Министерство образования и науки РФ

Министерство экономического развития и торговли РФ

Московский Энергетический Институт
(технический университет)

Факультет Автоматики и вычислительной техники

Кафедра Электротехники и интроскопии

**МАГИСТРСКАЯ ДИССЕРТАЦИЯ**

На тему Применение нейронных сетей для анализа дефектов

Студент группы № А15-00

Мартынов Н. С.
(Ф.И.О.)

Научный руководитель
зав. кафедрой, доцент, Лунин В. П.
(должность, звание, Ф.И.О.)

Консультант

_____
(должность, звание, Ф.И.О.)

Москва
2006

# Оглавление

# Аннотация

В работе производится анализ методики, позволяющей отстроиться от влияния изменения толщины стенки трубы и режима намагничивания при проведении классификации и определении параметров дефектов; методик автоматической настройки нейронных сетей; методик, позволяющих улучшить качество обучения. Продемонстрировано применение передовых подходов к разработке программных систем. Рассмотрено строение программного комплекса, реализующего данные подходы и методики для решения задачи определения класса дефекта и его параметров.

Работа содержит 95 страницы, 7 таблиц, 26 иллюстраций.

# 1 Введение

Россия обладает одним из самых больших в мире потенциалом топливно-энергетических ресурсов. На 13% территории Земли, в стране, где проживает менее 3% населения мира, сосредоточено около 13% всех мировых разведанных запасов нефти и 34% запасов природного газа. Ежегодное производство первичных энергоресурсов в России составляет более 12% от общего мирового производства. При средних ценах на российскую нефть на мировом рынке 25-35 долл. за баррель добыча нефти в России может достигнуть к 2020 г. 550-590 млн. т в год.

Природный газ используется в России как топливо для электрических станций, теплоэлектроцентралей и котельных практически повсеместно, за исключением некоторых районов Дальнего Востока и Крайнего Севера. Трубопроводный транспорт жидких и газообразных углеводородов – основная составляющая энергетической безопасности страны. Любые аварии на газопроводах приводят к перебоям или прекращению подачи газа на электростанции и котельные.

Разрывы на магистральных ветках газопровода наиболее опасны, поскольку в таком случае целым регионам угрожает ограничение подачи газа. Существенный риск возникает и при разрывах на распределительных газопроводах, непосредственно ведущих к электростанции, ТЭЦ или котельной. Наиболее критическими такие аварии могут быть во время отопительного сезона, особенно в пиковые дневные и вечерние часы будних дней.

На территории Российской Федерации эксплуатируется более 350 тыс. км внутрипромысловых трубопроводов, на которых ежегодно происходит множество инцидентов, связанных с повреждением трубопроводов и локальной разгерметизацией. По данным Госгортехнадзора России, в период с 1991 по 1994 годы на объектах трубопроводного транспорта страны произошло 138 крупных аварий. С октября 2001 года по февраль 2002 года на предприятиях ОАО "Газпром" зарегистрировано 5 разрывов газопроводов высокого давления. Из них 4 аварии сопровождались возгоранием транспортируемого природного газа. По статистике аварий на российских магистральных газопроводах, свыше 50% разрывов магистральных газопроводах сопровождаются интенсивными пожарами.

Сходные проблемы возникают и у зарубежных компаний, эксплуатирующих трубопроводные системы. Так, например, в США с июня 1999 по август 2000 годов произошли две крупнейшие аварии на магистральном газопроводе компании "Olympic Pipe

Line Co" и магистральном газопроводе компании "El Paso Natural Gas Co", вызвавшие серьезную обеспокоенность состоянием американского трубопроводного транспорта в широких общественных кругах. Обе аварии, помимо потерь большого количества транспортируемого продукта и затрат на восстановление трубопроводов, сопровождались сильными пожарами, приведшими к гибели 18 человек.

Помимо экологических последствий, аварии на газопроводах наносят ощутимый экономический урон. Так, по данным канадской фирмы "Associated limited", материальный ущерб от слабого факельного выброса природного газа в атмосферу доходит до 175 тыс. долл. США, а от сильного -- до 50 млн долл. США.

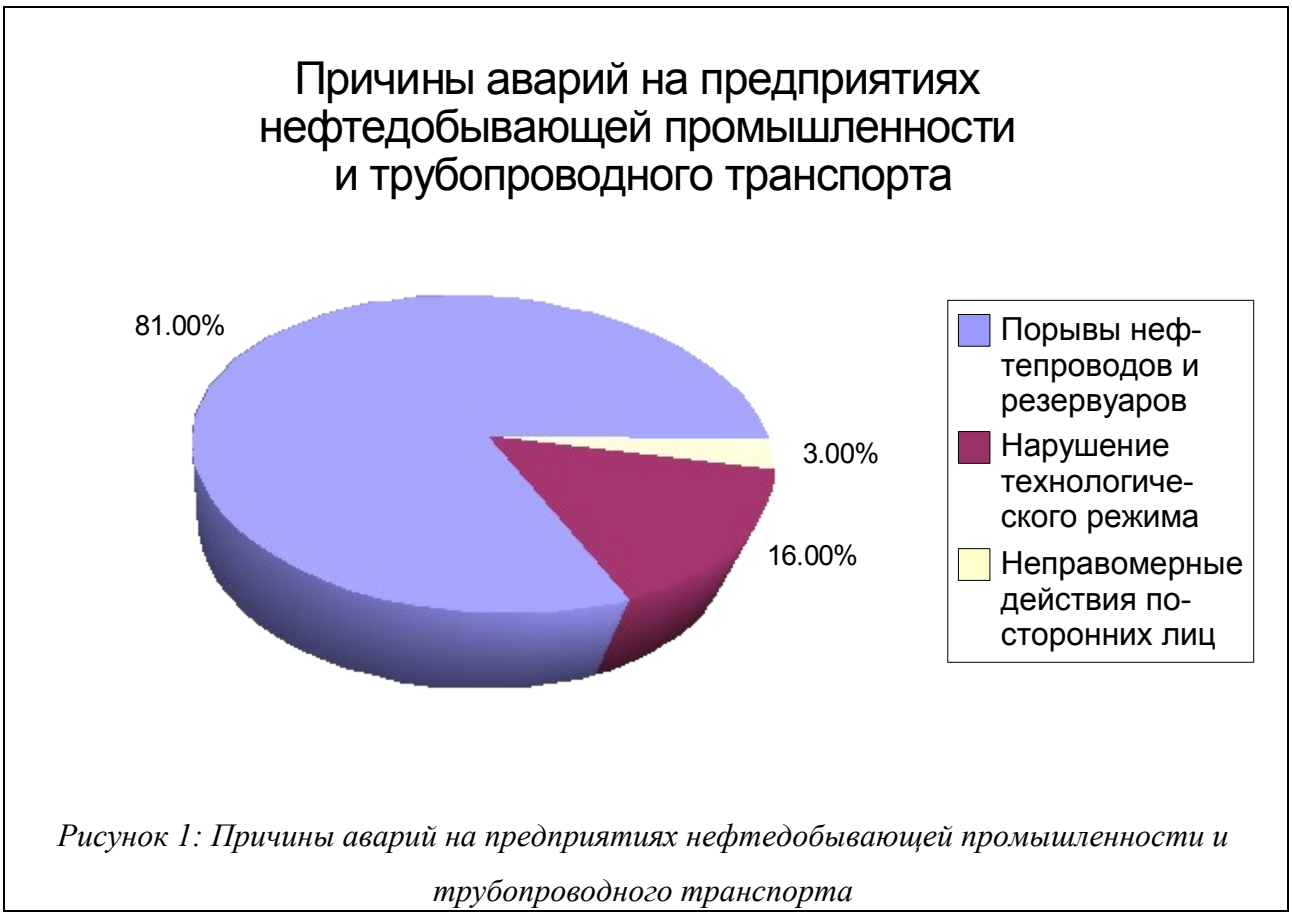


*Рисунок 1: Причины аварий на предприятиях нефтедобывающей промышленности и трубопроводного транспорта*

Основными проблемами в области предупреждения аварийности на магистральных трубопроводах большого диаметра является развитие разрушительных процессов коррозионного растрескивания стенок труб под напряжением, а также брак, допущенный при строительстве. Серьезной проблемой является то, что основной парк газопроводов высокого давления составляют трубопроводы, имеющие срок эксплуатации свыше 20 лет.


*Рисунок 2: Источники дефектов трубопроводов*

Дефекты, вызванные коррозией и строительным браком, с большой степенью точности обнаруживаются диагностическими внутритрубными приборами. Поэтому в основе технической политики компаний, занимающихся транспортировкой нефти и газа, должна быть 100%-ная внутритрубная диагностика магистральных трубопроводов и формирование планов капитального ремонта на основе ее результатов. Такая политика дает значительную экономию средств и увеличение протяженности отремонтированных трубопроводов.

Топливно-энергетический комплекс является важнейшей отраслью промышленности, обеспечивающей энергетическую безопасность и экономическую самостоятельность промышленно развитой страны. В то же время, с точки зрения экологии, топливно-энергетический комплекс выступает как один из главных загрязнителей окружающей среды. Поэтому, вопросы повышения безопасности и эффективности объектов топливно-энергетического комплекса являются ключевыми для экономического развития государства и снижения вредных воздействий на людей и окружающую среду. Все вышеперечисленное свидетельствует об актуальности повышения безопасности транспорта газов и нефти по трубопроводным системам топливно-энергетического комплекса.

Одним из широкораспространенных методов неразрушающего контроля трубопроводов является магнитный контроль, основанный на анализе полей рассеяния. Однако применение данного метода осложняется применением в промышленности труб различного диаметра с различной толщиной стенки. Это обусловлено тем, что классические методики анализа данных полученных от магнитных дефектоскопов опираются на модельные базы сигналов, рассчитанные для труб заданных размеров при применении заданных режимов намагничивания. Таким образом при изменении размера трубы или режима контроля требуется перенастраивать систему анализа сигналов, что может быть связано со значительными затратами времени и денег.

Целью данной работы является разработка методики определения типа дефектов и их размеров для трубопроводов различных диаметров при проведении магнитной дефектоскопии с применением полей различной напряженности без необходимости перенастройки системы анализа.

# 2 Структура системы анализа данных

## *2.1 Источник данных*

Одним из наиболее распространенных способов неразрушающего контроля трубопроводов является магнитный контроль. Внутритрубная диагностика трубопроводов основана на использовании автономных снарядов-дефектоскопов, движущихся внутри контролируемой трубы под напором перекачиваемого продукта (нефть, нефтепродукты, газ и т.п.). Снаряд снабжен магнитной аппаратурой для неразрушающего контроля трубы, записи и хранения в памяти данных контроля и вспомогательной служебной информации, а также источниками питания аппаратуры.

В магнитном снаряде ферромагнитный материал трубы намагничивается постоянными магнитами до состояния близкого к техническому насыщению, а потоки рассеяния, вызванные дефектами, регистрируются магнито чувствительными датчиками (например, датчиками Холла). Измерительная часть снаряда состоит из множества датчиков, расположенных так, чтобы зоны чувствительности датчиков охватывали весь периметр трубы. Это позволяет избежать пропуска дефектов трубы.

*Рисунок 3: Автономный снаряд-дефектоскоп*

Магнитный снаряд-дефектоскоп состоит из трех секций, соединенных между собой шарнирно для прохождения изгибов трубопровода. Постоянные магниты, размещенные на двух кольцах средней секции, создают в трубе продольный магнитный поток между двумя кольцами стальных проволочных щеток, скользящих по внутренней поверхности трубы. Кольцо с подпружиненными держателями блоков датчиков расположено между кольцами щеток, обеспечивая скольжение датчиков по поверхности трубы. Полиуретановые манжеты служат для создания перепада давления перед и позади снаряда, чем обеспечивается его движение в трубе.

Снаряд вводится в контролируемый трубопровод через специальную камеру пуска-приемки, проходит по трубе сотни километров, накапливая информацию о ее состоянии в бортовой памяти, а затем извлекается через аналогичную камеру. После выгрузки снаряда информация считывается на внешний терминал, а затем поступает на сервер базы данных.

Помимо магнитной аппаратуры в измерительной части снаряда-дефектоскопа может применяться акустическая. Датчики ультразвукового снаряда излучают ультразвук в тело трубы и принимают отраженные дефектами сигналы. Ультразвуковые снаряды используют обычно для контроля труб нефтепроводов, поскольку для прохождения ультразвука необходим акустический контакт датчиков с трубой, обеспечиваемый нефтью. Магнитные снаряды применяют для контроля как нефте-, так и газопроводов.

## 2.2 Обработка данных

Данные, выгруженные из снаряда-дефектоскопа в базу данных, после расшифровки представляют собой два двумерных массива значений для двух составляющих вектора магнитной индукции, одним "измерением" в котором, является номер датчика, а вторым - номер скана. Расстояние между датчиками - 8.2 мм, расстояние между сканами 3.3 мм.

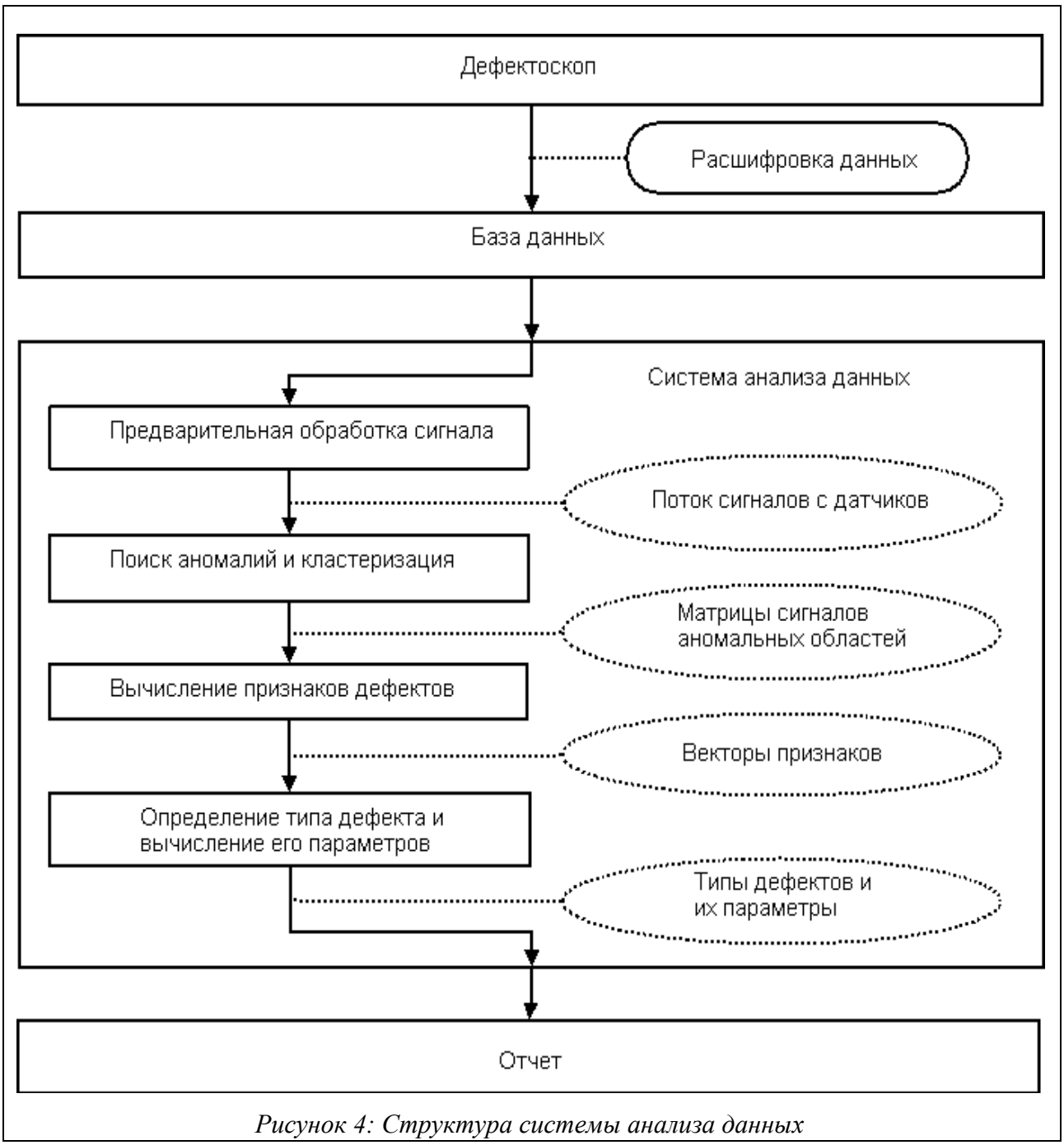


*Рисунок 4: Структура системы анализа данных*

Для обработки полученных данных и определения параметров дефектов могут применяться различные методики, такие как:

- регрессионный анализ;
- генетические алгоритмы;
- нейронные сети.

В данной работе рассматривается применение искусственных нейронных сетей как перспективный и быстроразвивающийся подход.

Для обработки данных с помощью нейронных сетей производится поиск аномалий путем определения областей, где сигнал превышает заданный порог. Данные области затем изолируются и анализируются независимо. Данный процесс называется кластеризацией.

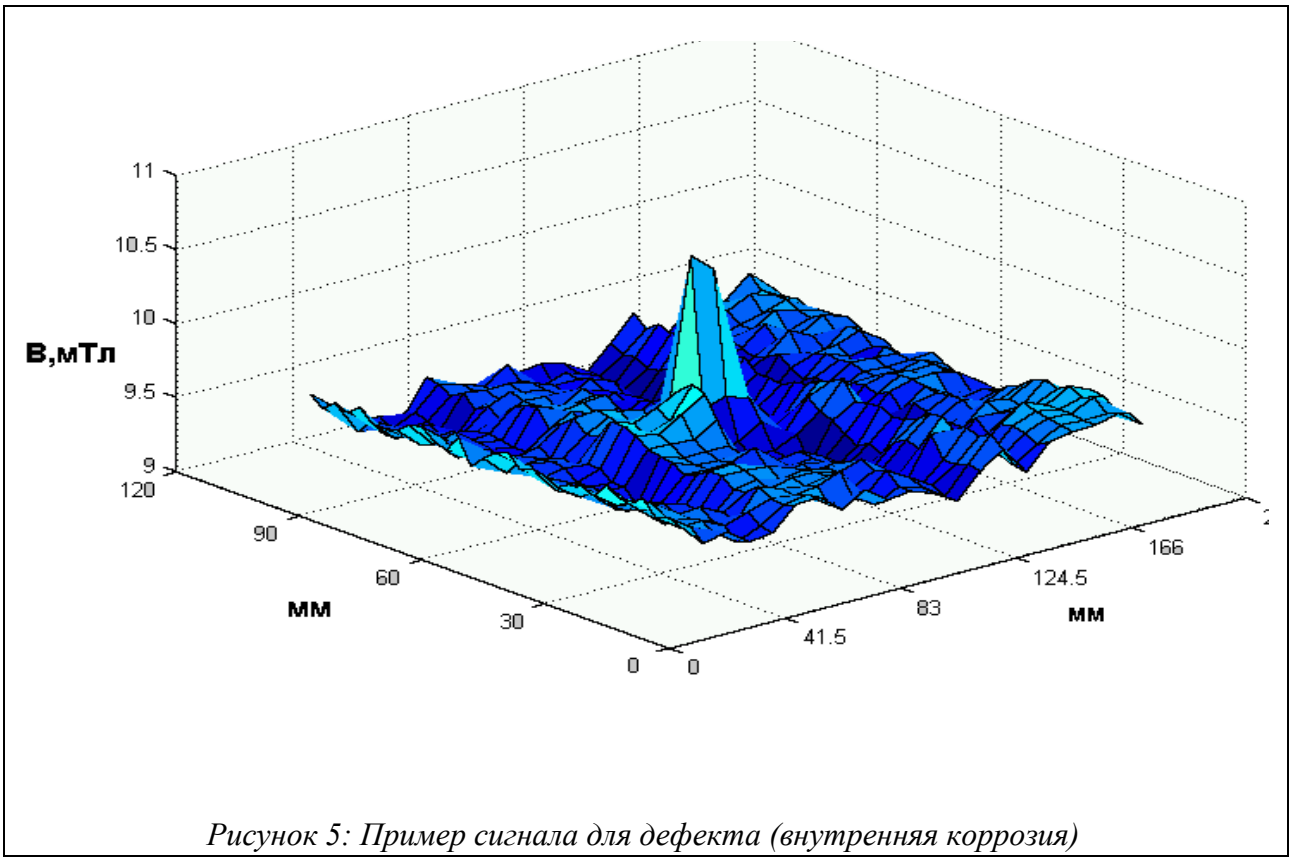


*Рисунок 5: Пример сигнала для дефекта (внутренняя коррозия)*

Полученные матрицы сигналов для аномальных областей подвергаются обработке с целью вычисления так называемых признаков дефекта. Признаки – величины вычисляемые на основе матриц сигналов - однозначно и максимально полно характеризовать дефект. Состав набора признаков выбирается исходя из теоритических предпосылок о корреляции

между определяемыми параметрами дефекта и самими признаками. На основании результатов сторонних исследований в данной работе используются следующие признаки:

1. Разность максимального и минимального значения осевой составляющей поля.
2. Площадь сечения осевой составляющей поля по уровню 0,7 от максимума.
3. Протяженность сигнала в угловом направлении по уровню 0,7 от максимума.
4. Протяженность сигнала в осевом направлении по уровню 0,7 от максимума.
5. Площадь сечения осевой составляющей поля по уровню 0,6 от максимума.
6. Протяженность сигнала в угловом направлении по уровню 0,6 от максимума.
7. Протяженность сигнала в осевом направлении по уровню 0,6 от максимума.
8. Площадь сечения осевой составляющей поля по уровню 0,5 от максимума.
9. Протяженность сигнала в угловом направлении по уровню 0,5 от максимума.
10. Протяженность сигнала в осевом направлении по уровню 0,5 от максимума
11. Разность максимального и минимального значения угловой составляющей поля.

Вычисленные таким образом признаки передаются в систему классификации дефектов и определения их параметров.

Определяемые классы дефектов:

1. Коррозия внешняя.
2. Коррозия внутренняя.
3. Трещина внешняя.
4. Трещина внутренняя.

Определяемые параметры включают:

1. Протяженность дефекта в угловом направлении (мм).
2. Протяженность дефекта в осевом направлении (мм).
3. Глубина дефекта относительно стенки трубы (%).

## *Система классификации и определения параметров*

Важнейшей характеристикой системы анализа должно быть отсутствие необходимости полной перенастройки при изменении толщины стенки трубы или напряженности магнитного поля. Классические методы анализа основаны на применении баз модельных сигналов, рассчитанных для труб определенных размеров и для определенных значений напряженности магнитного поля.

Базы модельных сигналов получают путем расчета индукции магнитного поля для заданного режима намагничивания для модельных дефектов. Модельные дефекты внутри класса отличаются различными геометрическими параметрами. Диапазоны этих параметров выбираются таким образом, чтобы модельные дефекты равномерно и наиболее полно покрывали пространство возможных дефектов, для которых потребуется анализ.

***Таблица 1: Объем базы модельных дефектов***

| ***Класс дефекта*** | ***Число модельных дефектов*** |
|---|---|
| Коррозия внешняя | 1119 |
| Коррозия внутренняя | 1118 |
| Трещина внешняя | 1926 |
| Трещина внутренняя | 1926 |

***Таблица 2: Параметры модельных дефектов***

| ***Класс дефекта*** | ***Параметр*** | ***Значения*** |
|---|---|---|
| Коррозия внешняя | Глубина, мм | 1.6 2.4 3.2 4.8 6.4 7.2 9.6 11.2 12.8 14.4 16 19.2 |
| | Ширина, мм | 8 12 16 24 32 48 |
| | Длина, мм | 8 12 16 24 32 48 |
| Коррозия внутренняя | Глубина, мм | 1.6 2.4 3.2 4.8 6.4 7.2 9.6 11.2 12.8 14.4 16 19.2 |
| | Ширина, мм | 8 12 16 24 32 48 |
| | Длина, мм | 8 12 16 24 32 48 |
| Трещина внешняя | Глубина, мм | 1.6 2.4 3.2 4.8 6.4 7.2 9.6 11.2 12.8 14.4 16 12 13.6 15.2 17.6 19.2 20.8 |
| | Ширина, мм | 0.5 1 1.5 2 3 |
| | Длина, мм | 40 60 80 |
| | Угол | 90 |
| Трещина внутренняя | Глубина, мм | 1.6 2.4 3.2 4.8 6.4 7.2 9.6 11.2 12.8 14.4 16 12 13.6 15.2 17.6 19.2 20.8 |
| | Ширина, мм | 0.5 1 1.5 2 3 |
| | Длина, мм | 40 60 80 |
| | Угол | 90 |

Для модельных дефектов производится рассчет индукции магнитного поля путем применения методов численного моделирования, в частности методом конечных элементов. Моделирование производится в среде Ansys.

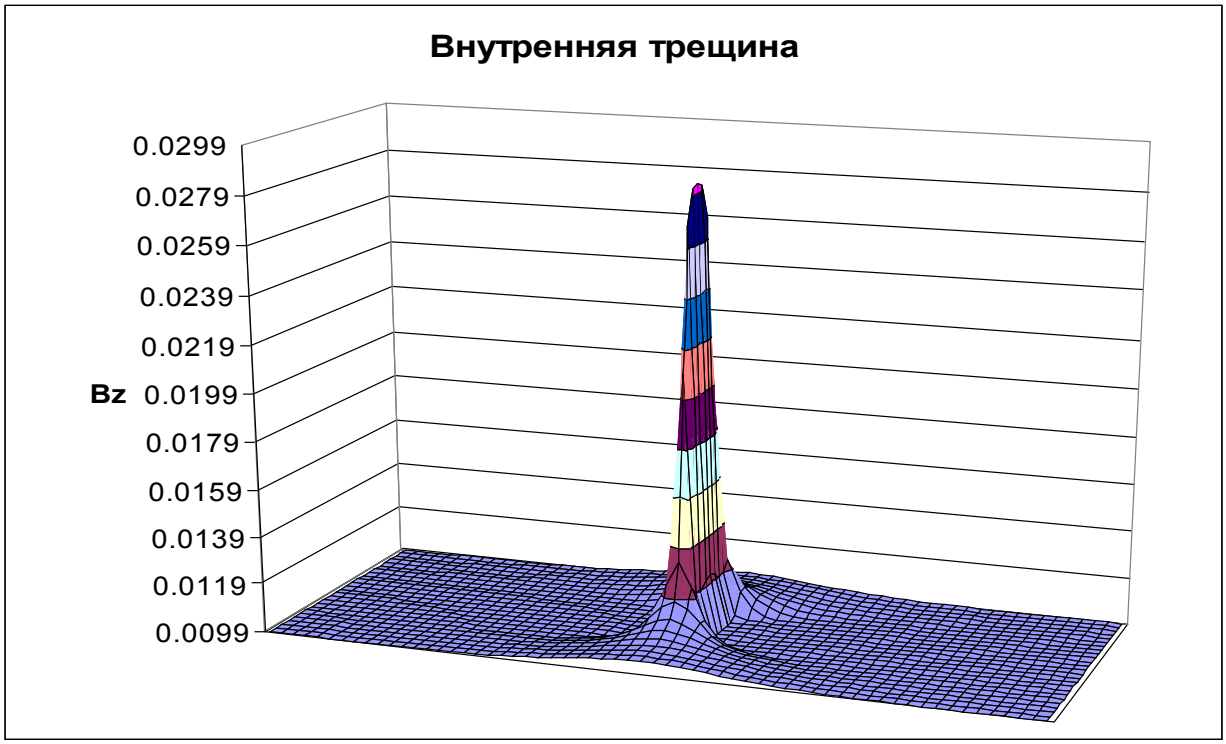


*Рисунок 6: Пример модельного сигнала для дефекта типа внутренняя трещина*

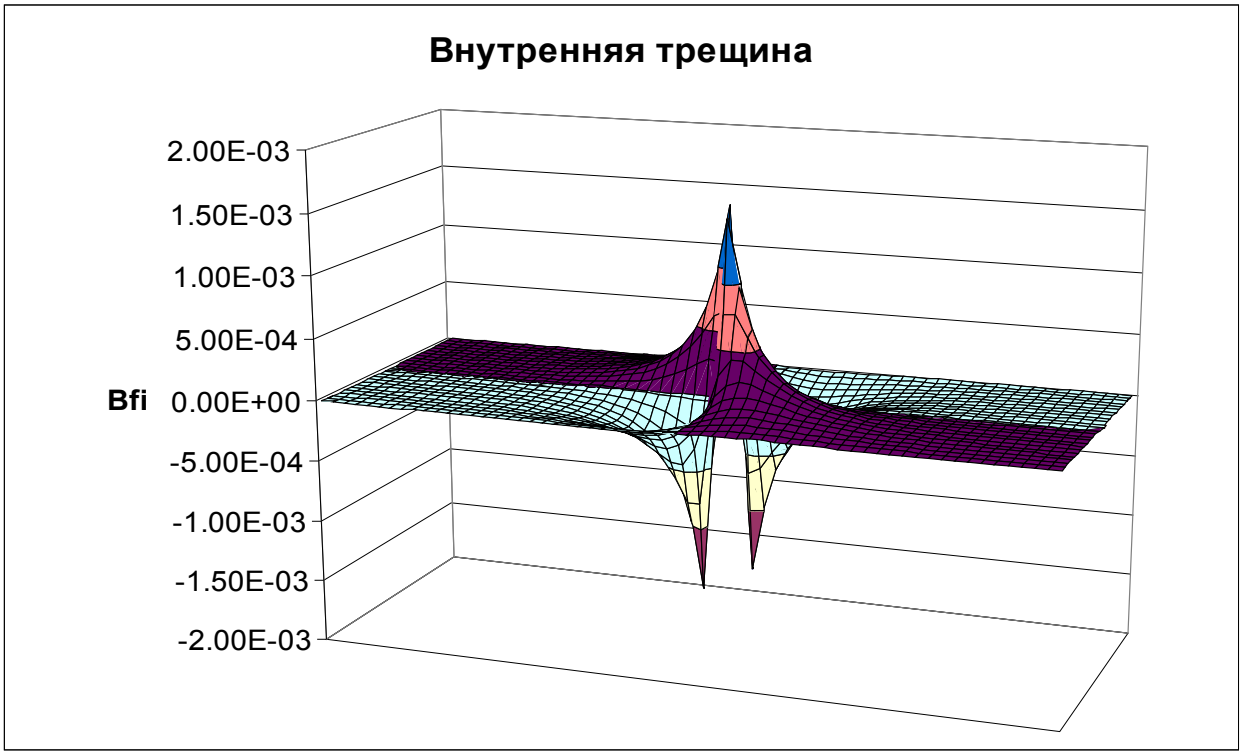


*Рисунок 7: Пример модельного сигнала для дефекта типа внутренняя трещина*

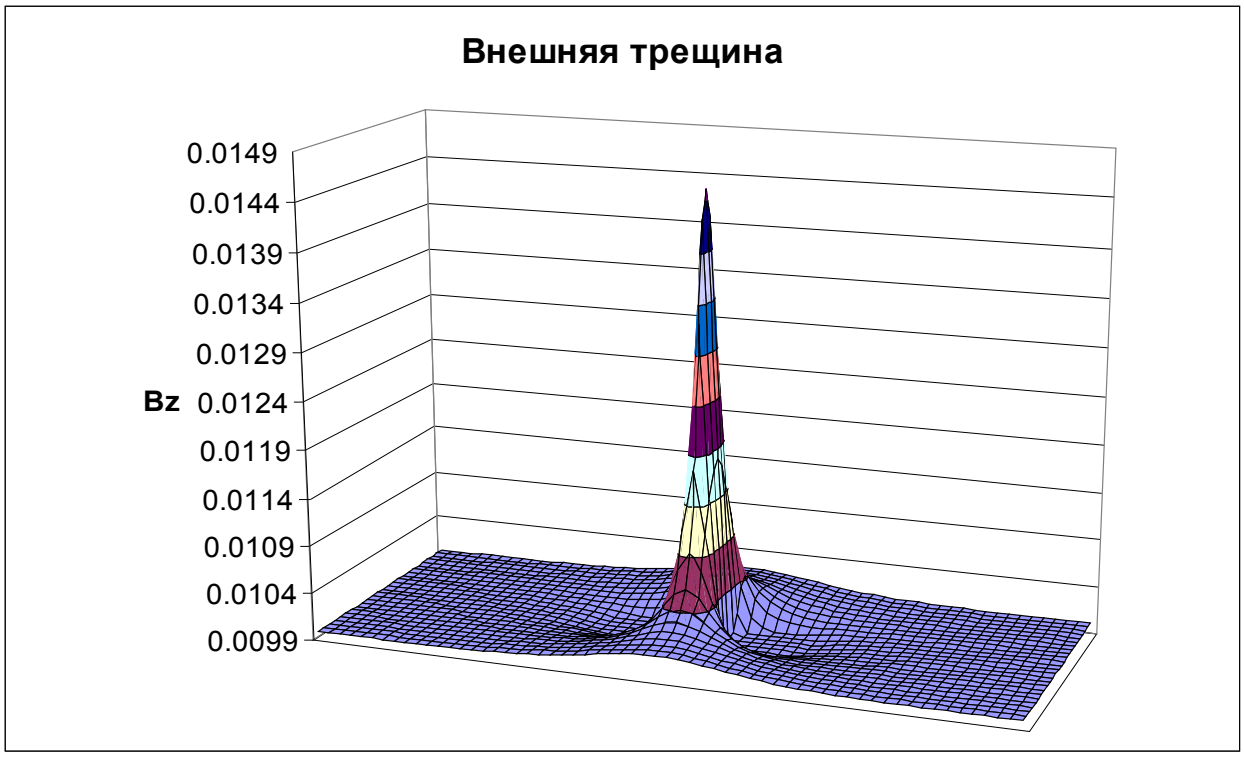


*Рисунок 8: Пример модельного сигнала для дефекта типа внешняя трещина*

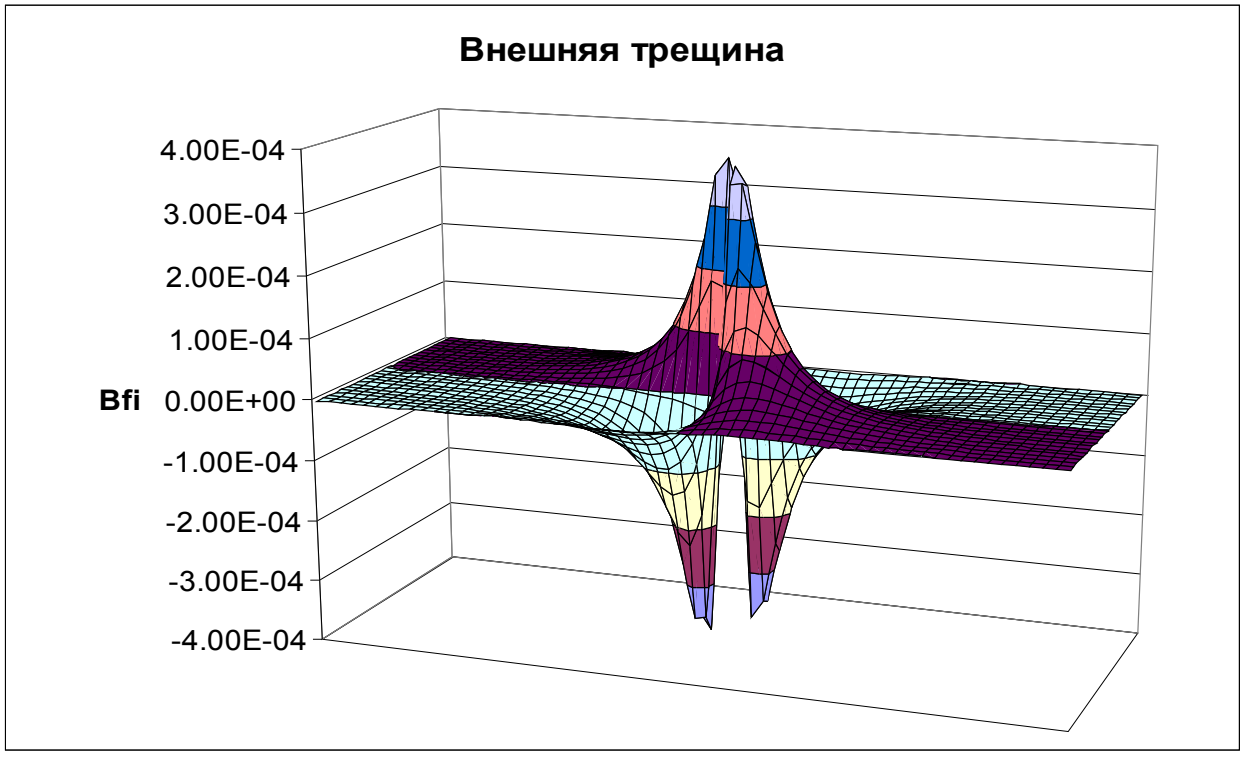


*Рисунок 9: Пример модельного сигнала для дефекта типа внешняя трещина*

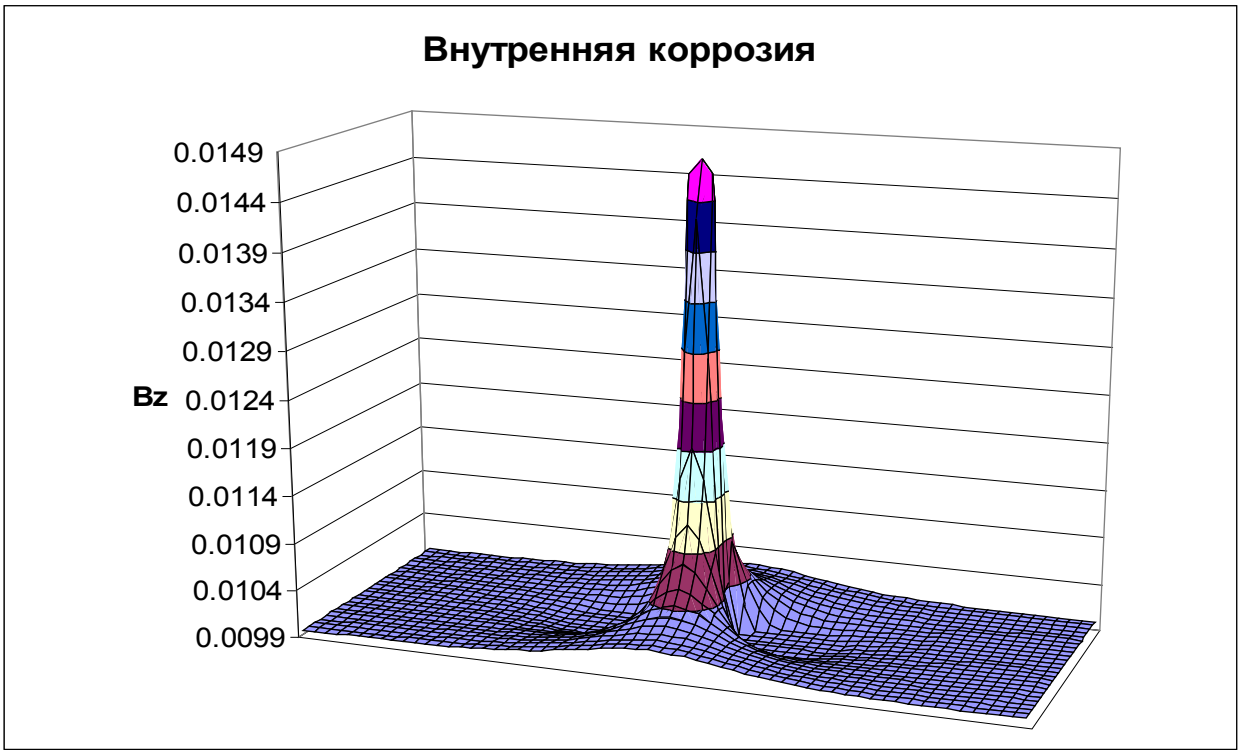


*Рисунок 10: Пример модельного сигнала для дефекта типа внутренняя коррозия*

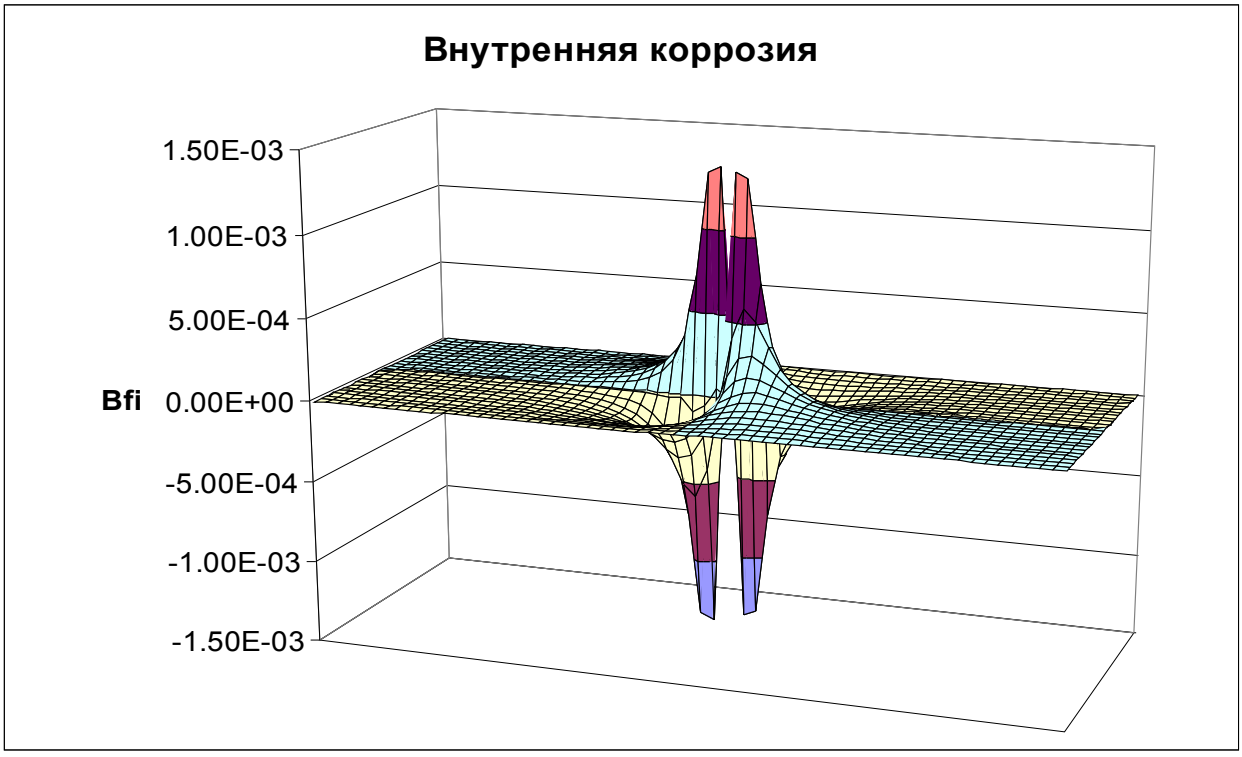


*Рисунок 11: Пример модельного сигнала для дефекта типа внутренняя коррозия*

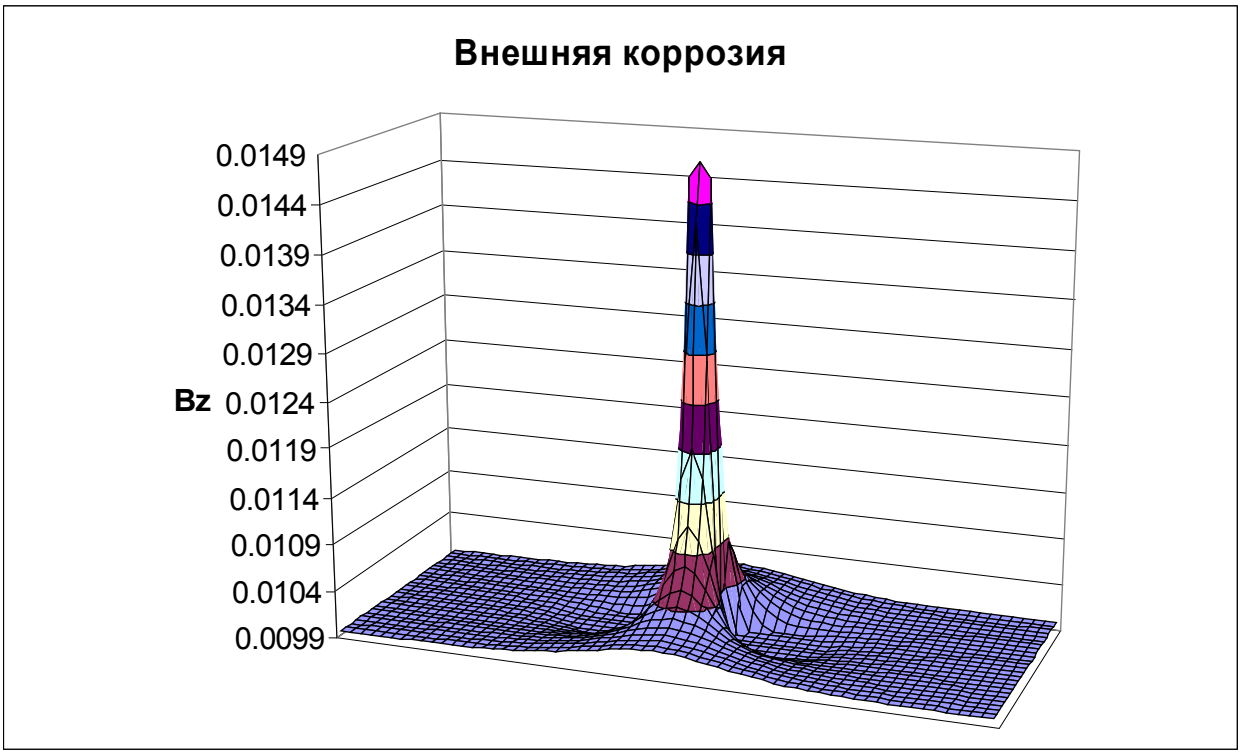


*Рисунок 12: Пример модельного сигнала для дефекта типа внешняя коррозия*

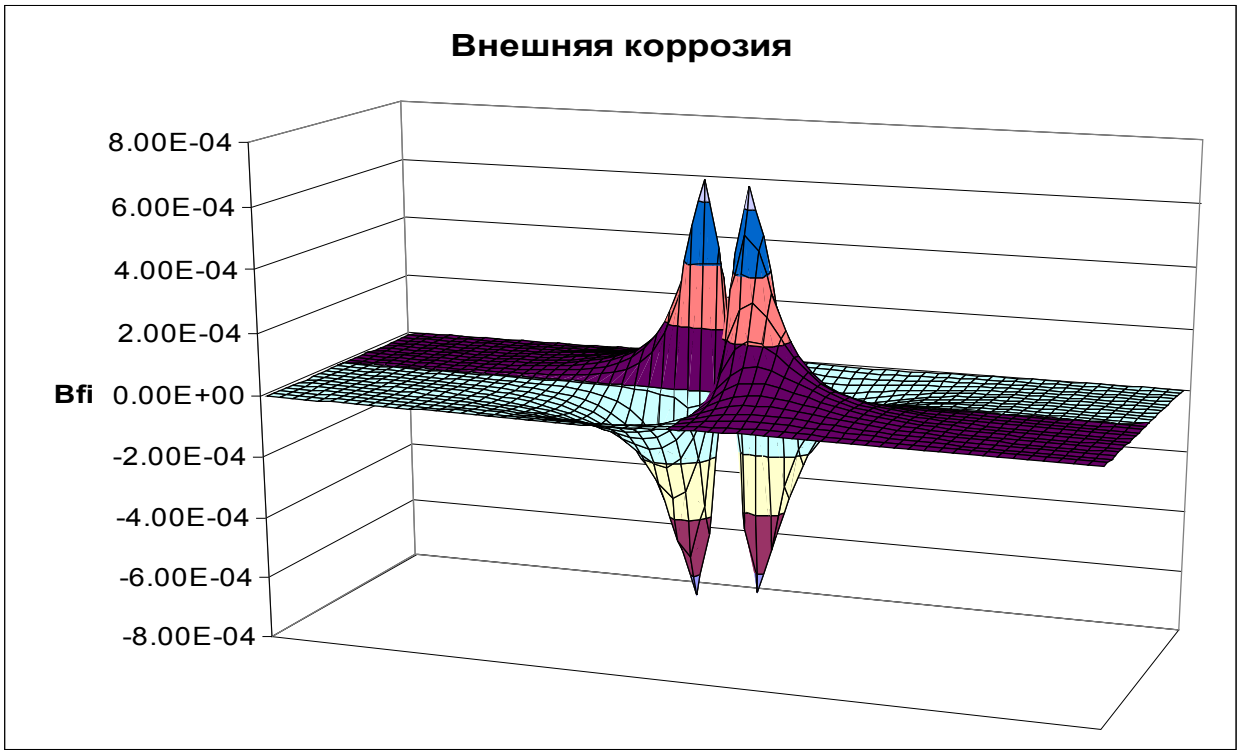


*Рисунок 13: Пример модельного сигнала для дефекта типа внешняя коррозия*

При применении базы дефектов рассчитанной для одного значения величины стенки трубы и режима намагничивания признаки непосредственно подаются на вход системы, вычисляющей их параметры.

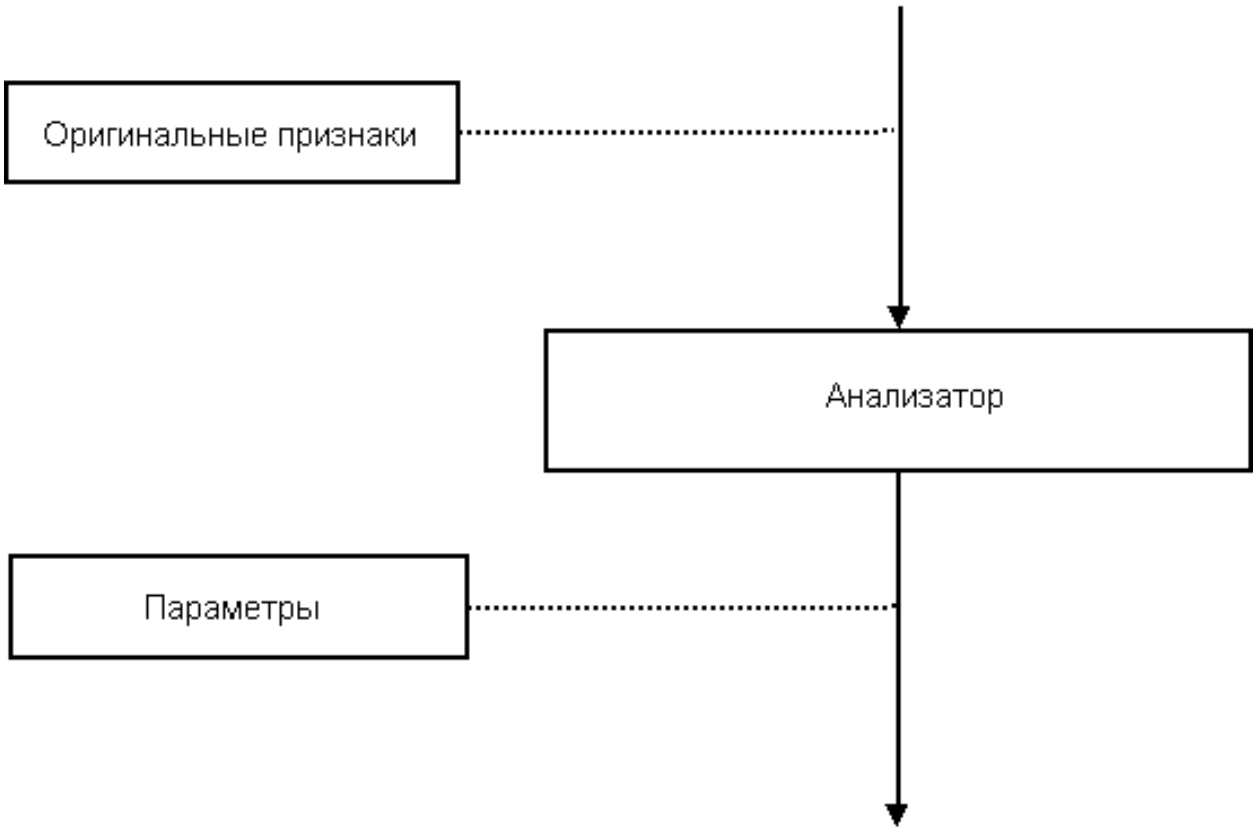


*Рисунок 14: Вычисление параметров в системе, рассчитанной на конкретный размер трубы и режим намагничивания*

При этом переход на другой размер трубы или режим намагничивания требует замены базы дефектов и перенастройки анализатора. Для решения этой проблемы была разработана система, в которой признаки перед тем как поступить на вход анализатора подвергаются дополнительной обработке с целью привести их значения к таким как если бы толщина стенки трубы и напряженность магнитного поля имели номинальные значения. В этом случае не требуется перенастройка анализатора.

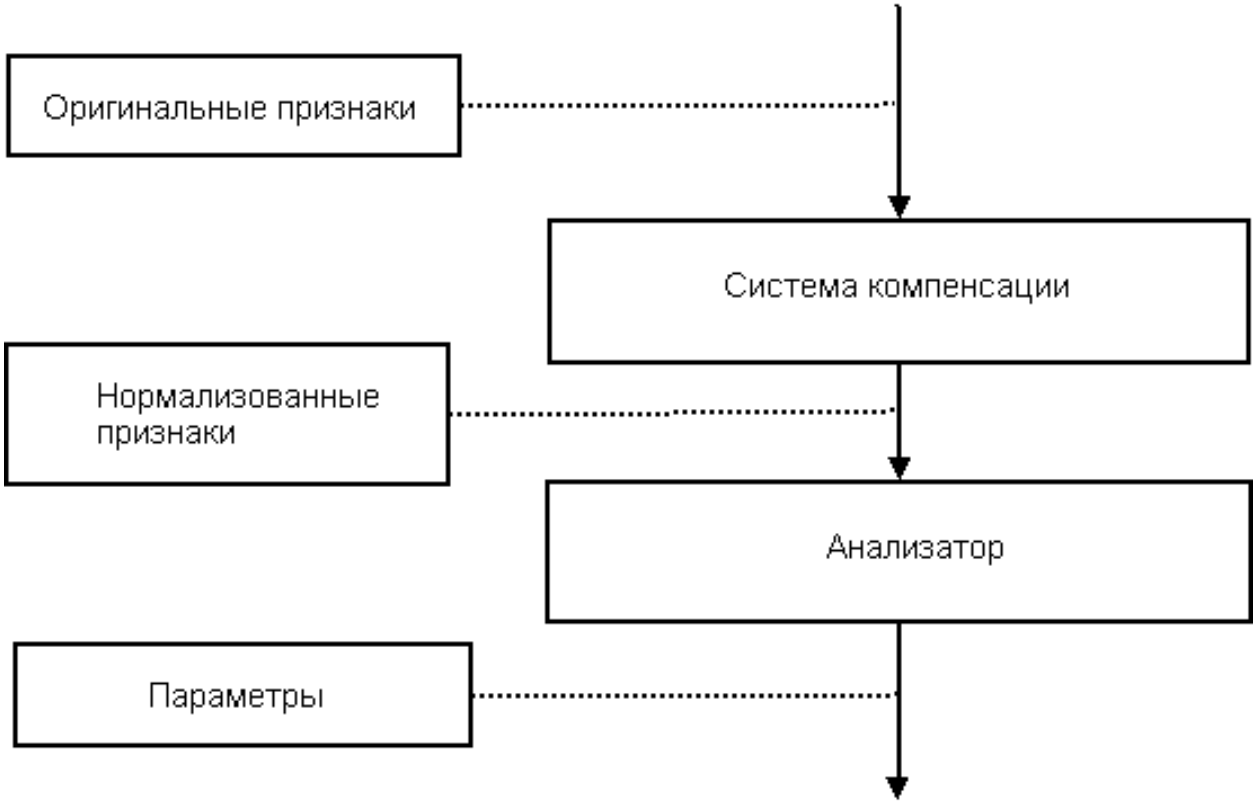


*Рисунок 15: Вычисление параметров в системе с компенсацией изменений размеров толщины стенки трубы и напряженности магнитного поля*

Для разработки системы компенсации, позволяющей исключить необходимость перенастройки системы анализа, были рассчитаны базы модельных дефектов для труб различной толщины при применении полей различной напряженности.

***Таблица 3: Учтенные толщины труб и напряженности поля***

| ***Параметр*** | ***Значения*** |
|---|---|
| Толщина стенки трубы, мм | 8 9.6 12 14.4 16 20.8 |
| Напряженность магнитного поля, А/м | 8000 10000 12000 14000 16000 20000 |

Таким образом полная база модельных дефектов содержит дефекты различных типов и размеров для всех сочетаний размера стенки трубы и напряженности поля. Это позволяет настроить систему компенсации таким образом, чтобы для всех значений

толщины стенки трубы и напряженности поля, входящих в учтенный диапазон, можно было получить вектор признаков, соответствующих номинальному режиму.

На рисунке 16 представлена структурная схема системы классификации и определения параметров.

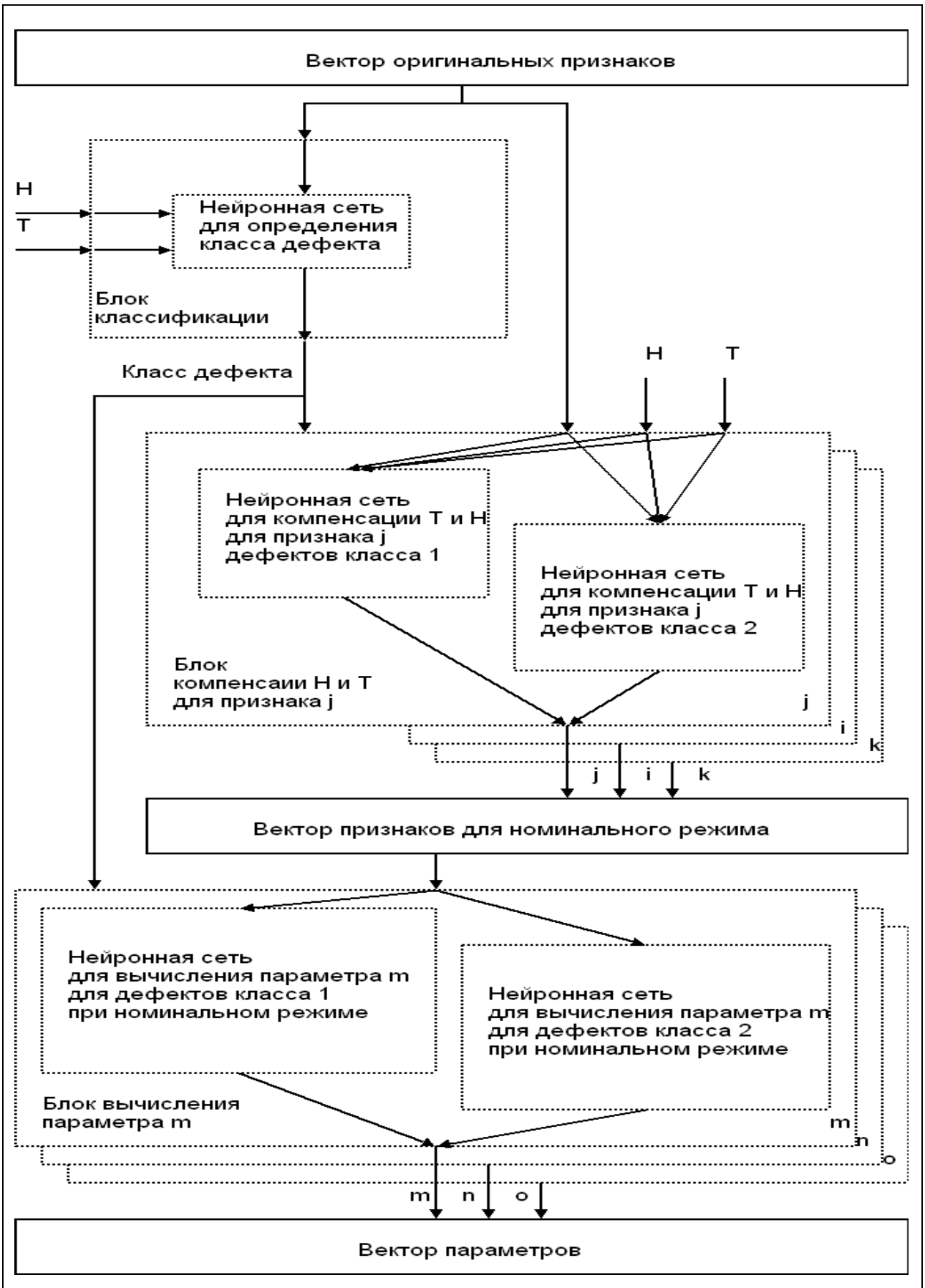


*Рисунок 16: Структурная схема системы классификации и определения параметров*

Рассмотрим работу данной системы:

1. Оригинальный вектор признаков, дополненный значениями толщины стенки трубы и напряженности магнитного поля, подается в блок классификации, состоящий из набора нейронных сетей, каждая из которых определяет вероятность принадлежности дефекта одному из классов. За класс дефекта принимается тот, который соответствует нейронной сети с максимальным значением на выходе.

2. После того, как станет известен класс дефекта, оригинальный вектор признаков, дополненный значениями толщины стенки трубы (Т) и напряженности магнитного поля (Н), подается в блок компенсации, работа которого заключается в расчете для каждого признака значения, соответствующего номинальному режиму. Данный блок состоит из подблоков, вычисляющих значения для одного из признаков. Каждый из подблоков состоит из набора нейронных сетей, осуществляющих компенсацию Т и Н для дефектов одного класса. Таким образом, блок компенсации сдержит 11 подблоков для каждого из признаков. В свою очередь, каждый из этих подблоков включает 4 нейронных сети, соответствующих одному из классов дефектов.

3. Далее вектор признаков, пересчитанный для номинального режима, поступает на вход блока вычисления параметров, который состоит из набора подблоков, осуществляющих вычисление одного из параметров. Каждый из подблоков состоит из набора нейронных сетей, каждая из которых соответствует одному классу дефекта.

Вычисленные подобным образом класс дефекта и его параметры в виде конечного результата сообщаются пользователю.

# 3 Реализация и настройка системы классификации и определения параметров

## *3.1 Архитектура програмного комплекса*

Структурная схема програмного комплекса, реализующего описанную ранее модель (рисунок 16), представлена на рисунке 17.

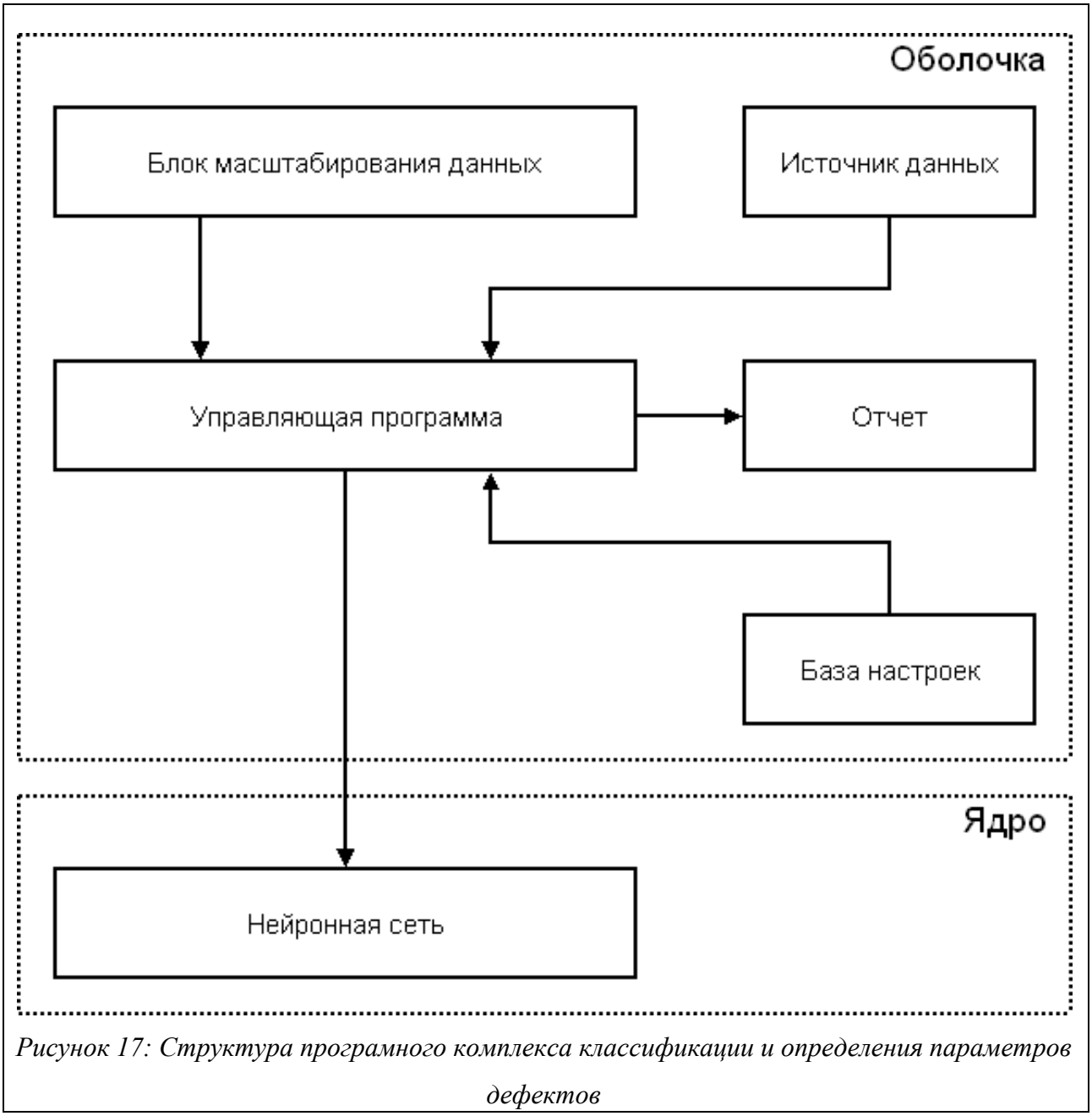


*Рисунок 17: Структура програмного комплекса классификации и определения параметров дефектов*

Программный комплекс состоит из двух основных частей:

1. Оболочки.
2. Ядра.

Оболочка представляет из себя набор блоков, реализованных на интерпретируемом языке программирования Python. В нее входят:

1. Управляющая программа, осуществляющая контроль над остальными блоками програмного комплекса и реализующая логику системы, представленную на рисунке 16.
2. Источник данных – модуль, организующий поступление входных данных в виде файлов заданий, содержащих вектора признаков дефектов, а также значения толщины стенки трубы и напряженности магнитного поля.
3. Блок масштабирования данных – модуль осуществляющий обработку векторов признаков и параметров, поступающих на вход нейронной сети, а также снимаемых с ее выхода.
4. База настроек – модуль, осуществляющий хранение и доступ к настройкам нейронных сетей, обученных вычислять определенный параметр для каждого из классов, а также осуществлять компенсацию изменения значений толщины стенки трубы и напряженности магнитного поля.
5. Отчет – модуль, формирующий файл отчета с результатами классификации и определения параметров дефектов.

Реализация оболочки на интерпретируемом языке программирования Python позволяет производить оперативную настройку поведения системы, а также параметров каждого из ее блоков. Одновременно упрощается процесс добавления новых функциональных возможностей. Применение интерпретируемого языка снижает производительность системы. Однако ее падение оказывается несущественным из-за того факта, что основные вычисления производятся не в оболочке, а в ядре.

Ядро – программа, осуществляющая симуляцию процесса распространения данных в нейронной сети. Она обладает минимальным набором функциональных возможностей, включающим в себя:

1. Создание нейронной сети с заданными параметрами.
2. Загрузка значений весов нейронов сети.
3. Загрузка векторов входных данных; подача их на нейроны входного слоя; осуществление процесса прямого распространения сигнала внутри нейронной сети; сохранение векторов данных, снятых с нейронов выходного слоя.

Ядро реализовано на языке программирования C++ и подключается к оболочке в виде модуля, интерфейс которого сгенерирован с помощью программной системы SWIG (Simplified Wrapper and Interface Generator – Упрощенный генератор оболочек и интерфейсов). SWIG – программный инструмент, позволяющий связывать программы, написанные на C и C++, с программами, написанными на целом наборе высокоуровневых языков, в том числе Python. Реализация ядра на языке программирования C++ позволяет повысить производительность критической части системы, в которой выполняются самые интенсивные вычисления.

### 3.1.1 Нейронная сеть

Искусственные нейронные сети - технология, появившаяся еще в 50-х годах прошлого столетия, однако только разработки 89-94 годов, позволили на практике использовать их потенциал, развитие этой области становится все более интенсивным. В настоящее время разработано много различных типов ИНС (искусственных нейронных сетей). В данной работе использовалась сеть типа "многослойный персептрон" (Multy Layer Perceptron), хорошо зарекомендовавшая себя при решении подобных задач. Она обладает следующими характеристиками:

- статическая (ее структура не изменяется во времени);
- обучается с учителем;
- имеет прямые связи (Feed Forward Network);
- полносвязна (Fully Connected Network), то есть все нейроны следующего слоя имеют связи со всеми нейронами предыдущего;
- обучается по методу обратного распространения ошибки (Error BackPropagation).

Выбор такой сети обусловлен следующими факторами:

- разработан математический аппарат, доказывающий способность подобных сетей аппроксимировать любую функцию N переменных, сопоставив N входным значениям M выходных [3];
- при решении задач классификации такая сеть при обучении "с учителем" вырабатывает внутреннее представление для исходных данных - такое автоматическое выделение признаков является большим преимуществом, например, перед сетями с радиально-базисной функцией активации (RBF, Radial Basis Function Network), которые только могут симулировать схему классификации "ближайший сосед" [6].

Исскуственная нейронная сеть типа многослойный персептрон состоит из слоев нейронов следующих типов:

1. входной;
2. скрытые;
3. выходной.

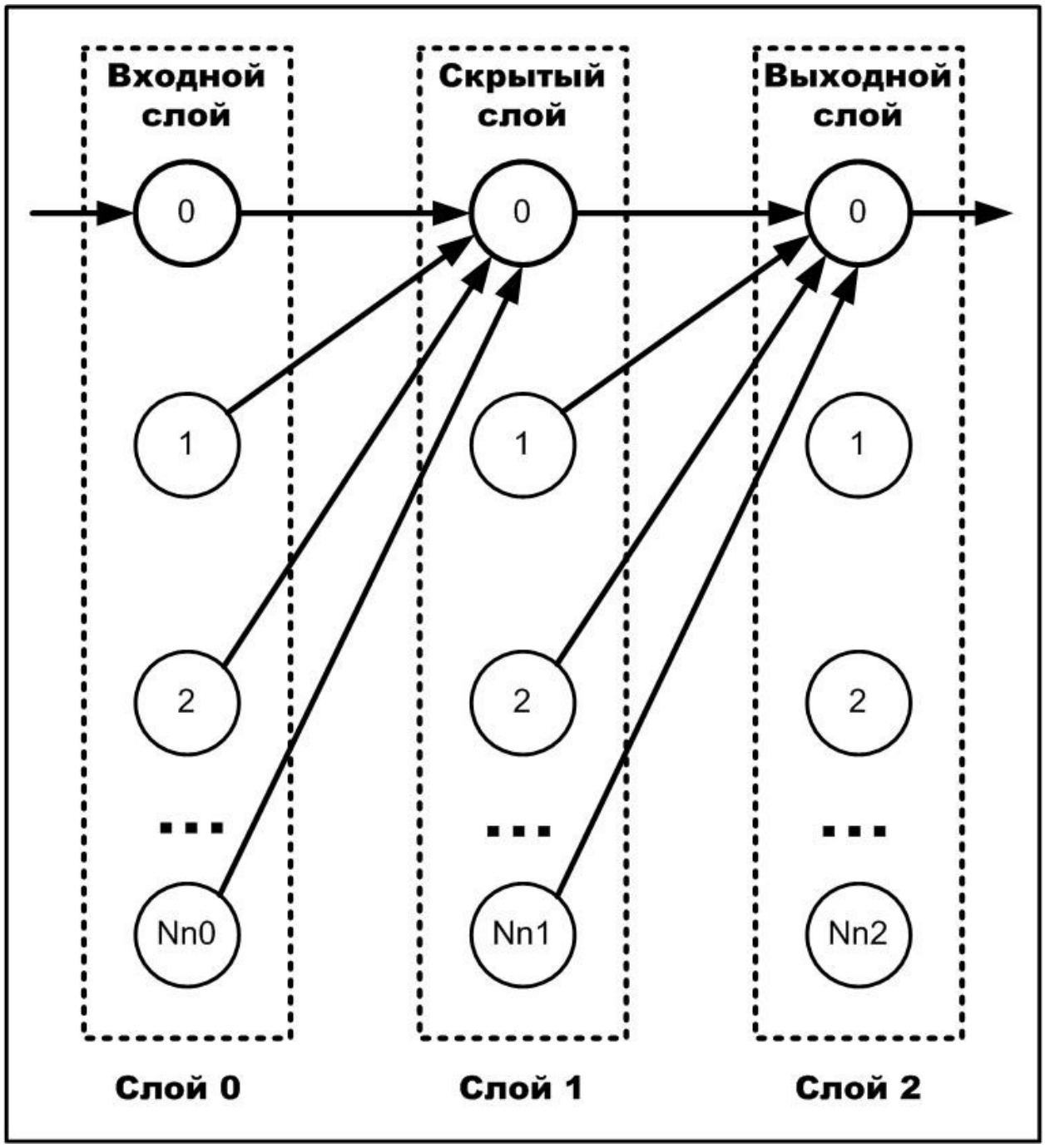


*Рисунок 18: Структура нейронной сети*

На рисунке 18 представлена структура используемой нейронной сети. Здесь m - номер слоя (0..Nm-1), Nm > 2 - число слоев в сети, включая входной, n - номер нейрона в слое (0..Nnm-1), Nnm - число нейронов в слое m.

Входной слой один и он является первым слоем сети. Входные данные в виде признаков подаются именно на этот слой. Нейроны входного слоя передают сигнал к

нейронам первого скрытого слоя без изменений.. Этот слой не обучается, его назначение – распределение входного вектора признаков. Количество нейронов во входном слое равно количеству входных переменных, то есть количеству признаков. Количество скрытых слоев и нейронов в них определяют емкость сети, то есть ее способность к обобщению. Выходной слой один и является последним слоем сети. При обучении на него подаются входные данные в виде закодированных типов дефектов или их параметров, а при использовании с него снимаются выходные данные. Этот слой обучается наравне со скрытыми слоями. При решении задачи классификации нейроны выходного слоя имеют сигмоидальную функцию активации, при определении параметров для скрытого слоя необходимо использовать линейную функцию активации.

Каждый нейрон рассматриваемой сети представляется набором следующих параметров:

1. w - вектор входных весов нейрона, таким образом, $w_{m,n,ni}$ - вес, связывающий вход нейрона n в слое m с выходом нейрона ni в слое m-1
2. T - порог нейрона (threshold), представляющий вес связи с поляризационным нейроном
3. y - выход нейрона, значение которого определяется как

$$y_{m,n} = F( S_{m,n} ) \quad (1)$$

где F - функция активации нейрона, $S_{m,n}$ - взвешенная сумма для нейрона n в слое m и определяется как

$$S_{m,n} = \sum ni ( y_{m-1,ni} * w_{m,n,ni} ) - T_{m,n} \quad (2)$$

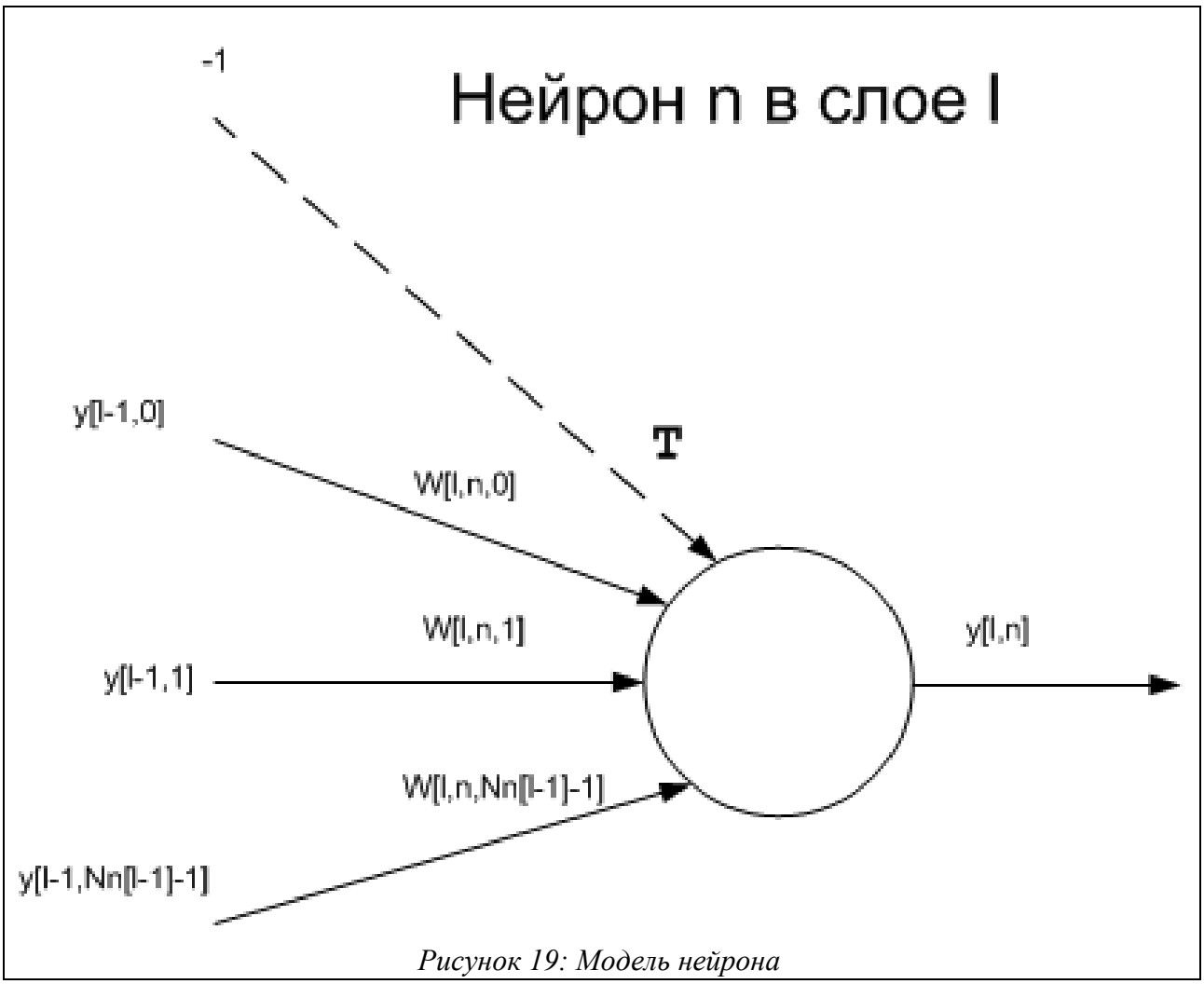


*Рисунок 19: Модель нейрона*

В приведенных далее алгоритмах нейроны также содержат следующие параметры:

- δ - произведение ошибки нейрона на значение производной функции активации
- $\partial E/\partial w$ - вектор, значений частной производной ошибки по весу
- $\partial E/\partial T$ - значение частной производной ошибки по порогу
- $\partial E/\partial w^{t-1}$ - вектор значений частной производной ошибки по весу в предыдущий момент времени
- $\partial E/\partial T^{t-1}$ - значение частной производной ошибки по порогу в предыдущий момент времени
- $\Delta w$ - вектор приращений весов
- $\Delta T$ - приращение порога
- $\Delta\Delta w$ - вектор приращений приращений весов
- $\Delta\Delta T$ - приращение приращений порога

Для оптимизации и ускорения процесса вычислений добавлены следующие параметры:

- S - взвешенная сумма для нейрона
- FdS - значение производной функции активации для нейрона.

С помощью функции активации вычисляется значение на выходе нейрона - степень его возбуждения (см. формулу (1)). Для классификации используется сигмоидальная функция активации, принимающая значения в диапазоне [0; 1]. Значение сигмоидальной функции вычисляется по формуле:

$$y = 1 / ( 1 - e^{-c*x} ) \qquad (3)$$

где c > 0 - коэффициент, характеризующий ширину линейной части по оси абсцисс.


*Рисунок 20: Сигмоидальная функция активации*

Как видно, сигмоидальная функция является монотонной и всюду дифференцируемой. Производную можно вычислить по следующей формуле:

$$y` = c * y * ( 1 - y ) \qquad (4)$$

Для вычисления параметров дефектов также используется линейная функция активации

$$y = k * x \quad (5)$$

ее производная

$$y` = k \quad (6)$$

Распространение сигнала в такой нейронной сети с прямыми связями происходит согласно следующему алгоритму:

1) *подать сигнал на входной слой*
2) *для каждого слоя, начиная с первого скрытого*
   a) *для всех нейронов слоя*
      i) *вычислить взвешенную сумму $S_{m,n}$ по формуле (2)*
      ii) *вычислить выходную активность $y_{m,n}$ по формуле (1)*

Ядро нейро симулятора, реализующее алгоритмы работы с нейронной сетью, построено на основе объектно-ориентированной технологии. Его структура представлена на рисунке 21.

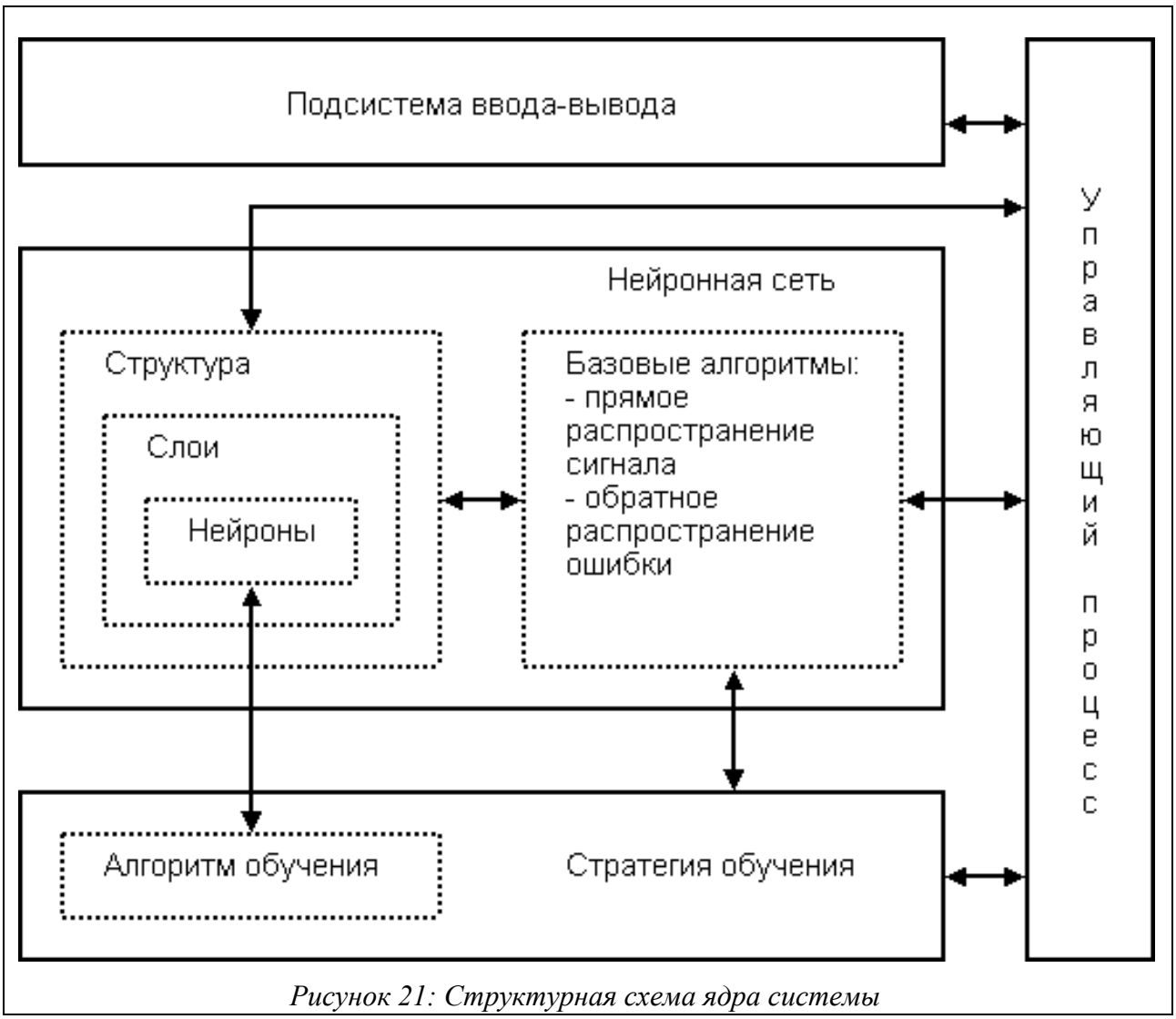


*Рисунок 21: Структурная схема ядра системы*

Применение объектно-ориентированного подхода к декомпозиции системы позволяет создавать изолированные интерфейсы с четкими протоколами взаимодействия между модулями. Это в свою очередь обуславливает возможность замены одних блоков другими при условии, что они имеют тот же интерфейс и четко следуют протоколу. В частности, применение данного подхода позволяет обеспечить возможность замены функций активации нейронов любого из слоев, замену алгоритма или стратегии обучения, замену алгоритма инициализации весов или обеспечить возможность поддержки ввода-вывода в файлы различных форматов.

### 3.1.2 Масштабирование данных

Теоретически на скрытые слои можно подавать непосредственно необработанные значения признаков. Однако на практике это делать нежелательно. Согласно обобщенному дельта правилу (generalized delta rule), изменения весов скрытых слоев прямо пропорциональны выходам нейронов входного слоя. Если выход входного нейрона равен нулю или близок к нулю, то вес, связывающий его с нейроном в следующем слое, не изменяется или меняется слабо. Это явление (low input prime factor) блокирует процесс обучения [6].

Если входные значения сдвинуты (абсолютное значение больше девиации), то веса нейронов первого скрытого слоя могут только вместе убывать или вместе возрастать, так как градиенты будут иметь один знак. Это означает, что значения весов будут изменяться "зигзагами". Это сильно замедлит процесс обучения, так как сдвиги в значениях на входе нейронов обуславливают преимущественные градиенты изменения весов [3].

Таким образом, одним из рациональных является следующий способ нормализации входных значений [3]:

$$x_n` = ( x_n - mean_n ) / ( dev_n ) \quad (7)$$

где mean - среднее арифметическое, а dev - девиация.

В результате нормализации входных значений по (7) на входы нейронов первого скрытого слоя будут подаваться числа в диапазоне от -1 до 1, не имеющие сдвига.

$$mean_n = 1/N_{mp} * \sum^i x_{ni} \quad (8)$$

$$dev_n = max( | mean_n - max\ x_n |, | mean_n - min\ x_n | ) \quad (9)$$

Среднее арифметическое и девиация рассчитываются для каждого признака в отдельности для всего обучающего набора. Данную задачу решает блок масштабирования данных оболочки. Схема его взаимодействия с другими частами системы представлена на рисунке 22.

Весь обмен данными между оболочкой и ядром системы проходит через блок масштабирования. Обработке подвергают векторы данных как для входного, так и для выходного слоев. Вычисление параметров масштабирования происходит однажды при настройке системы классификации и определения параметров. Параметры масштабирования уникальны для каждого набора данных. Таким образом, уникальный набор параметров масштабирования входит в состав настроек для каждой из нейронных сетей, решающих конкретную задачу. Значения в этом наборе целиком определяются характеристиками

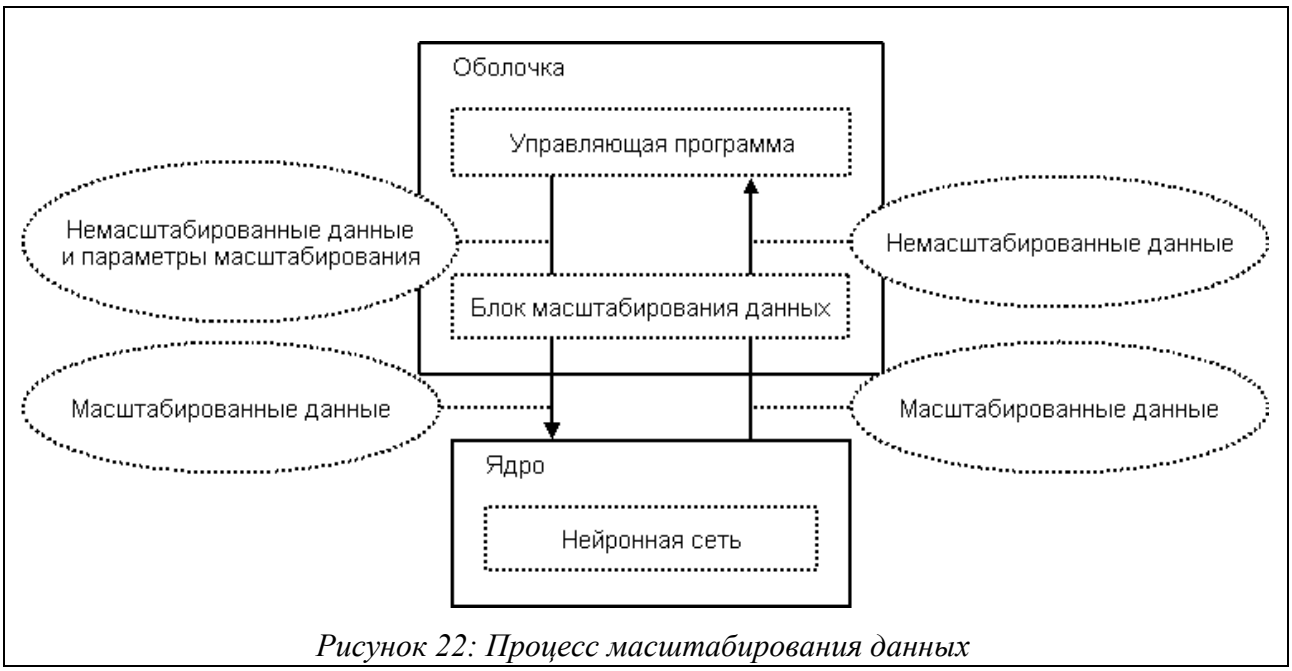


*Рисунок 22: Процесс масштабирования данных*

векторов данных, использованных при обучении нейронной сети и не меняются в последствии.

### 3.1.3 Кодирование данных

Значения на выходе сети определяются функцией активации выходного слоя. Поэтому при решении задачи классификации в процессе обучения следует так кодировать номер класса, чтобы требуемое значение на выходе нейрона находилось в диапазоне значений функции активации. Так как каждому типу дефекта соответствует свой нейрон, то его выход должен являться бинарным значением. Поэтому для кодирования выходных данных наиболее подходит следующая схема:

1. В выходных векторах обучающего набора элемент с номером, равным номеру класса, должен иметь значение, равное максимальному значению, которое может принимать функция активации.
2. В выходных векторах обучающего набора элемент с номером, не равным номеру класса, должен иметь значение, равное минимальному значению, которое может принимать функция активации.

При определении параметров дефектов для выходного слоя используется линейная функция активации, диапазон ее значений (-∞; +∞). Поэтому специальных действий по кодированию выходных данных при решении задачи определения параметров дефектов не требуется.

## *3.2 Настройка системы классификации и вычисления параметров*

### 3.2.1 Структура системы настройки

Программный комплекс классификации и вычисления параметров, структурная схема которого приведена на рисунке 17, реализует сложное логическое поведение, основанное на результатах вычисленных целым каскадом нейронных сетей (см. рисунок 16). Все эти нейронные сети требуют предварительной настройки, которая осуществляется специальным программным комплексом. Его структура приведена на рисунке 23.

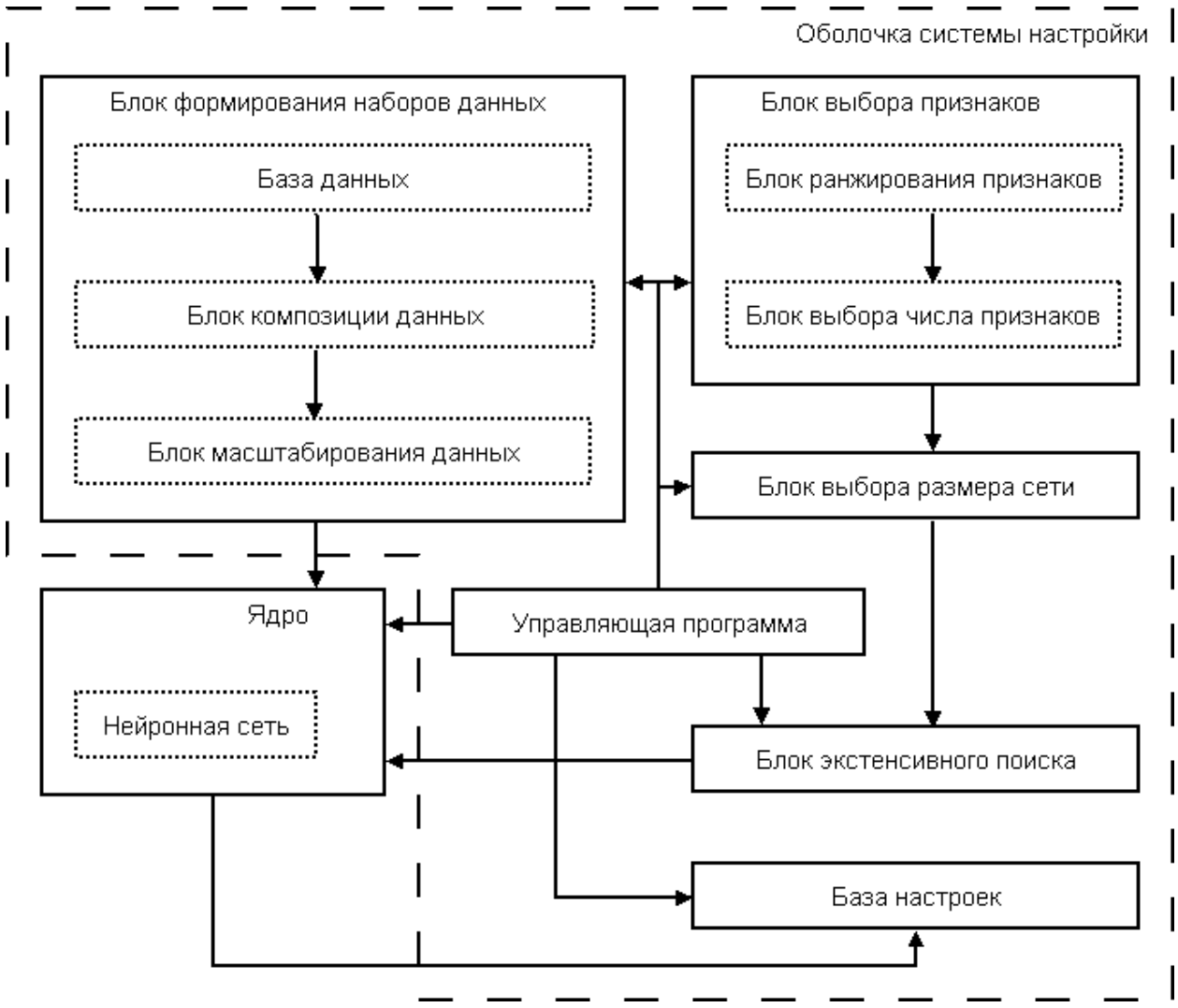


*Рисунок 23: Структура программного комплекса настройки системы классификации и определения параметров*

Данный программный комплекс состоит из двух основных частей:

1. Рассмотренного ранее ядра, осуществляющего непосредственную работу с нейронной сетью и реализованного на языке программирования C++.
2. Оболочки системы настройки, реализованной на языке программирования Python.

Оболочка системы настройки обеспечивает все сервисные функции, а также само логическое поведение. Она включает следущие подсистемы:

1. Управляющую программу, использующую интерфейсы, предоставляемые другими модулями, для реализации логики системы настройки.

2. Блок формирования наборов данных – подсистема, состоящая из пакета модулей, которые обеспечивают формирование наборов обучающих и проверочных данных, которые будут подаваться на нейронную сеть:

а) База данных – модули, обеспечивающие хранение и выборку данных модельных дефектов. Структура данного суб блока представлена на рисунке 24.

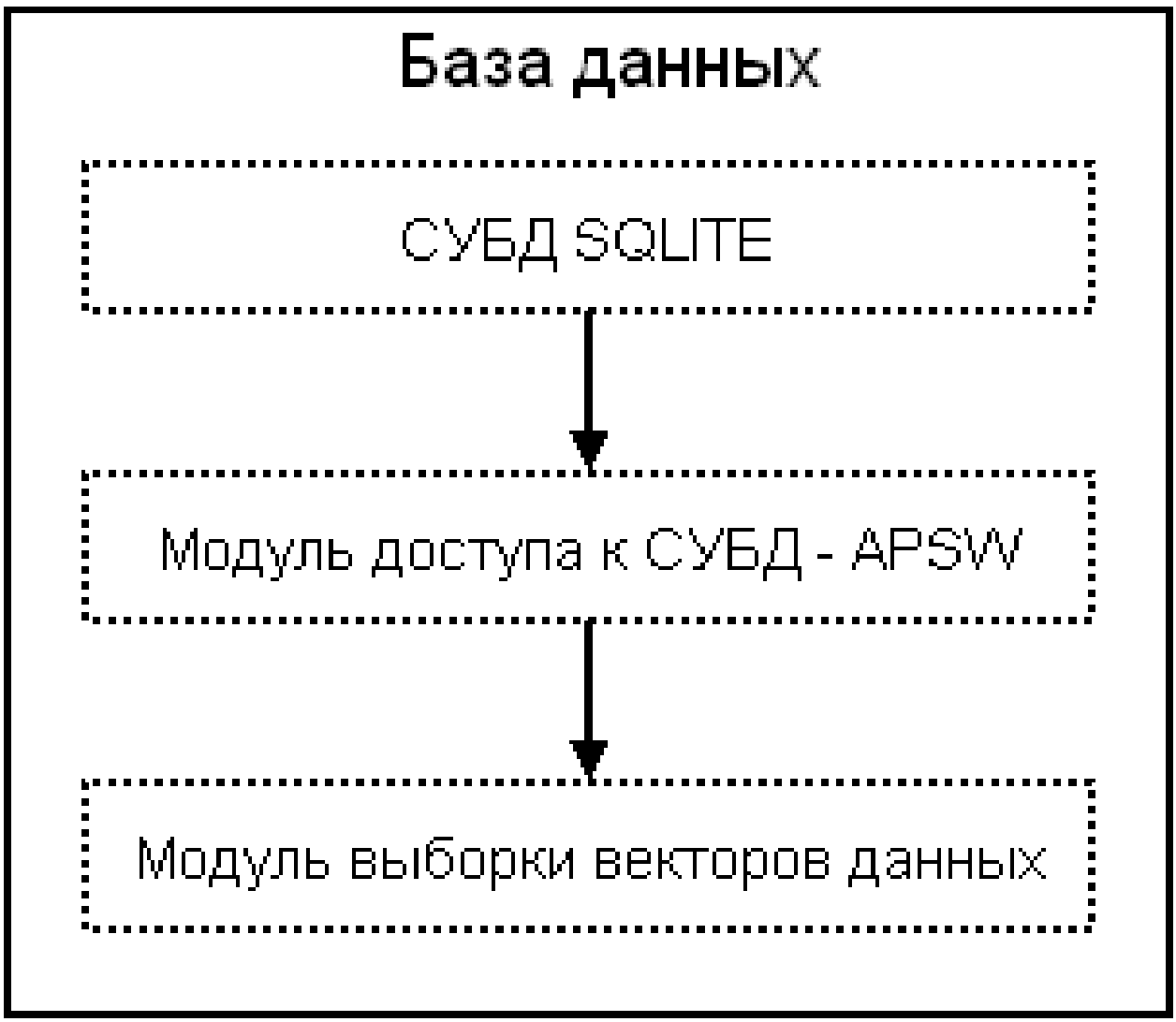


*Рисунок 24: Структура базы данных*

На самом нижнем уровне находится система управления базами данных SQLITE. Эта СУБД хранит все данные в единственном файле, не требует

установки внешних компонентов и дополнительного конфигурирования. Данные хранятся в виде таблиц, доступ к ним осуществляется с помощью стандартного языка работы с базами данных – SQL. Для доступа к функциям СУБД SQLITE используется библиотека APSW (Another Python SQLITE Wrapper), предоставляющая доступ ко внутреннему прикладному программному интерфейсу на языке программирования Python. Модуль выборки векторов данных предоставляет высокоуровневый интерфейс доступа к данным с возможностью фильтрации по заданным критериям.

3. Блок экстенсивного поиска. Так как при обучении нейронной сети используются методы, основанные на минимизации ошибки путем применения метода градиентного спуска, то не гарантируется нахождение глобального минимума. Если целевая функция обладает высокой сложностью, то поверхность функции ошибки в пространстве весов нейронов имеет множество локальных минимумов. Алгоритмы обучения основанные на методе градиентного спуска будут останавливаться в локальном минимуме ближайшем к точке начальных значений весов нейронов. Для увеличения вероятности нахождения лучшего решения, соответствующего более глубокому локальному минимуму могут применяться различные алгоритмы, такие как:

a) Добавление случайного шума в веса нейронов в процессе обучения, в частности использование метода имитации отжига (simulated annealing);

b) Использование генетических алгоритмов, для поиска лучшего набора весов.

Однако они обеспечивают лишь локальный поиск в области начальных значений весов, что не гарантирует нахождение решения лучшего, чем при использовании классических методов обучения.

В данной работе используется другой подход, заключающийся в проведении обширного поиска в пространстве начальных состояний весов. Он описывается следующим алгоритмом:

*1) повторять, пока не исчерпано максимальное число попыток*

*а) инициализировать веса нейронов сети случайным образом*

*б) обучить нейронную сеть*

*в) оценить ошибку*

*2) принять за результат обучения набор параметров, соответствующих минимальной ошибке*

Такой подход обладает следующими особенностями:

- Зона поиска не ограничена;
- Качество решения пропорционально максимальному числу попыток, что позволяет за счет увеличения времени обучения, производимого лишь при настройке системы до ее ввода в эксплуатацию, обеспечить нахождение лучшего набора весов, соответствующего меньшей ошибке.

4. Блок выбора размера сети, осуществляющий поиск числа скрытых слоев и нейронов в них, обеспечивающих наилучшее решение.

Для решения какой-либо задачи с применением искусственной нейронной сети следует, прежде всего, спроектировать структуру сети, адекватную поставленной задаче. Это предполагает выбор количества слоев сети и нейронов в каждом слое, а также определение необходимых связей между слоями.

Подбор количества нейронов во входном слое обусловлен размерностью входного вектора. Подобная ситуация и с выходным слоем, в котором количество нейронов принимается равным размерности ожидаемого вектора. Серьезной проблемой остается подбор количества скрытых слоев и числа нейронов в каждом из них. Теоретическое решение этой задачи в смысле условия достаточности было предложено математиками, занимающимися аппроксимацией функции нескольких переменных. Следует отметить, что искусственная нейронная сеть выступает в роли универсального аппроксиматора обучающих данных. В процессе обучения подбираются его функциональные коэффициенты (векторы весов отдельных нейронов). На этапе функционирования при зафиксированных значениях весов производится простой расчет значения аппроксимирующей функции при заданном входном векторе.

Одно из важнейших свойств нейронной сети - это способность к обобщению полученных знаний. Сеть, натренированная на некотором множестве обучающих выборок, генерирует ожидаемые результаты при подаче на ее вход данных, относящихся к тому же множеству, но не участвовавших непосредственно в процессе обучения. Разделение данных на обучающее и тестовое подмножества представлено на рисунке 25.

Множество данных, на котором считается истинным некоторое правило R, разбито на подмножества L и G, при этом в составе L, в свою очередь, можно выделить определенное подмножество контрольных данных V, используемых для верификации степени обучения сети. Обучение проводится на данных, составляющих

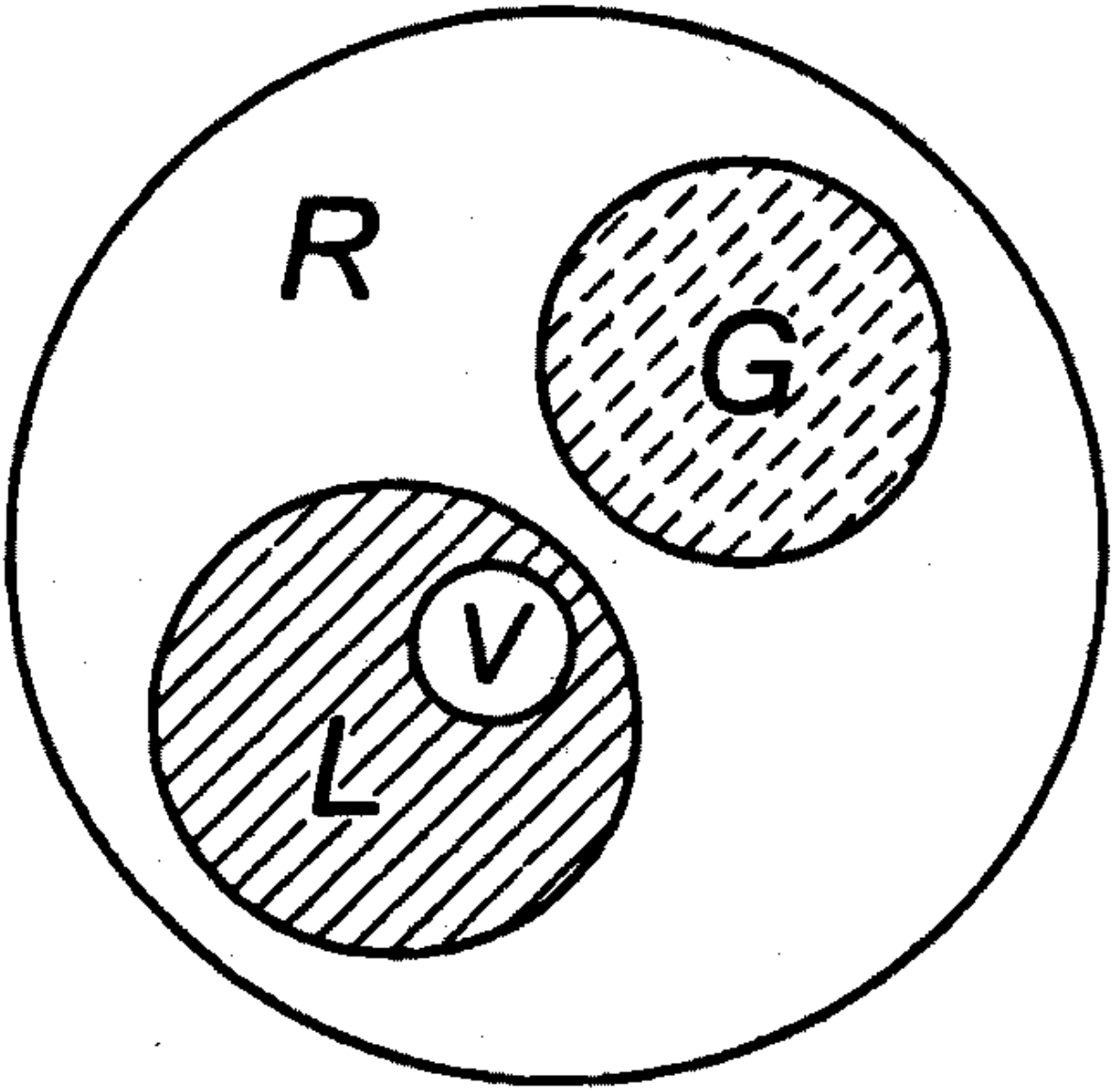


*Рисунок 25: Разделения данных, подчиняющихся правилу R, на обучающее подмножество L, тестовое подмножество G и контрольное подмножество V*

подмножество L. Способность отображения сетью элементов L может считаться показателем степени накопления обучающих данных, тогда как способность распознавания данных, входящих во множество G и не использованных для обучения, характеризует ее возможности обобщения (генерализации) знаний. Данные, входящие и в L, и в G, должны быть типичными элементами множества R. В обучающем подмножестве не должно быть уникальных данных, свойства которых отличаются от ожидаемых (типичных) значений.

Феномен обобщения возникает вследствие большого количества комбинаций входных данных, которые могут кодироваться в сети с N входами. Если в качестве простого примера рассмотреть однослойную сеть с одним выходным нейроном, то для нее может быть составлено $2^N$ входных выборок. Каждой выборке может соответствовать единичное или нулевое состояние выходного нейрона. Таким образом, общее количество различаемых сигналов составит $2^N$. Если для обучения сети используются p из общего числа $2^N$ входных выборок, то оставшиеся незадействованными ($2^N$ - p) допустимых комбинаций характеризуют потенциально возможный уровень обобщения знаний.

Подбор весов сети в процессе обучения имеет целью найти такую комбинацию их значений, которая наилучшим образом воспроизводила бы последовательность ожидаемых обучающих пар. При этом наблюдается тесная связь между количеством весов сети (числом степеней свободы) и количеством обучающих выборок. Если бы целью обучения было только запоминание обучающих выборок, их количество могло быть равным числу весов. В таком случае каждый вес соответствовал бы единственной обучающей паре. К сожалению, такая сеть не будет обладать свойством обобщения и сможет только восстанавливать данные. Для обретения способности обобщать информацию сеть должна тренироваться на избыточном множестве данных, поскольку тогда веса будут адаптироваться не к уникальным выборкам, а к их статистически усредненным совокупностям. Следовательно, для усиления способности к обобщению необходимо не только оптимизировать структуру сети в направлении ее минимизации, но и оперировать достаточно большим объемом обучающих данных.

Со статистической точки зрения погрешность обобщения зависит от уровня погрешности обучения $E_L$ и от доверительного интервала ε. Она характеризуется отношением

$$\mathbf{E_G => E_L + \varepsilon( p/h, E_L)} \qquad \mathbf{(10)}$$

Значение ε функционально зависит от уровня погрешности обучения $E_L$ и от отношения количества обучающих выборок p к фактическому значению h параметра, называемого мерой Вапника-Червоненкиса и обозначаемого $VC_{dim}$. Мера $VC_{dim}$ отражает уровень сложности нейронной сети и тесно связана с количеством содержащихся в ней весов. Значение ε уменьшается по мере возрастания отношения количества обучающих выборок к уровню сложности сети.

По этой причине обязательным условием выработки хороших способностей к обобщению считается грамотное определение меры Вапника-Червоненкиса для сети заданной структуры. Метод точного определения этой меры не известен, о нем можно лишь сказать, что ее значение функционально зависит от количества синоптических весов, связывающих нейроны между собой. Чем больше количество различных весов, тем больше сложность сети и соответственно значение меры $VC_{dim}$. Часто верхнюю и нижнюю границы этой меры определяют в виде

$$2[K/2]N=<VC_{dim}=<2N_w(1+\lg N_n) \qquad (11)$$

где [ ] обозначена целая часть числа, N - размерность входного вектора, К - количество нейронов скрытого слоя, $N_w$ - общее количество весов сети, а $N_n$ - общее количество нейронов сети.

Из выражения (10) следует, что нижняя граница диапазона приблизительно равна количеству весов, связывающих входной и скрытый слои, тогда как верхняя граница превышает двукратное суммарное количество всех весов сети. В связи с невозможностью точного определения меры $VC_{dim}$ в качестве ее приближенного значения используется общее количество весов нейронной сети.

Таким образом, на погрешность обобщения оказывает влияние отношение количества обучающих выборок к количеству весов сети. Небольшой объем обучающего подмножества при фиксированном количестве весов вызывает хорошую адаптацию сети к его элементам, однако не усиливает способности к обобщению, так как в процессе обучения наблюдается относительное превышение числа подбираемых параметров (весов) над количеством пар фактических и ожидаемых выходных сигналов сети. Эти параметры адаптируются с чрезмерной (а вследствие превышения числа параметров над объемом обучающего множества - и неконтролируемой) точностью к значениям конкретных выборок, а не к диапазонам, которые эти выборки должны представлять. Фактически задача аппроксимации подменяется в этом случае задачей приближенной интерполяции. В результате всякого рода нерегулярности обучающих данных и измерительные шумы могут восприниматься как существенные свойства процесса. Функция, воспроизводимая в точках обучения, будет хорошо восстанавливаться только при соответствующих этим точкам значениях. Даже минимальное отклонение от этих точек вызовет значительное увеличение погрешности, что будет восприниматься как ошибочное обобщение. По результатам разнообразных численных экспериментов установлено, что высокие показатели

обобщения достигаются в случае, когда количество обучающих выборок в несколько раз превышает меру $VC_{dim}$ [4].

На практике подбор количества скрытых нейронов (и связанный с ним подбор количества весов) может, в частности, выполняться путем тренинга нескольких сетей с последующим выбором той из них, которая содержит наименьшее количество скрытых нейронов при допустимой погрешности обучения.

Другим возможным решением является обучение нескольких сетей с выбором той из них, которая обеспечивает наименьшую ошибку обобщения. При этом, результирующая сеть может иметь большее число слоев и нейронов в них, чем если бы использовался предыдущий критерий. Однако использование в качестве целевого критерия именно ошибки обобщения позволяет получить нейронную сеть с наилучшей способностью к аппроксимации, что требуется при переходе от модельных дефектов к реальным. Именно такой подход используется в блоке поиска размера сети:

*1) разделить имеющееся множество векторов данных на два подмножества: обучающее и проверочное*

*2) для каждого элемента из заданного множества сочетаний числа скрытых слоев и числа нейронов в каждом из них выполнить:*

*а) создать нейронную сеть с тестируемой конфигурацией*

*б) провести обучение с экстенсивным поиском на обучающем подмножестве*

*в) оценить ошибку обобщения на проверочном подмножестве*

*3) принять за оптимальный размер конфигурацию сети, для которой была получена минимальная ошибка обобщения*

5. Блок выбора признаков, осуществляющий поиск оптимального набора признаков, подаваемых на вход нейронной сети, при решении каждой из задач. Необходимость применения данного блока вызвана тем фактом, что увеличение числа признаков приводит к увеличению числа нейронов во входном слое. Это влечет за собой увеличение числа связей в нейронной сети, что может обусловить переобучение при недостаточном количестве векторов в обучающей выборке, а также увеличение порядка и сложности целевой функции, которую должна будет аппроксимировать нейронная сеть. Последнее может служить источником увеличения погрешности. Кроме того, наличие шума в признаках также является причиной увеличения погрешности, поэтому нейронная сеть, вычисляющая параметры, по

большему количеству зашумленных признаков может показывать большую погрешность.

Ввиду вышесказанного, очевидно, что следует исключить признаки, не вносящие существенного вклада в увеличение точности аппроксимации, а также признаки, вариация которых вносит существенную ошибку. Для решения этой задачи предназначен блок выбора признаков, состоящий из двух суб блоков:

1. Блока ранжирования признаков, который определяет влияние каждого из них на точность нахождения решения.
2. Блока выбора числа признаков, находящего оптимальный их набор, основываясь на вкладе каждого из признаков в решение.

Блок выбора признаков реализует следующий алгоритм:

*1) для каждого из элементов множества признаков выполнить:*

*а) произвести обучение нейронной сети с автоматическим поиском оптимального размера и экстенсивным поиском в пространстве начальных значений весов при подаче на ее вход вектора состоящего из единственного тестируемого признака*

*б) оценить ошибку обобщения на проверочном множестве*

*2) упорядочить признаки в порядке возрастания полученной ошибки обобщения*

*3) для всех целых n от 1 до числа признаков выполнить:*

*а) произвести обучение нейронной сети с автоматическим поиском оптимального размера и экстенсивным поиском в пространстве начальных значений весов при подаче на ее вход вектора состоящего из первых n признаков множества, полученного на шаге 2)*

*б) оценить ошибку обобщения на проверочном множестве*

*4) принять за оптимальный набор признаков множество, для которого была получена минимальная ошибка обобщения.*

### 3.2.2 Обучение нейронной сети

Обучение нейронной сети начинается с инициализации весов ее нейронов. Выбор начальных значений весов и порогов влияет на скорость обучения сети. Если начальные значения весов большие, то это вводит нейронные элементы в насыщение, что обуславливает малые значения градиентов изменения весов и, как результат, малую скорость обучения. Если начальные значения весов слишком маленькие, это ведет к тому, что производная функции активации нейрона близка к нулю, что также замедляет скорость обучения [8].

Начальные значения весов следует выбирать так, чтобы ожидаемая девиация взвешенной суммы лежала между линейной частью и насыщением функции активации [12]. Для этого начальные значения весов следует выбирать случайным образом из диапазона $[-w\_im_m \ldots w\_im_m]$, где $w\_im_m$ вычисляется для каждого слоя

$$\mathbf{w\_im_m = 1 / \sqrt{Nn_{m-1}}} \qquad \mathbf{(12)}$$

где $Nn_m$ - число нейронов в слое m [5].

Стратегия обучения определяет общий способ обучения нейронной сети. В рассматриваемой схеме используется групповое обучение. При групповом обучении (batch training) параметры нейронной сети изменяются после того, как на нее будет подан весь набор векторов обучающей выборки. Это можно представить следующим алгоритмом [3, 5]:

*1)Для всего набора векторов обучающей выборки*

*a)Подать на сеть один вектор из обучающей выборки и вычислить градиент ∂E/∂W*

*b)Аккумулировать ∂E/∂W*

*2)Изменить параметры сети в соответствии с градиентом ∂E/∂W*

Если минимизируется ошибка сети

$$\mathbf{E = 1 / N_{mp} * \sum^{k} E_k} \qquad \mathbf{(13)}$$

где $N_{mp}$ - число векторов в обучающей выборке, $E_k$ - ошибка сети для k-го вектора, то аккумулировать $\partial E/\partial W$ следует следующим образом

$$\mathbf{\partial E/\partial W = 1 / N_{mp} * \sum^{k} \partial E/\partial W_k} \qquad \mathbf{(14)}$$

В качестве $E_k$ может использоваться среднеквадратичная ошибка

$$\mathbf{E_k = 1 / N_n N_{m-1} * \sum^{n} ( yN_{m-1,n} - out_{k,n} )} \qquad \mathbf{(15)}$$

где $N_nN_{m-1}$ - число нейронов в выходном слое, $yN_{m-1,n}$ - выход n-го нейрона выходного слоя, $out_{k,n}$ - ожидаемое значение n-го выходного элемента для k-го вектора обучающей выборки.

Достоинства используемой стратегии:

1. гарантированная сходимость к локальному минимуму;
2. большое количество приемов улучшения сходимости;

Недостатки:

1. низкая скорость обучения.

Для достижения лучшей способности к обобщению классическая схема группового обучения была модифицирована таким образом, чтобы оценивать на каждой итерации ошибку на проверочном наборе. За конечный результат обучения принимается состояние нейронной сети в момент, соответствующий минимальному значению ошибки обобщения.

На фазе обратного распространения ошибки рекуррентно вычисляется градиент $\partial E/\partial W$ для каждого слоя через ошибку следующего слоя сети [5]:

*1) для выходного слоя*

*а) для каждого нейрона выходного слоя вычислить:*

$$\delta_{m,n} = ( y_{m,n} - out_n ) * F`( S_{m,n} ) \quad (16)$$

$$\partial E/\partial T_{m,n} = - \delta_{m,n} \quad (17)$$

*i) для каждого веса нейрона вычислить*

$$\partial E/\partial w_{m,n,ni} = \delta_{m,n} * y_{m-1,ni} \quad (18)$$

*2) последовательно для скрытых слоев, начиная с последнего*

*а) для каждого нейрона в слое вычислить:*

*i)* $\delta_{m,n}$

$$\delta_{m,n} = F`( S_{m,n} ) * \sum_{ni} \delta_{m+1,ni} * w_{m+1,ni,n} \quad (19)$$

*здесь F` - производная функции активации следующего слоя.*

*ii)* $\partial E/\partial T_{m,n}$ *и* $\partial E/\partial w_{m,n,ni}$ *по (17) и (18)*

Для изменения параметров сети в процессе обучения применяется алгоритм RPROP.

Resilient error backpropagation - алгоритм, предложенный в 1992 году (последняя модификация в 1994) [10, 11]. Это прямой адаптивный алгоритм (он адаптирует изменение весов на основе градиента ошибки), не зависящий от величины производной функции активации. Таким образом, в этом алгоритме вычисляется адаптивное изменение для каждого параметра:

$$\Delta_{ij}(t) = \begin{cases} \Delta_{ij}(t-1)*\eta^+, \text{ если } \partial E/\partial w_{ij}(t-1)*\partial E/\partial w_{ij}(t) > 0, \\ \Delta_{ij}(t-1)*\eta^-, \text{ если } \partial E/\partial w_{ij}(t-1)*\partial E/\partial w_{ij}(t) < 0, \\ \Delta_{ij}(t-1), \text{ иначе} \end{cases}$$

где $0 < \eta^- < 1 < \eta^+$.

Алгоритм RPROP можно описать следующим образом:

*1) для каждого слоя сети, начиная с первого скрытого*

*a) для каждого нейрона слоя*

*i) Если* $\partial E/\partial w_{m,n,ni}^{t} * \partial E/\partial w_{m,n,ni}^{t-1} > 0$

$\Delta\Delta \mathbf{w_{m,n,ni}} = \mathbf{min(}\ \Delta\Delta_+ * \Delta\Delta \mathbf{w_{m,n,ni}}, \Delta\Delta_{max}\ \mathbf{)}$

$\Delta \mathbf{w_{m,n,ni}} = \mathbf{-SGN(}\ \partial \mathbf{E}/\partial \mathbf{w_{m,n,ni}}\ \mathbf{)} * \Delta\Delta \mathbf{w_{m,n,ni}}$

$\mathbf{w_{m,n,ni}} = \mathbf{w_{m,n,ni}} + \Delta \mathbf{w_{m,n,ni}}$

$\partial \mathbf{E}/\partial \mathbf{w_{m,n,ni}^{t-1}} = \partial \mathbf{E}/\partial \mathbf{w_{m,n,ni}^{t}}$

*ii) Если* $\partial E/\partial w_{m,n,ni}^{t} * \partial E/\partial w_{m,n,ni}^{t-1} < 0$

$\Delta\Delta \mathbf{w_{m,n,ni}} = \mathbf{max(}\ \Delta\Delta_- * \Delta\Delta \mathbf{w_{m,n,ni}}, \Delta\Delta_{min}\ \mathbf{)}$

$\partial \mathbf{E}/\partial \mathbf{w_{m,n,ni}^{t-1}} = \mathbf{0}$

*iii) Если* $\partial E/\partial w_{m,n,ni}^{t} * \partial E/\partial w_{m,n,ni}^{t-1} = 0$

$\Delta \mathbf{w_{m,n,ni}} = \mathbf{-SGN(}\ \partial \mathbf{E}/\partial \mathbf{w_{m,n,ni}}\ \mathbf{)} * \Delta\Delta \mathbf{w_{m,n,ni}}$

$\mathbf{w_{m,n,ni}} = \mathbf{w_{m,n,ni}} + \Delta \mathbf{w_{m,n,ni}}$

$\partial \mathbf{E}/\partial \mathbf{w_{m,n,ni}^{t-1}} = \partial \mathbf{E}/\partial \mathbf{w_{m,n,ni}^{t}}$

*iv) Аналогично для порога*

Таким образом, если алгоритм проскакивает локальный минимум, то изменение параметра не происходит, а шаг изменения уменьшается, если знак производной ошибки не изменяется, то происходит увеличение шага изменения параметра.

$\Delta\Delta_{max}$ задает максимальное значение изменения параметров.

$\Delta\Delta_{min}$ задает минимальное значение изменения параметров.

$\Delta\Delta_+$ задает скорость увеличения изменения параметров.

$\Delta\Delta_-$ задает скорость уменьшения изменения параметров.

Авторы алгоритма рекомендуют следующие значения для приведенных выше величин:

$\Delta\Delta_{max} = \mathbf{50}$

$\Delta\Delta_{min} = \mathbf{10^{-6}}$

$\Delta\Delta_+ = \mathbf{1,2}$

$\Delta\Delta_- = \mathbf{0,5}$

### 3.2.3 Классификация

Для обеспечения возможности классификации дефектов необходимо обучить 4 нейронные сети соответствующие классам дефектов, определять вероятность принадлежности аномалии, соответствующей вектору признаков, каждому из классов:

1. внутренняя коррозия;
2. внешняя коррозия;
3. внутренняя трещина;
4. внешняя трещина.

Набор данных для каждой из нейронных сетей формируется следующим образом:


*Рисунок 26: Структурная схема классификатора*

- входной вектор – вектор оригинальных признаков, дополненный значениями толщины стенки трубы и напряженности магнитного поля;
- выходной вектор, содержит единственный элемент равный 1, если номер нейронной сети соответствует классу дефекта, иначе 0.

Нейроны скрытых слоев каждой из сетей имеют биполярную сигмоидальную функцию активации с диапазоном значений [-1; 1]. Нейроны выходного слоя каждой из сетей

имеют униполярную сигмоидальную функцию активации с диапазоном значений [0; 1]. Таким образом, на выходе сети сразу формируется вероятность принадлежности дефекта конкретному классу. Значения с выходов всех сетей подаются на вход блока сравнения, который формирует на своем выходе значение, равное номеру сети с максимальным выходом, то есть класс дефекта, к которому с максимальной вероятностью принадлежит дефект.

### 3.2.4 Компенсация отличия значений толщины стенки трубы и напряженности магнитного поля от номинальных

Компенсатор состоит из 11 блоков, соответствующих 11 признакам, по 4 нейронной сети, соответствующих 4 классам дефектов, в каждом (см. рисунок 16).

Набор данных для каждой из сетей формируется следующим образом:

- производится выборка всех уникальных сочетаний параметров дефектов для данного класса:
  - глубина дефекта относительно стенки трубы;
  - протяженность дефекта в угловом направлении;
  - протяженность дефекта в осевом направлении;
- для каждого из сочетаний параметров из полученного множества (стоит отметить, что каждое сочетание параметров определяет уникальный дефект, для каждого из которых имеется набор векторов соответствующих различным сочетаниям толщины стенки трубы и напряженности магнитного поля):
  - выбрать вектор признаков, соответствующий рассматриваемому сочетанию параметров дефекта при номинальных значениях толщины стенки трубы (8мм) и напряженности магнитного поля (8000А/м);
  - выбрать все вектора признаков, соответствующие рассматриваемому сочетанию параметров дефекта;
  - для каждого вектора признаков из полученного набора добавить в обучающую выборку элемент, у которого входной вектор состоит из вектора признаков, дополненного значениями толщины стенки трубы и напряженности магнитного поля, а выходной вектор содержит единственное значение, равное значению соответствующего признака, полученного на первом шаге.

Нейронная сеть, обученная таким образом, будет компенсировать отличие значений толщины стенки трубы и напряженности магнитного поля от номинальных, пересчитывая значение признака для конкретного дефекта в номинальный режим.

Так как оригинальная база модельных дефектов рассчитана путем вариации абсолютной глубины дефекта, то в ней для каждого значения глубины дефекта относительно стенки трубы существует от 0 до 3 векторов соответствующих различным толщинам стенки трубы для каждого из дефектов. Такая база не может быть использована для обучения нейронных сетей, входящих в состав блока компенсации, поэтому требуется ее

корректировка путем регуляризации сетки признаков соответствующей сочетаниям глубины дефекта относительно стенки трубы и самой толщины трубы. Данная корректировка проводится путем аппроксимации недостающих значений в узлах регуляризованной сетки, соответствующей:

- значениям толщины стенки трубы (мм): 8, 9.6, 12, 14.4, 16;
- значениям глубины дефекта относительно стенки трубы (%): 20, 30, 40, 50, 60, 70, 80, 90.

Аппроксимация производится путем линейной интерполяции по двум ближайшим точкам, соответствующим вариации глубины дефекта относительно стенки трубы.

Нейроны скрытых слоев каждой из сетей имеют биполярную сигмоидальную функцию активации с диапазоном значений [-1; 1]. Нейроны выходного слоя каждой из сетей имеют линейную функцию активации с диапазоном значений [$-\infty$; $+\infty$].

### 3.2.5 Вычисление параметров

Подсистема вычисления параметров содержит 3 блока, соответствующих трем параметрам дефектов, по 4 нейронной сети, соответствующих 4 классам дефектов, в каждом.

Набор данных для каждой из сетей формируется следующим образом:

- производится выборка всех всех элементов, соответствующих рассматриваемому типу дефекта при номинальных значениях толщины стенки трубы (8мм) и напряженности магнитного поля (8000А/м);
- для каждого элемента из полученного набора добавить в обучающую выборку элемент, у которого входом является вектор признаков, а выход – значение единственного параметра, соответствующего рассматриваемому.

Нейроны скрытых слоев каждой из сетей имеют биполярную сигмоидальную функцию активации с диапазоном значений [-1; 1]. Нейроны выходного слоя каждой из сетей имеют линейную функцию активации с диапазоном значений $[-\infty; +\infty]$.

# 4 Практическое применение

## *4.1 Методика проведения численных экспериментов*

При настройке системы классификации и вычисления параметров, было проведено обучение 60 нейронных сетей, входящих в ее состав:

- 4 нейронные сети для определения класса дефекта;
- 44 нейронные сети для компенсации отличия значений толщины стенки трубы и напряженности магнитного поля от номинальных (11 блоков по 4 нейронные сети);
- 12 нейронных сетей для определения параметров дефектов (3 блока по 4 нейронные сети).

Для каждой из нейронных сетей проводился экстенсивный поиск в пространстве начальных значений весов по описанной ранее методике. Максимальное количество попыток при этом устанавливалось в 10.

Для каждой из нейронных сетей проводился поиск оптимального числа скрытых слоев и нейронов в них по описанной ранее методике. При этом рассматривались следующие сочетания:

- один скрытый слой с 5 нейронами;
- один скрытый слой с 10 нейронами;
- один скрытый слой с 20 нейронами;
- два скрытых слоя с 5 нейронами в каждом;
- два скрытых слоя с 10 нейронами в каждом;
- два скрытых слоя с 20 нейронами в каждом.

Для каждой из нейронных сетей проводился поиск оптимального набора признаков по описанной ранее методике.

Процесс обучения каждой из нейронных сетей останавливался после 1000 итераций.

Для каждой из нейронных сетей имеющийся набор данных делился на обучающее и проверочное множества в соотношении 1 вектор проверочного множества на каждые 3 вектора обучающего.

Коэффициент, характеризующий ширину линейной части по оси абсцисс, для сигмоидальной функции активации нейронов во всех случаях брался равным 1.

Для всех нейронных сетей проводилось масштабирование входных и выходных (кроме нейронных сетей, осуществляющих классификацию) данных по описаннной ранее методике.

Для алгоритма настройки весов нейронной сети RPROP во всех случаях параметры брались следующими:

- $\Delta\Delta_{max} = 50$
- $\Delta\Delta_{min} = 10^{-6}$
- $\Delta\Delta_{+} = 1,2$
- $\Delta\Delta_{-} = 0,5$

За погрешность классификации принималось отношение числа дефектов, для которых был неверно определен их класс, к общему числу дефектов в модельной базе данных.

За погрешность определения каждого из параметров принималась максимальная абсолютная погрешность при определении параметра среди всех дефектов модельной базы данных.

## 4.2 Результаты численных экспериментов

### 4.2.1 Найденные параметры

***Таблица 4: Настройки нейронных сетей при классификации***

| *Класс дефекта* | *Набор признаков* | *Конфигурация скрытых слоев* |
|---|---|---|
| Внутренняя трещина | 4 7 2 8 3 6 10 2 11 | 10 10 |
| Внешняя трещина | 4 1 5 10 6 9 | 5 5 |
| Внутренняя коррозия | 7 3 8 9 10 4 2 | 10 10 |
| Внешняя коррозия | 7 3 4 8 5 1 2 11 6 10 9 | 20 20 |

***Таблица 5: Настройки нейронных сетей при компенсации изменения изначений толщины стенки трубы и напряженности магнитного поля***

| *Класс дефекта* | *Признак* | *Набор признаков* | *Конфигурация скрытых слоев* |
|---|---|---|---|
| Внутренняя трещина | 1 | 1 7 11 6 2 8 3 | 10 10 |
| Внутренняя трещина | 2 | 2 8 4 7 6 1 5 3 | 20 |
| Внутренняя трещина | 3 | 3 6 7 8 11 4 1 9 2 5 10 | 10 10 |
| Внутренняя трещина | 4 | 4 10 7 6 3 1 11 | 20 |
| Внутренняя трещина | 5 | 5 2 8 3 6 1 10 11 9 | 20 |
| Внутренняя трещина | 6 | 6 7 9 11 4 1 10 8 5 3 | 20 20 |
| Внутренняя трещина | 7 | 7 4 2 5 6 3 8 9 | 5 5 |
| Внутренняя трещина | 8 | 8 6 2 4 11 9 10 1 | 20 20 |
| Внутренняя трещина | 9 | 9 6 4 8 3 7 1 11 | 20 20 |
| Внутренняя трещина | 10 | 10 7 4 6 3 1 5 8 | 10 10 |
| Внутренняя трещина | 11 | 11 7 3 2 1 4 9 10 8 5 6 | 10 10 |
| Внешняя трещина | 1 | 1 3 5 7 8 9 11 4 | 20 |
| Внешняя трещина | 2 | 2 8 5 7 6 4 3 | 20 |
| Внешняя трещина | 3 | 3 9 7 2 6 5 4 1 10 | 20 |
| Внешняя трещина | 4 | 4 7 6 10 1 2 9 3 11 8 5 | 5 5 |
| Внешняя трещина | 5 | 5 8 7 6 2 9 1 11 10 4 3 | 20 |
| Внешняя трещина | 6 | 6 7 4 10 2 9 1 5 8 11 | 20 20 |
| Внешняя трещина | 7 | 7 4 3 6 1 8 10 | 20 |
| Внешняя трещина | 8 | 8 10 4 2 11 1 6 5 9 | 20 |

| *Класс дефекта* | *Признак* | *Набор признаков* | *Конфигурация скрытых слоев* |
|---|---|---|---|
| Внешняя трещина | 9 | 9 3 6 7 5 2 4 8 11 10 1 | 10 10 |
| Внешняя трещина | 10 | 10 4 6 3 5 1 8 9 2 | 10 |
| Внешняя трещина | 11 | 11 7 1 6 9 4 3 5 2 8 10 | 20 |
| Внутренняя коррозия | 1 | 1 9 5 11 4 10 3 2 7 | 5 5 |
| Внутренняя коррозия | 2 | 2 4 5 8 6 7 9 10 1 11 | 10 10 |
| Внутренняя коррозия | 3 | 3 6 9 4 1 2 10 | 10 10 |
| Внутренняя коррозия | 4 | 4 7 3 11 9 8 10 | 5 5 |
| Внутренняя коррозия | 5 | 5 2 8 4 7 6 9 | 5 5 |
| Внутренняя коррозия | 6 | 6 7 9 11 3 2 4 1 | 10 10 |
| Внутренняя коррозия | 7 | 7 4 9 3 10 8 2 6 1 11 | 20 |
| Внутренняя коррозия | 8 | 8 5 6 4 7 2 9 3 1 11 | 10 10 |
| Внутренняя коррозия | 9 | 9 7 6 3 4 8 11 | 20 |
| Внутренняя коррозия | 10 | 10 7 4 2 6 8 11 1 | 20 |
| Внутренняя коррозия | 11 | 11 6 1 7 8 10 3 | 10 10 |
| Внешняя коррозия | 1 | 1 6 11 5 7 3 2 4 8 9 10 | 5 5 |
| Внешняя коррозия | 2 | 2 4 3 8 11 5 7 | 10 10 |
| Внешняя коррозия | 3 | 3 6 4 7 8 5 11 9 2 10 1 | 10 10 |
| Внешняя коррозия | 4 | 4 7 6 3 10 5 8 2 11 1 | 20 |
| Внешняя коррозия | 5 | 5 2 3 7 9 10 4 8 1 11 | 20 20 |
| Внешняя коррозия | 6 | 6 3 2 1 9 5 8 11 10 | 20 |
| Внешняя коррозия | 7 | 7 4 2 5 10 6 3 11 8 9 1 | 10 10 |
| Внешняя коррозия | 8 | 8 4 5 3 9 10 11 7 2 6 | 10 10 |
| Внешняя коррозия | 9 | 9 6 4 11 1 5 7 2 | 10 10 |
| Внешняя коррозия | 10 | 10 4 1 3 8 7 11 2 | 20 |
| Внешняя коррозия | 11 | 11 7 6 3 8 9 2 10 | 20 |

***Таблица 6: Настройки нейронных сетей при вычислении параметров дефектов***

| *Класс дефекта* | *Параметр* | *Набор признаков* | *Конфигурация скрытых слоев* |
|---|---|---|---|
| Внутренняя трещина | Протяж. угл. | 3 9 6 10 1 2 5 4 | 20 20 |
| Внутренняя трещина | Протяж. осев. | 10 7 8 4 9 2 5 | 10 10 |
| Внутренняя трещина | Глубина | 1 11 2 9 6 7 5 10 4 3 8 | 20 20 |
| Внешняя трещина | Протяж. угл. | 9 3 2 8 10 4 5 6 | 10 10 |

| ***Класс дефекта*** | ***Параметр*** | ***Набор признаков*** | ***Конфигурация скрытых слоев*** |
|---|---|---|---|
| Внешняя трещина | Протяж. осев. | 7 4 10 8 2 11 9 3 5 | 10 10 |
| Внешняя трещина | Глубина | 1 11 5 2 8 3 4 6 9 7 | 20 20 |
| Внутренняя коррозия | Протяж. угл. | 9 6 3 8 2 7 1 4 10 5 | 20 |
| Внутренняя коррозия | Протяж. осев. | 7 5 4 2 8 6 11 3 9 10 | 5 5 |
| Внутренняя коррозия | Глубина | 1 11 5 3 2 8 6 7 9 4 10 | 10 10 |
| Внешняя коррозия | Протяж. угл. | 3 7 10 11 6 2 5 4 9 8 | 10 10 |
| Внешняя коррозия | Протяж. осев. | 2 5 7 9 4 11 10 6 8 | 20 |
| Внешняя коррозия | Глубина | 1 8 11 5 6 3 2 4 10 | 10 10 |

### 4.2.2 Погрешности при классификации и определении параметров

***Таблица 7: Погрешности классификации и определения параметров***

| ***Задача*** | ***Погрешность*** |
|---|---|
| Классификация | 17.2% |
| Протяж. угл. | 18.7мм |
| Протяж. осев. | 7.2мм |
| Глубина | 42.4% |

# 5 Недостатки разработанной модели

В ходе исследования разработанных методик, практического применения, созданного программного комплекса, а также анализа полученных результатов были выявлены следующие недостатки:

1. Использование экстенсивного поиска в пространстве начальных значений весов нейронов многократно увеличивает время, необходимое на настройку системы. Причем оно растет экспоненциально с ростом максимального числа попыток. Эффективность же данного метода в плане улучшения качества настройки прямо проворциональна числу произведенных попыток. Поэтому для получения значимого улучшения качества настройки требуется многократное увеличение времени, затраченного на данный процесс.

2. Использование жестко заданного набора сочетаний числа скрытых слоев нейронной сети и количества нейронов в каждом из них не гарантирует нахождение оптимальной структуры сети в силу ограниченности количества элементов в данном наборе, а лишь обеспечивает выбор наилучего размера среди описанных этим набором.

3. Разработанные методики не позволяют отстроиться от влияния изменения зазора между датчиками и стенкой трубы, которое является неопределенным мешающим фактором и недопустимо увеличивает погрешности классификации и определения параметров. Поэтому требуется дополнительная система, компенсирующая влияние зазора, либо вычисляющая его.

Возможные пути улучшения разработанных методов:

1. Для сокращения времени, необходимого на настройку системы при одновременном сохранении и даже улучшении ее качесива, можно уменьшить максимальное число попыток экстенсивного поиска при условии одновременного использования методов с элементами стохастического обучения, таких как метод имитации отжига или генетические алгоритмы.

2. Для более полного поиска оптимального размера нейронной сети при одновременном сокращении времени настройки могут быть применены методы ее редукции (например, алгоритм Optimal brain damage, заключающийся в удалении связей) либо наращивания (например, алгоритм динамического построения каскада нейронов CasPer).

3. Проблема отстройки от зазора требует дальнейшего анализа и поиска решения. Одним из которых может быть применение метода инвариантных преобразований, позволяющим при некоторых условиях исключить влияние неопределенного мешающего фактора.

# 6 Заключение

В результате проведенных исследований были разработаны методики, позволяющие производить автоматическую настройку параметров искусственных нейронных сетей:

- количество скрытых слоев;
- количество нейронов в скрытых слоях;
- наборы используемых признаков.

Дополнительно была разработана методика, позволяющие улучшить качество настройки: экстенсивный поиск в пространстве начальных значений весов нейронов.

Был успешно опробован способ отстройки от влияния изменения толщины стенки трубы и режима намагничивания путем применения каскада нейронных сетей с пересчетом признаков в номинальный режим.

Были внедрены и доказали свою эффективность передовые методы в построении программных систем:

- использование СУБД для хранения и управления данными;
- использование объектно-ориентированной технологии для построения гибких систем;
- использование гетерогенных систем, сочитающих высокоскоростные модули, реализованные на языках программирования низкого уровня, и перенастраиваемые оболочки на высокоуровневых языках.

Был разработан программный комплекс, решающий поставленную задачу. О чем свидетельствуют полученные численные результаты.

# Библиографический список


1. А. А. Абакумов, А. А. Абакумов (мл.), Магнитная диагностика газонефтепроводов. - М.: Энергоатомиздат, 2001, 440 с.
2. В. Г. Герасимов, А. Д. Покровский, В. В. Сухоруков, Неразрушающий контроль. В 5 кн. Кн. 3. Электромагнитный контроль. - М.: Высш. шк., 1992, 312 с.
3. В. А. Головко, Нейронные сети: обучение, организация и применение. -М.: ИПРЖ, 2001, 256 с.
4. С. Осовский, Нейронные сети для обработки информации, Финансы и статистика, 2002, 344 с.
5. Yann Le Cun, Efficient BackProp. - Holmdel, 1996, 75 с.
6. Donald R. Tveter, Backpropagator`s Review, 1996
7. Martin Riedmiller, Advanced Supervised Learning in Multi-layer Perceptrons - From Backpropagation to Adaptive Learning Algorithms - Karlsruhe, 1994, 10 с.
8. Merten Joost, Speeding up backpropagation algorithms by using cross-entropy combined with pattern normalization, 1997, 20 с.
9. Lutz Prechelt, Automatic early stopping using cross validation: quantifying the criteria. - Karlsruhe, 1997, 12 с.
10. Martin Riedmiller, RPROP - A fast adaptive learning algorithm. - Karlsruhe, 1992, 12 с.
11. Martin Riedmiller, RPROP - Description and implementation details. - Karlsruhe, 1994, 16 с.
12. W. Shiffman, Comparision of optimized backpropagation algorithms. - Koblenz, 1993, 8 с.
13. Stewe Lawrence, What size neural network gives optimal generalization? Convergence properties of backpropagation. - Oxford, 1996, 37 с.
14. B. D. Ripley, Statistical ideas for selecting network architectures. - Princeton, 1995, 8 с.
15. N.K. Treadgold, T.D. Gedeon, THE SARPROP ALGORITHM: A SIMULATED ANNEALING ENHANCEMENT TO RESILIENT BACK PROPAGATION. - Sydney, 1998, 7 с.
16. P. A. Castilo, SA-Prop: Optimization of multilayer perceptron parameters using simulated annealing. - Granada, 1998, 10 с.

# Приложение А. Список иллюстраций

# Приложение Б. Список таблиц

# Приложение В. Исходные тексты алгоритмов

## *Алгоритм распространения сигнала в нейронной сети с прямыми связями*

```
nnval MultiLayerPerceptron::get_weighted_total( unsigned layer, unsigned neuron )
{
   nnval res = 0.0;
   unsigned prev_layer_size = get_layer_size( layer - 1 );
   std::vector< nnval > &prev_layer_f = f[ layer - 1 ];
   std::vector< nnval > &cur_neuron_weights = w[ layer ][ neuron ];
   for( unsigned prev_neuron = 0; prev_neuron < prev_layer_size; prev_neuron++ )
      res += cur_neuron_weights[ prev_neuron ] * prev_layer_f[ prev_neuron ];
   return res;
}

void MultiLayerPerceptron::feed( const Image &iimage )
{
   // set input
   unsigned input_layer_size = get_layer_size( 0 );
   std::vector< nnval > &input_layer = f[ 0 ];
   for( unsigned neuron = 0; neuron < input_layer_size - 1; neuron++ )
      input_layer[ neuron ] = iimage[ neuron ];
   input_layer[ input_layer_size - 1 ] = bias_neuron_out;
   // propogate signal
   unsigned layers_count = get_layers_count();
   for( unsigned layer = 1; layer < layers_count; layer++ )
   {
      unsigned layer_size = get_layer_size( layer );
      RealActivationFunction cur_layer_af = activation_functions[ layer ]->get_real_function();
      for( unsigned neuron = 0; neuron < layer_size - 1; neuron++ )
      {
         (*cur_layer_af)(
            get_weighted_total( layer, neuron ),
            f[ layer ][ neuron ],
            fds[ layer ][ neuron ]
            );
```

```
        }
        f[ layer ][ layer_size - 1 ] = bias_neuron_out;
    }
}
```

## *Алгоритм масштабирования данных*

```
def get_vectors_scale_params( vectors, nmin, nmax ):
    res = {
        'mean' : [ 0 ] * len( vectors[ 0 ] ),
        'dev' : [ 0 ] * len( vectors[ 0 ] ),
        'nmin' : [ 0 ] * len( vectors[ 0 ] ),
        'fac' : [ 0 ] * len( vectors[ 0 ] )
        }
    # mean
    for vector in vectors:
        for i in range( len( vector ) ):
            res[ 'mean' ][ i ] = res[ 'mean' ][ i ] + vector[ i ]
    for i in range( len( vector ) ):
        res[ 'mean' ][ i ] = float( res[ 'mean' ][ i ] ) / len( vectors )
    # dev
    for vector in vectors:
        for i in range( len( vector ) ):
            res[ 'dev' ][ i ] =                         \
                res[ 'dev' ][ i ]                       \
                +                                       \
                    ( vector[ i ] - res[ 'mean' ][ i ] )    \
                    * ( vector[ i ] - res[ 'mean' ][ i ] )
    for i in range( len( vector ) ):
        res[ 'dev' ][ i ] = math.sqrt( float( res[ 'dev' ][ i ] ) / len( vectors ) )
    # factor
    for i in range( len( vector ) ):
        res[ 'fac' ][ i ] = float( nmax - nmin ) / ( 1 - (-1) )
    for i in range( len( vector ) ):
        res[ 'nmin' ][ i ] = nmin
    return res

def scale_vectors( vectors, params ):
    res = vectors
    for v in range( len( vectors ) ):
        for i in range( len( vectors[ v ] ) ):
            if params[ 'dev' ][ i ] != 0:
                res[ v ][ i ] =                                         \
```

```
                (                                                   \
                    float( vectors[ v ][ i ] - params[ 'mean' ][ i ] )  \
                    / params[ 'dev' ][ i ]                          \
                    - (-1)
                ) * params[ 'fac' ][ i ] + params[ 'nmin' ][ i ]
    return res

def descale_vectors( vectors, params ):
    res = vectors
    for v in range( len( vectors ) ):
        for i in range( len( vectors[ v ] ) ):
            res[ v ][ i ] =                                         \
                (                                                   \
                    float( vectors[ v ][ i ] - params[ 'nmin' ][ i ] )  \
                    / params[ 'fac' ][ i ]                          \
                    + (-1)
                ) * params[ 'dev' ][ i ] + params[ 'mean' ][ i ]
    return res

def get_scale_tps_params( tps, nmin, nmax ):
    inl = []
    outl = []
    for tp in tps:
        inl.append( tp[ 0 ] )
        outl.append( tp[ 1 ] )
    return [
        get_vectors_scale_params( inl, nmin, nmax ),
        get_vectors_scale_params( outl, nmin, nmax )
        ]

def scale_tps( tps, params ):
    res = []
    for tp in tps:
        res.append( [
                scale_vectors( [ tp[ 0 ] ], params[ 0 ] )[ 0 ],
                scale_vectors( [ tp[ 1 ] ], params[ 1 ] )[ 0 ],
            ] )
    return res
```

```
def descale_tps( tps, params ):
    res = []
    for tp in tps:
        res.append( [
                descale_vectors( [ tp[ 0 ] ], params[ 0 ] )[ 0 ],
                descale_vectors( [ tp[ 1 ] ], params[ 1 ] )[ 0 ],
            ] )
    return res
```

## *Алгоритм экстенсивного поиска*

```
def    train_ann( ann, data ):
    global max_iterations, iterations_between_reports, desired_error
    ann.randomize_weights( -1, 1 )
    ann.train_on_data( data, max_iterations, iterations_between_reports, desired_error )

def ann_test_mse( ann, data ):
    res = 0
    for i in range( data.get_num_data() ):
        ov = ann.run( data.get_input( i ) )
        ce = 0
        for j in range( len( ov ) ):
            ce = ce + ( ov[ j ] - data.get_output( i )[ j ] ) * ( ov[ j ] - data.get_output( i )[ j ] )
        res = res + float( ce ) / len( ov )
    return float( res ) / data.get_num_data()

def long_train_ann( ann, idata, tdata ):
    global try_count
    min_mse = -1
    for i in range( try_count ):
        train_ann( ann, idata )
        mse = ann_test_mse( ann, tdata )
        if mse < min_mse or min_mse == -1:
            min_mse = mse
            ann.save( "tmp.best-try.net" )
    return fann.create_from_file( "tmp.best-try.net" )
```

### *Алгоритм поиска оптимального размера нейронной сети*

```
def find_geometry( iname, tname, dimensions, is_out_linear, log_name ):
    idata = read_input( iname )
    tdata = read_input( tname )
    bvmse = -1
    bd = []
    for d in dimensions:
        ann = create_ann( get_dimensions( idata, d ), is_out_linear )
        annt = long_train_ann( ann, idata, tdata )
        vmse = ann_test_mse( annt, tdata )
        log_progress( d, vmse, log_name )
        if vmse < bvmse or bvmse == -1:
            bvmse = vmse
            bd = d
            annt.save( "tmp.best-size.net" )
    import pickle
    fh = file( 'find_size.res.txt', 'w' )
    pickle.dump( { 'mse' : bvmse, 'dimensions' : bd }, fh )
    fh.close()
    return fann.create_from_file( "tmp.best-size.net" )

def find_size(
    iname = "train.data.txt",
    tname = "validate.data.txt",
    geometries = def_geomentries,
    is_out_linear = sigmoid,
    log_name = def_log_name,
    net_name = "best_size.net"
    ):
    random.seed()
    if os.path.exists( log_name ):
        os.unlink( log_name )
    ann = find_geometry( iname, tname, geometries, linear, log_name )
    os.unlink( "tmp.best-size.net" )
    os.unlink( "tmp.best-try.net" )
    ann.save( net_name )
    return ann
```

## *Алгоритм выбора признаков*

```
def try_features( features, target, criteria ):
    global t_v_ratio, skip_ratio
    raw_samples = dbselect( features + target, criteria )
    samples = data.vectors2tps( raw_samples, features, target )
    if skip_ratio < 1:
        data.shuffletps( samples )
        samples, tmp = data.splittps( samples, skip_ratio )
    p = data.get_scale_tps_params( samples, -1, 1 )
    samples = data.scale_tps( samples, p )
    data.shuffletps( samples )
    t, v = data.splittps( samples, t_v_ratio )
    data.save_tps( samples, 'all.dat.txt' )
    data.save_tps( t, 't.dat.txt' )
    data.save_tps( v, 'v.dat.txt' )
    ann = find_size.find_size(
        't.dat.txt',
        'v.dat.txt'
        )
    fh = file( 'find_size.res.txt', 'r' )
    fs_res = pickle.load( fh )
    fh.close()
    return {
        'features' : features,
        'target' : target,
        'criteria' : criteria,
        'mse' : fs_res[ 'mse' ],
        'dimensions' : fs_res[ 'dimensions' ]
        }

def find_features( features, target, criteria ):
    res = []
    for f in features:
        res.append( try_features( [ f ], target, criteria ) )
    res.sort( lambda a, b : cmp( a[ 'mse' ], b[ 'mse' ] ) )
    for i in res:
        log_progress( "%s : %s" % ( i[ 'features' ][ 0 ], i[ 'mse' ] ) )
```

```
res2 = []
for i in range( len( features ) ):
    sub_res = res[ : i + 1 ]
    new_features = []
    for j in sub_res:
        new_features.append(  j[ 'features' ][ 0 ] )
    try:
        res2.append( try_features( new_features, target, criteria ) )
    except:
        print "failed with %s" % new_features
res2.sort( lambda a, b : cmp( a[ 'mse' ], b[ 'mse' ] ) )
for i in res2:
    log_progress( "%s : %s : %s" % ( i[ 'features' ], i[ 'mse' ], i[ 'dimensions' ] ) )
```

## *Алгоритм инициализации весов нейронной сети*

```
class NN_IS_Practical : public NN_Initialization_Strategy
{
public:
  virtual void init( Neural_Net &nn )
    {
      // by layers
      for( unsigned int l = 0; l < nn.size(); l++ )
      {
        // by neurons
        for( unsigned int n = 0; n < nn[ l ].size(); n++ )
        {
          // by weights
          for( unsigned int w = 0; w < nn[ l ][ n ].w.size(); w++ )
          {
            if( l )
              nn[ l ][ n ].w[ w ] = ( (nnval)rand() / RAND_MAX - 0.5 ) * 2.0 / sqrt( nn[ l ].size() );
            else
              nn[ l ][ n ].w[ w ] = 1.0;
          } // by weights
          nn[ l ][ n ].T = 1.0 /*( (nnval)rand() / RAND_MAX - 0.5 ) * 2.0 / sqrt( nn[ l ].size() )*/;
        } // by neurons
      } // by layers
    };
  virtual std::string serialize()
    {
      return "init_practical";
    };
  virtual int deserialize( std::string cfg )
    {
      return cfg == "init_practical";
    };
};
```

## *Алгоритм групповой стратегии обучения*

```
void NN_LS_Batch::learn_set(
  Learning_Patterns_Set &pset,
  Neural_Net &nn,
  const Neural_Net_Conf &nnc,
  Learning_Patterns_Set &tset
  )
{
  assert( nnc.la );
  assert( nnc.ev );
  //
  nnval best_errg = -1.0;
  nnval best_iter = 0;
  Neural_Net best_net;
  //
  nnval error = 0.0;
  nnval errorg = 0.0;
  nnval last_error = 0.0;
  int iterations = 0;
  // Zero errors
  for( unsigned int l = 0; l < nn.size(); l++ )
  {
    for( unsigned int n = 0; n < nn[ l ].size(); n++ )
    {
      nn[ l ][ n ].ddT = nnc.dw0;
      nn[ l ][ n ].dE_T = 0.0;
      nn[ l ][ n ].dE_T_tm1 = 0.0;
      for( unsigned int ni = 0; ni < nn[ l ][ n ].dE_w.size(); ni++ )
      {
        nn[ l ][ n ].ddw[ ni ] = nnc.dw0;
        nn[ l ][ n ].dE_w[ ni ] = 0.0;
        nn[ l ][ n ].dE_w_tm1[ ni ] = 0.0;
      }
    }
  }
  //
  Neural_Net dEt = nn;
```

```
//
nnc.es->init();
Neural_Net old_nn;
do // while( error > nnc.desired_error );
{
 // Dump statistics
 stat::w( nn[ 1 ][ 0 ].w[ 0 ] );
 stat::dE_dw( nn[ 1 ][ 0 ].dE_w[ 0 ] );
 stat::dw( nn[ 1 ][ 0 ].dw[ 0 ] );
 // Remember error
 for( unsigned int l = 0; l < dEt.size(); l++ )
 {
  for( unsigned int n = 0; n < dEt[ l ].size(); n++ )
  {
   dEt[ l ][ n ].dE_T = 0.0;
   for( unsigned int ni = 0; ni < dEt[ l ][ n ].dE_w.size(); ni++ )
   {
    dEt[ l ][ n ].dE_w[ ni ] = 0.0;
   }
  }
 }
 //
 last_error = error;
 iterations++;
 nnc.la->epoch = iterations;
 old_nn = nn;
 // Study
 // by patterns
 for( unsigned int i = 0; i < pset.size(); i++ )
 {
  nnc.la->forward( pset[ i ], nn, nnc );
  nnc.la->backward( pset[ i ], nn, nnc );
  // Collect errors
  for( unsigned int l = 0; l < nn.size(); l++ )
  {
   for( unsigned int n = 0; n < nn[ l ].size(); n++ )
   {
    dEt[ l ][ n ].dE_T += nn[ l ][ n ].dE_T  / pset.size();
```

```
      for( unsigned int ni = 0; ni < nn[ l ][ n ].dE_w.size(); ni++ )
      {
       dEt[ l ][ n ].dE_w[ ni ] += nn[ l ][ n ].dE_w[ ni ] / pset.size();
      }
     }
    }
} // by patterns
// Set collected dE/dw and dE/dT back
for( unsigned int l = 0; l < dEt.size(); l++ )
{
  for( unsigned int n = 0; n < dEt[ l ].size(); n++ )
  {
    nn[ l ][ n ].dE_T = dEt[ l ][ n ].dE_T;
    for( unsigned int ni = 0; ni < dEt[ l ][ n ].dE_w.size(); ni++ )
    {
      nn[ l ][ n ].dE_w[ ni ] = dEt[ l ][ n ].dE_w[ ni ];
    }
  }
}
// Compute error
error = 0.0;
for( unsigned int i = 0; i < pset.size(); i++ )
{
  assert( pset[ i ].size() > 1 );
  NN_Image img = nnc.ev->compute( pset[ i ].at( 0 ), nn, nnc );
  assert( img.size() == pset[ i ].at( 1 ).size() );
  for( unsigned int j = 0; j < img.size(); j++ )
  {
    error += sqr( ( img[ j ] - pset[ i ][ 1 ][ j ] ) / nnc.afs[ 1 ]->divider );
  }
}
error /= nn[ nn.size() - 1 ].size() * pset.size();
nnc.la->error = error;
nnc.la->update( pset[ 0 ], nn, nnc );
// Compute error
error = 0.0;
for( unsigned int i = 0; i < pset.size(); i++ )
{
```

```
  assert( pset[ i ].size() > 1 );
  NN_Image img = nnc.ev->compute( pset[ i ].at( 0 ), nn, nnc );
  assert( img.size() == pset[ i ].at( 1 ).size() );
  for( unsigned int j = 0; j < img.size(); j++ )
  {
    error += sqr( ( img[ j ] - pset[ i ][ 1 ][ j ] ) / nnc.afs[ 1 ]->divider );
  }
}
error /= nn[ nn.size() - 1 ].size() * pset.size();
//
// Compute generalization error
errorg = 0.0;
for( unsigned int i = 0; i < tset.size(); i++ )
{
  assert( tset[ i ].size() > 1 );
  NN_Image img = nnc.ev->compute( tset[ i ].at( 0 ), nn, nnc );
  assert( img.size() == tset[ i ].at( 1 ).size() );
  for( unsigned int j = 0; j < img.size(); j++ )
  {
    errorg += sqr( ( img[ j ] - tset[ i ][ 1 ][ j ] ) / nnc.afs[ 1 ]->divider );
  }
}
errorg /= nn[ nn.size() - 1 ].size() * tset.size();
if( ( errorg < best_errg ) || ( best_errg < 0.0 ) )
{
  best_errg = errorg;
  best_iter = iterations;
  best_net = nn;
}
//
cout
  << "\rIteration "
  << iterations
  << " error "
  << error
  << " generr "
  << errorg
  << " best iter "
```

```
      << best_iter
      << " gerr "
      << best_errg
      << "     "
      << flush;
    stat::le( error );
    stat::ve( errorg );
    //
    if( nnc.break_iter && ( iterations > nnc.max_iter ) )
      break;
    int esd = nnc.es->is_need_stop( error, errorg );
    if( esd > 0 )
      break;
    if( esd < 0 )
    {
      nn = old_nn;
      break;
    }
  } while(
      nnc.break_lower
      ? ( error >= nnc.desired_error )
      : ( fabs( error - last_error ) > nnc.desired_error )
    );
  cout << "\n";
  nn = best_net;
}
```

## *Алгоритм обратного распространения ошибки*

```
void MultiLayerPerceptron::calculate_gradient( const Image &oimage )
{
   // calculate error for each neuron
   // for output layer
   unsigned layers_count = get_layers_count();
   unsigned cur_layer = layers_count - 1;
   unsigned cur_layer_size = get_layer_size( cur_layer );
   for( unsigned neuron = 0; neuron < cur_layer_size; neuron++ )
   {
      theta[ cur_layer ][ neuron ] = f[ cur_layer ][ neuron ] - oimage[ neuron ];
   }
   // for hidden layers
   for( cur_layer = layers_count - 2; cur_layer > 0; cur_layer-- )
   {
      cur_layer_size = get_layer_size( cur_layer );
      unsigned next_layer = cur_layer + 1;
      unsigned next_layer_size = get_layer_size( next_layer );
      for( unsigned cur_neuron = 0; cur_neuron < cur_layer_size; cur_neuron++ )
      {
         theta[ cur_layer ][ cur_neuron ] = 0.0;
         for( unsigned next_neuron = 0; next_neuron < next_layer_size; next_neuron++ )
         {
            theta[ cur_layer ][ cur_neuron ] +=
               theta[ next_layer ][ next_neuron ]
               * w[ next_layer ][ next_neuron ][ cur_neuron ];
         }
         theta[ cur_layer ][ cur_neuron ] *= fds[ cur_layer ][ cur_neuron ];
      }
   }
   // calculate gradient based on error for each neuron
   for( cur_layer = 1; cur_layer < layers_count; cur_layer++ )
   {
      cur_layer_size = get_layer_size( cur_layer );
      unsigned prev_layer = cur_layer - 1;
      unsigned prev_layer_size = get_layer_size( prev_layer );
      for( unsigned cur_neuron = 0; cur_neuron < cur_layer_size; cur_neuron++ )
```

```
      {
         for( unsigned prev_neuron = 0; prev_neuron < prev_layer_size; prev_neuron++ )
         {
            dedw[ cur_layer ][ cur_neuron ][ prev_neuron ] =
               theta[ cur_layer ][ cur_neuron ]
               * f[ prev_layer ][ prev_neuron ];
         }
      }
   }
}
```

## *Алгоритм изменения весов нейронов RPROP*

```
class BEP_RPROP : public BackErrorPropagation
{
public:
  virtual std::string serialize()
    {
      return "rprop";
    };
  virtual int deserialize( std::string cfg )
    {
      return cfg == "rprop";
    };
  inline nnval sgn( nnval v )
    {
      return
        ( v > 0.0 )
        ? 1.0
        : (
          ( v < 0.0 )
          ? -1.0
          : 0.0
          );
    };
  inline nnval maximum( nnval v1, nnval v2 )
    {
      return
        ( v1 > v2 )
        ? v1
        : v2;
    };
  inline nnval minimum( nnval v1, nnval v2 )
    {
      return
        ( v1 < v2 )
        ? v1
        : v2;
    };
```

```
  virtual void update(
    const Learning_Pattern &pat,
    Neural_Net &net,
    const Neural_Net_Conf &nnc
    )
    {
      nnval k1 = 0.1;
      nnval k2 = 0.1;
      nnval k3 = 3.0;
      nnval r = rand() / (float)RAND_MAX;
      nnval T = 0.02;
      // Weights and theresholds change.
      // by layers
      for( unsigned int l = 1; l < net.size(); l++ )
      {
        // by neurons
        for( unsigned int n = 0; n < net.at( l ).size(); n++ )
        {
          // Update W
          for( unsigned int ni = 0; ni < net[ l ][ n ].w.size(); ni++ )
          {
            if( net[ l ][ n ].dE_w[ ni ] * net[ l ][ n ].dE_w_tm1[ ni ] > 0.0 )
            {
              // ddw = min( ddw * dd_inc, max_ddw )
              net[ l ][ n ].ddw[ ni ] = minimum( net[ l ][ n ].ddw[ ni ] * nnc.dd_inc, nnc.max_dd )
;
              // dw = -sgn( dE/dw ) * ddw
              net[ l ][ n ].dw[ ni ] = - sgn( net[ l ][ n ].dE_w[ ni ] ) * net[ l ][ n ].ddw[ ni ];
              // w = w + dw
              net[ l ][ n ].w[ ni ] += net[ l ][ n ].dw[ ni ];
              // dE/dw(t-1) = dE/dw(t)
              net[ l ][ n ].dE_w_tm1[ ni ] = net[ l ][ n ].dE_w[ ni ];
            }
            else if( net[ l ][ n ].dE_w[ ni ] * net[ l ][ n ].dE_w_tm1[ ni ] < 0.0 )
            {
              // ddw = max( ddw * dd_dec, min_ddw )
              net[ l ][ n ].ddw[ ni ] = maximum( net[ l ][ n ].ddw[ ni ] * nnc.dd_dec,
nnc.min_dd );
```

```
          // dE/dw(t-1) = 0
          net[ l ][ n ].dE_w_tm1[ ni ] = 0.0;
         }
         else
         {
          // dw = -sgn( dE/dw ) * ddw
          net[ l ][ n ].dw[ ni ] = - sgn( net[ l ][ n ].dE_w[ ni ] ) * net[ l ][ n ].ddw[ ni ];
          // w = w + dw
          net[ l ][ n ].w[ ni ] += net[ l ][ n ].dw[ ni ];
          // dE/dw(t-1) = dE/dw(t)
          net[ l ][ n ].dE_w_tm1[ ni ] = net[ l ][ n ].dE_w[ ni ];
         }
        } // By weights
        // Update T
        if( net[ l ][ n ].dE_T * net[ l ][ n ].dE_T_tm1 > 0.0 )
        {
         // Same for T
         net[ l ][ n ].ddT = minimum( net[ l ][ n ].ddT * nnc.dd_inc, nnc.max_dd );
         net[ l ][ n ].dT = - sgn( net[ l ][ n ].dE_T ) * net[ l ][ n ].ddT;
         net[ l ][ n ].T += net[ l ][ n ].dT;
         net[ l ][ n ].dE_T_tm1 = net[ l ][ n ].dE_T;
        }
        else if( net[ l ][ n ].dE_T * net[ l ][ n ].dE_T_tm1 < 0.0 )
        {
         net[ l ][ n ].ddT = maximum( net[ l ][ n ].ddT * nnc.dd_dec, nnc.min_dd );
         net[ l ][ n ].dE_T_tm1 = 0.0;
        }
        else
        {
         // Same for T
         net[ l ][ n ].dT = - sgn( net[ l ][ n ].dE_T ) * net[ l ][ n ].ddT;
         net[ l ][ n ].T += net[ l ][ n ].dT;
         net[ l ][ n ].dE_T_tm1 = net[ l ][ n ].dE_T;
        }
       } // by neurons
     } // by layers
   };
};
```

## *Алгоритм корректировки базы модельных дефектов для компенсации отличия значений толщины стенки трубы и напряженности магнитного поля от номинальных*

```
def inter( xs, ys, x):
    ileft = -1
    iright = -1
    for i in range( len( xs ) ):
        if xs[ i ] < x or ileft == -1:
            ileft = i
        else:
            break
    if ileft == -1:
        ileft = 0
        iright = 1
    else:
        if ileft == len( xs ) - 1:
            ileft = len( xs ) - 2
            iright = len( xs ) - 1
        else:
            iright = ileft + 1
    k = float( ys[ ileft ] - ys[ iright ] ) / ( xs[ ileft ] - xs[ iright ] )
    b = ys[ ileft ] - k * xs[ ileft ]
    y = k * x + b
    return y

target_t_points = [ "8", "9.6", "12", "14.4", "16" ]
target_dr_points = [ "20", "30", "40", "50", "60", "70", "80", "90" ]
target_y = lstfeatures

def mk_y_by_dr( t, data ):
    datat = {}
    datat[ 'depth_rel' ] = []
    for i in lstfeatures:
        datat[ i ] = []
    # order by depth rel
    for i in range( len( data ) ):
```

```
        if data[ i ][ 0 ] == t:
            datat[ 'depth_rel' ].append( data[ i ][ 1 ] )
            for j in range( len( lstfeatures ) ):
                datat[ lstfeatures[ j ] ].append( data[ i ][ 2 + j ] )
    return datat

def interp_def( defect ):
    print
    print defect
    new_defect_data = []
    # get defect data
    datad = dbselect( [ 'thickness', 'depth_rel' ] + lstfeatures, defect )
    if datad == []:
        return []
    print "data %s" % datad
    # order by depth_rel
    datad.sort( lambda a, b : cmp( float( a[ 1 ] ), float( b[ 1 ] ) ) )
    # for each target t
    for t in target_t_points:
        # filter data leaving 1 t
        datat = mk_y_by_dr( t, datad )
        if datat[ 'depth_rel' ] == []:
            continue
        print "depth rel", data.floatize( datat[ 'depth_rel' ] )
        print "data", datat
        # for each target dr
        for dr in target_dr_points:
            # make features vector
            vec = [ t, dr ]
            print "t", t, "dr", dr
            # for each feature
            for f in lstfeatures:
                print "feature", f, data.floatize( datat[ f ] )
                # interpolate
                vec.append(                                          \
                    inter.inter(                                     \
                        data.floatize( datat[ 'depth_rel' ] ),       \
                        data.floatize( datat[ f ] ),                 \
```

```
                    float( dr )                                    \
                    )                                              \
                )
            new_defect_data.append( vec )
    return new_defect_data

def matrix2s( matrix ):
    res = ''
    for i in range( len( matrix ) ):
        for j in range( len( matrix[ i ] ) ):
            res = "%s%s\t" % ( res, matrix[ i ][ j ] )
        res = "%s\n" % res
    return res

def remap( keys, values ):
    res = []
    for i in range( len( keys ) ):
        res.append( "%s='%s'" % ( keys[ i ], values[ i ] ) )
    return res

def mkinsert( keys, vals, add ):
    return                                                         \
        'insert into ' + tblname + '_new set ' +                   \
        dbvalues( remap( keys, vals ) + add, '', '' )

def process_defect( defect ):
    data = interp_def( defect )
    for i in data:
        print mkinsert(                                            \
            [ 'thickness', 'depth_rel' ] + lstfeatures,            \
            i,                                                     \
            defect                                                 \
            )

def get_possible_vals( param, condition = [] ):
    raw = dbselect( [ param ], condition, 'distinct', '' )
    res = []
    for i in raw:
```

```
        res.append( i[ 0 ] )
    return res

def get_d_descrs():
    res = []
    dclasses = get_possible_vals( 'class' )
    lifoffs = get_possible_vals( 'liftoff' )
    fields = get_possible_vals( 'field' )
    fordclass in dclasses:
        cond = [ "class='%s'" % dclass ]
        widths = get_possible_vals( 'width', cond )
        lengths = get_possible_vals( 'length', cond )
        angles = get_possible_vals( 'angle', cond )
        for li in lifoffs:
            for f in fields:
                for w in widths:
                    for le in lengths:
                        for a in angles:
                            res.append(                                    \
                                [                                          \
                                "class='%s'" % dclass,                     \
                                "liftoff='%s'" % li,                       \
                                "field='%s'" % f,                          \
                                "width='%s'" % w,                          \
                                "length='%s'" % le,                        \
                                "angle='%s'" % a                           \
                                ]                                          \
                                )
    return res

for i in get_d_descrs():
    process_defect( i )
```