие:

- коэффициент обучения η = 0.03;
- количество нейронов в скрытом слое n = 7;
- момент обучения m = 0.5

ЗАКЛЮЧЕНИЕ

Целью данной ВКР являлась разработка искусственной нейронной сети радиально-базисных функций и исследование ее возможностей применительно к задачам управления. Для достижения цели были решены следующие задачи:

- проведен аналитический обзор наиболее популярных архитектур искусственных нейронных сетей;
- проведен разбор предметной области;
- обоснован выбор наиболее оптимального решения;
- спроектирована искусственная нейронная сеть;
- реализована искусственная нейронная сеть на одном из языков программирования;
- проведены исследования эффективности разработанной искусственной нейронной сети.

В ходе данной работы было выявлено, что оптимальным решением для задач управления технологическими процессами является искусственная нейронная сеть радиально-базисных функций.

Была описана общая структура такой нейронной сети, приведена диаграмма ее деятельности.

Помимо этого, в общих чертах была описана реализация описанной нейронной сети на языке программирования Java.

Так же были проведены исследования эффективности нейронной сети, в ходе которых эмпирическим путем были подобраны значения таких гиперпараметров искусственной нейронной сети, как скорость обучения, момент обучения и количество нейронов скрытого слоя.

Оптимальными значениями гипепараметров были выявлены следующие:

- коэффициент обучения η = 0.03;
- количество нейронов в скрытом слое n = 7;

- момент обучения m = 0.5.

Из недостатков реализованной нейронной сети можно отметить отсутствие возможности сохранения ее параметров в файл, а также отсутствие предварительной кластеризации входных данных одним из алгоритмов кластеризации, например, k-means.

Исправление последнего недостатка может являться предметом последующих научных изысканий.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1.Шишов, О.В. Элементы систем автоматизации: предприятие как целостный объект автоматизации: пособие./ О.В. Шишов. — М.: Directmedia, 2015. — 41с. — ISBN 978-5-4475-5276-3

2. АСУТП . ru Искусственный интеллект: принятие решении в сложных системах управления [Электронный ресурс]//АСУТП . ru – средства и системы компьютерной автоматизации. URL: http://asutp.ru/?p=600157 (дата обращения: 17.04.2018)

3. Frank, E. (2014). Fully supervised training of Gaussian radial basis function networks in WEKA (Computer Science Working Papers, 04/2014). Hamilton, NZ: Department of Computer Science, The University of Waikato.

4. Сирота, А.А. Методы и алгоритмы анализа данных и их моделирование в MATLAB/ Учебное пособие/ А.А Сирота. — СПб.: БХВ-Петербург, 2016. — 384с. — ISBN 978-5-9775-3778-0

5. Буч, Г. Введение в UML от создателей языка./Г. Буч , Д. Рамбо, А. Якобсон — М.: ДМК-Пресс, 2015. — 496с.—ISBN 978-5-97060-157-0

6. The 7 Most In-Demand Programming Languages of 2018 | Coding Dojo [Электронный ресурс]. URL: https://www.codingdojo.com/blog/7-most-in-demand-programming-languages-of-2018/ (дата обращения: 11.05.2018)

7. Ивченко, Г. И. Введение в математическую статистику : учебник. Изд. 2-е, испр. и доп./ Г. И. Ивченко, Ю. И. Медведев – СПб.:Ленанд, 2017. — 606с. — ISBN: 978-5-9710-4535-9

8. Schmidhuber, J., Deep learning in neural networks: An overview// Neural Networks. — 2015. —№61. — С.85-117

9. Q. Que, M. Belkin. Back to the future: Radial basis function networks revisited. In AISTATS, pages 1375–1383, 2016.

10. Jaderberg, M., Czarnecki, W. M., Osindero, S., Vinyals, O., Graves, A., and Kavukcuoglu, K. (2016). Decoupled neural interfaces using synthetic gradients. arXiv preprint arXiv:1608.05343.

11. Zhongqi Wang, Bo Yang, Yonggang Kang, and Yuan Yang, “Development of a Prediction Model Based on RBF Neural Network for Sheet Metal Fixture Locating Layout Design and Optimization,” Computational Intelligence and Neuroscience, vol. 2016, Article ID 7620438, 6 pages, 2016

12. Галушкин, А.И. Нейронные сети: история развития теории: учебное пособие/ А.И. Галушкин, Б. Иошуа — М.: АльянС, 2015. — 840с. — ISBN 978-5-91872-067-7

13. Кохонен, Т. Самоорганизующеся карты. / Т. Кохонен; пер.с анг. В. Агеева; ред. Ю.Тюменцев. — М.: Бином. Лаборатория знания, 2017. — 656с. — ISBN 978-5-94774-352-4

14. Рассел, С. Искусственный интеллект. Современный подход. /С. Рассел, П. Норвиг; пер. с анг. К. Птицын; ред. К.Птицын. — М.: Вильямс, 2018. — 1408с. — ISBN 978-5-8459-1968-7

15. Рашид, Т. Создаем Нейронную сеть./ Т.Рашид — М.: Вильямс, 2017. — 272с. — ISBN 978-5-9909445-7-2

16. Флах, П. Машинное обучение. Наука и искусство построения алгоритмов, которые извлекают знания из данных. Учебник. / П. Флах; пер. с анг. А.Слинкин. — М.:ДМК-Пресс, 2015. — 400с. — ISBN 978-5-97060-273-7

17. Хайкин, С. Нейронные сети. Полный курс./С.Хайкин — М.: Вильямс, 2016. — 1104с. — ISBN 978-5-8459-2069-0

18. Хортсманн, К.С. Java. Библиотека профессионала. Том 1. Основы — М.: Вильямс, 2017. — 864с. — ISBN 978-5-8459-2084-3

19. Эккель, Б. Философия Java/ Б. Эккель - СПб.: Питер, 2016. — 1168 с. — ISBN 978-5-496-01127-3

20. Ян, Г. Глубокое обучение/ Б. Иошуа, Г.Ян, К. Аарон. – М.: ДМК-Пресс, 2018. — 652с. — ISBN 978-5-97060-618-6

Приложение А.

Код реализованной нейронной сети

```
package com.shwedsoft.rbfnetwork;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.StringTokenizer;

public class BackPropagation {

    public FileWriter sw;
    public FileWriter scosw;
    public FileReader sr;
    private Network network;
    private double teach;
    private double teach1;
    private double maxCycles;
    private double Error;
    private double moment;
    double[] outputI = new double[300];
    double[] outputF = new double[300];

    ArrayList<Double> values = new ArrayList<>();
    ArrayList<Double> results = new ArrayList<>();

    public BackPropagation(Network network, String fileName, AlgorithmConfig ac,
boolean isLoad) {
        this.network = network;
        this.teach = ac.LearningRate;
        this.teach1 = ac.LearningRate;
        this.moment = ac.Moment;
        this.maxCycles = ac.MaxCycles;
        this.Error = ac.MinError;
        StringTokenizer st;
        try {
            sr = new FileReader(fileName);
            sw = new FileWriter(fileName.replace(".txt", "Result.txt"));
            scosw = new FileWriter("sco.txt");
            BufferedReader bsr = new BufferedReader(sr);
            int j = 0;
            while (bsr.ready()) {
                String text = String.valueOf(bsr.readLine().replace(",", "."));
```

```
            st = new StringTokenizer(text);
            double sum = 0;
            for (int i = 0; i < 9; i++) {
               sum += Double.parseDouble(st.nextToken());

            }
            values.add(sum);
            outputI[j] = Double.parseDouble(st.nextToken());
            outputF[j] = Double.parseDouble(st.nextToken());
            j++;

         }
      } catch (IOException e) {
         e.printStackTrace();
      }
      double min = values.get(0);
      for (double d : values) {
         if (d < min) {
            min = d;
         }
      }
      double max = values.get(0);
      for (double d : values) {
         if (d > max) {
            max = d;
         }
      }
      if (!isLoad) {
         network.initializeCenters(min,  max);
         network.initializeWeights();
         network.initializeSigmas(min,  max);
      }
      this.teachingWeights();
   }

   public ArrayList<Double> getResults() {
      return results;
   }

   public ArrayList<Double> getValues() {
      return this.values;
   }

   public void teachingWeights() {
      int len = values.size() - network.VectorLength;
```

```
        double[] momentArr = new double[network.neurons + 1];//изменение веса в
предыдущем цикле
        for (int i = 0; i < momentArr.length; i++) {
            momentArr[i] = 0;
        }
        double cycles = 0;
        double sco = 0;
        do {
            double[] actual = new double[network.VectorLength];//входной вектор

            for (int start = 0; start < len; start++) {
                for (int i = 0; i < actual.length; i++) {
                    actual[i] = values.get(i + start);
                }
                double output = outputI[start]; //ожидаемые значения
                double[] gausses = new double[network.neurons];//массив вычисленных
функций Гаусса для каждого нейрона
                double[][] gaussesCenters = new double[network.neurons][];//Массив
вычисленных исправленных центров
                double[][] gaussesSigmas = new double[network.neurons][];

                for (int i = 0; i < gausses.length; i++) {
                    gaussesCenters[i] = new
double[network.VectorLength];//инициализация массива изменения центров
нейрона в массиве
                    gaussesSigmas[i] = new double[network.VectorLength];
                    gausses[i] = network.calculateGauss(actual, i);//4.14

                    for (int j = 0; j < network.VectorLength; j++) {
                        gaussesCenters[i][j] = network.gradientCenters(actual, i, j);
                        gaussesSigmas[i][j] = network.gradientSigmas(actual, i, j);
                    }
                }
                double result = 0;//реальное значение на выходе
                for (int i = 0; i < network.neurons; i++) {
                    result += network.calculateGauss(actual, i) *
network.getNeuronWeights(i);
                }

                result += network.getNeuronWeights(network.neurons);//добавление
порогового веса
                sco += Math.pow(result - output, 2);
                for (int i = 0; i < network.neurons; i++) {
                    double newWeight = network.getNeuronWeights(i) - this.teach * (result -
output) * gausses[i] + this.moment * momentArr[i];//корректировка весов
                    double[] newCenter = new double[network.VectorLength];
```

```
                double[] newSigma = new double[network.VectorLength];
                for (int j = 0; j < network.VectorLength; j++) {
                    newCenter[j] = network.getCenters(i)[j] - this.teach * (result - output)
* gaussesCenters[i][j] + this.moment * momentArr[i];//корректировка центров (не
уверен совсем)
                    newSigma[j] = network.getSigma(i)[j] - this.teach * (result - output) *
gaussesSigmas[i][j] + this.moment * momentArr[i];//
                }
                momentArr[i] = this.teach * (result - output) * gausses[i];
                network.setNeuronWeights(i, newWeight);
                network.setNeuronCenters(i, newCenter);
                network.setSigma(i, newSigma);
            }
            network.setNeuronWeights(network.neurons,
network.getNeuronWeights(network.neurons) - this.teach * (result - output));// +
moment[network.neurons]);//корректировка порогового веса
            momentArr[network.neurons] = this.teach * (result - output);
        }
        sco = Math.sqrt((1 / (double) len) * sco);
        try
        {
        scosw.write(String.valueOf(sco)+ System.lineSeparator());
        } catch (IOException e) {e.printStackTrace();}
        cycles++;
    } while ((cycles < maxCycles) && (sco > Error));
  }

  public double forwardProp() {
    double sco = 0;

    int len = values.size() - network.VectorLength;
    for (int start = 0; start < len; start++)//кол-во пачек
    {

      double[] actual = new double[network.VectorLength];
      for (int j = 0; j < actual.length; j++) {
        actual[j] = values.get(j + start);//инициализация входного вектора
      }
      double output = outputI[start];
      double sum = 0;
      for (int j = 0; j < network.arrayofneurons.length; j++) {
        sum += network.getNeuronWeights(j) * network.calculateGauss(actual, j);
      }


      sum += network.getNeuronWeights(network.neurons);
```

```
            sco += Math.pow((sum - output), 2);
            try {
                sw.write(String.valueOf(sum) + System.lineSeparator());
            } catch (IOException e) {
                e.printStackTrace();
            }
            results.add(sum);
        }
        sco = Math.sqrt((1 / (double) len) * sco);
        try {
            sw.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println("debug");
        return sco;
    }
}
```

```
package com.shwedsoft.rbfnetwork;

import java.util.Random;

public class Network {

    public Neuron[] arrayofneurons;
    public int neurons;
    public Neuron lastneuron;
    public int VectorLength;

    public Neuron[] getArrayofneurons() {
        return arrayofneurons;
    }

    public void setArrayofneurons(Neuron[] arrayofneurons) {
        this.arrayofneurons = arrayofneurons;
    }

    public int getNeurons() {
        return neurons;
    }

    public void setNeurons(int neurons) {
        this.neurons = neurons;
    }
```

```
public Neuron getLastneuron() {
   return lastneuron;
}

public void setLastneuron(Neuron lastneuron) {
   this.lastneuron = lastneuron;
}

public int getVectorLength() {
   return VectorLength;
}

public void setVectorLength(int VectorLength) {
   this.VectorLength = VectorLength;
}

public Network(int amountOfNeurons, int vectorLength) {
   neurons = amountOfNeurons;
   arrayofneurons = new Neuron[neurons];
   for (int i = 0; i < amountOfNeurons; i++) {
      arrayofneurons[i] = new Neuron(vectorLength, false);
   }
   lastneuron = new Neuron(arrayofneurons.length, true);
   this.VectorLength = vectorLength;

}

public void initializeCenters(double min, double max) {
   Random rand = new Random();
   double[] newCenters;
   for (Neuron arrayofneuron : arrayofneurons) {
      newCenters = arrayofneuron.c;
      for (int j = 0; j < newCenters.length; j++) {
         newCenters[j] = rand.nextDouble() * (max - min) + min;
      }
      arrayofneuron.c = newCenters;
   }

}

public void initializeWeights() {
   Random r = new Random();
   for (int i = 0; i < neurons + 1; i++) {
      lastneuron.setWeight(i, r.nextDouble()
            * (Math.sqrt(neurons + 1)) - Math.sqrt(neurons + 1) / 2);
```

```
        }
    }

    public void initializeSigmas(double min, double max) {
        Random rand = new Random();
        double[] newSigmas;
        for (Neuron arrayofneuron : arrayofneurons) {
            newSigmas = arrayofneuron.sigma;
            for (int j = 0; j < newSigmas.length; j++) {
                newSigmas[j] = rand.nextDouble() * (max - min) + min;
            }
            arrayofneuron.sigma = newSigmas;
        }
    }

    public double getNeuronWeights(int index) {
        return lastneuron.getWeight(index);
    }

    public void setNeuronWeights(int index, double value) {
        lastneuron.setWeight(index, value);
    }


    public double calculateGauss(double[] actual, int index) {
        return Math.exp(-0.5 * (this.eucklidNorm(actual, arrayofneurons[index].c,
                arrayofneurons[index].sigma)));
    }

    public double eucklidNorm(double[] actual, double[] ci, double[] sigma) {
        double res = 0;
        for (int i = 0; i < actual.length; i++) {
            res += ((Math.pow(actual[i] - ci[i], 2)) / Math.pow(sigma[i], 2));
        }
        return res;
    }

    public double gradientCenters(double[] actual, int indexI, int indexJ) {
        return lastneuron.getWeight(indexI)
                * ((actual[indexJ] - arrayofneurons[indexI].c[indexJ])
                / Math.pow(arrayofneurons[indexI].sigma[indexJ], 2)) *
(calculateGauss(actual, indexI));
    }

    public double gradientSigmas(double[] actual, int indexI, int indexJ) {
        return lastneuron.getWeight(indexI)
```

```
                * ((actual[indexJ] - arrayofneurons[indexI].c[indexJ])
                / Math.pow(arrayofneurons[indexI].sigma[indexJ], 3)) *
(calculateGauss(actual, indexI));
    }


    public double[] getCenters(int index) {
        return arrayofneurons[index].c;
    }

    public void setNeuronCenters(int index, double[] newCenters) {
        arrayofneurons[index].c = newCenters;
    }

    public void setSigma(int index, double[] sigma) {
        arrayofneurons[index].sigma = sigma;
    }

    public double[] getSigma(int index) {
        return arrayofneurons[index].sigma;
    }

}


package com.shwedsoft.rbfnetwork;

public class Neuron {

    public int VectorLength; // Длина вектора входных данных
    public double[] weights; // Веса входного вектора
    public double[] c;
    public double[] sigma;
    private boolean lastone;
    private double lastState; // Ответ Нейрона Последнее состояние нейрона
    private double lastInput; // Последняя сумма произведений значений входного
вектора на веса
    public double LastInput;
    public double LastPow;

    public double getWeight(int index) {
        return weights[index];
    }

    public void setWeight(int index, double value) {
        weights[index] = value;
```

```
}

public double getSigma(int index) {
    return sigma[index];
}

public void setSigma(int index, double value) {
    sigma[index] = value;
}

public int getVectorLength() {
    return VectorLength;
}

public void setVectorLength(int VectorLength) {
    this.VectorLength = VectorLength;
}

public double[] getC() {
    return c;
}

public void setC(double[] c) {
    this.c = c;
}

public double[] getSigma() {
    return sigma;
}

public void setSigma(double[] sigma) {
    this.sigma = sigma;
}

public double getLastState() {
    return lastState;
}

public void setLastState(double lastState) {
    this.lastState = lastState;
}

public double getLastInput() {
    return lastInput;
}
```

```
    public void setLastInput(double lastInput) {
        this.lastInput = lastInput;
    }

    public double getLastPow() {
        return LastPow;
    }

    public void setLastPow(double LastPow) {
        this.LastPow = LastPow;
    }

    public Neuron(int vectorLength, boolean order) {
        this.lastone = order;
        VectorLength = vectorLength;
        weights = new double[vectorLength + 1];
        c = new double[vectorLength];
        sigma = new double[vectorLength];
        for (int i = 0; i < vectorLength; i++) {
            sigma[i] = 0.0;
        }
    }

}
public class NewMain {

    public static void main(String[] args) {
        int amountOfNeurons = 3;
        int vectorLength = 5;
        double teta = 0.001;
        double SCO = 0.01;
        int maxCycles = 10000;
        double moment = 0.09;
        boolean isLoad = false;
        Network net = new Network(amountOfNeurons, vectorLength);//(количество
нейронов, длина вектора)
        AlgorithmConfig ac = new AlgorithmConfig(teta, maxCycles, moment, SCO,
amountOfNeurons, vectorLength);
        BackPropagation bp = new BackPropagation(net, "sin.txt", ac, isLoad);
        bp.teachingWeights();
        SCO = bp.forwardProp();
        System.out.println(SCO);
    }
}
```