

Федеральное государственное бюджетное образовательное учреждение  
высшего образования «Новосибирский государственный технический  
университет»

На правах рукописи



Сивак Мария Алексеевна

**Робастное обучение нейронных сетей с простой  
архитектурой для решения задач классификации**

Специальность 05.13.17 —  
«Теоретические основы информатики»

Диссертация на соискание учёной степени  
кандидата технических наук

Научный руководитель:  
доктор технических наук, доцент  
Тимофеев Владимир Семенович

Новосибирск — 2022

## Оглавление

	Стр.
<b>Введение</b> . . . . .	4
<b>Глава 1 Основные понятия теории машинного обучения</b> . . . . .	11
1.1 Задачи и методы . . . . .	11
1.1.1 Основные задачи . . . . .	12
1.1.2 Классические методы . . . . .	13
1.1.2.1 Задача регрессионного анализа . . . . .	13
1.1.2.2 Задача кластеризации . . . . .	16
1.1.2.3 Задача классификации . . . . .	19
1.1.3 Робастный подход . . . . .	21
1.1.3.1 Метод наименьших модулей, $L_p$ -оценки . . . . .	23
1.1.3.2 $M$ -оценки и итеративный метод наименьших квадратов . . . . .	24
1.2 Определение и виды нейронных сетей . . . . .	27
1.3 Простая нейронная сеть для задачи классификации . . . . .	29
1.3.1 Модель нейронной сети . . . . .	29
1.3.2 Алгоритм обучения нейронной сети . . . . .	31
1.3.3 Оценка качества работы нейронной сети . . . . .	34
1.4 Особенности практического использования искусственных нейронных сетей . . . . .	37
1.5 Программные средства для построения нейронных сетей . . . . .	38
1.6 Обоснование цели и задач исследования . . . . .	40
<b>Глава 2 Построение робастных нейронных сетей</b> . . . . .	43
2.1 Исследование применимости робастных функций потерь в нейронных сетях . . . . .	43
2.2 Робастная модификация алгоритма обучения нейронной сети . . . . .	50
2.3 Исследование влияния планов эксперимента на точность работы робастной нейронной сети . . . . .	51
Выводы по главе 2 . . . . .	55
<b>Глава 3 Исследование устойчивости робастных нейронных сетей</b> . . . . .	56
3.1 Исследуемые данные и модели робастных нейронных сетей . . . . .	56

	Стр.	
3.2	Настройка робастных нейронных сетей . . . . .	58
3.3	Исследование устойчивости робастных искусственных нейронных сетей при различной доле засоряющих наблюдений . .	65
3.4	Исследование устойчивости робастных искусственных нейронных сетей при различном числе объектов . . . . .	69
	Выводы по главе 3 . . . . .	72
 <b>Глава 4 Программный модуль для построения робастных нейронных сетей . . . . .</b>		 <b>73</b>
4.1	Особенности реализации и системные требования . . . . .	73
4.2	Архитектура и основные методы . . . . .	75
4.3	Использование программного модуля «RobustNN» для работы в различных режимах . . . . .	79
	Выводы по главе 4 . . . . .	82
 <b>Глава 5 Практическое применение робастных нейронных сетей</b>		 <b>84</b>
5.1	Классификация нефтяных месторождений . . . . .	84
5.1.1	Постановка задачи . . . . .	84
5.1.2	Результаты классификации . . . . .	86
5.2	Определение местоположения проводника при коронарном стендировании . . . . .	88
5.2.1	Постановка задачи и анализируемые данные . . . . .	88
5.2.2	Результаты исследований . . . . .	89
	Выводы по главе 5 . . . . .	94
 <b>Заключение . . . . .</b>		 <b>95</b>
 <b>Список литературы . . . . .</b>		 <b>96</b>
 <b>Приложение А Листинги программного кода . . . . .</b>		 <b>106</b>
 <b>Приложение Б Свидетельства о регистрации программы для ЭВМ . . . . .</b>		 <b>108</b>
 <b>Приложение В Акты внедрения . . . . .</b>		 <b>110</b>

## Введение

### **Актуальность темы исследования и степень ее разработанности.**

В настоящее время исследователям в различных областях науки и техники приходится сталкиваться со все более сложными задачами. Нередки ситуации, когда у исследователя есть возможность получить достаточно большой объем данных, необходимых для решения той или иной практической задачи, но нет возможности программно реализовать эффективный алгоритм решения этой задачи в явном виде. В связи с этим большую популярность приобрела такая область информатики, как искусственный интеллект, и отдельный его раздел – машинное обучение. Особенность инструментов искусственного интеллекта заключается в том, что они работают за счет выделения статистических закономерностей в анализируемых данных, а не за счет выполнения непосредственных инструкций алгоритма.

Машинное обучение включает в себя множество инструментов, предназначенных для решения самого широкого круга задач, например для построения различных прогнозных моделей, разбиения данных на группы по определенному признаку, распознавания образов и др. Большой вклад в развитие этой области внесли такие известные зарубежные специалисты, как Самюэль А, Тьюринг А., Робертс Л., Dejong G. Нельзя недооценивать вклад в развитие этого направления, сделанный советскими и российскими учеными, авторами научных работ в области кибернетики и теории автоматического управления, такими как Китов А.И., Глушков В.Г., Ивахненко А.Г., Цыпкин Я.З.

Одним из наиболее известных инструментов машинного обучения сегодня являются искусственные нейронные сети (ИНС). Они используются повсеместно – от создания компьютерных игр и «умных» бытовых приборов до решения сложных наукоемких задач, требующих высокой точности получаемых результатов. Основоположниками в области нейронных сетей традиционно считаются Ф. Розенблатт, У. МакКаллок и У. Питтс. Дальнейшее развитие данная область получила благодаря Вербосу П.Дж., который считается автором алгоритма обратного распространения ошибки, использующегося при обучении нейронных сетей, а также Румельхарту Д.И. и Хинтону Дж.Е, которые популяризировали данный алгоритм. Здесь следует отметить также работы Галушкина А.И., Барцева С.И., Охонина В.А. – советских ученых, которые развивали идеи обучения нейронных сетей одновременно с западными специалистами и независимо

от них. В практическом плане большое значение имели работы Минского М.Л. и Сейновски Т., носящие прикладной характер. Кроме того, говоря о развитии и популяризации нейронных сетей, нельзя не упомянуть таких специалистов, как Хайкин С. и Бишоп К.М. – авторов одних из общепризнанных в настоящее время работ [57; 71], описывающих основы теории нейронных сетей.

Со временем стало ясно, что простые нейронные сети не всегда позволяют добиться высокой точности при решении задач, поскольку на практике в подавляющем большинстве случаев анализируемые данные не являются идеальными – так или иначе в них присутствуют ошибки и нетипичные наблюдения, которые не подчиняются каким-то общим закономерностям. В связи с этим появился ряд работ за авторством Хопфилда Д.Д., Коско Б.Э, Лекуна Я., Elman J.L., Jordan M.I., в которых описывались более сложные модели нейронных сетей: рекуррентные нейронные сети, сети с памятью, сверточные нейронные сети. Такое развитие данной области со временем привело к формированию отдельного подраздела в машинном обучении – глубокого обучения (термин введен Хинтоном Дж. в 2006 году). Очевидно, что усложнение архитектуры нейронной сети влечет за собой рост вычислительной сложности алгоритмов обучения и, как следствие, увеличение или временных затрат на построение модели, или материальных затрат на приобретение более мощного аппаратного обеспечения.

В то же время в области прикладного статистического анализа данных получил широкое развитие робастный подход, позволяющий снизить негативное влияние нетипичных наблюдений без существенного усложнения архитектуры используемой модели. Значительный вклад в развитие этого направления внесли такие ученые, как Хьюбер П., Хампель Ф., Rousseeuw P., Денисов В.И., Смоляк С.А., Титаренко Б.П. Наиболее широко робастный подход используется в рамках регрессионного анализа. Устойчивые методы хорошо зарекомендовали себя при анализе зашумленных данных [6; 13; 14; 43; 45; 92], поэтому представляется перспективным применить основные идеи робастного подхода при построении нейронных сетей. Кроме того, чтобы улучшить точность оценок, прогнозов и выводов, иногда в регрессионном анализе при сборе данных используют идеи теории планирования оптимального эксперимента. Представляется интересным исследовать эффекты от выбора различных планов эксперимента при подготовке данных для обучения нейронной сети.

**Цель и задачи исследования.** Целью данного исследования является разработка математического и алгоритмического обеспечения для построения робастных нейронных сетей, позволяющих корректно обрабатывать сильно зашумленные данные. Для достижения этой цели были поставлены и решены следующие задачи:

- исследовать применимость основных идей робастного подхода в области нейронных сетей;
- разработать, реализовать и исследовать алгоритм робастного обучения нейронных сетей с простой архитектурой;
- провести исследование устойчивости построенных нейронных сетей при анализе зашумленных данных и сформулировать рекомендации относительно настройки и использования робастных нейронных сетей, исследовать эффекты от использования различных планов эксперимента при обучении ИНС;
- разработать программный модуль, позволяющий строить робастные нейронные сети с произвольной простой архитектурой, и использовать его для решения задач прикладного характера.

**Область исследования.** Содержание диссертации соответствует п.5 области исследований «Разработка и исследование моделей и алгоритмов анализа данных, обнаружения закономерностей в данных и их извлечениях, разработка и исследование методов и алгоритмов анализа текста, устной речи и изображений» паспорта специальности 05.13.17 – «Теоретические основы информатики» (в области технических наук).

**Методы исследования.** Для решения поставленных задач использовались методы и положения математического анализа, прикладной математической статистики, регрессионного анализа, теории планирования эксперимента, а также методы статистического моделирования и методы оптимизации.

**Достоверность и обоснованность** научных положений, рекомендаций и выводов обеспечивается корректным использованием методов исследования, а также подтверждением полученных выводов результатами вычислительных экспериментов, проведенных с использованием технологии статистического моделирования.

**Научная новизна** работы заключается в следующем:

- предложен общий подход к построению робастных нейронных сетей с простой архитектурой на основе идеи алгоритма обратного распространения ошибки;

- сформулированы рекомендации относительно выбора значений внутренних параметров робастных функций потерь, позволяющие ускорить процесс настройки и обеспечить более высокую точность работы нейронных сетей;
- исследована устойчивость построенных робастных нейронных сетей при анализе зашумленных данных, впервые показано влияние качества плана эксперимента на точность работы робастной нейронной сети.

**Теоретическая значимость** работы заключается в развитии методов машинного обучения, а именно в предложенной модификации алгоритма обратного распространения ошибки, а также в исследовании применимости робастных функций потерь в нейронных сетях. Данная модификация отличается от классического алгоритма обратного распространения ошибки использованием робастных функций потерь вместо квадратичной, что позволяет снизить негативное влияние нетипичных наблюдений, не исключая их из рассмотрения.

**Практическая значимость** работы заключается в повышении точности классификации при работе с зашумленными данными, а также в сокращении времени обучения нейронной сети. Для обеспечения возможности использования полученных результатов на практике был разработан кроссплатформенный программный модуль «RobustNN», функционал которого позволяет выбрать функцию потерь, а также задать количество скрытых слоев и число нейронов на каждом слое. Данный программный модуль был зарегистрирован в виде объекта интеллектуальной собственности как программа для ЭВМ (№ гос. рег. 2021618329 от 26 мая 2021 г.) [33]. С использованием данного модуля решены две прикладные задачи технического характера: задача классификации нефтяных месторождений и задача определения положения проводника при коронарном стентировании.

**Положения, выносимые на защиту.** На защиту выносятся:

- модификация алгоритма обратного распространения ошибки, позволяющая построить и обучить робастную нейронную сеть с простой архитектурой;
- результаты исследования устойчивости построенных робастных нейронных сетей при анализе различных зашумленных данных и рекомендации по настройке параметров различных робастных нейронных сетей, результаты исследования точности работы робастных нейронных сетей при использовании различных планов эксперимента;
- программный модуль для построения робастных нейронных сетей с произвольной простой архитектурой «RobustNN»;

- результаты решения задачи классификации нефтяных месторождений.

**Личный творческий вклад автора** в совместных публикациях заключается в:

- исследовании применимости идей робастного подхода при построении нейронных сетей;
- разработке и реализации робастной модификации алгоритма обратного распространения ошибки;
- исследовании свойств построенных робастных нейронных сетей и формировании рекомендаций по их настройке;
- разработке программного модуля «RobustNN», позволяющего строить робастные нейронные сети с произвольной простой архитектурой, и использовании реализованных программных средств для решения прикладных задач.

**Апробация результатов диссертации.** Результаты работы докладывались на XIII Всероссийской научной конференции молодых ученых “Наука. Технология. Инновации” (НТИ-2019, г. Новосибирск, 2-6 декабря 2019 г.), XIV Всероссийской научной конференции молодых ученых “Наука. Технология. Инновации” (НТИ-2020, г. Новосибирск, 30 нояб.–4 дек. 2020 г.), XV Всероссийской научной конференции молодых ученых “Наука. Технология. Инновации” (НТИ-2021, г. Новосибирск, 6-10 декабря 2021 г.); на XIV международной научно-технической конференции «Актуальные проблемы электронного приборостроения» (АПЭП-2018, г. Новосибирск, 2-6 октября 2018 года); на XV международной научно-технической конференции «Актуальные проблемы электронного приборостроения» (АПЭП-2021, г. Новосибирск, 19-21 ноября 2021 года).

Разработанные методы и алгоритмы используются в работе Татарского научно-исследовательского и проектного института нефти публичного акционерного общества «Татнефть» имени В.Д. Шашина, а также в учебном процессе кафедры теоретической и прикладной информатики НГТУ, что подтверждено соответствующими актами о внедрении. Работа выполнена при финансовой поддержке РФФИ в рамках научного проекта № 20-37-90077, Министерства науки и высшего образования в рамках Госзадания (проект № FSUN-2020-0009), федеральной целевой программы «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2014-2020 годы» (проект № 075-15-2019-1853), а также в рамках выполнения научно-исследовательских работ по договору № 0750/2021/4150.



**Публикации.** По результатам диссертационного исследования опубликованы 12 печатных работ [33–41; 44; 85; 96], в том числе: 3 статьи в изданиях, рекомендованных ВАК, 2 статьи в изданиях, индексируемых в международных базах данных Scopus и/или Web of Science, 2 свидетельства о регистрации программы для ЭВМ.

**Объем и структура работы.** Диссертация состоит из введения, 5 глав основного содержания, заключения, списка литературы и 3 приложений. Полный объем диссертации составляет 111 страниц, включая 18 рисунков и 18 таблиц. Список литературы содержит 105 наименований.

**Краткое содержание работы.** В первой главе проводится анализ основных задач и методов машинного обучения – как классических, так и робастных, рассматриваются основные положения теории искусственных нейронных сетей и особенности их использования на практике. Кроме того, проводится анализ существующую программных средств для построения нейронных сетей. Приводится обоснование цели и задач исследования.

Вторая глава посвящена вопросам построения робастных нейронных сетей. В ней выполняется исследование применимости различных робастных функций потерь в нейронных сетях, предлагается робастная модификация алгоритма обратного распространения ошибки, а также проводится исследование точности работы робастной нейронной сети при различных планах эксперимента.

В третьей главе приводятся рекомендации по настройке различных робастных нейронных сетей, а также результаты исследования устойчивости построенных нейронных сетей при различной доле засоряющих наблюдений в данных, а также при различном числе объектов в наборе данных.

Четвертая глава содержит описание разработанного программного модуля для построения робастных нейронных сетей «RobustNN». Приводятся особенности реализации и системные требования, описание архитектуры и методов модуля, рассматриваются различные режимы работы, в которых можно использовать программный модуль.

В пятой главе приводятся результаты решения двух практических задач с использованием разработанного аппарата робастных нейронных сетей: задача классификации нефтяных месторождений и задача определения местоположения проводника в сосуде.

В приложении А приводятся листинги программного кода, иллюстрирующие использование модуля «RobustNN» при работе в различных режимах. В приложении Б представлены копии свидетельств о регистрации программы для ЭВМ. В приложении В представлены копии актов внедрения.

## Глава 1 Основные понятия теории машинного обучения

### 1.1 Задачи и методы

Машинное обучение – одна из наиболее быстро развивающихся сфер в области информационных технологий, методы которой имеют наиболее широкое применение. Как уже было отмечено во введении, термином «машинное обучение» обозначают раздел искусственного интеллекта, методы которого позволяют получить решение поставленной задачи не напрямую, а за счет решения множества схожих задач [95]. Модель машинного обучения – это результат работы алгоритма машинного обучения на каком-то наборе данных. Алгоритмы машинного обучения направлены на построение моделей с целью получения некоторого прогноза или решения именно на основе статистических данных и выделения закономерностей в них, а не за счет выполнения непосредственных инструкций. Часто методы машинного обучения относят к прогнозной аналитике или к прогнозному моделированию.

Иногда машинное обучение отождествляют с data mining – нередко в этих областях используются одни и те же методы, однако у них есть существенное различие: методы data mining направлены непосредственно на анализ уже имеющегося набора данных, а машинное обучение предполагает, что данные заранее не известны или известны не полностью. В основе машинного обучения лежат понятия, методы и алгоритмы математической оптимизации. Часто одним из этапов при решении задач машинного обучения является оптимизация некоторой функции потерь – функции, характеризующей потери при принятии решений на основе предъявляемых модели данных.

Подходы, основанные на выделении закономерностей в данных, находят свое применение на практике в тех случаях, когда данные имеют стохастический характер. Примерами таких задач могут служить фильтрация спама, оптическое распознавание символов (OCR), реализация поисковых движков, компьютерное зрение. В данном разделе рассматривается классификация типовых задач машинного обучения, а также наиболее известные алгоритмы, применяемые для их решения.

### 1.1.1 Основные задачи

Одна из наиболее известных классификаций задач машинного обучения предполагает их деление на типы в зависимости от способа реализации механизма обучения [93]. Обычно выделяют три больших класса задач: задачи обучения с учителем, задачи обучения без учителя и задачи обучения с подкреплением.

Обучение с учителем предполагает, что модели предъявляются образцы данных с известным ответом. В ходе обучения модель вырабатывает общее правило, сравнивая полученный ответ с истинным, а затем по этому правилу находит ответ для новых данных. При обучении без учителя ответ заранее не известен, поэтому модель обучается только за счет поиска закономерностей в данных и после обучения дает ответ на основании этих закономерностей. Обучение с подкреплением в какой-то мере похоже на обучение с учителем, однако в роли учителя здесь выступает внешняя среда, с которой взаимодействует модель путем выполнения каких-либо действий (например, управление транспортным средством). Принимая решение выполнить то или иное действие, модель получает сигнал подкрепления от внешней среды, на основании которого корректируется поведение модели.

Кроме того, задачи машинного обучения нередко делят на группы в зависимости от того, что является ожидаемым результатом работы модели [58]. Здесь выделяют, например, задачи классификации (распознавания образов), регрессионного анализа (идентификации регрессионных зависимостей), кластеризации, снижения размерности. При решении задачи классификации предполагается, что анализируемые данные разделены на классы, а построенная модель используется, чтобы определить класс для тех данных, которые ранее ей не предъявлялись. Как правило, такие задачи решаются в рамках обучения с учителем. Задачи регрессионного анализа также относятся к обучению с учителем и сводятся к установлению зависимости между откликом модели и входными факторами.

При кластеризации происходит разделение предъявляемых данных на группы, которые, в отличие от классификации, заранее не известны. Снижение размерности данных направлено на то, чтобы упростить структуру предъявляемых модели данных путем перевода их в пространство меньшей размерности. Эти две задачи относятся к обучению без учителя.

Несмотря на то что искусственные нейронные сети являются подходящим инструментом для решения рассмотренных задач машинного обучения, вместо них могут быть использованы и другие методы, многие из которых достаточно хорошо изучены. Остановимся в следующем подразделе на наиболее известных подходах к решению задач регрессионного анализа, кластеризации и классификации.

## 1.1.2 Классические методы

### 1.1.2.1 Задача регрессионного анализа

Задача регрессионного анализа заключается в построении различных моделей, описывающих функционирование многофакторных систем [12]. При этом система представляет собой так называемый «черный ящик» – она имеет входы и выходы, однако внутренний алгоритм ее работы полностью не известен.

Пусть имеется многофакторная система с набором входных факторов  $F_1, \dots, F_k$ . Предположим также, что истинная зависимость отклика  $Y$  такой системы от входных факторов может быть описана следующим линейно параметризованным регрессионным уравнением:

$$y = Z\theta + \varepsilon, \quad (1.1)$$

где  $Z = \begin{bmatrix} f_1(z_{11}) & \dots & f_k(z_{1k}) \\ \vdots & \ddots & \vdots \\ f_1(z_{N1}) & \dots & f_k(z_{Nk}) \end{bmatrix}$  – неслучайная матрица значений регрессионных функций, имеющая полный столбцовый ранг, равный числу параметров модели;  $\theta = (\theta_1, \dots, \theta_k)^T$  – вектор неизвестных параметров, значения которых необходимо оценить,  $k$  – число неизвестных параметров, значения которых необходимо оценить,  $N$  – количество проведенных экспериментов;  $f_i(z)$  – известные действительные функции вещественного аргумента  $z$ ,  $z_{ij}$  – детерминированные значения входных факторов  $F_1, \dots, F_k$ , полученные в течение  $N$  экспериментов;  $y = (y_1, \dots, y_N)^T$  – вектор значений отклика,  $\varepsilon = (\varepsilon_1, \dots, \varepsilon_N)^T$  – вектор случайных ошибок наблюдений, которые не зависят от значений  $z_{ij}$ .

Будем полагать, что ошибки наблюдений  $\varepsilon_i$  – независимые случайные величины, имеющие одинаковое распределение [6]. Для них верно следующее:

$$E(\varepsilon_i) = 0, D(\varepsilon_i) = \sigma^2 < \infty.$$

Решить задачу построения регрессионной модели означает как можно точнее оценить вектор неизвестных параметров  $\theta$  регрессии (1.1), используя имеющиеся значения отклика и входных факторов.

Наиболее известный метод, использующийся для оценки параметров регрессионного уравнения – это метод наименьших квадратов (МНК) [6; 21]. Основная идея этого метода заключается в минимизации суммы квадратов отклонений наблюдаемых значений отклика от предсказанных. Оптимизационная задача данного метода записывается следующим образом:

$$\min_{\theta} (y - Z\theta)^T (y - Z\theta). \quad (1.2)$$

У оптимизационной задачи (1.2) существует аналитическое решение, имеющее вид [17; 21; 32]

$$\hat{\theta} = (Z^T Z)^{-1} Z^T y. \quad (1.3)$$

Данный метод хорошо изучен и представлен во множестве работ, например в [6; 17; 21; 32]. МНК-оценки параметров (1.3) являются несмещенными, состоятельными и эффективными [6; 17; 21; 32]. При нормальности распределения ошибок регрессии (1.1) оценки, полученные по методу наименьших квадратов, совпадают с оценками максимального правдоподобия [21; 32] – это значит, что они являются наилучшими линейными оценками, а также асимптотически нормальными [6].

Иногда решаемая задача допускает выбор значений входных факторов. В таких случаях можно использовать теорию планирования оптимального эксперимента, что потенциально позволяет увеличить точность результатов. Основная идея этой теории заключается в построении такого плана эксперимента, при котором исследователь с наименьшими затратами на проведение эксперимента получает наибольший возможный в данных условиях объем информации об изучаемом объекте (явлении) [46].

Одним из базовых понятий данной теории является понятие эксперимента – комплекса математических, технических, технологических и социально-экономических мероприятий, направленных на получение сведений об изучаемом объекте. При этом выделяют активный и пассивный эксперимент.

При пассивном эксперименте предполагается, что факторы рассматриваются только в виде входных переменных с контролируруемыми значениями, а

исследователь находится в положении пассивного наблюдателя. Задача планирования эксперимента в этом случае сводится к оптимальной организации сбора информации и решению таких вопросов, как выбор количества и частоты измерений, выбор метода обработки результатов измерений.

При активном эксперименте предполагается, что факторы являются управляемыми и независимыми, а у исследователя есть возможность воздействия на ход процесса и выбора уровней факторов. При планировании активного эксперимента решается задача рационального выбора значений факторов, существенно влияющих на объект исследования, и определения соответствующего числа наблюдений. При этом нужно учитывать, что увеличение числа наблюдений влечет за собой рост вычислительных и временных затрат, а уменьшение может приводить к существенному увеличению погрешности.

Еще одним базовым понятием теории планирования оптимального эксперимента является понятие плана эксперимента. Планом эксперимента  $\xi$  называют совокупность величин  $x_1, x_2, \dots, x_n; r_1, r_2, \dots, r_n$  вида

$$\xi = \left\{ \begin{array}{cccc} x_1 & x_2 & \dots & x_n \\ r_1 & r_2 & \dots & r_n \end{array} \right\},$$

где  $\sum_{i=1}^n r_i = N$ ,  $x_i$  – значение фактора, при котором проводится  $r_i$  наблюдений,  $N$  – общее число наблюдений [10]. При этом точки  $x_1, x_2, \dots, x_n$ , соответствующие значениям факторов, обязательно различные – они образуют спектр плана эксперимента  $\xi_N$ .

Нормированным планом эксперимента называют совокупность величин вида

$$\varepsilon = \left\{ \begin{array}{cccc} x_1 & x_2 & \dots & x_n \\ p_1 & p_2 & \dots & p_n \end{array} \right\}.$$

При этом план эксперимента будет дискретным, если  $\sum_{i=1}^n p_i = 1, p_i = \frac{r_i}{N}$ , и непрерывным [10], если  $\sum_{i=1}^n p_i = 1, p_i \geq 0$ .

Чтобы перейти к рассмотрению критериев оптимальности плана, необходимо ввести еще два понятия: информационной и дисперсионной матрицы. Нормированной информационной матрицей дискретного или непрерывного плана называется величина

$$M = \sum_{j=1}^n p_j f(x_j) f^T(x_j),$$

где  $f(x) = (f_1(x), f_2(x), \dots, f_k(x))^T$  – вектор регрессионных функций модели (1.1). Дисперсионной матрицей называется матрица, обратная к информационной:

$$D(\varepsilon) = M^{-1}(\varepsilon).$$

Цель планирования эксперимента для регрессионных моделей заключается в том, чтобы в области возможных значений входных факторов  $\tilde{X}$ , часто называемой областью планирования, выбрать точки спектра  $x_i$  так, чтобы по результатам проведенных экспериментов оценить неизвестные параметры модели наилучшим образом (согласно некоторому критерию оптимальности). Качество плана  $\varepsilon$  оценивается на основании значения некоторого функционала от информационной или соответствующей ей дисперсионной матрицы.

План  $\varepsilon^*$  называют  $D$ -оптимальным, если соответствующая ему информационная матрица имеет наибольшее возможное значение определителя (или соответствующая дисперсионная матрица имеет наименьшее возможное значение определителя):

$$\varepsilon^* = \mathop{\text{Arg max}}_{\varepsilon} |M(\varepsilon)| \quad \text{или} \quad \varepsilon^* = \mathop{\text{Arg min}}_{\varepsilon} |D(\varepsilon)|.$$

План  $\varepsilon^*$  называют  $A$ -оптимальным, если соответствующая ему дисперсионная матрица имеет наименьший след:

$$\varepsilon^* = \mathop{\text{Arg min}}_{\varepsilon} \text{tr}[D(\varepsilon)].$$

План  $\varepsilon^*$  называют  $Q$ -оптимальным, если

$$\varepsilon^* = \mathop{\text{Arg min}}_{\varepsilon} \int_Z d(x, \varepsilon) dx,$$

где область  $Z$  может не совпадать с областью  $\tilde{X}$ , а  $d(x, \varepsilon) = f^T(x)M^{-1}(\varepsilon)f(x)$  – дисперсия оценки функции отклика.

### 1.1.2.2 Задача кластеризации

Перейдем к постановке задачи кластеризации. Пусть имеется множество объектов  $X = \{X_1, \dots, X_{|X|}\}$ , каждый из которых описывается вектором признаков этого объекта  $x_p = \{x_{p1}, x_{p2}, \dots, x_{pK}\}$ , где  $x_{pi}, i = 1, \dots, K$  – значения признаков, а  $K$  – количество признаков. Кроме того, пусть имеется конечное множество номеров кластеров  $Y = \{Y_1, \dots, Y_k\}$ , где  $Y_i$  – номер кластера,  $k$  –



количество кластеров. Кроме того, пусть на множестве  $X$  задана функция расстояния между объектами следующим образом:

$$\begin{aligned} \bar{d} : X \times X &\rightarrow \mathbb{R}_+ \cup \{0\}, \bar{d}(X_i, X_j) = \bar{d}(X_j, X_i) \quad \forall i \neq j, \\ \bar{d}(X_i, X_i) &= 0 \quad \forall X_i \in X. \end{aligned} \quad (1.4)$$

Решить задачу кластеризации означает разделить объекты данной выборки на подмножества (кластеры), то есть каждому объекту  $X_i \in X$  поставить в соответствие номер кластера  $Y_j \in Y$  таким образом, чтобы значение функции  $\bar{d}(X_i, X_j)$  для объектов из одного кластера было относительно мало, а для объектов из разных кластеров – относительно велико.

Как правило, результатом работы алгоритма кластеризации является набор подмножеств  $C = (C_1, \dots, C_k)$  множества  $X$  (кластеров). Для этого набора верно следующее:  $\cup_{i=1}^k C_i = X, C_i \cap C_j = \emptyset, \forall i \neq j$ . Существует несколько подходов к выполнению кластеризации – наиболее известными среди них являются иерархическая и статистическая кластеризация [18; 23; 26]. Рассмотрим их подробнее.

Иерархическая кластеризация или кластеризация на основе связности – вероятно, один из наиболее простых и прямолинейных подходов. В основе этого подхода лежит идея аггломерации [18]. Изначально все объекты рассматриваются как самостоятельные кластеры. Затем на каждой итерации алгоритма выполняется объединение наиболее близко расположенных кластеров, полученных на предыдущем этапе. С каждой итерацией алгоритма число кластеров уменьшается. Работа алгоритма продолжается до тех пор, пока не сработает критерий останова алгоритма или все объекты не будут объединены в один общий кластер. Если критерий останова не задан, то результатом работы алгоритма будет являться не набор кластеров  $C$ , а дендрограмма, показывающая взаимные связи между объектами из множества  $X$ , – дерево, листьями которого являются одноэлементные подмножества множества  $X$ , а корнем – само это множество [18].

Критерий останова обычно определяется каким-либо из следующих способов. Либо задается фиксированное число кластеров  $k$  – в этом случае алгоритм заканчивает свою работу, когда получено столько кластеров, сколько указано. Либо определяется верхняя граница расстояния между кластерами  $r \in \mathbb{R}_+$ . В этом случае алгоритм заканчивает свою работу, как только расстояние меж-

ду кластерами становится больше  $r$ . Можно также задавать масштабируемую верхнюю границу расстояния:  $r = \alpha \max_{i,j} \{\bar{d}(X_i, X_j) : X_i, X_j \in X\}$ ,  $\alpha < 1$ .

Кроме того, для работы алгоритма необходимо определить функцию расстояния между кластерами  $D(C_i, C_j)$ . Чаще всего используется один из следующих подходов:

1. Метод одиночной связи – расстояние между кластерами определяется как наименьшее расстояние между объектами этих кластеров:

$$D(C_i, C_j) = \min_{p,t} \{\bar{d}(X_p, X_t) : X_p \in C_i, X_t \in C_j\}.$$

2. Метод средней связи – расстояние между двумя кластерами определяется как среднее расстояние между объектами этих кластеров:

$$D(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{X_p \in C_i, X_t \in C_j} \bar{d}(X_p, X_t).$$

3. Метод полной связи – расстояние между кластерами определяется как наибольшее расстояние между объектами этих кластеров:

$$D(C_i, C_j) = \max_{p,t} \{\bar{d}(X_p, X_t) : X_p \in C_i, X_t \in C_j\}.$$

Другим распространенным подходом к решению данной задачи является статистическая кластеризация. Наиболее известным алгоритмом, реализующим такой подход, является алгоритм  $k$ -средних [95]. В рамках данного алгоритма каждому кластеру  $C_i$  ставится в соответствие центроид  $\mu_i$ , описываемый вектором  $\tilde{\mu}_i = \{\mu_{i1}, \mu_{i2}, \dots, \mu_{iK}\}$ . Начальные значения компонент этого вектора задаются случайным образом. В основе работы данного алгоритма лежит решение задачи оптимизации – минимизации суммы квадратов расстояний от объектов кластеров до центроидов этих кластеров:

$$\min_C \sum_{i=1}^k \sum_{X_p \in C_i} \bar{d}(X_p, \mu_i)^2.$$

Алгоритм  $k$ -средних является итерационным и заключается в поочередном выполнении двух шагов. На первом шаге каждой итерации  $t$  для каждого объекта  $X_p$  определяется кластер  $C_i^{(t)}$  таким образом, чтобы расстояние от этого объекта до центроида  $\mu_j^{(t)}$  данного кластера на текущей итерации было минимальным:

$$C_i^{(t)} = \{X_p : \bar{d}(X_p, \mu_i^{(t)})^2 \leq \bar{d}(X_p, \mu_j^{(t)})^2, 1 \leq j \leq k\},$$

причем для каждого объекта  $X_p$  выбирается только один кластер  $C_i^{(t)}$ , даже если этот объект можно включить в большее число кластеров.

На втором шаге алгоритма выполняется перерасчет компонент векторов центроидов каждого кластера:

$$\tilde{\mu}_i^{(t+1)} = \frac{1}{|C_i^{(t)}|} \sum_{X_p \in C_i^{(t)}} x_p.$$

Работа алгоритма прекращается в тот момент, когда кластеры объектов перестают изменяться. Как правило, в алгоритме используется евклидова метрика расстояния – использование других метрик может негативным образом сказаться на сходимости алгоритма [29]. Основной недостаток этого алгоритма заключается в том, что необходимо заранее знать число кластеров  $k$ .

### 1.1.2.3 Задача классификации

В отличие от задачи кластеризации при решении задачи классификации для обучения модели используются данные с уже известным ответом [58]. Это позволяет сравнить ответ, полученный с использованием модели, с истинным результатом и на основании этого скорректировать работу алгоритма. Рассмотрим постановку этой задачи.

Пусть имеется конечное множество классов  $Q = \{q_1, \dots, q_{|Q|}\}$ , где  $q_k$  – непересекающиеся между собой классы, а также конечное множество объектов  $X = \{X_1, \dots, X_{|X|}\}$ . При этом каждый объект  $X_m, m = 1, \dots, |X|$  можно описать вектором  $x_m = \{x_{m1}, x_{m2}, \dots, x_{mY}\}$ , состоящим из значений признаков этого объекта  $x_{mi}, i = 1, \dots, Y$ , где  $Y$  – количество признаков. Классифицировать объект значит указать для каждого объекта  $X_m$  класс  $q_k$ , к которому этот объект относится. В зависимости от количества классов выделяют задачи бинарной классификации (два класса) и задачи множественной классификации или мультиклассификации (более двух классов) [58].

При решении задачи классификации, как правило, всё множество объектов  $X$  разделяется на обучающую и тестовую подвыборки, которые представляют собой два непересекающихся подмножества [58; 94]. Обучающая выборка  $L = \{X_1, \dots, X_{|L|}\}$  представляет собой набор объектов, с помощью которого происходит обучение алгоритма классификации. Тестовая выборка  $D = \{X_{|L|+1}, \dots, X_{|X|}\}$  представляет собой набор объектов, на котором производится оценка точности работы алгоритма классификации.

Одними из простейших в реализации являются метрические алгоритмы классификации, основная идея которых является в оценке сходства объектов друг с другом [4; 30]. Как и метод  $k$ -средних, они требуют определения функции расстояния между объектами множества  $X$  аналогично (1.4). Наиболее известный алгоритм из этого семейства – метод ближайших соседей.

Существует несколько разновидностей данного алгоритма. В самом простом варианте рассматривается один ближайший сосед – классифицируемый объект  $X_u$  относится к тому классу  $q_k$ , которому принадлежит ближайший к нему объект обучающей выборки  $X_i$ . Более сложный вариант алгоритма предполагает рассмотрение  $k$  ближайших объектов обучающей выборки. Очевидный недостаток такого подхода заключается в необходимости задания числа соседей  $k$ . Здесь может возникнуть неоднозначная ситуация – когда одинаковое число соседей объекта  $X_u$  принадлежит к разным классам. В случае бинарной классификации такая проблема решается выбором нечетного количества соседей объекта. В случае мультиклассификации обычно используют метод взвешенных ближайших соседей, который предполагает, что в зависимости от расстояния до классифицируемого объекта каждому соседу приписывается вес  $w_i$  (чем дальше расположен сосед, тем меньше его вес). Рассматриваемый объект  $X_u$  относят к тому классу, для которого суммарный вес среди  $k$  соседей максимален. Рассмотрим метод ближайших соседей в общем виде.

Введем дополнительно функцию отклика  $y(X_i)$ , которая ставит в соответствие объекту обучающей выборки  $X_i$  его класс:

$$y(X_i) = q_p.$$

Оптимизационную задачу алгоритма ближайших соседей можно записать следующим образом:

$$\max_{q_p \in Q} \sum_{i=1}^m I \left[ y(X_i^{(u)}) = q_p \right] w(i, u),$$

где  $X_i^{(u)}$  –  $i$ -й сосед классифицируемого объекта  $X_u$ ,  $w(i, u)$  – весовая функция, оценивающая важность  $i$ -го соседа для объекта  $X_u$ . При этом для разных вариантов алгоритма весовая функция вводится следующим образом:  $w(1, u) = 1$  – для метода ближайшего соседа,  $w(i, u) = 1, i \leq k$  – для метода  $k$  ближайших соседей,  $w(i, u) = \alpha^i, i \leq k, 0,5 \leq \alpha \leq 1$  – для метода взвешенных ближайших соседей.

Наиболее очевидным методом поиска ближайших соседей является инкрементный поиск. Упорядочим для классифицируемого объекта  $X_u$  объекты обучающей выборки  $X_i$  в порядке возрастания расстояния от них до  $X_u$ :

$$\bar{d}(X_u, X_1^{(u)}) \leq \bar{d}(X_u, X_2^{(u)}) \leq \dots \leq \bar{d}(X_u, X_{|L|}^{(u)}).$$

Таким образом для каждого классифицируемого объекта  $X_u$  получим новый порядок следования объектов обучающей выборки, которая отражает важность соседей этого объекта. На основе этого порядка отбираются  $k$  объектов, расположенных ближе всего к объекту  $X_u$ . На практике инкрементный поиск рассматривается как отдельная самостоятельная задача. Методы решения такой задачи рассматриваются в [29].

Помимо метрических классификаторов существует множество альтернативных подходов к решению данной задачи, например, наивный байесовский классификатор, метод опорных векторов, деревья решений. Данные методы подробно рассматриваются в [4; 5; 7; 65].

### 1.1.3 Робастный подход

Методы решения задач машинного обучения, рассмотренные в предыдущем разделе, позволяют добиться высокой точности работы только в тех случаях, когда анализируемые данные являются достаточно «хорошими». На практике же исследователи часто сталкиваются с тем, что данные таковыми не являются – они могут содержать в себе ряд сильно отличающихся от остальных, нетипичных наблюдений (выбросов) или же быть сильно перемешанными между собой, что имеет важное значение при решении задач классификации и кластеризации. Это происходит из-за того, что данные, как правило, являются результатом реальных измерений. Выбросы могут возникать в результате грубых ошибок при проведении измерений или же их наличие может быть обусловлено природой данных. Излишнее смешивание данных может быть вызвано, в том числе, небрежным округлением измерений. Такие особенности анализируемых данных могут негативным образом сказаться на точности выводов при использовании классических методов.

Одним из вариантов решения этой проблемы является предобработка исследуемых данных и исключение нетипичных наблюдений. Однако применение стандартных методов отбраковки, рассмотренных в [27], может привести к по-

тере важной информации. Это может быть продемонстрировано при помощи следующего простого примера [92].

Пусть параметры линейной регрессии оцениваются по пяти наблюдениям, которым на рис. 1.1 соответствуют точки серого цвета. Проходящая через точку  $A$  прямая линия соответствует истинной зависимости.

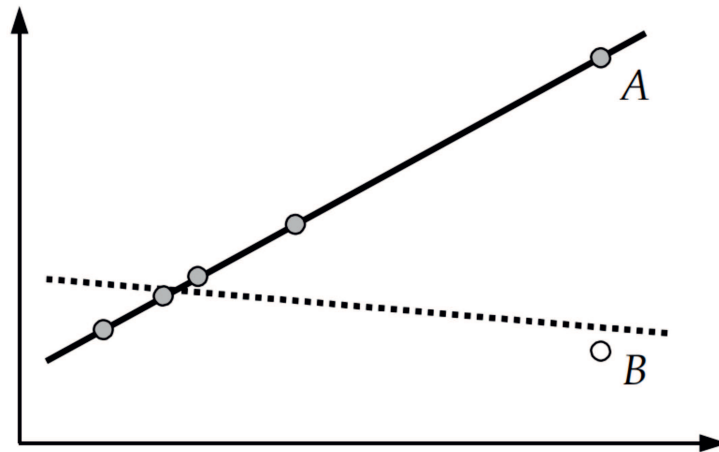


Рисунок 1.1 — Влияние выброса на оценивание параметров линейной регрессии

Предположим, что при сборе исследуемых данных была допущена ошибка, в результате чего вместо значения входной переменной, соответствующего точке  $A$ , было зафиксировано значение отклика, соответствующее точке  $B$ . При оценивании параметров регрессии по экспериментальным данным с использованием метода наименьших квадратов будет получена прямая, обозначенная на рисунке пунктирной линией. Расположение этой прямой значительно отличается от истинной зависимости.

Нетрудно заметить, что использование методов отбраковки позволит корректным образом построить зависимость во втором случае (когда в рассмотрение включена точка  $B$ ). В первом же случае точка  $A$ , находящаяся на достаточно большом расстоянии от основной группы наблюдений, будет исключена из рассмотрения. Однако она не является выбросом на самом деле, и ее исключение приведет к потере важной с точки зрения оценивания информации.

Из приведенного на рис. 1.1 примера можно заметить, что наличие даже одного выброса в исследуемых данных приводит к резкому увеличению суммы квадратов остатков, что плохо скажется при оценивании параметров с помощью метода наименьших квадратов. В связи с этим применяют альтернативный подход к решению данной проблемы – использование так называемых робастных

или устойчивых методов, которые позволят не исключать нетипичные наблюдения, а снизить их негативное влияние на работу модели [73; 91]. Наиболее часто применение такого подхода можно встретить как раз при решении задач регрессионного анализа. Рассмотрим наиболее известные устойчивые методы оценивания параметров линейной регрессии.

### 1.1.3.1 Метод наименьших модулей, $L_p$ -оценки

Метод наименьших модулей является широко известной альтернативой методу наименьших квадратов [25]. В рамках этого метода вместо квадратов остатков используются их абсолютные величины:

$$\hat{\theta} = \mathit{Arg} \min_{\theta} \sum_{i=1}^N |y_i - z_i \theta|, \quad (1.5)$$

где  $z_i$  –  $i$ -я строка матрицы  $Z$ .

Оценки (1.5) называются МНМ-оценками (оценками метода наименьших модулей) [25]. Одним из недостатков этого метода является то, что, в отличие от метода наименьших квадратов, для решения оптимизационной задачи (1.5) приходится использовать методы линейного программирования [8]. Это связано с тем, что производные от функции модуля будут иметь разрывы второго рода [20], следовательно, поставленную задачу нельзя решать аналитическими методами.

МНМ-оценки, в отличие от МНК-оценок, не так чувствительны к наличию нетипичных наблюдений в исследуемых данных – это вполне очевидно, так как при использовании абсолютных величин не происходит столь резкого роста суммы остатков, как при использовании квадратов. При обобщении МНМ- и МНК-оценок получаются так называемые  $L_p$ -оценки [11]. Для получения таких оценок требуется решить задачу оптимизации следующего вида:

$$\hat{\theta} = \mathit{Arg} \min_{\theta} \sum_{i=1}^N |y_i - z_i \theta|^p,$$

где  $p \in \mathbb{Z}^+$ .

Метод наименьших модулей называют  $L_1$ -методом, тогда как метод наименьших квадратов –  $L_2$ -методом (соответственно, МНМ-оценки –  $L_1$ -оценки, а МНК-оценки –  $L_2$ -оценки). Случай, когда  $p = \infty$ , соответствует минимаксным оценкам. Сами по себе  $L_p$  оценки являются частным случаем  $M$ -оценок, которые будут рассмотрены далее.

### 1.1.3.2 $M$ -оценки и итеративный метод наименьших квадратов

$M$ -оценки были предложены швейцарским математиком П. Хьюбером [73]. Такие оценки предполагают использование специальных функций от остатков  $\rho(z)$ , также называемых функциями потерь, которые часто (хоть и не всегда) при малых значениях аргумента являются квадратичными, а при больших значениях – «менее возрастающими», чем квадратичная:

$$\hat{\theta} = \mathit{Arg} \min_{\theta} \sum_{i=1}^N \rho(y_i - z_i \theta). \quad (1.6)$$

Кроме того,  $M$ -оценки могут быть получены при решении следующей системы уравнений, если взять производную суммарной функции потерь по параметрам:

$$\sum_{i=1}^N \psi(e_i) f_j(z_{ij}) = 0, j = 1, \dots, k, \quad (1.7)$$

где  $e_i = y_i - z_i \hat{\theta}$  – остатки регрессии,  $\psi(e_i) = \rho'(e_i)$  – производная по параметрам. При использовании в качестве  $\rho(z)$  выпуклой функции этот подход эквивалентен решению задачи (1.6).

Для поиска  $M$ -оценок может быть использован итеративный метод наименьших квадратов (ИМНК), построенный на основе обобщенного (взвешенного) МНК [11; 62]. Рассмотрим данный метод подробнее.

Запишем левую часть системы уравнений (1.7) следующим образом:

$$\sum_{i=1}^N \frac{\psi(e_i)}{e_i} f_j(z_{ij}) e_i = \sum_{i=1}^N \omega(e_i) f_j(z_{ij}) e_i, j = 1, \dots, k, \quad (1.8)$$

где  $\omega(e_i) = \frac{\psi(e_i)}{e_i}$ .

Переходя к матричным обозначениям, с учетом (1.8) систему уравнений (1.7) можно переписать следующим образом:

$$Z^T W(\theta)(y - Z\theta) = 0, \quad (1.9)$$

где  $W(\theta)$  – диагональная матрица размерности  $N$ ,  $W(\theta)_{ii} = \omega(e_i)$ . Оценка параметров в этом случае будет иметь следующий вид:

$$\hat{\theta} = [Z^T W(\hat{\theta}) Z]^{-1} Z^T W(\hat{\theta}) y.$$



Система уравнений (1.9) соответствует схеме обобщенного МНК, где веса зависят от параметров  $\theta$ . Для поиска решения исходной задачи может быть использована следующая итерационная схема:

$$[Z^T W(\hat{\theta}^{(t-1)}) Z] \hat{\theta}^{(t)} = Z^T W(\hat{\theta}^{(t-1)}) y,$$

где  $t$  – номер итерации,  $\hat{\theta}^{(t)}$  – оценки параметров регрессии, полученные на итерации  $t$ :

$$\hat{\theta}^{(t)} = [Z^T W(\hat{\theta}^{(t-1)}) Z]^{-1} Z^T W(\hat{\theta}^{(t-1)}) y.$$

Важно также обратить внимание на случай, когда функция  $\rho(z)$  невыпуклая. При использовании такой функции может существовать несколько локальных минимумов, каждый из которых будет соответствовать решению системы (1.7) – при таких условиях этот подход не будет эквивалентен оценке (1.6). Кроме того, необходимо корректно выбрать начальное приближение оценок параметров  $\hat{\theta}^{(0)}$  – если оно будет расположено далеко от глобального экстремума функции, то в результате работы алгоритма могут получиться оценки, соответствующие локальному минимуму и не являющиеся устойчивыми.

Для получения устойчивых оценок в качестве  $\rho(z)$  можно использовать одну из следующих робастных функций потерь, рассмотренных в [2; 3; 6; 17; 54; 59; 60; 67; 68; 103] (здесь  $\alpha, \beta, \gamma$  – параметры функций потерь):

1. Функция потерь Хьюбера:

$$\rho(z) = \begin{cases} \frac{1}{2} z^2, & |z| \leq \beta, \\ \beta |z| - \frac{1}{2} \beta^2, & |z| > \beta. \end{cases}$$

2. Функция потерь Эндрюса:

$$\rho(z) = \begin{cases} \beta (1 - \cos \frac{z}{\beta}), & |z| < \pi \beta, \\ 2\beta, & |z| \geq \pi \beta. \end{cases}$$

3. Функция потерь Хампеля:

$$\rho(z) = \begin{cases} z^2, & 0 \leq |z| \leq \eta, \\ \eta^2 + \eta(|z| - \eta), & \eta < |z| \leq \beta, \\ \eta^2 + 2\eta(\beta - \eta) - \frac{\eta(\gamma - |z|)^2}{\gamma - \beta} + \eta(\gamma - \beta), & \beta < |z| \leq \gamma, \\ \eta^2 + 2\eta(\beta - \eta) + \eta(\gamma - \beta), & \gamma < |z|. \end{cases}$$

4. Функция потерь Тьюки:

$$\rho(z) = \begin{cases} z^2, & |z| \leq \beta, \\ \beta^2, & |z| > \beta. \end{cases}$$

5. Биквадратная функция потерь Тьюки:

$$\rho(z) = \begin{cases} \frac{z^6}{6\beta^4} - \frac{z^4}{2\beta^2} + \frac{z^2}{2}, & |z| < \beta, \\ \frac{\beta^2}{6}, & |z| \geq \beta. \end{cases}$$

6. Функция потерь Рамсея:

$$\rho(z) = \frac{1 - (1 + \beta |z|) \exp(-\beta |z|)}{\beta^2}.$$

7. Функция потерь Коши (Лоренца):

$$\rho(z) = \ln\left(\frac{1}{2}\left(\frac{z}{\beta}\right)^2 + 1\right).$$

8. Функция потерь Geman-McCluer:

$$\rho(z) = \frac{z^2/\beta}{1 + z^2/\beta}.$$

9. Функция потерь «Fair»:

$$\rho(z) = \beta^2 \left( \frac{|z|}{\beta} - \ln \left( 1 + \frac{|z|}{\beta} \right) \right).$$

10. Функция потерь Charbonnier:

$$\rho(z) = \sqrt{\left(\frac{z}{\beta}\right)^2 + 1}.$$

11. Функция потерь Уэлша:

$$\rho(z) = 1 - \exp\left(-\frac{1}{2}\left(\frac{z}{\beta}\right)^2\right).$$

12. Функция потерь Мешалкина:

$$\rho(z) = \beta^{-1} \left( 1 - \exp\left(\frac{-\beta z^2}{2}\right) \right).$$

Хотя методы, рассмотренные в пунктах 1.1.2 и 1.1.3, были разработаны специально для решения конкретных видов задач машинного обучения, в настоящее время исследователи все чаще отдают предпочтение использованию искусственных нейронных сетей, представляющих собой в каком-то роде обобщенный инструмент. Рассмотрим основные понятия теории нейронных сетей подробнее.

## 1.2 Определение и виды нейронных сетей

Искусственная нейронная сеть представляет собой структуру, которая состоит из простых элементов, называемых нейронами. Нейроны связаны друг с другом посредством однонаправленных каналов (ребер). У ребер нейронной сети есть весовые коэффициенты, за счет корректировки которых и происходит обучение модели [57; 71]. Как правило, все элементы нейронной сети расположены послойно. Простейшая сеть – однослойный перцептрон – включает в себя только два слоя: входной и выходной. Более сложная сеть – многослойный перцептрон – включает в себя один или несколько скрытых слоев, что позволяет находить более сложные закономерности в анализируемых данных. Схема многослойного перцептрона в наиболее общем виде представлена на рис. 1.2.

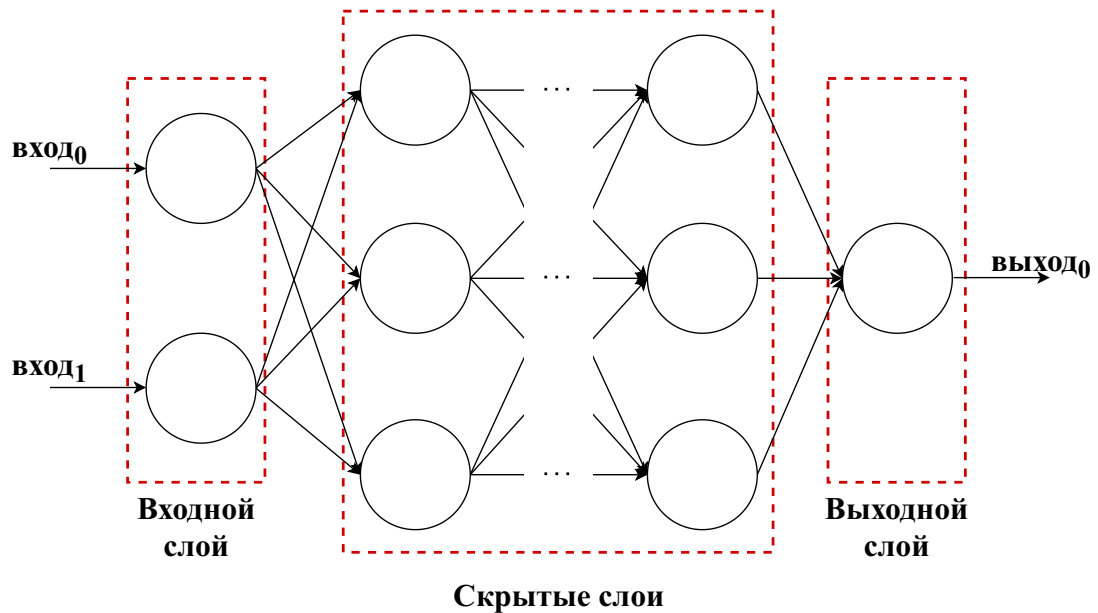


Рисунок 1.2 — Многослойный перцептрон в общем виде

Первые исследования в области искусственных нейронных сетей датируются концом XIX – началом XX века. Они представляют собой работы на стыке нескольких дисциплин (физики, психологии и нейрофизиологии), выполненные рядом ученых, таких как, например, Эрнст Мах [80] и Иван Павлов [28]. В их работах рассматривались общие положения теории обучения, но не были представлены конкретные математические модели функционирования нейронов.

Искусственные нейронные сети в современном понимании начали рассматривать в 1940-х годах Уоррен Мак-Каллок и Уолтер Питтс [82]. Они показали, что сети из искусственных нейронов в принципе могут быть использованы для

вычисления любых арифметических или логических функций. Часто именно их работа считается основополагающей в области искусственных нейронных сетей.

Постепенно данная область получила серьезное развитие, а нейросетевые модели нашли свое применение для решения множества практических задач: оптическое распознавание символов на изображениях, анализ текстов, синтез речи, построение различных прогнозов и др. По мере усложнения природы анализируемых данных появлялись и более сложные нейросетевые модели.

Существует несколько критериев классификации нейронных сетей. Самые распространенные из них – по виду модели, по способу обучения и по типу решаемых задач.

По виду модели нейронные сети делятся на сети прямого распространения (перцептрон, сверточные сети), рекуррентные сети (например, LSTM модели – сети с долговременной и краткосрочной памятью), сети с радиально-базисными функциями и нейронные сети (карты) Кохонена. С точки зрения способа обучения для нейронных сетей применяется такая же классификация, как и для задач машинного обучения, рассмотренная в п. 1.1.1. Наиболее распространенными являются модели, обучаемые с учителем, например, тот же самый многослойный перцептрон или сверточные сети. К сетям, обучаемым без учителя, традиционно относят карты Кохонена [24].

С помощью нейронных сетей обычно решаются следующие виды задач: оценка значения параметров и прогнозирование, распознавание образов (классификация), отбор информативных признаков, кластеризация, оптимизация. Для прогнозирования и оценки параметров обычно используются перцептрон (как однослойный, так и многослойный) и радиально-базисные сети, для кластеризации – сети Кохонена, для оптимизации – ИНС Хопфилда [78] и машина Больцмана [48]. Для отбора информативных признаков – перцептрон. Кроме того, все перечисленные выше модели могут быть использованы для решения задачи распознавания образов.

Будем называть модель нейронной сети типа перцептрон простой нейронной сетью или нейронной сетью с простой архитектурой. Одним из наиболее часто используемых вариантов этой архитектуры является многослойный перцептрон с одним скрытым слоем. Перейдем к подробному рассмотрению модели с такой архитектурой для решения задачи классификации.

## 1.3 Простая нейронная сеть для задачи классификации

### 1.3.1 Модель нейронной сети

Искусственные нейронные сети могут использоваться для решения задач как бинарной, так и множественной классификации. Часто при решении задач бинарной классификации строят нейросетевую модель с одним выходом, значение которого есть вероятность принадлежности объекта к определенному классу. При этом устанавливают порог классификации – значение  $P$ , относительно которого определяется, к какому именно классу объект принадлежит:  $X_m \in q_1$ , если  $y_1 < P$ ,  $X_m \in q_2$ , если  $y_1 \geq P$ . Часто выбирают пороговое значение  $P = 0,5$ .

Теперь рассмотрим подробнее пример задачи множественной классификации, в которой каждый объект  $X_m$  имеет 4 признака ( $Y = 4$ ), а множество классов  $Q$  включает в себя 3 класса. Для решения этой задачи рассмотрим нейронную сеть с простой архитектурой, представленную на рис.1.3.

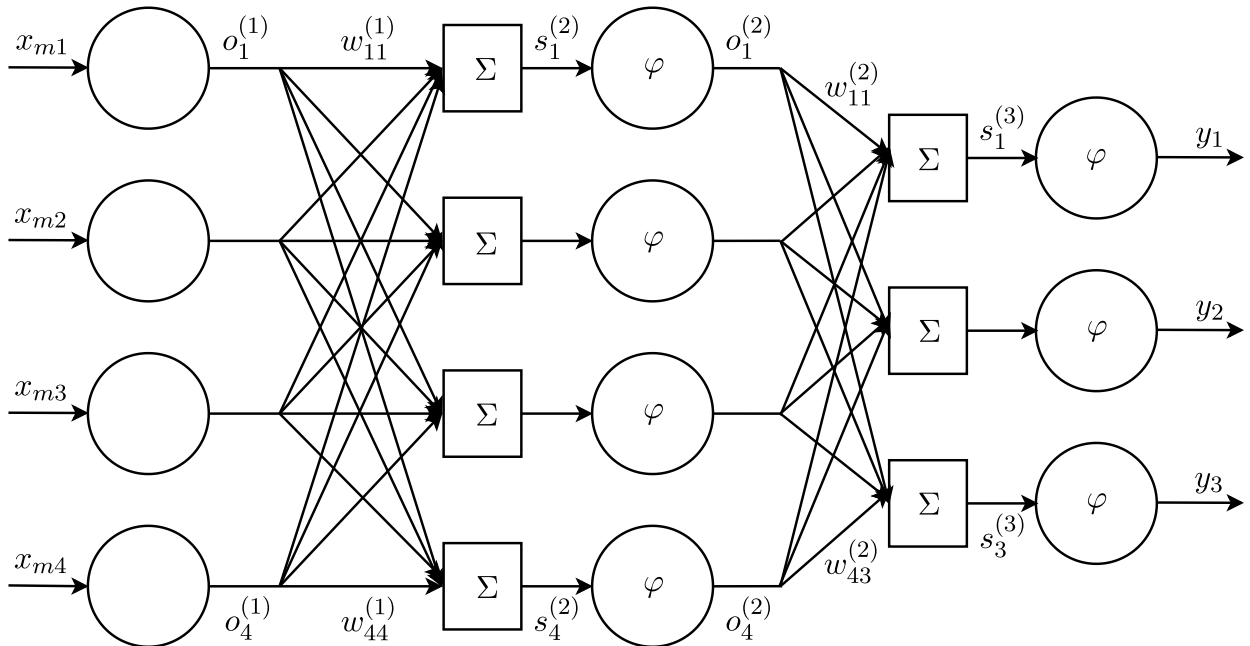


Рисунок 1.3 — Искусственная нейронная сеть с одним скрытым слоем

Данная сеть состоит из трёх слоев ( $N = 3$ ). Входной слой сети, на который подаются признаки объекта  $X_m$ , включает в себя четыре нейрона, выходной – три (по одному для каждого класса  $q_k$ ). Скрытый слой рассматриваемой сети состоит из четырёх нейронов. В качестве функции активации  $\varphi = \varphi(z)$  желательно использовать функцию, обладающую следующими свойствами:

1.  $\varphi(\alpha x + \beta y) \neq \alpha \varphi(x) + \beta \varphi(y)$ .

2.  $\varphi \in C^1(\mathbb{R})$ .
3.  $\varphi(x) \geq \varphi(y)$  при  $x \geq y$ .
4.  $\varphi(x) \sim x$  при  $x \rightarrow 0$ .

На рис.1.3 через  $y_1, y_2, y_3$  обозначены значения нейронов на выходном слое сети. Номер нейрона  $k$ , имеющего наибольшее значение, будет соответствовать номеру класса  $q_k$  для объекта, поданного нейронной сети на вход. Веса на ребрах между нейронами первого и второго слоя обозначены  $w_{ij}^{(1)}, i, j = 1, \dots, 4$ ; веса на ребрах между нейронами второго и третьего слоя –  $w_{jk}^{(2)}, j = 1, \dots, 4, k = 1, \dots, 3$ . Входные значения нейронов на втором слое обозначаются как  $s_j^{(2)}, j = 1, \dots, 4$ , входные значения на третьем слое –  $s_k^{(3)}, k = 1, \dots, 3$ ; выходные значения нейронов на первом слое –  $o_i^{(1)}, i = 1, \dots, 4$ , на втором слое –  $o_j^{(2)}, j = 1, \dots, 4$ .

Данные обозначения очевидным образом обобщаются для более сложных нейронных сетей:  $y_k, k = 1, \dots, |Q|$  – значения на выходном слое нейронной сети;  $w_{ij}^{(n-1)}, i = 1, \dots, l^{(n-1)}, j = 1, \dots, l^{(n)}$  – вес между  $j$ -м нейроном слоя  $n$  и  $i$ -м нейроном слоя  $n - 1$  ( $l^{(n)}$  – количество нейронов на слое  $n$ );  $s_j^{(n)}$  – входное значение  $j$ -го нейрона на слое  $n$ , определяемое на основе  $o_i^{(n-1)}$  – выходных значений нейронов на слое  $n - 1$ :

$$s_j^{(n)} = \sum_{i=1}^{l^{(n-1)}} w_{ij}^{(n-1)} o_i^{(n-1)}, n = 2, 3, \dots, N.$$

Если же нейрон находится на входном слое, то для него будет выполняться

$$s_i^{(1)} = o_i^{(1)}.$$

При обучении нейронной сети каждый объект  $X_m$  из обучающей выборки  $L$  подается на вход сети. Поэтому будет справедливо следующее:

$$s_i^{(1)} = o_i^{(1)} = x_{mi}, m = 1, \dots, |L|.$$

При оценке точности работы сети аналогичным образом на вход сети подаются объекты тестовой выборки  $D$ :

$$s_i^{(1)} = o_i^{(1)} = x_{mi}, m = |L + 1|, \dots, |X|.$$

Для каждого нейрона  $j$  на слое  $n$  ( $n > 1$ ) его выходное значение  $o_j^{(n)}$  определяется в соответствии с выражением

$$o_j^{(n)} = \varphi(s_j^{(n)}). \quad (1.10)$$

Тогда выходные значения нейронной сети будут вычисляться как

$$y_k = \varphi(s_k^{(N)}). \quad (1.11)$$

Таким образом, значения на выходном слое сети зависят от всех весовых коэффициентов.

### 1.3.2 Алгоритм обучения нейронной сети

Классическим алгоритмом обучения нейронной сети считается алгоритм обратного распространения ошибки. Процесс обучения нейронной сети итерационный – предполагается, что обучение происходит в течение некоторого количества эпох, при этом в рамках одной эпохи нейронной сети предъявляются все объекты обучающей выборки [71].

Изначально все весовые коэффициенты сети инициализируются достаточно малыми случайными числами. Согласно [57; 71] существует три варианта корректировки весовых коэффициентов нейронной сети в рамках одной эпохи. Первый из них предполагает корректировку после предъявления каждого объекта из обучающей выборки; второй предполагает разделение обучающей выборки на части (обычно равные) и корректировку весов после предъявления каждой такой части; в третьем варианте корректировка весов выполняется после предъявления всех объектов обучающей выборки. В данной работе рассматривается первый вариант, в связи с чем суммарная функция потерь  $E$  может быть представлена как сумма значений функций потерь  $f(t_j, y_j)$  на каждом выходе нейронной сети. Поэтому обучение сети сводится к решению следующей задачи оптимизации:

$$E = \sum_{j=1}^{l^{(N)}} f(t_j, y_j) \rightarrow \min_{w_{ij}^{(1)}, \dots, w_{ij}^{(N-1)}}, \quad (1.12)$$

где  $t_j$  – требуемый ответ на  $j$ -м выходе сети – определяется следующим образом:

$$t_j = \begin{cases} 1, & X_m \in q_j, \\ 0, & \text{иначе.} \end{cases}$$

Как правило, в качестве функции потерь используется квадратичная функция:

$$f(t_j, y_j) = \frac{1}{2}(y_j - t_j)^2. \quad (1.13)$$

Будем называть нейронные сети, построенные с использованием квадратичной функции потерь, классическими.

Для минимизации суммарной функции потерь необходимо вычислить её производную по весам нейронной сети. Исходя из (1.10) и (1.11), частная производная (1.12) вычисляется по следующему цепному правилу [57]:

$$E'_{ji}{}^{(n)} = \frac{\partial E}{\partial w_{ij}^{(n-1)}} = \frac{\partial E}{\partial o_j^{(n)}} \frac{\partial o_j^{(n)}}{\partial s_j^{(n)}} \frac{\partial s_j^{(n)}}{\partial w_{ij}^{(n-1)}}. \quad (1.14)$$

Во входном значении нейрона  $s_j^{(n)}$  от  $w_{ij}^{(n-1)}$  зависит только одно слагаемое, таким образом:

$$\frac{\partial s_j^{(n)}}{\partial w_{ij}^{(n-1)}} = \frac{\partial}{\partial w_{ij}^{(n-1)}} \left( \sum_{i=1}^{l^{(n-1)}} w_{ij}^{(n-1)} o_i^{(n-1)} \right) = o_i^{(n-1)}. \quad (1.15)$$

Производная выходного значения нейрона  $o_j^{(n)}$  по его входному значению  $s_j^{(n)}$  – это производная функции активации:

$$\frac{\partial o_j^{(n)}}{\partial s_j^{(n)}} = \frac{d\varphi(s_j^{(n)})}{ds_j^{(n)}}. \quad (1.16)$$

Если нейрон находится на выходном слое, то  $n = N$  и первый множитель в (1.14) можно легко вычислить, поскольку  $o_j^{(N)} = y_j$ :

$$\frac{\partial E}{\partial o_j^{(N)}} = \frac{\partial E}{\partial y_j} = \frac{\partial f(y_j, t_j)}{\partial y_j}. \quad (1.17)$$

Теперь получим выражение для производной  $E$  по  $o_j^{(n)}$  в случае, когда  $n$  – произвольный внутренний слой сети. Для этого рассмотрим  $E$  как функцию от входных значений нейронов следующего слоя:

$$E = E(s_k^{(n+1)}), k = 1, \dots, l^{(n+1)},$$

а затем возьмем производную по  $o_j^{(n)}$  [57]:

$$\frac{\partial E}{\partial o_j^{(n)}} = \sum_{k=1}^{l^{(n+1)}} \left( \frac{\partial E}{\partial s_k^{(n+1)}} \frac{\partial s_k^{(n+1)}}{\partial o_j^{(n)}} \right) = \sum_{k=1}^{l^{(n+1)}} \left( \frac{\partial E}{\partial o_k^{(n+1)}} \frac{\partial o_k^{(n+1)}}{\partial s_k^{(n+1)}} w_{jk}^{(n)} \right). \quad (1.18)$$



Производную (1.18) можно вычислить, если известны все производные по выходным значениям для следующего слоя.

Таким образом, производная суммарной функции потерь будет вычисляться в соответствии с

$$E'_{ji}{}^{(n)} = \delta_j^{(n)} o_i^{(n-1)}, \quad (1.19)$$

где  $\delta_j^{(n)}$ , исходя из (1.14)-(1.18), вычисляется следующим образом:

$$\delta_j^{(n)} = \frac{\partial E}{\partial o_j^{(n)}} \frac{\partial o_j^{(n)}}{\partial s_j^{(n)}} = \begin{cases} \frac{\partial f(y_j, t_j)}{\partial y_j} \varphi'(y_j), & n = N, \\ \left( \sum_{k=1}^{l^{(n+1)}} w_{jk}^{(n)} \delta_k^{(n+1)} \right) \varphi'(s_j^{(n)}), & \text{иначе.} \end{cases} \quad (1.20)$$

Поскольку соотношение для вычисления производной известно, для решения задачи оптимизации можно воспользоваться методом градиентного спуска [47]. В этом случае корректировка весов  $w_{ij}^{(n-1)(p)}$  на итерации  $p$  имеет вид:

$$w_{ij}^{(n-1)(p)} = w_{ij}^{(n-1)(p-1)} + \Delta w_{ij}^{(n-1)(p)}.$$

В данном соотношении  $\Delta w_{ij}^{(n-1)(p)}$  вычисляется как

$$\Delta w_{ij}^{(n-1)(p)} = -\eta E'_{ji}{}^{(n)(p)} = -\eta \delta_j^{(n)(p)} o_i^{(n-1)(p)},$$

где  $\eta > 0$  – скорость обучения сети,  $E'_{ji}{}^{(n)(p)}$  – производная суммарной функции потерь на итерации  $p$ , а  $\delta_j^{(n)(p)}$  на итерации  $p$  вычисляется в соответствии с соотношением (1.20), в котором вместо весов  $w_{jk}^{(n)}$  используются веса  $w_{jk}^{(n-1)(p)}$ .

При реализации алгоритма обратного распространения ошибки в качестве функции активации  $\varphi = \varphi(z)$  часто используют одну из следующих функций [58]:

1. Сигмоидальная (логистическая) функция:

$$\varphi(z) = \frac{1}{1 + e^z}. \quad (1.21)$$

2. Гиперболический тангенс:

$$\varphi(z) = \frac{e^{2z} - 1}{e^{2z} + 1}.$$

3. Функция ReLU:

$$\varphi(z) = \max(0, z).$$

Следует отметить, однако, что последняя из перечисленных функций (ReLU) чаще используется в более сложных нейронных сетях (при глубоком обучении). К тому же она обладает существенным недостатком: нулевая производная на части области определения может привести к тому, что корректировка весов не будет оказывать никакого эффекта и модель перестанет обучаться [61; 64]. Как правило, в нейронных сетях с простой архитектурой отдают предпочтение сигмоидальной функции активации или гиперболическому тангенсу.

### 1.3.3 Оценка качества работы нейронной сети

Для оценки качества работы нейросетевых моделей существует множество различных метрик. При построении модели и оценке ее работоспособности крайне важным является выбрать корректную (соответствующую задаче и природе данных) метрику. При решении задачи классификации, как правило, используются метрики *accuracy*, *precision*, *recall*, *F1-score*, ROC (receiver operating characteristic), AUC (area under the ROC curve) [71]. При вычислении данных показателей рассматриваются значения, связанные с ошибками первого (ложноположительное заключение) и второго рода (ложноотрицательное заключение). Рассмотрим таблицу сопряженности (табл. 1.1), которая строится на основе ответов модели о принадлежности объектов классу  $q_i$  и фактической принадлежности объекта этому классу.

В табл. 1.1 использованы следующие обозначения:  $D(q_i)_{TP}$  – количество объектов класса  $q_i$ , для которых класс определен верно (истинно положительные заключения);  $D(q_i)_{TN}$  – количество объектов, которые не относятся к классу  $q_i$  и были отнесены моделью к другому классу (истинно отрицательные заключения);  $D(q_i)_{FP}$  – количество объектов, которые не относятся к классу  $q_i$  но были отнесены моделью к этому классу (ложноположительные заключения);  $D(q_i)_{FN}$  – количество объектов класса  $q_i$ , которые были ошибочно отнесены моделью к другому классу (ложноотрицательные заключения).

Таблица 1.1 – Таблица сопряженности

Ответ модели	Фактически принадлежит	Фактически не принадлежит
Принадлежит	$D(q_i)_{TP}$	$D(q_i)_{FP}$
Не принадлежит	$D(q_i)_{FN}$	$D(q_i)_{TN}$

Метрика *accuracy* является наиболее распространенной. Она показывает долю объектов, для которых класс был определен верно, среди всех объектов:

$$\alpha(q_i) = \frac{D(q_i)_{TP} + D(q_i)_{TN}}{|D|}, \quad (1.22)$$

где  $|D|$  – объем тестовой выборки.

Однако данная метрика плохо подходит для случаев, когда классы данных несбалансированны по размеру. В связи с этим вводятся другие метрики, например *precision*. Эта метрика показывает долю объектов, для которых класс был определен верно, среди всех объектов, отнесенных моделью к конкретному классу:

$$p(q_i) = \frac{D(q_i)_{TP}}{D(q_i)_{TP} + D(q_i)_{FP}}. \quad (1.23)$$

Помимо метрики (1.23), часто применяется метрика *recall*, отражающая чувствительность алгоритма классификации. Эта метрика показывает долю объектов класса  $q_i$ , для которых класс был определен верно, среди всех объектов, на самом деле относящихся к этому классу:

$$r(q_i) = \frac{D(q_i)_{TP}}{D(q_i)_{TP} + D(q_i)_{FN}}. \quad (1.24)$$

В случае, когда метрики (1.23) и (1.24) дают одинаковое значение, для оценки качества модели используют метрику *F1-score* – их среднее гармоническое значение:

$$F1(q_i) = \frac{\sqrt{p(q_i)r(q_i)}}{p(q_i) + r(q_i)}.$$

Изначально рассмотренные оценки были разработаны для бинарной классификации. В задачах, где речь идет о множественной классификации, рассматриваются значения  $\alpha$ ,  $p$ ,  $r$  и  $F1$ , представляющие собой усредненные суммы значений метрик  $\alpha(q_i)$ ,  $p(q_i)$ ,  $r(q_i)$  и  $F1(q_i)$ ,  $i = 1, \dots, |Q|$ , соответственно.

Кроме перечисленных выше метрик, при решении задач бинарной классификации для оценки качества работы модели также используются графические критерии. Наиболее известные из них – ROC и AUC. ROC-кривая (кривая ошибок) отображает зависимость между количеством объектов из класса  $q_i$ , верно отнесенных к данному классу (метрика (1.24)), и количеством объектов, не принадлежащих классу  $q_i$ , но ошибочно отнесенных к нему (метрика  $FPR(q_i)$ ). При

этом значение  $FPR(q_i)$  вычисляется следующим образом:

$$FPR(q_i) = \frac{D(q_i)_{FP}}{D(q_i)_{FP} + D(q_i)_{TN}}. \quad (1.25)$$

Строится ROC-кривая следующим образом: для всех значений порога классификации  $P$ , взятых на отрезке  $[0, 1]$  с шагом разбиения  $\tilde{d}$ , выполняется расчет метрик (1.24) и (1.25). Затем строится график зависимости, где по оси абсцисс откладываются полученные значения метрики (1.25) в каждой точке, а по оси ординат – все полученные значения метрики (1.24).

Помимо ROC-кривых, для оценки точности модели часто используется критерий AUC, который вычисляется как площадь фигуры, образованной ROC-кривой и осью абсцисс в области между точками  $(0,0)$  и  $(1,1)$ . Этот показатель является совокупной оценкой эффективности построенной модели по всем пороговым значениям классификации. Часто AUC интерпретируют как вероятность того, что модель оценит случайный объект, относящийся к рассматриваемому классу, более высоко, чем случайный объект, не относящийся к нему. Значение AUC лежит в пределах от 0,0 до 1,0. Для модели, не способной верно классифицировать ни один объект, значение AUC будет равняться 0,0; для модели, не допускающей ошибок при классификации вообще, – 1,0.

Критерий AUC нечувствителен к значению порога классификации. Он отражает качество работы модели в целом – независимо от того, какой порог классификации выбран. Однако есть такие задачи, в которых может быть важным минимизировать ошибку только одного типа, как, например, в задаче обнаружения спама. Очевидно, при решении этой задачи в приоритете будет минимизация количества ложноположительных решений (когда «нормальное» письмо относится к спаму), даже если это приведет к существенному увеличению ложноотрицательных решений (пропусков писем, которые действительно являются нежелательными).

Известно, что существуют модификации ROC и AUC критериев для задач, в которых количество классов больше двух. В частности, для задач с тремя классами вместо AUC вводится критерий VUC, который вычисляется как объем под ROC-поверхностью, однако этот подход является достаточно трудоемким [83; 102].

## 1.4 Особенности практического использования искусственных нейронных сетей

При использовании нейронных сетей существует ряд моментов, которые необходимо принимать во внимание, чтобы построить качественную модель. Например, на практике нередко могут возникнуть ситуации недообучения и переобучения модели. В случае недообучения модель неспособна улавливать зависимости в данных на достаточно хорошем уровне. Переобучение – обратная ситуация, когда модель слишком хорошо улавливает закономерности в предъявляемых ей данных при обучении, но неспособна обобщить их на другие данные. Оба этих явления приводят к тому, что после завершения процедуры обучения модель будет выдавать низкую точность классификации на предъявляемых ей данных. Проблему недообучения можно решить, например, усложнив архитектуру модели, или использовав больше данных и времени для обучения (если такая возможность есть). Переобучение встречается чаще, чем недообучение, и для решения этой проблемы часто предлагается использовать стратегию «ранней остановки» – использовать дополнительный проверочный набор данных и прекращать обучение сети, когда точность классификации на проверочном наборе начнет падать.

Кроме того, при использовании искусственных нейронных сетей прямого распространения необходимо учитывать особенности алгоритма обратного распространения ошибки. Во-первых, в основе его лежит решение задачи оптимизации методом градиентного спуска – это приводит к тому, что результат работы алгоритма будет сильно зависеть от выбора начального приближения для весовых коэффициентов: при неудачной инициализации весов градиентный спуск может сойтись к локальному минимуму, в результате чего точность работы сети будет ниже, чем в случае, когда алгоритм сходится к глобальному минимуму. Во-вторых, может возникнуть так называемый «паралич сети» – ситуация, когда значения весовых коэффициентов становятся очень большими, а значение производной функции активации, наоборот, слишком малым. В результате при обратном распространении ошибки весовые коэффициенты сети будут изменяться незначительно, и модель перестанет обучаться.

Кроме того, как правило, в данном алгоритме используется квадратичная функция потерь. Из-за этого нейронные сети с простой архитектурой очень часто неудовлетворительно работают на реальных данных, для которых

характерно не только частичное смешивание наблюдений, принадлежащих различным классам, но и наличие нетипичных наблюдений [9; 22; 24]. Это приводит к тому, что приходится или выполнять тщательную предобработку данных, исключая из рассмотрения все нетипичные наблюдения, или переходить к глубокому обучению, используя более сложные и более требовательные с точки зрения вычислительных ресурсов модели.

Однако несмотря на то что нейронные сети обладают рядом особенностей и даже недостатков, которые необходимо учитывать при построении модели, они являются мощным инструментом и широко используются при решении различных практических задач. В связи с этим было создано множество программных библиотек и фреймворков для построения нейронных сетей. Рассмотрим наиболее популярные из них в следующем разделе.

## 1.5 Программные средства для построения нейронных сетей

Программная реализация нейронной сети с нуля – достаточно трудоемкая задача. Помимо того, что необходимо корректно реализовать алгоритм обучения, желательно также оптимизировать вычисления, что не всегда можно сделать, не обладая достаточным опытом. Существенно упростить эту задачу можно путем использования готовых программных продуктов, позволяющих сконструировать и обучить собственную нейронную сеть без необходимости самостоятельной реализации и оптимизации алгоритмов обучения.

Одним из наиболее широко известных инструментов для работы с нейросетевыми моделями является фреймворк TensorFlow [97], разработанный компанией Google. Он прост в освоении, предоставляет интерфейсы для мониторинга и визуализации работы модели. Его программные средства постоянно дорабатываются членами сообщества и программистами Google, также существует возможность запуска построенных моделей на мобильных устройствах или через браузер. Основным его недостатком является сравнительно невысокая скорость работы (по сравнению с другими фреймворками), а также большое количество дублирующих методов [49].

Еще одно широко известное решение для построения нейронных сетей – это фреймворк PyTorch [89], разработанный компанией Facebook. Кроме моделей, реализованных на языке Python [79], он также поддерживает и модели, реализованные на языке C++ [16]. Этот фреймворк часто используется для распознавания изображений, обработки текстов на естественном языке, а так-

же для решения задач обучения с подкреплением. Одним из его преимуществ является поддержка облачных технологий для разработки, предоставляемых такими облачными гигантами, как Amazon (AWS) [50] и Microsoft (Azure) [1]. Кроме того, в данном фреймворке есть большое количество предобученных моделей. Основным его недостатком в том, что фреймворк является достаточно новым и еще не так развит и удобен в использовании, как TensorFlow.

Существуют также фреймворки, которые поддерживают больший набор языков программирования, чем PyTorch: Apache MXNet [52] и Microsoft CNTK [99]. Фреймворк MXNet, помимо Python и C++, также поддерживает такие языки программирования, как JavaScript [75], Scala [101] и R [100]. К его преимуществам можно отнести производительность, масштабируемость, вычислительную эффективность, поддержку графического ускорения, наличие простого и высокопроизводительного прикладного программного интерфейса. Однако, по сравнению с TensorFlow, сообщество данного фреймворка не так развито, что негативным образом сказывается на скорости исправления ошибок и в целом развитии данного программного продукта.

Решение от компании Microsoft – фреймворк CNTK (Cognitive Toolkit) – позволяет работать с нейронными сетями как с направленными графами. Этот фреймворк реализован на языке программирования C++, но также поддерживает разработку на таких языках, как C# [15], Python и Java. Преимуществами данного фреймворка являются надежность, производительность, масштабируемость, оптимизация вычислений, возможность интеграции с инструментами для предобработки и анализа данных, такими как Apache Spark [53], поддержка работы с облачными технологиями (Azure Cloud). Недостатки – практически полное отсутствие открытого сообщества.

Каждое из перечисленных программных решений обладает своими преимуществами и недостатками. При решении практических задач машинного обучения крайне важно выбрать правильный фреймворк, удовлетворяющий всем необходимым требованиям. Одним из таких требований может быть, например, кроссплатформенность используемого набора инструментов. Так, все рассмотренные фреймворки поддерживают работу как с операционной системой Windows, так и с операционными системами семейства Linux. Общим недостатком перечисленных программных решений можно назвать то, что фактически они предназначены для построения классических (не робастных) нейронных сетей с различной архитектурой. Однако в мае 2021 года в

TensorFlow появилась возможность использования функции потерь Хьюбера в нейронных сетях [98], что говорит о заинтересованности разработчиков в развитии нейронных сетей, которые были бы устойчивы к зашумлению данных.

## 1.6 Обоснование цели и задач исследования

Анализ основных задач машинного обучения и подходов к их решению показывает, что в большинстве своем существующие методы предназначены для работы с «достаточно чистыми» данными, то есть такими, которые не содержат в себе нетипичных наблюдений и не являются перемешанными. Из-за этого исследователям часто приходится или тратить время на выполнение тщательной предобработки данных, или использовать более сложные в вычислительном плане модели. Этим недостатком обладает и наиболее популярный в наше время инструмент машинного обучения – искусственные нейронные сети.

Однако в области прикладного статистического анализа широко применяется робастный подход, который является альтернативой классическим методам. Он позволяет не исключать нетипичные наблюдения, а снизить их негативное влияние при обучении модели и, тем самым, добиться достаточно высокой точности работы модели при анализе зашумленных данных.

При решении других задач машинного обучения, однако, такой подход до сих пор используется не очень часто. Существующие работы демонстрируют возможность его применения только в отдельных случаях. Так, в [69] рассматривается применение функции потерь Хьюбера для реализации робастного алгоритма обучения с подкреплением. В [55] приводятся примеры различных функций потерь и рассматривается их использование в алгоритмах обучения без учителя; также выполняется построение адаптивной функции потерь и сравнение ее с уже существующими.

В [51] авторы предлагают модификацию алгоритма Левенберга-Марквардта с использованием функции потерь Хьюбера. Но при этом рассматривается только нейронная сеть с одним скрытым слоем и ее применение для решения конкретной практической задачи – для прогнозирования цен европейских опционов. Принципиальное отличие данного алгоритма от алгоритма обратного распространения ошибки заключается в подходе к корректировке весов: в алгоритме обратного распространения ошибки пересчет весов сети происходит на каждой итерации, в то время как в алгоритме Левенберга-Марквардта корректировка весов выполняется уже после всех итераций.



Кроме того, как показывает анализ наиболее распространенных программных средств для построения нейронных сетей, существующие наборы инструментов предназначены, главным образом, для построения нейронных сетей с использованием квадратичной функции потерь. И только разработчиками фреймворка TensorFlow была предпринята попытка введения робастной функции потерь Хьюбера в нейронных сетях, но большого развития в сообществе это направление пока не получило.

Таким образом, можно сделать вывод о том, что в существующих алгоритмах обучения искусственных нейронных сетей в основном используется квадратичная функция потерь, что приводит к определенным проблемам при анализе реальных (зашумленных) данных. При этом предпринимаются отдельные попытки решения этих проблем, как, например, модификация алгоритма Левенберга-Марквардта в [51], или введение функции потерь Хьюбера в фреймворке TensorFlow. Однако единое комплексное решение для построения нейронных сетей, устойчивых при работе с зашумленными данными, пока предложено не было.

В связи с изложенным выше возникает предположение о возможности использования идей робастного подхода при обучении искусственных нейронных сетей. Цель данного исследования состоит в разработке алгоритмического и программного обеспечения для построения робастных нейронных сетей с простой архитектурой. Для достижения данной цели требуется решить ряд задач.

Во-первых, необходимо провести исследование применимости различных робастных функций потерь в нейронных сетях с учетом ограничений, накладываемых алгоритмом обратного распространения ошибки. Это даст возможность выбрать не одну функцию потерь для построения робастной нейросетевой модели, как было сделано в TensorFlow, а несколько разных функций, что позволит получить целый класс абсолютно новых нейронных сетей.

Во-вторых, нужно построить робастную модификацию алгоритма обратного распространения ошибки и понять, как использование робастной функции потерь вместо квадратичной будет (и будет ли) влиять на логику работы алгоритма и нейронной сети в целом. Кроме того, необходимо провести исследование построенной модификации, чтобы сформулировать рекомендации относительно настройки и использования робастных нейронных сетей.

В-третьих, необходимо провести исследование устойчивости полученных нейросетевых моделей при анализе зашумленных данных и проанализировать

то, как будет влиять на точность работы моделей выбор робастной функции потерь, уровень шума, доля засоряющих наблюдений, число объектов в обучающей и тестовой выборках. Кроме того, представляется интересным исследовать влияние выбора плана эксперимента при обучении робастной нейронной сети.

В-четвертых, как показал анализ существующих программных решений, они фактически не предусматривают построение робастных нейронных сетей с различными функциями потерь, поэтому актуальной является разработка собственного программного модуля для этих целей. Функционал такого модуля должен позволять строить многослойный перцептрон с произвольным количеством скрытых слоев и числом нейронов на каждом из них, а также предусматривать выбор робастной функции потерь из нескольких заданных. Кроме того, как и существующие программные решения, разработанный модуль должен быть кроссплатформенным.

Обозначенные выше задачи рассматривались автором в качестве основных задач, поставленных в рамках данного исследования. Их решение представлено далее в настоящей работе.

## Глава 2 Построение робастных нейронных сетей

В данной главе приводятся результаты исследования применимости робастных функций потерь в нейронных сетях, а также предлагается робастная модификация алгоритма обратного распространения ошибки. Данная модификация излагается в наиболее общем виде и может быть использована для построения нейронных сетей прямого распространения с произвольным числом скрытых слоев и с различными функциями потерь. Кроме того, рассматриваются эффекты от выбора плана эксперимента при обучении робастных нейронных сетей.

### 2.1 Исследование применимости робастных функций потерь в нейронных сетях

Как уже отмечалось в п. 1.3.2, алгоритм обратного распространения ошибки предполагает использование квадратичной функции потерь. Однако можно попробовать использовать робастные функции потерь вместо квадратичной и таким образом обеспечить большую устойчивость данного алгоритма при работе с зашумленными данными. Тогда, чтобы построить робастную модификацию алгоритма обратного распространения ошибки, сначала необходимо выполнить анализ робастных функций потерь и определить, какие из них можно и имеет смысл использовать. Необходимо учесть, что функция потерь, используемая в алгоритме обратного распространения ошибки, должна быть непрерывно дифференцируемой – это означает, что она должна иметь непрерывную производную в каждой точке области определения [19].

Рассмотрим робастные функции потерь, приведенные в п. 1.1.3.2. Сами функции приводятся в левом столбце табл. 2.1, в правом столбце данной таблицы приводятся их производные. Производные всех функций, кроме функции потерь Charbonnier, можно найти в [2; 3; 6; 17; 54; 59; 60; 67; 68]. Производная функции потерь Charbonnier была получена следующим образом:

$$\begin{aligned} \rho'(z) &= \left( \sqrt{\left(\frac{z}{\beta}\right)^2 + 1} \right)' = \\ &= \frac{1}{2\sqrt{\left(\frac{z}{\beta}\right)^2 + 1}} \cdot 2 \frac{z}{\beta^2} = \frac{1}{\beta^2} \frac{z}{\sqrt{\left(\frac{z}{\beta}\right)^2 + 1}}. \end{aligned}$$

Таблица 2.1 — Робастные функции потерь и их производные

Функция потерь $\rho(z)$	Производная функции потерь $\rho'(z)$
Хьюбера	
$\begin{cases} \frac{1}{2}z^2, &  z  \leq \beta \\ \beta z  - \frac{1}{2}\beta^2, &  z  > \beta \end{cases}$	$\begin{cases} z, &  z  \leq \beta \\ -\beta, & z < -\beta \\ \beta, & z > \beta \end{cases}$
Эндрюса	
$\begin{cases} \beta(1 - \cos \frac{z}{\beta}), &  z  < \pi\beta \\ 2\beta, &  z  \geq \pi\beta \end{cases}$	$\begin{cases} \sin \frac{z}{\beta}, &  z  < \pi\beta \\ 0, &  z  \geq \pi\beta \end{cases}$
Хампеля	
$\begin{cases} z^2, & 0 \leq  z  \leq \eta \\ \eta^2 + \eta( z  - \eta), & \eta <  z  \leq \beta \\ \eta^2 + 2\eta(\beta - \eta) - \\ -\frac{\eta(\gamma -  z )^2}{\gamma - \beta} + & \beta <  z  \leq \gamma \\ + \eta(\gamma - \beta), & \\ \eta^2 + 2\eta(\beta - \eta) + \\ + \eta(\gamma - \beta), & \gamma <  z  \end{cases}$	$\begin{cases} 2z, & 0 \leq  z  \leq \eta \\ \text{sign}(z)2\eta, & \eta <  z  \leq \beta \\ \frac{2\eta}{\gamma - \beta}(\gamma -  z )* & \\ * \text{sign}(z), & \beta <  z  \leq \gamma \\ 0, & \gamma <  z  \end{cases}$
Тьюки	
$\begin{cases} z^2, &  z  \leq \beta \\ \beta^2, &  z  > \beta \end{cases}$	$\begin{cases} 2z, &  z  < \beta \\ 0, &  z  \geq \beta \end{cases}$
Биквадратная Тьюки	
$\begin{cases} \frac{z^6}{6\beta^4} - \frac{z^4}{2\beta^2} + \frac{z^2}{2}, &  z  < \beta \\ \frac{\beta^2}{6}, &  z  \geq \beta \end{cases}$	$\begin{cases} \frac{z^5}{\beta^4} - \frac{2z^3}{\beta^2} + z, &  z  < \beta \\ 0, &  z  \geq \beta \end{cases}$
Рамсея	
$\frac{1 - (1 + \beta z )\exp(-\beta z )}{\beta^2}$	$z \exp\{-\beta z \}$
Коши (Лоренца)	
$\ln\left(\frac{1}{2}\left(\frac{z}{\beta}\right)^2 + 1\right)$	$\frac{z}{\frac{1}{2}z^2 + \beta^2}$
Geman-McCluer	
$\frac{z^2/\beta}{1 + z^2/\beta}$	$\frac{2z}{\beta(1 + z^2/\beta)^2}$
«Fair»	
$\beta^2 \left( \frac{ z }{\beta} - \ln \left( 1 + \frac{ z }{\beta} \right) \right)$	$\frac{z}{(1 +  z /\beta)}$

Таблица 2.1 – продолжение

Функция потерь $\rho(z)$	Производная функции потерь $\rho'(z)$
Charbonnier	
$\sqrt{\left(\frac{z}{\beta}\right)^2 + 1}$	$\frac{1}{\beta^2} \frac{z}{\sqrt{z^2/\beta^2 + 1}}$
Уэлша	
$1 - \exp\left(-\frac{1}{2}\left(\frac{z}{\beta}\right)^2\right)$	$\frac{1}{\beta^2} z \exp\left(-\frac{1}{2}\left(\frac{z}{\beta}\right)^2\right)$
Мешалкина	
$\beta^{-1}(1 - \exp(\frac{-\beta z^2}{2}))$	$\exp(\frac{-\beta z^2}{2}) z$

Все рассмотренные функции можно разделить на две группы. В первую группу входят робастные функции потерь Хьюбера, Эндрюса, Хампеля, Тьюки и биквадратная функция потерь Тьюки – их параметр определяет не только поведение функции на интервалах области определения, но и граничные точки этих интервалов. Ко второй группе относятся функции потерь Рамсея, Коши, Geman-McCluer, «Fair», Charbonnier, Уэлша и Мешалкина – их параметр определяет поведение функции на всей области определения.

Покажем, удовлетворяют ли рассматриваемые функции потерь ограничению алгоритма обратного распространения ошибки. Согласно [42], функция  $f(x)$ , определенная при  $x = c$  и всех значениях  $x$ , достаточно близких к  $c$ , называется непрерывной при  $x = c$  (в точке  $c$ ), если существует предел  $f(x)$  при  $x \rightarrow c \pm 0$  и этот предел равен  $f(c)$ :

$$\lim_{x \rightarrow c \pm 0} f(x) = f(c). \quad (2.1)$$

Это равносильно тому, что существуют пределы  $f(c - 0)$  и  $f(c + 0)$  слева и справа и что эти пределы равны между собой и равны  $f(c)$ , т.е.  $f(c - 0) = f(c + 0) = f(c)$ .

Известно также, что сумма или произведение произвольного конечного числа непрерывных функций есть также непрерывная функция. Частное двух непрерывных функций за исключением тех значений независимой переменной, при которых знаменатель обращается в нуль, является непрерывной функцией. Кроме того, сложная функция  $z = F(f(x))$  непрерывна при условии непрерывности функции  $y = f(x)$  на промежутке  $a \leq x \leq b$  и функции  $z = F(y)$  на промежутке  $c \leq y \leq d$  [42].

Используя эти определения, проверим, все ли производные рассматриваемых робастных функций потерь являются непрерывными. Очевидно, что производные функций потерь Рамсея, Уэлша и Мешалкина удовлетворяют описанным выше условиям и являются непрерывными на множестве действительных чисел  $\mathbb{R}$ . Производные оставшихся функций потерь из второй группы (Коши, Geman-McCluer, «Fair», Charbonnier) представляют собой частное двух функций. Поскольку  $\beta > 0$  и  $z \in \mathbb{R}$ , знаменатели этих производных не будут обращаться в ноль – значит производные этих четырех функций потерь также являются непрерывными. Таким образом, все робастные функции потерь из второй группы могут быть использованы в алгоритме обратного распространения ошибки.

Производные робастных функций потерь из первой группы имеют предполагаемые точки разрыва. Проверим, можно ли использовать данные функции потерь при обучении нейронной сети.

**Утверждение 2.1.** Робастные функции потерь Хьюбера, Эндрюса, Хампеля и биквадратная функция потерь Тьюки имеют непрерывные производные и могут быть использованы в алгоритме обратного распространения ошибки.

*Доказательство.* Покажем, что производные рассматриваемых функций потерь действительно являются непрерывными. Для этого возьмем пределы слева и справа в предполагаемых точках разрыва каждой производной.

1. Функция потерь Хьюбера.

Возьмем предел производной функции потерь Хьюбера слева в первой предполагаемой точке разрыва  $z = \beta$ :

$$\lim_{z \rightarrow \beta^-} \rho'(z) = \lim_{z \rightarrow \beta^-} z = \beta,$$

затем возьмем предел в этой же точке справа:

$$\lim_{z \rightarrow \beta^+} \rho'(z) = \lim_{z \rightarrow \beta^+} \beta = \beta.$$

Теперь рассмотрим вторую предполагаемую точку разрыва –  $z = -\beta$  и по аналогии с первой точкой вычислим пределы слева и справа:

$$\lim_{z \rightarrow -\beta^-} \rho'(z) = \lim_{z \rightarrow -\beta^-} (-\beta) = -\beta,$$

$$\lim_{z \rightarrow -\beta^+} \rho'(z) = \lim_{z \rightarrow -\beta^+} z = -\beta.$$

## 2. Функция потерь Эндрюса.

Первая предполагаемая точка разрыва производной данной функции —  $z = -\pi\beta$ . Возьмем пределы слева и справа в этой точке:

$$\lim_{z \rightarrow -\pi\beta^-} \rho'(z) = \lim_{z \rightarrow -\pi\beta^-} 0 = 0,$$

$$\lim_{z \rightarrow -\pi\beta^+} \rho'(z) = \lim_{z \rightarrow -\pi\beta^+} \sin\left(\frac{z}{\beta}\right) = \sin\left(\frac{-\pi\beta}{\beta}\right) = 0.$$

Теперь возьмем пределы слева и справа во второй предполагаемой точке разрыва производной функции потерь Эндрюса  $z = \pi\beta$ :

$$\lim_{z \rightarrow \pi\beta^-} \rho'(z) = \lim_{z \rightarrow \pi\beta^-} \sin\left(\frac{z}{\beta}\right) = \sin\left(\frac{\pi\beta}{\beta}\right) = 0,$$

$$\lim_{z \rightarrow \pi\beta^+} \rho'(z) = \lim_{z \rightarrow \pi\beta^+} 0 = 0.$$

3. Функция потерь Хампеля: Производная данной робастной функции потерь имеет следующие предполагаемые точки разрыва:  $z = \pm\eta$ ,  $z = \pm\beta$  и  $z = \pm\gamma$ . Рассмотрим точку разрыва  $z = \eta$  — вычислим пределы производной функции потерь Хампеля слева и справа в этой точке:

$$\lim_{z \rightarrow \eta^-} \rho'(z) = \lim_{z \rightarrow \eta^-} 2z = 2\eta,$$

$$\lim_{z \rightarrow \eta^+} \rho'(z) = \lim_{z \rightarrow \eta^+} \text{sign}(z)2\eta = 2\eta.$$

Проведем аналогичные вычисления для точки  $z = -\eta$ :

$$\lim_{z \rightarrow -\eta^-} \rho'(z) = \lim_{z \rightarrow -\eta^-} \text{sign}(z)2\eta = -2\eta,$$

$$\lim_{z \rightarrow -\eta^+} \rho'(z) = \lim_{z \rightarrow -\eta^+} 2z = -2\eta.$$

Теперь рассмотрим точку разрыва  $z = \beta$ . Как и ранее, получим значение предела производной в этой точке слева и справа:

$$\lim_{z \rightarrow \beta^-} \rho'(z) = \lim_{z \rightarrow \beta^-} \text{sign}(z)2\eta = 2\eta,$$

$$\begin{aligned} \lim_{z \rightarrow \beta^+} \rho'(z) &= \lim_{z \rightarrow \beta^+} \frac{2\eta}{\gamma - \beta} (\gamma - |z|) \text{sign}(z) = \\ &= \frac{2\eta}{\gamma - \beta} (\gamma - |\beta|) \text{sign}(\beta) = 2\eta. \end{aligned}$$

Аналогичным образом вычислим значение пределов слева и справа в точке  $z = -\beta$ :

$$\begin{aligned}\lim_{z \rightarrow -\beta^-} \rho'(z) &= \lim_{z \rightarrow -\beta^-} \frac{2\eta}{\gamma - \beta} (\gamma - |z|) \text{sign}(z) = \\ &= \frac{2\eta}{\gamma - \beta} (\gamma - |-\beta|) \text{sign}(-\beta) = -2\eta, \\ \lim_{z \rightarrow -\beta^+} \rho'(z) &= \lim_{z \rightarrow -\beta^+} \text{sign}(z) 2\eta = -2\eta.\end{aligned}$$

Значение предела производной функции потерь Хампеля в точке  $z = \gamma$  слева и справа будет вычисляться следующим образом:

$$\begin{aligned}\lim_{z \rightarrow \gamma^-} \rho'(z) &= \lim_{z \rightarrow \gamma^-} \frac{2\eta}{\gamma - \beta} (\gamma - |z|) \text{sign}(z) = \\ &= \frac{2\eta}{\gamma - \beta} (\gamma - |\gamma|) \text{sign}(\gamma) = 0, \\ \lim_{z \rightarrow \gamma^+} \rho'(z) &= 0.\end{aligned}$$

В точке  $z = -\gamma$  предел слева будет равен

$$\lim_{z \rightarrow -\gamma^-} \rho'(z) = 0,$$

предел справа будет равен

$$\lim_{z \rightarrow -\gamma^+} \rho'(z) = \lim_{z \rightarrow -\gamma^+} \frac{2\eta}{\gamma - \beta} (\gamma - |z|) \text{sign}(z) = \frac{2\eta}{\gamma - \beta} (\gamma - |-\gamma|) \text{sign}(-\gamma) = 0.$$

4. Биквадратная функция потерь Тьюки: Первая предполагаемая точка разрыва производной данной функции –  $z = \beta$ . Вычислим значения пределов слева и справа в этой точке:

$$\begin{aligned}\lim_{z \rightarrow \beta^-} \rho'(z) &= \lim_{z \rightarrow \beta^-} \left( \frac{z^5}{\beta^4} - \frac{2z^3}{\beta^2} + z \right) = \frac{\beta^5}{\beta^4} - \frac{2\beta^3}{\beta^2} + \beta = \beta - 2\beta + \beta = 0, \\ \lim_{z \rightarrow \beta^+} \rho'(z) &= \lim_{z \rightarrow \beta^+} 0 = 0.\end{aligned}$$

Аналогичным образом возьмем пределы слева и справа во второй предполагаемой точке разрыва –  $z = -\beta$ :

$$\begin{aligned}\lim_{z \rightarrow -\beta^-} \rho'(z) &= \lim_{z \rightarrow -\beta^-} 0 = 0, \\ \lim_{z \rightarrow -\beta^+} \rho'(z) &= \lim_{z \rightarrow -\beta^+} \left( \frac{z^5}{\beta^4} - \frac{2z^3}{\beta^2} + z \right) = \frac{-\beta^5}{\beta^4} + \frac{2\beta^3}{\beta^2} - \beta = -\beta + 2\beta - \beta = 0.\end{aligned}$$



Приведенные рассуждения показывают, что для всех производных рассматриваемых робастных функций потерь выполняется условие (2.1), значит, они являются непрерывными. Следовательно, данные функции потерь могут быть использованы вместо квадратичной при обучении нейронной сети. Таким образом, утверждение доказано.  $\square$

**Утверждение 2.2.** Робастная функция потерь Тьюки не удовлетворяет ограничению алгоритма обратного распространения ошибки.

*Доказательство.* Проверим выполнение условия (2.1) для производной функции потерь Тьюки, которая имеет две предполагаемые точки разрыва. Сначала возьмем пределы слева и справа в первой точке разрыва  $z = \beta$ :

$$\lim_{z \rightarrow \beta^-} \rho'(z) = \lim_{z \rightarrow \beta^-} 2z = 2\beta,$$

$$\lim_{z \rightarrow \beta^+} \rho'(z) = \lim_{z \rightarrow \beta^+} 0 = 0.$$

Теперь возьмем пределы слева и справа во второй точке разрыва  $z = -\beta$ :

$$\lim_{z \rightarrow -\beta^-} \rho'(z) = \lim_{z \rightarrow -\beta^-} 0 = 0,$$

$$\lim_{z \rightarrow -\beta^+} \rho'(z) = \lim_{z \rightarrow -\beta^+} 2z = -2\beta.$$

Из приведенных рассуждений видно, что  $\lim_{z \rightarrow \beta^-} \rho'(z) \neq \lim_{z \rightarrow \beta^+} \rho'(z)$  и  $\lim_{z \rightarrow -\beta^-} \rho'(z) \neq \lim_{z \rightarrow -\beta^+} \rho'(z)$ , следовательно, условие (2.1) не выполняется. Значит, функция потерь Тьюки не является непрерывно дифференцируемой и не удовлетворяет ограничениям алгоритма обратного распространения ошибки. Утверждение доказано.  $\square$

Таким образом, все робастные функции потерь из табл. 2.1, кроме функции потерь Тьюки, подходят для построения робастной модификации алгоритма обратного распространения ошибки. Хотя производная функции потерь Тьюки претерпевает скачкообразный разрыв, формально она определена на всей области определения. Однако использовать эту функцию потерь при обучении нейронной сети не имеет смысла, поскольку выбросы в данном случае будут попросту исключаться.

Производная функции потерь Эндрюса обращается в ноль на интервалах  $[-\infty, -\beta]$  и  $[\beta, +\infty]$ , а производная биквадратной функции потерь Тьюки – на интервалах  $[-\infty, -\pi\beta]$  и  $[\pi\beta, +\infty]$ , однако значение параметра функции потерь

здесь влияет не только на ширину интервалов, но и на значение самой производной, поэтому полностью выбросы исключаться не будут. Кроме того, среди функций первой группы особое внимание следует уделить функции потерь Хампеля, поскольку в ней используется не один параметр, а три, что потенциально может обеспечить более гибкую настройку алгоритма. Очевидно, что настройка сети при использовании этой функции потребует больших временных и вычислительных затрат по сравнению с функциями, у которых один параметр, в связи с чем эта функция потерь не рассматривалась в рамках исследования.

Проведенный анализ позволил определить, какие робастные функции потерь подходят для построения робастной модификации алгоритма обратного распространения ошибки. Из дальнейшего рассмотрения были исключены функции потерь Тьюки и Хампеля по обозначенным выше причинам.

## 2.2 Робастная модификация алгоритма обучения нейронной сети

Перейдем к построению робастного алгоритма обратного распространения ошибки. Предлагаемая модификация данного алгоритма заключается в использовании робастной функции потерь  $f_R(y_j, t_j) = \rho(z)$ , где  $z = y_j - t_j$  вместо квадратичной функции (1.13):

$$f(y_j, t_j) = f_R(y_j, t_j),$$

где в качестве  $f_R(y_j, t_j)$  будет использована любая из функций, представленных в табл. 2.1, кроме функции потерь Хампеля или Тьюки.

Следующее утверждение позволяет увидеть изменения в алгоритме обратного распространения ошибки при замене квадратичной функции потерь на робастную.

**Утверждение 2.3.** Использование робастной функции потерь вместо квадратичной функции потерь (1.13) в алгоритме обратного распространения ошибки приведет к изменению только тождества (1.20).

*Доказательство.* Подставим в (1.12) вместо квадратичной функции потерь робастную. В этом случае задача оптимизации будет следующей:

$$E = \sum_{j=1}^{l^{(N)}} f_R(t_j, y_j) \rightarrow \min_{w_{ij}^{(1)}, \dots, w_{ij}^{(N-1)}}.$$

Замена квадратичной функции потерь на робастную не влияет на структуру нейронной сети, поэтому заметим, что соотношения (1.14), (1.15), (1.16) не

изменяться, так как сами по себе от вида функции потерь они не зависят. При этом множитель (1.17) теперь примет вид

$$\frac{\partial E}{\partial o_j^{(N)}} = \frac{\partial E}{\partial y_j} = \frac{\partial f_R(y_j, t_j)}{\partial y_j}. \quad (2.2)$$

Цепное правило для нейрона на произвольном внутреннем слое сети (1.18) также зависит только от структуры сети и поэтому не изменится. Следовательно, останется прежним и соотношение для вычисления производной суммарной функции потерь (1.19). Однако в силу (2.2) множитель  $\delta_j^{(n)}$  в (1.19) теперь примет вид

$$\delta_j^{(n)} = \frac{\partial E}{\partial o_j^{(n)}} \frac{\partial o_j^{(n)}}{\partial s_j^{(n)}} = \begin{cases} \frac{\partial f_R(y_j, t_j)}{\partial y_j} \varphi'(y_j), & n = N, \\ \left( \sum_{k=1}^{l^{(n+1)}} w_{jk}^{(n)} \delta_k^{(n+1)} \right) \varphi'(s_j^{(n)}), & \text{иначе,} \end{cases} \quad (2.3)$$

что согласуется с утверждением. Таким образом, утверждение доказано.  $\square$

Доказанное утверждение позволяет построить модификацию алгоритма обратного распространения ошибки путем замены (1.20) на (2.3), оставляя без изменений большинство шагов алгоритма. При этом за счет использования различных функций потерь появляется возможность получить целый класс совершенно новых нейронных сетей.

### 2.3 Исследование влияния планов эксперимента на точность работы робастной нейронной сети

Рассмотрим использование робастных нейронных сетей для решения таких задач классификации, когда при сборе данных у исследователя есть возможность тем или иным образом влиять на значение хотя бы одного из признаков, подаваемых на вход сети. Иными словами, проведем исследование эффектов от выбора плана эксперимента при сборе данных для обучения нейронных сетей.

Рассмотрим следующую задачу бинарной классификации. Пусть имеется множество объектов  $X = \{X_1, \dots, X_{|X|}\}$ , каждый из которых описывается двухкомпонентным вектором значений признаков  $x_m = \{x_{m1}, x_{m2}\}$ , а множество классов включает в себя два класса:  $Q = \{q_1, q_2\}$ . При этом набор данных обладает следующими свойствами. Значения признака  $x_{m1}$  для класса  $q_1$  выбираются случайным образом на отрезке  $[1, 15]$ , а для класса  $q_2$  – на отрезке

[9, 25]. Значения признака  $x_{m2}$  для класса  $q_1$  изменяются на отрезке [10, 14] с шагом 1, а для класса  $q_2$  – на отрезке [15, 20] так же с шагом 1. Кроме того, в обоих классах могут присутствовать следующие нетипичные наблюдения: для таких наблюдений, принадлежащих классу  $q_1$ , характерно значение второго признака  $x_{m2} = 20$ ; для нетипичных наблюдений из класса  $q_2$  характерно значение второго признака  $x_{m2} = 10$ . Доля таких наблюдений во всем множестве объектов  $X$  составляет 0,1.

Предположим, что при сборе данных у исследователя есть возможность контролировать значения признака  $x_{m2}$ . Пусть проводятся два эксперимента, каждый из которых включает в себя 500 наблюдений (по 250 для каждого класса). План первого эксперимента выглядит следующим образом:

$$\xi^{(1)} = \begin{Bmatrix} 10 & 20 \\ r_1 & r_2 \end{Bmatrix},$$

где  $r_1 + r_2 = 500$ . План второго эксперимента выглядит следующим образом:

$$\xi^{(2)} = \begin{Bmatrix} 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\ r_1 & r_2 & r_3 & r_4 & r_5 & r_6 & r_7 & r_8 & r_9 & r_{10} & r_{11} \end{Bmatrix},$$

где  $\sum_{i=1}^{11} r_i = 500$ .

В результате проведения этих экспериментов были получены два набора данных. Для каждого набора данных дополнительно выполнялось моделирование выбросов для признака  $x_{m1}$  следующим образом:

$$\tilde{x}_{m1} = x_{m1} + \varepsilon_{m1},$$

где  $\varepsilon_{m1}$  – независимые и одинаково распределенные случайные ошибки. Эти ошибки имели следующую функцию распределения:

$$F_1(x) = (1 - \lambda)F_1(x, 0, \sigma_{11}) + \lambda F_2(x, 0, \sigma_{12}),$$

где  $F_j(x, 0, \sigma_{1,j})$ ,  $j = 1, 2$  – функция нормального распределения с нулевым математическим ожиданием и дисперсией  $\sigma_{1,j}^2$ ,  $\lambda \in [0, 1]$  – параметр смеси, играющий роль доли засоряющих наблюдений. В данном случае задавались не сами значения дисперсий  $\sigma_{11}^2, \sigma_{12}^2$ , а соответствующие им значения уровня шума. Уровень шума, введенный в [13], определяется следующим образом:

$$\rho_{ij} = \frac{\sigma_{ij}}{c} 100\%, \quad (2.4)$$

где  $c^2$  – дисперсия незашумлённой выборки.

При моделировании выбросов в данной задаче полагалось, что дисперсия  $\sigma_{11}^2$  соответствовала уровню шума  $\rho_{11} = 30\%$ , а дисперсия  $\sigma_{12}^2$  – уровню шума  $\rho_{12} = 120\%$ . Значение доли засоряющих наблюдений  $\lambda$  полагалось равным 0,25.

Для решения рассматриваемой задачи предлагается использовать робастную ИНС с функцией потерь Хьюбера (параметр функции потерь  $\beta = 0,5$ ) и одним скрытым слоем, имеющую следующую архитектуру: входной слой включает в себя 2 нейрона (по числу признаков объектов), скрытый – 3 нейрона, выходной – 2 нейрона (по числу классов). В качестве функции активации  $\varphi = \varphi(x)$  выбрана сигмоида.

Поскольку классы объектов имеют одинаковый размер, для оценки точности работы нейронных сетей можно использовать только метрику  $\alpha$ . Все нейронные сети обучались в течение 1000 эпох, значение метрики  $\alpha$  фиксировалось при числе эпох обучения  $h$ , равном 2, 5, 10, 50, 100 и далее до 1000 эпох с шагом в 100. Для обоих наборов данных выполнялось по 50 вычислительных экспериментов (в ходе каждого вычислительного эксперимента 100 раз проводилось обучение нейронной сети с разным начальным приближением для весов), результаты которых затем усреднялись. Полученные результаты приводятся на рис. 2.1.

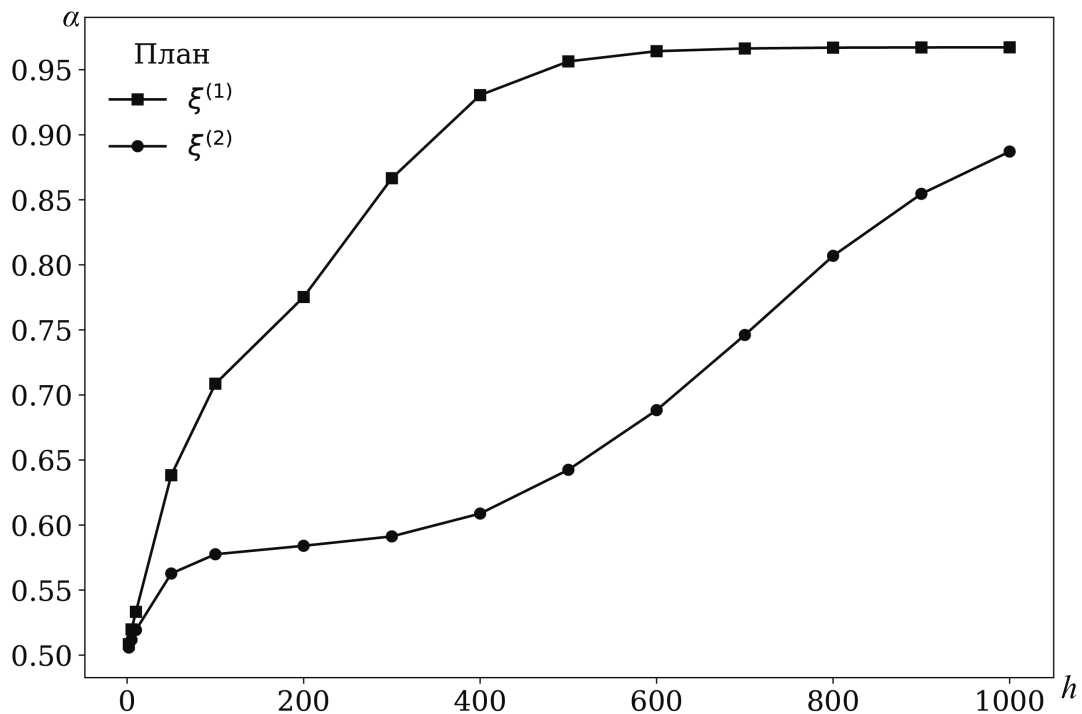


Рисунок 2.1 — Точность работы нейронных сетей для планов эксперимента  $\xi^{(1)}$  и  $\xi^{(2)}$

Из рис. 2.1 видно, что для плана эксперимента  $\xi^{(1)}$  точность работы робастной нейронной сети, начиная с 500 эпох обучения, достигает значения 0,95 по метрике  $\alpha$  и фактически перестает изменяться. В то же время для плана эксперимента  $\xi^{(2)}$  точность работы нейронной сети по метрике  $\alpha$  не превышает значение 0,90 даже на 1000 эпох обучения. В связи с этим для плана  $\xi^{(2)}$  проводилось дополнительно еще 50 вычислительных экспериментов, в ходе которых нейронная сеть обучалась в течение 2000 эпох. Однако даже на 2000 эпох точность работы нейронной сети по метрике  $\alpha$  достигла только значения 0,93.

После этого для обоих планов эксперимента выполнялся расчет среднего значения дисперсии отклика (выходного значения нейронной сети), полученного в ходе вычислительных экспериментов (для каждого рассматриваемого числа эпох обучения). Иными словами, выполнялось сравнение качества планов с точки зрения критерия  $Q$ -оптимальности. График, представленный на рис. 2.2, иллюстрирует изменение среднего значения дисперсии отклика с увеличением числа эпох обучения.

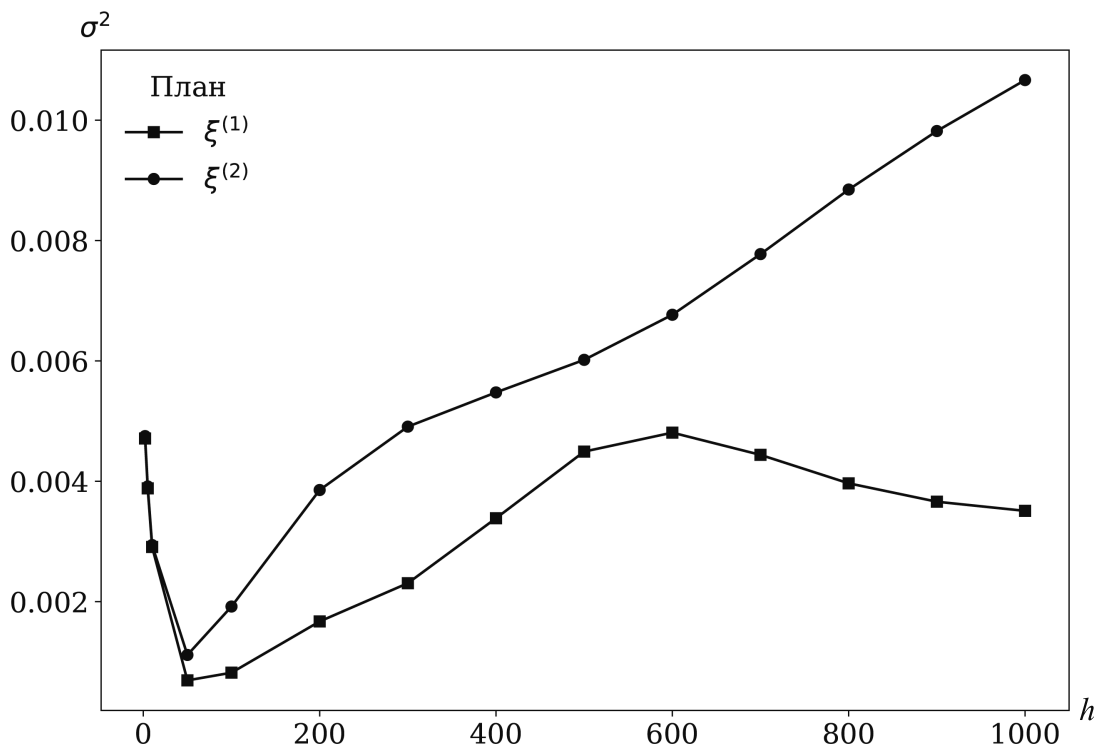


Рисунок 2.2 — Изменение дисперсии отклика с ростом числа эпох для планов  $\xi^{(1)}$  и  $\xi^{(2)}$

Нетрудно заметить, что для плана  $\xi^{(2)}$  значение дисперсии отклика с ростом числа эпох обучения становится значительно выше, чем для плана  $\xi^{(1)}$ .

Кроме того, для плана  $\xi^{(1)}$  дисперсия отклика со временем начинает снижаться, тогда как для плана  $\xi^{(2)}$  наоборот растет. Следует отметить, что динамика изменения дисперсии отклика соответствует изменениям значений метрики  $\alpha$ , представленных на рис. 2.1. Начиная с 600 эпох обучения, точность работы нейронной сети для плана  $\xi^{(1)}$  фактически перестает изменяться, и в то же время значение дисперсии отклика начинает немного убывать. Для плана  $\xi^{(2)}$  после 600 эпох обучения точность работы нейронной сети продолжает расти, как и средняя дисперсия отклика. Все это говорит о том, что процесс обучения нейронной сети для плана эксперимента  $\xi^{(1)}$  является более быстрым, чем для плана  $\xi^{(2)}$ , а сам план эксперимента  $\xi^{(1)}$  – существенно ближе к оптимальному.

Проведенные исследования показали, что выбор более близкого к оптимальному плана эксперимента будет способствовать повышению точности работы нейронной сети, а также существенному сокращению числа эпох обучения модели, требующихся для достижения высокой точности, и, как следствие, времени вычисления.

## Выводы по главе 2

Основные результаты, полученные в данной главе, можно сформулировать следующим образом:

1. Проведено исследование применимости 12 робастных функций потерь при построении нейронных сетей. Доказано утверждение, показывающее возможность использования в алгоритме обратного распространения ошибки робастных функций потерь Хьюбера, Эндрюса, Хампеля, Рамсея, Коши, Geman-McCluer, «Fair», Charbonnier, Уэлша, Мешалкина и биквадратной функции потерь Тьюки. Кроме того, доказано утверждение, показывающее, что робастная функция потерь Тьюки не может быть использована при обучении нейронных сетей. Выбрано 10 функций потерь, подходящих для использования в алгоритме обратного распространения ошибки.

2. Предложена робастная модификация алгоритма обратного распространения ошибки. Сформулировано и доказано утверждение, позволяющее построить предложенную модификацию с использованием различных робастных функций потерь, не меняя при этом основную логику алгоритма.

3. Продемонстрированы эффекты использования различных планов эксперимента при построении робастных нейронных сетей.

## Глава 3 Исследование устойчивости робастных нейронных сетей

В этой главе приводятся результаты исследования устойчивости робастных нейронных сетей, которые были построены с использованием функций потерь, выбранных в разделе 2.1. Кроме того, представлены рекомендации по настройке этих сетей (выбору значений параметра робастной функции потерь).

### 3.1 Исследуемые данные и модели робастных нейронных сетей

Использование робастных функций потерь позволило получить целый ряд новых нейронных сетей (см. п. 2.2), работоспособность которых при анализе различных зашумленных данных была исследована с помощью многочисленных вычислительных экспериментов. Исследования проводились в три этапа: настройка робастных нейронных сетей, исследование их устойчивости при различной доле засоряющих наблюдений, исследование их устойчивости при различном числе объектов в наборе данных.

Основным набором данных, использованным при проведении исследований, является набор «Ирисы Фишера» [104] – он часто используется именно для иллюстрации работы различных алгоритмов классификации, поскольку является достаточно компактным и позволяет построить классификатор при минимуме признаков объекта. Набор состоит из 150 объектов, которые представляют собой экземпляры ирисов 3 видов (*iris setosa*, *iris versicolor*, *iris virginica*). Каждый ирис  $X_m$ ,  $m = 1, \dots, 150$  характеризуется 4 признаками  $x_{mi}$  (длина и ширина чашелистика, длина и ширина лепестка) и классом  $q_k$ ,  $k = 1, 2, 3$ . Столь малое количество признаков позволяет наглядно отслеживать существующие закономерности и работу алгоритма.

Набор данных «Ирисы Фишера» не является синтетическим – это означает, что данные представляют собой результат реальных измерений и могут содержать погрешности (фоновый шум). С учетом этого при зашумлении данных выполнялось моделирование именно нетипичных наблюдений, что позволило оценить их влияние на работу нейронных сетей.

На рис. 3.1 для каждой пары признаков объектов рассматриваемого набора данных приводятся диаграммы рассеивания, позволяющие наглядно увидеть влияние признаков на взаимное расположение классов. Из рис. 3.1 видно, что наиболее важными с точки зрения различия объектов являются признаки  $x_{m3}$ ,  $x_{m4}$ , поэтому моделирование выбросов выполнялось именно для этих



двух признаков:

$$\tilde{x}_{mi} = x_{mi} + \varepsilon_{mi}, i = 3,4, \quad (3.1)$$

где  $\varepsilon_{mi}$  – случайные ошибки, которые моделировались независимыми и одинаково распределенными. Функция распределения  $\varepsilon_{mi}$  следующая:

$$F_i(x) = (1 - \lambda)F_1(x,0,\sigma_{i1}) + \lambda F_2(x,0,\sigma_{i2}), i = 3,4, \quad (3.2)$$

где  $F_j(x,0,\sigma_{ij}), j = 1,2$  – функция нормального распределения с нулевым математическим ожиданием и дисперсией  $\sigma_{ij}^2, \lambda \in [0,1]$  – параметр смеси, который здесь играет роль доли засоряющих наблюдений. В проведённых экспериментах полагалось, что  $\sigma_{i1}^2 < \sigma_{i2}^2$ . При этом в ходе моделирования, как и в вычислительных экспериментах, представленных в разделе 2.3, задавались не сами значения дисперсий  $\sigma_{i1}^2$  и  $\sigma_{i2}^2$ , а соответствующие им значения уровня шума (2.4). Для третьего признака дисперсия  $\sigma_{31}^2$  соответствовала уровню шума  $\rho_{31} = 30\%$ , дисперсия  $\sigma_{32}^2$  – уровню шума  $\rho_{32} = 120\%$ ; для четвертого признака  $\sigma_{41}^2$  соответствовала уровню шума  $\rho_{41} = 40\%$ , а  $\sigma_{42}^2$  – уровню шума  $\rho_{42} = 150\%$ .

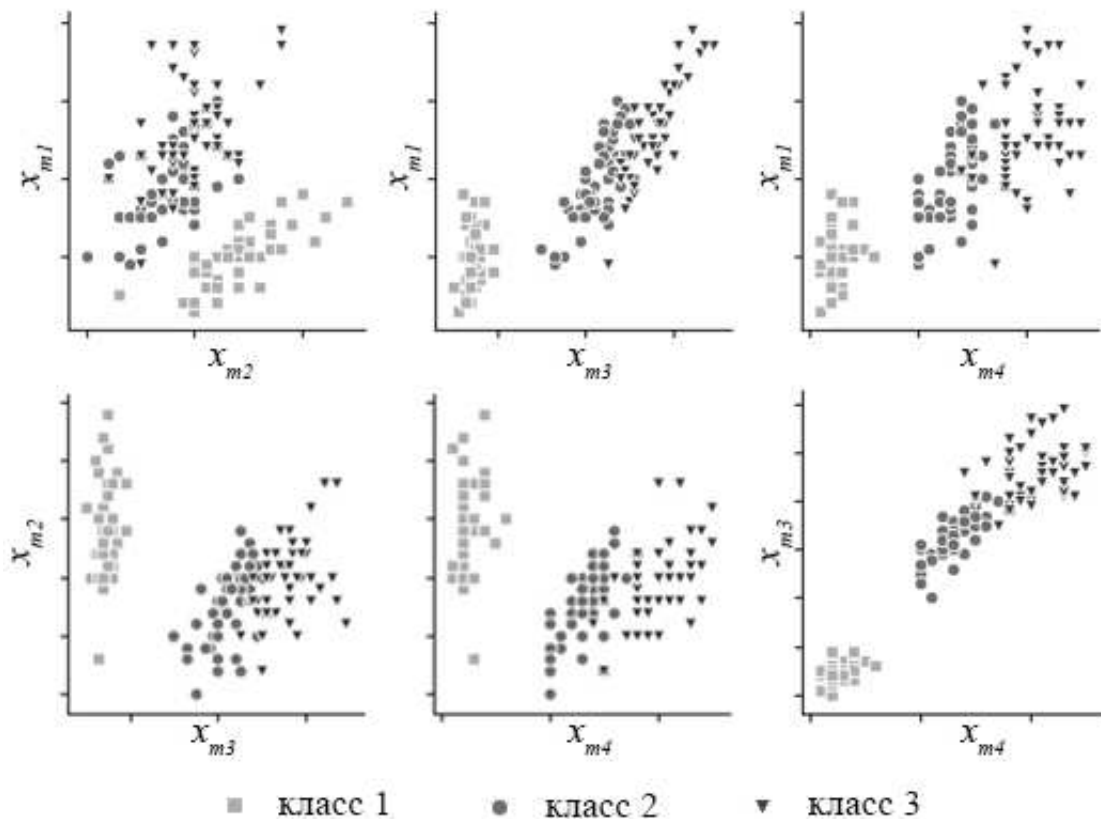


Рисунок 3.1 — Попарные графики признаков объектов набора «Ирисы Фишера»

При моделировании выбросов на основе набора данных «Ирисы Фишера» было получено 100 различных зашумленных наборов данных, которые затем

использовались в ходе вычислительных экспериментов. Эти наборы данных делились на обучающую и тестовую выборки, причем обучающая выборка  $L$  включала в себя 80% объектов ( $|L| = 120$  объектов – объем обучающей выборки), а тестовая выборка  $D$  включала в себя оставшиеся 20% объектов ( $|D| = 30$  объектов – объем тестовой выборки). При настройке робастных нейронных сетей и при исследовании их устойчивости разделение наборов данных на обучающую и тестовую выборки выполнялось случайным образом, т.е. на разных этапах исследований рассматривались разные тестовые выборки. Формально в условиях малого количества объектов в наборе данных такой подход можно считать эквивалентным выделению специальной валидационной выборки [105], которая обычно используется для настройки гиперпараметров нейронных сетей.

Исходя из конфигурации анализируемого набора данных, для исследований были построены нейронные сети с такой же архитектурой, как и рассмотренная в пункте 1.3.1: скрытый слой сети состоял из 4 нейронов, входной слой – также из 4 нейронов (по количеству признаков объектов), выходной – из 3 нейронов (по количеству классов). Рассматривались модели нейронных сетей со следующими робастными функциями потерь: Эндрюса, Хьюбера, Уэлша, Коши, биквадратной функцией потерь Тьюки, Geman-McCluer, Мешалкина, Рамсея, Charbonnier, «Fair». Таким образом, было построено 10 моделей робастных нейронных сетей. В качестве функции активации во всех моделях использовалась сигмоида (1.21). Поскольку в анализируемом наборе данных классы были сбалансированы по объему, оценка точности классификации для построенных моделей проводилась только на основе метрики  $\alpha$ . Перейдем к рассмотрению результатов проведенных исследований.

### 3.2 Настройка робастных нейронных сетей

На первом этапе исследований необходимо было определить наилучшие значения параметра  $\beta$  робастных функций потерь – иными словами, такие значения, при которых точность работы сети была бы наиболее высокой. Для этого значения параметра варьировались на различных интервалах ( $\beta_{min}, \beta_{max}$ ].

Для определения этих интервалов было проведено предварительное исследование. Для всех рассматриваемых функций потерь были зафиксированы левая граница интервала  $\beta_{min} = 0,00$ , правая граница  $\beta_{max} = 20,00$  и шаг разбиения 0,10. Значение доли засоряющих наблюдений было зафиксировано в точке  $\lambda = 0,25$ . Для каждого значения  $\beta$  на этом интервале проводилось

по 10 вычислительных экспериментов, в ходе которых фиксировалось значение метрики  $\alpha$  на 500 эпох обучения. Полученные результаты затем усреднялись. Рассмотрим результаты предварительного исследования на примере функции потерь Коши (табл. 3.1).

Таблица 3.1 — Результаты предварительного исследования для функции потерь Коши

$\beta$	5,50	6,00	6,50	7,00	7,50	10,00	15,00	20,00
$\alpha$	0,93834	0,95678	0,96489	0,98462	0,98477	0,98556	0,9856	0,98572

Из таблицы видно, что с ростом значения параметра  $\beta$  значение метрики  $\alpha$  перестает значимо изменяться: разница в точности при  $\beta = 7,0$  и  $\beta = 7,5$  составляет всего 0,00015, а при  $\beta = 7,0$  и  $\beta = 20,0$  – 0,0011. Тогда как при  $\beta = 6,5$  и  $\beta = 7,0$  разница в точности работы составила 0,01973. Таким образом, рассматривать значения параметра  $\beta > 7,0$  в данном случае не имеет смысла. С использованием приведенной логики рассуждений, правая граница интервала значений параметра  $\beta$  была скорректирована для всех других рассматриваемых робастных функций потерь. Кроме того, для функций потерь Хьюбера, «Fair» и Geman-McCluer был скорректирован шаг разбиения интервала. В первом случае шаг разбиения 0,10 оказался большим для выбранного интервала, что потенциально могло бы привести к пропуску наилучшего значения  $\beta$ . Во втором и третьем случаях выбранный изначально шаг оказался слишком маленьким – с каждым новым значением  $\beta$  при таком разбиении точность работы сети изменялась незначительно (не более, чем на 0,0005).

Полученные в результате предварительного исследования границы и шаг разбиения интервалов приводятся в табл. 3.2. Следует также отметить, что для функции Geman-McCluer, помимо значений из интервала, приведенного в табл. 3.2, рассматривались значения  $\beta = 20, 50, 100$ , так как было отмечено улучшение точности работы нейронной сети с ростом значения параметра.

После того, как были определены интервалы, на которых рассматривались значения параметра  $\beta$ , исследования проводились следующим образом. Значение доли засоряющих наблюдений  $\lambda$  в обучающей выборке изменялось от 0,05 до 0,40 с шагом в 0,05. Для каждого значения  $\lambda$  при всех  $\beta$  из указанных интервалов с заданным шагом разбиения вычислялось значение метрики  $\alpha$  и фиксировалось при различном числе эпох, в течение которых проходило обучение нейронной сети (50, 100 и далее до 1000 с шагом в 100 эпох). Исходя из

Таблица 3.2 — Интервалы значений параметра робастных функций потерь

Функция потерь	$\beta_{min}$	$\beta_{max}$	Шаг разбиения
Эндрюса	0,00	5,00	0,10
Уэлша	0,00	8,00	0,10
Хьюбера	0,00	1,00	0,05
Рамсея	0,00	5,00	0,10
«Fair»	0,00	10,00	0,50
Биквадратная Тьюки	0,00	5,00	0,10
Коши	0,00	7,00	0,10
Geman-McCluer	0,00	10,00	0,50
Charbonnier	0,00	7,00	0,10
Мешалкина	0,00	5,00	0,10

соотношения числа эпох обучения и значения метрики  $\alpha$ , можно судить о скорости обучения нейронной сети – чем больше значение метрики и чем меньше число эпох, тем быстрее обучается ИНС.

В ходе исследований для каждой робастной сети проводилось по 100 вычислительных экспериментов, результаты которых затем усреднялись. На основании полученных результатов для каждой нейронной сети были зафиксированы наилучшие значения параметра  $\beta$  – такие, при которых точность классификации была максимальной. Эти значения приводятся в табл. 3.3 для каждого  $\lambda$  на 100, 300, 500 и 1000 эпох обучения.

Полученные результаты позволяют сделать вывод о том, что практически для всех рассмотренных функций потерь явная зависимость между значением параметра  $\beta$  и долей засоряющих наблюдений  $\lambda$  не прослеживается. Для функции потерь Хьюбера при сравнительно небольшой доле выбросов ( $\lambda < 0,15$ ) характерно убывание значения  $\beta$  с ростом числа эпох обучения, при большей доле выбросов значение параметра с ростом числа эпох начинает колебаться либо не изменяется ( $\lambda = 0,35$ ). Для нейронной сети с функцией потерь Рамсея при доле выбросов  $\lambda < 0,25$  наилучшее значение  $\beta$  возрастает с ростом числа эпох. При большей доле выбросов и числе эпох обучения до 500 оно так же возрастает, а при числе эпох от 501 до 1000, наоборот, начинает убывать.

Таблица 3.3 — Наилучшие значения параметра робастных функций потерь

$\lambda$	Эпохи	Функция потерь									
		Энд-рюса	Уэлша	Хьюбера	Рамсея	«Fair»	Коши	Биквадратная Тьюки	Geman-McCluer	Мешалкина	Charbonnier
0,05	100	1,5	1,1	0,6	0,6	1,5	0,7	1,7	2,0	1,4	1,2
	300	5,0	2,3	0,5	0,8	1,5	2,9	1,6	10,0	1,8	2,4
	500	5,0	3,1	0,5	1,4	0,5	4,7	1,5	20,0	2,2	3,1
	1000	5,0	3,5	0,3	1,7	0,1	5,2	1,2	20,0	3,6	3,8
0,1	100	1,6	1,2	0,6	0,6	1,0	0,8	1,9	3,0	1,4	1,3
	300	5,0	2,6	0,3	1,6	0,5	3,0	1,3	10,0	3,2	2,6
	500	5,0	3,2	0,3	1,7	0,5	5,0	1,3	20,0	4,0	3,2
	1000	5,0	3,9	0,2	1,8	0,1	5,6	1,2	20,0	4,0	3,4
0,15	100	1,7	1,2	0,5	0,8	1,0	0,7	1,6	2,0	2,0	1,2
	300	4,9	2,4	0,3	1,4	0,5	2,8	1,3	10,0	3,2	2,5
	500	5,0	3,2	0,3	1,7	0,5	5,1	1,2	20,0	3,8	3,1
	1000	5,0	3,7	0,3	1,7	0,1	5,7	1,2	20,0	3,7	4,1
0,2	100	4,8	2,1	0,4	1,1	0,5	2,2	1,5	3,5	2,3	2,1
	300	5,0	2,5	0,3	1,6	0,5	3,3	1,3	10,0	3,3	2,7
	500	5,0	3,2	0,2	1,7	0,5	5,0	1,3	20,0	3,5	3,3
	1000	5,0	4,1	0,3	1,7	0,1	6,3	1,2	20,0	3,5	3,6

Таблица 3.3 – продолжение

λ	Эпохи	Функция потерь									
		Энд-рюса	Уэлша	Хьюбера	Рамсея	«Fair»	Коши	Биквадратная Тьюки	Geman-McCluer	Мешалкина	Charbonnier
0,25	100	2,3	1,5	0,4	0,9	0,5	1,1	1,5	3,5	2,1	1,5
	300	5,0	2,5	0,3	1,6	0,5	2,7	1,3	10,0	3,3	2,4
	500	5,0	2,9	0,3	1,7	0,1	4,5	1,2	20,0	3,5	3,0
	1000	5,0	3,7	0,3	1,7	0,1	6,9	1,3	20,0	3,3	4,4
0,3	100	2,4	1,5	0,5	0,9	0,5	1,1	1,6	4,0	1,9	1,5
	300	4,7	2,2	0,3	1,6	0,5	2,4	1,3	9,5	2,8	2,2
	500	5,0	2,7	0,3	1,6	0,5	3,6	1,3	20,0	3,6	2,8
	1000	5,0	4,4	0,4	1,4	0,5	6,9	1,3	20,0	2,8	3,8
0,35	100	2,9	1,6	0,4	1,0	0,5	1,3	1,5	5,0	2,0	1,7
	300	5,0	2,4	0,3	1,6	0,5	3,1	1,3	10,0	3,3	2,4
	500	5,0	3,1	0,3	1,7	0,5	5,2	1,3	20,0	3,8	3,2
	1000	5,0	3,7	0,3	1,5	0,5	6,9	1,3	20,0	3,8	4,5
0,4	100	2,5	0,5	0,4	1,0	0,5	1,1	1,5	4,0	2,2	1,4
	300	5,0	2,4	0,3	1,6	0,5	3,0	1,3	10,0	3,5	2,5
	500	5,0	2,8	0,3	1,6	0,5	4,1	1,2	20,0	3,8	2,9
	1000	5,0	4,6	0,4	1,5	0,1	6,9	1,4	20,0	3,0	3,9

Для остальных рассмотренных функций наблюдается зависимость значения параметра только от числа эпох обучения. Так, в случае функций потерь Эндрюса и Уэлша с ростом числа эпох значение параметра, при котором достигается наилучшая точность работы сети, возрастает. В случае функции потерь «Fair» наилучшее значение параметра с ростом числа эпох обучения чаще всего убывает (либо не изменяется при  $\lambda = 0,30$  и  $\lambda = 0,35$ ). Для ИНС с функциями потерь Коши, Geman-McCluer, Мешалкина и Charbonnier в целом характерен рост наилучшего значения параметра при увеличении числа эпох обучения нейронной сети. Для нейронной сети с биквадратной функцией потерь Тьюки характерно убывание значения  $\beta$  с ростом числа эпох.

На рис. 3.2 продемонстрирована полученная зависимость точности классификации от значения параметра  $\beta$  для нейронной сети с функции потерь Charbonnier при различном числе эпох обучения и доли засоряющих наблюдений  $\lambda = 0,25$ . Нетрудно заметить, что с ростом числа эпох обучений возрастает и наилучшее значение параметра функции, например, для 100 эпох  $\beta = 1,5$ , а для 500 эпох  $\beta = 3,0$ . Однако слишком большие значения параметра приводят к снижению точности работы нейронной сети.

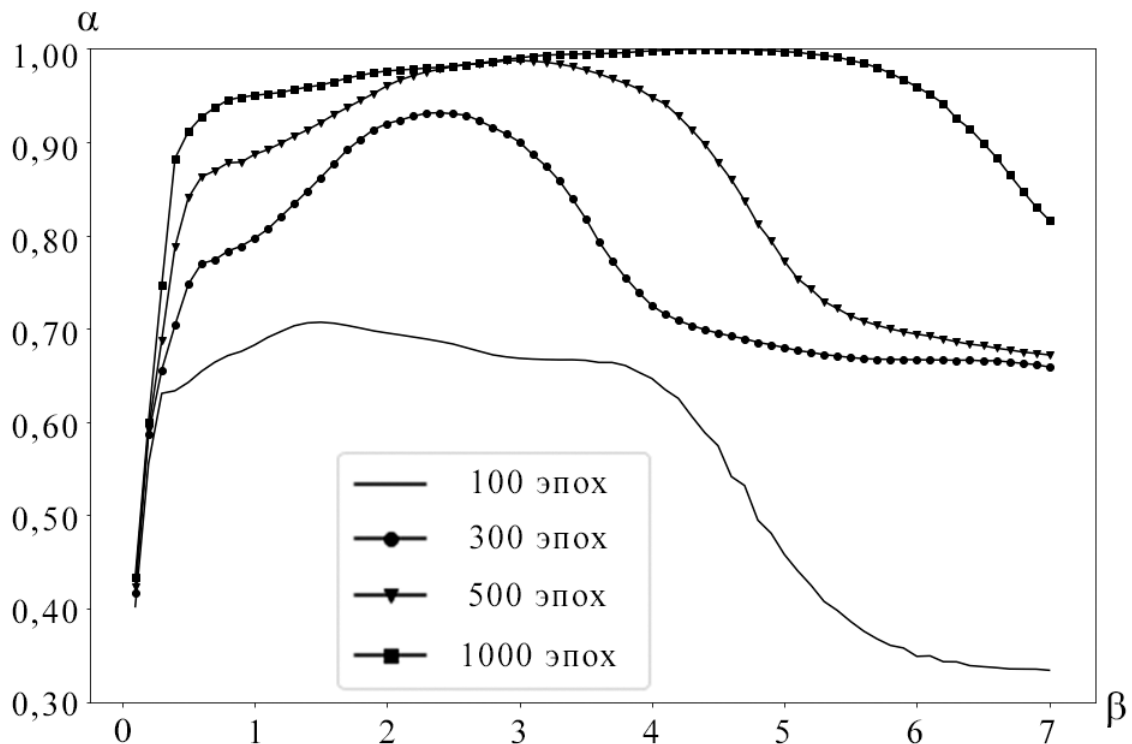


Рисунок 3.2 — Зависимость наилучшего значения параметра функции потерь Charbonnier от числа эпох обучения

В табл. 3.4 приводятся интервалы значений параметра  $\beta$  для ИНС с рассматриваемыми робастными функциями потерь, на которых точность классификации была наилучшей. На основе полученных результатов можно дать следующие рекомендации относительно выбора значения параметра робастных функций потерь для ИНС с одним скрытым слоем. Для функции потерь Хьюбера значение параметра  $\beta \geq 1$  выбирать не следует – в этом случае она будет полностью совпадать с квадратичной функцией. Для функций потерь Эндрюса, Рамсея, Коши, Charbonnier и Мешалкина значение параметра следует постепенно увеличивать с ростом числа эпох, однако следует помнить о том, что слишком большие значения  $\beta$  могут негативно сказаться на точности классификации.

Таблица 3.4 — Рекомендованные интервалы значений параметров робастных функций потерь

Функция потерь	100 эпох	101–300 эпох	301–500 эпох	>500 эпох
Эндрюса	[1,5; 2,9]	[4,7; 5,0]	[4,7; 5,0]	[4,7; 5,0]
Уэлша	[0,5; 2,1]	[0,5; 2,1]	[2,3; 3,2]	[3,5; 4,6]
Хьюбера	[0,4; 0,55]	[0,2; 0,35]	[0,2; 0,35]	[0,2; 0,35]
Рамсея	[0,6; 1,0]	[0,8; 1,7]	[0,8; 1,7]	[0,8; 1,7]
«Fair»	[0,5; 1,5]	[0,5; 1,5]	[0,1; 0,5]	[0,1; 0,5]
Биквадратная Тьюки	[1,5; 1,7]	[1,3; 1,6]	[1,2; 1,5]	[1,2; 1,4]
Коши	[0,7; 1,3]	[2,4; 3,3]	[3,6; 5,2]	[5,2; 6,9]
Geman-McCluer	[2,0; 5,0]	[9,5; 10]	>20	>20
Charbonnier	[1,2; 1,7]	[2,2; 2,7]	[2,8; 3,3]	[2,8; 3,8]
Мешалкина	[1,4; 2,2]	[1,8; 3,3]	[2,2; 3,8]	[3,8; 4,9]

Так, для функции потерь Эндрюса не следует выбирать значения параметра больше 5,0, для функции потерь Коши не следует выбирать значения параметра больше 7,0, для функции потерь Уэлша и Charbonnier – больше 4,5-4,6, для функции потерь Мешалкина – больше 4,0, для функции потерь Рамсея – больше 1,7. Для функции потерь «Fair» и для биквадратной функции потерь Тьюки, наоборот, с ростом числа эпох обучения следует уменьшить значение  $\beta$ . Что касается функции потерь Geman-McCluer, то с ростом числа эпох наилучшее значение параметра возрастает достаточно резко – при малом числе эпох (до 300) можно выбирать  $\beta$  на интервале [2,0; 10,0], а с ростом числа эпох можно рассмотреть значения параметра больше 20. Следует отметить,



что данные рекомендации справедливы для робастных ИНС с одним скрытым слоем, однако значения параметров на интервалах из табл. 3.4 можно рекомендовать использовать в качестве начального приближения и при построении ИНС с большим числом скрытых слоев.

После того, как была выполнена настройка робастных нейронных сетей, проводилось исследование их устойчивости. При этом точность работы всех построенных сетей сравнивалась с точностью работы классической ИНС, имеющей такую же архитектуру. Перейдем к рассмотрению результатов исследований.

### **3.3 Исследование устойчивости робастных искусственных нейронных сетей при различной доле засоряющих наблюдений**

На данном этапе исследований рассматривалась точность работы робастных нейронных сетей в зависимости от доли выбросов в наборе данных. Для этого при построении моделей использовались значения параметра  $\beta$ , для которых на предыдущем этапе исследования были получены наилучшие значения метрики  $\alpha$ . Значение доли засоряющих наблюдений  $\lambda$  варьировалось в пределах от 0,05 до 0,40 с шагом в 0,05. В табл. 3.5 приводятся полученные значения метрики  $\alpha$  при 100, 300 и 500 эпохах обучения. При большем числе эпох точность работы всех сетей резко возрастала и в отдельных случаях достигала значений 98-99%, что однозначно говорит о переобучении. При сравнении точности работы построенных сетей рассматривать такие случаи нецелесообразно.

Анализируя полученные результаты, можно сделать вывод о том, что при сравнительно малом числе эпох (100) даже при небольших значениях доли засоряющих наблюдений ( $\lambda = 0,05$  и  $\lambda = 0,10$ ) точность работы, а также скорость обучения робастных нейронных сетей сопоставимы с показателями классической нейронной сети. Кроме того, можно заметить, что при малой доле засоряющих наблюдений ( $\lambda = 0,05$ ) даже с увеличением числа эпох до 300 робастные нейронные сети работают незначительно точнее классической (на 1,5% в среднем), а если увеличить число эпох до 500, то робастные сети начинают работать точнее в среднем на 4,2%.

Таблица 3.5 — Точность классификации при различных значениях доли выбросов

λ	Эпохи	Функция потерь										
		Энд-рюса	Уэлша	Хьюбера	Рамсея	«Fair»	Коши	Биквадратная Тьюки	Geman-McCluer	Мешалкина	Charbonnier	Квадратичная
0,05	100	0,766	0,770	0,767	0,771	0,769	0,769	0,770	0,767	0,770	0,769	0,750
	300	0,912	0,913	0,907	0,904	0,904	0,913	0,906	0,913	0,906	0,913	0,892
	500	0,966	0,968	0,943	0,950	0,951	0,968	0,945	0,967	0,945	0,967	0,914
0,10	100	0,723	0,723	0,719	0,722	0,722	0,723	0,721	0,723	0,721	0,723	0,716
	300	0,894	0,898	0,832	0,839	0,832	0,898	0,821	0,894	0,825	0,898	0,778
	500	0,927	0,956	0,913	0,916	0,898	0,957	0,912	0,955	0,915	0,956	0,835
0,15	100	0,710	0,713	0,715	0,715	0,715	0,712	0,714	0,716	0,716	0,711	0,698
	300	0,862	0,866	0,848	0,846	0,844	0,866	0,840	0,865	0,848	0,866	0,778
	500	0,910	0,943	0,906	0,902	0,888	0,943	0,899	0,943	0,908	0,943	0,824
0,20	100	0,682	0,680	0,677	0,680	0,680	0,680	0,677	0,680	0,678	0,681	0,672
	300	0,795	0,799	0,786	0,787	0,779	0,798	0,779	0,798	0,787	0,798	0,726
	500	0,887	0,912	0,891	0,889	0,860	0,913	0,879	0,913	0,883	0,913	0,789
0,25	100	0,707	0,707	0,693	0,700	0,701	0,707	0,695	0,706	0,696	0,701	0,673
	300	0,927	0,931	0,873	0,883	0,871	0,931	0,867	0,931	0,874	0,931	0,766
	500	0,971	0,987	0,940	0,947	0,940	0,986	0,941	0,986	0,943	0,897	0,852

Таблица 3.5 – продолжение

λ	Эпохи	Функция потерь										
		Энд-рюса	Уэлша	Хьюбера	Рамсея	«Fair»	Коши	Биквадратная Тьюки	Geman-McCluer	Мешалкина	Charbonnier	Квадратичная
0,30	100	0,690	0,691	0,673	0,681	0,684	0,691	0,677	0,691	0,676	0,691	0,663
	300	0,874	0,876	0,851	0,863	0,855	0,877	0,856	0,877	0,847	0,875	0,764
	500	0,939	0,955	0,922	0,927	0,917	0,955	0,922	0,946	0,924	0,955	0,848
0,35	100	0,696	0,696	0,677	0,687	0,689	0,696	0,679	0,696	0,678	0,695	0,656
	300	0,885	0,895	0,841	0,842	0,824	0,895	0,829	0,892	0,826	0,895	0,747
	500	0,925	0,960	0,908	0,913	0,900	0,960	0,901	0,959	0,899	0,960	0,818
0,40	100	0,693	0,694	0,690	0,692	0,691	0,694	0,690	0,694	0,691	0,694	0,670
	300	0,863	0,870	0,849	0,857	0,828	0,869	0,831	0,864	0,841	0,871	0,743
	500	0,937	0,954	0,907	0,918	0,899	0,954	0,899	0,952	0,909	0,954	0,803

С дальнейшим ростом числа эпох и доли засоряющих наблюдений разница в точности работы классической и робастных ИНС становится все более существенной. Так, при классификации сильно зашумленных данных (доля выбросов  $\lambda = 0,35$  и  $\lambda = 0,40$ ) после 500 эпох обучения робастные сети начинают работать точнее в среднем на 11%, а в отдельных случаях позволяют получить выигрыш в точности до 15% (ИНС с функциями потерь Уэлша и Charbonnier).

Из табл. 3.5 видно, что все робастные нейронные сети можно условно разделить на два множества. Первое будет включать в себя сети с функциями потерь Уэлша, Коши, Geman-McCluer и Charbonnier, второе – сети с функциями потерь Мешалкина, Хьюбера, Рамсея, «Fair» и биквадратной функцией потерь Тьюки. Между собой сети из первого и второго множества при всех значениях  $\lambda$  сопоставимы по точности работы, однако значение метрики  $\alpha$  для сетей, входящих во второе множество, в среднем на 3,5% ниже, чем для сетей из первого множества. Отдельно можно выделить функцию потерь Эндрюса – значение точности работы нейронной сети, построенной с помощью этих функций, выше, чем для функций из первого множества, но ниже, чем для функций из второго.

Помимо всего прочего, полученные результаты позволяют отчетливо увидеть ухудшение точности работы всех ИНС – и классической, и робастных – при доле засоряющих наблюдений  $\lambda = 0,20$ , а также резкое улучшение точности при доле выбросов  $\lambda = 0,25$  и последующее ее ухудшение с ростом значения доли выбросов. Данный «парадокс» носит локальный характер – его можно объяснить конфигурацией выборок, а именно тем, насколько различные классы объектов удалены друг от друга. При этом расстояние между классами можно вычислить по формуле

$$\bar{d}(q_i, q_j) = \sqrt{\sum_k (\bar{q}_{ik} - \bar{q}_{jk})^2}, \quad (3.3)$$

где  $\bar{q}_{ik}$  – геометрический центр точек для признака  $k$  – вычисляется следующим образом:

$$\bar{q}_{ik} = \frac{1}{|q_i|} \sum_{X_m \in q_i} x_{mk}.$$

В данном случае моделирование выбросов производилось только для 3 и 4 признаков объектов, поэтому точки, соответствующие объектам, рассматривались в двумерном Евклидовом пространстве ( $k = 3,4$ ). Значение расстояния

(3.3) было получено для каждой из 100 выборок при значениях доли выбросов  $\lambda = 0,20$  и  $\lambda = 0,25$ . Вычисленные значения затем усреднялись. Полученные результаты представлены в табл. 3.6.

Таблица 3.6 — Расстояния между классами объектов при различной доле засоряющих наблюдений

$\lambda$	Расстояние между классами		
	1 и 2 класс	2 и 3 класс	1 и 3 класс
0,20	3,098	1,399	4,484
0,25	2,946	1,686	4,624

Из таблицы нетрудно заметить, что с ростом доли выбросов расстояние между первым и вторым классами уменьшилось на 4,9%, а между первым и третьим классами увеличилось на 3%. В то же время расстояние между вторым и третьим классами увеличилось на 20,5%. Исследования показали, что при  $\lambda = 0,20$  ИНС гораздо чаще «путают» объекты этих классов между собой, чем при  $\lambda = 0,25$ , что и приводит к такому резкому перепаду в точности работы сетей. Фактически ситуация, когда значение доли засоряющих наблюдений равно 0,20, в данном случае эквивалентна большей степени зашумления выборки.

### 3.4 Исследование устойчивости робастных искусственных нейронных сетей при различном числе объектов

На данном этапе исследований изучалось изменение точности работы построенных нейронных сетей при различном объеме анализируемых данных. Поскольку набор «Ирисы Фишера» является достаточно малым, а использование технологии bootstrap [66] привело бы к внесению дополнительного шума в данные, на этом этапе использовались наборы данных, полученные с помощью генератора *make\_blobs* данных языка Python 3.6 [72]. Этот инструмент позволяет генерировать облака нормально распределенных данных, которые можно использовать в качестве данных для классификации.

Используемый генератор данных принимает на вход следующие параметры: количество объектов в каждом кластере, количество признаков объектов, координаты центра для каждого кластера, а также среднеквадратическое отклонение признаков объектов. Конфигурация сгенерированных наборов данных была аналогична набору «Ирисы Фишера»: объекты делились на три

одинаковых по размеру кластера, каждый из которых соответствовал отдельному классу, объекты описывались четырьмя признаками. Задавались следующие координаты центров кластеров:  $(1,0; 1,0; 2,0; 1,0)$  – для первого кластера;  $(4,0; 4,0; 4,0; 4,0)$  – для второго кластера;  $(2,5; 4,5; 2,5; 2,0)$  – для третьего кластера. Среднеквадратическое отклонение для всех признаков объектов равнялось 0,5. Наиболее различительными признаками являлись признаки 3 и 4, моделирование выбросов выполнялось в соответствии с (3.1), (3.2). Уровни шума, которым соответствовали дисперсии  $\sigma_{ij}^2$ , были выбраны такие же, как на предыдущих этапах исследований. Разбиение набора данных на обучающую и тестовую выборки также проводилось аналогично предыдущим этапам.

Рассматривались наборы данных из 150, 450, 600 и 900 объектов при доле выбросов  $\lambda = 0,25$ . Для робастных функций потерь значения параметра  $\beta$  выбирались согласно данным ранее рекомендациям. Точность работы нейронных сетей, как и на предыдущем этапе, фиксировалась после 100, 300 и 500 эпох обучения. Результаты данного этапа исследований представлены в табл. 3.7 ( $|X|$  – объем набора данных,  $h$  – число эпох).

Анализируя представленные в табл. 3.7 значения метрики  $\alpha$ , можно сказать, что в целом с ростом числа объектов в наборе данных скорость обучения робастных нейронных сетей падает. Хотя увеличение числа объектов со 150 до 450 зачастую не сказывается на точности работы нейронных сетей негативным образом, дальнейшее увеличение числа объектов приводит к снижению точности работы нейронных сетей. Так, по сравнению с набором, состоящим из 450 объектов, точность классификации для набора из 600 объектов снизилась примерно на 1%, а для набора из 900 объектов – на 2%. Это объясняется тем, что с ростом объема данных количество нетипичных наблюдений также растет.

По сравнению с классической ИНС наибольший выигрыш в точности дают ИНС с функциями потерь Уэлша, Коши и Geman-McCluer, наименьший – с функцией потерь Рамсея и биквадратной функцией потерь Тьюки. Кроме того, можно отметить слишком высокие значения метрики  $\alpha$  для функций потерь Charbonnier, Geman-McCluer, Эндрюса и Уэлша при 150 объектах в наборе. Такие значения говорят о переобучении нейронных сетей, в которых использовались указанные робастные функции потерь, то есть в данном случае для их обучения требуется менее 100 эпох.

Таблица 3.7 — Точность классификации при различном числе объектов в наборе данных

$ X $	$h$	Энд-рюса	Рамсея	"Fair"	Уэлша	Хьюбера	Коши	Биквадратная Тьюки	Geman-McCluer	Мешалкина	Charbonnier	Квадратичная
		$\beta = 4,7$	$\beta = 1,2$	$\beta = 0,4$	$\beta = 2,5$	$\beta = 0,2$	$\beta = 4,5$	$\beta = 1,3$	$\beta = 20,0$	$\beta = 3,3$	$\beta = 2,8$	–
150	100	0,996	0,873	0,914	0,996	0,830	0,884	0,838	0,783	0,846	0,978	0,742
	300	0,980	0,895	0,914	0,999	0,909	0,999	0,888	0,999	0,893	0,999	0,859
	500	0,990	0,937	0,952	0,997	0,956	0,999	0,927	1,000	0,934	0,999	0,910
450	100	0,672	0,668	0,668	0,709	0,670	0,865	0,670	0,920	0,670	0,761	0,670
	300	0,962	0,908	0,922	0,971	0,923	0,976	0,889	0,977	0,895	0,973	0,876
	500	0,988	0,961	0,967	0,991	0,974	0,992	0,957	0,993	0,959	0,991	0,941
600	100	0,669	0,676	0,675	0,670	0,675	0,725	0,681	0,786	0,680	0,677	0,675
	300	0,953	0,918	0,930	0,959	0,939	0,964	0,913	0,964	0,919	0,961	0,873
	500	0,974	0,958	0,963	0,980	0,965	0,982	0,955	0,983	0,956	0,980	0,930
900	100	0,670	0,684	0,680	0,669	0,685	0,680	0,696	0,700	0,695	0,670	0,683
	300	0,935	0,891	0,902	0,954	0,913	0,968	0,885	0,971	0,887	0,961	0,880
	500	0,964	0,928	0,937	0,974	0,946	0,983	0,922	0,985	0,926	0,979	0,912

### Выводы по главе 3

Основные результаты, полученные в данной главе, можно сформулировать следующим образом:

1. Для каждой робастной функции потерь определены наилучшие значения внутренних параметров, т.е. такие значения, при которых точность работы нейронной сети будет наиболее высокой. По результатам исследования для всех робастных нейронных сетей сформулированы рекомендации относительно выбора значения параметра функции потерь.

2. Последствием ряда вычислительных экспериментов проведено исследование устойчивости робастных нейронных сетей при различной доле засоряющих наблюдений в наборе данных. Отмечено, что при малом значении доли засоряющих наблюдений робастные модели сопоставимы по точности с классической. При работе с сильно зашумленными данными робастные нейронные сети дают значительный выигрыш в точности.

3. Исследована устойчивость робастных ИНС при классификации наборов данных различного объема. Результаты исследований на данном этапе позволили сделать вывод о том, что с увеличением количества объектов в анализируемом наборе данных точность работы робастных сетей незначительно снижается (на 1-2%), однако они по-прежнему работают гораздо точнее классической модели.



## Глава 4 Программный модуль для построения робастных нейронных сетей

В данной главе описывается разработанный программный модуль для построения робастных нейронных сетей «RobustNN» – рассматриваются его архитектура, основные методы и режимы работы. Модуль позволяет построить и обучить модель робастной нейронной сети с произвольной простой архитектурой, оценить точность работы модели, а также сохранить модель для дальнейшего использования.

### 4.1 Особенности реализации и системные требования

Программный модуль «RobustNN» предназначен для построения нейронных сетей прямого распространения (однослойный и многослойный перцептрон), которые могут использоваться для решения различного рода прикладных задач. Для обучения нейронной сети в нем реализован рассмотренный в главе 2 алгоритм обратного распространения ошибки, причем как робастная его версия, так и классическая.

Разработанный программный продукт не предполагает наличие графического интерфейса, поскольку по сути является библиотекой, предоставляющей необходимый набор методов и классов. Данный модуль реализован на языке Python 3.6, что обеспечивает его кроссплатформенность по аналогии с существующими программными продуктами. Еще одним преимуществом данного языка программирования является наличие большого количества готовых модулей и библиотек, предназначенных, например, для работы с файловой системой компьютера, для быстрой и удобной реализации математических операций, для генерации случайных величин.

Так, для инициализации массивов и выполнения векторно-матричных операций использовался модуль *numpy* [84], в котором реализованы высокоуровневые методы для работы с многомерными массивами. Для реализации математических функций, таких как экспонента, синус, взятие квадратного корня использовался модуль *math* [81]. Для реализации сохранения и загрузки модели использовались методы модулей *json* (работа с файлами в формате .json) [77] и *os* (работа непосредственно с файловой системой компьютера) [86]. Для генерации случайных величин при инициализации весов нейронной сети использовался модуль *random* [90]. Из перечисленных модулей все, кроме *numpy*,

являются стандартными и не требуют дополнительной установки. Способы установки модуля *numru* описаны в [74].

Выделение оперативной памяти под использующиеся массивы происходит динамически при первоначальной инициализации нейронной сети – значит, ее объем, необходимый для работы программы, определяется заданной пользователем архитектурой нейронной сети, а также объемом данных, используемых при решении конкретной задачи. Размерности массивов определяются автоматически, исходя из архитектуры реализуемой нейронной сети. Объем памяти, требующийся для сохранения обученной модели, также будет определяться архитектурой нейронной сети, но само по себе сохранение модели не является обязательным.

Следует отметить также, что модуль «RobustNN» не предполагает реализацию метода для считывания данных, поскольку универсальным такой метод сделать фактически невозможно. Вследствие этого предполагается, что считывание нужных данных будет реализовано конечным пользователем самостоятельно с учетом формата файлов с данными (например, это могут быть текстовые [88], бинарные файлы [56] или файлы формата *.csv* [63]). На вход нейронной сети данные подаются в виде четырех массивов: признаки объектов обучающей выборки, классы объектов обучающей выборки, признаки объектов тестовой выборки, классы объектов тестовой выборки. При этом класс каждого объекта должен быть представлен в виде целочисленного списка, в котором элемент, соответствующий номеру класса объекта, равен 1, а остальные элементы равны 0 (например, при решении задачи бинарной классификации первый класс будет представляться как [1, 0], второй – как [0, 1]).

Директория программного модуля «RobustNN» называется *robustnn* содержит в себе следующие файлы: *RobustNN.py* – основной файл с методами модуля, *\_\_init\_\_.py* – служебный файл инициализации пакета, *\_\_pycache\_\_* – служебная директория, содержащая в себе файлы байт-кода модуля. Минимальные системные требования для работы с модулем: IBM PC-совместимый ПК с ОЗУ не менее 1 Гб, ОС Windows 7 или более поздней версии, либо ОС Linux семейства Debian, либо ОС Linux семейства Red Hat, интерпретатор языка Python 3.6, установленный модуль *numru*. Объем основного файла составляет 15,8 КБ (16,0 КБ занятого дискового пространства). Объем директории полностью – 38,6 КБ (44,0 КБ занятого дискового пространства).

## 4.2 Архитектура и основные методы

Программный модуль «RobustNN» включает в себя два класса: RobustLayer, обеспечивающий формирование каждого отдельного слоя нейронной сети, и RobustNet, использующийся для связи слоев модели между собой. На рис. 4.1 приведена диаграмма классов модуля, в которой отражены все атрибуты и основные методы классов (опущены, например, методы для вычисления значений различных функций потерь, set- и get-методы [70]).

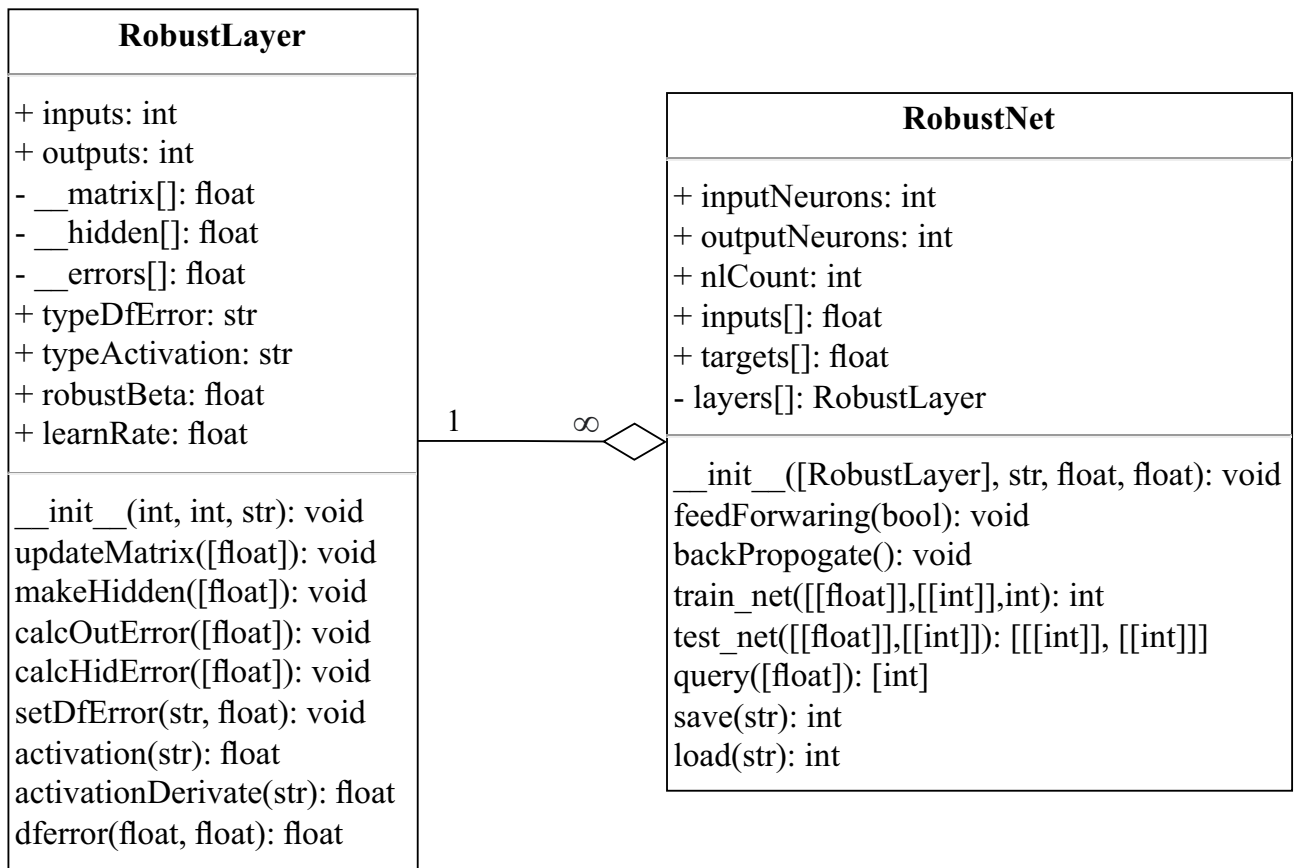


Рисунок 4.1 — Диаграмма классов модуля «RobustNN»

Класс RobustLayer включает в себя следующие атрибуты: inputs (количество входов на слое), outputs (количество выходов на слое), \_\_matrix (матрица весов слоя), \_\_hidden (матрица значений функции активации), \_\_errors (матрица ошибок), typeDfError (вид функции потерь), typeActivation (вид функции активации), robustBeta (значение параметра функции потерь), learnRate (коэффициент скорости обучения). Атрибуты matrix, hidden и errors являются приватными и недоступны пользователю.

Класс `RobustNet` включает в себя следующие атрибуты: `inputNeurons` (количество нейронов на входном слое), `outputNeurons` (количество нейронов на выходном слое), `nlCount` (число слоев), `inputs` (значения признаков объекта на входе нейронной сети), `targets` (значения выходных нейронов в соответствии с классом объекта), `__layers` (список слоев нейронной сети). Атрибут `__layers` является приватным и является списком типа `RobustLayer` – таким образом, имеет место агрегация класса `RobustNN` типа «многие к одному».

Несмотря на то что в языке Python 3.6 фактически не реализован механизм, позволяющий объявлять приватные методы, доступ к которым можно было бы получить только внутри класса [87], условно методы обоих классов программного модуля «`RobustNN`» можно разделить на два типа: предназначенные непосредственно для вызова пользователем и внутренние. К первой группе относятся, например, конструктор слоя и модели, методы для запуска обучения, сохранения и загрузки модели. Во вторую группу входят методы, в которых реализованы алгоритм обучения нейронной сети, вычисление различных функций потерь и функций активации. Рассмотрим подробнее методы этих двух классов, которые предназначены для вызова пользователем.

В классе `RobustLayer` для вызова пользователем предназначен только метод конструктора. Он принимает на вход следующие три параметра: количество входов на слое, количество выходов на слое, вид функции активации. На данный момент в программном модуле предусмотрено две функции активации: сигмоида (значение соответствующего параметра конструктора "SIG") и гиперболический тангенс (значение соответствующего параметра конструктора "TANH"). По умолчанию задается сигмоида. Ввиду особенностей реализации на каждом слое требуется хранить вид и параметр функции потерь (они задаются при инициализации модели). Для их задания используется внутренний метод `setDfError()`, принимающий на вход два параметра (первый – вид функции потерь, второй – значение параметра функции потерь). Помимо робастных функций потерь, выбранных для исследования в разделе 2.1, предусмотрена также возможность выбора квадратичной функции потерь (значение второго параметра метода в этом случае должно быть равно нулю). По умолчанию задается функция потерь Хьюбера с параметром  $\beta = 0,5$ . В табл. 4.1 приводятся значения первого параметра метода `setDfError()`, соответствующие различным функциям потерь.

Таблица 4.1 — Соответствие функций потерь и значений первого параметра метода `setDfError()`

Функция потерь	Значение параметра
Квадратичная	SQR
Хьюбера	HUB
Биквадратная Тьюки	BTUK
Эндрюса	ANDR
Рамсея	RAM
Коши	CAU
Geman-McCluer	GM
Charbonnier	CBNR
Мешалкина	MESH
Уэлша	WEL
"Fair"	FAIR

Здесь следует также отметить такую особенность: создавать входной слой нейронной сети отдельно не требуется, поскольку для него весовых коэффициенты не хранятся – число нейронов на нем задается в количестве входов на скрытом слое. Кроме того, число выходов на слое  $i$  должно совпадать с числом входов на слое  $i + 1$ , иначе при запуске программы будет выдано сообщение об ошибке.

Среди методов класса `RobustNet` для вызова пользователем предназначены конструктор класса, методы для запуска обучения нейронной сети и ее работы на тестовой выборке, а также функции сохранения и загрузки модели. Конструктор класса принимает на вход такие параметры, как список слоев нейронной сети, вид функции потерь, параметр функции потерь. В самом конструкторе происходит инициализация модели: определяется количество входных и выходных нейронов, число слоев, все переданные на вход слои объединяются в модель.

Метод `train_net()` предназначен для запуска обучения нейронной сети. В качестве параметров в него необходимо передать массив с признаками объектов обучающей выборки, массив с классами объектов обучающей выборки, а также число эпох, в течение которых будет выполняться обучение нейронной сети. При успешном обучении модели метод возвращает значение 0, при возникновении ошибки – значение -1.

Метод `test_net()` предназначен для запуска работы обученной нейронной сети на тестовой выборке. В качестве параметров в него необходимо передать массив с признаками объектов тестовой выборки и массив с классами объектов тестовой выборки. В качестве результата метод возвращает двумерный массив, в котором первая компонента  $i$ -го элемента – это ответ нейронной сети, полученный для  $i$ -го объекта тестовой выборки, а вторая компонента – реальный класс данного объекта. Чтобы получить ответ нейронной сети для одного конкретного объекта, можно использовать метод `query()`, принимающий на вход набор признаков объекта.

Методы `save()` и `load()` используются для сохранения и загрузки обученной модели соответственно. В качестве параметра они принимают путь к директории, где необходимо сохранить или откуда необходимо загрузить файлы модели (по умолчанию задается текущая директория проекта). Модель хранится в виде файла в формате `.json`, а также обычных текстовых файлов. В текстовых файлах хранятся внутренние параметры нейронной сети, такие как значения весовых коэффициентов на каждом слое, значение функции потерь и функции активации для каждого нейрона. В файле формата `.json` хранится общая информация о нейронной сети (число слоев, число нейронов на входном слое и на выходном), а также детальная информация о каждом слое (число входов и выходов слоя, функция потерь, параметр функции потерь, а также путь к текстовым файлам, соответствующим данному слою). Пример структуры файла `.json` для ИНС с одним скрытым слоем представлен на рис. 4.2.

Помимо методов, реализованных в рамках этих двух классов, вне классов в модуле также реализованы функции для расчета метрик  $\alpha(q_i)$  (1.22),  $p(q_i)$  (1.23) и  $r(q_i)$  (1.24) – `accuracy()`, `precision()` и `recall()`, соответственно. В качестве параметров они принимают список истинных классов объектов выборки и классов, полученных с использованием модели, а также класс, для которого выполняется расчет метрики. Эти функции возвращают значение метрики для заданного класса. Кроме того, функция `accuracy()` допускает запуск без указания класса в качестве параметра – в этом случае будет выполнен расчет по всем классам в целом (метрика  $\alpha$ ).

Исходя из выше сказанного, можно сделать вывод, что разработанный модуль может быть использован в трех режимах: режим обучения модели, режим оценки качества модели, режим получения ответа для конкретного объекта.

```

{
  "network": {
    "layer_count": 2,
    "input_neurons": 9,
    "output_neurons": 3
  },

  "layer1": {
    "inputs": 9,
    "outputs": 4,
    "dferror": "BTUK",
    "activation": "SIG",
    "beta": 1.5,
    "fmatrix": "carbon_new_btuk/layer1_matrix",
    "ferrors": "carbon_new_btuk/layer1_errors",
    "fhidden": "carbon_new_btuk/layer1_hidden"
  },

  "layer2": {
    "inputs": 4,
    "outputs": 3,
    "dferror": "BTUK",
    "activation": "SIG",
    "beta": 1.5,
    "fmatrix": "carbon_new_btuk/layer2_matrix",
    "ferrors": "carbon_new_btuk/layer2_errors",
    "fhidden": "carbon_new_btuk/layer2_hidden"
  }
}

```

Рисунок 4.2 — Пример структуры .json-файла для модели ИНС с одним скрытым слоем

Рассмотрим работу в каждом из этих режимов при решении задачи классификации подробнее.

### 4.3 Использование программного модуля «RobustNN» для работы в различных режимах

Будем рассматривать использование модуля «RobustNN» для работы в различных режимах при решении задачи классификации на примере модели нейронной сети, имеющей архитектуру, представленную на рис. 1.3. Пусть в качестве функции потерь выбрана функция потерь Уэлша с параметром  $\beta = 1,2$ . Будем полагать также, что классы объектов сбалансированы по размеру. Кроме того, предполагается, что считывание обучающей и тестовой выборки в массивы  $X\_learn$ ,  $y\_learn$ ,  $X\_test$ ,  $y\_test$  конечный пользователь реализует удобным ему способом. Здесь следует отметить, что для начала работы независимо от

режима необходимо выполнить импорт классов и методов модуля с использованием инструкций `import` и `from` [31].

Первый режим работы с программным модулем – это режим обучения модели. На рис. 4.3 представлена диаграмма последовательностей, отражающая в общем виде порядок действий пользователя при использовании программного модуля в данном режиме.



Рисунок 4.3 — Диаграмма последовательностей: работа в режиме обучения модели

Чтобы организовать работу в режиме обучения модели, во-первых, необходимо определить слои нейронной сети с использованием конструктора класса `RobustLayer`. Поскольку в данном случае у модели имеется всего один скрытый слой, программно необходимо задать два слоя: у первого (соответствующего скрытому и неявно указывающего на число нейронов входного слоя) будет 4 входа и 4 выхода, у второго (соответствующего выходному слою) – 4 входа и 3 выхода. После этого созданные слои нужно объединить в модель, указав в



качестве функции потерь робастную функцию потерь Уэлша и задав ее параметр  $\beta = 1,2$ , а затем вызвать метод *train\_net()* класса RobustNet для запуска обучения модели. После этого разумным будет оценить точность работы модели – поскольку классы объектов в данном случае сбалансированы по размеру, можно использовать функцию *accuracy()* без указания класса объектов.

По окончании процедуры обучения полученную модель рекомендуется сохранить. Следует также обратить внимание на то, что в рассматриваемом примере задается конкретное значение параметра  $\beta$ . Если такое значение заранее неизвестно, то при реализации процедуры его выбора для каждого нового  $\beta$  необходимо создавать слои и саму модель заново. Программный код, реализующий описанный порядок действий, представлен в приложении А (рис. А.1).

Теперь рассмотрим использование модуля «RobustNN» для организации работы в режиме оценки качества построенной модели. Предположим, что модель была ранее обучена и сохранена. Тогда необходимо выполнить следующий набор действий: загрузить модель, указав нужную директорию, запустить работу модели на нужной выборке, выполнить расчет нужных метрик для оценки точности модели. Описанный порядок действий отражен на следующей диаграмме последовательностей (рис. 4.4).

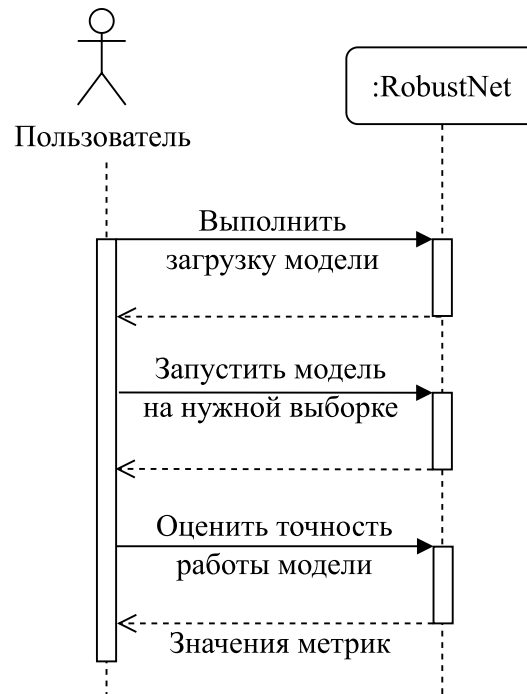


Рисунок 4.4 — Диаграмма последовательностей: работа в режиме оценки качества модели

В общем случае значения метрик рассчитываются для каждого класса исследуемых объектов в отдельности, а затем при необходимости вычисляется их среднее. В приложении А (рис. А.2) продемонстрирован пример оценки точности работы модели с помощью метрики  $r$ , вычисляемой на основе значений  $r(q_i)$ . В данном случае все объекты делятся на три класса, поэтому покласовое вычисление метрики  $r(q_i)$  реализовано без использования цикла.

Перейдем к рассмотрению ситуации, когда имеется уже обученная модель и необходимо получить решение для какого-то конкретного объекта – на практике это достаточно частое явление, когда модель обучили на имеющихся данных, а затем используют для анализа новых данных, поступающих в систему. Очевидно, что в таком случае необходимо выполнить загрузку уже обученной модели из указанной директории, а затем вызвать метод, позволяющий получить ответ нейронной сети для конкретного объекта. Описанный порядок действий продемонстрирован на диаграмме последовательностей, изображенной на рис. 4.5. Программный код, реализующий работу в данном режиме, приводятся в приложении А (рис. А.3).

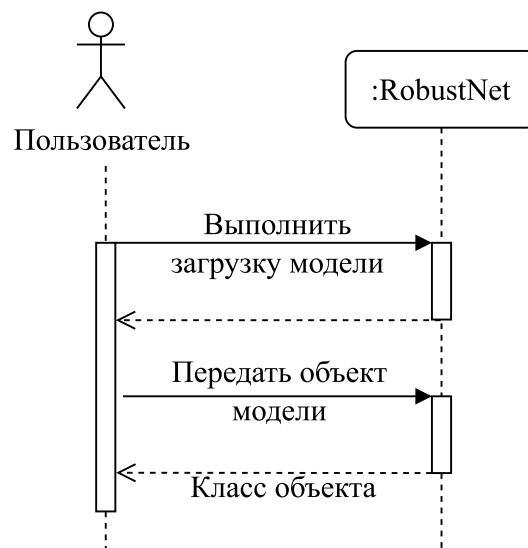


Рисунок 4.5 — Диаграмма последовательностей: работа в режиме оценки качества модели

#### Выводы по главе 4

Основным результатом, полученным в данной главе, является разработанный программный модуль для построения робастных нейронных сетей («RobustNN»). Для него приводятся минимальные системные требования, а

также описание структуры классов и различных режимов работы. Реализованный программный модуль для построения робастных нейронных сетей («RobustNN») был зарегистрирован в виде объекта интеллектуальной собственности как программа для ЭВМ (№ гос. рег. 2021618329 от 26 мая 2021 г.) [33].

## Глава 5 Практическое применение робастных нейронных сетей

В данной главе показано использование робастных нейронных сетей для решения двух практических задач. Первая задача заключалась в классификации нефтяных месторождений на основе ряда признаков. Вторая задача заключалась в определении местоположения проводника в сосуде при проведении операции кардиостентирования по измеренным значениям частоты и импеданса.

### 5.1 Классификация нефтяных месторождений

Одной из актуальных задач в рамках разработки и эксплуатации нефтяных месторождений является задача планирования бурения скважин на объектах разработки нефтяного месторождения. Одним из важных этапов решения этой задачи является прогнозирование объемов добычи нефти и жидкости для различных сценариев разработки. Однако использование единой прогнозной модели для всех месторождений может приводить к некорректным результатам. Ситуация должна улучшиться, если выполнить классификацию объектов разработки нефтяного месторождения на основании ряда признаков и для каждого класса построить собственную прогнозную модель. Рассматриваемая задача решалась совместно с сотрудниками Института «ТатНИПИнефть» ОАО «Татнефть». Результатом выполнения данного проекта стали алгоритмы, вошедшие в программный модуль Epsilon, который был зарегистрирован в качестве объекта интеллектуальной собственности как программа для ЭВМ [34].

#### 5.1.1 Постановка задачи

Рассматриваемая задача является не совсем типичным примером задачи классификации. Набор данных включает в себя 96 объектов  $X = \{X_1, \dots, X_{96}\}$ , соответствующих различным нефтяным месторождениям. Каждый объект описывается вектором признаков, состоящим из 9 компонент,  $x_m = \{x_{m1}, x_{m2}, \dots, x_{m9}\}$ , где  $x_{mi}$  – значение  $i$ -го признака объекта  $X_m$ . Описание признаков приводится в Таблице 5.1.

Данная задача отличается от обычной задачи классификации тем, что изначально классы объектов указаны не были – чтобы определить их, использовались алгоритмы кластеризации (алгоритм  $k$  средних [23] и EM алгоритм [76]). С их помощью были получены два варианта кластеризации объектов, пред-

полагающие разбиение исходного набора данных на три кластера, на основе которых и определялись классы объектов  $q_i, i = 1, 2, 3$ . Варианты разбиения приведены в Таблице 5.2.

Таблица 5.1 — Признаки объектов разработки нефтяного месторождения

Признак	Описание признака
$x_{m1}$	Средний дебит нефти, т/сут
$x_{m2}$	Обводненность, %
$x_{m3}$	Водонефтяной фактор, доли ед.
$x_{m4}$	Начальные балансовые запасы нефти, тыс. т
$x_{m5}$	Выработанность извлекаемых запасов, %
$x_{m6}$	Среднее значение проницаемости, мкм <sup>2</sup>
$x_{m7}$	Коэффициент вариации проницаемости, мкм <sup>2</sup>
$x_{m8}$	Коэффициент расчлененности по продуктивным пластам, доли ед.
$x_{m9}$	Вязкость нефти в пластовых условиях, мПа*с

Таблица 5.2 — Варианты кластеризации исходного набора данных

№ варианта	Алгоритм	Размер 1 кластера	Размер 2 кластера	Размер 3 кластера
Вариант 1	k средних	13	24	59
Вариант 2	EM	16	37	43

При обоих вариантах кластеризации объекты, входящие в разные кластеры, существенно отличались по начальным балансовым запасам нефти, среднему значению проницаемости, вязкости нефти в пластовых условиях, а также по выработанности извлекаемых запасов. При этом месторождения, входящие в первый кластер, имели наибольшие начальные балансовые запасы, проницаемость и вязкость, а выработанность извлекаемых запасов была наименьшей. Для месторождений из второго кластера, наоборот, были характерны наименьшие значения первых трех признаков и наиболее высокие значения выработанности извлекаемых запасов. Месторождения, входящие в третий кластер, имели средние значения всех перечисленных признаков.

Исходя из количества признаков объектов, а также из числа классов, для решения поставленной задачи были использованы модели нейронных сетей с

одним скрытым слоем, имеющие следующую архитектуру: входной слой включал в себя 9 нейронов (по числу признаков), выходной – 3 нейрона (по числу классов), скрытый – 4 нейрона. В качестве функции активации использовалась сигмоида (1.21).

### 5.1.2 Результаты классификации

В ходе исследований для каждого варианта кластеризации объектов была построена классическая модель нейронной сети (с квадратичной функцией потерь), а также три робастные модели: с функциями потерь Коши, Уэлша и биквадратной функцией потерь Тьюки. Поскольку классы объектов в данной задаче несбалансированные, для оценки качества работы моделей использовались значения метрик  $\alpha$ ,  $p$  и  $r$ . В Таблице 5.3 и Таблице 5.4 приводятся полученные значения этих метрик, число эпох, в течение которых обучалась нейронная сеть, а также значение параметра функции потерь (для робастных моделей), при котором были достигнуты наибольшие значения метрик, для первого и второго варианта кластеризации объектов соответственно.

Таблица 5.3 — Результаты работы моделей при первом варианте кластеризации

Функция потерь	Параметр функции потерь	Эпохи	$\alpha$	$p$	$r$
Квадратичная	-	39	0,95	0,945	0,889
Коши	1,1	63	0,95	0,945	0,889
Уэлша	1,2	48	0,95	0,945	0,889
Биквадратная Тьюки	2,5	33	0,95	0,945	0,889

Таблица 5.4 — Результаты работы моделей при втором варианте кластеризации

Функция потерь	Параметр функции потерь	Эпохи	$\alpha$	$p$	$r$
Квадратичная	-	500	0,905	0,939	0,875
Коши	2,4	283	0,952	0,967	0,958
Уэлша	1,1	176	0,952	0,967	0,958
Биквадратная Тьюки	1,5	262	0,952	0,967	0,958

Нетрудно заметить, что при первом варианте кластеризации качество работы классической ИНС и робастных моделей является сопоставимым. Однако значение метрики  $r$  для всех моделей значительно меньше, чем значения остальных метрик. Рассмотрим значения метрик  $r(q_i)$  для каждого класса в отдельности (табл. 5.5) – для всех построенных моделей они получились одинаковыми.

Таблица 5.5 — Значения метрик  $r(q_i)$  при первом варианте кластеризации

Метрика	$r(q_1)$	$r(q_2)$	$r(q_3)$
Значение метрики	0,667	1,0	1,0

Приведенные в табл. 5.5 значения метрик  $r(q_i)$  позволяют сделать вывод о том, что модели очень хорошо определяют объекты второго и третьего классов, однако потенциально могут плохо обнаруживать объекты первого класса.

Для второго варианта кластеризации полученные значения метрик позволяют сделать вывод, что робастные ИНС работают точнее, чем классическая модель. Все три метрики для робастных сетей имеют примерно одинаковые значения, что говорит о хорошем качестве работы моделей. При этом значение метрики  $r$  для классической модели значительно ниже значения этой метрики для робастных сетей. Как и в первом случае, рассмотрим значения метрик  $r(q_i)$  для каждого класса (табл. 5.6). В первой строке таблицы приведены значения для классической модели. Значения метрик  $r(q_i)$  для всех трех робастных моделей совпадают – они приведены во второй строке.

Таблица 5.6 — Значения метрик  $r(q_i)$  при втором варианте кластеризации

	$r(q_1)$	$r(q_2)$	$r(q_3)$
Классическая ИНС	0,750	0,875	1,0
Робастные ИНС	1,0	0,875	1,0

Из таблицы можно заметить, что классическая модель гораздо хуже определяет объекты первого класса, как и при первом варианте кластеризации. При этом все четыре модели потенциально могут не обнаружить какие-то объекты, относящиеся ко второму классу.

Анализируя число эпох, в течение которых обучались нейронные сети, можно отметить, что при втором варианте кластеризации быстрее всех обучилась нейронная сеть с функцией потерь Уэлша. При первом же варианте

кластеризации, хотя качество работы моделей и является сопоставимым, классическая модель обучилась быстрее робастных. Но, несмотря на это, на взгляд автора, использование робастных моделей в перспективе является более предпочтительным, поскольку новые объекты разработки нефтяного месторождения потенциально могут оказаться выбросами, что приведет к ухудшению качества работы классической ИНС.

## 5.2 Определение местоположения проводника при коронарном стентировании

На практике врачи-кардиохирурги часто сталкиваются с наличием сужений (стеноза) в коронарных артериях пациента, что в конечном итоге может привести к развитию инфаркта миокарда. Наиболее эффективным способом лечения таких пациентов является коронарное стентирование – протезирование сердечных артерий с использованием специальных металлических каркасов.

Такие операции проводятся при постоянной рентгенологической визуализации. В ходе операции используется специальный тонкий металлический проводник, позволяющий определить местоположение стеноза. При этом для хирурга, выполняющего операцию, важно знать, во-первых, до или после стеноза располагается проводник, а во-вторых, не произошел ли разрыв сосуда проводником. Это не всегда можно точно определить, опираясь на рентген сосудов, однако используемый проводник позволяет получать значение импеданса и частоты, на которой происходит его измерение, в каждой конкретной точке. Используя эти измерения, можно построить модель нейронной сети, позволяющую определять местоположение проводника в режиме реального времени.

### 5.2.1 Постановка задачи и анализируемые данные

Данные, используемые при решении этой задачи, были предоставлены федеральным государственным бюджетным учреждением «Национальный медицинский исследовательский центр имени академика Е.Н. Мешалкина» Министерства здравоохранения Российской Федерации. Все измерения были получены в ходе проведения реальных операций. Каждый объект  $X_m$  из множества объектов  $X = \{X_1, \dots, X_{|X|}\}$  описывался вектором признаков из двух компонент  $x_m = \{x_{m1}, x_{m2}\}$ , при этом признак  $x_{m1}$  соответствовал значению импеданса в точке, где помещен проводник, а признак  $x_{m2}$  – значению частоты, на



которой производилось измерение импеданса. Очевидно, что значения частоты и импеданса могут лежать в совершенно разных диапазонах, что потенциально может плохо сказаться на точности работы нейронной сети, поэтому выполнялась нормализация данных (z-нормализация или нормализация средним):

$$x_{mi} = (x_{mi} - \bar{x}_i) / \tilde{\sigma}_i, i = 1, 2,$$

где  $\bar{x}_i$  – среднее значение признака  $i$ ,  $\tilde{\sigma}_i^2$  – дисперсия значения признака  $i$ .

На основании этих признаков было необходимо, во-первых, определить положение проводника в сосуде (до или после стеноза), а во-вторых, определить, не произошел ли разрыв сосуда. Таким образом, в обоих случаях решаемая задача соответствовала задаче бинарной классификации. Для решения каждой из этих подзадач были построены модели робастных нейронных сетей двух видов. Все построенные модели представляли собой перцептрон с одним скрытым слоем. Модели первого вида имели следующую архитектуру: входной и выходной слои включали в себя по 2 нейрона, скрытый – 3 нейрона. Архитектура моделей второго вида отличалась только тем, что выходной слой включал в себя один нейрон, а для определения класса объекта выполнялось сравнение значения выходного нейрона с пороговым значением  $P$ . В качестве функции потерь во всех нейронных сетях была использована функция потерь Эндрюса, в качестве функции активации – сигмоида.

При решении задачи распознавания местоположения проводника внутри сосуда (до или после стеноза) использовался набор данных, включающий в себя 399 017 объектов. При этом положению «до стеноза» соответствовал 324 971 объект (класс объекта  $q_1$ ), а положению «после стеноза» – оставшиеся 74 064 объектов (класс объекта  $q_2$ ). При решении задачи распознавания разрыва сосудов использовался набор данных, состоящий из 5535 объектов, из которых 2460 объектов соответствовали разрыву сосуда (класс объекта  $q_1$ ), а остальные 3075 – отсутствию разрыва (класс объекта  $q_2$ ). Нетрудно заметить, что классы объектов в анализируемых наборах данных являются несбалансированными, поэтому для оценки точности классификации наряду с метрикой (1.22) также использовалась метрика (1.24).

### 5.2.2 Результаты исследований

На первом этапе исследований были построены модели робастных нейронных сетей для решения задачи определения местоположения проводника

в сосуде относительно стеноза. С учетом рекомендаций по настройке робастных нейронных сетей, данных в разделе 3.2, для построения нейронных сетей в рамках данной задачи были выбраны следующие значения параметра функции потерь Эндрюса:  $\beta = \{0.9, 1.5, 2.3, 6.0\}$ . Сначала были построены модели первого вида. Для них фиксировалось число эпох, по истечении которого была достигнута точность работы (по метрике  $\alpha$ ) не менее 0,980 (если такая точность вообще достигалась). Поскольку анализируемый набор данных включал достаточно большое число объектов, наибольшее число эпох при обучении нейронной сети было ограничено 10. Результаты исследований представлены в табл. 5.7.

Таблица 5.7 — Точность работы моделей при определении положения проводника относительно стеноза

$\beta$	Эпохи	$\alpha$	$r$	$r(q_1)$	$r(q_2)$
0,90	3	0,981	0,989	0,978	0,999
1,50	4	0,980	0,987	0,976	0,998
2,30	9	0,982	0,989	0,978	0,999
6,00	10	0,961	0,935	0,976	0,894

Из таблицы видно, что с увеличением значения параметра  $\beta$  модели начинают обучаться медленнее – значение метрики  $\alpha \geq 0,98$  достигалось за большее число эпох обучения или не достигалось вовсе. Кроме того, во время проведения исследований было отмечено, что уже после первой эпохи обучения значение метрики  $\alpha$  равнялось 0,815. Однако при этом значения метрики  $r$  для каждого класса были следующие:  $r(q_1) = 1,0$ ,  $r(q_2) = 0,0$ . Из этих значений становится очевидным то, что модель безошибочно распознавала положение проводника «до стеноза» (объекты класса  $q_1$ ), но абсолютно не распознавала положение «после стеноза» (объекты класса  $q_2$ ). Такой результат вполне объясним конфигурацией набора данных – в нем гораздо больше объектов первого класса, поэтому сеть сначала учится распознавать их (это происходит за одну эпоху, поскольку объем набора данных очень большой), а затем постепенно учится распознавать объекты второго класса. Такой вывод подтверждается и графиком, представленным на рис. 5.1, – на нем показано изменение значения метрики  $r(q_2)$  в процессе обучения модели с параметром  $\beta = 6,0$ .

Нетрудно заметить, что с ростом числа эпох обучения (а следовательно, в данном случае и точности работы модели по метрике  $\alpha$ ) значение метрики  $r(q_2)$

также растет. При этом в течение первых пяти эпох значение  $\alpha$  не изменялось, и значение  $r(q_2)$  оставалось нулевым.

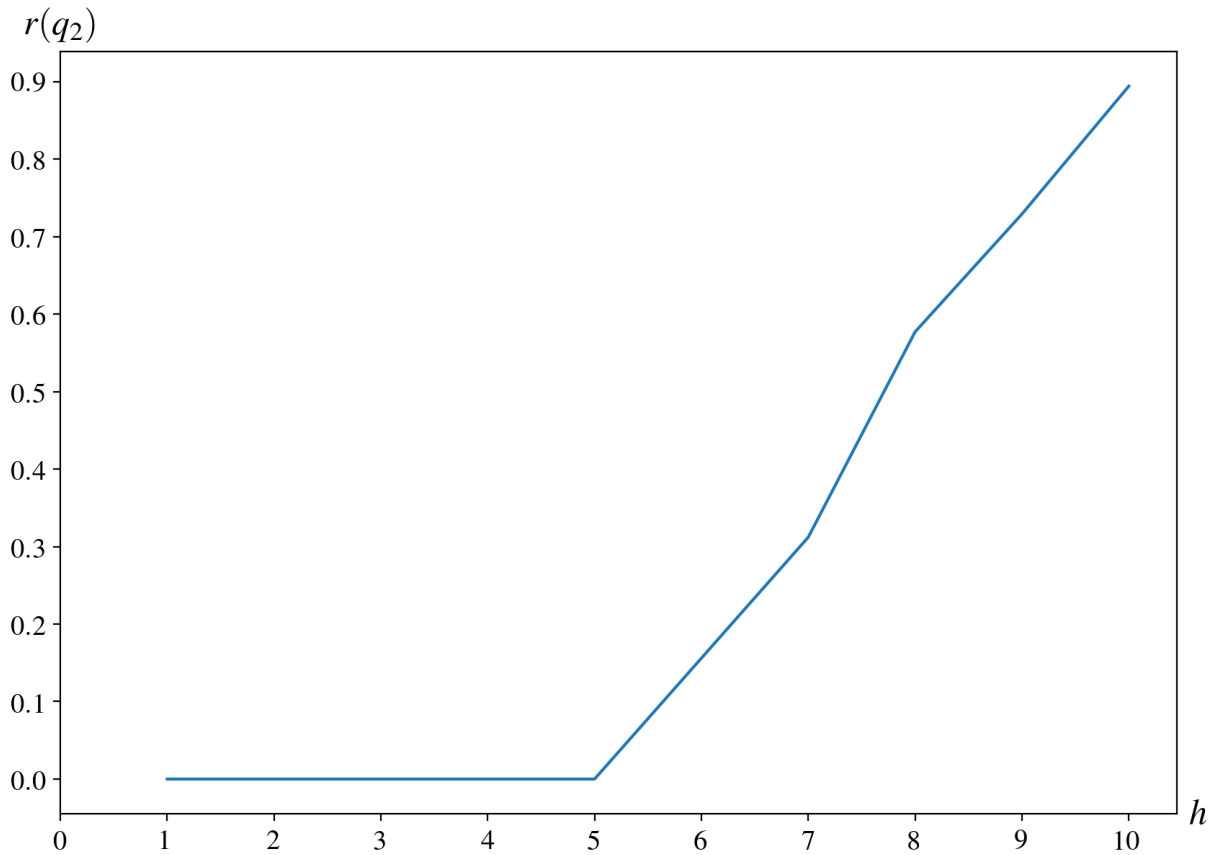


Рисунок 5.1 — Изменение значения  $r(q_2)$  с ростом числа эпох обучения ( $h$ )

После этого были построены модели нейронных сетей второго вида. В качестве значений параметра функции потерь рассматривались те же значения, что и для моделей первого вида. Для оценки качества моделей второго вида использовались ROC-кривые, алгоритм построения которых описан в п. 1.3.3. Все модели обучались в течение 5 эпох – число эпох обучения было снижено для возможности более наглядно сравнить модели (при большом числе эпох модели потенциально могут переобучиться и работать с одинаково высокой точностью). Пороговое значение  $P$  варьировалось от 0 до 1 с шагом 0,01. Значения метрик  $r(q_i)$  (1.24) и  $FPR(q_i)$  (1.25) вычислялись для  $q_1$ . Полученные кривые представлены на рис. 5.2.

Из рис. 5.2 видно следующее: наиболее высокую точность работы имеет модель с параметром  $\beta = 1,5$ , наименее точно работает модель с параметром  $\beta = 6,0$ . При этом видно, что модели с параметрами  $\beta = 0,9$  и  $\beta = 2,3$  работают незначительно хуже, чем модель с параметром  $\beta = 1,5$ . Это показывает, что небольшие отклонения при выборе значения параметра функции потерь

от рекомендованных значений не должны критически отразиться на точности работы модели, в то время как значительное отклонение потенциально может привести к резкому ухудшению точности работы нейронной сети.

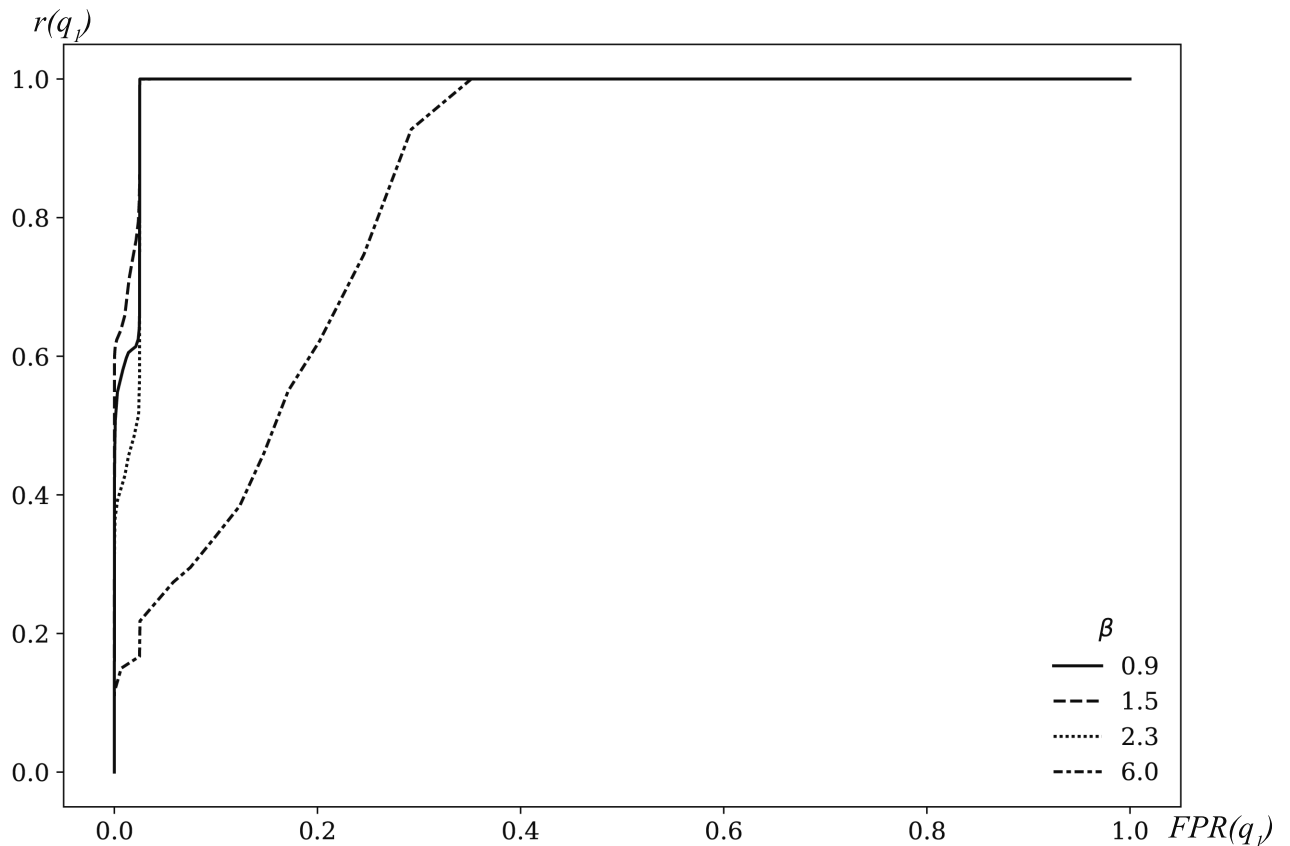


Рисунок 5.2 — Определение положения проводника в сосуде: ROC-кривые

На втором этапе исследований были построены модели, позволяющие определять, произошел разрыв сосуда проводником или нет. Значения параметра функции потерь рассматривались такие же, как и на первом этапе, для моделей первого вида значения метрик фиксировались аналогичным образом. Однако ввиду меньшего объема анализируемых данных максимальное число эпох обучения составляло 50. Полученные результаты представлены в табл. 5.8.

Таблица 5.8 — Точность работы моделей при определении разрыва сосуда

$\beta$	Эпохи	$\alpha$	$r$	$r(q_1)$	$r(q_2)$
0,90	19	0,988	0,987	0,974	1,000
1,50	25	0,989	0,988	0,976	1,000
2,30	39	0,989	0,988	0,976	1,000
6,00	50	0,797	0,776	0,552	1,000

Полученные результаты позволяют сделать вывод о том, что так же, как и при решении первой задачи, с ростом значения параметра  $\beta$  модели обучаются медленнее. Так, модель с параметром  $\beta = 6,0$  за 50 эпох обучения достигла точности только 0,797 по метрике  $\alpha$ . При этом из значения  $r(q_2)$  также видно, что модели безошибочно распознают объекты второго класса (соответствующие отсутствию разрыва) – это закономерно, поскольку таких объектов в наборе данных больше, чем объектов, соответствующих наличию разрыва. Однако и для объектов первого класса значение метрики  $r(q_1)$  достаточно высоко (кроме последней модели), что позволяет сделать вывод о хорошем качестве построенных нейронных сетей.

Аналогично первому этапу исследования для моделей второго вида были построены *ROC*-кривые. Число эпох обучения для всех моделей составляло 10, пороговое значение  $P$  варьировалось таким же образом, как на первом этапе. Построенные кривые изображены на рис. 5.3.

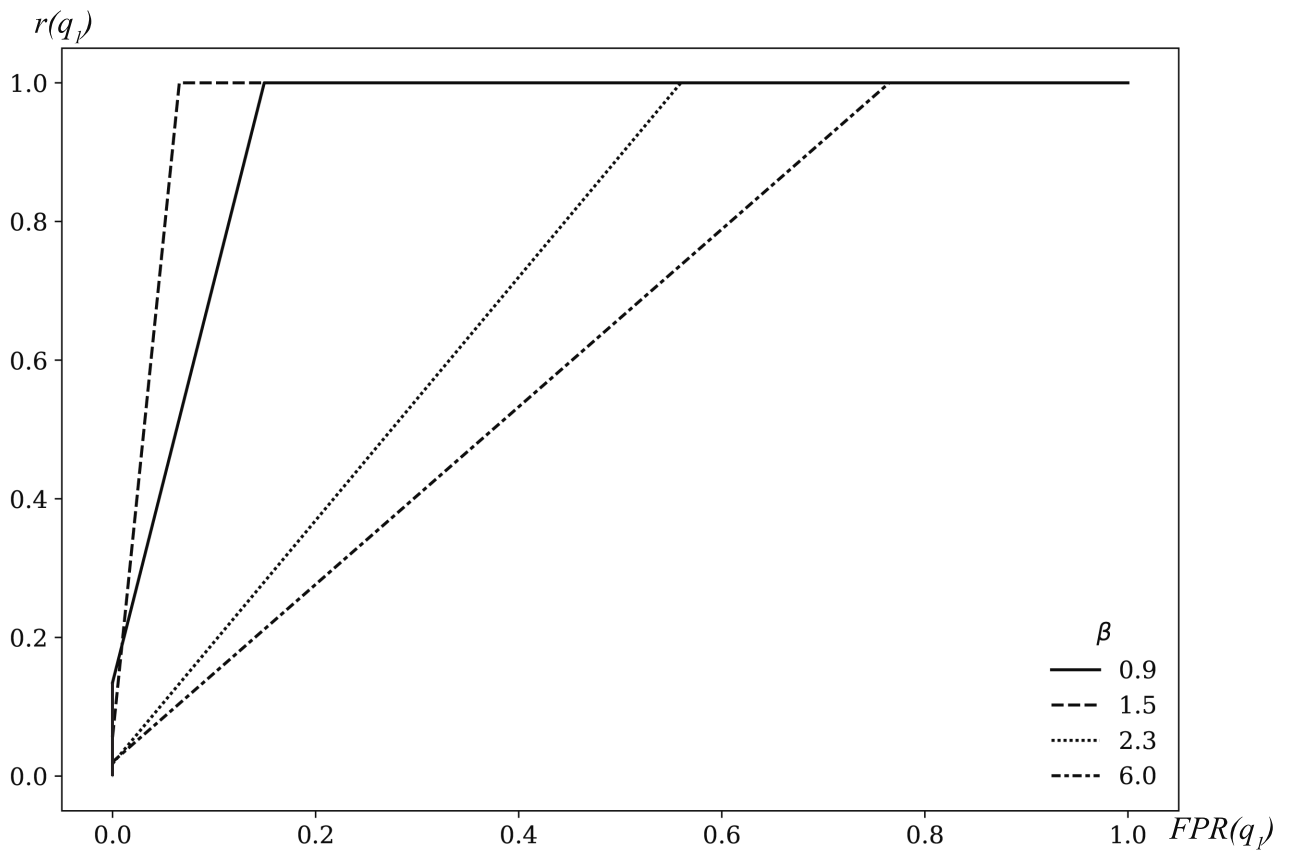


Рисунок 5.3 — Определение разрыва сосуда: ROC-кривые

Из рис. 5.3 видно, что как и на первом этапе, модель с параметром  $\beta = 1,5$  работает точнее остальных, а модель с параметром  $\beta = 6,0$  – наименее точно. При этом также можно заметить, что при отклонении значения параметра от

рекомендуемого интервала в меньшую сторону точность работы модели снижается не так резко, как при отклонении значения параметра в большую сторону. Кривые, построенные для моделей с параметрами  $\beta = 0,9$  и  $\beta = 2,3$ , показывают, что данные нейронные сети имеют достаточно низкую точность работы – очевидно, что такие модели использовать не следует. При правильно подобранном значении параметра точность работы модели достаточно высока.

Таким образом, для решения обеих подзадач был построен ряд моделей, позволяющих с высокой точностью получить решение. Все предварительно обученные модели могут быть использованы при работе в режиме реального времени.

### Выводы по главе 5

Основные результаты, полученные в данной главе, можно сформулировать следующим образом:

1. Решена задача классификации нефтяных месторождений. Для решения данной задачи был построен ряд моделей робастных нейронных сетей, а также модель классической нейронной сети. При этом рассматривались два варианта разбиения исходного набора данных на классы. В первом случае значения метрик, используемых для оценки точности работы моделей, показали, что робастные ИНС работают не хуже классической. Во втором случае точность работы робастных ИНС была выше, чем у модели с квадратичной функцией потерь.

2. Решена задача определения местоположения проводника при коронарном стентировании, которая включала в себя две подзадачи: определение расположения проводника относительно стеноза или определение наличия разрыва сосуда. Они представляли собой задачи бинарной классификации. Для решения обеих подзадач был построен ряд робастных нейронных сетей с различными параметрами. Практически все построенные модели показали высокую точность работы.

## Заключение

Основные результаты, полученные в данной работе, заключаются в следующем:

1. Проведен анализ возможности использования идей робастного подхода при построении искусственных нейронных сетей. Сформулированы и доказаны два утверждения, показывающие возможность использования 12 робастных функций потерь в нейронных сетях, 10 из которых выбраны для дальнейших исследований.

2. Предложен общий подход к построению робастных нейронных сетей на основе алгоритма обратного распространения ошибки. Для этого сформулировано и доказано утверждение, позволяющее получить робастную модификацию алгоритма обратного распространения ошибки с использованием различных функций потерь, не меняя при этом основную логику алгоритма.

3. Выполнена настройка полученных робастных нейронных сетей, по результатам вычислительных экспериментов сформированы рекомендации относительно выбора значений параметров робастных функций потерь. Проведено исследование работоспособности построенных моделей при анализе различных зашумленных данных. Впервые продемонстрировано влияние качества плана эксперимента при обучении робастных нейронных сетей.

4. Разработан программный модуль для построения робастных нейронных сетей, позволяющий задать архитектуру модели, провести обучение нейронной сети и оценить качество ее работы.

5. С использованием робастных нейронных сетей решены две практические задачи: задача классификации нефтяных месторождений и задача определения местоположения проводника при коронарном стентировании.

## Список литературы

1. Microsoft Azure: Службы облачных вычислений [Электронный ресурс]. — URL: <https://azure.microsoft.com/ru-ru/> (дата обр. 06.02.2022).
2. *Айвазян, С. А.* Линейная и нелинейная регрессии [Текст] / С. А. Айвазян, И. С. Енюков, Л. Д. Мешалкин. — М. : Финансы и статистика, 1981. — С. 304.
3. *Айвазян, С. А.* Прикладная статистика: Исследование зависимостей [Текст] / С. А. Айвазян, И. С. Енюков, Л. Д. Мешалкин. — М. : Финансы и статистика, 1985. — С. 448.
4. Анализ данных и процессов [Текст] / А. А. Барсегян [и др.]. — 3-е изд., переаб. и доп. — СПб. : БХВ-Петербург, 2009. — 512 с.
5. *Бидюк, П. И.* Построение и методы обучения Байесовских сетей [Текст] / П. И. Бидюк, А. Н. Терентьев // Таврический вестник информатики и математики. — 2004. — № 2.
6. *Вучков, И.* Прикладной линейный регрессионный анализ [Текст] / И. Вучков, Л. Бояджиева, Е. Салаков. — М. : Финансы и статистика, 1987. — 119 с.
7. *Вьюгин, В. В.* Математические основы теории машинного обучения и прогнозирования [Текст] / В. В. Вьюгин. — М., 2013.
8. *Гасс, С.* Линейное программирование [Текст] / С. Гасс. — М. : Физико-математическая литература, 1961. — 300 с.
9. Глубокие нейросети (Часть I). Подготовка данных. [Электронный ресурс]. — URL: <https://www.mql5.com/ru/articles/3486> (дата обр. 16.11.2021).
10. *Денисов, В. И.* Математическое обеспечение системы ЭВМ-экспериментатор (регрессионный и дисперсионный анализы) [Текст] / В. И. Денисов. — М. : Наука, 1977. — 252 с.
11. *Денисов, В. И.* Методы построения многофакторных моделей по неоднородным, негауссовским, зависимым наблюдениям. [Текст] / В. И. Денисов, Д. В. Лисицин. — Новосибирск : Издательство НГТУ, 2008. — 360 с.



12. *Денисов, В. И.* Экспертная система для анализа многофакторных объектов. Дисперсионный анализ. Прецедентный подход. [Текст] / В. И. Денисов, И. А. Полетаева, В. И. Хабаров. — Новосибирск, 1992. — 127 с.
13. *Денисов, В. И.* Знаковый метод: преимущества, проблемы, алгоритмы [Текст] / В. И. Денисов, В. С. Тимофеев // Научный вестник НГТУ. — Новосибирск, 2001. — Т. 10, № 1.
14. *Денисов, В. И.* Оценивание параметров регрессионных зависимостей с использованием аппроксимации Грама-Шарлье [Текст] / В. И. Денисов, В. С. Тимофеев // Автометрия. — Новосибирск, 2008. — Т. 44, № 6.
15. Документация по языку C# [Электронный ресурс]. — URL: <https://docs.microsoft.com/ru-ru/dotnet/csharp/> (дата обр. 06.02.2022).
16. Документация по языку C++ [Электронный ресурс]. — URL: <https://docs.microsoft.com/ru-ru/cpp/cpp/?view=msvc-170> (дата обр. 06.02.2022).
17. *Дрейпер, Н.* Прикладной регрессионный анализ [Текст] / Н. Дрейпер, Н. Смит. — М. : Статистика, 1973. — 392 с.
18. *Жамбю, М.* Иерархический кластер-анализ и соответствия [Текст] / М. Жамбю. — М. : Финансы и статистика, 1988. — 345 с.
19. *Зорич, В. А.* Математический анализ. Часть I [Текст] / В. А. Зорич. — Изд. 10-е, испр. — М. : МЦНМО, 2019. — 564 с.
20. *Ильин, В. А.* Основы математического анализа. Учебник. В 2-х частях. Часть 1 [Текст] / В. А. Ильин, Э. Г. Позняк. — Физматлит, 1998. — 648 с.
21. *Кендалл, М.* Статистические выводы и связи [Текст] / М. Кендалл, А. Стюарт. — М. : Наука, 1973. — 899 с.
22. *Ланкин, Ю. П.* Нейросетевой анализ сложноорганизованных экологических данных [Электронный ресурс] / Ю. П. Ланкин, Т. Ф. Басканова, Т. И. Лобова. — 2012. — URL: <https://www.science-education.ru/ru/article/view?id=6754> (дата обр. 16.11.2021).
23. *Мандель, И. Д.* Кластерный анализ [Текст] / И. Д. Мандель. — М. : Финансы и статистика, 1988. — 176 с.

24. *Манжула, В. Г.* Нейронные сети Кохонена и нечеткие нейронные сети в интеллектуальном анализе данных [Электронный ресурс] / В. Г. Манжула, Д. С. Федяшов. — 2011. — URL: <https://www.fundamental-research.ru/ru/article/view?id=21239> (дата обр. 16.11.2021).
25. *Мудров, В. И.* Метод наименьших модулей [Текст] / В. И. Мудров, В. Л. Кушко. — М. : Знание, 1971. — 61 с.
26. *Олдендерфер, М. С.* Кластерный анализ / Факторный, дискриминантный и кластерный анализ [Текст] / М. С. Олдендерфер, Р. К. Блэшфилд. — М. : Финансы и статистика, 1989. — 215 с.
27. *Орлов, А. И.* Неустойчивость параметрических методов отбраковки резко выделяющихся наблюдений [Текст] / А. И. Орлов // Заводская лаборатория. — 1992. — Т. 58, № 7.
28. *Павлов, И. П.* Лекции о работе больших полушарий головного мозга [Текст] / И. П. Павлов. — Изд. стер. — М. : Либроком, 2021. — 287 с.
29. *Полевой, Д. В.* Методы поиска ближайших соседей в задаче анализа графического образа структурированного документа [Электронный ресурс] / Д. В. Полевой, В. В. Постников // Труды ИСА РАН. — 2007. — Т. 29. — Режим доступа: <http://www.isa.ru/proceedings/images/documents/2007-29/302-319.pdf>.
30. Прикладная статистика: классификация и снижение размерности [Текст] / С. А. Айвазян [и др.]. — М. : Финансы и статистика, 1989.
31. Работа с модулями: создание, подключение инструкциями import и from [Электронный ресурс]. — URL: <https://inlnk.ru/LABV6E> (дата обр. 16.12.2021).
32. *Рао, С. Р.* Линейные статистические методы и их применение [Текст] / С. Р. Рао. — М. : Наука, 1968. — 548 с.
33. *Свидетельство о государственной регистрации программы для ЭВМ.* Модуль для построения робастных нейронных сетей (RobustNN) [Текст] / М. А. Сивак, В. С. Тимофеев (Россия). — № 2021618329 ; заявл. 26.05.2021 ; опубл. 26.05.2021. — 1 с.

34. *Свидетельство о государственной регистрации программы для ЭВМ.* Модуль прогнозирования параметров добычи нефти системы Estimating Performance of System Investment in Ling-term Oil production using Neuronet [Текст] / В. С. Тимофеев [и др.] (Россия). — № 2020667173 ; заявл. 23.11.2020 ; опубли. 21.12.2020. — 1 с.
35. *Сивак, М. А.* Исследование применимости робастных функций потерь в нейронных сетях [Текст] / М. А. Сивак // Сборник научных трудов НГТУ. — 2020. — № 4. — С. 50—58.
36. *Сивак, М. А.* Классификация зашумленных данных при различных объемах выборки [Текст] / М. А. Сивак // Наука. Технологии. Инновации : сб. науч. тр.: в 9 ч., Новосибирск, 6–10 дек. 2021 г. Ч. 2. — Изд-во НГТУ, 2021. — С. 277—279.
37. *Сивак, М. А.* Система поддержки принятия решений для анализа образовательных данных в вузе [Текст] / М. А. Сивак, В. М. Стасышин // Информатизация и связь. — 2019. — № 5. — С. 128—132.
38. *Сивак, М. А.* Настройка робастных нейронных сетей для решения задачи классификации [Текст] / М. А. Сивак, В. С. Тимофеев // Доклады Томского государственного университета систем управления и радиоэлектроники. — 2021. — Т. 24, № 3. — С. 26—32.
39. *Сивак, М. А.* Оптимизация работы робастной нейронной сети для задачи классификации [Текст] / М. А. Сивак, В. С. Тимофеев // Наука. Технологии. Инновации : сб. науч. тр.: в 9 ч., Новосибирск, 30 нояб.—4 дек. 2020 г. Ч. 2. — Изд-во НГТУ, 2020. — С. 298—300.
40. *Сивак, М. А.* Построение робастных нейронных сетей с различными функциями потерь [Текст] / М. А. Сивак, В. С. Тимофеев // Системы анализа и обработки данных. — 2021. — Т. 82, № 2. — С. 67—83.
41. *Сивак, М. А.* Применение нейронных сетей различной архитектуры для решения задачи фильтрации спама [Текст] / М. А. Сивак, В. С. Тимофеев // Наука. Технологии. Инновации : сб. науч. тр. : в 9 ч., Новосибирск, 2–6 дек. 2019 г. Ч. 2. — Изд-во НГТУ, 2019. — С. 231—232.
42. *Смирнов, В. И.* Курс высшей математики. Том I [Текст] / В. И. Смирнов. — 24-е изд. — СПб. : БХВ-Петербург, 2008. — С. 624.

43. Тимофеев, В. С. Устойчивое оценивание параметров регрессионных моделей с использованием идей метода наименьших квадратов [Текст] / В. С. Тимофеев, Е. А. Вострецова // Научный вестник НГТУ. — Новосибирск, 2007. — Т. 27, № 2.
44. Тимофеев, В. С. Робастная нейронная сеть с простой архитектурой [Текст] / В. С. Тимофеев, М. А. Сивак // Сибирский журнал индустриальной математики. — 2021. — Т. 24, № 4. — С. 126—138.
45. Тимофеев, В. С. Об оценивании статистических характеристик при анализе многофакторных объектов [Текст] / В. С. Тимофеев, В. Ю. Щеколдин // Научный вестник НГТУ. — Новосибирск, 2006. — Т. 24, № 3.
46. Федоров, В. В. Теория оптимального эксперимента [Текст] / В. В. Федоров. — М. : Наука, 1971. — 312 с.
47. Химмельблау, Д. Прикладное нелинейное программирование [Текст] / Д. Химмельблау. — Москва : Мир, 1975. — 534 с.
48. Ackley, D. H. A Learning Algorithm for Boltzmann Machines [Текст] / D. H. Ackley, G. E. Hinton, T. J. Sejnowski // Cognitive Science. — 1985. — Vol. 9, no. 1.
49. Advantages and Disadvantages of TensorFlow [Электронный ресурс]. — URL: <https://techvidvan.com/tutorials/pros-and-cons-of-tensorflow/> (visited on 03/04/2022).
50. Amazon Web Services [Электронный ресурс]. — URL: <https://aws.amazon.com/> (visited on 02/06/2022).
51. Andreou, P. Robust Artificial Neural Networks for Pricing of European Options [Текст] / P. Andreou, C. Charalambous, S. Martzoukos // Computational Economics. — 2006. — Vol. 2, no. 27.
52. Apache MXNet [Электронный ресурс]. — URL: <https://mxnet.apache.org/versions/1.8.0/> (visited on 12/16/2021).
53. Apache Spark – Unified Engine for large-scale data analytics [Электронный ресурс]. — URL: <https://spark.apache.org/> (visited on 02/06/2022).

54. *Banerjee, R.* On the unification of line processes, outlier rejection, and robust statistics with applications in early vision [Текст] / R. Banerjee, H. Division // International Conference on Signal Processing and Communication, ICSC 2013. — 2013. — P. 445—448.
55. *Barron, J. T.* A General and Adaptive Robust Loss Function [Текст] / J. T. Barron. — 2017. — URL: <https://arxiv.org/abs/1701.03077>.
56. Binary Data Services [Электронный ресурс]. — URL: <https://docs.python.org/3/library/binary.html> (visited on 02/24/2022).
57. *Bishop, C. M.* Neural Networks for Pattern Recognition [Текст] / C. M. Bishop. — New York, US : Oxford University Press, 1995. — 502 p.
58. *Bishop, C. M.* Pattern Recognition and Machine Learning [Текст] / C. M. Bishop. — Springer, 2006. — 758 p.
59. *Black, M. J.* The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields [Текст] / M. J. Black, P. Anandan // Computer Vision and Image Understanding. — 1996. — Vol. 63, no. 1. — P. 75—104.
60. *Black, M. J.* On the unification of line processes, outlier rejection, and robust statistics with applications in early vision [Текст] / M. J. Black, A. Rangarajan // International Journal of Computer Vision. — 1996. — Vol. 19. — P. 57—91.
61. *Brownlee, J.* A Gentle Introduction to the Rectified Linear Unit (ReLU) [Электронный ресурс] / J. Brownlee. — 2019. — URL: <https://inlnk.ru/agw2E2> (visited on 02/19/2022).
62. *Chartrand, R.* Iteratively reweighted algorithms for compressive sensing [Текст] / R. Chartrand, Y. Wotao // 2008 IEEE International Conference on Acoustics, Speech and Signal Processing. — 2008. — С. 3869—3872.
63. csv – CSV File Reading and Writing [Электронный ресурс]. — URL: <https://docs.python.org/3/library/csv.html> (visited on 02/24/2022).
64. *Danqing, L.* A Practical Guide to ReLU [Электронный ресурс] / L. Danqing. — 2017. — URL: <https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7> (visited on 02/19/2022).

65. Data Classification. Algorithms and Applications [Текст] / ed. by C. C. Aggarwal. — CRC Press, 2014. — 707 p.
66. *Efron, B.* An Introduction to the Bootstrap [Текст] / B. Efron, R. J. Tibshirani. — Chapman & Hall, 1994. — 456 p.
67. Fair Loss: Margin-Aware Reinforcement Learning for Deep Face Recognition [Текст] / B. Liu [et al.] // 2019 IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Korea (South). — 2019. — P. 10051—10060.
68. *Fan, J.* Local Polynomial Modelling and Its Applications [Текст] / J. Fan, I. Gijbels. — UK : Chapman & Hall, 1996. — P. 360.
69. *Fujimoto, S.* An Equivalence between Loss Functions and Non-Uniform Sampling in Experience Replay [Текст] / S. Fujimoto, D. Meger, D. Precup. — 2020. — URL: <https://inlnk.ru/poZ0mB>.
70. Getter and Setter in Python [Электронный ресурс]. — URL: <https://www.geeksforgeeks.org/getter-and-setter-in-python/> (visited on 03/15/2022).
71. *Haykin, S.* Neural Networks and Learning Machines [Текст] / S. Haykin. — 3rd ed. — Pearson Education, 2009. — 906 p.
72. How to Generate Test Datasets in Python with scikit-learn. [Электронный ресурс]. — URL: <https://inlnk.ru/5713yJ> (visited on 12/16/2021).
73. *Huber, J. P.* Robust statistics [Текст] / J. P. Huber. — 2nd ed. — New Jersey : Wiley, 2009. — 370 p.
74. Installing NumPy [Электронный ресурс]. — URL: <https://numpy.org/install/> (visited on 12/27/2021).
75. JavaScript – MDN Web Docs – Mozilla [Электронный ресурс]. — URL: <https://developer.mozilla.org/ru/docs/Web/JavaScript> (visited on 02/06/2022).
76. *Jin, X.* Expectation Maximization Clustering [Текст] / X. Jin, J. Han // Encyclopedia of Machine Learning / ed. by C. Sammut, G. I. Webb. — Boston, MA : Springer US, 2010. — P. 382—383.
77. json – JSON encoder and decoder [Электронный ресурс]. — URL: <https://docs.python.org/3/library/json.html> (visited on 02/24/2022).

78. Comparison of the Hopfield scheme to the hybrid of Lagrange and transformation approaches for solving the travelling salesman problem [Текст]. — 1995.
79. *Lutz, M.* Learning Python [Текст] / M. Lutz. — 5th ed. — CA : O'Reilly Media, Inc, 2013. — 1648 p.
80. *Mach, E.* On the effect of the spatial distribution of the light stimulus on the retina. In Ratliff F. (Ed.) [Текст] / E. Mach // Mach Bands: Quantitative Studies on Neural Networks in the Retina. — 1865.
81. Math – Mathematical functions [Электронный ресурс]. — URL: <https://docs.python.org/3/library/math.html> (visited on 02/24/2022).
82. *McCulloch, W.* A logical calculus of the ideas immanent in nervous activity [Текст] / W. McCulloch, W. Pitts // Bulletin of Mathematical Biophysics. — 1943. — Vol. 5.
83. *Mossman, D.* Three-way ROCs [Текст] / D. Mossman // Medical Decision Making. — 1999. — Vol. 19.
84. NumPy [Электронный ресурс]. — URL: <https://numpy.org/> (visited on 12/27/2021).
85. Oilfield Classification with Various Neural Networks [Текст] / M. A. Sivak [et al.] // 2021 XV International Scientific-Technical Conference on Actual Problems Of Electronic Instrument Engineering (APEIE). — 2021. — P. 600—603.
86. os – Miscellaneous operating system interfaces [Электронный ресурс]. — URL: <https://docs.python.org/3/library/os.html> (visited on 02/24/2022).
87. Private Methods in Python [Электронный ресурс]. — URL: <https://favtutor.com/blogs/python-private-methods> (visited on 02/24/2022).
88. Python File IO – Read and Write Files [Электронный ресурс]. — URL: <https://www.tutorialsteacher.com/python/python-read-write-file> (visited on 02/24/2022).
89. PyTorch [Электронный ресурс]. — URL: <https://pytorch.org/> (visited on 12/16/2021).

90. random — Generate pseudo-random numbers [Электронный ресурс]. — URL: <https://docs.python.org/3/library/random.html> (visited on 02/24/2022).
91. *Rousseeuw, P. J.* Tutorial to robust statistics [Текст] / P. J. Rousseeuw // Journal of chemometrics. — 1991. — Vol. 5, no. 1.
92. *Rousseeuw, P. J.* Robust regression and outlier detection [Текст] / P. J. Rousseeuw, A. M. Leroy. — NY. : John Wiley & Sons, 1987. — 334 p.
93. *Russell, S.* Artificial Intelligence: A Modern Approach [Текст] / S. Russell, P. Norvig. — 3rd ed. — Prentice Hall, 2010. — 1132 p.
94. *Sebastiani, F.* Text Categorization [Текст] / F. Sebastiani // Text mining and Its Applications. — Southampton, UK, 2005.
95. *Shai, S.-S.* Understanding machine learning. From theory to algorithms [Текст] / S.-S. Shai, B.-D. Shai. — Cambridge : Cambridge University Press, 2014. — 449 p.
96. *Skvortsova, E. B.* Statistical Methods in the Problem of Studying Apology Speech Formulas and Their Satellites in the English language [Текст] / E. B. Skvortsova, A. I. Bochkarev, M. A. Pepelyaeva // 2018 XIV International Scientific-Technical Conference on Actual Problems of Electronics Instrument Engineering (APEIE). Vol. 1. — 2018. — P. 33—36.
97. TensorFlow [Электронный ресурс]. — URL: <https://www.tensorflow.org/> (visited on 12/16/2021).
98. tf.keras.losses.Huber [Электронный ресурс]. — URL: [https://www.tensorflow.org/api\\_docs/python/tf/keras/losses/Huber](https://www.tensorflow.org/api_docs/python/tf/keras/losses/Huber) (visited on 12/16/2021).
99. The Microsoft Cognitive Toolkit [Электронный ресурс]. — URL: <https://docs.microsoft.com/en-us/cognitive-toolkit/> (visited on 12/16/2021).
100. The R Project for Statistical Computing [Электронный ресурс]. — URL: <https://www.r-project.org/> (visited on 02/06/2022).
101. The Scala Programming Language [Электронный ресурс]. — URL: <https://www.scala-lang.org/> (visited on 02/06/2022).



102. *Till, D. J.* A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems [Текст] / D. J. Till, R. J. Hand // *Machine Learning : journal.* — 2012. — Vol. 45.
103. Two deterministic half-quadratic regularization algorithms for computed imaging [Текст] / P. Charbonnier [и др.] // 1994 IEEE International Conference on Image Processing. Т. 2. — 1994. — С. 168—172.
104. UCI. Machine Learning Repository. Iris Data Set [Электронный ресурс]. — URL: <https://archive.ics.uci.edu/ml/datasets/iris> (visited on 12/16/2021).
105. What is the Difference Between Test and Validation Datasets? [Электронный ресурс]. — URL: <https://machinelearningmastery.com/difference-test-validation-datasets/> (visited on 03/06/2022).

## Приложение А

### Листинги программного кода

На рис. А.1 представлен пример программного кода, позволяющего организовать работу с модулем «RobustNN» в режиме обучения модели.

```
1  from robustnn import RobustNN as rnn
2
3  # step 1: creating layers
4  hidden_layer = rnn.RobustLayer(4, 4, "SIG")
5  output_layer = rnn.RobustLayer(4, 3, "SIG")
6
7  # step 2: building the model
8  my_nn = rnn.RobustNet([hidden_layer, output_layer], "WEL", 1.2)
9
10 # step 3: training the model
11 my_nn.train_net(X_learn, y_learn, 1)
12
13 # step 4: testing and evaluating the model
14 result = my_nn.test_net(X_test, y_test)
15 acc = rnn.accuracy(result)
16
17 # step 5: saving the model
18 my_nn.save("./model_well.2")
```

Рисунок А.1 — Обучение модели

На рис. А.2 приводится пример программного кода, позволяющего организовать работу с модулем «RobustNN» в режиме оценки точности модели. В данном примере для оценки точности используется метрика  $r$ .

```

1  from robustnn import RobustNN as rnn
2
3  # step 1: loading the model
4  trained_model = rnn.RobustNet()\n",
5  trained_model.load("./\model_well.2")
6
7  # step 2: launching the model using test set
8  result = trained_model.test_net(X_test, y_test)
9
10 # step 3: evaluating the model performance
11 rec1 = rnn.recall(result, 1)
12 rec2 = rnn.recall(result, 2)
13 rec3 = rnn.recall(result, 3)
14
15 rec = (rec1 + rec2 + rec3) / 3

```

Рисунок А.2 — Оценка точности модели с помощью метрики *recall*

На рис. А.3 проиллюстрирован программный код, позволяющий организовать работу с модулем «RobustNN» в режиме получения ответа для конкретного объекта.

```

1  from robustnn import RobustNN as rnn
2
3  # step 1: loading the model
4  trained_model = rnn.RobustNet()\n",
5  trained_model.load("./\model_well.2")
6
7  # step 2: launching the model using test set
8  answer = trained_model.query(X)

```

Рисунок А.3 — Получение ответа для конкретного объекта



## Приложение Б

## Свидетельства о регистрации программы для ЭВМ

РОССИЙСКАЯ ФЕДЕРАЦИЯ



## СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2021618329

**Модуль для построения робастных нейронных сетей  
(RobustNN)**

Правообладатель: **ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ  
БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ  
ВЫСШЕГО ОБРАЗОВАНИЯ «НОВОСИБИРСКИЙ  
ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ  
УНИВЕРСИТЕТ» (RU)**

Авторы: **Сивак Мария Алексеевна (RU), Тимофеев Владимир  
Семенович (RU)**

Заявка № **2021617682**Дата поступления **26 мая 2021 г.**

Дата государственной регистрации

в Реестре программ для ЭВМ **26 мая 2021 г.**

*Руководитель Федеральной службы  
по интеллектуальной собственности*

ДОКУМЕНТ ПОДПИСАН ЭЛЕКТРОННОЙ ПОДПИСЬЮ  
Сертификат Фид2A5CF8C00B7A5CF59A4A2F08092E9A118  
Владелиц, Ивлиев Григорий Петрович  
Действителен с 15.01.2021 по 15.01.2035

*Г.П. Ивлиев*



РОССИЙСКАЯ ФЕДЕРАЦИЯ



## СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2020667173

Модуль прогнозирования параметров добычи нефти  
системы Estimating Performance of System Investment in  
Long-term Oil production using Neuronet (Epsilon)

Правообладатель: *Государственное бюджетное образовательное  
учреждение высшего образования «Альметьевский  
государственный нефтяной институт» (RU)*

Авторы: *см. на обороте*



Заявка № 2020665149

Дата поступления 23 ноября 2020 г.

Дата государственной регистрации  
в Реестре программ для ЭВМ 21 декабря 2020 г.

Руководитель Федеральной службы  
по интеллектуальной собственности

*Г.П. Ивлиев* Г.П. Ивлиев

Авторы: *Тимофеев Владимир Семенович (RU), Фаддеенков Андрей  
Владимирович (RU), Тимофеева Анастасия Юрьевна (RU), Сивак  
Мария Алексеевна (RU), Насыбуллин Арслан Валерьевич (RU),  
Хаярова Динара Рафаэлевна (RU), Маннанов Ильдар Илгизович  
(RU), Орехов Евгений Валерьевич (RU)*

## Приложение В

## Акты внедрения

Татарский научно-исследовательский и проектный институт нефти (ТатНИПИнефть)  
публичного акционерного общества "Татнефть" имени В.Д.Шашина  
Республика Татарстан, г. Бугульма, ул. М.Джалиля, 32

## АКТ

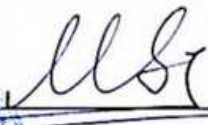
## о внедрении результатов научных исследований

Настоящим подтверждается, что Татарский научно-исследовательский и проектный институт нефти (ТатНИПИнефть) публичного акционерного общества "Татнефть" имени В.Д.Шашина использует в аналитической работе алгоритмы классификации, основанные на нейронных сетях с робастными функциями потерь, разработанные и исследованные в кандидатской диссертации Сивак Марии Алексеевны.

Применение авторских алгоритмов обучения нейронных сетей позволило более корректно решить задачу классификации нефтяных месторождений по их характеристикам. В совокупности с результатами прогнозирования показателей добычи нефти и жидкости это дает возможность комплексного решения задачи планирования и оптимизации геолого-технологических мероприятий разработки и эксплуатации нефтяных месторождений.

Директор института «ТатНИПИнефть»



 Залыатов М.М.

26.02.2022



УТВЕРЖДАЮ



Проректор по учебной работе НГТУ

Чернов С.С.

февраль 2022 г.

**АКТ О ВНЕДРЕНИИ**

результатов диссертационной работы Сивак М.А. в учебный процесс  
кафедры теоретической и прикладной информатики

Результаты диссертационной работы Сивак Марии Алексеевны, в частности, разработанный алгоритм робастного обучения нейронных сетей и соответствующее программное обеспечение использованы при реализации образовательной программы по направлению 02.04.03 – «Математическое обеспечение и администрирование информационных систем» в рамках дисциплины «Статистический анализ нечисловых данных». Освоение студентами соответствующих разделов дисциплины способствует приобретению необходимых знаний и умений для применения нейронных сетей при решении практических задач.

Заведующий кафедрой ТПИ,  
д.т.н., профессор

Чубич В.М.