МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Кафедра компьютерных технологий и систем

ШУШКЕВИЧ
Дмитрий Витальевич

АЛГОРИТМЫ И ИНСТРУМЕНТЫ НАСТРОЙКИ И ОЦЕНКИ ЭФФЕКТИВНОСТИ НЕЙРОННЫХ СЕТЕЙ, СОПОСТАВЛЕНИЕ ПРОГРАММНЫХ РЕАЛИЗАЦИЙ

Дипломная работа

Научный руководитель:
профессор кафедры КТС,
доктор физико-математических наук
профессор Таранчук В.Б.

Допущен к защите
«____» _____ 2025 г.

Зав. кафедрой компьютерных технологий и систем
доктор педагогических наук,
кандидат физико-математических наук,
профессор В. В. Казачёнок

Минск, 2025

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет прикладной математики и информатики
Кафедра компьютерных технологий и систем

УТВЕРЖДАЮ
Заведующий кафедрой

_____
(подпись, фамилия, инициалы)
«__» _____ 20__ г.

ЗАДАНИЕ
на дипломную работу

Обучающемуся Шушкевичу Дмитрию Витальевичу
Курс 4 Учебная группа 12
Специальность прикладная информатика

Тема дипломной работы:
«Алгоритмы и инструменты настройки и оценки эффективности нейронных сетей, сопостовление программных реализаций»

Утверждена приказом ректора БГУ от 12.11.2024 № 1295-ПС

Исходные данные к дипломной работе:

1. Библиотеки программных модулей д/с «Технологии интерактивной визуализации (2022)».
2. Головко, В.А. Нейросетевые технологии обработки данных / В.А. Головко, В.В. Краснопрошин – Минск : БГУ, 2017. – 270с.
3. Каталог «WOLFRAM Demonstrations Project». [Электронный ресурс]. – Режим доступа: `https://demonstrations.wolfram.com`.
4. TensorFlow Documentation. [Электронный ресурс]. – Режим доступа: `https://www.tensorflow.org/api_docs`.
5. Keras Guides. [Электронный ресурс]. – Режим доступа: `https://keras.io/guides/`.
6. Таранчук, В.Б. Средства и примеры интеллектуальной обработки данных для геологических моделей / В.Б. Таранчук // Проблемы физики, математики и техники. – 2019. – № 3 (40). – С. 117–122.

Перечень подлежащих разработке вопросов или краткое содержание расчётно-пояснительной записки:

1) Собрать материалы (к обоснованию актуальности, теоретические основы разрабатываемой темы), изучить.
2) Изучение нововведений в пакете нейронных сетей в Wolfram Mathematica.
3) Изучение библиотек машинного обучения на платформе Python.
4) Проведение сравнительного анализа эффективности нейронных сетей на платформах Wolfram и Python на примере задачи аппроксимации функции.
5) Выводы из результатов исследования.

Перечень подлежащих разработке вопросов или краткое содержание расчётно-пояснительной записки:

– **Сентябрь - Октябрь 2024:** Изучение инструментов для настройки и оценки нейронных сетей на платформах Wolfram и Python
– **Ноябрь 2024:** Подготовка данных для сравнительного анализа на основе математической задачи
– **Декабрь 2024:** Проведение сравнительного анализа
– **Февраль – Март 2025:** Подготовка данных для сравнительного анализа на основе прикладной задачи
– **Апрель – Май 2025:** Проведение сравнительного анализа для прикладной задачи

Дата выдачи задания 19.09.2024 г.

Срок сдачи законченной дипломной работы _____

Руководитель дипломной работы _____ В.Б. Таранчук
(подпись)

Подпись обучающегося _____
(подпись)

Дата _____ 20__ г.

Проинформирован о недопустимости привлечения третьих лиц к выполнению дипломной работы, плагиата, фальсификации или подлога материалов.

_____ Д.В. Шушкевич
(подпись, инициалы, фамилия обучающегося)

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ **9**

1 ТЕОРЕТИЧЕСКИЕ ОСНОВЫ **11**

1.1 Развитие и области применения нейронных сетей 11

1.2 Основные элементы и архитектура нейронных сетей 12

1.3 Активационные функции: виды и особенности 15

1.4 Современные подходы автоматизации подбора гиперпараметров 22

1.4.1 Байесовская оптимизация (Bayesian Optimization) 22

1.4.2 Алгоритм Tree-structured Parzen Estimator (TPE) 24

1.4.3 Методы досрочного прекращения (Early Stopping, Hyperband) 24

2 ПРОГРАММНЫЕ СРЕДСТВА ДЛЯ НАСТРОЙКИ И ОЦЕНКИ НЕЙРОННЫХ СЕТЕЙ НА ПЛАТФОРМЕ WOLFRAM MATHEMATICA **27**

2.1 Инструменты Wolfram Mathematica для построения и настройки нейронных сетей 27

2.2 Функционал Wolfram Mathematica для обучения нейронных сетей 30

2.3 Настройка и оптимизация геперпараметров в Wolfram Matematica 34

3 ПРОГРАММНЫЕ СРЕДСТВА ДЛЯ НАСТРОЙКИ И ОЦЕНКИ НЕЙРОННЫХ СЕТЕЙ НА ЯЗЫКЕ PYTHON **37**

3.1 Возможности библиотеки TensorFlow 38

3.2 Возможности библиотеки Keras 41

3.3 Оптимизация гиперпараметров с использованием Optuna 43

3.4 Применение технологии Ray Tune для распределённой оптимизации гиперпараметров 45

4 СРАВНИТЕЛЬНЫЙ АНАЛИЗ ПРОГРАММНЫХ РЕАЛИЗАЦИЙ **48**

4.1 Реализация нейронных сетей в среде Wolfram Mathematica 48

4.2 Реализация нейронных сетей в среде Python 53

ЗАКЛЮЧЕНИЕ 61

ПРИЛОЖЕНИЕ А 65

ПРИЛОЖЕНИЕ Б 66

РЕФЕРАТ

Дипломная работа содержит: 65 страниц, 25 иллюстраций, 17 формул, 2 таблицы, 15 использованных литературных источников.

Ключевые слова: АЛГОРИТМЫ, ИНСТРУМЕНТЫ, НЕЙРОННЫЕ СЕТИ, WOLFRAM MATHEMATICA, PYTHON, OPTUNA, TENSORFLOW, KERAS.

Объекты исследования – алгоритмы и программные средства настройки и оценки эффективности нейронных сетей.

Цель исследования – исследование, сравнение и практическая реализация алгоритмов и инструментов настройки, обучения и оценки нейронных сетей на примере задачи аппроксимации функции с шумами с использованием Wolfram Mathematica и Python (TensorFlow, Keras, Optuna).

Методы исследования – анализ литературы, программирование на языках Wolfram Language и Python, практическая реализация моделей, экспериментальная оценка эффективности.

Результаты работы: проведён сравнительный анализ возможностей Wolfram Mathematica и Python для построения и настройки нейронных сетей, реализованы и протестированы нейронные сети с автоматизированным подбором гиперпараметров, выявлены преимущества и ограничения каждого инструмента при решении прикладных задач машинного обучения.

РЭФЕРАТ

Дыпломная праца ўтрымлівае: 65 старонак, 25 малюнкаў, 17 формул, 2 табліцы, 15 выкарыстаных літаратурных крыніц.

Ключавыя словы: АЛГАРЫТМЫ, ІНСТРУМЕНТЫ, НЕЙРОННЫЯ СЕТКІ, WOLFRAM MATHEMATICA, PYTHON, OPTUNA, TENSORFLOW, KERAS.

Аб'екты даследавання – алгарытмы і праграмныя сродкі наладкі і ацэнкі эфектыўнасці нейронных сетак.

Мэта даследавання – даследаванне, параўнанне і практычная рэалізацыя алгарытмаў і інструментаў наладкі, навучання і ацэнкі нейронных сетак на прыкладзе задачы апраксімацыі функцыі з шумамі з выкарыстаннем Wolfram Mathematica і Python (TensorFlow, Keras, Optuna).

Метады даследавання – аналіз літаратуры, праграмаванне на мовах Wolfram Language і Python, практычная рэалізацыя мадэляў, эксперыментальная ацэнка эфектыўнасці.

Вынікі працы: праведзены параўнальны аналіз магчымасцей Wolfram Mathematica і Python для пабудовы і наладкі нейронных сетак, рэалізаваныя і пратэставаныя нейронныя сеткі з аўтаматызаваным падборам гіперпараметраў, вызначаны перавагі і абмежаванні кожнага інструмента для задач машыннага навучання.

ABSTRACT

The diploma work contains: 65 pages, 25 illustrations, 17 formulas, 2 tables, 15 sources.

Keywords: ALGORITHMS, TOOLS, NEURAL NETWORKS, WOLFRAM MATHEMATICA, PYTHON, OPTUNA, TENSORFLOW, KERAS.

Objects of research – algorithms and software tools for tuning and evaluating the efficiency of neural networks.

The goal – to research, compare and practically implement algorithms and tools for configuring, training, and evaluating neural networks on the example of function approximation with noise, using Wolfram Mathematica and Python (TensorFlow, Keras, Optuna).

Research methods – literature review, programming in Wolfram Language and Python, practical implementation of models, experimental evaluation of efficiency.

Results: a comparative analysis of Wolfram Mathematica and Python capabilities for building and tuning neural networks was conducted; neural networks with automated hyperparameter optimization were implemented and tested; the advantages and limitations of each tool for applied machine learning tasks were identified.

ВВЕДЕНИЕ

В последние годы искусственный интеллект стал неотъемлемой частью современного мира — от автоматизации производства и диагностики заболеваний до создания интеллектуальных транспортных систем и анализа финансовых рынков. Одним из наиболее заметных прорывов стали искусственные нейронные сети, которые уже не воспринимаются как абстрактная технология из научных публикаций, а активно используются в повседневной жизни: они лежат в основе систем распознавания изображений, голосовых помощников, рекомендательных сервисов, а также многих решений для бизнеса и науки.

Современные нейросетевые технологии позволяют не только автоматизировать рутинные задачи, но и анализировать огромные объёмы сложных, неструктурированных данных, выявлять скрытые закономерности и прогнозировать события с точностью, ранее недоступной для классических алгоритмов. Вместе с тем развитие этой области сопровождается новыми вызовами: инженеры и исследователи сталкиваются с необходимостью гибко подбирать архитектуру нейросетей, тонко настраивать гиперпараметры и объективно оценивать качество моделей, чтобы достигать лучших результатов и избегать проблем переобучения или неоптимальных решений.

Параллельно с ростом популярности нейронных сетей быстро эволюционируют программные инструменты для их разработки и тестирования. В распоряжении специалистов оказываются как универсальные вычислительные платформы вроде Wolfram Mathematica [8], так и гибкие фреймворки на языке Python: TensorFlow, Keras, PyTorch, а также специализированные библиотеки для автоматизации подбора гиперпараметров — Optuna, Ray Tune и другие [1]. Выбор подходящей экосистемы становится самостоятельной задачей, от которой во многом зависит успех внедрения интеллектуальных технологий в промышленность, медицину, транспорт или сферу цифровых сервисов.

Цель данной работы — провести детальный сравнительный анализ алгоритмов настройки и методов оценки эффективности нейронных сетей с использованием различных программных платформ (Wolfram Mathematica, Python — TensorFlow, Keras, Optuna и др.), а также реализовать экспериментальную часть на конкретной задаче аппроксимации функции с шумами,

включая этапы предобработки данных и формирования интерполирующей функции.

Практическая значимость работы заключается в формировании рекомендаций по выбору эффективных инструментов и методов настройки нейронных сетей для решения прикладных задач. Результаты исследования могут быть использованы специалистами в области машинного обучения при проектировании и оптимизации нейросетевых моделей, а также при выборе программного обеспечения для ускорения разработки и повышения качества решений.

Теоретические основы построения, обучения и диагностики нейронных сетей отражены в фундаментальных трудах и учебных пособиях [5, 6, 7], а также в лекциях и электронных ресурсах, посвящённых практической реализации моделей [9, 10]. Для изучения фундаментальных аспектов создания нейронных сетей с помощью программного кода можно обратиться к современным руководствам и онлайн-ресурсам по языку Python и его библиотекам [15, 8]В них раскрываются ключевые темы, включая конструирование нейросетевых архитектур, применение обучающих алгоритмов и анализ готовых моделей. Также рассматриваются методики синтеза сетей, отличающихся по своей структуре, освещаются этапы их диагностики и тестирования надёжности.

В качестве дополнительных источников теоретических сведений используются современные статьи и монографии [2], а примеры практического построения и визуализации нейросетей на языке Wolfram Language приведены в открытых онлайн-репозиториях и демонстрационных проектах [9, 4].

Эта работа демонстрирует различные способы и средства для настройки нейронных сетей, применяемых к задаче аппроксимации функций. Для выполнения всех операций и расчетов были задействованы программные среды Wolfram Mathematica и Python. Подробно описываются подходы к построению моделей, особенности реализации и оценки, а также рассматриваются современные методы оптимизации гиперпараметров и автоматизации процесса поиска эффективных архитектур.

ГЛАВА 1
ТЕОРЕТИЧЕСКИЕ ОСНОВЫ

1.1 Развитие и области применения нейронных сетей

В последние десятилетия развитие вычислительных технологий привело к появлению новых подходов к анализу и обработке информации, что вызвало рост интереса к системам, способным адаптивно обучаться и принимать решения на основе сложных данных. Одним из наиболее значимых достижений в этой области стали искусственные нейронные сети (ИНС), которые имитируют принципы функционирования биологических нервных систем и обеспечивают возможность автоматического выявления закономерностей, прогнозирования и оптимизации в разнообразных приложениях [7, 6, 5].

Исторически первые математические модели нейрона были предложены в середине XX века, а их дальнейшее развитие позволило сформировать основы для построения многослойных архитектур и алгоритмов обучения. С течением времени нейронные сети превратились в универсальный инструмент анализа данных, который нашёл широкое применение в различных отраслях науки, техники и бизнеса

Основные направления использования нейронных сетей включают:

- **Классификацию** — автоматическое распределение объектов по категориям на основе анализа признаков (например, распознавание рукописных символов, анализ изображений и медицинских снимков) [7, 5];
- **Распознавание образов и сигналов** — идентификация объектов, лиц, речи, жестов или иных сложных структур в потоках данных с помощью специализированных архитектур, таких как сверточные и рекуррентные нейронные сети [7, 13];
- **Прогнозирование** — моделирование и предсказание будущих значений временных рядов, трендов и событий (например, прогноз спроса, финансовых показателей, динамики технологических процессов) [6, 13];
- **Обработка естественного языка** — анализ текстов, перевод, генерация и классификация сообщений, синтез речи.

Особое значение имеет способность нейронных сетей к обучению на больших объёмах неструктурированных данных, что позволяет им превосходить традиционные алгоритмы в ряде задач, где сложно явно формализовать правила или требуется обрабатывать сложные взаимосвязи между признаками.

1.2 Основные элементы и архитектура нейронных сетей

Искусственные нейронные сети состоят из множества простых вычислительных элементов — искусственных нейронов, которые объединяются в слои, формируя сложную многоуровневую структуру для обработки информации [7, 5]. Такая организация позволяет моделировать различные зависимости между входными данными и выходами, а также решать широкий спектр задач — от классификации до регрессии.

Искусственный нейрон, являющийся базовым элементом нейросетей, имитирует функции биологического нейрона. Каждый нейрон принимает несколько входных сигналов, умножает их на соответствующие веса, суммирует результаты и пропускает итоговое значение через функцию активации, которая придаёт модели нелинейность. Представление искусственного нейрона показано на рисунке 1.1

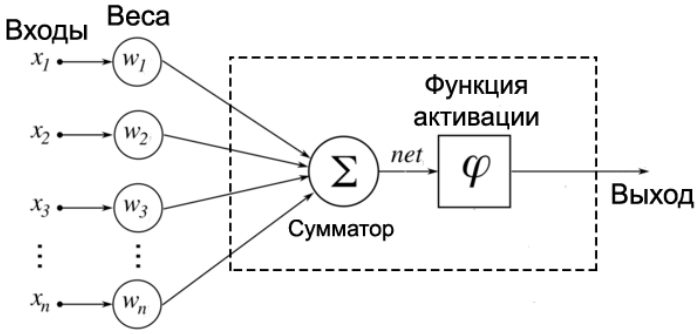

Рисунок 1.1 — Структура искусственного нейрона

Процесс передачи сигнала в нейронной сети происходит следующим образом: аксоны одного нейрона, выступая в роли его выходов, соединяются с синапсами других нейронов. Для генерации собственного выходного сигнала каждый искусственный нейрон сначала вычисляет свое текущее состояние. Для этого все входные сигналы умножаются на весовые коэффициенты, отражающие их важность, а затем полученные произведения суммируются. Результат вычисления поступает в функцию активации, которая определяет выход нейрона, формируя необходимую нелинейность в работе всей сети.

Наиболее часто используются функции активации с диапазонами значений [0;1] или [-1;1]. В зависимости от типа задачи и требований к обучению могут применяться линейная функция, сигмоидальная функция или гиперболический тангенс. Окончательное решение о применении той или иной функции активации зависит от особенностей решаемой проблемы или ограничений, накладываемых выбранным методом обучения.

Линейная функция описывается простым соотношением:

$$f(x) = x, \tag{1.1}$$

где x — входное значение.

Сигмоидная функция выражается следующей формулой:

$$f(x) = \frac{1}{1 + e^x}, \tag{1.2}$$

где x — входное значение.

Гиперболический тангенс определяется следующим образом:

$$f(x) = \frac{e^{2x} - 1}{e^{2x} + 1}, \tag{1.3}$$

где x — входное значение.

В начале работы с нейронной сетью, когда веса ещё не обучены, результат прогнозирования, как правило, далёк от истинных значений. Для корректировки параметров используется процесс обучения, который основан на последовательной обработке набора обучающих данных (так называемого тренировочного сета). После каждого полного прохода по тренировочным данным увеличивается специальный счётчик — итерация, а полный проход по всем исходам называется эпохой обучения. С увеличением числа эпох сеть становится всё более обученной и способной к качественному предсказанию. Количество эпох, как правило, задаётся вручную, а эффективность обучения оценивается по ошибке — разнице между ожидаемым и полученным результатом, которая постепенно снижается в процессе тренировок.

Нейроны в составе сети обычно группируются в несколько последовательных слоёв: входной, один или несколько скрытых, а также выходной слой[11]. Входной слой предназначен для приёма исходной информации, скрытые слои осуществляют её сложную обработку, выявляя скрытые взаимосвязи и зависимости между признаками, а выходной слой формирует итоговый результат

работы модели в требуемой для задачи форме. В нейронных сетях соединения между различными слоями обычно ориентированы от входного слоя к выходному. При этом нейроны, находящиеся в одном и том же слое, не имеют прямых связей между собой. Этот конкретный тип архитектуры носит название прямой или полносвязной нейронной сети 1.2.

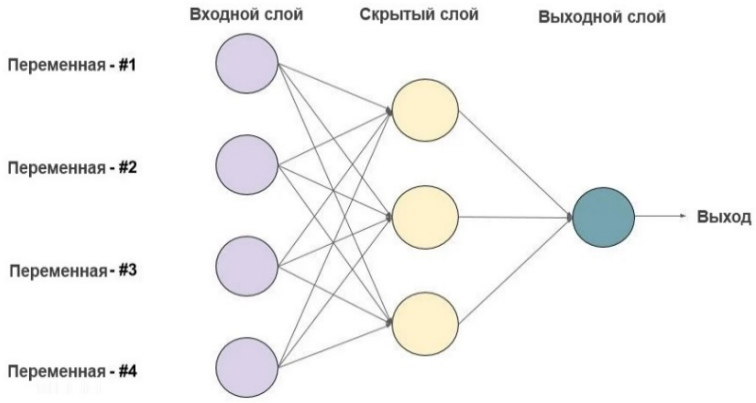

Рисунок 1.2 — Схематичное устройство нейронной сети

Обучение нейронной сети можно представить как постепенный процесс подстройки её внутренних параметров (так называемых весов), чтобы получаемые от сети результаты максимально совпадали с реальными данными. На первых шагах обучения веса задаются случайным образом, поэтому первые прогнозы сети обычно сильно отличаются от правильных ответов. Затем нейросеть многократно «проходит» по специально подготовленному набору данных (это называется «обучающей выборкой»), постепенно корректируя свои веса, чтобы снизить ошибку и приблизить результат к нужному значению. Каждый полный проход по всем обучающим данным принято называть эпохой, и с увеличением количества эпох модель становится всё точнее.

При этом важно соблюдать баланс: если обучение идёт слишком долго, нейросеть может начать идеально запоминать обучающие примеры, теряя способность правильно реагировать на новые, ранее не встречавшиеся данные. Это явление называется переобучением.

Важную роль в обучении играют и функции активации — специальные математические операции, которые определяют, как нейрон реагирует на входящий сигнал. Именно функции активации придают нейронной сети способность находить сложные, нелинейные зависимости в данных. Таким образом, не только сами веса, но и архитектура нейросети — количество слоёв, выбор функций активации и их взаимное расположение — имеют решающее

значение для качества обучения и способности модели успешно решать поставленные задачи.

1.3 Активационные функции: виды и особенности

В процессе построения и обучения нейронных сетей важную роль играют функции активации, которые позволяют управлять выходными значениями нейронов в зависимости от их входных сигналов и внутренних параметров. В многоуровневых структурах нейронных сетей каждый нейрон обладает собственными весовыми коэффициентами, входными значениями и сумматором, который вычисляет взвешенную сумму всех поступающих сигналов. Поскольку диапазон таких сумм может значительно отличаться на различных слоях и для разных комбинаций входных данных, возникает необходимость использовать функцию активации, нормализующую результат и обеспечивающую устойчивую работу сети независимо от начальных значений.

Основная задача функции активации — не только нормализация выходных значений, но и внесение в модель необходимой нелинейности. Благодаря этому нейронная сеть способна моделировать сложные взаимосвязи и выявлять зависимости, которые невозможно описать с помощью простых линейных преобразований. Помимо этого, функция активации ограничивает диапазон возможных выходных сигналов, что важно для стабильности обучения и интерпретируемости результата. В зависимости от выбора функции активации, одни нейроны могут быть "активированы"и влиять на дальнейшую обработку информации, тогда как другие остаются неактивными, минимизируя вклад в итоговое предсказание.

Выбор подходящей функции активации является одним из самых важных решений при создании нейронных сетей глубокого обучения. От того, какую именно функцию активации использует модель, во многом зависит, насколько успешно она сможет обучаться, быстро ли будет достигать нужного результата и насколько устойчивой будет к типичным проблемам, таким как переобучение. Сегодня в арсенале специалистов по нейронным сетям есть несколько популярных функций активации, каждая из которых имеет свои сильные и слабые стороны и подходит под конкретные задачи.

Рассмотрим основные из них подробнее[3]:

1. **Линейная функция активации.** Она представлена следующей формулой (1.4).

$$f(x) = x, \tag{1.4}$$

где x – входное значение.

Здесь выходное значение нейрона $f(x)$ напрямую соответствует его входному значению x. Иными словами, нейрон передает входной сигнал на свой выход без каких-либо преобразований

Несмотря на кажущуюся простоту и рациональность, у этого подхода есть принципиальное ограничение: использование линейной функции не позволяет обогатить модель нелинейными свойствами. То есть, нейросеть, построенная на линейной активации, не может идентифицировать сложные паттерны в данных, выходящие за рамки прямопропорциональных отношений. Вопреки этому, данный вид активации успешно применяется в таких задачах, как линейная регрессия, где взаимосвязь между входными сигналами и итоговыми значениями является по своей природе линейной.

Кроме того, линейная функция активации часто применяется в учебных целях, чтобы наглядно показать, как работает нейронная сеть, и подчеркнуть важность применения более сложных нелинейных функций активации для решения комплексных задач.

2. **Ступенчатая (пороговая).**

Ещё одним классическим вариантом является ступенчатая, или пороговая, функция активации, которая также известна как функция Хевисайда. Принцип работы данной функции заключается в сравнении суммарного входного сигнала с установленным порогом. В том случае, если входящий сигнал превышает это пороговое значение, функция на выходе генерирует единицу. Если же сигнал оказывается меньше порога, выходное значение равно нулю. Таким образом, ступенчатая функция эффективно моделирует бинарный переключатель по принципу «включено/выключено», создавая четкую и резкую границу между возбужденным и спокойным состоянием нейрона. Математически это выражается формулой (1.5):

$$f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}, \tag{1.5}$$

где x – входное значение.

Ступенчатая функция представлена на рисунке 1.3.

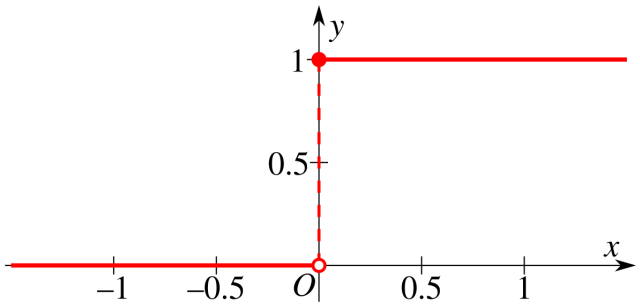

Рисунок 1.3 — Вид ступенчатой функции

Несмотря на очевидную простоту и интуитивную понятность, пороговая функция имеет серьёзные ограничения, которые препятствуют её широкому применению в современных глубоких нейронных сетях. Во-первых, она не является непрерывной и не поддаётся дифференцированию. Это создаёт существенные трудности при обучении сети методом обратного распространения ошибки, поскольку этот алгоритм требует вычисления производных функции активации для корректировки весов. Во-вторых, резкие скачки выходного значения порождают проблему так называемых «застреваний» сети, когда при малых изменениях входа результат прыгает между двумя уровнями. Это мешает плавной корректировке параметров модели, усложняет процесс оптимизации и часто не позволяет достичь желаемой сходимости.

В связи с этими особенностями ступенчатая функция практически не используется в глубоких архитектурах, хотя она по-прежнему может быть полезна в простых моделях и для объяснения базовых принципов функционирования искусственных нейронов.

3. **Сигмоидная.**

Сигмоидная функция активации занимает особое место среди всех видов функций, используемых в нейронных сетях. Её основное свойство заключается в том, что она отображает любое действительное число на диапазон от 0 до 1, что делает её удобной для интерпретации результата как вероятности возникновения того или иного события. График сигмоиды имеет гладкий, плавно возрастающий характер, а по своей форме напоминает сглаженную ступенчатую функцию, обеспечивая тем самым

непрерывность перехода между состояниями нейрона. Функция представлена следующей формулой (1.6):

$$f(x) = \frac{1}{1 + e^{-x}}, \tag{1.6}$$

где x – входное значение.

Вид сигмоидной функции показан на рисунке 1.4.

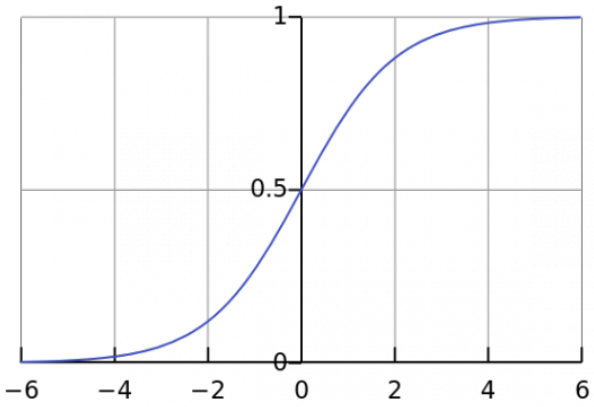

Рисунок 1.4 — График сигмоидной функции

К числу главных преимуществ сигмоидной функции относится её выраженная нелинейность, благодаря чему нейронная сеть способна моделировать сложные зависимости и решать задачи, выходящие за рамки линейных преобразований. Кроме того, сигмоида является дифференцируемой функцией, что критически важно для применения алгоритма обратного распространения ошибки, поскольку позволяет эффективно корректировать веса в процессе обучения. Ещё одним плюсом является возможность трактовать выходное значение нейрона как вероятность, что особенно удобно при решении задач классификации, в том числе в логистической регрессии. Однако применение сигмоидной функции в глубоких нейронных сетях связано с определёнными трудностями.

Одной из основных проблем является явление «вымирающего» градиента: при больших по модулю входных значениях производная сигмоиды становится крайне малой, из-за чего скорость обучения заметно падает, а корректировка весов почти не происходит. Ещё одним недостатком является асимметрия функции относительно нуля. Эта особенность может стать причиной возникновения систематической ошибки (смещения) в ходе обучения модели.

Несмотря на это, сигмоида по-прежнему востребована для решения определённых задач, прежде всего для моделирования вероятностей и при построении простых классификаторов.

4. **Гиперболический тангенс.**

Является одной из наиболее распространённых функций активации, применяемых в искусственных нейронных сетях. Эта функция представляет собой отношение гиперболического синуса к гиперболическому косинусу и позволяет преобразовывать любое действительное число в диапазон от -1 до 1. Благодаря такому свойству значения выхода оказываются ограниченными, что способствует устойчивости работы сети и облегчает интерпретацию результатов. Функция представлена следующей формулой (1.7):

$$f(x) = \frac{2}{1 + e^{-2x}} - 1, \tag{1.7}$$

где x – входное значение.

Вид гиперболического тангенса обозначен на рисунке 1.5.

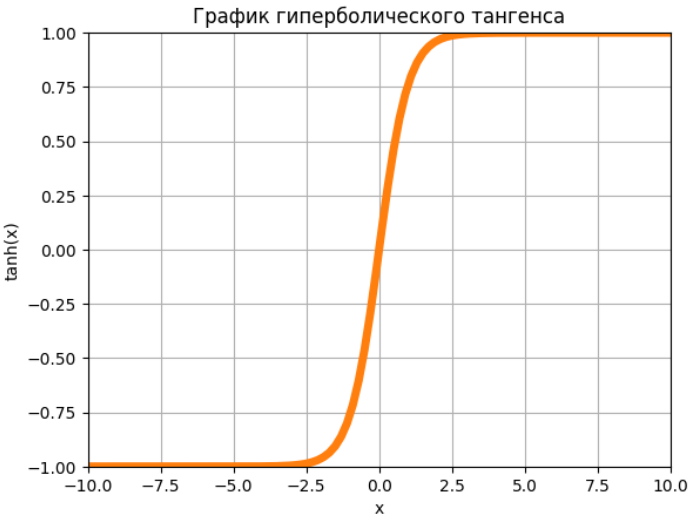

Рисунок 1.5 — График гиперболического тангенса

Важным преимуществом гиперболического тангенса является его симметрия относительно начала координат: функция принимает значение ноль при нулевом входе и изменяется равномерно в положительную и отрицательную стороны, что позволяет нейронной сети более гибко моделировать данные с как положительными, так и отрицательными отклонениями. Кроме того, гиперболический тангенс является непрерывной и

дифференцируемой функцией на всей числовой прямой, что обеспечивает возможность применения её в алгоритмах оптимизации, в частности — при обучении нейронных сетей методом обратного распространения ошибки. Нелинейный характер функции позволяет эффективно решать сложные задачи классификации и регрессии, выходящие за пределы возможностей линейных моделей.

Однако, как и сигмоидная функция, гиперболический тангенс подвержен проблеме «вымирающего» градиента: при больших по модулю входных значениях производная функции стремится к нулю, из-за чего скорость обучения существенно снижается, а корректировка весов становится затруднительной. Тем не менее, благодаря своим свойствам ограниченности, симметрии и дифференцируемости, гиперболический тангенс широко применяется не только в машинном обучении, но и в других областях науки и техники, включая физику, экономику и инженерное дело.

5. **ReLU.**

Среди современных функций активации особое место занимает ReLU (Rectified Linear Unit), которая в последние годы стала одной из наиболее популярных при построении глубоких нейронных сетей. Данная функция определяется очень просто: если входное значение положительно, на выходе оно сохраняется без изменений; если отрицательно — результат равен нулю. Таким образом, ReLU позволяет эффективно пропускать положительные сигналы и полностью блокировать отрицательные значения, что придаёт модели свойства разреженности и ускоряет обучение. Функция представлена следующей формулой (1.8):

$$f(x) = \max(0, x), \tag{1.8}$$

где x – входное значение.

Вид функции представлен на рисунке 1.6.

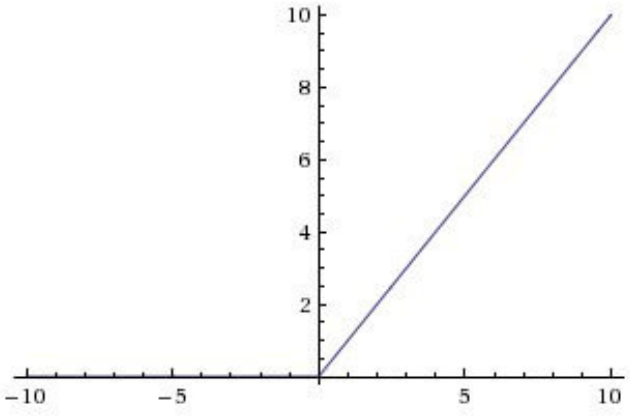

Рисунок 1.6 — График функции ReLU

Широкое применение функции ReLU объясняется её вычислительной простотой, отсутствием необходимости в сложных преобразованиях и способностью устранять проблему "вымирающего"градиента для положительных входов. Однако классическая ReLU имеет и недостатки: для отрицательных значений входа производная функции равна нулю, что может привести к тому, что некоторые нейроны перестают обучаться вовсе (так называемый эффект "умирающих нейронов").

Для того чтобы устранить недостатки функции активации ReLU, были разработаны различные её модификации. Одной из наиболее популярных является Leaky ReLU («протекающая» ReLU). В отличие от стандартной ReLU, в Leaky ReLU даже при отрицательных значениях входного сигнала выход не обнуляется полностью, а остаётся небольшим и пропорциональным входу — например, $y = 0.01x$ при $x < 0$. Это позволяет нейрону не «отключаться» полностью, поддерживая небольшой градиент и возможность дальнейшей подстройки в процессе обучения.

В целом ReLU и её модификации пользуются популярностью благодаря своей простоте и низким вычислительным затратам, что особенно важно при работе с большими и глубокими сетями. Их применение позволяет не только ускорить процесс обучения, но и существенно улучшить качество работы сложных нейронных моделей.

1.4 Современные подходы автоматизации подбора гиперпараметров

При создании нейронных сетей одной из наиболее важных задач является настройка гиперпараметров. Гиперпараметры — это параметры сети, которые нельзя оптимизировать автоматически в процессе обучения, их нужно выбирать вручную перед началом работы модели. К ним относятся, например, количество скрытых слоёв, число нейронов в слоях, скорость обучения сети (learning rate), методы регуляризации, а также тип используемых функций активации. От того, насколько удачно подобраны гиперпараметры, напрямую зависит эффективность работы нейронной сети и её способность правильно решать поставленные задачи.

Традиционные методы поиска оптимальных значений гиперпараметров, такие как *перебор по сетке (Grid Search)* и *случайный поиск (Random Search)*, обладают рядом ограничений:

- Высокая вычислительная сложность при росте размерности пространства параметров;
- Отсутствие учета информации о предыдущих запусках при планировании новых испытаний;
- Неэффективность в условиях ограниченных вычислительных ресурсов.

Эти недостатки привели к развитию более интеллектуальных методов, основанных на теории оптимизации и вероятностном моделировании [1].

1.4.1 Байесовская оптимизация (Bayesian Optimization)

Байесовская оптимизация – это эффективный подход, который строит вероятностную модель зависимости функции отклика от гиперпараметров. Наиболее часто в качестве такой модели используется *Gaussian Process (GP)* – гауссовский процесс.

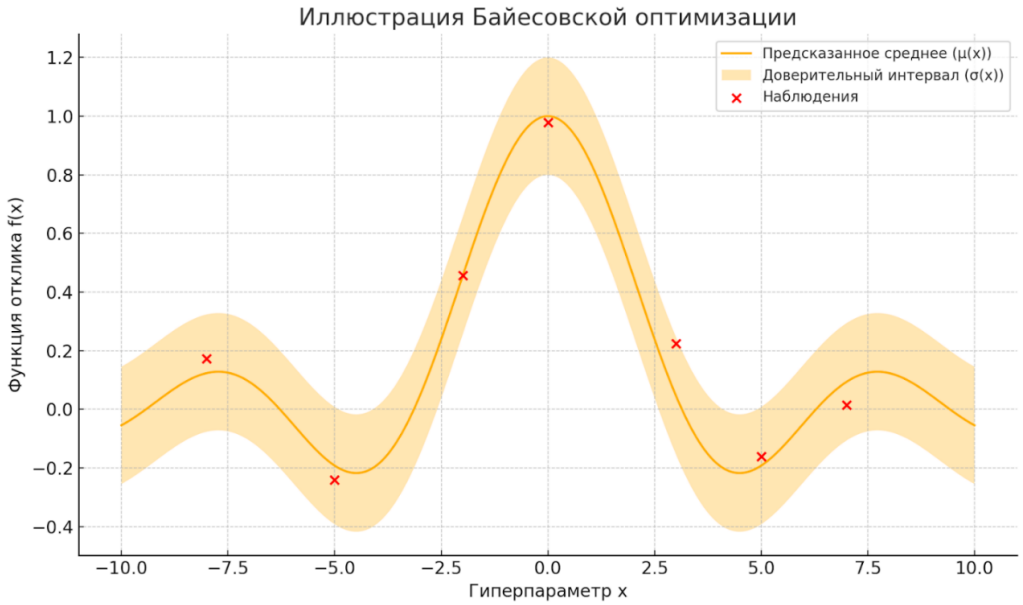

Рисунок 1.7 — Иллюстрация Байесовской оптимизации

Целью Байесовской оптимизации является максимизация так называемой *функции приобретения (acquisition function)*, которая учитывает как предсказанное среднее значение функции отклика, так и степень неопределенности в данной точке пространства параметров. Формально, пусть требуется максимизировать неизвестную функцию $f(x)$, где x – вектор гиперпараметров. На каждой итерации решается следующая задача:

$$x^* = \arg\max_x a(x|D), \tag{1.9}$$

где $a(x|D)$ – функция приобретения, основанная на текущем наборе наблюдений D.

Наиболее распространённые виды функций приобретения:

- **Expected Improvement (EI)**:

$$\mathrm{EI}(x) = E[\max(0, f(x) - f(x^+))], \tag{1.10}$$

  где $f(x^+)$ – текущее наилучшее значение.

- **Upper Confidence Bound (UCB)**:

$$\mathrm{UCB}(x) = \mu(x) + \kappa \cdot \sigma(x), \tag{1.11}$$

  где $\mu(x)$ – предсказанное среднее, $\sigma(x)$ – стандартное отклонение, κ – параметр баланса между исследованием и эксплуатацией.

1.4.2 Алгоритм Tree-structured Parzen Estimator (TPE)

TPE – это улучшение Байесовской оптимизации, предназначенное для работы с большими и сложными пространствами гиперпараметров. Вместо того чтобы моделировать непосредственно функцию отклика $p(y|x)$, как в традиционной Байесовской оптимизации, TPE строит раздельные вероятностные модели:

$$p(x|y) = \begin{cases} l(x), & \text{если } y < y^* \\ g(x), & \text{если } y \geq y^* \end{cases}, \tag{1.12}$$

где:

- $l(x)$ – аппроксимация распределения параметров, приводящих к хорошим результатам (лучше порогового значения y^*);
- $g(x)$ – аппроксимация распределения параметров, дающих менее успешные результаты;
- y^* – определенный порог, часто выбирается как квантиль по результатам предыдущих испытаний (например, 15-й процентиль).

Оптимальные параметры выбираются путём максимизации отношения:

$$x^* = \arg\max_x \frac{l(x)}{g(x)}. \tag{1.13}$$

Это позволяет фокусировать поиск в областях пространства параметров, где вероятность получения хороших результатов высока, так как алгоритм TPE выбирает такие параметры, где отношение $\frac{l(x)}{g(x)}$ максимально. На графике ниже (рисунок 1.8) представлены два распределения:

- Зеленая кривая ($l(x)$) – область, где модели показали хорошие результаты;
- Оранжевая кривая ($g(x)$) – область, где результаты оказались хуже порога.

1.4.3 Методы досрочного прекращения (Early Stopping, Hyperband)

Методы досрочного прекращения ориентированы на сокращение времени экспериментов за счет остановки неэффективных испытаний. *Hyperband* яв-

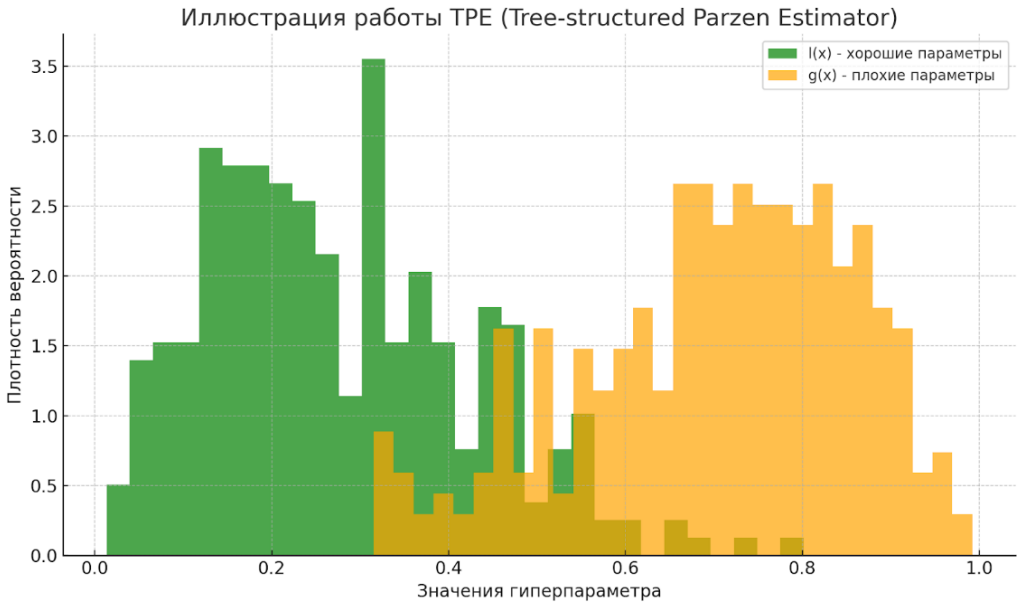

Рисунок 1.8 — Иллюстрация работы TPE (Tree-structured Parzen Estimator)

ляется одним из таких алгоритмов, комбинирующим идеи раннего останова и аллокации ресурсов.

$$s = \log_\eta(R), \tag{1.14}$$

где R – общее количество ресурсов (например, эпох обучения), η – коэффициент сокращения.

Количество испытаний и выделение ресурсов на каждом этапе определяются как:

$$n = R\eta s, \quad r = R \cdot \eta^{-s}. \tag{1.15}$$

Идея Hyperband заключается в следующем:

1. Разбивается бюджет ресурсов на несколько этапов.
2. После каждой итерации отбрасывается определённая доля худших испытаний.
3. На оставшиеся испытания выделяется больше ресурсов.

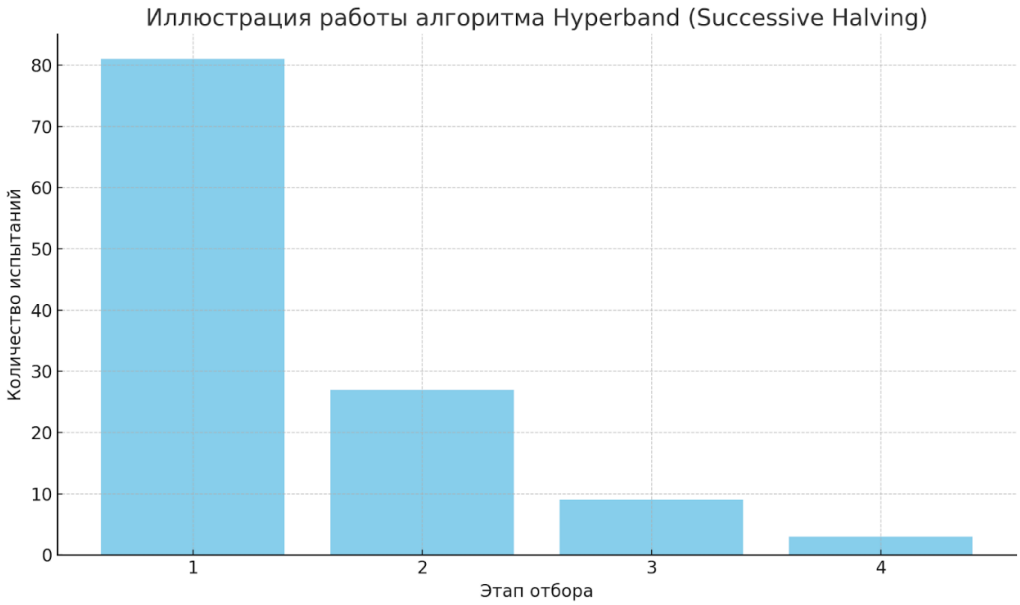

Рисунок 1.9 — Иллюстрация работы алгоритма Hyperband (Successive Halving)

Данная стратегия заметно ускоряет процесс поиска, значительно уменьшая потери мощностей на заведомо неэффективные гиперпараметры.

ГЛАВА 2

ПРОГРАММНЫЕ СРЕДСТВА ДЛЯ НАСТРОЙКИ И ОЦЕНКИ НЕЙРОННЫХ СЕТЕЙ НА ПЛАТФОРМЕ WOLFRAM MATHEMATICA

2.1 Инструменты Wolfram Mathematica для построения и настройки нейронных сетей

Нейронные сети сегодня считаются одним из самых эффективных подходов в машинном обучении. Их популярность объясняется не только способностью решать сложные задачи, но и гибкостью структуры, позволяющей адаптироваться к разным типам данных и решать широкий спектр задач — от распознавания образов и анализа текстов до прогнозирования и оптимизации. Благодаря своей архитектуре нейросети устойчивы к зашумлённым данным и способны успешно обобщать полученные знания на новые примеры, что делает их универсальным инструментом для практического использования.

Для быстрой и удобной работы с нейронными сетями разработчики программного обеспечения создают специальные среды, одной из которых является Wolfram Mathematica. Эта платформа предоставляет множество инструментов для удобного построения и настройки нейронных сетей без необходимости вручную программировать каждую деталь. В Mathematica интегрирован пакет Neural Networks, который тесно связан с языком Wolfram Language [11]. Это позволяет разработчикам использовать простой и понятный синтаксис для быстрого создания многослойных нейросетевых моделей практически любой сложности.

Более подробно о возможностях и конкретных примерах использования нейронных сетей в среде Wolfram Mathematica можно ознакомиться в специализированных источниках [14, 13].

Важной особенностью Wolfram Mathematica является наличие широкого набора базовых и специализированных слоёв, предназначенных для различных этапов обработки информации. Пользователь может создавать последовательные и разветвлённые композиции слоёв, комбинируя такие элементы, как:

- `ConvolutionLayer` — для выделения пространственных признаков в изображениях;
- `LinearLayer` — для выполнения линейных преобразований;
- `PoolingLayer` — для уменьшения размерности и повышения устойчивости к шуму;
- `ElementwiseLayer` и `BatchNormalizationLayer` — для реализации поэлементных и нормализующих операций;
- `ReLU`, `Tanh`, `Softmax` и другие функции активации — для внедрения нелинейности и корректного формирования выходного сигнала.

Конструкция нейронной сети в Wolfram Mathematica строится декларативно, путём последовательного описания слоёв. Пример базовой архитектуры можно увидеть в листинге 2.1.

```
net = NetChain[{
    LinearLayer[64],
    Tanh,
    LinearLayer[32],
    Tanh,
    LinearLayer[10],
    SoftmaxLayer[]
}];
```

Листинг 2.1 — Пример базовой архитектуры

Особое внимание в Wolfram Mathematica уделяется поддержке различных форматов входных и выходных данных. Благодаря встроенным кодировщикам и декодерам, сеть может работать не только с числовыми векторами, но и с изображениями, текстами, аудиосигналами и их комбинациями. Это существенно расширяет спектр решаемых задач — от распознавания образов до обработки естественного языка и анализа временных рядов. Подключение данных производится с помощью автоматических конвертеров типов, что делает процесс подготовки и тестирования сети максимально удобным и прозрачным.

Дополнительным преимуществом среды является наличие большого числа предобученных моделей и специализированного Wolfram Neural Net Repository, где представлены архитектуры для решения самых различных задач.

Пользователь может быстро импортировать готовую сеть, адаптировать её под собственные данные, визуализировать структуру и динамику работы, а также анализировать метрики качества непосредственно в интерактивной среде.

Для большей гибкости и полноты анализа возможностей среды Wolfram Mathematica целесообразно рассмотреть процесс обучения собственной нейронной сети с нуля, без использования предобученных моделей. Рассмотрим более подробно, как реализуется этот процесс на примере задачи классификации изображений рукописных цифр.

На первом этапе проводится подготовка исходных данных. В среде Wolfram Mathematica для этого доступен широкий спектр встроенных ресурсов, среди которых особо выделяется набор данных MNIST. Этот датасет содержит изображения рукописных цифр, каждая из которых сопровождается правильной меткой класса. Благодаря высокому уровню интеграции, загрузка и первичная обработка данных реализуется буквально в одну команду, при этом данные автоматически структурируются в виде пар «изображение–метка». Для дополнительной наглядности исследователь может визуализировать примеры из обучающей выборки, что даёт представление о структуре и качестве данных. Простой пример загрузки данных MNIST можно увидеть в листинге 2.2.

```
mnist = ResourceData["MNIST"];
trainingData = mnist["TrainingData"];
testData = mnist["TestData"];
```

Листинг 2.2 — Пример загрузки данных MNIST

Для дополнительной наглядности исследователь может визуализировать примеры из обучающей выборки, что даёт представление о структуре и качестве данных.

Следующий важный этап — определение архитектуры нейронной сети. Wolfram Mathematica предоставляет широкий набор инструментов для гибкой и наглядной сборки сети любого уровня сложности. Классическая архитектура для задач классификации изображений может включать сверточные слои для извлечения признаков, объединяющие слои для снижения размерности, нелинейные функции активации для повышения гибкости модели, а также линейные и softmax-слои для финальной классификации. Архитек-

тура сети формируется декларативно: пользователь указывает последовательность необходимых слоёв и их параметры в единой структуре, что существенно облегчает чтение, настройку и сопровождение кода. Благодаря встроенным энкодерам и декодерам, возможно автоматически преобразовать входные изображения и выходные классы в необходимый для вычислений формат. На этом этапе удобно визуализировать граф сети, чтобы убедиться в правильности компоновки и структуре всех слоёв. Окончательный вариант архитектуры для нейронной сети показан на рисунке 2.1.

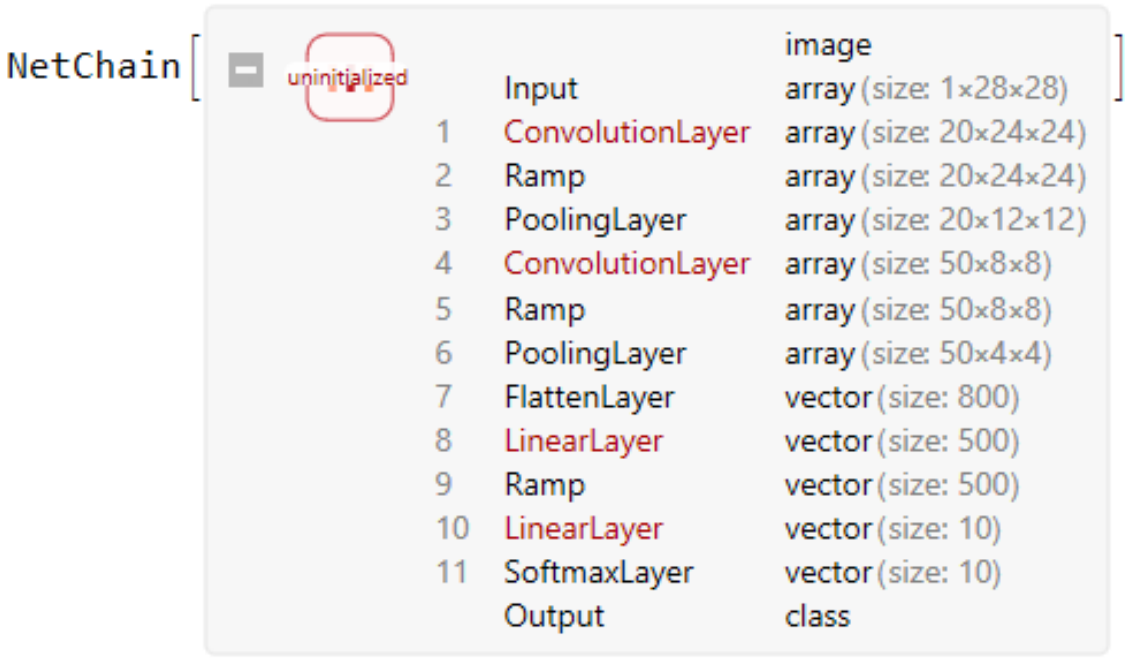

Рисунок 2.1 — Описание архитектуры нейронной сети

2.2 Функционал Wolfram Mathematica для обучения нейронных сетей

После того как архитектура нейронной сети определена и подготовлены данные, следующим шагом становится непосредственно обучение модели. В Wolfram Mathematica для этого используется специальная функция под названием `NetTrain`. С её помощью можно легко запустить процесс обучения, указав обучающий набор данных и нужные параметры оптимизации[12].

При вызове функции `NetTrain` пользователь задаёт количество эпох (число полных проходов по всем обучающим данным), размер мини-пакета данных (batch size), а также выбирает устройство, на котором будет выполняться обучение (например, центральный процессор или графическая видеокарта). Чтобы контролировать качество обучения и избежать проблемы переобуче-

ния, можно указать дополнительный набор данных (валидационный), который будет использоваться для оценки промежуточных результатов.

Очень удобной особенностью Mathematica является встроенная визуализация процесса обучения. В режиме реального времени платформа показывает, как изменяется ошибка и как растёт точность модели по мере её обучения. Это даёт возможность исследователю оперативно видеть, насколько эффективно продвигается процесс, и своевременно заметить, если модель перестаёт улучшаться. В такой ситуации можно остановить обучение, скорректировать архитектуру нейросети или изменить гиперпараметры. Процесс обучения нейронной сети показан на рисунке 2.2.

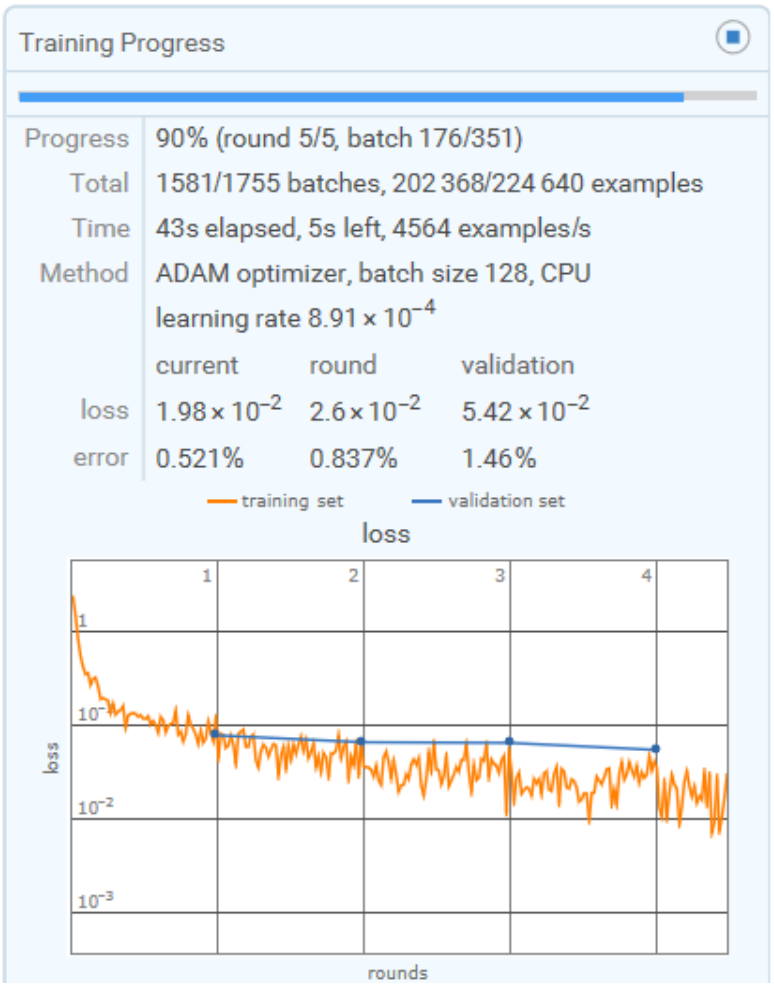

Рисунок 2.2 — Процесс обучения нейронной сети

После завершения процесса обучения сеть тестируется на отложенной выборке. Оценка качества производится с помощью встроенных инструментов, например, вычислением точности (accuracy) или построением матрицы ошибок (confusion matrix), которая позволяет детально проанализировать, какие классы данных распознаются лучше всего, а какие вызывают наибольшие за-

труднения у модели. Такой анализ помогает выявить узкие места в архитектуре или проблемные области в обучающих данных. Особенно ценным оказывается визуальный анализ ошибок классификации, когда можно наглядно увидеть примеры ошибочно распознанных изображений. Оценка точности показана на рисунках 2.3 и 2.4.

```
In[137]:= accuracy = NetMeasurements[
            trainedNet, testSet, "Accuracy"
          ]

Out[137]= 0.9888
```

Рисунок 2.3 — Оценка точности модели на тестовой выборке

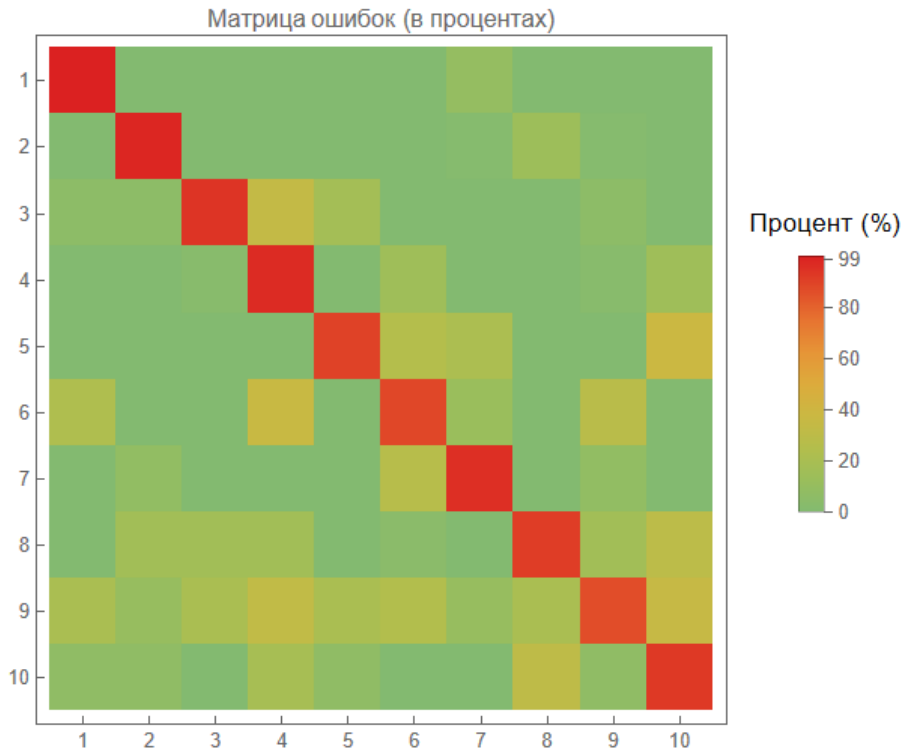

Рисунок 2.4 — Визуализация матрицы ошибок (confusion matrix)

Платформа предоставляет возможность в любой момент проверить работу сети на отдельном примере, что особенно полезно для отладки и иллюстрации результатов. Пользователь может выбрать произвольное изображение из тестовой выборки, подать его на вход сети и сравнить предсказание с реальной меткой, визуализируя этот процесс в графическом виде. Это позволяет

не только количественно, но и качественно оценить успехи обучения. Пример показан на рисунке 2.5.

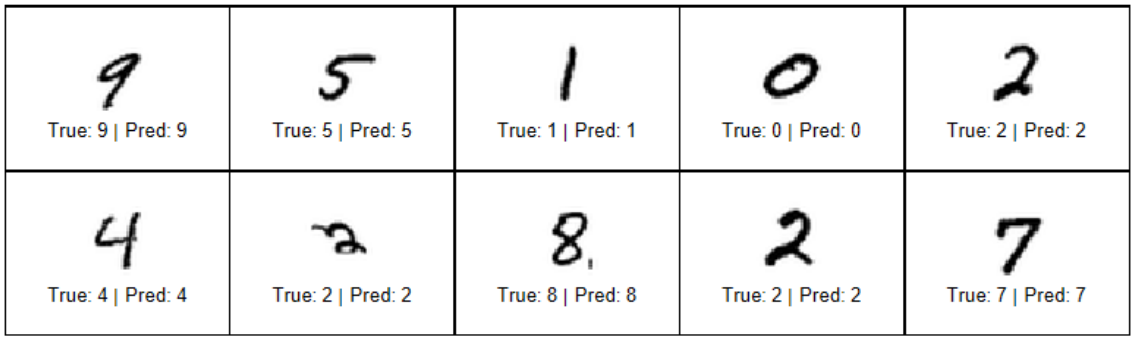

Рисунок 2.5 — Тестирование обученной сети на отдельном примере

Завершив работу, исследователь может сохранить полученную модель для дальнейшего использования, обмена или внедрения в другие проекты. Обученная сеть легко экспортируется в собственном формате Wolfram Mathematica, а при необходимости может быть быстро загружена и применена к новым данным — например, для классификации вновь поступающих изображений или интеграции в более сложные аналитические цепочки. Процесс экспорта модели показан в листинге 2.3.

```
Export["TrainedMNISTNet.wlnet", trainedNet]
```

Листинг 2.3 — Сохранение и загрузка обученной нейронной сети

Таким образом, процесс обучения собственной нейронной сети в Wolfram Mathematica отличается высокой степенью автоматизации, интеграцией инструментов для подготовки, визуализации, мониторинга и анализа моделей. Это позволяет не только быстро запускать эксперименты и достигать высоких результатов в стандартных задачах, но и настраивать каждую деталь под индивидуальные требования проекта, получая максимальный контроль над всеми стадиями построения и оптимизации нейронной сети. Благодаря простоте синтаксиса, мощным средствам визуализации и широкому набору доступных компонентов, среда Mathematica служит удобной и эффективной платформой для образовательных, научных и прикладных исследований в области искусственного интеллекта.

2.3 Настройка и оптимизация геперпараметров в Wolfram Matematica

Одним из важнейших этапов в обучении нейронных сетей является настройка гиперпараметров, от которых напрямую зависит точность модели, скорость сходимости и общая устойчивость к переобучению. Гиперпараметры, в отличие от параметров модели, не обучаются напрямую во время градиентного спуска, а определяются до начала процесса обучения. В экосистеме Wolfram Mathematica реализация настройки гиперпараметров осуществляется с использованием встроенных возможностей функции `NetTrain`, а также при помощи процедурного программирования и визуализации.

Система Wolfram Mathematica предоставляет пользователю прямой доступ к наиболее значимым гиперпараметрам в виде опций, передаваемых в функцию `NetTrain`. В частности, к числу часто настраиваемых параметров относятся: скорость обучения (`LearningRate`), тип оптимизатора (`Method`), размер минипакета (`BatchSize`), количество эпох (`MaxTrainingRounds`), а также состав и доля валидационного набора (`ValidationSet`). Например, изменение скорости обучения осуществляется путем передачи аргумента, как показано в листинге 2.4.

```
NetTrain[net, data, Method -> {"ADAM", "LearningRate" -> 0.001}]
```

Листинг 2.4 — Пример изменения скорости обучения

Подобным образом можно вручную управлять другими параметрами процесса обучения.

Однако, несмотря на широкие возможности ручной настройки, в текущей версии Mathematica отсутствует полноценный встроенный модуль автоматизированного подбора гиперпараметров, аналогичный TPE, Hyperband или Population-Based Training, реализованным в других фреймворках (например, Optuna, Ray Tune). Тем не менее, Mathematica предоставляет пользователю богатый инструментарий для самостоятельного построения процедурного поиска оптимальных значений. Для симуляции методов Grid Search и Random Search могут использоваться такие конструкции языка, как `Table`, `RandomReal`, `Map`, `NestList` и другие итеративные средства. Например, перебор значений скорости обучения можно реализовать в следующем виде (листинг 2.5).

```
learningRates = {0.001, 0.01, 0.1};
models = Table[
  NetTrain[net, data, Method -> {"ADAM", "LearningRate" -> lr},
    MaxTrainingRounds -> 10],
  {lr, learningRates}
];
```

Листинг 2.5 — Пример перебора значений скорости обучения

Для того чтобы понять, насколько успешно обучилась нейронная сеть, необходимо использовать специальные метрики качества. В Wolfram Mathematica для этих целей предусмотрена функция `NetMeasurements`. С её помощью можно легко вычислить такие важные показатели, как точность (accuracy), среднеквадратичная ошибка (MSE) или кросс-энтропия (CrossEntropyLoss). Эти метрики позволяют объективно оценить, насколько хорошо модель справляется с поставленной задачей. Также, Mathematica предоставляет ещё одну полезную функцию — `NetInformation`. С её помощью можно быстро получить подробную информацию о самой модели: сколько в ней параметров, какой её размер, а также какую точность она показывает на обучающих и валидационных данных. Благодаря этому исследователь может легко сравнивать различные варианты нейронных сетей и выбирать наиболее подходящий на основании конкретных статистических данных.

Следует отметить, что в некоторых случаях пользователи могут комбинировать Wolfram Mathematica с внешними средами (например, Python) через интерфейс `ExternalEvaluate`, `WolframScript` или `REST API`, чтобы подключать сторонние оптимизаторы и расширять функциональность системы. Также возможно использование стандартных функций оптимизации, таких как `FindMinimum`, `NMinimize` или `MachineLearningModelParameterOptimization`, в контексте простых моделей или ограниченного пространства параметров.

Несмотря на отсутствие специализированных алгоритмов байесовской оптимизации или методов досрочного останова, Wolfram Mathematica предоставляет достаточные средства для построения прототипов процедурного поиска и оценки гиперпараметров, особенно в образовательных и исследовательских задачах. Интуитивно понятный синтаксис, поддержка визуализации и возможность программного управления процессами позволяют эффективно использовать систему для настройки моделей и анализа их производительности.

Таким образом, Wolfram Mathematica, хотя и уступает в автоматизации некоторым современным фреймворкам, тем не менее, предоставляет пользователю мощный и гибкий инструментарий для построения и отладки процедур подбора гиперпараметров в рамках единичных экспериментов или серии вычислительных запусков. Это делает платформу востребованной для решения исследовательских задач, требующих детального контроля над параметрами и процесса анализа качества моделей.

ГЛАВА 3

ПРОГРАММНЫЕ СРЕДСТВА ДЛЯ НАСТРОЙКИ И ОЦЕНКИ НЕЙРОННЫХ СЕТЕЙ НА ЯЗЫКЕ PYTHON

Язык программирования Python занимает ведущие позиции среди средств разработки в современной науке и промышленности благодаря своей универсальности, лаконичному синтаксису и огромному количеству специализированных библиотек. Благодаря низкому порогу вхождения и возможности интеграции с низкоуровневыми языками, Python широко применяется не только в создании прикладных программ и автоматизации рутинных задач, но и в сфере анализа данных и разработки систем искусственного интеллекта. Особенно важное значение Python приобрёл в области построения и обучения нейронных сетей, что обусловлено наличием мощных, гибких и постоянно развивающихся инструментов для работы с моделями машинного обучения [10, 11, 15].

Среди наиболее популярных и востребованных библиотек для работы с нейронными сетями на языке Python сегодня выделяются `TensorFlow`, `Keras`, `PyTorch` и `Theano`. Данная работа концентрируется на связке `TensorFlow` и `Keras`, так как они сочетают в себе простой и интуитивный интерфейс, широкие функциональные возможности, а также активную поддержку большого сообщества разработчиков и исследователей. Эти инструменты позволяют быстро создавать нейронные сети разного уровня сложности, оперативно проводить эксперименты и прототипирование, а также эффективно применять готовые решения в реальных задачах, связанных с обработкой и анализом больших объёмов данных.

Кроме того, в рамках практической части были рассмотрены современные технологии автоматизированной настройки гиперпараметров, такие как `Optuna` и `Ray Tune`. Их применение позволяет существенно повысить качество и производительность моделей за счёт эффективного поиска оптимальных конфигураций, включая параметры обучения, архитектуру сети и алгоритмы оптимизации. Подробный анализ и реализация этих инструментов приведены в третьей главе настоящей работы.

3.1 Возможности библиотеки TensorFlow

`TensorFlow` является одной из ведущих и наиболее широко используемых библиотек для построения, обучения и внедрения моделей машинного обучения и искусственных нейронных сетей. Разработанная и поддерживаемая компанией Google, `TensorFlow` с момента своего появления в 2015 году стал стандартом де-факто для создания масштабируемых систем глубокого обучения как в научных, так и в прикладных задачах. Популярность этой платформы объясняется не только её высокой производительностью, но и гибкостью, открытым исходным кодом и широкой поддержкой сообщества исследователей и инженеров.

В основе `TensorFlow` лежит концепция построения вычислительных графов. Любая модель, реализованная с помощью данной библиотеки, представляет собой граф взаимосвязанных операций, где узлами выступают математические функции или преобразования, а рёбрами — потоки данных (тензоры). Такой подход позволяет эффективно распределять вычисления между различными устройствами, включая процессоры (`CPU`), графические процессоры (`GPU`) и специализированные ускорители (`TPU`), что особенно важно при работе с большими объемами данных и сложными нейросетевыми архитектурами.

`TensorFlow` поддерживает различные уровни абстракции: от низкоуровневого управления графом и сессиями до использования высокоуровневых API, которые позволяют быстро создавать и тестировать модели. Наиболее распространённым и удобным инструментом для быстрого прототипирования является интегрированная библиотека `Keras`, которая с версии 2.x стала частью самого `TensorFlow`. Благодаря этому стало возможным создавать сложные модели буквально в несколько строк кода, не теряя доступа к низкоуровневой настройке при необходимости.

Одна из главных особенностей `TensorFlow` — поддержка работы с многомерными массивами (тензорами), которые являются основным форматом представления данных на всех этапах построения и обучения моделей. Все операции внутри графа работают с тензорами, что позволяет эффективно манипулировать большими наборами данных, выполнять матричные преобразования и проводить оптимизацию параметров сети. `TensorFlow` предоставляет богатый набор инструментов для построения различных типов нейронных сетей: от классических полносвязных и сверточных архитектур до ре-

куррентных сетей, автоэнкодеров и генеративных моделей. Реализация пользовательских слоев, функций активации и собственных методов потерь также не вызывает затруднений благодаря гибкости и расширяемости библиотеки.

Одним из самых больших преимуществ TensorFlow является то, что он позволяет легко масштабировать обучение нейронных сетей и использовать для этого распределённые вычислительные ресурсы. Другими словами, TensorFlow отлично подходит для работы с большими задачами, когда обучение одной модели может занимать много времени. В таких ситуациях можно использовать сразу несколько видеокарт (GPU) или даже целые кластеры серверов. Это значительно ускоряет процесс тренировки модели и позволяет применять глубокое обучение в серьёзных задачах промышленного уровня, где объёмы данных огромны.

Ещё одной сильной стороной TensorFlow является хорошо проработанная система обратного распространения ошибки и богатый набор алгоритмов оптимизации. Пользователь может легко выбрать наиболее подходящий алгоритм обучения — будь то классический стохастический градиентный спуск (SGD) или более продвинутые методы, такие как Adam или RMSProp. Такая гибкость даёт возможность эффективно подбирать стратегию обучения под конкретную задачу и тип данных, обеспечивая максимально качественный результат.

Важной составляющей экосистемы является система мониторинга и визуализации — TensorBoard. Этот инструмент позволяет в реальном времени отслеживать динамику обучения, строить графики ошибок, анализировать структуру сети и выявлять возможные проблемы с переобучением или неправильной настройкой гиперпараметров. TensorBoard интегрируется с процессом тренировки через специальные коллбеки и автоматически сохраняет ключевые метрики и параметры модели.

Для наглядности рассмотрим базовый пример построения и обучения многослойного перцептрона для задачи классификации изображений с использованием TensorFlow и Keras. Пример создания и обучения простой модели в TensorFlow можно найти в Приложении А. В приведённом примере показано, как всего в несколько строк загрузить стандартный датасет, определить структуру нейросети, выбрать функцию потерь и оптимизатор, а затем провести процесс обучения и оценки качества.

TensorFlow активно используется не только для задач обработки изображений, но и для анализа текста, аудиосигналов, временных рядов и других

сложных данных. Поддержка кастомных энкодеров, декодеров, а также предварительной обработки (например, токенизации текста или преобразования аудиосигналов в спектрограммы) реализована на уровне встроенных инструментов, что позволяет строить многоуровневые гибридные модели.

Ещё одним достоинством является поддержка переноса обучения (transfer learning). Пользователь может загружать предобученные модели (например, для задач компьютерного зрения или обработки языка), дообучать их на своих данных и интегрировать в собственные приложения. Это существенно ускоряет процесс разработки и позволяет достигать высоких результатов даже при ограниченных вычислительных ресурсах или небольшом объёме размеченных данных. `TensorFlow` предоставляет интеграцию с языками C++, JavaScript (через `TensorFlow.js`) и мобильными платформами (`TensorFlow Lite`), что открывает возможности для внедрения обученных моделей в веб-приложения, мобильные устройства и системы реального времени. Кроме того, экосистема включает мощные средства для экспорта и деплоя моделей, позволяя развертывать решения как на локальных серверах, так и в облачных инфраструктурах.

Для повышения эффективности и качества моделей в `TensorFlow` реализованы инструменты для автоматического поиска оптимальных архитектур и гиперпараметров, в том числе интеграция с такими системами, как `Keras Tuner` и `Optuna`. Благодаря этому пользователь может экспериментировать с различными конфигурациями сети, автоматически подбирать размер слоёв, функции активации, оптимизаторы и другие параметры без необходимости вручную запускать десятки экспериментов.

В целом, библиотека `TensorFlow` является мощным и универсальным инструментом для решения широкого круга задач в области искусственного интеллекта и глубокого обучения. Гибкость архитектуры, богатый выбор встроенных слоёв и функций, поддержка распределённых вычислений и развитая система мониторинга делают `TensorFlow` предпочтительным выбором для научных исследований, образовательных целей и промышленной разработки. Интеграция с экосистемой Python, возможность работать с различными типами данных, поддержка современных методов оптимизации и автоматизации гиперпараметров обеспечивают максимальную производительность и удобство на всех этапах жизненного цикла модели. Более детально принципы работы `TensorFlow` рассмотрены в источнике [10].

3.2 Возможности библиотеки Keras

`Keras` — представляет собой высокоуровневый API для разработки и тренировки нейросетевых моделей. Представленная в 2015 году, эта библиотека в короткие сроки стала востребованной в сообществе специалистов, занимающихся искусственным интеллектом и машинным обучением, благодаря своей простоте и удобству. Главная идея, которую преследовали создатели `Keras`, заключалась в том, чтобы сделать процесс разработки, обучения и использования нейросетей максимально простым и понятным. Эта библиотека отличается понятным и минималистичным синтаксисом, что делает её особенно удобной для быстрого прототипирования и экспериментов. В результате `Keras` стал своеобразным стандартом для специалистов, которым нужно быстро реализовать и протестировать нейросетевые модели, не погружаясь в технические детали слишком глубоко.

`Keras` также выделяется своей гибкостью и модульностью: он позволяет использовать разные вычислительные платформы («бэкенды») — например, `TensorFlow`, `Microsoft Cognitive Toolkit` или `Theano`. Благодаря глубокой интеграции с экосистемой языка Python, `Keras` легко комбинируется с другими инструментами и библиотеками, что существенно расширяет его возможности и делает его идеальным выбором как для начинающих разработчиков, так и для профессионалов в сфере глубокого обучения.

Одно из главных преимуществ `Keras` заключается в его лаконичном и читаемом синтаксисе. Создание сложных нейросетевых архитектур и запуск экспериментов занимает всего несколько строк кода, что значительно ускоряет рабочий процесс и позволяет сосредоточиться на решаемой задаче, а не на технических деталях реализации. `Keras` поддерживает широкий набор слоёв, включая полносвязные (`Dense`), сверточные (`Conv`), рекуррентные (`LSTM`, `GRU`), объединяющие (`Pooling`), а также разнообразные функции активации, нормализации и регуляризации. Модульная структура библиотеки позволяет строить как последовательные (`Sequential API`), так и произвольно сложные, разветвлённые и объединённые модели (`Functional API`).

Последовательный API `Keras` предназначен для быстрого построения линейных цепочек слоёв, тогда как функциональный API открывает возможности для создания более сложных архитектур, таких как остаточные сети (`ResNet`), многовходовые или многовыходные модели, а также сети с произвольной топологией. `Keras` полностью совместима с экосистемой `TensorFlow`,

что обеспечивает высокую производительность, возможность масштабирования на различные вычислительные устройства и интеграцию с дополнительными инструментами, такими как `TensorBoard` для мониторинга, `Keras Tuner` для оптимизации гиперпараметров, а также поддержка экспорта моделей для деплоя на мобильных устройствах и в облачных сервисах.

Стандартный процесс работы с `Keras` включает несколько этапов: подготовка данных, построение архитектуры модели, компиляция с выбором оптимизатора и функции потерь, обучение (`fit`), а затем тестирование и визуализация результатов. Базовый пример построения и обучения нейросети для задачи классификации изображений расположен в Приложении А.

Для создания более сложных моделей используется функциональный API, который позволяет работать с несколькими входами и выходами, а также реализовывать нетривиальные соединения слоёв. Пример использования `Functional API` показан в листинге 3.1.

```
inputs = keras.Input(shape=(784,))
x = keras.layers.Dense(128, activation='relu')(inputs)
x = keras.layers.Dropout(0.2)(x)
outputs = keras.layers.Dense(10, activation='softmax')(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

Листинг 3.1 — Пример использования Functional API

В `Keras` реализованы все современные методы борьбы с переобучением — такие как `Dropout`, L1/L2-регуляризация, `Batch Normalization`, что позволяет повысить устойчивость моделей и добиться лучших результатов на независимых данных. Кроме того, поддерживаются различные стратегии ранней остановки (`EarlyStopping`), автоматического сохранения лучших весов (`ModelCheckpoint`) и мониторинга динамики обучения. `Keras` тесно интегрируется с `TensorBoard`, что обеспечивает удобный мониторинг процесса обучения в режиме реального времени. Пользователь может отслеживать изменение метрик качества, функцию потерь, а также визуализировать архитектуру модели, веса слоёв и распределение активаций. Это существенно облегчает отладку моделей, анализ их поведения и поиск оптимальных решений.

Одна из ключевых возможностей `Keras` — простота использования предобученных моделей для решения прикладных задач с помощью технологии *transfer learning*. Благодаря наличию большого числа моделей, доступных в модуле `keras.applications` (например, `VGG`, `ResNet`, `Inception` и др.), поль-

зователь может легко загрузить уже обученную на крупном датасете модель и дообучить её на своих данных, затратив минимум времени на подготовку. Пример загрузки предобученной модели и её подготовки для дообучения показан в листинге 3.2.

```
from tensorflow.keras.applications import ResNet50

# ResNet50
base_model = ResNet50(
    weights='imagenet',
    include_top=False,
    input_shape=(224,224,3)
)

base_model.trainable = False
```

Листинг 3.2 — Пример использования предобученной модели

`Keras` отличается отличной совместимостью с другими фреймворками, поддерживает экспорт моделей в различные форматы (`TensorFlow SavedModel`, `ONNX` и др.), а также может быть интегрирована с инструментами для автоматизации поиска гиперпараметров (например, `Keras Tuner`, `Optuna`). Это делает библиотеку оптимальным выбором для экспериментов, промышленного внедрения и исследовательской работы.

В целом, `Keras` предоставляет пользователю мощный и одновременно интуитивно понятный инструмент для построения и обучения нейронных сетей любой сложности. Благодаря модульности, поддержке современных методов глубокого обучения, тесной интеграции с `TensorFlow` и большим возможностям для мониторинга, оптимизации и деплоя, библиотека `Keras` остаётся одним из наиболее востребованных решений в современной практике машинного обучения. Использование `Keras` существенно ускоряет процесс прототипирования, снижает вероятность ошибок, а также облегчает внедрение моделей в реальные бизнес-приложения и научные исследования.

3.3 Оптимизация гиперпараметров с использованием Optuna

Современные задачи машинного обучения, включая обучение нейронных сетей, часто требуют настройки большого количества гиперпараметров, ока-

зывающих существенное влияние на итоговую точность и устойчивость моделей. Ручная настройка таких параметров является неэффективной и трудозатратной. Для решения этой проблемы были разработаны автоматизированные инструменты оптимизации, среди которых одним из наиболее популярных является библиотека `Optuna`.

`Optuna` основана на методах байесовской оптимизации и реализует улучшенный алгоритм поиска – `Tree-structured Parzen Estimator (TPE)`, который позволяет эффективно исследовать пространство гиперпараметров. В отличие от традиционной байесовской оптимизации, которая моделирует распределение $p(y|x)$, `TPE` строит раздельные модели для хороших и плохих результатов, это отражает функция, заданная в уравнении (1.13).

Целевая функция выбора новых гиперпараметров строится на основе максимизации отношения, выраженной в формуле (1.14). Это позволяет фокусироваться на областях пространства, где вероятность достижения улучшенных результатов наиболее высока.

Ключевые механизмы Optuna:

1. **Define-by-Run API.** В отличие от традиционных инструментов (например, `Hyperopt`), в которых пространство поиска параметров задается статически, `Optuna` позволяет динамически определять его в процессе исполнения кода. Это особенно важно для сложных моделей, где одни гиперпараметры зависят от значений других. Пример определения пространства параметров представлен в листинге 3.3.

```
import optuna

def objective(trial):
    lr = trial.suggest_loguniform(
        'learning_rate', 1e-5, 1e-1
    )
    n_layers = trial.suggest_int('n_layers', 1, 5)
    return train_model(lr, n_layers)
```

Листинг 3.3 — Динамическое определение пространства поиска в Optuna

2. **Механизм раннего останова (Pruners).** `Optuna` реализует продвинутую стратегию остановки неэффективных испытаний на ранних этапах обучения. На основе анализа промежуточных метрик, такие испытания

досрочно завершаются, что позволяет существенно сэкономить вычислительные ресурсы. Типичный алгоритм останова – `Median Pruner`, который завершает эксперименты, результаты которых хуже медианных значений по предыдущим испытаниям на аналогичных шагах.

3. **Визуализация и анализ экспериментов.** `Optuna` предоставляет встроенные средства визуализации:

   - График истории оптимизации (`optuna.visualization.plot_optimization_history`)
   - Важность параметров (`plot_param_importances`)
   - Взаимосвязь параметров (`plot_parallel_coordinate`)

   Эти инструменты позволяют проанализировать, какие гиперпараметры оказывают наибольшее влияние на результат. Формальные метрики и критерии включают в себя *Expected Improvement (EI)* и *Upper Confidence Bound (UCB)*, выраженные в формулах (1.11) и (1.12).

3.4 Применение технологии Ray Tune для распределённой оптимизации гиперпараметров

Современные модели машинного обучения, в особенности глубокие нейронные сети, включают множество гиперпараметров, влияющих на итоговую производительность. Поиск оптимальных значений этих параметров становится критически важным при разработке эффективных моделей. Однако при наличии больших наборов данных и высоких требований к вычислениям применение классических методов, таких как `Grid Search` или `Random Search`, оказывается нецелесообразным из-за высокой ресурсоёмкости. Одним из наиболее продвинутых инструментов, решающих эту задачу, является `Ray Tune` – модуль библиотеки `Ray`, обеспечивающий масштабируемую, распределённую и высокоэффективную оптимизацию гиперпараметров.

`Ray Tune` предоставляет удобный интерфейс для интеграции с популярными фреймворками глубокого обучения (`PyTorch`, `TensorFlow`, `Keras`, `XGBoost`) и позволяет организовывать масштабируемые вычисления как в локальной среде, так и в облачных кластерах. Он поддерживает автоматическое логирование, работу с чекпоинтами, распределение вычислительных ресурсов, а также множество алгоритмов оптимизации.

Ключевые особенности `Ray Tune`:

- параллельный и распределённый запуск экспериментов;
- поддержка современных методов оптимизации: `Hyperband`, `ASHA`, `Bayesian Optimization`, `Population-Based Training (PBT)`;
- возможность динамического изменения гиперпараметров во время обучения;
- совместимость с инструментами мониторинга и логирования (`TensorBoard`, `MLflow`, `Weights & Biases`).

`Ray Tune` поддерживает несколько алгоритмов подбора гиперпараметров, каждый из которых может использоваться в зависимости от специфики задачи:

Hyperband – метод, основанный на принципе раннего останова. Объём вычислительных ресурсов – R (например, количество эпох обучения) делится между множеством конфигураций. На каждом этапе наихудшие по качеству конфигурации отбрасываются, а оставшиеся получают больше ресурсов. Этот процесс повторяется до тех пор, пока не останется одна конфигурация. Формально, число шагов определяется по формуле (1.15).

ASHA (Asynchronous Successive Halving Algorithm) – улучшенная версия `Hyperband`, в которой отсутствуют синхронные барьеры. Это позволяет моделям завершать обучение в разное время, что повышает общую эффективность при распределённых вычислениях.

Bayesian Optimization – метод, основанный на моделировании зависимости между параметрами и значениями функции потерь. В `Ray Tune` реализован через интеграцию с библиотекой `HyperOpt`, использующей алгоритм `TPE (Tree-structured Parzen Estimator)`, аналогично `Optuna`. Новые точки в пространстве параметров выбираются на основе максимизации отношения плотностей.

Population-Based Training (PBT) – эволюционный алгоритм, в котором несколько моделей обучаются параллельно, а затем лучшие решения частично копируются в другие модели с возможной «мутацией» параметров. Таким образом осуществляется не только поиск параметров, но и их динамическая адаптация в процессе обучения.

Процесс использования `Ray Tune` включает следующие этапы:

1. Определение пространства поиска гиперпараметров, например:

```
config = {
    "lr": tune.loguniform(1e-5, 1e-1),
    "batch_size": tune.choice([32, 64, 128]),
    "hidden": tune.randint(64, 256),
}
```

Листинг 3.4 — Определение пространства поиска в Ray Tune

2. Определение целевой функции, в которой осуществляется обучение модели и логирование метрик:

```
def train_model(config):
    for epoch in range(config["epochs"]):
        loss = train(...)
        tune.report(loss=loss)
```

Листинг 3.5 — Определение целевой функции в Ray Tune

3. Запуск эксперимента с заданным алгоритмом:

```
tune.run(
    train_model, config=config, scheduler=ASHAScheduler()
)
```

Листинг 3.6 — Запуск эксперимента в Ray Tune

Во время выполнения экспериментов `Ray Tune` автоматически сохраняет промежуточные результаты, строит визуализации и предоставляет доступ к `Ray Dashboard`, где можно отслеживать обучение в реальном времени.

ГЛАВА 4

СРАВНИТЕЛЬНЫЙ АНАЛИЗ ПРОГРАММНЫХ РЕАЛИЗАЦИЙ

В современной практике построения и оптимизации нейронных сетей критически важным аспектом становится не только выбор архитектуры, но и подбор эффективных инструментальных средств. В данной главе проводится сопоставление двух подходов — `Wolfram Mathematica` и `Python (TensorFlow / Keras)` — применительно к задаче аппроксимации функции с шумами. Особое внимание уделяется автоматизированному подбору гиперпараметров и на сравнительному анализу качества созданных моделей. Оценка проводится на нескольких типах данных с применением различных алгоритмов оптимизации.

4.1 Реализация нейронных сетей в среде Wolfram Mathematica

В этом разделе демонстрируется пошаговый процесс создания, конфигурации и использования нейронной сети на языке `Wolfram Language` для решения задачи аппроксимации функций. Такой подход позволяет не только продемонстрировать возможности выбранной среды, но и проанализировать ключевые аспекты реализации, влияющие на качество и эффективность моделей. В качестве дополнительных источников, раскрывающих особенности настройки нейронных сетей в среде Wolfram Mathematica, можно обратиться к источникам [2, 4].

На первом этапе необходимо определить параметры задачи и функцию, которую требуется аппроксимировать с помощью нейронной сети. Для большей наглядности выбирается двухмерная поверхность с характерными элементами рельефа, что позволяет проиллюстрировать процесс построения и обучения модели на данных с ярко выраженными нелинейностями. В качестве переменных задаются диапазоны по оси абсцисс и ординат, а также границы отображения поверхности. Такой подход обеспечивает визуальное представление исходной задачи и позволяет на практике оценить, насколько эффективно построенная модель справляется с восстановлением сложной

зависимости на заданном промежутке. На рисунке 4.1 отображен график поверхности с элементами рельефа, функция которой будет аппроксимирована.

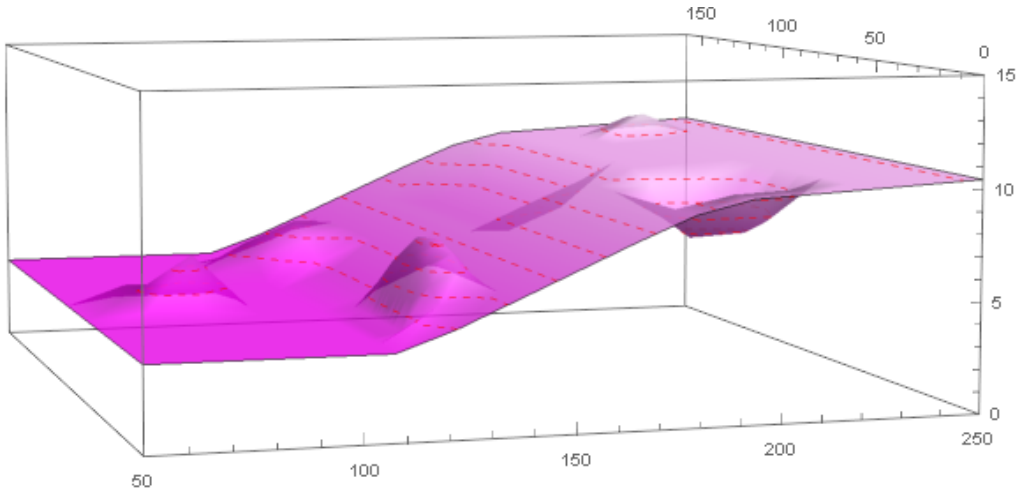

Рисунок 4.1 — График поверхности с типовыми элементами рельефа

Далее исходная выборка преобразуется в два зашумлённых набора данных. Для этого в значения функции вносятся псевдослучайные помехи, которые подчиняются равномерному и нормальному законам распределения. Этот метод даёт возможность оценить, насколько устойчиво и качественно. Рисунки 4.2 и 4.3 иллюстрируют полученные данные: зелёным показана исходная выборка, синим — выборка с равномерным шумом, а красным — с нормальным.

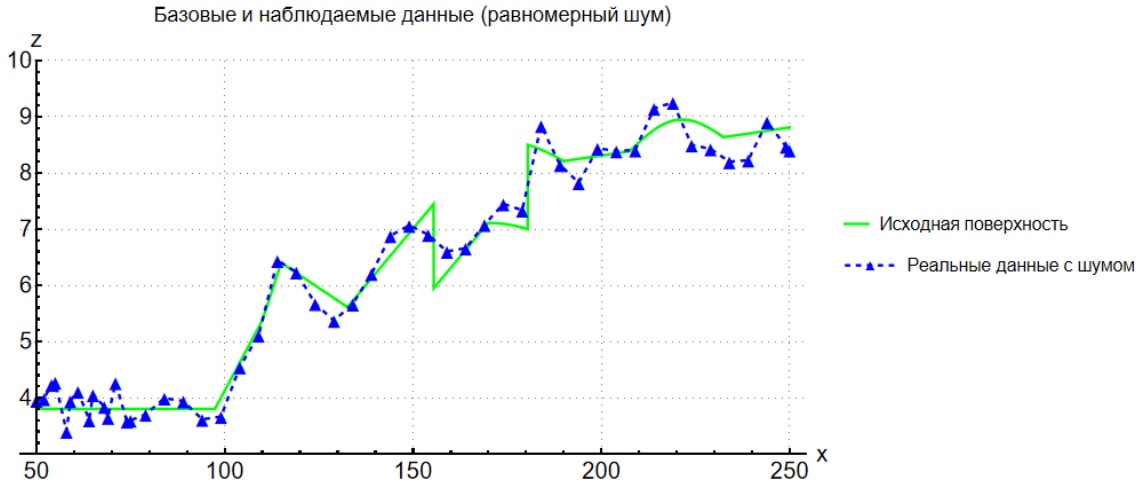

Рисунок 4.2 — Базовые и наблюдаемые наборы данных, равномерное распределение

Рисунок 4.3 — Базовые и наблюдаемые наборы данных, нормальное распределение

В рамках практической части исследования была реализована задача аппроксимации функции поверхности с использованием нейронных сетей в среде Wolfram Mathematica. Для построения моделей применялась функция `NetChain`, позволяющая гибко формировать структуру сети за счет добавления скрытых слоев различной мощности. Для повышения качества аппроксимации особое внимание уделялось выбору числа нейронов в каждом скрытом слое — этот параметр подбирался индивидуально для каждой конфигурации сети, что обеспечивало поиск наилучшего баланса между точностью модели и её сложностью. В процессе эксперимента были рассмотрены четыре варианта моделей — комбинации двух методов оптимизации (`ADAM` и `RMSProp`) и двух типов шума в данных (равномерное и нормальное распределения). В качестве функции активации во всех случаях использовался гиперболический тангенс, что позволило добиться необходимой нелинейности при аппроксимации сложных поверхностей.

Важной особенностью данной работы стал не просто подбор гиперпараметров методом случайного перебора, но и использование регрессионного подхода для ускорения и автоматизации поиска оптимальных значений. На первом этапе был сгенерирован набор случайных конфигураций гиперпараметров (размер скрытого слоя, тип оптимизатора, число эпох), для каждой из которых выполнялось обучение сети и вычислялась итоговая метрика качества (MSE). Полученные результаты были аккуратно структурированы в виде обучающей выборки для построения регрессионной модели. В качестве регрессионной модели использовался алгоритм случайного леса (`RandomForest`). Такая модель позволила выявить зависимость между значе-

ниями гиперпараметров и среднеквадратичной ошибкой (MSE), тем самым существенно ускорив процесс поиска оптимальных конфигураций. Это позволило заменить дорогостоящий полный перебор параметров более интеллектуальным и быстрым предсказанием, что особенно актуально при ограниченных вычислительных ресурсах.

Для каждого типа шума (равномерный/нормальный) и каждого оптимизатора (`ADAM`, `RMSProp`) были определены конфигурации, минимизирующие MSE (см. таблицу 4.1):

Таблица 4.1 — Оптимальные параметры, прогнозируемые и реальные MSE лучших моделей в Wolfram Mathematica

Модель	**HiddenSize**	**Epochs**	**Predicted MSE**	**Real MSE**
Uniform + ADAM	250	200	0.330587	0.260127
Uniform + RMSProp	30	200	0.231552	0.195987
Normal + ADAM	250	200	0.448879	0.395219
Normal + RMSProp	100	250	0.352595	0.319599

Графическая иллюстрация результатов (рисунки 4.4–4.7) демонстрирует, что такой подход позволяет добиться более стабильной и быстрой оптимизации, а итоговые аппроксимирующие функции существенно точнее повторяют исходную поверхность даже при наличии шума в данных.

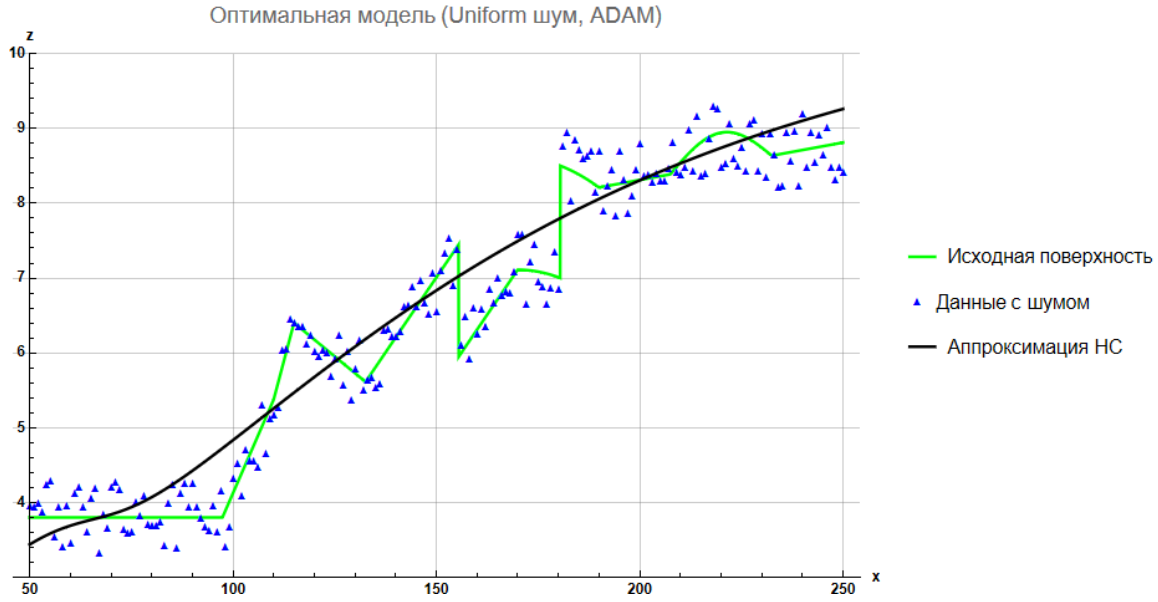

Рисунок 4.4 — График аппроксимации функции с использованием метода ADAM при равномерном распределении шума

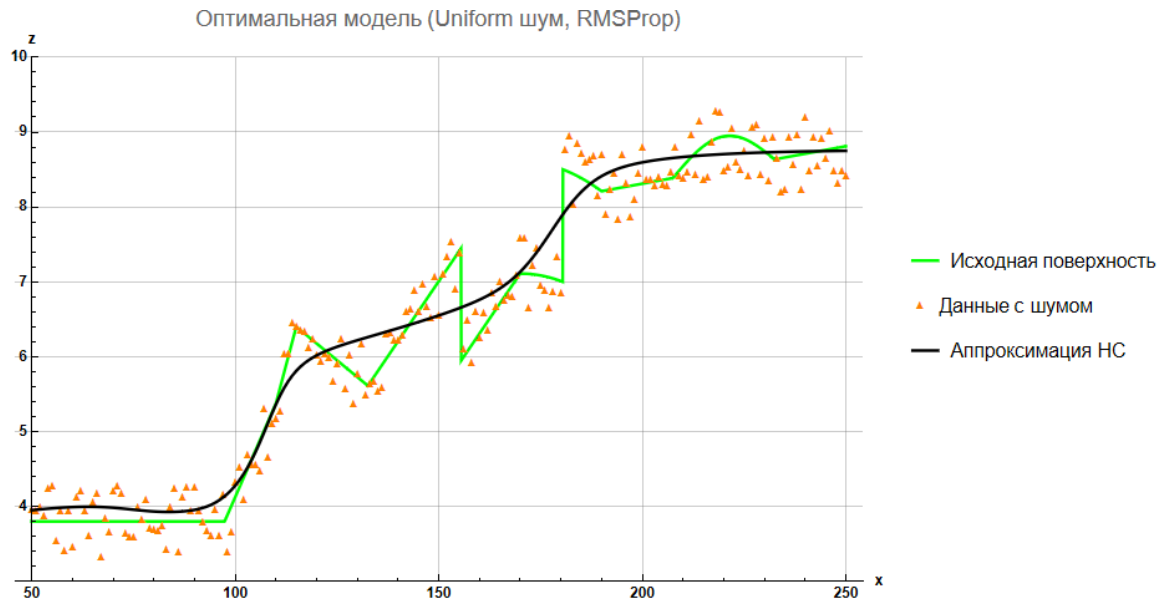

Рисунок 4.5 — График аппроксимации функции с использованием метода RMSProp при равномерном распределении шума

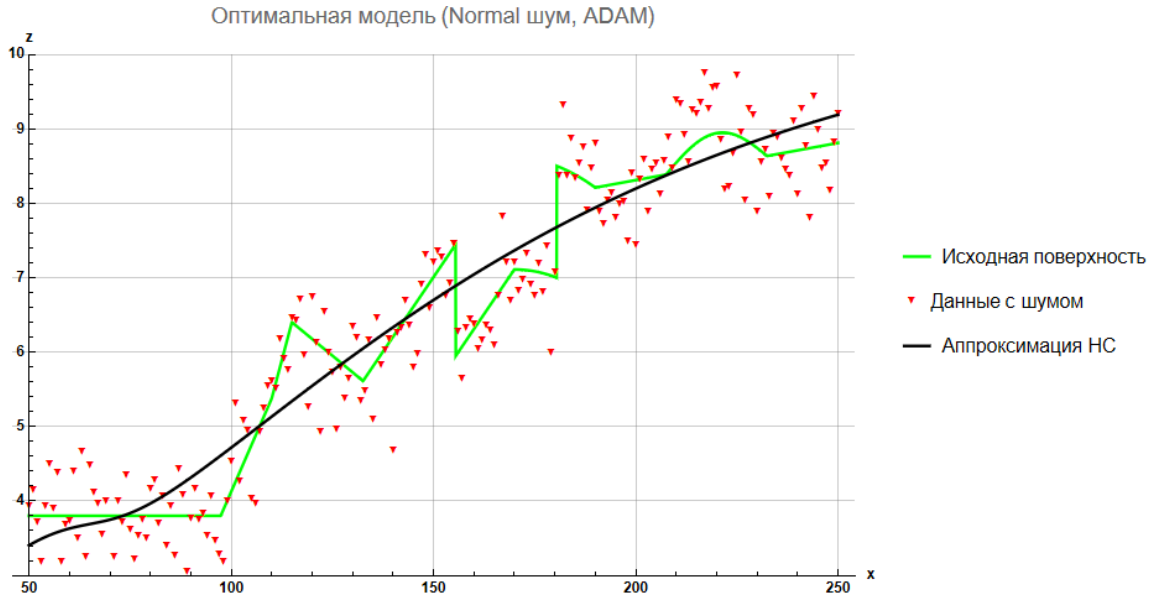

Рисунок 4.6 — График аппроксимации функции с использованием метода ADAM при нормальном распределении шума

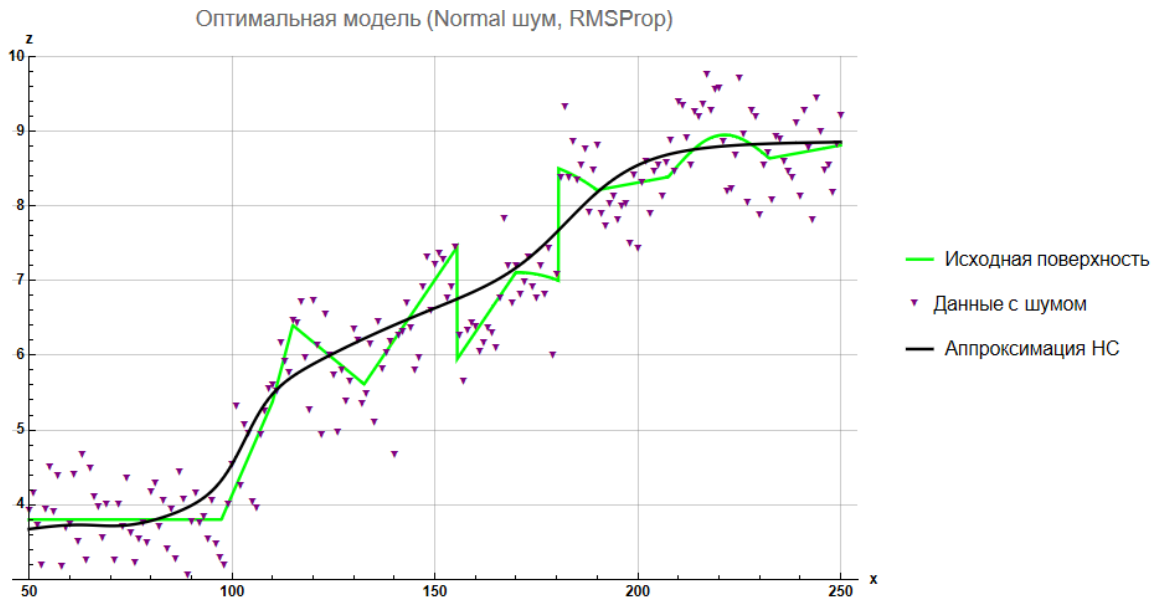

Рисунок 4.7 — График аппроксимации функции с использованием метода RMSProp при нормальном распределении шума

Таким образом, интеграция регрессионных методов в процесс подбора гиперпараметров продемонстрировала свою эффективность и применимость не только для ускорения вычислений, но и для повышения итогового качества моделей, что подтверждается полученными минимальными значениями MSE для всех экспериментальных конфигураций.

4.2 Реализация нейронных сетей в среде Python

В данной части исследования задача аппроксимации функции поверхности с добавленным шумом реализована на языке программирования Python, что обусловлено его гибкостью, широким выбором специализированных библиотек и высокой популярностью в сфере машинного обучения. Особое внимание в работе было уделено автоматизации поиска оптимальных гиперпараметров, что позволило сделать процесс построения и оценки нейронных сетей более объективным и воспроизводимым.

Для построения моделей использовалась библиотека `TensorFlow` с интерфейсом `Keras`. Архитектура нейронных сетей повторяла ранее применённую структуру в среде Wolfram Mathematica и включала два скрытых слоя с переменным числом нейронов, функцию активации `tanh` и регуляризацию методом `Dropout`. Благодаря этому обеспечивалась корректность сравнительного анализа.

Существенным преимуществом реализации на Python стало применение современного инструмента автоматизированного подбора гиперпараметров `Optuna`. Это позволило отказаться от ручного и полуавтоматического перебора конфигураций, значительно сократить продолжительность экспериментов и исключить субъективность при выборе параметров. Подбор оптимальных значений числа нейронов, количества эпох и типа оптимизатора (`ADAM` или `RMSProp`) осуществлялся отдельно для равномерного и нормального распределения шума с целью минимизации среднеквадратичной ошибки на валидационных данных.

Генерация исходных и зашумленных данных осуществлялась по тем же принципам, что и в реализации на Wolfram Mathematica, что обеспечило идентичность исходных условий для сравнения. Входные значения были нормализованы, что положительно сказывалось на устойчивости и скорости обучения моделей. Для каждой конфигурации автоматически создавалась модель с заданными параметрами, осуществлялось обучение с выбранным оптимизатором, после чего производилась оценка точности аппроксимации.

На следующем шаге мы приступаем к построению нейронной сети с прямой связью, использующей алгоритм обратного распространения ошибки. За основу взята модель `Sequential`, в которую добавлены три скрытых слоя. В первых двух слоях задано по 150 нейронов. Общий принцип прямого распространения для трёхслойной архитектуры отражён в формуле (4.1).

$$\begin{aligned} h_1^{(2)} &= f\left(w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + w_{13}^{(1)}x_3 + b_1^{(1)}\right), \\ h_2^{(2)} &= f\left(w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 + w_{23}^{(1)}x_3 + b_2^{(1)}\right), \\ h_3^{(2)} &= f\left(w_{31}^{(1)}x_1 + w_{32}^{(1)}x_2 + w_{33}^{(1)}x_3 + b_3^{(1)}\right), \\ h_W(x) &= f\left(w_{11}^{(2)}h_1^{(2)} + w_{12}^{(2)}h_2^{(2)} + w_{13}^{(2)}h_3^{(2)} + b_1^{(2)}\right), \end{aligned} \tag{4.1}$$

где $h_i^{(2)}$ – выход узлов во втором слое, $f(x)$ – активационная функция. В качестве передаточной функции применяется гиперболический тангенс. Создание модели и добавление слоёв показано в листинге 4.1.

```
def train_network(data, hidden_size, optimizer, epochs, verbose=0)
    :
    x = data[:,0].reshape(-1,1)
    y = data[:,1].reshape(-1,1)
    model = keras.Sequential([
        layers.Dense(
            hidden_size, activation='tanh', input_shape=(1,)
        ),
        layers.Dropout(0.1),
        layers.Dense(hidden_size, activation='tanh'),
        layers.Dropout(0.1),
        layers.Dense(1)
    ])
    model.compile(optimizer=optimizer, loss='mse')
    model.fit(x, y, epochs=epochs, verbose=verbose)
    return model
```

Листинг 4.1 — Создание модели и добавление слоёв

В рамках исследования для построенных моделей в качестве функции потерь использовалась стандартная среднеквадратичная ошибка (mean squared error, MSE), которая позволяет количественно оценить степень отклонения прогнозируемых значений, вычисленных нейронной сетью, от фактических результатов. Применение MSE обеспечивает объективную и удобную для интерпретации метрику качества аппроксимации, способствуя эффективному контролю процесса обучения и оптимизации модели. Математическое выражение данной функции приведено в формуле (4.2).

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(a(x_i) - y_i)^2, \tag{4.2}$$

В качестве алгоритмов оптимизации для настройки весов нейронной сети применялись современные адаптивные методы — «ADAM» и «RMSProp». Оба этих метода основаны на инструментах математического анализа и обеспечивают ускоренную сходимость, гибкую адаптацию скорости обучения и более стабильную динамику корректировки параметров по сравнению с классическими подходами.

Следующим шагом является тренировка нейросети, которая выполняется с помощью метода `fit()`. Метод принимает на вход обучающую выборку X и соответствующие ей целевые значения Y. В ходе каждой итерации модель:

- генерирует прогнозы на основе входных данных,
- оценивает ошибку, сравнивая их с правильными ответами;
- корректирует свои параметры для повышения точности.

Для проверки результатов обучения и получения прогнозов на новых данных используется функция `predict()`, позволяющая применять уже обучённую сеть для обработки неизвестных примеров и анализа её практической эффективности.

Ключевым элементом реализации стала интеграция современного инструмента автоматизированного подбора гиперпараметров — `Optuna`. Этот фреймворк позволяет значительно повысить эффективность поиска оптимальных конфигураций моделей по сравнению с ручным перебором. В отличие от традиционных подходов, `Optuna` использует интеллектуальные алгоритмы оптимизации (например, байесовскую оптимизацию), позволяя с высокой скоростью исследовать пространство параметров и находить значения, минимизирующие целевую функцию. В рамках эксперимента подбирались следующие гиперпараметры:

- число нейронов в скрытых слоях (в диапазоне от 100 до 300);
- количество эпох обучения (от 100 до 300);
- тип оптимизатора (ADAM или RMSProp).

В качестве целевой функции использовалась среднеквадратичная ошибка (MSE) аппроксимации на валидационной выборке, а оптимизация проводилась независимо для каждой конфигурации исходных данных — с равномерным и нормальным распределением шума.

Процесс подбора параметров через `Optuna` реализовывался следующим образом. Для каждого испытания (trial) автоматически создавалась и обучалась нейронная сеть с очередной комбинацией параметров. После обучения вычислялась метрика MSE, которая передавалась в систему `Optuna` для оценки и дальнейшего поиска. Алгоритмы платформы направляют поиск в те области пространства параметров, где вероятность найти лучшее значение ошибки выше, что позволяет существенно сократить количество необязательных вычислений. Автоматизация процесса позволила не только ускорить подбор параметров, но и провести более глубокий анализ. Настройка подбора и выявление оптимальных параметров показаны в листинге 4.2.

```
def objective(trial, data):
    hidden_size = trial.suggest_int('hidden_size', 30, 250)
    optimizer = trial
        .suggest_categorical('optimizer', ['adam', 'rmsprop'])
    epochs = trial.suggest_int('epochs', 40, 350)
    model = train_network(data, hidden_size, optimizer, epochs)
    x = data[:,0].reshape(-1,1)
    y = data[:,1]
    y_pred = model.predict(x, verbose=0).flatten()
    mse = np.mean((y - y_pred)**2)
    return mse

study_uniform = optuna.create_study(direction='minimize')
study_uniform.optimize(lambda trial: objective(trial,
   uniformNoisyPointsNorm), n_trials=25)
best_uniform = study_uniform.best_params

study_normal = optuna.create_study(direction='minimize')
study_normal.optimize(lambda trial: objective(trial,
   normalNoisyPointsNorm), n_trials=25)
best_normal = study_normal.best_params

print(f"Uniform:␣{best_uniform}")
print(f"Normal:␣{best_normal}")
```

Листинг 4.2 — Схема алгоритма подбора гиперпараметров для 2-ух наборов данных

Результаты оптимизации через `Optuna` представлены в таблице 4.2. Для каждого типа шума и выбранного оптимизатора были получены конфигурации, обеспечивающие наименьшее значение ошибки аппроксимации.

Таблица 4.2 — Оптимальные параметры, прогнозируемые и реальные MSE лучших моделей полученные Optuna

Набор данных	**HiddenSize**	**Epochs**	**Predicted MSE**	**Real MSE**
Uniform + ADAM	31	347	0.404564	0.376503
Uniform + RMSProp	143	127	0.381552	0.415987
Normal + ADAM	56	319	0.328879	0.385219
Normal + RMSProp	142	346	0.306595	0.308599

Для каждой из четырёх комбинаций типа шума и метода оптимизации была выбрана отдельная модель нейронной сети с оптимально подобранны-

ми гиперпараметрами. Визуализация результатов аппроксимации позволяет наглядно оценить качество восстановления исходной функции поверх зашумленных данных. Для этого воспользуемся библиотекой `matplotlib`. На рисунках 4.8, 4.9, 4.10 и 4.11 показаны реальные экспериментальные данные, истинная функция и предсказания нейронных сетей, полученные для каждой конфигурации.

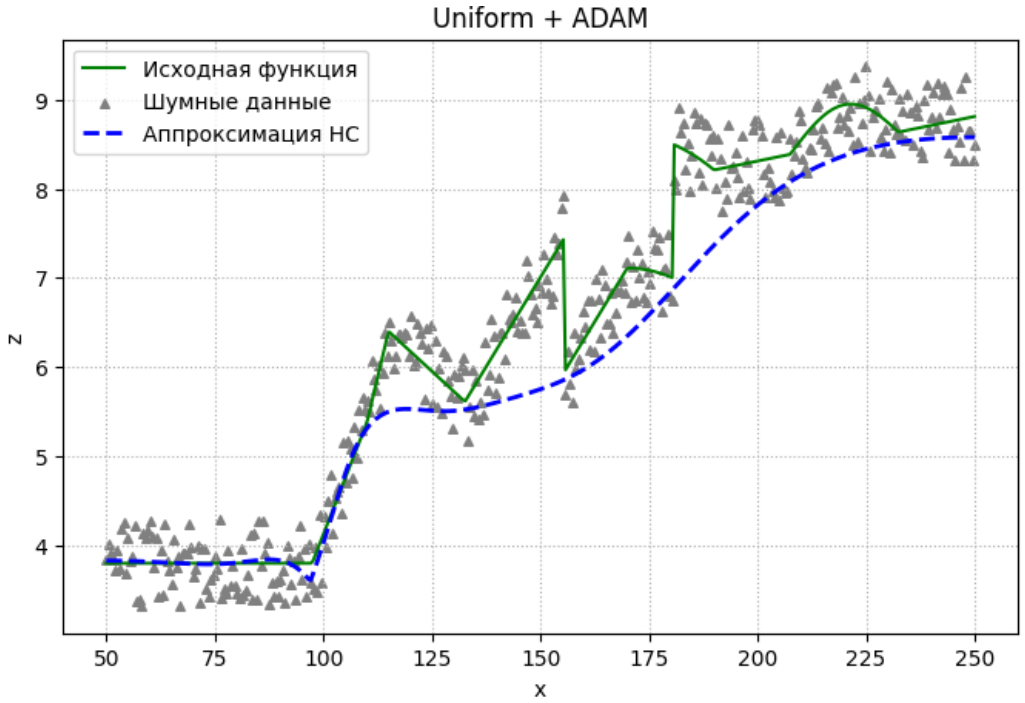

Рисунок 4.8 — График аппроксимации функции с использованием метода ADAM при равномерном распределении шума

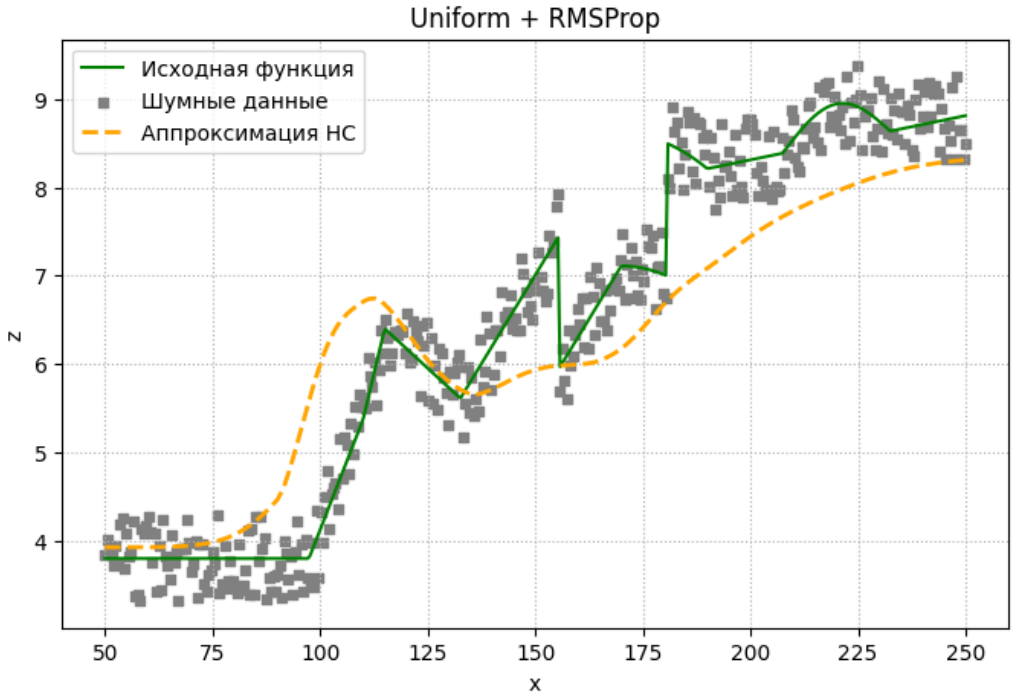

Рисунок 4.9 — График аппроксимации функции с использованием метода RMSProp при равномерном распределении шума

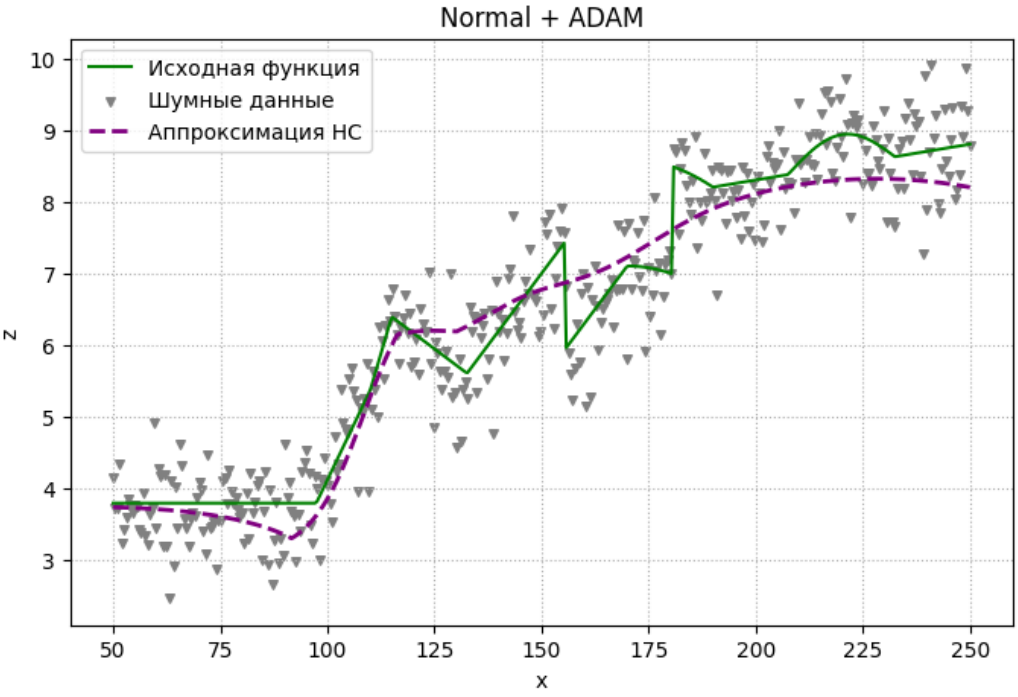

Рисунок 4.10 — График аппроксимации функции с использованием метода ADAM при нормальном распределении шума

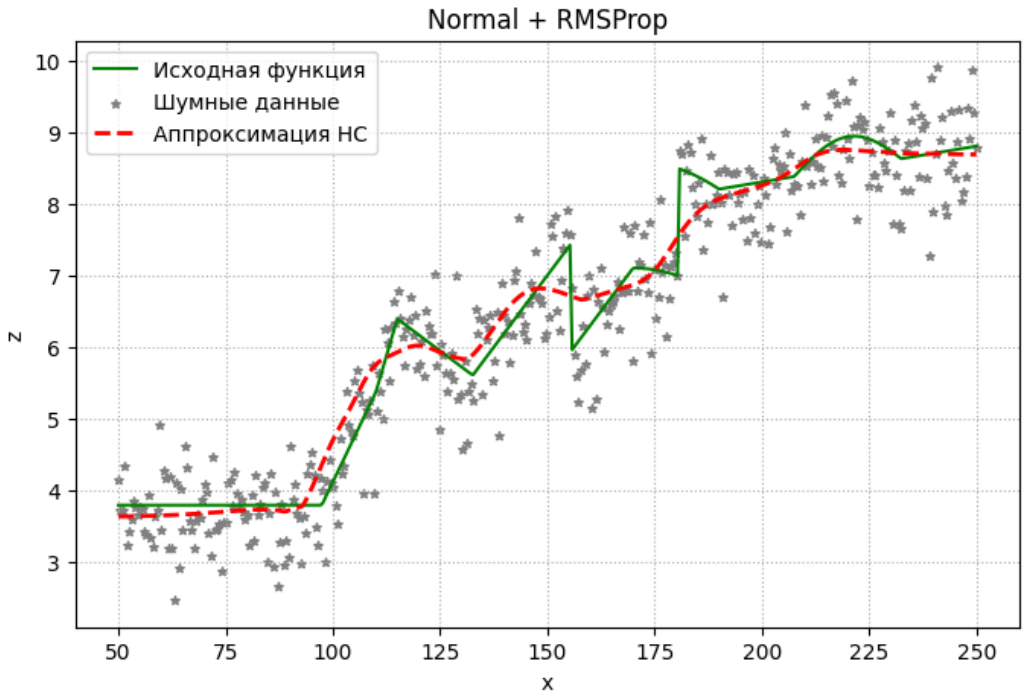

Рисунок 4.11 — График аппроксимации функции с использованием метода RMSProp при нормальном распределении шума

Видно, что наилучшие результаты по метрике MSE были получены для наборов с нормальным шумом, однако и для равномерного шума качество аппроксимации остается на высоком уровне. Различия в значениях ошибки между комбинациями минимальны, что подтверждает корректность выбора архитектуры моделей и эффективности процесса автоматизированной оптимизации гиперпараметров.

В целом, проведённый эксперимент показал, что применение современных средств автоматизации в подборе гиперпараметров позволяет не только повысить воспроизводимость, но и достичь устойчиво высоких результатов аппроксимации для различных условий постановки задачи.

ЗАКЛЮЧЕНИЕ

Выполненное исследование включает в себя комплексное сопоставление современных техник создания, обучения и тонкой настройки нейронных сетей, продемонстрированное на задаче аппроксимации функции с шумами. Акцент в равной степени сделан на теоретические принципы проектирования моделей и на практические аспекты автоматизации поиска гиперпараметров. Данное исследование было реализовано на базе двух технологических стеков: `Wolfram Mathematica` и `Python` (`TensorFlow/Keras`).

В рамках работы реализованы и протестированы программные решения для обеих платформ. Это позволило оценить их функциональные возможности, удобство, гибкость настройки и качество результатов в едином экспериментальном контуре. Для каждой комбинации распределения шума и метода оптимизации индивидуально подобраны гиперпараметры моделей с помощью автоматизированных методов поиска, что позволило минимизировать человеческий фактор и повысить воспроизводимость эксперимента.

Ключевые результаты продемонстрировали, что автоматизация процесса оптимизации гиперпараметров существенно повышает эффективность моделирования и обеспечивает более стабильные и объективные итоги. При этом обе исследуемые платформы подтвердили свою пригодность для сложных задач анализа и аппроксимации данных: реализованные нейронные сети показали высокое качество восстановления исходной функции даже в условиях существенного шумового искажении входных данных.

Полученные результаты были представлены в удобном и понятном виде — в форме графиков и сводных таблиц. Это помогло наглядно сравнить возможности использованных платформ и выявить сильные и слабые стороны каждой из них. Экспериментальные данные показали, что точность аппроксимации функции зависит не только от выбранной платформы, но и от характеристик шума в исходных данных, а также от метода оптимизации гиперпараметров. Вместе с тем, при правильной настройке обе исследуемые платформы (`Wolfram Mathematica` и `Python` с библиотеками `TensorFlow`, `Keras` и `Optuna`) способны стабильно показывать хорошие результаты.

Подводя итог, можно отметить, что данная работа подтвердила важность использования современных инструментов автоматизации для подбора гиперпараметров нейронных сетей. Грамотное сочетание теоретических знаний

и практических навыков позволяет эффективно решать задачи машинного обучения любой сложности. Выводы и разработанные методики могут стать полезными не только для дальнейших исследований, но и для реальных прикладных задач в области анализа данных.

Результаты данной работы были успешно представлены автором на международном уровне в рамках III международной научно-практической конференции студентов и аспирантов «Цифровизация экономики и финансов: модели, методы и технологии» проводимой Финансовым Университетом при Правительстве Российской Федерации, где автор был удостоен диплома II степени (см. Приложение Б). Это подчёркивает практическую значимость и актуальность проведённого исследования.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

[1] J. Bergstra и др. «Algorithms for Hyper-Parameter Optimization». В: *Advances in Neural Information Processing Systems*. Т. 24. 2011.

[2] *Neural Networks. [Электронный ресурс]*. `https://reference.wolfram.com/language/guide/NeuralNetworks.html`. Дата доступа: 23.02.2025.

[3] C. Nwankpa и др. *Activation Functions: Comparison of trends in Practice and Research for Deep Learning*. `https://arxiv.org/abs/1811.03378`. Дата доступа: 08.06.2025. 2018.

[4] *Wolfram Data Repository. [Электронный ресурс]*. `https://datarepository.wolframcloud.com`. Дата доступа: 23.02.2025.

[5] А.И. Галушкин. *Нейронные сети: основы теории*. Москва: Горячая линия – Телеком, 2012, с. 496.

[6] Ф.М. Гафаров и А.Ф. Галимянов. *Искусственные нейронные сети и их приложения*. Казань: Издательство Казанского университета, 2018, с. 121.

[7] В.А. Головко и В.В. Краснопрошин. *Нейросетевые технологии обработки данных*. Издание университетское, учебное пособие. Минск: БГУ, 2017, с. 270.

[8] Антонио Джулли и Суджит Пал. *Библиотека Keras - инструмент глубокого обучения. Реализация нейронных сетей с помощью библиотек Theano и TensorFlow*. Москва: ДМК Пресс, 2018, с. 294.

[9] *Каталог «WOLFRAM Demonstrations Project». [Электронный ресурс]*. `https://demonstrations.wolfram.com`. Дата доступа: 10.12.2024.

[10] *Краткое руководство по Tensorflow. [Электронный ресурс]*. `https://pythonist.ru/kratkoe-rukovodstvo-po-tensorflow/`. Дата доступа: 23.02.2025.

[11] *Нейронные сети. [Электронный ресурс]*. `http://statsoft.ru/home/textbook/modules/stneunet.html`. Дата доступа: 23.02.2025.

[12] *Справочный центр Wolfram Language. [Электронный ресурс]*. `https://reference.wolfram.com/language/`. Дата доступа: 09.06.2025.

[13] В.Б. Таранчук. *Основы программирования на языке Wolfram: учеб. материалы для студентов фак. прикладной математики и информатики спец. 1-31 03 04 «Информатика»*. Минск: БГУ, 2015, с. 49.

[14] В.Б. Таранчук. *Введение в графику системы Mathematica: учеб. материалы для студентов фак. прикладной математики и информатики.* Минск: БГУ, 2017, с. 53.

[15] *Теория Python. [Электронный ресурс].* `https://www.yuripetrov.ru/edu/python/ch_02_01.html`. Дата доступа: 23.02.2025.

ПРИЛОЖЕНИЕ А

ПРИМЕР ПОСТРОЕНИЯ И ОБУЧЕНИЯ МОДЕЛИ В TENSORFLOW

```
import tensorflow as tf
from tensorflow import keras

# Загрузка и подготовка данных (например, MNIST)
(x_train, y_train), (x_test, y_test) =
 ↪  keras.datasets.mnist.load_data()
x_train = x_train.reshape(-1, 28 * 28).astype("float32") / 255
x_test = x_test.reshape(-1, 28 * 28).astype("float32") / 255

# Определение архитектуры модели
model = keras.Sequential(
    [
        keras.layers.Dense(128, activation="relu",
         ↪  input_shape=(784,)),
        keras.layers.Dense(64, activation="relu"),
        keras.layers.Dense(10, activation="softmax"),
    ]
)

# Компиляция модели
model.compile(
    optimizer="adam", loss="sparse_categorical_crossentropy",
     ↪  metrics=["accuracy"]
)

# Обучение модели
model.fit(
    x_train, y_train, epochs=10, batch_size=128,
     ↪  validation_split=0.1
)

# Оценка на тестовых данных
test_loss, test_acc = model.evaluate(x_test, y_test)
print("Test accuracy:", test_acc)
```

ПРИЛОЖЕНИЕ Б

ДИПЛОМ II СТЕПЕНИ МЕЖДУНАРОДНОЙ НАУЧНО-ПРАКТИЧЕСКОЙ КОНФЕРЕНЦИИ СТУДЕНТОВ И АСПИРАНТОВ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ

ДИПЛОМ

II степени

III Международная научно-практическая конференция студентов и аспирантов
III International Scientific and Practical Conference of Students and Postgraduates
Цифровизация экономики и финансов: модели, методы и технологии
Digitalization of economics and finance: models, methods and technologies

Шушкевич Дмитрий Витальевич
Лебедевич Артем Владимирович
Белорусский Государственный Университет, Минск, Республика Беларусь

Алгоритмы и инструменты настройки и оценки эффективности нейронных сетей, сопоставление программных реализаций

Научный руководитель: Таранчук В. Б.,д.ф.-м.н., профессор, профессор кафедры компьютерных технологий и систем
Белорусский Государственный Университет, Минск, Республика Беларусь

Председатель программного комитета, Проректор по цифровизации Финансового университета

Остапенко Г.А.

г. Москва, 23-24 апреля 2025 г.