МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Кафедра компьютерных технологий и систем

Жук
Даниил Дмитриевич

МЕТОДЫ И ИНСТРУМЕНТЫ НАСТРОЙКИ НЕЙРОННЫХ СЕТЕЙ В СИСТЕМАХ КОМПЬЮТЕРНОЙ АЛГЕБРЫ

Дипломная работа

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

Допущена к защите

«___» _____ 20__ г.

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

Минск, 2025

ОГЛАВЛЕНИЕ

РЕФЕРАТ

Диплом содержит 59 страниц, 37 иллюстраций, 30 формул, 6 таблиц, 14 использованных литературных источников.

Ключевые слова: МЕТОДЫ, ИНСТРУМЕНТЫ, НЕЙРОННЫЕ СЕТИ, WOLFRAM MATHEMATICA, PYTHON, PYTORCH.

Объектом исследования являются инструменты для настройки нейронных сетей и методы их обучения и оценки эффективности.

Целью является исследование инструментов для настройки нейронных сетей и методов для обучения и оценки эффективности нейронных сетей в задаче аппроксимации гладкой функции и в задаче прогнозирования при помощи системы компьютерной алгебры Wolfram Mathematica и фреймворка PyTorch для языка программирования Python.

Методами исследования являются программирование на языках Wolfram Language и Python, изучение соответствующей литературы и электронных источников.

Результатами работы являются изученные возможности настройкии обучения нейронных сетей в задаче аппроксимации гладкой функций и в задаче прогнозирования с использованием методов языка Wolfram Language, средств системы компьютерной алгебры Wolfram Mathematica и языка программирования Python и фреймворка для глубокого обучения PyTorch, описанные достоинства платформ.

РЭФЕРАТ

Дыпломная праца ўтрымлівае 59 старонак, 37 малюнкаў, 30 формул, 6 табліц, 14 крыніц.

Ключавыя словы: МЕТАДЫ, ІНСТРУМЕНТЫ, НЕЙРОНАВЫЯ СЕТКІ, WOLFRAM MATHEMATICA, PYTHON, PYTORCH.

Аб’ектам даследавання з'яўляюцца інструменты для настройкі нейронавых сетак і метады іх навучання і ацэнкі эфектыўнасці.

Мэтай з'яўляецца даследаванне прылад для налады нейронавых сетак і метадаў для навучання і адзнакі эфектыўнасці нейронавых сетак у задачы апраксімацыі гладкай функцыі і ў задачы прагназавання пры дапамозе сістэмы кампутарнай алгебры Wolfram Mathematica і фрэймворка PyTorch для мовы праграмавання Python.

Метадамі даследавання з'яўляюцца праграмаванне на мовах Wolfram Language і Python, вывучэнне адпаведнай літаратуры і электронных крыніц.

Вынікамі працы з'яўляюцца вывучаныя магчымасці наладкі навучання нейронавых сетак у задачы апраксімацыі гладкай функцый і ў задачы прагназавання з выкарыстаннем метадаў мовы Wolfram Language, сродкаў сістэмы кампутарнай алгебры Wolfram Mathematica і мовы праграмавання Python і фрэймворка для глыбокага навучання PyTorch, апісаныя добрыя якасці.

ABSTRACT

The diploma work contains 59 pages, 37 illustrations, 30 formulas, 6 tables, 14 sources.

Keywords: METHODS, TOOLS, NEURAL NETWORKS, WOLFRAM MATHEMATICA, PYTHON, PYTORCH.

The object of the research is tools for tuning neural networks and methods for training and evaluating their performance.

The goal is a study of tools for tuning neural networks and methods for training and evaluating the performance of neural networks in smooth function approximation and prediction problems using Wolfram Mathematica and the PyTorch framework for the Python programming language.

Research methods include programming in Wolfram Language and Python, study of relevant literature and electronic sources.

The results of the work are the studied possibilities of tuning and training of neural networks in the task of approximation of smooth functions and in the task of prediction using methods of Wolfram Language, means of computer algebra system Wolfram Mathematica and programming language Python and framework for deep learning PyTorch, the described advantages of the platforms.

ВВЕДЕНИЕ

Глубокое обучение является перспективной отраслью, которая в последнее время набирает все большую актуальность. Крупнейшие компании, такие как Google, Amazon, Apple, Яндекс и другие внедряют искусственный интеллект в свои сервисы, а новые мобильные и графические процессоры оснащаются дополнительными нейросетевыми ядрами. Нейронные сети являются неотъемлемой частью глубокого обучения и являются мощными инструментом при работе с большими объемами данных.

Нейронные сети нашли применение в самых различных областях. Анализируя с помощью нейронных сетей запросы пользователей, медиа сервисы, такие как Spotify или YouTube формируют рекомендации и контекстную рекламу. Поисковые сервисы, такие как Google и Яндекс используют нейронные сети для выдачи краткого поискового результата, чтобы пользователю не требовалось посещать сторонние сайты. Камеры с помощью нейронных сетей регулируют движение транспорта в Москве: отслеживают скорость, наличие пристегнутых ремней безопасности. Нейронные сети также широко используются в медицине, промышленности и военной отрасли.

Нейронные сети могут быть задействованы в решении широкого спектра задач, где необходим анализ большого объема данных и нахождение нетривиальных закономерностей. К типовым задачам нейронных сетей относят классификацию, прогнозирование, распознавание. Для использования нейронных сетей для решения своих задач необходима мощная платформа, позволяющая удобно создавать нейронные модели и эффективно их обучать.

Python был разработан Гвидо ван Россумом в 1991 году. Это мультипарадигменный язык программирования, изначально проектируемый как объектно-ориентированный. Ключевыми особенностями Python являются динамическая типизация и автоматический контроль памяти. Python это интерпретируемый язык, ввиду чего он медленнее, чем языки C и C++. Однако, отсутствие необходимости компиляции ускоряет работу при экспериментах и обучении.

Python стал основным языком в сфере глубоко обучения, поэтому для него существует большое количество фреймворков глубокого обучения. Фреймворки глубокого обучения – это большие библиотеки, которые позволяют создавать, обучать и анализировать собственные нейронные модели. Обычно они содержат меньшие библиотеки, которые позволяют создавать нейронные сети, используя объектно-ориентированный подход.

Системы компьютерной алгебры – это программы, рассчитанные на математические вычисления. Они позволяют выполнять сложные

математический вычисления, визуализировать их. Некоторые из них также имеют функционал для работы с нейронными сетями.

Система компьютерной алгебры Wolfram Mathematica, начиная с 11 версии, обладает широким инструменталом для работы с нейронными сетями. В дополнение к этому, Wolfram Neural Net Repository обладает широким набором обученных моделей, которые можно использовать в своей работе.

Таким образом, актуальность темы дипломной работы обусловлена стремительным развитием нейронных сетей и сферы глубокого обучения. В данной работе, на основании изучения работ отечественных и зарубежных авторов, изложены основные теоретические аспекты нейронных сетей, средств оценки их эффективности и методов обучения.

Целью дипломной работы является исследование и описание методов настройки нейронных сетей в системе компьютерной алгебры Wolfram Mathematica, а также сравнение результатов решения двух задач: аппроксимация гладкой функции и прогнозирование с результатом решения аналогичных задач в одном из самых популярный фреймворков глубоко обучения для языка программирования Python – PyTorch.

ГЛАВА 1. ТЕОРЕТИКО-МЕТОДОЛОГИЧЕСКИЕ ВОПРОСЫ

Ниже пояснены основные теоретические аспекты нейронных сетей, их важность в контексте развития информационных технологий. Приведена характеристика литературным источникам, которые использовались в данной работе. На основе этих работ строится описание архитектур нейронных сетей, методов их обучения и оценки эффективности.

1.1 Теоретические аспекты

В настоящее время, когда технологии развиваются с огромной скоростью, а объемы данных, которые необходимо обрабатывать, увеличиваются с еще большей скоростью, возникает потребность в инструментах, способных анализировать большие объемы данных, искать на их основе закономерности и прогнозировать результаты. Инструментом, покрывающем такие задачи, является нейронная сеть.

Искусственная нейронная сеть – это математическая модель, которая в программном виде имитирует работу биологических нейронных сетей. Вдохновением для искусственных нейронных сетей послужили нейронные сети человеческого мозга. В основу обучения нейронных сетей также легли аспекты из обучения человека.

Нейронные сети применяются в самых различных областях. К основным областям применения можно отнести: обработка больших и сложных наборов данных, компьютерное зрение, обработка текста и голоса, медицина и биоинформатика, рекомендательные системы, искусственный интеллект.

К основным задачам, где используются нейронные сети, относятся распознавание, прогнозирование и классификация. Распознавание – это задача компьютера или программы опознать объекты из реальной жизни: объекты на изображении или видео, человеческая речь или звуки. Прогнозирование – задача предсказания результата в будущем на основе какого-то набора данных ща определенный период, например прогнозирование акций на фондовом рынке. Классификация – разбиение данных на классы на основе каких-то признаков. Может сочетаться с задачей распознавания, например, распознать на изображении человека и отнести его к какому-либо полу.

Приведем краткое описание использованных в работе библиографических источников.

Теоретические аспекты нейронных сетей были в основном взяты из книг [1, 5]. В книге [1] дается подробное описание истории возникновения и развития искусственных нейронных сетей, сходства с биологическими нейронными сетями, а также подробно описывается математическая модель

простейшей искусственной нейронной сети. В книге [5] подробно описываются самые распространенные архитектуры нейронных сетей, их особенности и основные механизмы для их обучения.

При построении задачи для описания алгоритмов обучения была использована задача из статьи [6]. В этой статье дается подробное описание метода стохастического градиентного спуска, а также доказывается его сходимостью. Описание алгоритмов обучения было взято из статьи [14]. В ней автор подробно описывает основные методы обучения нейронных сетей, в частности такие как RMSProp (среднеквадратичное распространение) и Adam (адаптивная оценка момента), которые мы будем использовать в данной дипломной работе.

Для изучения основных аспектов фреймворка PyTorch была использована документация PyTorch [11], туториалы, содержащие обучающие примеры [12] и книга [7].

Для изучения основных инструментов настройки нейронных сетей в Wolfram Mathematica была использована документация Wolfram [10].

При постановке задачи в качестве основы была взята задача из статьи [9]. В этой статье автор средствами Wolfram Mathematica генерирует поверхность, получает профиль этой поверхности и сглаживает его, используя нейронные сети в системе Wolfram Mathematica. Для решения поставленной задачи была взята нейронная сеть из примера [3].

Данные для решения задачи прогнозирования были взяты с электронного ресурса автомобильной ассоциации БАА [6], в котором собранны помесячные продажи новых автомобилей на территории Республики Беларусь, Национального статистического комитета Республики Беларусь [4], откуда были взяты данные, описывающие помесячную среднюю заработную плату населения, Национального Банка Республики Беларусь [2], откуда была взята помесячная кредитная ставка для физических лиц. Архитектура нейронная сеть для решения задачи прогнозирования была взята из примера Wolfram Mathematica [13].

1.2 Биологические основы нейронных сетей

Биологический нейрон. Нейронами называются особые клетки, обладающие электрической активностью. В головном мозге человека содержится около 100 миллиардов нейронов. Они соединены между собой в сложную сеть, которая играет роль в памятных, творческих и интеллектуальных процессах человека.

Нейроны состоят из следующих частей: тело нейрона, дендриты, аксоны. Дендриты представляют собой отростки, которые отвечают за

получение информации от других нейронов. Информация представляет собой электрический сигнал. Для передачи этого электрического сигнала используются аксоны. На концах аксонов находятся синапсы, которые отвечают за передачу сигнала. Многочисленные переплетения аксонов образуют нейронную сеть. На рисунке 1.1 представлен биологический вид нейронной клетки.

Таким образом, электрические сигналы передаются от одного нейрона к другому по всей нейронной сети, формируя нервный импульс. Такой принцип чем-то напоминает функционирование процессора компьютера. Распространение нервных импульсов отвечает за функционирование нервной системы человека, что в свою очередь формирует человеческую жизнедеятельность.

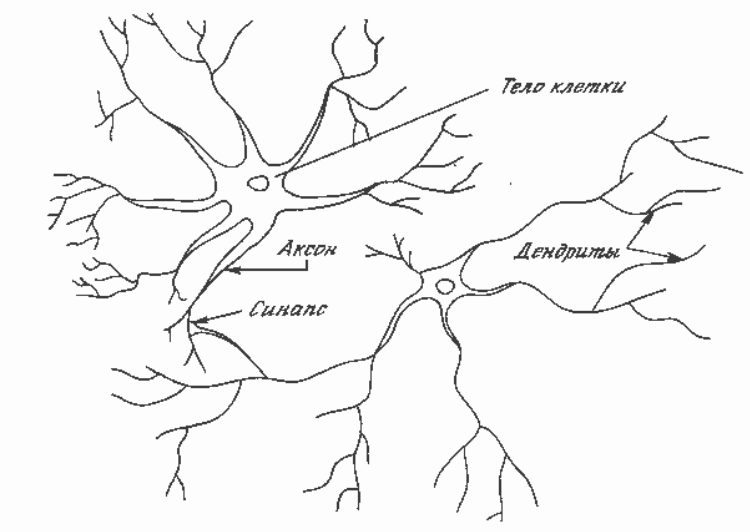

Рисунок 1.1 – Биологический нейрон

В синапсах электрический импульсы преобразовываются в химические сигналы, которые копят в теле нейрона положительные заряды, или положительные ионы. Когда число положительных ионов достигает порогового значения, в теле нейрона возникает электрический сигнал, который по аксоны передается к другому нейрону. Таким образом информация передается по нейронной сети. Передача, получение и преобразование информации в нейроне является нелинейным преобразованием, и она не сводится к простому линейному суммированию.

Аналогично полупроводнику в процессоре компьютера, нейрон человеческого мозга может быть в двух состояниях: выключенном и «включенном» или возбужденном состоянии. Частота передачи электрических сигналов в возбужденном состоянии больше, чем в выключенном. Частота в нейронах составляет от 10 Гц до 200 Гц, в свою очередь, частоты современных процессоров измеряются в ГГц. Однако, в отличии от процессоров,

человеческому мозгу для какой-либо «операции» требуется цепочка из куда меньшего числа нейронов, чем процессору простейших команд. К тому же, мозг эффективно работает в параллельном режиме.

Принято разделять человеческий мозг на два вещества: серое и белое. Серое вещество состоит из нейронных клеток и дендритов и отвечает за принятие и формирование нервных импульсов. Из серого вещества состоят основные зоны человеческого мозга, которые отвечают за жизнедеятельность человека. Белое вещество состоит из аксонов, покрытых миелином, что и дает ему белый цвет. В нем отсутствуют нейронные клетки, поэтому эта условная часть человеческого мозга отвечает за передачу сигналов по нейронной сети и связывает зоны человеческого мозга.

Физически мозг разделен на два полушария: левое и правое. Каждое полушарие разделено на особые зоны, которые отвечают за различные функции человека: память, воображение, речь, зрение и так далее.

Например, гиппокамп отвечает за функцию кратковременной памяти. Подобно кэшу процессора, он запоминает небольшой объем информации на короткий срок и в зависимости от надобности этой информации, либо отдает ее в зону долгосрочной памяти, либо забывает. Кратковременная память необходима для обучения человека, ведь в процессе обучения человек с течением времени трансформирует полученную информацию. Эти операции выполняются в зоне кратковременной памяти, а по истечению какого-то времени полученная в ходе обучения информация переходит в зону долгосрочной памяти.

Процесс обучения. Человек обучается в течении всей своей жизни, начиная с самого рождения. Взаимодействую с внешней средой, человек адаптируется к этой среде и к ее изменениям, формируя привычки, осваивая новые навыки. Во время обучения в человеческом мозге происходит динамическая перестройка нейронной сети. Связи между нейронами перестраиваются, формируются новые. Свойство нейронов изменяться является основой для обучения. Это свойство получило название синаптическая пластичность.

Синаптическая связь является основой для формирования новых связей в нейронных сетях, что в свою очередь формирует сознание человека. Д. Хебб на основе свойства синаптической пластичности сформулировал принцип обучения. Согласно этому принципу для усиления связи между пресинаптическим и постсинаптическим нейронами необходимо совпадение их активности во времени [1, с. 18].

Исследователи Л. Тауц и Э. Кендел в противовес принципу Д. Хебба предложили свой принцип обучения. Суть его заключается в наличии одного особенного нейрона между пресинаптическим и постсинаптическим

нейронами, который называется модулирующим нейроном. Этот модулирующий нейрона участвует в усилении синаптической связи между пресинаптическим и постсинаптическим нейронами без прямого участия постсинаптического нейрона.

Согласно описанным выше принципам Д. Хебба, и Т. Лауца и Э. Кендела, во время обучения происходит усиление синаптический связей, что в свою очередь формирует кратковременную память, что позволяет запоминать и модифицировать информацию в процессе обучения.

После обучения полученная информация отправляется в зону долговременной памяти. В отличии от кратковременной памяти, в основе долговременной памяти лежат более сложные процессы, в основе которых лежит участие определенных генов и создание новых синаптических связей.

В некоторых случаях обучения могут участвовать оба механизма запоминания, что позволяет человеку осваивать более сложную и комплексную информацию.

Можно выделить два вида обучения в зависимости от взаимодействия обучающего индивида с окружающей средой: обучение с учителем и обучения без учителя. При обучении с учителем у обучающегося индивида есть «учитель», в роли которого может выступать как отдельный индивид, так и какой-то набор информации. Учитель дает обучающемуся обратную связь, на основе которой обучающийся индивид корректирует свои ответы, пытаясь подогнать результат под результаты учителя, которые называют эталонными. Обучение сводится к минимизации разницы между результатом обучающегося и учителя.

Обучение без учителя сводится к взаимодействию обучающегося индивида с окружающей средой самостоятельно. В таком случае, обучающийся сам формирует связи, ищет последовательности и закономерности на основе предоставленной информации.

Во время обучения обучающийся индивид может получать положительную и отрицательную обратные связи. Отрицательная обратная связь служит для минимизации разницы между примером учителя и результатом обучающегося, с свою очередь положительная обратная связь ускоряет процесс обучения при взаимодействии обучающегося с внешней средой.

Таким образом, описанное строение биологических нейронов и механизмы обучения человеческого мозга послужили основой для построения математической модели искусственного нейрона и искусственных нейронных сетей.

1.3 Искусственные нейронные сети

Искусственные нейронные сети – это математическая модель, которая копирует поведения нейронов человеческого мозга и предназначенная для обработки информации. Первая модель искусственного нейрона была представлена У. Маккаллоком и У. Питтсом в 1943 году. На примере этой модели Маккаллок и Питтс продемонстрировали, как искусственный нейрон может передавать информацию. В 1950-х года на основе модели Маккаллока-Питтса Ф. Розенблатт предложил свою модель искусственного нейрона – перцептрон. Перцептрон стал первой работающей моделью глубокого обучения и положил основу для развития искусственных нейронных сетей. Однако уже в 1960-х годах Минский и Паперт издали книгу «Перцептроны», в которой указали на ограниченность перцептрона, что значительно затормозило процесс развития нейронных сетей.

В 1980-х годах развитие интерес к нейронным сетям снова возрос. Компьютерная техника значительно продвинулась по части производительности с 1960-х и были разработаны новые модели многослойных перцептронов. Также были предложены новые алгоритмы для обучения нейронных сетей, такие как алгоритм обратного распространения ошибки. В 2000-х годах началось развитие глубоких нейронных сетей. Производительные мощности тогдашних систем уже позволяли строить сложные многослойные модели с механизмами памяти, использовать ресурсоемкие алгоритмы, обрабатывать большие объемы данных. Это послужило толчком к развитию направления глубокого обучения. Развитие глубокого обучения привело к прогрессу в области обработки речи, распознавания образов, компьютерного зрения.

Основным элементом нейронной сети является искусственный нейрон. Он выполняет операцию нелинейного преобразования суммы произведения входных сигналов на весовые коэффициенты [1, с. 26]. Это можно выразить через формулу (1.1).

$$y = F\left(\sum_{i=1}^{n} \omega_i x_i\right) = F(WX), \tag{1.1}$$

где F — оператор нелинейного преобразования;

$W = (\omega_1, \omega_2, \dots, \omega_n)$ — вектор весов;

$X = (x_1, x_2, \dots, x_n)^T$ — входной вектор.

На рисунке 1.2 схема строения искусственного нейрона в матричном виде.

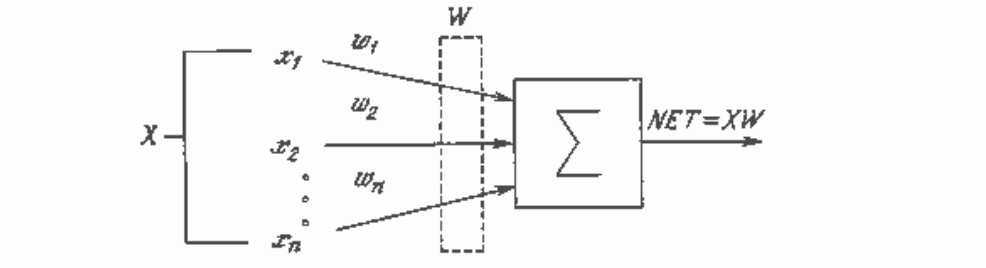

Рисунок 1.2 – Искусственный нейрон

Каждому i-му входу нейрона соответствует весовой коэффициент ω_i, который характеризует силу синаптической связи. Если $\omega_i > 0$, то сила связи называется усиливающей, в противном случае – тормозящей.

Оператор нелинейного преобразования F называется функцией активации нейронного элемента. Выбор функции активации зависит от конкретной задачи и типа нейронной сети. Приведем некоторые из наиболее распространенных:

1. Линейная функция. Линейная функция представляет собой прямую линию и пропорциональна входу. Так как она не ограничена, то можно получить целый спектр различных значений. Однако из-за того, что производная линейной функции константа, она непригодна для обучения методом градиентного спуска. График линейной функции активации представлен на рисунке 1.3.

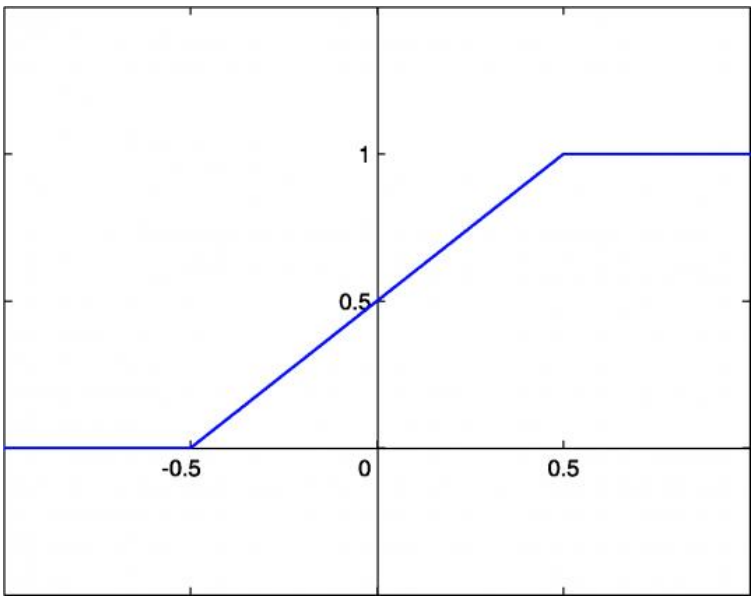

Рисунок 1.3 – Линейная функция активации

2. Сигмоидная функция. Эта функция представляет собой непрерывную, возрастающую, ограниченную функцию в диапазоне значений [0, 1]. Определяется по формуле (1.2). График сигмоидной функции активации представлен на рисунке 1.4.

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

где x – вход.

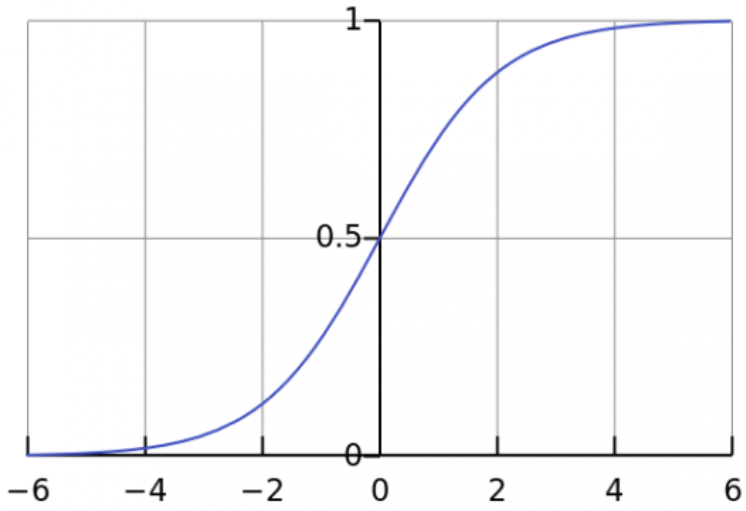

Рисунок 1.4 – Сигмоидная функция активации

3. ReLU (ректификационная функция активации). Эта функция возвращает 0 для всех отрицательных значений, а при положительных аргументах возвращает этот же аргумент. ReLU имеет простую форму и обеспечивает быструю вычислительную скорость. Задать ее можно по формуле (1.3). График функции активации ReLU представлен на рисунке 1.5.

$$f(x) = \max(0, x), \tag{1.3}$$

где x – вход.

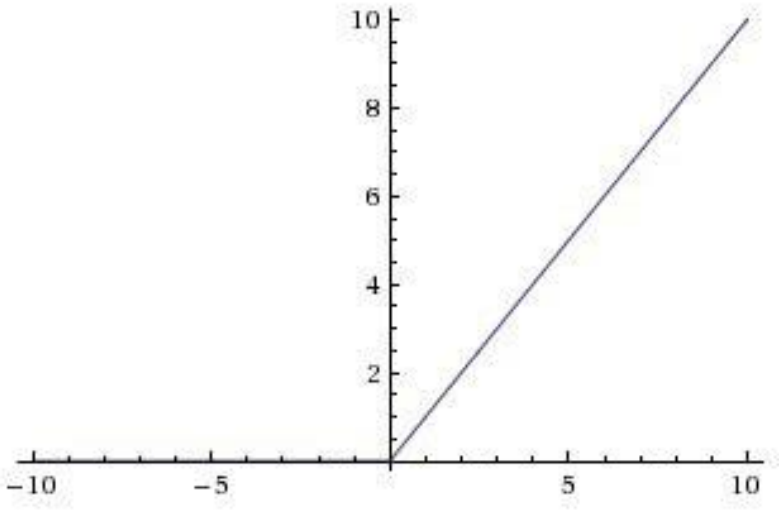

Рисунок 1.5 – Функция активации ReLU

4. Гиперболический тангенс. Эта функция преобразует входное значение в диапазоне от -1 до 1. В отличие от сигмоидной функции, гиперболический тангенс имеет симметричную форму и может быть полезен при работе с нулевым центром данных. Определяется по формуле (1.4). График гиперболического тангенса представлен на рисунке 1.6.

$$f(x) = th(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{1.4}$$

где x – вход.

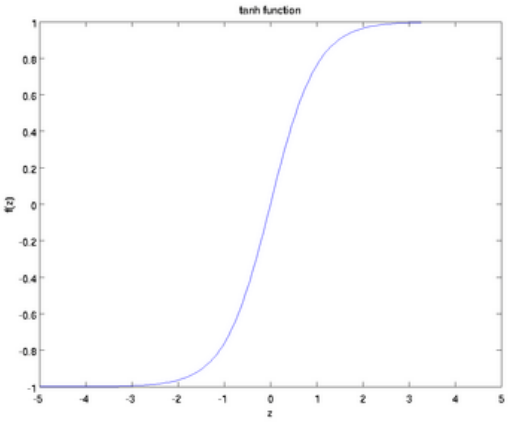

Рисунок 1.6 – Гиперболический тангенс

Перцептрон. Перцептроном называют математическую модель, которая моделирует поведение нейронов человеческого мозга. Эта модель была разработана Ф. Розенблаттом в 1950-х годах и им же была запрограммирована на компьютере.

Классическая модель перцептрона состоит из трех основных элементов: входных сигналов, ассоциативных элементов и реагирующих элементов. Входные сигналы отвечают за получение входных значений. Ассоциативные элементы отвечают за связь между входными сигналами и реагирующими элементами. Реагирующие элементы формируют выход перцептрона. Связь между элементами характеризуется с помощью весовых коэффициентов или весов. В классическом перцептроне, предложенном Ф. Розенблаттом веса между входными сигналами и ассоциативными элементами задаются значениями ±1 и 0. Таким образом задается наличие связи и ее характер. Весовые коэффициенты между ассоциативными элементами и реагирующими элементами могут принимать любые значения. Они характеризуют силу связей. Значения, полученные с выходных сигналов, умножаются на весовые коэффициенты и суммируются. Эта сумма называется взвешенной суммой. Реагирующий элемент имеет пороговое значение, с которым сравнивается

взвешенная сумма. Если взвешенная сумма меньше порогового значения, то реагирующий элемент «неактивен» и возвращает значение 0. В противном случае 1.

Перцептрон является простейшей нейронной сетью с одним скрытым слоем, но котором обычно задают некую функцию активации. В случае отсутствия функции активация перцептрон называют нейроном линейного преобразования.

Обучение перцептрона сводится к изменению матрицы весовых коэффициентов. Выделяют различные виды перцептронов, такие как: однослойный перцептрон, многослойный перцептрон по Розенблатту и многослойный перцептрон по Румельхарту. Последний использует метод обратного распространения ошибки для обучения.

Обучение нейронной сети. В процессе обучения нейронной сети, ей предъявляются входные векторы, которые представляют собой некоторое множество входов, и ожидаемые выходные векторы, которые составляют желаемое множество выходов. Обучение заключается в настройке весовых коэффициентов сети в соответствии с определенной процедурой.

В процессе обучения веса сети постепенно изменяются таким образом, чтобы каждый входной вектор порождал соответствующий выходной вектор. Целью обучения является нахождение подходящего набора весовых коэффициентов, который обеспечивает желаемое отображение входных векторов на выходные.

Существуют два основных типа обучения нейронных сетей: обучение с учителем и обучение без учителя. В обучении с учителем для каждого входного вектора известен ожидаемый выходной вектор, и сеть корректирует свои веса на основе разницы между полученным выходом и желаемым выходом. В обучении без учителя ожидаемые выходные векторы отсутствуют, и сеть самостоятельно находит внутренние закономерности и структуры во входных данных.

Алгоритм обратного распространения ошибки. Метод обратного распространения ошибки используется для обновления весовых коэффициентов в многослойных нейронных сетях. Алгоритм является модификацией метода стохастического градиентного спуска. Цель алгоритма минимизировать разницу между фактическим и ожидаемыми выходными значениями.

В начале обучения значения весовых коэффициентов устанавливаются случайными небольшими числами. Алгоритм обратного распространения ошибки выполняется итеративно и состоит их двух фаз. В первой фазе входные значения подаются на сеть и проходят по ней. Для работы алгоритма важно, чтобы характеристика вход-выход была неубывающей и имела ограниченную

производную. Это необходимо для нахождения градиента. Во второй фазе полученный из сети результат сравнивается с ожидаемым значением. Для качественной оценки этой разницы используется функция потерь. Если значение функции потерь равно нулю, что обучение не происходит. В противном случае эта разница последовательно передается по сети в обратном направлении, обновляя значения весовых коэффициентов по дельта-правилу. Процесс повторяется до достижения заданного критерия остановки или сходимости.

Функции потерь. На второй фазе алгоритма обратного распространения ошибки вычисляется разница между полученным результатом и требуемым или эталонным. Значения разницы зависят от величин, поступающих на вход сети, поэтому важно подобрать качественную оценку этой разницы. Для этой цели существуют различные виды функции потерь, приведем некоторые простейшие из них.

1. Средняя абсолютная ошибка (Mean Absolute Error, MAE). Рассчитывается, как среднее значение абсолютных разностей между фактическим и ожидаемым значениями. Таким образом, формула имеет вид (1.5).

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - y_i'\right|, \tag{1.5}$$

где y – вектор ожидаемых значений;

y' – вектор фактических значений.

2. Среднеквадратичная функция потерь (Mean Squared Error, MSE). Эта функция потерь рассчитывается, как среднее значение квадратов разностей между фактическим и ожидаемыми значениями. Формула имеет вид (1.6).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - y_i'\right)^2, \tag{1.6}$$

где y – вектор ожидаемых значений;

y' – вектор фактических значений.

3. Среднеквадратичное отклонение (Root Mean Squared Error, RMSE). Эта функция потерь является просто квадратным корнем из среднеквадратичной ошибки. Формула имеет вид (1.7).

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - y_i'\right)^2}, \quad (1.7)$$

где y – вектор ожидаемых значений;

y' – вектор фактических значений.

4. Среднеквадратичная логарифмическая функция потерь (Mean Squared Logarithmic Error). Функция представляет собой среднеквадратичную ошибку, вычисленную в логарифмическом масштабе. Имеет формулу (1.8).

$$MSLE = \frac{1}{n}\sum_{i=1}^{n}\left(\log(1 + y_i) - \log(1 + y_i')\right)^2, \quad (1.8)$$

где y – вектор ожидаемых значений;

y' – вектор фактических значений.

В этой формуле к значениям y_i, y_i' добавляется 1 для того, чтобы предотвратить 0 в логарифме. Может использоваться также корень среднеквадратичной логарифмической ошибки (Root Mean Squared Logarithmic Error, RMSLE). Так как логарифм приводит к сжатию значений, то целесообразно использовать среднеквадратичную логарифмическую ошибку или корень из нее тогда, когда фактической и ожидаемое значения отличаются на порядок и больше.

1.4 Архитектуры искусственных нейронных сетей

Архитектура является важной составляющей при разработке любого программного обеспечения. Она еще на стадии разработки определяет основные функции программного обеспечения. Архитектура программного обеспечения проектируется исходя из поставленных задач и особенностей предметной области и программной среды. В нейронных сетях выделяют следующие виды архитектур: глубокие нейронные сети, сверточные нейронные сети, рекуррентные нейронные сети. Каждая из перечисленных архитектур подробно описана в книге [5], приведем краткой описание каждой архитектуры, подчеркнув их особенности и задачи, для решения которых они применяются.

Глубокие нейронные сети. Глубокая нейронная сеть представляет собой классический перцептрон с несколькими скрытыми слоями. Попытки разработки и описания глубоких нейронных сетей были предприняты еще в середине 1960-х годов, однако окончательная архитектура закрепилась лишь в

середине 2000-х годов. В первую очередь это связано с развитием технологий и ростом вычислительной мощности компьютеров, так как глубокие нейронные сети требуют куда большие ресурсы для обучения, чем классический перцептрон. В свою очередь, глубокие нейронные сети позволяют решать куда больший объем задач, в отличии от классического перцептрона. Большое число скрытых слоев позволяет создавать сложные архитектуры нейронных сетей, задавать распределенное представление, что значительно ускоряет обработку большого набора данных. Обучение глубоких нейронных сетей происходит поэтапно. Сначала обучаются отдельные слои нейронной сети, после к ним применяется алгоритм градиентного спуска или его модификации.

Развитие глубоких нейронных сетей привело к появлению различных методов оптимизации обучения нейронных сетей, так как их обучение все еще требует больших ресурсов. К таким методам можно отнести различные модификации стохастического градиентного спуска, такие как RMSProp, Adagrad, Adam и другие. Также появились методы регуляризации. Они позволяют сокращать значения весов для избежания переобучения. Другим методом оптимизации, получившим распространение в середине 2000-х годов, является дропаут.

Суть метода дропаут заключается в том, что на все нейроны в сети, за исключением выходных нейронов, накладывается вероятность, с которой этот нейрон будет выброшен из вычислений, иными словами, его выход будет равен 0. Такой метод позволяет получить усредненный результат обучения сразу нескольких моделей, так как по сути, обучая одну модель, разработчик обучает сразу несколько моделей с меньшим числом нейронов.

Глубокие нейронные сети подходят для решения различных задач глубокого обучения, таких как классификация, распознавание образов, регрессия.

Сверточные нейронные сети. Идея сверточных нейронных сетей состоит в применении механизма свертки при обучении. Суть этого механизма состоит в том, что ко входу нейронной сети применяется небольшое окно фиксированного размера, называемое ядром свертки, на котором нейронная сеть выделяет некоторые признаки. И так ядро свертки покрывает весь вход нейронной сети, после чего формируется выход посредством замены окна на центральный признак. Таким образом, через несколько итераций число признаков для обработки значительно уменьшится.

В глубоком обучении данные часто имеют пространственную архитектуру. Иными словами, конкретный вход может представлять собой числовой вектор. В случае с изображениями каждый пиксель представляет собой вектор с тремя числами, именуемых каналами, которые характеризуют

цвет изображения. Таким образом, на вход нейронной сети подается не одно число, а некий одномерный вектор, именуемый тензором. После применения свертки на выходе получается также некий тензор, но он может иметь другое число компонент. Теперь признаки представляют собой матрицы, именуемые картой признаков.

Операция применения свертки представляет собой линейное преобразование, которое можно описать формулой (1.9).

$$y_{i,j}^{l} = \sum_{-d \le a,b \le d} W_{a,b} x_{i+a,j+b}^{l}, \tag{1.9}$$

где x^{l}– карта признаков в слое под номером l, то есть вход на этом слое;
$y_{i,j}^{l}$ – компонента i,j выхода слоя l;
W – матрица весов размерности $(2d+1) \times (2d+1)$;
В этой формуле применяется двумерная свертка с ядром размера $2d+1$.

После сверточного слоя всегда идет слой функции активации, таким образом итоговый выход из сверточного слоя нейронной сети можно записать формулой (1.10).

$$z_{i,j}^{l} = h\left(y_{i,j}^{l}\right), \tag{1.10}$$

где h – функция активации. В качестве нее часто используется функция активации ReLU, но также может встречаться и гиперболический тангенс.

Сверточные нейронные сети получили широкое распространения в задачах распознавания изображений и компьютерном зрении.

Рекуррентные нейронные сети. Основное отличие рекуррентных нейронных сетей от глубоких нейронных сетей заключается в том, что отдельный нейрон помимо связи с нейронами следующего слоя также имеет связь со своим предыдущим значением.

Необходимость в рекуррентных нейронных сетях возникает тогда, когда данные представляют собой последовательность. Под последовательностью можно понимать временной ряд, речь или любые другие данные, где одна точка зависит от соседних.

Связь нейрона со своими предыдущими значениями позволяет сети запоминать свое предыдущее состояние. С этим свойством связаны дальнейшие модификации, вносимые в архитектуру рекуррентных нейронных сетей. В частности, часто реализуется механизм LSTM (Long Short-Term Memory) или механизм кратковременной памяти. Этот механизм представляет собой отдельную ячейку со своими весами, которая имеет связь с

рекуррентным нейроном и запоминает его предыдущие состояния. Подробно архитектура LSTM механизма описана в книге [5, с. 242].

LSTM является самым распространенным на данных момент механизмом памяти, однако существуют и другие. Например, существует модификация LSTM, называемая GRU (gated recurrent unit). Основное отличие от LSTM в том, что в этом механизме совмещены выходной и забывающие гейты. В последнее время этот механизм памяти замещает LSTM в реальных приложениях.

Описанные механизмы памяти реализует кратковременную память, которая имеет свойство «затухать» с течением времени обучения. Существуют также архитектуры, которые предполагают долгую память. Примером такой архитектуры является SCRN (Structurally Constrained Recurrent Network).

Механизм памяти является главной отличительной особенностью рекуррентных нейронных сетей. С помощью этого механизма стало возможным обработка человеческой речи, генерация текста, изображений и другие сложные задачи, для которых необходим анализ данных и связи между этими данными.

Описанные в данной главе теоретические и алгоритмические аспекты нейронных сетей реализованы на различных программных платформах посредством отдельных нейросетевых пакетов или фреймворков глубокого обучения.

ГЛАВА 2. НЕЙРОННЫЕ СЕТИ В WOLFRAM MATHEMATICA

2.1 Wolfram Mathematica 11

Функциональность нейронных сетей появилась в Wolfram Mathematica после выхода 11 версии в ноябре 2016 года. В ней разработчики Wolfram Research добавили инструменты для создания, настройки и обучения нейронных сетей.

С течением времени разработчики продолжали развивать функционал нейронных сетей в Wolfram Mathematica. Были значительно расширены возможности создания и обучения нейронных моделей, добавлены новые уже обученные модели, оптимизированы алгоритмы обучения.

Следующим этапом в развитии нейронных сетей в Wolfram Mathematica стал запуск Wolfram Neural Net Repository в 2018 году. Wolfram Neural Net Repository – это репозиторий, который содержит предобученные нейронные сети. В нем содержатся самые широко используемы в машинном обучение модели, включая LeNet, CapsNet, VGG-16. Они способны распознавать изображения, человеческую речь, классифицировать объекты на изображениях, прогнозировать на основе временных рядов.

2.2 Основные инструменты для работы с нейронными сетями в Wolfram Mathematica

Дадим краткое описание основным функциям и компонентам, которые нужны для создания и обучения нейронных сетей.

Функция NetChain. Функция используется для создания нейронной сети. Она задает нейронную сеть в виде цепочки слоев или операций.

Функция NetChain задается следующим образом: $NetChain[layer_1, layer_2, \dots, layer_n]$.

Где $layer_1, layer_2, \dots, layer_n$ – это слои или операции, из которых состоит сеть.

Каждый слой в NetChain преобразует данные согласно своему функционалу и передает их дальше по цепочке. Если в параметр *layer* передается целое число, то это задание линейного слоя с указанным числом слоев. Линейный слой преобразует значения, перемножая их на весовые коэффициенты и суммируя. Также есть слои функций активации и сверточные слои. Слои функции активации применяют соответствующую функцию активации, а сверточные слои применяют ядро свертки. После слоя функции активации или сверточного слоя обязательно должен идти линейный слой.

Функция возвращает объект типа NetModel, который представляет собой модель нейронной сети. Пример использования функции NetChain представлен на рисунке 2.1.

Рисунок 2.1 – Пример использования NetChain

В этом примере создается нейронная сеть, состоящая из четырех скрытых слоев: 2 линейных слоя, слой функции активации Ramp, который представляет собой функцию активации ReLU, слой гиперболического тангенса. Входом нейронной сети является массив, а выходом вектор с двумя компонентами. Пример этой нейронной сети взят из документации Wolfram Mathematica [10].

Функция NetGraph. Функция NetGraph используется для задания нейронной сети с несколькими входами. Благодаря таким операциям, как CatenateLayer, входные данные из нескольких входов объединяются и обрабатываются дальше по цепочке. Функция также позволяет представить модель нейронной сети в виде графа, если передать ей в качестве аргумента объект типа NetModel. Представление в виде графа позволяет лучше оценить структуру нейронной сети. Вершинами графа являются слои нейронной сети, а ребра графа – направление данных от одного слоя к другому. В качестве слоя могут также выступать некоторые операции по преобразованию данных.

Синтаксис выглядит функции следующим образом:

NetGraph[{$layer_1$*,* $layer_2$*,... ,* $operation_1$*,* $operation_2$*, ...}, {1 → 2, 2 → 3, ...}]*,

Где $layer_1, layer_2, \dots$ – слои нейронной сети,

$operation_1, operation_2, \dots$ – операции, которые необходимо выполнить в модели, помимо стандартных слоев нейронных сетей. Это могут быть операции для объединения данных или их преобразования,

1 → 2, 2 → 3, ... – определение связи между вершинами графа. Числа указывают порядковые номера вершин графа (слоев или операций), которые связаны между собой. Нумерация начинается с 1.

Как отмечалось ранее, в качестве аргумента можно также передать объект типа NeMode, который возвращает функция NetChain.

Пример использования функции NetGraph представлен на рисунке 2.2.

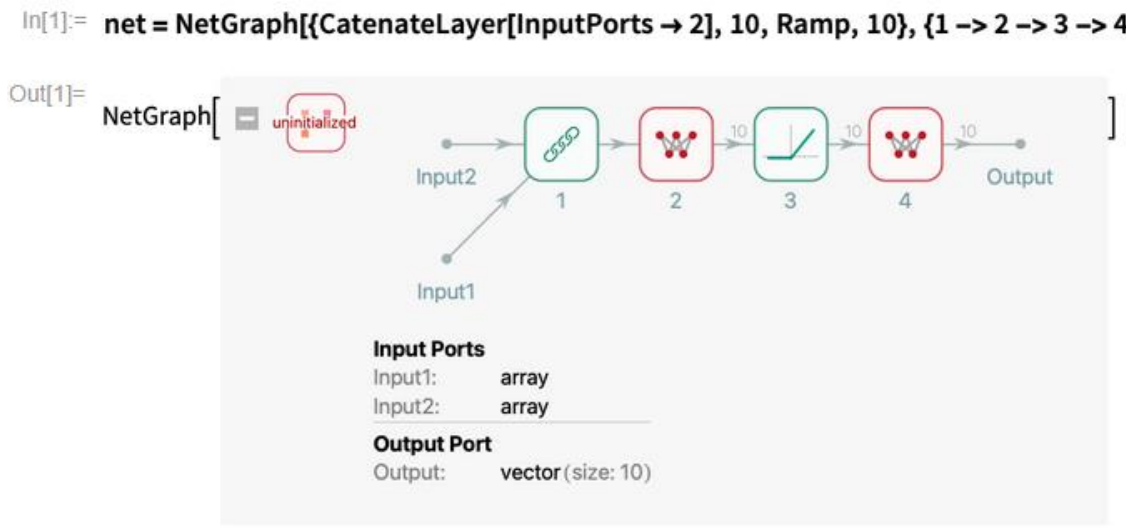

Рисунок 2.2 – Пример использования NetGraph

В этом примере сеть имеет два входных слоя с явно указанными размерностями. Wolfram Mathematica автоматически визуализирует архитектуру, отображая:

1. Все слои сети
2. Соединения между ними
3. Потоки данных

Функция NetInitialize. Функция используется для инициализации параметров нейронной сети перед ее использованием.

Синтаксис функции выглядит следующим образом: *NetInitialize[model]*:

Где *model* – это объект типа NetModel или NetGraph, которые возвращают соответствующие функции. В качестве параметра *model* может также использоваться нейронная сеть, загруженная из Wolfram Neural Net Repository.

Пример использования функции NetInitialize представлен на рисунке 2.3.

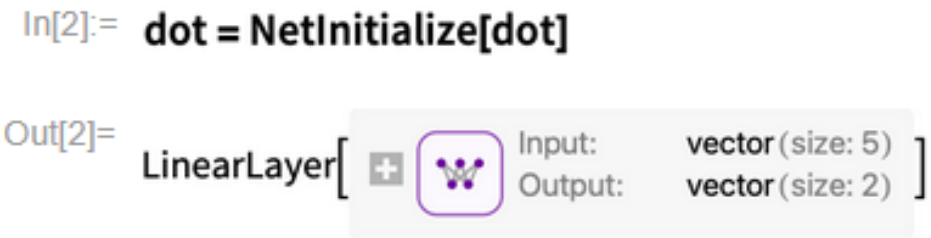

Рисунок 2.3 – Пример использования NetInitialize

Отметим, что NetInitialize только инициализирует начальные параметры нейронной сети. Для обучения нейронной сети используются другие функции.

Функция NetTrain. Функция NetTrain используется для обучения нейронной сети. Она позволяет детально настроить процесс обучения, задав такие параметры, как оптимизатор или метод обучения, функцию потерь, метрику оценки производительности.

Пример синтаксиса: NetTrain [*model, trainingData*],

Где *model* – модель нейронной сети для обучения;

trainingData – набор обучающих данных.

Функция выполняет обучение нейронной сети, применяя выбранный метод обучения (по умолчанию это метод Adam). Помимо обязательных аргументов, функция NetTrain позволяет задать дополнительные параметры для детальной настройки процесса обучения. Опишем некоторые из них.

- LossFunction – задает функцию потерь. Можно выбрать из предопределенных функций потерь или определить собственную.
- Method – задает метод обучения, который определяет, как обновляются параметры модели во время обучения. В Wolfram Mathematica предопределены такие методы обучения, как Adam, RMSProp, SGD и так далее.
- TrainingProgressFunction – этот параметр включает графический интерфейс, содержащий информацию о прогрессе обучения. Информация включает: текущую эпоху, функцию потерь, метрики.
- ValidationSet – набор валидационных данных для оценки нейронной сети после обучения на каком-то батче из набора тренировочных данных. С их помощью можно контролировать переобучение модели.
- TargetDevice – этот параметр определяет то, на каком аппаратном устройстве будет выполняться обучение. По умолчанию это центральный процессор, но можно использовать графический процессор.

Функция также предоставляет графический интерфейс для отслеживания процесса обучения сети. На этом графическом интерфейсе демонстрируется график ошибки сети на эпохе обучения как на тренировочных, так и на тестовых данных. Пример использования функции NetTrain представлен на рисунке 2.4.

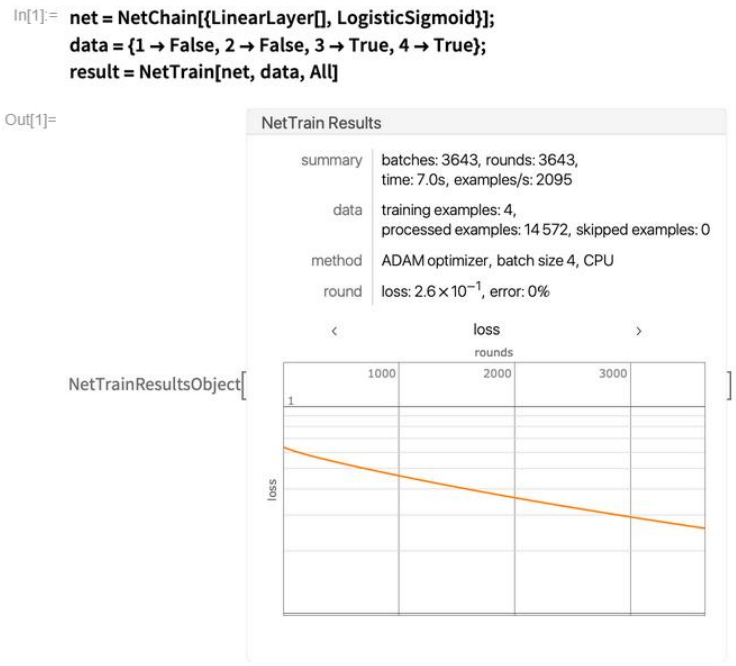

Рисунок 2.4 – Пример использования NetTrain

В этом примере создается простая сеть с одним полносвязным слоем и слоем активации сигмоидной функции и обучается на простейших данных. На графическом интерфейсе представлена информация о данных, которые использовались для обучения сети, об алгоритме обучения и аппаратном устройстве, на мощностях которого происходило обучения, а также информация о функции потерь и ошибке.

После завершения обучения функция NetTrain возвращает обученную модель нейронной сети, которую можно использовать для применения к новым данным и получения результатов.

Функции NetEncoder и NetDecoder. На практике работа с нейронными сетями требует преобразования данных в подходящий формат. В Wolfram Mathematica для этого используются две ключевые функции.

1. NetEncoder - преобразует исходные данные (изображения, текст, аудио) в числовые тензоры, понятные нейронной сети. Как показано на рисунке 2.5, при создании кодировщика указывается:

- Тип данных (Image/Text/Audio)
- Параметры специфичные для формата
- Дополнительные настройки обработки

```
encoder = NetEncoder["type"]
encodedData = encoder[data]
```

Рисунок 2.5 – Пример использования NetEncoder

2. NetDecoder - выполняет обратную операцию, переводя выходы сети обратно в удобочитаемый вид. Например:

- Восстанавливает изображения из числовых представлений
- Преобразует векторные выходы в текст
- Декодирует аудиосигналы

Особенности работы:

- Обе функции используют схожий синтаксис инициализации
- Поддерживают основные типы данных "из коробки"
- Позволяют настраивать параметры преобразования

Главное преимущество - возможность объединять кодировщики/декодировщики непосредственно в архитектуру сети, создавая сквозные потоки обработки данных.

2.3 Методы обучения нейронных сетей в Wolfram Mathematica

Для дальнейшего описания методов обучения нейронных сетей построим упрощенную математическую модель. В основе нашей математической модели лежит модель, описанная в статье [6].

Пусть заданы m точек в n-мерном пространстве $\mathbb{R}^n$: $p_1, p_2, \dots, p_m \in \mathbb{R}^n$. Эти точки будут описывать объекты некоторых классов, координаты точек играют роль признаков. Это множество объектов будет представлять собой обучающую выборку. На ней мы обучаем некоторую модель, которая описывается параметрами: $\omega_1, \omega_2, \dots, \omega_k$.

Набор этих параметров обозначим, как вектор k-мерного пространства $\omega \in \mathbb{R}^k$. Задача обучения нейронной сети состоит в подборе таких весовых коэффициентов, чтобы разница между полученными результатами и эталонными была минимальной. Таким образом, если задать некоторую функцию потерь, то задача обучения нейронной сети сводится к минимизации этой функции, запишем это в виде формулы (2.1).

$$H(\omega, p_1, p_2, \dots, p_m) \to min, \tag{2.1}$$

где ω – параметры модели;

$p_1, p_2, \dots, p_m$ – обучающая выборка.

Функция $H(\omega, p_1, p_2, \dots, p_m)$ зависит от ω и вычисляется на всем множестве объектов $p_1, p_2, \dots, p_m$. Чаще всего функция потерь представляется в виде суммы или среднего значения потери на отдельных элементах выборки, как в формуле (2.2).

$$H(\omega, p_1, p_2, \dots, p_m) = {}^{1}/_{m} \cdot \sum_{i=1}^{m} L(\omega, p_i), \quad (2.2)$$

где $L(\omega, p_i)$ – функция потери на одном элементе p_i из обучающей выборки;

m – число точек.

Метод стохастического градиентного спуска (SGD). Простейшим методом минимизации функции потерь является метод градиентного спуска, однако он обладает рядом недостатков, самым главным из которых является необходимость высчитывать градиент на всех элементах обучающей выборки. В реальных задачах объем данных может превышать десятки тысяч, из-за чего вычисление градиента функции потерь может занимать существенное время и ресурсы компьютера.

Суть метода стохастического градиентного спуска заключается в выборе случайного элемента p_i из всего обучающего набора и вычислении на нем функции потерь по формуле (2.3).

$$H_i(\omega) = L(\omega, p_i), \quad (2.3)$$

где $L(\omega, p_i)$ – функция потери на одном элементе p_i из обучающей выборки.

Затем по формуле (2.4) вычисляется градиент функции потерь по ω:

$$g = \nabla H_i(\omega), \quad (2.4)$$

где $\nabla H_i(\omega)$ – градиент функции потерь.

Вектор g используется на очередном шаге стохастического градиентного спуска для вычисления следующего приближения $\omega^{(k+1)} = \omega^{(k)} - \alpha \cdot g$, где α – шаг обучения.

Метод стохастического градиента чаще используется с модификацией, называемой Mini-Batch. Суть заключается в том, что на каждом шаге используется не один случайный элемент, а какое-то фиксированное множество случайных элементов, называемое мини-пакетом (англ. Mini-Batch). Обозначим этот набор случайных элементов формулой (2.5).

$$B = \{p_{i_1}, p_{i_2}, \dots, p_{i_s}\} \subset \{p_1, p_2, \dots, p_m\}, s \leq m, \quad (2.5)$$

где $p_1, p_2, \dots, p_m$ – обучающая выборка;

s – размерность мини-пакета.

Элементы p_{i_j} можно выбирать либо случайным образом, либо последовательно, начиная с какого-то индекса, в циклическом порядке. Для мини-пакета B из s элементов функция потерь будет вычисляться по формуле (2.6).

$$H_B(\omega) = {}^{1}/_{S} \cdot \sum_{p \in B} L(\omega, p), \tag{2.6}$$

где $L(\omega, p)$ – функция потерь на наборе p.

Градиент этой функции, который выражается формулой (2.7),

$$g = \nabla H_B(\omega), \tag{2.7}$$

где $\nabla H_B(\omega)$ – градиент функции потерь на мини-пакете B.

Он используется на очередном шаге стохастического спуска. После выполнения каждого шага формируется новый мини-пакет либо из s случайно выбранных элементов, либо из следующих s элементов в циклическом порядке после конца предыдущего мини-пакета.

Так как значение вектора g отличается от «реального» значения градиента функции потерь на всем наборе обучающей выборки, то для достижения минимума функции потерь потребуется больше итераций. Однако, за счет того, что вычисления g упрощаются, стохастический градиентный спуск выигрывает по скорости и ресурсоемкости.

На рисунке 2.6 проиллюстрировано поведение метода стохастического градиентного спуска.

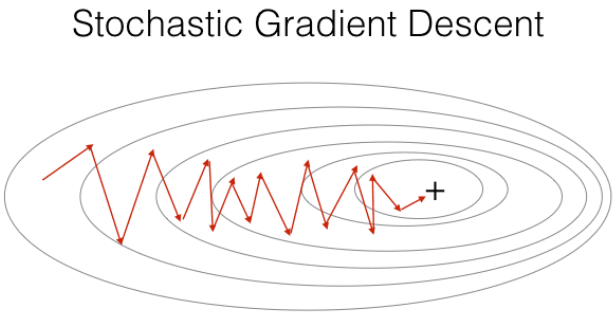

Рисунок 2.6 – Стохастический градиентный спуск

Метод RMSProp (running mean square). Одним из недостатков стохастического градиентного спуска является постоянный шаг обучения. Так как размерность вектора ω не ограничена, то частные производные от функции потерь по каждой компоненте ω могут отличаться, что влечет за собой сильные колебания и излишнее число итераций.

Для решения этой проблемы был предложен ряд алгоритмов, которые пересчитывают шаг обучения, одним из которых является RMSProp. Суть метода в перенормировке шага обучения по формуле (2.8).

$$G^{(t)} = \gamma \cdot G^{(t-1)} + (1 - \gamma) \cdot \left(\nabla L\left(\omega^{(t)}, p_i\right)\right)^2, \gamma \in (0, 1), \quad (2.8)$$

где $G^{(t)}$– норма шага обучения в момент времени t;

γ – какой-то постоянный параметр, задаваемый заранее;

$\nabla L\left(\omega^{(t)}, p_i\right)$ – градиент вектора функции потерь.

Здесь вектор $\nabla L\left(\omega^{(t)}, p_i\right)$ возводиться в квадрат поэлементно. Значение следующего веса при этом рассчитывается по формуле (2.9).

$$\omega^{(t+1)} = \omega^{(t)} - \alpha \cdot \frac{\nabla L\left(\omega^{(t)}, p_i\right)}{\sqrt{G^{(t)}} - \varepsilon}, \quad (2.9)$$

где $\varepsilon > 0$ – это какая-то небольшая константа для исключения деления на ноль;

α – постоянный шаг обучения.

Деление векторов выполняется поэлементно. В результате, каждая компонента вектора градиента $\nabla L\left(\omega^{(t)}, p_i\right)$ будет поделена на пропорциональное значение $\sqrt{G^{(t)}} - \varepsilon$ и, таким образом, несколько нормирована. Благодаря этому выравнивается скорость изменения компонент вектора ω: значения с большими градиентами начинают меняться несколько медленнее, а значения с малыми градиентами – несколько быстрее.

На рисунке 2.7 проиллюстрировано поведение метода RMSProp.

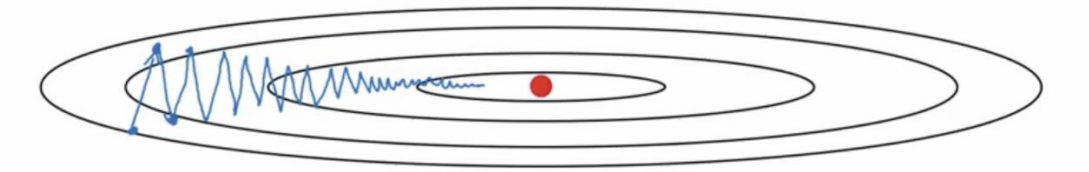

Рисунок 2.7 – метод RMSProp

Метод Adam (adaptive momentum). Еще одним методом, который поддерживает адаптивную скорость изменения компонент вектора ω, является метод Adam. Adam сочетает в себе идею RMSProp и метода импульса. Алгоритм использует подход экспоненциально затухающего бегущего среднего для градиентов $\nabla L\left(\omega^{(t)}, p_i\right)$ и его квадратов. Вводятся оценки (2.10) и (2.11).

$$m^{(t)} = \beta_1 \cdot m^{(t-1)} + (1 - \beta_1) \cdot \nabla L\left(\omega^{(t)}, p_i\right), \quad (2.10)$$

$$v^{(t)} = \beta_2 \cdot v^{(t-1)} + (1 - \beta_2) \cdot \left(\nabla L\left(\omega^{(t)}, p_i\right)\right)^2, \quad (2.11)$$

где $m^{(t)}$ – оценка первого момента (среднее градиентов);

$v^{(t)}$ – оценка второго момента (средняя нецентрированная дисперсия градиентов);

β_1, β_2 – постоянные параметры, задаваемы заранее из диапазона (0, 1).

Однако при этом существует проблема долгого накопления $m^{(t)}$ и $v^{(t)}$ в начале работы алгоритма, особенно в том случае, если значения коэффициентов β_1 и β_2 близки к 1. Для того, чтобы избавиться от этой проблемы без введения дополнительных параметров, оценки первого и второго моментов видоизменяются по формулам (2.12) и (2.13) соответственно.

$$\widehat{m}^{(t)} = \frac{m^{(t)}}{1 - \beta_1^t}, \quad (2.12)$$

$$\hat{v}^{(t)} = \frac{v^{(t)}}{1 - \beta_2^t}, \quad (2.13)$$

где $\widehat{m}^{(t)}$ и $\hat{v}^{(t)}$ – новые оценки первого и второго моментов.

Итоговое выражение для обновления вектора ω записывается по формуле (2.14).

$$\omega^{(t+1)} = \omega^{(t)} - \alpha \cdot \frac{\widehat{m}^{(t)}}{\sqrt{\hat{v}^{(t)}} + \varepsilon}, \quad (2.14)$$

где $\omega^{(t)}$ – весовые коэффициенты в момент t;

α – шаг обучения;

ε – какое-то малое число, чтобы избежать деления на 0.

Стоит отметить, что автор статьи [14], из которой было взято описание работы алгоритмов обучения нейронных сетей, утверждает, что Adam является одним из самых эффективных методов обучения, из-за чего пользуется популярностью в задачах глубокого обучения.

Все вышеописанные алгоритмы реализованы в системе Wolfram Mathematica и задаются в функции NetTrain.

ГЛАВА 3. ФРЕЙМВОРК PYTORCH

3.1 Фреймворки глубокого обучения

Сегодня при разработке нейросетевых моделей трудно обойтись без специализированных фреймворков – программных платформ, которые радикально упростили и ускорили создание и обучение искусственных нейронных сетей. Если раньше разработчикам приходилось вручную реализовывать все математические алгоритмы, то теперь фреймворки берут на себя основную техническую нагрузку, предоставляя:

1. Удобные инструменты для проектирования архитектур нейросетей в объектно-ориентированной парадигме
2. Готовые, оптимизированные реализации ключевых алгоритмов обучения
3. Возможность использовать аппаратные ускорения — в том числе с помощью GPU и технологии CUDA

Рассмотрим три фреймворка, которые оказали наибольшее влияние на развитие технологий глубокого обучения.

TensorFlow. Разработанный компанией Google в 2015 году, TensorFlow вырос из внутренней системы DistBelief в полноценную открытую платформу для машинного обучения. Он основан на вычислительных графах с автоматическим дифференцированием, поддерживает несколько уровней абстракции и может работать в самых разных вычислительных средах – от локальных серверов до облачных решений. Благодаря своей гибкости и масштабируемости, TensorFlow особенно хорошо подходит для промышленного применения, распределённых систем и интеграции с Google Cloud.

Caffe. Этот фреймворк был создан в лаборатории BAIR (Berkeley AI Research) и изначально ориентировался на задачи компьютерного зрения. Его отличают высокая производительность (за счёт реализации на C++) и удобная система описания моделей через protobuf-файлы. На практике Caffe часто используют для быстрого прототипирования и классификации изображений – особенно там, где важна скорость работы и стабильность.

PyTorch. В отличие от TensorFlow, где изначально использовались статические графы вычислений, PyTorch предложил принципиально иной подход – так называемую динамическую модель (define-by-run). Это означает, что вычислительный граф формируется на лету, в момент выполнения кода. Такой подход делает работу с нейросетями гораздо более гибкой и понятной.

PyTorch хорошо интегрируется с остальной Python-экосистемой, что позволяет использовать привычные инструменты вроде NumPy, SciPy и других. Благодаря этому, отладка моделей становится интуитивно понятной, а модификация архитектур – простой и быстрой.

На практике PyTorch особенно удобен:

- в исследовательских проектах, где часто приходится экспериментировать с нестандартными архитектурами;
- в задачах, где важна гибкость и наглядность кода;
- при быстром итеративном прототипировании и тестировании новых идей.

3.2 Основные компоненты PyTorch

PyTorch был выбран в качестве основного фреймворка для выполнения дипломной работы в связи с рядом принципиальных преимуществ, критически важных для исследовательской работы. Ключевым фактором стала глубокая интеграция фреймворка с экосистемой Python, что делает его в каком-то роде похожим на инструменты для глубокого обучения в Wolfram Mathematica. Как и в Wolfram, нет необходимости в сторонних фреймворках для обработки данных, так как PyTorch полностью совместим с NumPy. Другим фактором стала поддержка динамических вычислительных графов (define-by-run paradigm), что обеспечивает беспрецедентную гибкость при проектировании и модификации нейросетевых архитектур. Далее опишем основные компоненты PyTorch.

Тензоры. Тензоры являются основной структурой данных в PyTorch. В отличие от традиционных многомерных массивов, тензоры реализуют три ключевых аспекта:

1. Аппаратно-оптимизированные вычисления. Тензорная библиотека PyTorch обеспечивает прозрачное распределение вычислений между различными вычислительными устройствами. Особенностью реализации является единый интерфейс для CPU и GPU вычислений, где переключение между устройствами осуществляется методом to(). При этом внутренняя организация данных оптимизирована для векторных операций, характерных для задач машинного обучения.

2. Динамическая система градиентов. Механизм autograd в PyTorch реализует концепцию автоматического дифференцирования через построение динамического вычислительного графа. Каждый тензор, созданный с параметром requires_grad=True, становится узлом в этом графе, сохраняя не только данные, но и историю операций. Это позволяет вычислять градиенты

произвольных композиций функций методом обратного распространения ошибки.

3. Типовая система и организация памяти. Тензоры PyTorch поддерживают:

- Строгую типизацию (32/64-битные числа с плавающей точкой)
- Различные схемы хранения (contiguous memory layout)
- Эффективные view-операции без копирования данных

Практический опыт работы с тензорами выявляет несколько важных аспектов:

- Неявное приведение типов может вызывать неожиданные потери точности
- Операции с невыровненной памятью снижают производительность
- Градиенты по умолчанию аккумулируются, что требует ручного обнуления

Сравнивая с NumPy-массивами, следует отметить, что тензоры PyTorch:

1. Требуют явного указания вычислительного устройства
2. Поддерживают распределенные вычисления
3. Интегрированы с системой автоматического дифференцирования
4. Оптимизированы для пакетной обработки данных

Datasets и DataLoaders. Система обработки данных в PyTorch построена вокруг двух ключевых абстракций:

1. Класс Dataset. Абстрактный базовый класс, требующий реализации трех основных методов:

- __init__ - инициализация и загрузка данных
- __getitem__ - доступ к конкретному образцу по индексу
- __len__ - получение общего количества образцов

На практике разработчики часто используют:

- Встроенные датасеты (FashionMNIST, CIFAR-10)
- Кастомные реализации для специфичных данных
- Композиции нескольких датасетов

2. Класс DataLoader. Обеспечивает эффективную загрузку данных с поддержкой:

- Пакетной обработки (параметр batch_size)
- Перемешивания (shuffle)
- Многопоточной загрузки (num_workers)
- Предварительной выборки (prefetch_factor)

Критические аспекты реализации:

- Для больших датасетов рекомендуется использовать IterableDataset
- Оптимальное значение num_workers зависит от CPU и размера данных
- Pin_memory=True ускоряет передачу данных на GPU

Библиотека torch.nn. Библиотека torch.nn реализует модульный подход к построению нейронных сетей, сочетая гибкость исследовательского инструмента с эффективностью промышленного фреймворка. Ее архитектура основана на нескольких фундаментальных принципах:

1. Иерархия компонентов. Базовый класс nn.Module определяет:
   - Стандартный интерфейс для всех моделей
   - Механизм автоматического распространения градиентов
   - Систему управления параметрами (parameters(), buffers())
2. Слои нейронных сетей. Библиотека предлагает оптимизированные реализации:
   - Линейных преобразований (nn.Linear)
   - Сверточных операций (nn.Conv1d/2d/3d)
   - Рециркулярных структур (nn.LSTM, nn.GRU)
   - Нормализации (nn.BatchNorm, nn.LayerNorm)
3. Функции потерь. Реализации включают:
   - Кросс-энтропию с обработкой edge-cases
   - Регрессионные метрики с поддержкой маскирования
   - Специализированные функции для задач сегментации

3.3 Основные инструменты для построения простейшей нейронной сети в PyTorch

Опишем процесс построения простейшей нейронной сети. Построение нейронной сети начинается с создания класса нейронной сети, который наследуется от класса nn.Module. В этом классе в методе инициализации создается секвенция, в которой задается то, как будет выглядеть нейронная сеть. Секвенция является объектом класса nn.Sequential. В конструкторе перечисляются слои, которые будут использоваться в нейронной сети.

На рисунке 3.1 представлен код класса DenoisingNet, который будет основой для модели. Класс nn.Linear представляет собой слой линейного преобразования, а класс nn.Tanh – слой с функцией активации гиперболический тангенс. Конструктор классов принимает в качестве аргументов два целых числа – вход и выход соответственно.

```
class DenoisingNet(nn.Module):
    def __init__(self):
        super(DenoisingNet, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(1, 60),
            nn.Tanh(),
            nn.Linear(60, 60),
            nn.Tanh(),
            nn.Linear(60, 60),
            nn.Tanh(),
            nn.Linear(60, 1)
        )

    def forward(self, x):
        return self.net(x)
```

Рисунок 3.1 – Пример построения конфигурации

Далее для начала обучения модели необходимо подготовить данные и, создать и настроить саму модель. На рисунке 3.2 представлен код обучения модели.

```
num_epochs = 2000
for epoch in range(num_epochs):
    optimizer.zero_grad()
    outputs = model(x_train)
    loss = criterion(outputs, y_train)
    loss.backward()
    optimizer.step()

    if (epoch + 1) % 200 == 0:
        print(f"Epoch [{epoch + 1}/{num_epochs}], Loss: {loss.item():.4f}")
```

Рисунок 3.2 – Пример обучения нейронной сети

В представленном коде optimizer – алгоритм обучения нейронной сети. Из разновидности будут описаны далее, loss – функция потерь. Опишем процесс обучения:

- *optimizer.zero_grad()* – сброс градиентов;
- *outputs = model(x_train)* – получение данных из нейронной сети;
- *loss = criterion(outputs, y_train)* – вычисление ошибки;
- *loss.backward()* – вычисление градиентов функции потерь;
- *optimizer.step()* – обновление весов сети с учетом вычисленных градиентов;

Для получения предсказания модели используется метод detach(), который принимает на вход данные, на основе которых нужно сделать предсказание.

ГЛАВА 4. СРАВНЕНИЕ МЕТОДОВ ОБУЧЕНИЯ НЕЙРОННЫХ СЕТЕЙ НА ПРИМЕРЕ ЗАДАЧИ АППРОКСИМАЦИИ ФУНКЦИИ В WOLFRAM MATHEMATICA И В PYTORCH

4.1 Постановка задачи аппроксимации функции

Задачей дипломной работы является оценка эффективности методов обучения нейронной сети Adam и RMSProp, а также сравнение полученных результатов с результатами в Wolfram Mathematica. В качестве эталонных данных будем использовать функцию $f(x)$, определенную по следующей формуле (4.1). График этой функции представлен на рисунке 4.1.

$$f(x) = \frac{\cos x^2}{e^{(0.2 \cdot x^2)}} + 0.2 \cdot x, \tag{4.1}$$

где x – вход сети.

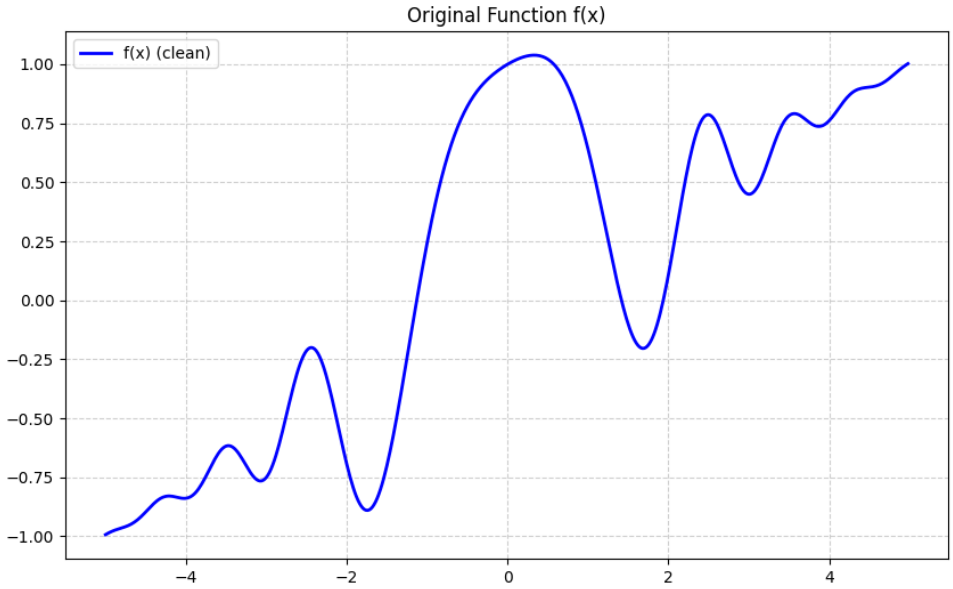

Рисунок 4.1 – График функции $f(x)$

Для получения данных, на основе которых будем обучать модель, добавим «искажения» на функцию $f(x)$. Для этого воспользуемся встроенными функциями Wolfram Mathematica и библиотеки Numpy для Python. Будем искажать функцию, используя нормальное распределение из теории вероятностей. На рисунке 4.2 представлен код функции генерации шума на Python.

```python
# Добавление искажений (шумов)
def epsilon(x, n=250, a=5, w=1/250):
    c = np.random.normal(0, 1, n)  # Коэффициенты случайного шума
    return w * sum(c[i] * np.cos(np.pi * i * (x - a) / (2 * a)) for i in range(n))

def f2(x):
    return f1(x) + epsilon(x)
```

Рисунок 4.2 – Функция для получения искажений

Первая функция epsilon(x) получает в качестве параметров массив аргументов функции x, число значений в массиве для генерации случайных коэффициентов n, который у нас равен 250. Параметр a нужен для функции сглаживания шумов, чтобы они не так сильно искажали исходную функцию. Параметр w является весом шумов и полагаем его равным 0.004. В итоге формула функции для генерации искажений имеет вид (4.2).

$$\varepsilon = w * \sum_{i=0}^{n-1} NormalDistribution(0,1) * \cos\left(\pi i \frac{x-a}{2a}\right), \qquad (4.2)$$

где w – вес искажений;

a – параметр сглаживания;

Под $NormalDistribution(0,1)$ в формуле понимаются коэффициенты, полученные из нормального распределения с математическим ожиданием 0 и стандартным отклонением 1. В Python для этого используется функция из библиотеки Numpy random.normal(0, 1). Таким образом, значения, которые будут использоваться для обучения модели получаются по формуле (4.3). График функции с шумом представлен на рисунке 4.3.

$$f_{noise}(x) = f(x) + \varepsilon, \qquad (4.3)$$

где ε – искажения.

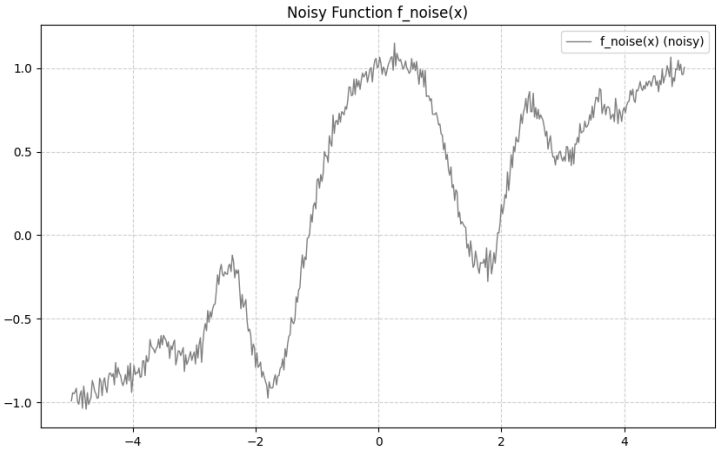

Рисунок 4.3 – график функции $f_{noise}(x)$

Таким образом, задача сводится к получению из графика функции $f_{noise}(x)$ график функции $f(x)$.

4.2 Создание нейронной сети

Для решения задачи была взята нейронная сеть из примеров Wolfram Mathematica [3]. Выбранная сеть состоит из 5 слоев: 3 слоя линейного преобразования и 2 слоя активации. В качестве функции активации выбран гиперболический тангенс. Архитектура нейронной сети представлена на рисунке 4.4.

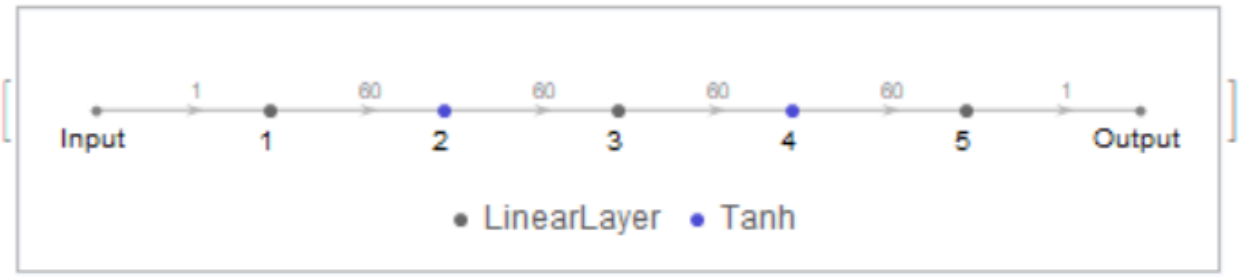

Рисунок 4.4 – Архитектура нейронной сети

На рисунке 4.4 60 – это значение переменной vector_length, которая обозначает число нейронов на скрытых слоях. Это значение демонстрируется на рисунке в качестве примера. В дальнейшем в работе значения этой переменной будут корректироваться для поиска значения, которое выдаст минимальное значение функции потерь.

4.3 Обучение нейронной сети с помощью метода RMSProp

Рассмотрим простейшие варианты обработки данных. Нашим результатом является аппроксимирующая гладкая функция. Для объективности оценки возьмем значения переменной vector_length в диапазоне от 1 до 100. Получим для каждого значения vector_length значение функции потерь для модели. В качестве функции потерь возьмем среднеквадратичную ошибку, которая вычисляет среднеквадратичную ошибку между результатом модели и целевым значением по формуле (4.4).

$$loss = \frac{1}{N}\sum_{i=1}^{N}(y_{pred,i} - y_{true,i})^2, \qquad (4.4)$$

где y_{pred} – вектор предсказанных значений размерности N;
y_{true} – вектор истинных значений размерности N.

Диапазон значений vect_length выбран от 1 до 100 ввиду того, что по мере увеличения числа нейронов в модели время обучения увеличивается. На рисунке 4.5 представлен график зависимости времени обучения модели от числа нейронов в ней. По линии тренда видно, что время увеличивается по мере увеличения числа нейронов.

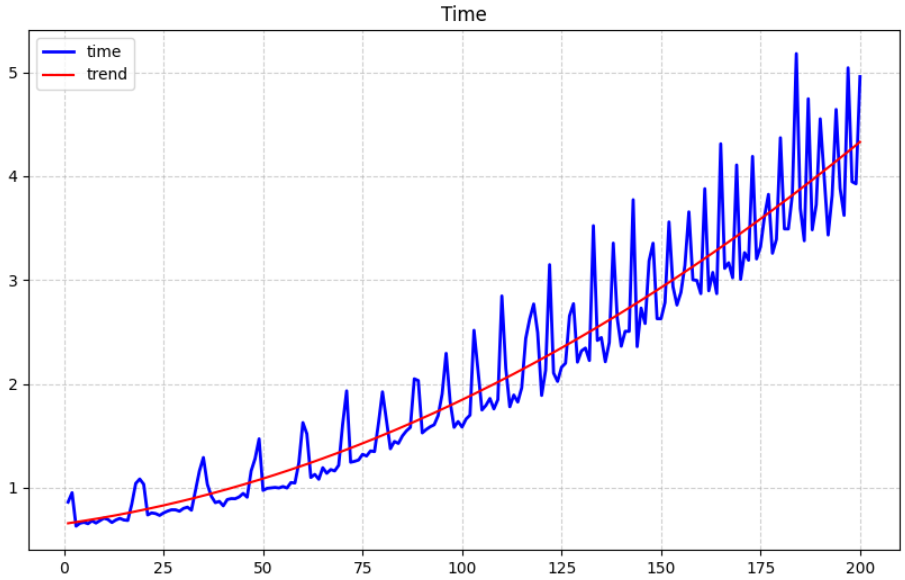

Рисунок 4.5 – График зависимости времени от числа нейронов

При этом, значения функции потерь с ростом числа нейронов уменьшаются незначительно, как представлено на рисунке 4.6. Поэтому подбирать оптимальное число векторов в сети будем в диапазоне до 100.

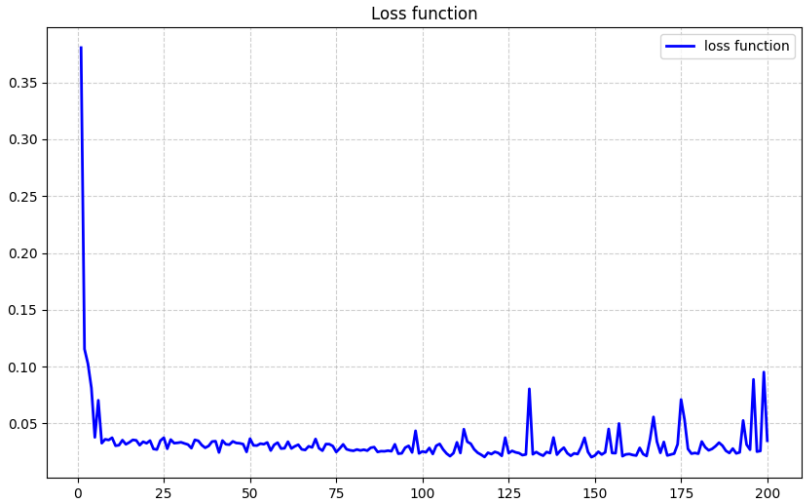

Рисунок 4.6 – График зависимости значений функции потерь от числа нейронов

На рисунке 4.7 представлен график, иллюстрирующий зависимость ошибки от числа нейронов в сети. Красным кружком выделен минимум, который в данном примере равен 87. Значение функции потерь при таком числе нейронов в сети составила 0.0025. Эта модель будет использоваться для сглаживания.

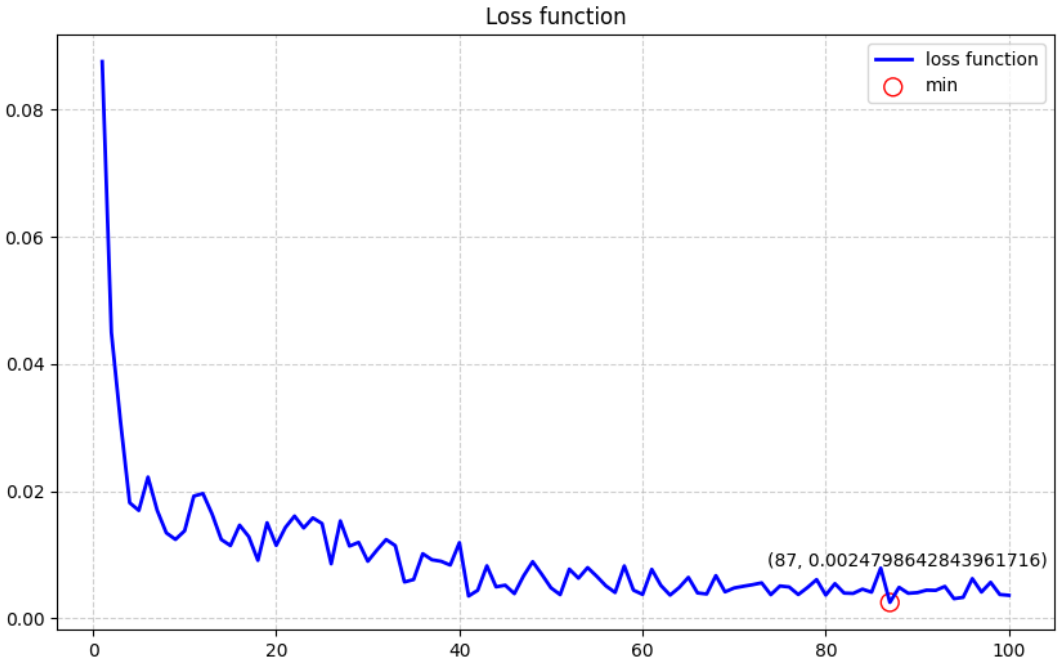

Рисунок 4.7 – График значений функции потерь с методом RMSProp

На рисунке 4.8 изображен результат обучения нейронной сети с методом обучения RMSProp. Синим цветом выделена эталонная функция, серым–исходные данные. Результаты обучения продемонстрированы красным цветом.

Как отмечалось ранее, в разделе 3.3, метод RMSProp основан на адаптивной скорости обучения, который получается посредством сглаживания градиента. Как видно на рисунке 4.8, полученная функция получилась слишком сглаженной на концах отрезка. Значение функции потерь по формуле (4.4) при таком результате получилось равным 0.9789, что близко к 1.

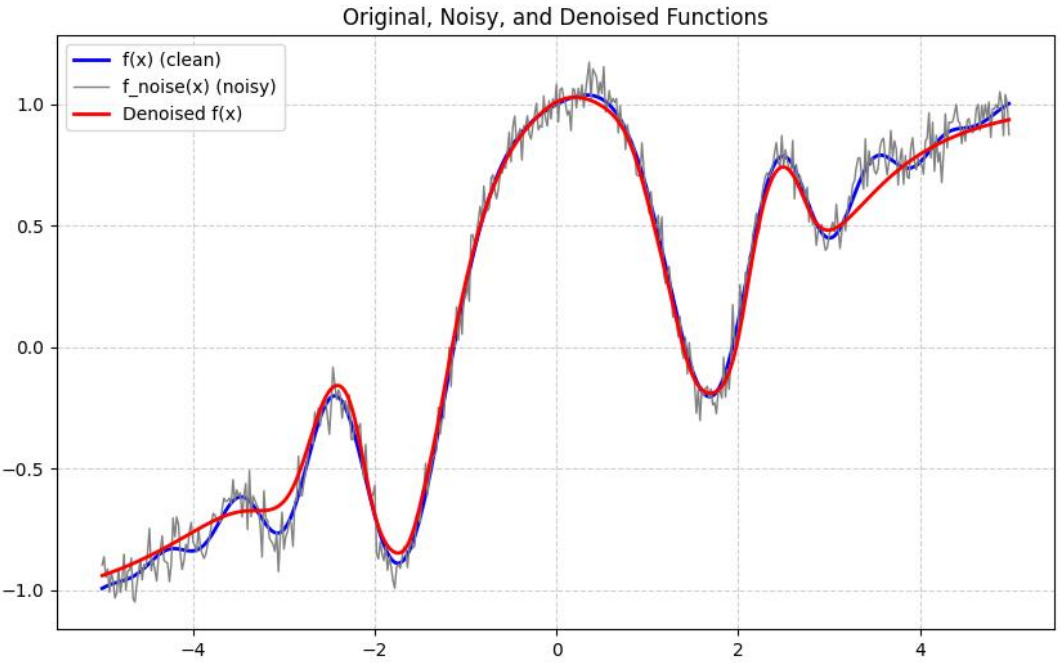

Рисунок 4.8 – Результаты расчетов методом RMSProp в PyTorch

В то же время, в Wolfram Mathematica после обучения нейронной сети был получен следующий результат (см. рисунок 4.9). Значение функции потерь после сглаживания составило 0.00046, что значительно меньше, чем на Python. Также стоит отметить, что данный результат был получен при числе нейронов равным 10, что значительно ускоряет обучение сети, в сравнении с Python.

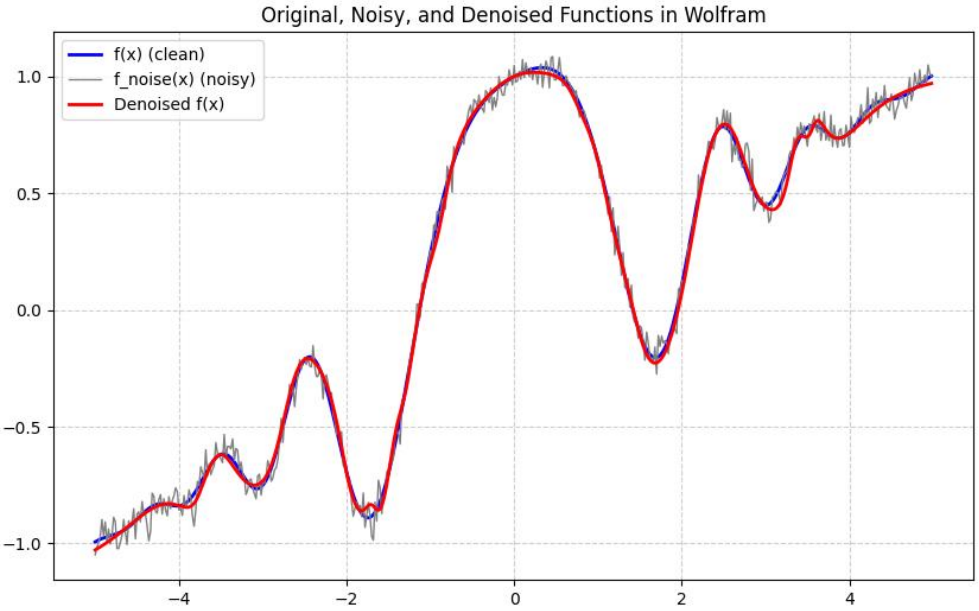

Рисунок 4.9 – Результаты расчетов методом RMSProp в Wolfram Mathematica

4.4 Обучение нейронной сети с помощью метода Adam

Рассмотрим теперь результаты расчетов с применением метода Adam. Как было описано в пункте 3.3, метод Adam основан на экспоненциальном затухающем бегущем среднем градиентов и его квадратов. На рисунке 4.10 представлены результаты подбора оптимального числа векторов на основе значения ошибки во время обучения модели. Минимальное значение достигнута при vect_length равным 85.

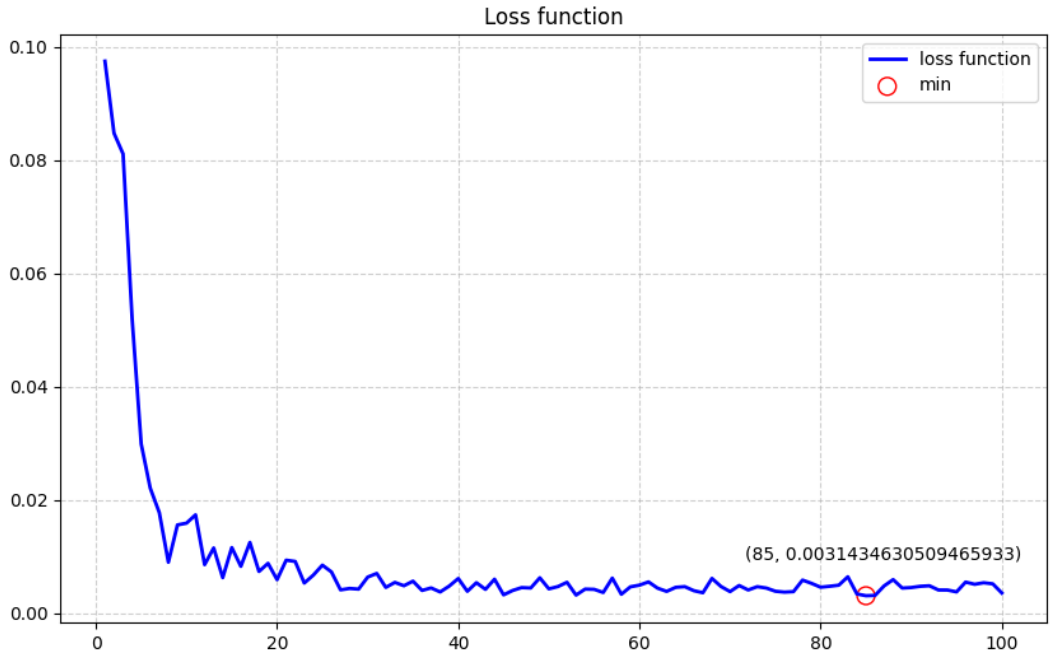

Рисунок 4.10 – График значений функции потерь с методом Adam

Результаты расчетов в Python продемонстрированы на рисунке 4.11. Значение функции потерь после сглаживания получилось равным 1.0051, что хуже, чем при применении метода RMSProp. Также на рисунке видно, что на концах отрезка функция получилась слишком сглаженной.

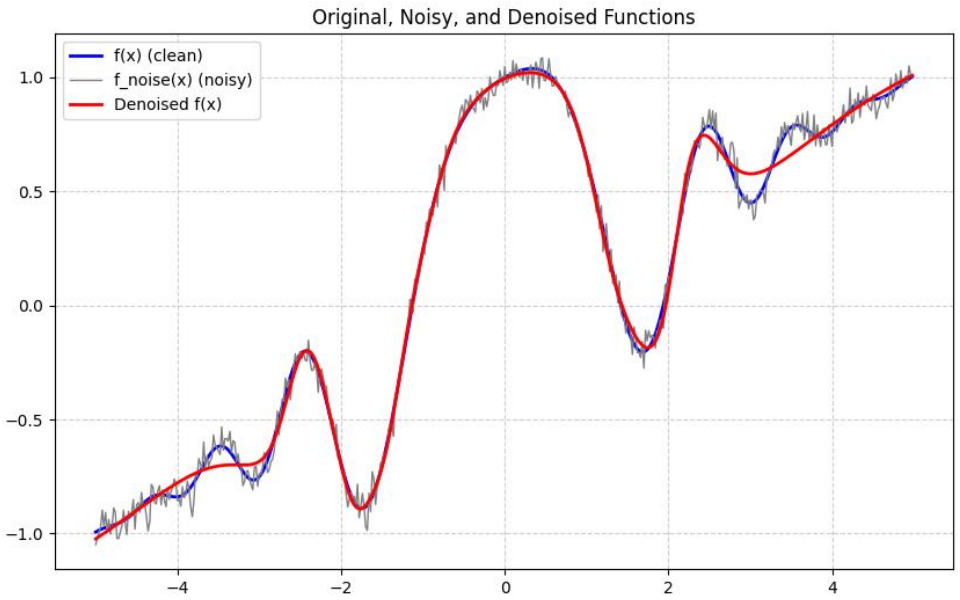

Рисунок 4.11 – Результаты расчетов методом Adam в PyTorch

Теперь рассмотрим результаты, полученные в Wolfram Mathematica. Для настройки метода обучения в параметр функции NetTrain передается строка «ADAM». Результат представлен на рисунке 4.12.

Значение функции потерь после сглаживания получилось равным 0.000766, что значительно ниже, чем полученный результат в Python. Также это значение больше, чем значение, полученное при расчетах с помощью метода RMSProp. Такие результаты могут говорить о том, что метод Adam для задачи сглаживания функции подходит хуже, чем метод RMSProp.

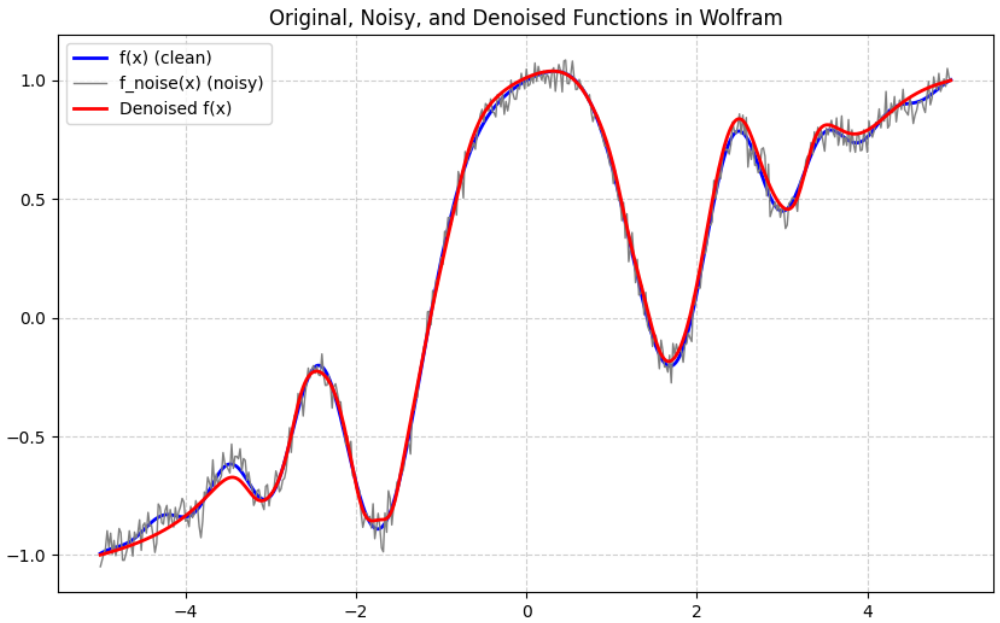

Рисунок 4.12 – Результаты расчетов методом Adam в Wolfram Mathematica

В таблице 4.1 представлены результаты исследования.

Таблица 4.1 – Сравнение результатов в задаче аппроксимации функции

Платформа	RMSProp	Adam
Python PyTorch	0.9789	1.0051
Wolfram Mathematica	0.00046	0.000766

Таким образом полученные результаты говорят о том, что для решения поставленной задачи система компьютерной алгебры Wolfram Mathematica подходит лучше, чем фреймворк PyTorch. Такие результаты вероятнее всего получены ввиду разницы реализации конкретных алгоритмов и моделей в системе Wolfram Mathematica и PyTorch. Фреймворк PyTorch в первую очередь предназначен для решения типичных задачи машинного обучения, таких как классификация, регрессия и т. д. В то время, как система Wolfram Mathematica рассчитана в первую очередь для работы с математическими моделями, выражениями, поэтому лучше справляется с задачей, рассмотренной в данной работе.

ГЛАВА 5. СРАВНЕНИЕ ЭФФЕКТИВНОСТИ НЕЙРОННЫХ СЕТЕЙ В ЗАДАЧЕ ПРОГНОЗИРОВАНИЯ В WOLFRAM MATHEMATICA И В PYTORCH

5.1 Постановка задачи прогнозирования

Во время прохождения преддипломной практики на унитарном предприятии «Белтехосмотр», учитывая тему дипломной работы, перед мной была поставлена задача прогнозирования доли автомобилей, не прошедших линию технического осмотра. Для выполнения поставленной задачи были предоставленные данные, содержащие информацию о числе автомобилей, которые проходили линию технического осмотра, числе автомобилей, которые эту линию не прошли, а также о числе автомобилей, не старше трех лет. Пример данных представлен в таблице 5.1.

Таблица 5.1 – Данные о числе автомобилей, прошедших и не прошедших линию технического осмотра

Год	Месяц	Прошел линию?	Число автомобилей
2017	1	0	13790
2017	1	1	69374

Данные о числе автомобилей, проходивших линию и не старше 3 лет были предоставлены в схожем формате.

Для решения задачи прогнозирования удобно использовать нейронные сети, но для начала необходимо определить набор признаков, которые влияют на вероятность прохождения линии технического осмотра.

При анализе данных, предоставленных предприятием, было выявлено, что доля автомобилей, не прошедших линию технического осмотра, меньше среди автомобилей, возраст которых не старше трех лет. На рисунке 5.1 представлена гистограмма долей автомобилей не старше трех лет, не прошедших линию технического осмотра, по месяцам, начиная с января 2017. На гистограмме видно, что за исключением нескольких месяцев, в основном доля не превышает 0.08. Также было вычислено среднее значение доли автомобилей не старше трех лет, не прошедших линию технического осмотра, которое составило 0.064. Для сравнения, среднее значение доли всех автомобилей, не прошедших линию технического осмотра, начиная с января 2017 года, составило 0.154. На основании этого был сделан вывод о том, что

возраст автомобиля прямо влияет на вероятность прохождения линии технического осмотра.

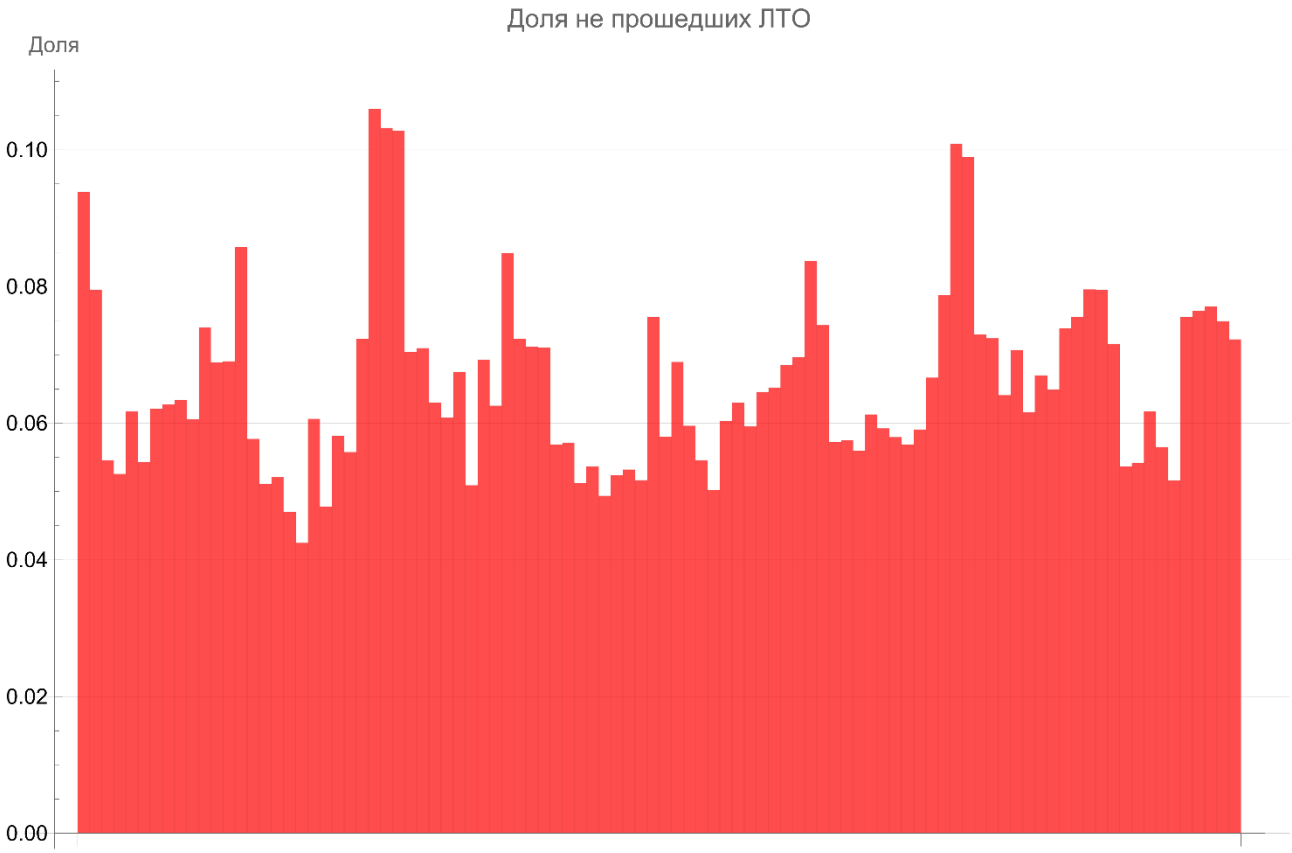

Рисунок 5.1 – Гистограмма доли автомобилей не старше трех лет, не прошедших линию технического осмотра

Таким образом, дальнейшая задача сводится к подбору признаков, которые влияют на число новых автомобилей, заявленных на прохождении линии технического осмотра.

Одним из признаков, который будем использовать для прогнозирования, будет число автомобилей не старше трех лет, которые заявлены на прохождение линии технического осмотра. Другим признаком, который влияет на число таких автомобилей, будет объем рынка новых автомобилей. Другими словами, число автомобилей, которые были проданы в автомобильных салонах за месяц, начиная с января 2017 года. Данные о продажах автомобилей были взяты с электронного ресурса автомобильной ассоциацией БАА [6]. Третьим признаком, который будет связывать эти два признака, будет средняя заработная плата населения. Значения средней заработной платы по месяцам была взята с сайта Национального статистического комитета Республики Беларусь [4]. Достаточно часто новые автомобили из салона берутся в кредит, поэтому в качестве четвертого признака будет выступать средняя кредитная ставка юридическим лицам на

срок больше 1 года. Данные о средней кредитной ставке были взяты с сайта Национального банка Республики Беларусь [2].

Таким образом, было выделено 4 признака, которые будем использовать для обучения нейронной сети. Пример значений признаков представлен в таблице 5.2.

Таблица 5.2 – Пример данных для обучения нейронной сети

Объем рынка новых автомобилей	Автомобили не старше 3 лет, заявленные на прохождение ЛТО	Средняя заработная плата населения	Средняя кредитная ставка для юридических лиц на срок больше 1 года	Доля автомобилей, не прошедших ЛТО
1652	938	720.70	17.42	0.1658

Объем данных для обучения модели составляет 89. Объем данных для тестирования модели составляет 10.

Задача состоит в следующем: подобрать модель; обучить модель; протестировать, сравнивая результаты прогнозирования с тестовыми данными; изобразить результаты прогнозирования на графиках.

5.2 Выбор нейронной сети

Выбор подходящей архитектуры нейронной сети, а также подбор оптимального числа нейронов в скрытых слоях является важной задачей в построении модели, которая может потребовать много времени и большого объема данных для тестирования. Документация Wolfram Mathematica содержит подробное описание использования нейронных сетей в задачах прогнозирования. Для решения нашей задачи мы будем использовать нейронную сеть, представленную в примере [13].

На рисунке 5.2 представлена архитектура нейронной сети из примера, которую будем использовать для решения задачи. Данная нейронная сеть имеет 7 слоев. LinearLayer (Линейный слой) просто вычисляет взвешенные значения и передает их на выход. BatchNormalizationLayer отвечает за нормализацию входных данных с целью ускорения обучения нейронной сети. Ramp это слой с функцией активации ReLU. Входными данными у нас является вектор из четырех признаков, выходными – скаляр, представляющий долю автомобилей, не прошедших линию технического осмотра.

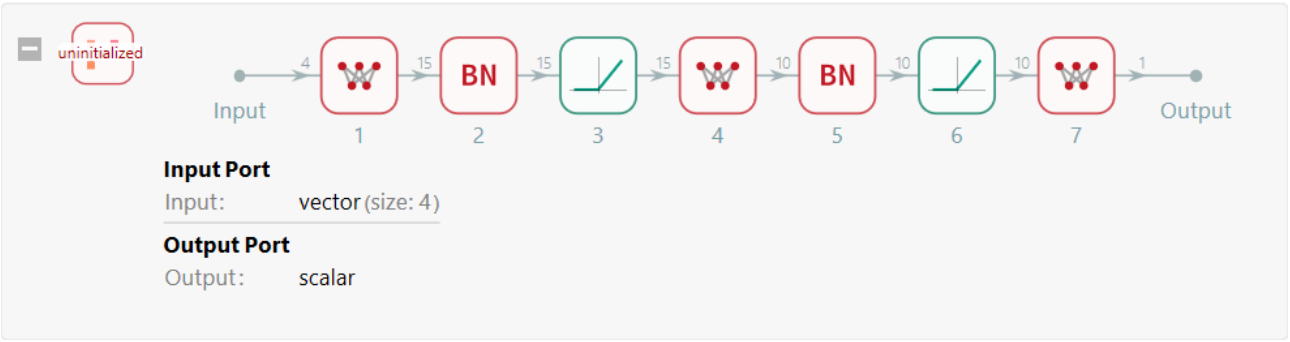

Рисунок 5.2 – Архитектура нейронной сети

Как видно на рисунке 5.2, 3 первых скрытых слоя нейронной сети имеют 15 нейронов, 3 следующие слоя нейронной сети имеют 10 нейронов. В примере [13] данная нейронная сеть обучается на датасете Boston Homes, который представляет собой набор из 14 признаков, описывающих факторы, влияющие на цену жилья в каком-либо районе города Бостон. Задача, описанная в примере, схожа с нашей, поэтому нейронную сеть, представленную в примере, можно использовать для решения нашей задачи.

5.3 Обучение нейронной сети в Wolfram Mathematica

Выбрав архитектуру нейронной сети, можно приступать к обучению модели. Перед обучением необходимо импортировать данные и преобразовать их таким образом, чтобы их можно было использовать в функциях Wolfram Mathematica. Наши данные представлены в файлах типа .csv и представляют собой строки значений, разделенных точкой с запятой. Для импорта данных воспользуемся функцией Import. Функция импортирует данные в формате, представленном в формуле (5.1), num это какое-то число, в нашем случае значение признака. Стоит отметить, что по типу все данные после импорта являются строкой.

$$\{num1; num2; num3; num4; num5\}, \quad (5.1)$$

где $num1; num2; num3; num4; num5$ – числа, которые поступают в виде строк.

Нам необходимо преобразовать эти данные в словарь, где ключом будет выступать массив из 4 чисел-признаков, а значением будет пятое число – доля автомобилей, не прошедших линию технического осмотра. Код преобразования данных в нужную структуру представлен на рисунке 5.3.

```
processString[str_] := Module[{nums}, nums = ToExpression /@ StringSplit[str, ";"];
   {nums[[1]], nums[[2]], nums[[3]], nums[[4]]} → nums[[5]];
result = processString /@ Flatten[data]
```

Рисунок 5.3 – Код преобразования данных

В этом коде мы создаем функцию processString, которая принимает на вход строку. Функция Module объявляет локальную переменную nums, которая является массивом из 5 чисел. Функция ToExpression переводит строку в число, а функция StringSplit делит строку по разделителю, в нашем случае этот разделитель точка с запятой. Структура преобразованных данных представлена в формуле (5.2).

$$\{feature1, feature2, feature3, feature4\} \rightarrow result, \qquad (5.2)$$

где *feature(i)* это *i*-й признак;

result – выходное значение, в нашем случае доля автомобилей, не прошедших линию.

Далее мы делим данные на обучающие и тестовые в пропорции 9:1.

После подготовки данных можно приступать к обучению модели. Инициализируем нейронную сеть с помощью функции NetChain. Пример кода представлен на рисунке 5.4.

```
net = NetChain[
  {LinearLayer[15],
   BatchNormalizationLayer[],
   ElementwiseLayer[Ramp],
   LinearLayer[10],
   BatchNormalizationLayer[],
   ElementwiseLayer[Ramp],
   LinearLayer[1]},
  "Input" → inputSize, "Output" → "Scalar"
 ]
```

Рисунок 5.4 – Код инициализации нейронной сети в Wolfram Mathematica

Обучение нейронной сети будем проводить двумя методами: Adam и RMSProp. Сперва обучим с помощью метода RMSProp. Время обучения составило 35 секунд, значение функции потерь 7.94×10^{-7}. Спрогнозируем

долю автомобилей, не прошедших линию, на основе тестовых данных и сравним полученные значения с валидными. Результат представлен на рисунке 5.5. Значение среднеквадратичной ошибки составило 0.00131.

Рисунок 5.5 – Результат прогнозирования в Wolfram Mathematica при обучении методом RMSProp

На графике красным цветом обозначены реальная доля автомобилей, не прошедших линию технического осмотра, а синей пунктирной линией обозначены спрогнозированные значения.

Обучение нейронной сети с помощью метода Adam заняло 39 секунд. Значение функции потерь во время обучения составило 2.51×10^{-6}. После прогнозирования был получен график, представленный на рисунке 5.6. Среднеквадратичная ошибка составила 0.0103. В таблице 5.3 представлено сравнение результатов обучения. Столбец Loss – значение функции потерь во время обучения. Столбец MSE – значение среднеквадратичной ошибки на тестовых данных.

Таблица 5.3 – Сравнения результатов обучения нейронной сети в системе Wolfram Mathematica

Метод обучения	Loss	MSE
RMSProp	7.94×10^{-7}	0.00131
Adam	2.51×10^{-6}	0.0103

Как видно на рисунке 5.6, результат прогнозирования при обучении методом Adam значительно хуже, чем результат прогнозирования при

обучении методом RMSProp. Это выражается и в значениях среднеквадратичных ошибок. Ошибка при обучении методом Adam в 10 раз больше на тестовых данных.

Рисунок 5.6 – Результат прогнозирования в Wolfram Mathematica при обучении методом Adam

Таким образом, в системе Wolfram Mathematica при решении поставленной задачи метод RMSProp дал значительно лучший результат, чем метод Adam.

5.4 Обучение нейронной сети в PyTorch

Для подготовки данных к обучению нейронной сети с использованием фреймворка PyTorch необходимо импортировать дополнительную библиотеку pandas. После импорта данных и разделения его на признаки и значения необходимо преобразовать полученные массивы в pytorch-тензоры, которые используются в PyTorch моделях. Код подготовки данных представлен на рисунке 5.7.

```
data = pd.read_csv('train_dataset2.csv', sep=';', header=0)

X = data.iloc[:, :4]
y = data.iloc[:, 4]

X_train_tensor = torch.FloatTensor(X.values)
y_train_tensor = torch.FloatTensor(y.values).view(-1, 1)

train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
```

Рисунок 5.7 – Код подготовки данных в Python

TensorDataset это класс-обертка, который работает как словарь и каждому набору признаков ставит соответствующее значение. DataLoader класс, который настраивает размер выборки, смешивать ли данные и загружать ли их параллельно.

Построение модели происходит через класс Sequential, который подобно функции NetChain из Wolfram Mathematica создает последовательность из слоев. Код создания нейронной сети представлен на рисунке 5.8. Приведенная нейронная сеть аналогична той, которую мы использовали в Wolfram Mathematica.

```
class NeuralNetwork(nn.Module):
    def __init__(self, input_size):
        super(NeuralNetwork, self).__init__()

        self.net = nn.Sequential(
            nn.Linear(input_size, 15),
            nn.BatchNorm1d(15),
            nn.ReLU(),

            nn.Linear(15, 10),
            nn.BatchNorm1d(10),
            nn.ReLU(),

            nn.Linear(10, 1)
        )

    def forward(self, x):
        return self.net(x)
```

Рисунок 5.8 – Код создания нейронной сети в Python

Обучение нейронной сети происходит через цикл с ручной настройкой функции потерь, метода обучения.

После обучения нейронной сети методом RMSProp значение функции потерь составило 6.063×10^{-5} , время обучения составило 7 секунд. Результаты прогнозирования представлены на рисунке 5.9. Среднеквадратичная ошибка составила 3.112×10^{-5} , что значительно меньше, чем при обучении методом RMSProp в Wolfram Mathematica.

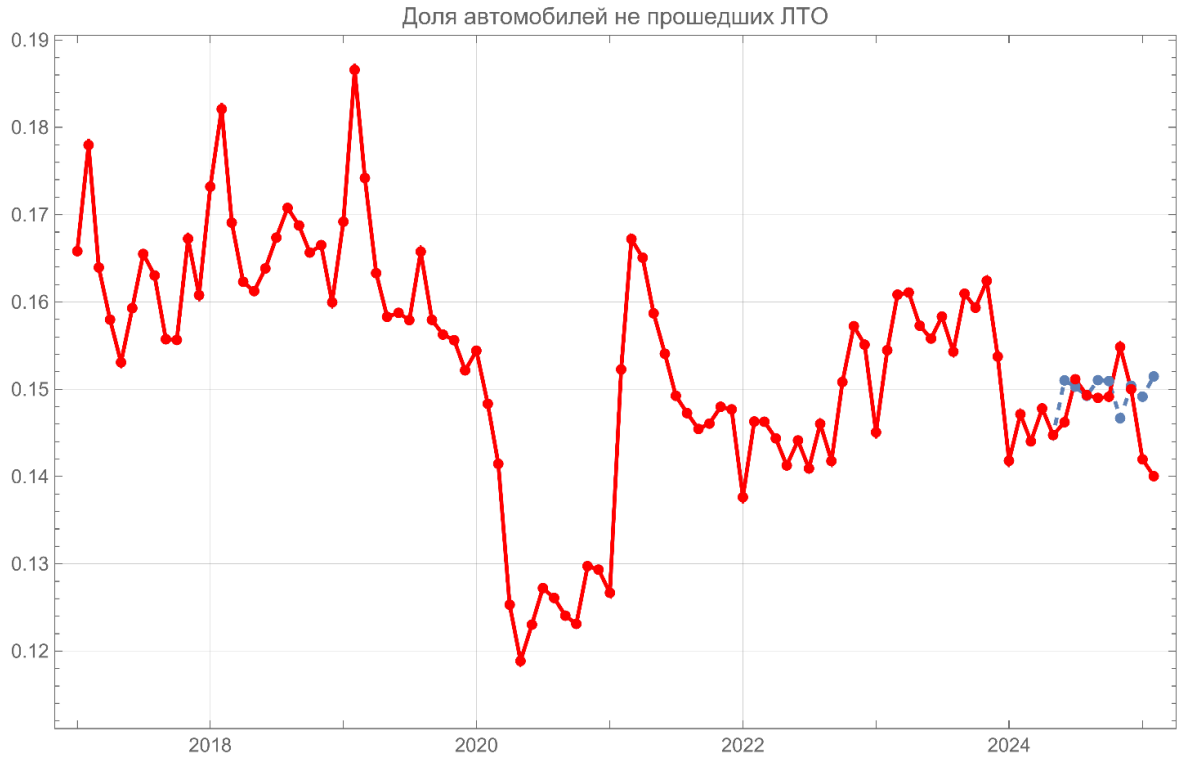

Рисунок 5.9 – Результат прогнозирования в PyTorch при обучении методом RMSProp

После обучения нейронной сети методом Adam в PyTorch значение функции потерь составило 5.478×10^{-5}, время обучения составило 7 секунд. Результаты прогнозирования представлены на рисунке 5.10.

Рисунок 5.10 – Результат прогнозирования в PyTorch при обучении методом Adam

Среднеквадратичная ошибка на тестовых данных составила 0.00292, что больше, чем при обучении методом RMSProp в PyTorch, но меньше, чем при обучении методом Adam в Wolfram Mathematica. Сравнение среднеквадратичных ошибок и функции потерь при обучения методами Adam и RMSProp в PyTorch представлено в таблице 5.4.

Таблица 5.4 – Сравнения результатов обучения нейронной сети в PyTorch

Метод обучения	Loss	MSE
RMSProp	6.063×10^{-5}	3.112×10^{-5}
Adam	5.478×10^{-5}	0.00292

Как видно на рисунке 5.10, при обучении методом Adam присутствуют аномальные значения, как и в случае обучения методом Adam в Wolfram Mathematica. Из этого можно сделать вывод, что этот метод с нашим набором признаком работает хуже, чем метод RMSProp.

5.5 Сравнение результатов обучения в Wolfram Mathematica и PyTorch

Как было отмечено ранее, метод Adam в обоих случаях показал неудовлетворительные результаты. В обоих случаях присутствуют аномальные значения, что видно на рисунках 5.6 и 5.10. В то же время, метод RMSProp в PyTorch показал наилучший результат, что видно и по рисунку 5.9, и по среднеквадратичной ошибке. Сравнение всех квадратичных ошибок при обучении методами RMSProp и Adam в Wolfram Mathematica и PyTorch представлено в таблице 5.5.

Таблица 5.5 – Сравнения среднеквадратичных ошибок в Wolfram Mathematica и PyTorch

Метод обучения	Wolfram Mathematica	PyTorch
RMSProp	0.00131	3.112×10^{-5}
Adam	0.0103	0.00292

На основании результатов можно сделать вывод, что для решения данной задачи прогнозирования лучше подходит фреймворк PyTorch. Подготовка данных и обучение в PyTorch занимает меньше времени, в сравнении с Wolfram Mathematica, однако требует больше кода.

ЗАКЛЮЧЕНИЕ

Нейронные сети представляют собой мощный инструмент в области обработки данных и искусственного интеллекта. Они открывают новые возможности в различных областях, включая компьютерное зрение, естественный язык, рекомендательные системы и другие.

Изучены основные инструменты для построения и обучения нейронных сетей в фреймворке PyTorch для языка Python. В работе апробированы и описаны три алгоритма обучения нейронной сети, на примерах продемонстрирована работа алгоритмов Adam и RMSProp. Также проведено сравнение результатов обучения в PyTorch с результатами, полученными на аналогичном примере в системе компьютерной алгебры Wolfram Mathematica на примерах двух задач. На основании сравнения результатов можно сделать вывод, что фреймворк PyTorch хуже подходит для решения задачи аппроксимации функции с нейронными сетями, так как не обладает широким функционалом для работы со сложными математическими вычислениями, в сравнении с системами компьютерной алгебры. Для решения задачи прогнозирования фреймворк PyTorch подходит лучше, так как подготовка данных требует меньше преобразований, время обучения занимает меньше времени и результаты получаются более точными, что видно на примере, описанном в данной работе.

Несовпадение результатов на аналогичных примерах объясняется различной реализацией конкретных алгоритмов на платформах Wolfram Mathematica и PyTorch, а также тем, что параметры в настройках алгоритмов по умолчанию устанавливаются разработчиками и их значение скрыто реализацией. Также существует различие в ходе обучения моделей. В Wolfram процесс обучения запускается одной функции и пользователь может задать лишь некоторые параметры заранее. Напротив, в PyTorch процесс обучения задается пользователем вручную, что позволяет более точно контролировать процесс обучения.

Обсуждаемые результаты, примеры обработки и визуализации данных, методы настройки инструментов искусственных нейронных сетей являются подтверждением широкого спектра возможностей рассматриваемой технологии интеллектуальной обработки данных.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Головко, В.А. Нейросетевые технологии обработки данных / В.А. Головко, В.В. Краснопрошин – Минск: БГУ, издание университетское, учебное пособие, 2017. – 270с.
2. Динамика ставок кредитно-депозитного рынка [Электронный ресурс]. – Режим доступа: https://www.nbrb.by/statistics/creditdepositmarketrates/. – Дата доступа: 21.02.2025.
3. Избежание чрезмерной аппроксимации путём использования тестовой выборки [Электронный ресурс]. – Режим доступа: https://www.wolfram.com/language/11/neural-networks/avoid-overfitting-using-a-hold-out-set.html.ru?product=mathematica. – Дата доступа: 12.11.2024.
4. Национальный статистический комитет Республики Беларусь [Электронный ресурс]. – Режим доступа: https://www.belstat.gov.by/. – Дата доступа: 21.02.2025.
5. Николенко, С. Глубокое обучение / С. Николенко, А. Кадурин, Е. Архангельская – Санкт-Петербург: Питер, 2018. – 480с.
6. Статистика продаж автомобилей в Республике Беларусь [Электронный ресурс]. – Режим доступа: https://auto-baa.by/statistic. – Дата доступа: 21.02.2025.
7. Стивенс Эли. PyTorch. Освещая глубокое обучение / Стивенс Эли, Антига Лука, Виман Томас – Санкт-Петербург: Питер, 2022. – 576с.
8. Стохастический градиентный спуск [Электронный ресурс]. – Режим доступа: http://mech.math.msu.su/~vvb/MasterAI/SGD.html. – Дата доступа: 12.11.2024.
9. 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.
10. Neural Networks. [Электронный ресурс]. – Режим доступа: https://reference.wolfram.com/language/guide/NeuralNetworks.html. – Дата доступа: 15.11.2024.
11. PyTorch Documentation. [Электронный ресурс]. – Режим доступа: https://PyTorch.org/docs/stable/index.html. – Дата доступа: 24.11.2024.
12. PyTorch Tutorials. [Электронный ресурс]. – Режим доступа: https://PyTorch.org/tutorials/. – Дата доступа: 24.11.2024.
13. Regression with Neural Networks [Электронный ресурс]. – Режим доступа:

https://reference.wolfram.com/language/tutorial/NeuralNetworksRegression.html. – Дата доступа: 15.03.2025.
14. Ruder, S. An overview of gradient descent optimization algorithms / S. Ruder. – Dublin; Insight Centre for Data Analytics: NUI Galway Aylien Ltd., 2017. – 14 p.