

Министерство науки и высшего образования Российской Федерации

ФГБОУ ВО «АЛТАЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт цифровых технологий, электроники и физики

Кафедра Вычислительной техники и электроники

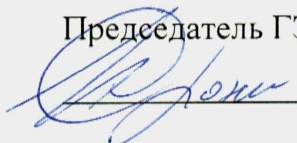
УДК 004.02

Бакалаврская работа защищена

«23» июня 2021 г.

Оценка Отлично

Председатель ГЭК д.т.н., проф.

 С.П. Пронин

Допустить к защите в ГЭК

«18» июня 2021 г.

Зав. кафедрой к.ф.-м.н., доц.

 В.В. Пашнев

**ПРОЕКТИРОВАНИЕ НЕЙРОННОЙ СЕТИ ДЛЯ РЕШЕНИЯ ОБРАТНОЙ
ЗАДАЧИ В ОПТИКЕ**

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ БАКАЛАВРА

БР 09.03.01.576.070ПЗ

обозначение документа

Студент группы

576



О.В. Деминов

и.о., фамилия

Руководитель работы

доцент, к.ф.-м.н.

должность, ученое звание



В.В. Пашнев


и.о., фамилия

Консультанты:

Нормоконтролер

доцент, к.ф.-м.н.

должность, ученое звание



А.В. Калачев

и.о., фамилия

Барнаул 2021

РЕФЕРАТ

Количество страниц.....	51
Количество рисунков.....	26
Количество источников.....	17
Количество приложений.....	3

РЕГРЕССИЯ, ВОССТАНОВЛЕНИЕ РЕГРЕССИИ, НЕЙРОННАЯ СЕТЬ,
ОБРАТНАЯ ЗАДАЧА В ОПТИКЕ, КОЭФФИЦИЕНТ АСИММЕТРИИ
АЭРОЗОЛЬНОГО РАССЕЯНИЯ, АНАЛИТИЧЕСКАЯ АППРОКСИМАЦИЯ
НЕЙРОННОЙ СЕТЬЮ

Данная работа посвящена решению обратной задачи в оптике, а именно вычислению коэффициента асимметрии аэрозольного рассеяния для длины волны 0.675 мкм методом аналитической аппроксимации нейронной сетью прямого распространения. В качестве исходных используются данные из результатов наблюдений интегрального коэффициента рассеянных световых потоков, оптической толщи и альбедо местности. Рассмотрены архитектуры нейронных сетей, описаны методы проектирования нейронных сетей прямого распространения, приложены исходные коды и блок-схемы программ, приведены графики обучения, выполнен интерполяционный анализ метода.

Разработаны программы для расчета коэффициента асимметрии аэрозольного рассеяния посредством работы с файлами табличного формата в виде скрипта и для обучения нейронной сети в формате оконного приложения для ОС Windows.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	5
ГЛАВА 1. ОБРАТНАЯ ЗАДАЧА И НЕЙРОННЫЕ СЕТИ.....	7
1.1 Постановка задачи	7
1.2 Биологический нейрон	8
1.3 Определение искусственного нейрона	10
1.4 Архитектуры нейронных сетей	11
1.5 Вычислительные мощности нейронных сетей	13
1.6 Виды нейронных сетей для решения регрессионных задач	14
ГЛАВА 2. ПРОЕКТИРОВАНИЕ НЕЙРОННОЙ СЕТИ ПРЯМОГО РАСПРОСТРАНЕНИЯ.....	20
2.1 Прямой ход	20
2.2 Функция ошибки.....	24
2.3 Градиентный спуск	25
2.4 Обратное распространение ошибки.....	25
2.5 Стохастический градиентный спуск с импульсом	26
2.6 Проблема переобучения.....	27
2.7 Нормализация и стандартизация данных	28
2.8 Класс нейронной сети прямого распространения	29
ГЛАВА 3. РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ НЕЙРОННОЙ СЕТИ ДЛЯ РЕШЕНИЯ ОБРАТНОЙ ЗАДАЧИ	31
3.1 Программа для нахождения коэффициента асимметрии аэрозольного рассеяния.....	31
3.2 Программа для обучения нейронной сети	33
3.3 Тестирование синтезированной нейронной сети	37

ЗАКЛЮЧЕНИЕ	42
СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ.....	43
ПРИЛОЖЕНИЕ 1	45
ПРИЛОЖЕНИЕ 2.....	46
ПРИЛОЖЕНИЕ 3.....	47

ВВЕДЕНИЕ

Как известно, на Земле происходят глобальные изменения климата, и вместе с этим растёт потребность в систематическом контроле и наблюдении за состоянием окружающей среды и атмосферы. Особое внимание привлекают к себе исследования радиационных свойств атмосферы и подстилающей поверхности, которые и формируют климат. Одной из характеристик, определяющей объёмы поступающей на поверхность Земли радиации, является коэффициент асимметрии аэрозольного рассеяния Γ_a [1-2].

Определение коэффициента асимметрии аэрозольного рассеяния – задача сложная. Здесь может быть два пути к её решению. Первый – использование аэрозольных ловушек в изучаемых пунктах на различных высотах, сбор проб частиц, их анализ и интегрирование результатов. В общем и целом это дорогостоящая операция. Второй путь – это измерение других компонент, которые имеют причинно-следственные связи с искомой, благодаря которым можно её рассчитать [2].

Проблемой второго пути является большая сложность или невозможность математически выразить причинно-следственную связь компонент. Но имея достаточно большую базу наблюдений, решение можно получить, воспользовавшись интерполяционными методами [1].

В последнее время идет развитие анализа больших данных с помощью нейронных сетей. Это произошло совместно с развитием вычислительных устройств, и Deep Learning перестал быть грёзой, сливаясь с жизнью человека. Нейронные сети – универсальные аппроксиматоры, и они могут выявлять закономерности и связывать параметры между собой, поэтому их можно использовать для решения различных задач.

В настоящей работе спроектирована нейронная сеть для вычисления коэффициента асимметрии аэрозольного рассеяния из результатов

наблюдений интегрального коэффициента рассеянных световых потоков Γ , оптической толщи τ_a и альбедо местности q .

Задачи исследования:

- 1) Изучение архитектур нейронных сетей.
- 2) Выбор архитектуры нейронной сети для решения поставленной задачи.
- 3) Выбор методов обучения нейронной сети.
- 4) Реализовать программно выбранные методы.
- 5) Обучить нейронную сеть.
- 6) Протестировать нейронную сеть.

ГЛАВА 1. ОБРАТНАЯ ЗАДАЧА И НЕЙРОННЫЕ СЕТИ

1.1 Постановка задачи

Вычисление параметра коэффициента асимметрии аэрозольного рассеяния Γ_a – операция дорогостоящая и сложная, и не каждая исследовательская группа имеет доступ к таким технологиям и средствам. В работах [5-6] были изучены поведение величины коэффициента асимметрии рассеянных световых потоков в зависимости от различных атмосферных параметров. На основе этих исследований был разработан метод [7-8], который может использоваться для определения коэффициента асимметрии аэрозольного рассеяния Γ_a из результатов наблюдений яркости неба в солнечном альмукантарате $f_n(\varphi)$, оптической толщи τ_a и альбедо подстилающей поверхности q без применения аппарата решения обратных задач. Результаты расчетов были приведены в трудах [1] в виде подробных таблиц. В них описано, что с помощью интерполяционных методов можно выражать атмосферные параметры, пользуясь приложенными табличными данными. Но практическое применение становится сложным ввиду ручного определения границ нахождения внутри таблицы интегрального Γ с последующим интерполированием.

Для решения задачи восстановления регрессии мы воспользуемся статистическими методами аналитической аппроксимации, которые относятся к численным методам решения обратных задач оптики [3]:

$$R(\varphi(\tau_a, Z_0, q), f^*(\Gamma)) = \Gamma_a, \quad (1.1)$$

где φ – вектор, состоящий из результатов измерений оптической толщи τ_a , зенитного угла солнца Z_0 , альбедо подстилающей поверхности q , $f^*(\Gamma)$ – результат интегрирования переноса излучения по результатам наблюдения $f_n(\varphi)$, R – какой-либо нелинейный оператор. Он должен быть нелинейным, исходя из статьи [1]. В роли R удобнее всего использовать

нейронную сеть, поскольку в случае использования методов классической регрессии требуется немало времени, чтобы сделать необходимые преобразования исходных данных, которые можно было бы использовать для построения функций регрессии. Нейронные сети могут моделировать нелинейности автоматически в отличие от ручного моделирования сплайнов в многомерной линейной регрессии. Существуют различные виды, архитектуры и способы обучения нейронных сетей, поэтому нужно их исследовать и подобрать наиболее оптимальные.

1.2 Биологический нейрон

Нейронные сети появились в результате исследований в области искусственного интеллекта. Были предприняты попытки воспроизвести способность реальных нейронов обучаться и корректировать ошибки, моделируя низкоуровневую структуру мозга [4].

Основной областью исследований по искусственному интеллекту во второй половине прошлого века были экспертные системы. Эти системы проектировались на основе высокоуровневого моделирования процесса мышления, например, на манипуляциях с символами. В результате стало ясно, что эти модели хоть и могут приносить пользу, но они не охватывают некоторые важные аспекты интеллекта человека. В частности, ученые были ограничены вычислительными мощностями того времени, и чтобы воссоздать интеллект, нужно построить систему с похожей архитектурой.

Число нейронов в человеческом мозге может достигать десятков миллиардов штук. Это число ошеломляет. При этом каждый нейрон связан с другими нейронами и имеет в среднем несколько тысяч связей с другими нейронами. Сложность модели неподъемная. Но реализовать простые модели реально. Отталкиваясь от специфики выполняемой задачи, искусственную

нейронную сеть можно спроектировать и обучить так, что они будут работать даже лучше человеческого мозга.

Нейроны – это клетки (рис. 1.1), которые способны распространять и обрабатывать электрохимические сигналы. Нейрон состоит из разветвленной структуры ввода (дендрит) и выхода (аксон) информации, а также из ядра. Аксоны одной клетки соединяются с дендритами других клеток с помощью синапсов. При активации нейрона посылается электрохимический сигнал по аксону. По синапсу этот сигнал передается другим нейронам, которые в свою очередь тоже могут активироваться. Активация нейрона происходит тогда, когда суммарный уровень сигналов, принятых в ядре из дендритов, превысит какой-то уровень, называемый порогом активации [16].

Интенсивность сигнала, который получает нейрон, зависит от активности синапсов. Каждый синапс имеет свою протяженность, и особые химические вещества передают сигнал вдоль неё. Суть обучения главным образом состоит в изменении “силы” синаптических связей.

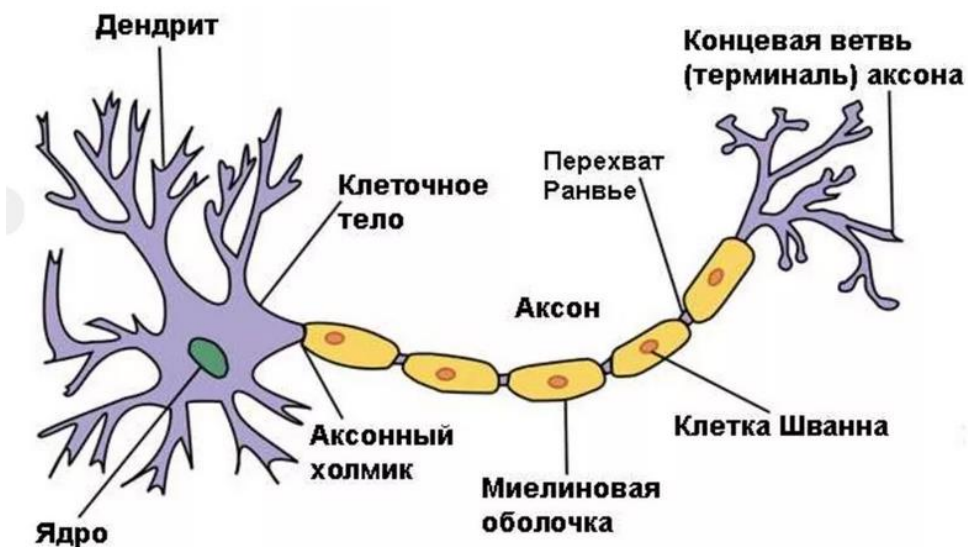


Рис. 1.1 Строение биологического нейрона.

1.3 Определение искусственного нейрона

Искусственная нейронная сеть – математическая модель и её аппаратная или программная реализация, основанная на принципах организации и функционирования биологических нейронных сетей. Впоследствии, после разработки алгоритмов обучения, модели ушли в практическое пользование: задачи прогнозирования, задачи управления, распознавания образов и др.

Основой искусственной нейронной сети составляют простые и, зачастую, однотипные элементы (рис. 1.2), имитирующие работу мозга человека .

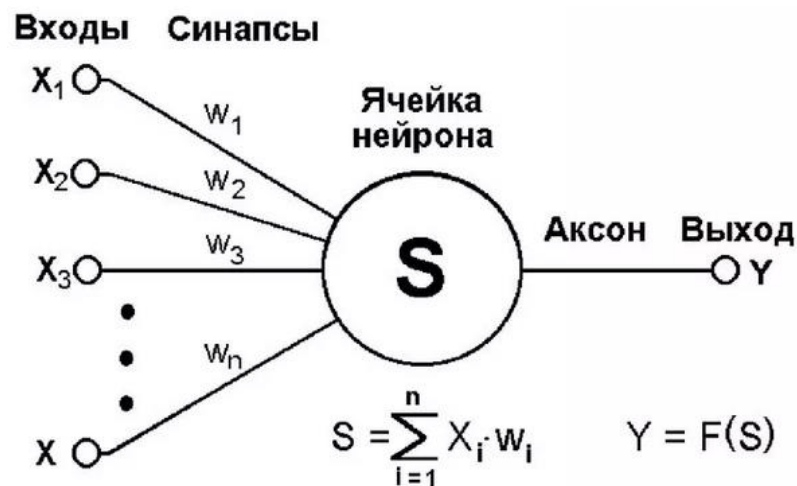


Рис. 1.2 Модель искусственного нейрона.

Нейрон имеет группу синапсов – однонаправленных входов, соединенных с выходами других нейронов. Каждый синапс имеет вес w_i – величина синаптической связи.

Каждый нейрон имеет текущее состояние, которое обычно определяется, как взвешенная сумма его входов:

$$S = \sum_{i=1}^n x_i w_i. \quad (1.2)$$

Нейрон имеет аксон - выходную связь, с которой сигнал возбуждения или торможения поступает на синапсы следующих нейронов. Выход нейрона есть функция его состояния:

$$y = f(s). \tag{1.3}$$

Функция f называется активационной, и может иметь различный вид (рис. 1.3).

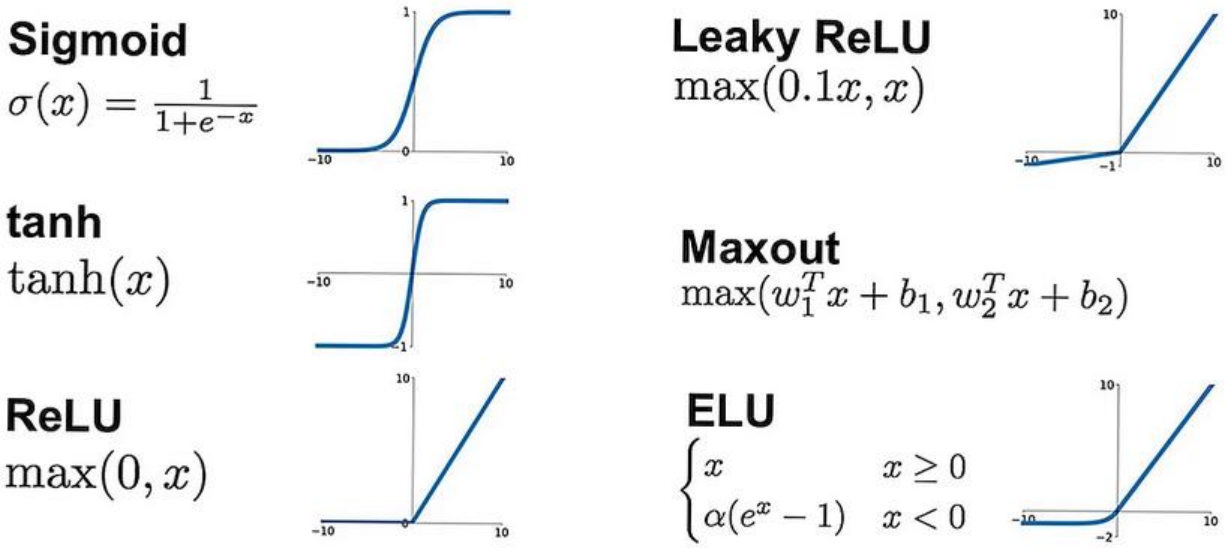


Рис. 1.3 Различные функции активации.

Обработка входных сигналов происходит параллельно. Это следствие того, что нейроны объединяют в слои, и операции выполняются послойно.

Выбор структуры нейронной сети осуществляется в соответствии с особенностями и сложностью задачи. Для некоторых классов задач уже существуют оптимальные конфигурации. Если же задача не может быть сведена ни к одному из известных классов, разработчику приходится решать задачу синтеза новой конфигурации. В большинстве случаев оптимальный вариант искусственной нейронной сети получается опытным путем.

1.4 Архитектуры нейронных сетей

ИНС можно представить в виде направленного графа со взвешенными связями, в котором узлы являются нейронами. ИНС можно сгруппировать в два класса по типу связей (рис. 1.4): сеть прямого распространения и рекуррентные сети.

Графы сетей прямого распространения не имеют петель, а рекуррентные наоборот. Петли образуют обратную связь, и часто рекуррентные сети называют сетями с обратной связью.

Сети прямого распространения являются статическими, и на один заданный вход они реагируют одной совокупностью выходных значений, не зависящих от предыдущего состояния сети. В то же время рекуррентные сети являются динамическими, т.к. из-за обратных связей в них модифицируются входы нейронов, что изменяет состояние сети.

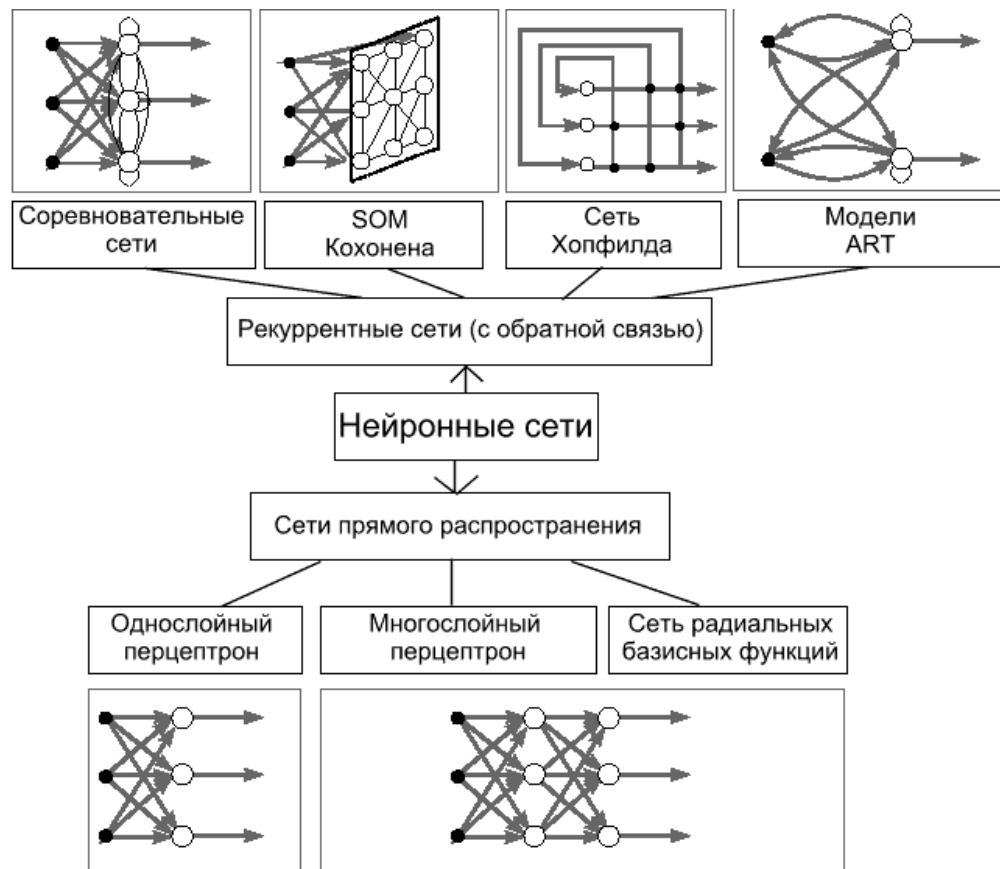


Рис. 1.4 Виды нейронных сетей.

Рекуррентные нейронные сети (RNN), как правило, подходят для задач анализа последовательностей ввиду присутствия обратной связи. Поэтому сети RNN применимы в таких задачах, где нечто целостное разбито на части, например: распознавание рукописного текста или распознавание речи.

Для сетей прямого распространения существуют различные методы восстановления регрессии, аппроксимации и прогнозирования временных

рядов, поэтому в разрезе данной исследовательской работы имеет смысл разобрать их подробнее.

1.5 Вычислительные мощности нейронных сетей

Любая непрерывная функция n аргументов на единичном кубе $[0, 1]^n$ представима в виде суперпозиции непрерывных функций одного аргумента и операции сложения [10]:

$$f(x^1, x^2, \dots, x^n) = \sum_{k=1}^{2n+1} h_k \left(\sum_{i=1}^n \varphi_{ik}(x^i) \right), \quad (1.4)$$

где h_k, φ_{ik} – непрерывные функции, причём φ_{ik} не зависят от выбора f .

Записанное здесь выражение имеет структуру нейронной сети прямого распространения с одним скрытым слоем из $2n + 1$ нейронов. Таким образом, двух слоёв достаточно, чтобы вычислять произвольные непрерывные функции с любой точностью.

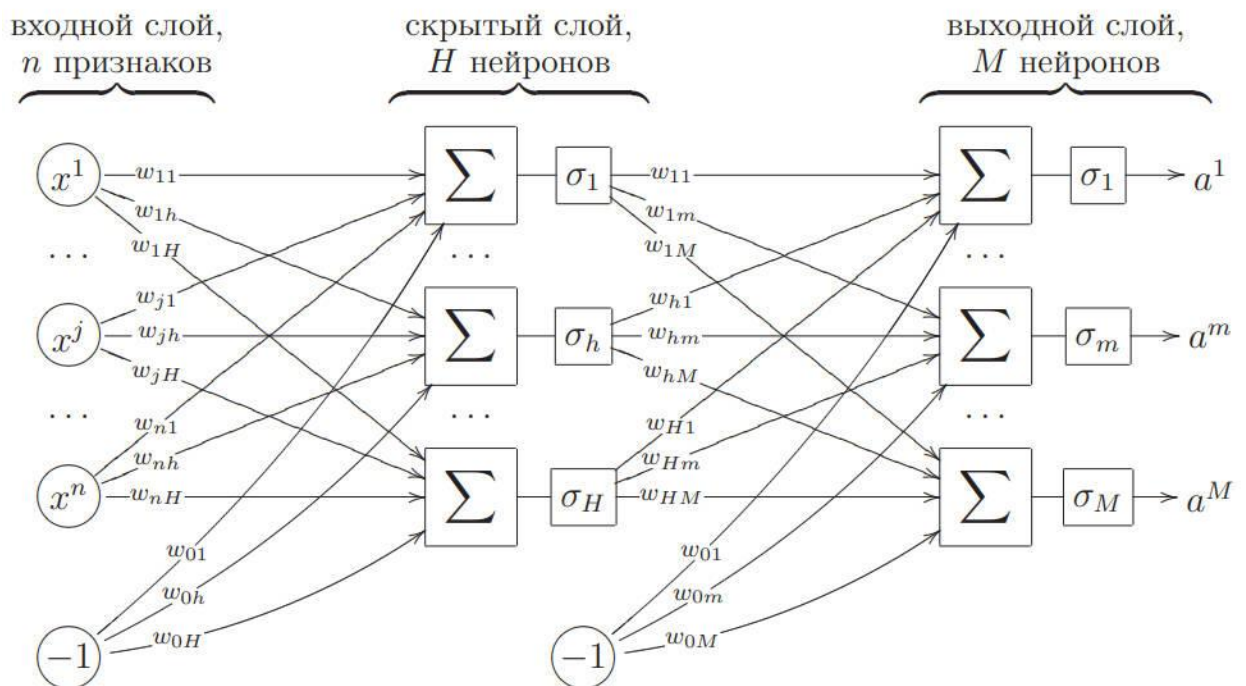


Рис. 1.5 Структурная схема сети прямого распространения.

Пусть X – компактное пространство, $C(X)$ – алгебра непрерывных на X вещественных функций, F – линейное подпространство в $C(X)$, замкнутое относительно нелинейной непрерывной функции ϕ , содержащее константу ($1 \in F$) и разделяющее точки множества X . Тогда F плотно в $C(X)$ [12].

Это интерпретируется как утверждение об универсальных аппроксимационных возможностях произвольной нелинейности: с помощью линейных операций и единственного нелинейного элемента ϕ можно получить модель, способную вычислять любую непрерывную функцию с любой желаемой точностью. Однако данная теорема ничего не говорит о количестве слоёв нейронной сети (уровней вложенности суперпозиции) и о количестве нейронов, необходимых для аппроксимации произвольной функции [9].

Таким образом, нейронные сети являются универсальными аппроксиматорами функций. Возможности сети возрастают с увеличением числа слоёв и числа нейронов в них. Двух-трёх слоёв, как правило, достаточно для решения подавляющего большинства практических задач классификации, регрессии и прогнозирования.

1.6 Виды нейронных сетей для решения регрессионных задач

1.6.1. Сеть радиально-базисных функций

Радиально-базисная функция – это функция вида

$$\varphi(x) = f(|x - c|), \quad (1.5)$$

где c – центр. Для них характерно то, что они меняются радиально вокруг некоторого центра. Типичным примером такой функции может послужить функция Гаусса (рис. 1.6):

$$f(x) = \exp\left(-\frac{(x-c)^2}{\sigma}\right). \quad (1.6)$$

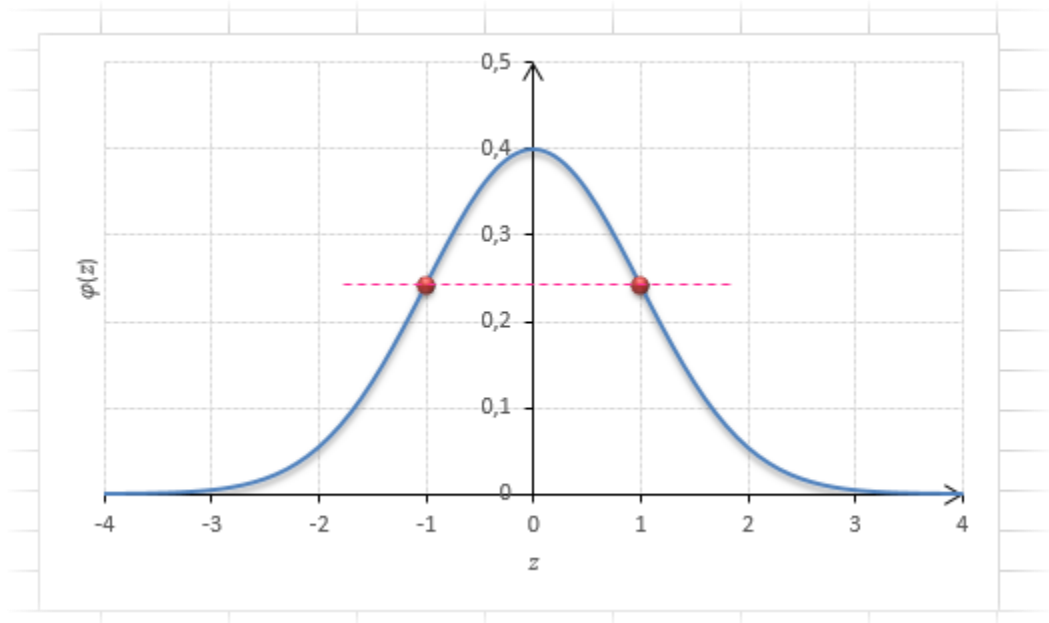


Рис. 1.6 График функции Гаусса при $c = 0$ и $\sigma = 1$.

Отличие таких функций от сигмоидальных состоит в том, что вместо деления пространства гиперплоскостью в зависимости от $wx < 0$ и $wx > 0$, она делит пространство решений гиперсферой (рис. 1.7).

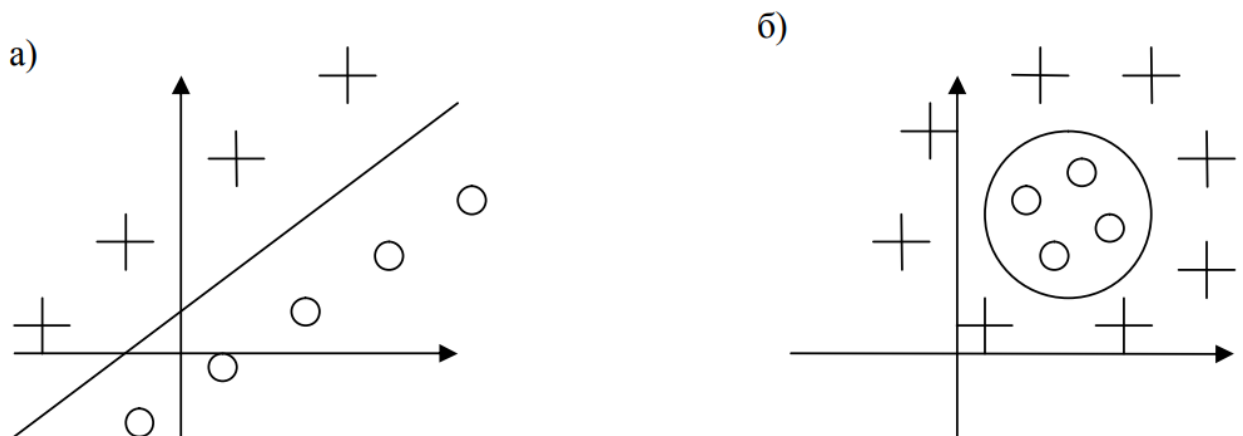


Рис. 1.7 Разделение двумерного пространства: а) сигмоидальная функция; б) радиальная функция [10].

Нейронные сети, в которых есть слой нейронов с активационной функцией радиально-базисного вида, называются радиальными.

Классическая радиальная нейронная сеть строится из трёх слоёв: входной, скрытый слой нейронов с радиальной функцией активации, выходной слой, осуществляющий взвешенное суммирование результата работы скрытого слоя (рис. 1.8).

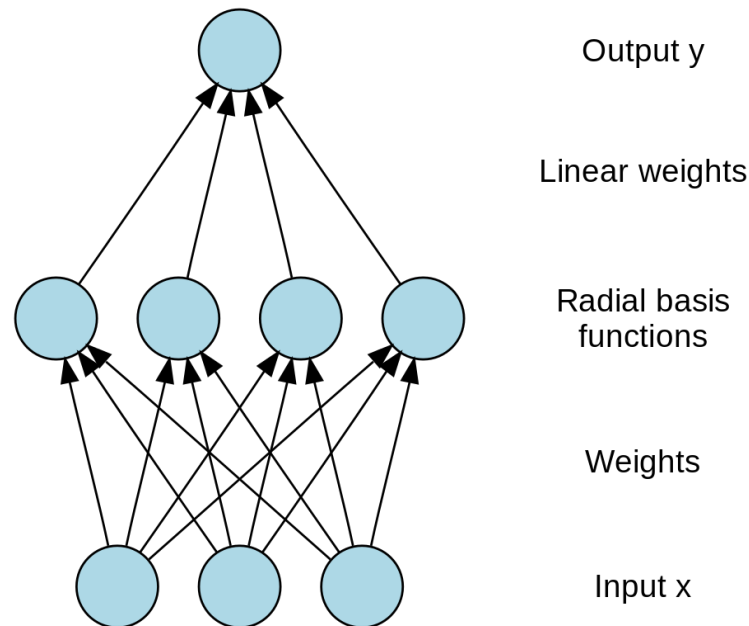


Рис. 1.8 Структурная схема радиальной нейронной сети.

Основанием для определения аппроксимирующих возможностей радиальных нейронных сетей служит теорема Т. Ковера о распознаваемости образов. В этой теореме утверждается, что нелинейные проекции образов в некоторое многомерное пространство могут быть линейно разделены с большей вероятностью, чем при их проекции в пространство с меньшей размерностью [10].

На практике, может быть поставлена интерполяционная задача: отразить n различных векторов $x^i \in X (i = 1, 2, \dots, n)$ из d -мерного пространства в множество действительных чисел $d^i \in R (i = 1, 2, \dots, n)$. Для этого нужно порядка n радиальных нейронов в скрытом слое, а сеть будет реализовывать функцию $F(x): X \rightarrow R$ такую, что $F(x^i) = d^i$. Причем вблизи интерполяции можно достигнуть очень хорошей точности [10].

Однако зачастую выборка бывает очень большой, и количество нейронов скрытого слоя от этого возрастет. Можно взять количество нейронов $M < n$, и задача будет выглядеть в виде

$$F(x) = \sum_{i=1}^M w^i \varphi(\|x - c^i\|), \quad (1.7)$$

где $M < p$, а c^i - центры, которые необходимо определить.

Таким образом, задача аппроксимации может быть сформулирована как задача подбора количества радиальных функций, а поиск значений настраиваемых параметров сети, при которых решение уравнения (1.7) было бы наиболее точным.

Ясно, что в максимальной оптимизации, радиально-базисные функции будут вырождаться в функции Дирака, и аппроксимация вне узлов будет плохая. Поэтому, несмотря на то, что эти сети быстро обучаются, необходимо ширину радиальной функции подбирать практически, что не очень удобно с точки зрения программирования и слишком уж зависит от удачи. Или же искать полином, который будет ортогонален всем другим точкам.

Этот недостаток можно частично решить с помощью генеративных алгоритмов обучения, где параметры формируются случайно, и выбирается та конфигурация сети, которая показала наилучшие результаты в тестах.

1.6.2. Метод кусочно-постоянной аппроксимации

Метод кусочно-постоянной аппроксимации можно реализовать с помощью нейросети гибридного вида, состоящей из слоя Кохонена (рис. 1.9) и линейного нейрона.

Слой Кохонена состоит из нейронов Кохонена, и относится к типу рекуррентных сетей. Он применяется для того, чтобы разделить входные данные на классы по определенным признакам, которые выделяет сама сеть. Количество нейронов Кохонена определяют количество классов. При

совпадении признаков, сеть выдает “1” на совпадающий класс. Обучается без учителя.

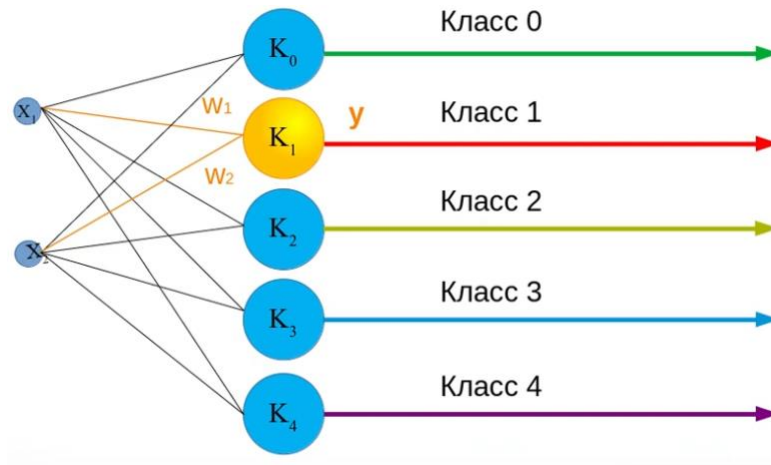


Рис. 1.9 Слой Кохонена.

Чтобы кластеризацию Кохонена превратить в алгоритм кусочной аппроксимации, к нему добавляется ещё один нейрон с линейной функцией активации, образующий третий слой с весами v_1, v_2, \dots, v_m , которые определяют, какое значение будет на выходе нейронной сети для данного класса входного вектора [9].

Нам недостаточно кусочно-постоянной аппроксимации для решения задачи, т.к. хотелось бы достичь дифференцируемости выходных данных.

1.6.3. Метод гладкой аппроксимации

Метод гладкой аппроксимации можно также реализовать с помощью слоя Кохонена, в котором будут заданы ядра для каждого кластера. Ядра могут иметь вид функции Гаусса (1.6). Выходы нейронов будут равны вероятности принадлежности входного вектора к кластерам. Далее происходит взвешенное суммирование и применение формулы Надорая-Ватсона для заданного ядра в выходном нейроне:

$$a(x) = \sum_{m=1}^M \frac{v_m K(\rho(x, w_m))}{\sum_{s=1}^M K(\rho(x, w_s))}, \quad (1.8)$$

где $K(\rho(x, w_m))$ – ядро выходного слоя, $\sum_{s=1}^M K(\rho(x, w_s))$ – сумма ядер слоя Кохонена.

Из минусов данного метода можно отметить плохую экстраполяцию и возможные искажения данных при работе с многомерными данными.

1.6.4. Многослойный персептрон

Многослойный персептрон, или же персептрон Румельхарта имеет хорошие аппроксимационные свойства (см. п. 1.4). С развитием глубокого обучения нейросетей стало доступно использование сравнительно меньшего количества нейронов при большем количестве слоев.

Проектировка нейронной сети этого типа проста, легко масштабируема, в то же время эффективна, и требует от программиста сравнительно меньше затрат, чем другие в процессе обучения. Например, можно воспользоваться методом кросс-валидации данных и не работать самому с набором входных данных, в отличие от сетей, рассмотренных ранее, в которых предполагается строгая интерполяция в узлах для осуществления аппроксимации, и в случае, если один из важных узлов будет пропущен, то экстраполяция будет плохой.

Из минусов этого метода можно отметить отсутствие единой методологии проектировки, т.к. не всегда ясно, за что какой нейрон отвечает ввиду большого количества скрытых слоев.

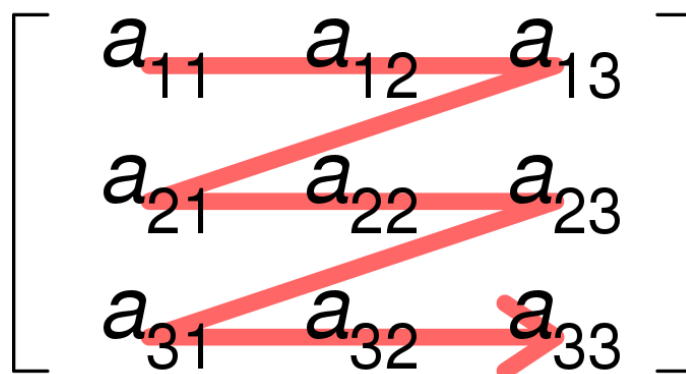
Таким образом, предпочтительный способ реализации нейросети для восстановления регрессии – это многослойный персептрон с нелинейными функциями активаций. Для него нужно подобрать параметры сети и практически опробовать в решении обратной задачи.

ГЛАВА 2. ПРОЕКТИРОВАНИЕ НЕЙРОННОЙ СЕТИ ПРЯМОГО РАСПРОСТРАНЕНИЯ

2.1 Прямой ход

Давайте обозначим массив подаваемых на вход данных как X . Одно решение в нём можно ориентировать по строкам или по колонкам:

Row-major order



Column-major order

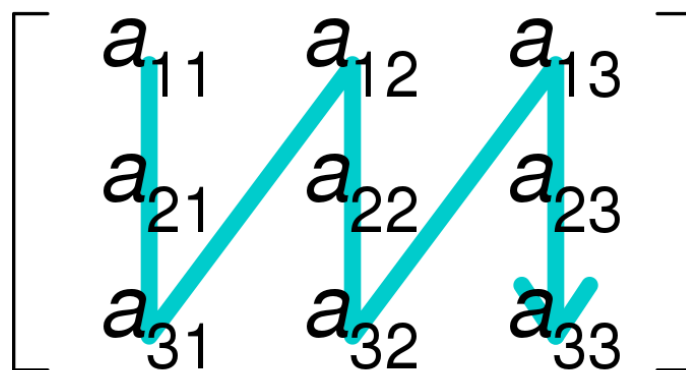


Рис. 2.1 Основной порядок строк.

Выберем ориентацию по колонкам (Column-major order). Тогда первая строка будет отвечать за подаваемые значения угла зенита Солнца Z_0 , вторая – за величину оптической толщи τ_a , третья – за альbedo подстилающей

поверхности q , четвертая – за коэффициент асимметрии рассеяния световых потоков Γ . Сразу можно сказать, что количество входов 4.

В сетях прямого распространения каждый нейрон принимает на вход взвешенную сумму входов. Давайте обозначим под W вектор весов. Тогда W^i – матрица i -того слоя, W_j^i – строка весов j -ого нейрона i -того слоя, W_{jk}^i – вес для k -ого параметра j -ого нейрона i -ого слоя. Тогда для того, чтобы получить взвешенную сумму входов нейронов первого скрытого слоя нужно перемножить матрицы между собой. Также нужно для каждого i -ого нейрона задать смещение b_j^i , чтобы нейроны могли описать зависимость не только относительно нуля. Для этого можно создать отдельный вход, на который всегда будет подаваться единица, и указать для него веса, но это не очень экономично. Поэтому будем их указывать в векторе смещений. Тогда взвешенная сумма входов всех нейронов преобразуется в матрицу:

$$S^1 = W^1 X + b^1. \quad (2.1)$$

Далее нейроны должны полученные взвешенные суммы преобразовать, т.е. среагировать на них. Это делается с помощью функций активации, результат которых зависят преимущественно от знака аргумента. Как правило, для задач восстановления регрессии подходят следующие функции:

1) Линейная

$$y(x) = x. \quad (2.2)$$

Это обычная линейная функция. Эта функция в отличие от многих может выдавать отрицательные значения и работать в качестве усилителя. Её можно использовать в качестве функции активации нейронов выходного слоя. Сумма линейных функций тоже линейная. Её график можно посмотреть на рисунке 2.2.

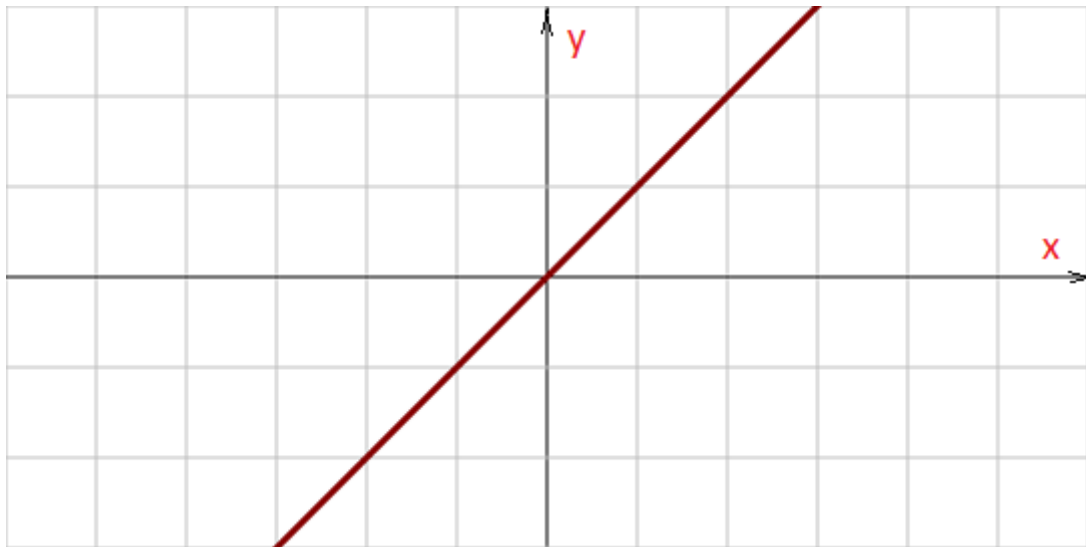


Рис. 2.2 График линейной функции.

2) ReLu

$$y(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases} \quad (2.3)$$

Эта функция активации похожа на предыдущую за исключением того, что она по своей природе нелинейная. Нейронная сеть, основанная на таких функциях, может выдавать на своем выходе нелинейные значения. Поскольку многие регрессионные зависимости нелинейные, то это может пригодиться. Но у неё есть небольшой минус – она не дифференцируема, плюс в отрицательной части аргумента её градиент равен нулю, что нам может помешать в обучающих алгоритмах. Частично этот недостаток можно решить, вводя вместо нуля другую функцию ax , где a прижимает функцию к оси абсцисс, но не приравнивает её к нулю. Ввиду своей простоты, быстро выполняется на ЭВМ. Её график можно посмотреть на рисунке 2.3.

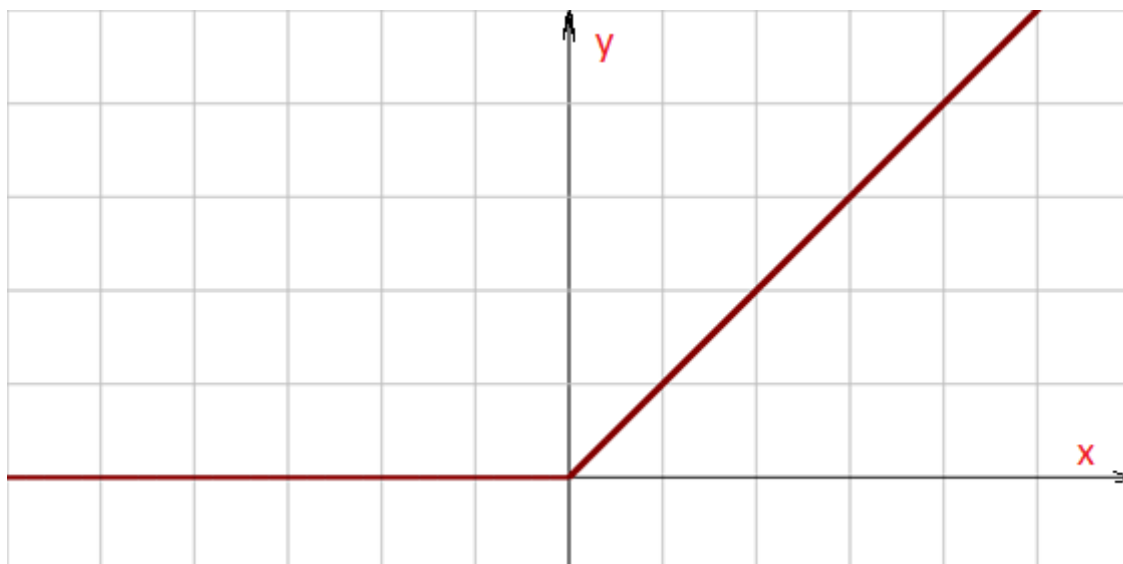


Рис. 2.3 График функции ReLu.

3) Softplus

$$y(x) = \ln(e^x + 1) . \quad (2.4)$$

Эта функция похожа на ReLu, но при этом дифференцируема в нуле.

В отличие от сигмоидальных функций, у них слабо наблюдается явление затухающего градиента, что не мешает обучать глубокие нейронные сети. Её график можно посмотреть на рисунке 2.4.

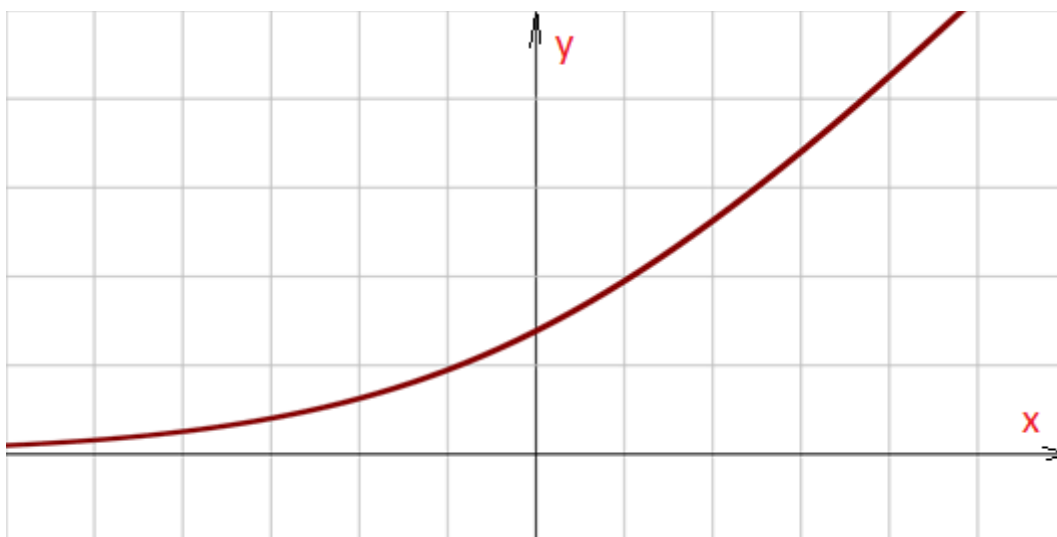


Рис. 2.4 График функции softplus.

Таким образом, вводим матрицу функций активаций Y , значения которых определяются как $Y_{jk}^i = y(S_{jk}^i)$. Они являются входными значениями

для нейронов следующего слоя: $S^{i+1} = W^{i+1}Y^i + b^{i+1}$. Матрица активаций выходного слоя и будет искомым коэффициентом асимметрии аэрозольного рассеяния Γ_a . Количество нейронов выходного слоя 1.

2.2 Функция ошибки

Сети прямого распространения обучаются по алгоритму обратного распространения ошибки, который устремляет ошибку к нулю. Это подразумевает обучение с учителем. В качестве учителя используется таблица из статьи [1].

Для реализации нужно ввести функцию ошибки μ . Можно позаимствовать её из метода наименьших квадратов:

$$\mu(Y^*) = \frac{1}{2}(Y^* - Y)^2, \quad (2.5)$$

где Y^* значения функций активации выходного слоя, т.е. проверка сходимости с табличным значением Γ_a .

Для того, чтобы оценить среднее квадратичное отклонение ошибки, μ можно разделить на количество примеров N :

$$\mu(Y^*) = \frac{1}{2N}(Y^* - Y)^2. \quad (2.6)$$

Для функции ошибки существуют различные методы регуляризации. Например, ещё к функции ошибки прибавляются квадраты весов. Это будет означать, что они не будут принимать большие значения. Если прибавлять модули весов, то некоторые веса будут зануляться, что поможет удалить ненужные нейроны, что поможет сформировать архитектуру сети [15].

2.3 Градиентный спуск

Пусть дана некая функция $f(x)$. Известно, что в её области решений имеется экстремум – минимум. Для того, чтобы численно его найти можно воспользоваться методом градиентного спуска.

Из курса математического анализа известно, что функция возрастает по направлению её градиента. Следовательно, чтобы прийти к минимуму нам нужно двигаться в сторону, противоположную направлению градиента:

$$x^{i+1} = x^i - \nabla f(x^i) * l, \quad (2.7)$$

где l – величина шага(learning grate). От его величины зависят скорость и точность нахождения минимума. Больше l – меньше затраченное время, но меньше точность, меньше l – больше время выполнения, больше точность.

В окрестностях точки минимума градиент стремится к нулю. Поэтому сигналом к остановке алгоритма может служить то, что градиент по модулю стал меньше какой-то заданной окрестности нуля ε . Или же можно выполнять заданное количество итераций.

Именно такой метод стоит в основе метода обратного распространения ошибки.

2.4 Обратное распространение ошибки

После того, как была определена ошибка выходного слоя, можно посчитать ошибку каждого нейрона, обусловленной неправильно настроенными синапсами, точнее его весами.

Как известно, градиент функции

$$\nabla f(x) \approx \frac{x+\varepsilon}{\varepsilon}, \quad (2.8)$$

где ε малое по модулю приращение. Можно было бы так же изменять на величину ε вес, отслеживать изменение ошибки и менять вес в зависимости от этого. Но это не эффективно, т.к. количество синапсов может быть велико и считать для каждого прямое распространение по 2 раза за

итерацию тратит очень много ресурсов. Поэтому учеными был придуман алгоритм обратного распространения ошибки, с помощью которого можно за 1 итерацию посчитать градиенты всех весов. [11]

Этот метод заключен в 4 формулы:

$$\delta^L = \nabla_y J \circ y'(s^L) \quad (2.9)$$

$$\delta^l = ((W^{l+1})^T \delta^{l+1}) \circ y'(s^l) \quad (2.10)$$

$$\frac{\partial J}{\partial b_j^l} = \delta_j^l \quad (2.11)$$

$$\frac{\partial J}{\partial w_{jk}^l} = y_k^{l-1} \delta_j^l \quad (2.12)$$

Где δ^l – вектор ошибок нейронов скрытых слоев, δ^L – вектор ошибок нейронов выходного слоя.

Функция активации в промежуточных слоях у нас будет (2.4), а в выходном (2.2). Так мы можем дать нейронной сети больше раскрыться, т.к. линейный нейрон на выходе дает возможность трансформироваться другим нейронам так, как им будет удобно, а выходные данные будут масштабироваться.

Для выходного слоя:

$$y'(s^L) = 1 \quad (2.13)$$

Для скрытых:

$$y'(s^l) = \frac{1}{e^x + 1} e^x \quad (2.14)$$

Градиент целевой функции:

$$\nabla_y J = \frac{(y - y^*)}{2N} \quad (2.15)$$

Так можно получить все градиенты и обучить нейронную сеть градиентным спуском, отнимая от весов полученные значения.

2.5 Стохастический градиентный спуск с импульсом

Одной из модификаций градиентного спуска является моделирование катящегося шара по пространству целевой функции. Этот метод позволяет

“проезжать” через локальные минимумы и задерживаться в экстремумах, а также полезен при вытянутой функции потерь. Именно его и будем использовать в нашей нейронной сети при её обучении [11].

Для этого вводятся скорости v_{jk}^i для каждого веса w_{jk}^i , которые будут определять скорость движения по пространству целевой функции (2.6). В начале скорость равна нулю. Тогда в следующей итерации алгоритма (2.9-2.12) она будет равна:

$$v_{jk}^i(t+1) = v_{jk}^i(t)B - \frac{\partial J}{\partial w_{jk}^i}, \quad (2.16)$$

где B – величина импульса от 0 до 1, t – номер текущей итерации.

Веса корректируются согласно:

$$w_{jk}^i = w_{jk}^i + l v_{jk}^i(t), \quad (2.17)$$

где l – величина шага. Чем больше B – тем быстрее движемся по плоскости функции потерь, но тем медленнее будет сходиться алгоритм обратного распространения в экстремуме.

2.6 Проблема переобучения

Для того, чтобы нейронная сеть не переобучилась мы поделим данные на обучающую и тестовую выборки. На одну длину волны приходится 1200 измерений параметров. Из них 1100 выделим на обучающую выборку, а 100 на тестовую. Тогда мы сможем отслеживать поведение нейронной сети на тестовой выборке. В случае переобучения, функция ошибки (2.6) будет возрастать на тестовом наборе данных, а на обучающем уменьшаться. Поэтому мы будем отслеживать этот процесс на графике, где по оси абсцисс будут отложены эпохи обучения, т.е. полная итерация алгоритма обратного распространения, а по оси ординат значение функции ошибки для текущей эпохи, и в случае переобучения остановим нейронную сеть (рис. 2.5).

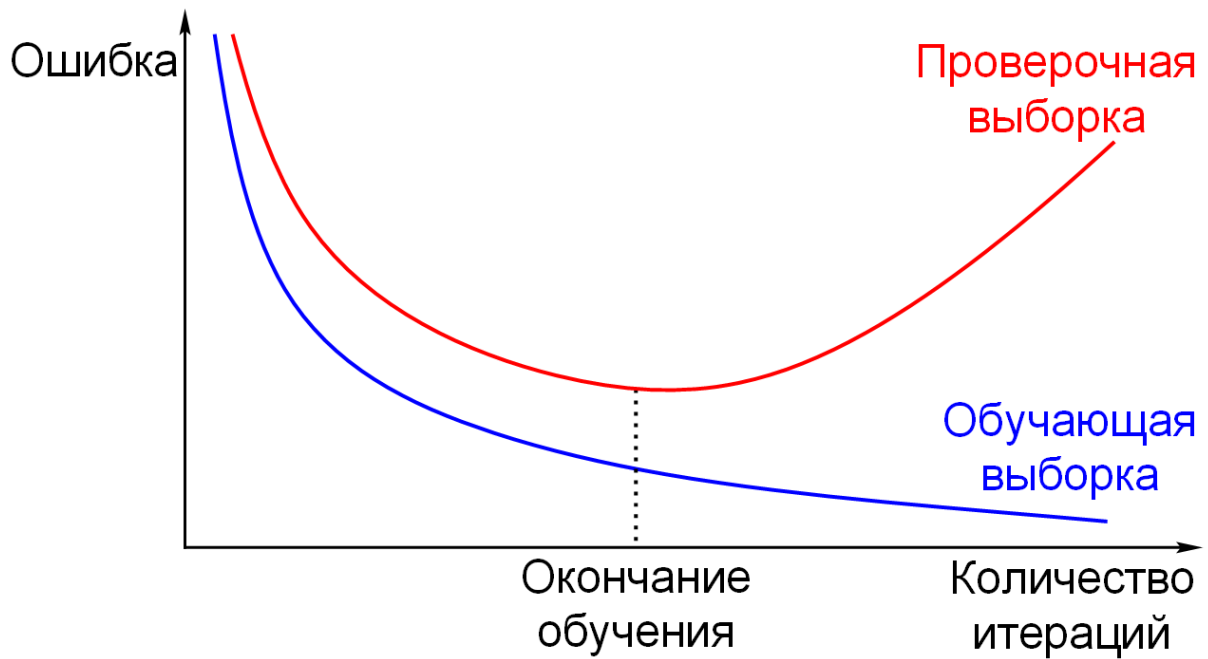


Рис. 2.5 Типичный график обучение сети.

2.7 Нормализация и стандартизация данных

Мы будем использовать батч-нормализацию только для входного батча и обычную нормализацию по наибольшей величине для выходного в процессе обучения. Это делается для того, чтобы входные данные имели одинаковое влияние на нейроны входного слоя, и нейронная сеть обучалась быстрее [14].

Для начала для каждой выборки Γ , q , Z_0 и τ_a рассчитывается среднее значение:

$$\bar{\mu} = (\sum_{i=1}^N \frac{z_0^i}{N}, \sum_{i=1}^N \frac{\tau_a^i}{N}, \sum_{i=1}^N \frac{q^i}{N}, \sum_{i=1}^N \frac{\Gamma^i}{N}), \quad (2.18)$$

где N – количество обучающих данных. В п. 2.6. выбрано это значение 1100.

Через неё находим стандартное отклонение:

$$\bar{\sigma} = (\sqrt{\frac{\sum_{i=1}^N (z_0^i - \bar{\mu}_{z_0})^2}{N-1}}, \sqrt{\frac{\sum_{i=1}^N (\tau_a^i - \bar{\mu}_{\tau})^2}{N-1}}, \sqrt{\frac{\sum_{i=1}^N (q^i - \bar{\mu}_q)^2}{N-1}}, \sqrt{\frac{\sum_{i=1}^N (\Gamma^i - \bar{\mu}_{\Gamma})^2}{N-1}}). \quad (2.19)$$

Далее центруем данные и делим на стандартное отклонение:

$$x^* = \frac{x - \bar{\mu}}{\bar{\sigma}}. \quad (2.20)$$

Эти значения подаем на вход нейронной сети.

Максимальное значение для Γ_a в таблице – 13.85. На это значение делим вектор значений Γ_a , чтобы они принимали значения от 0 до 1. Для восстановления данных из нейронной сети также придется умножать на 13.85.

2.8 Класс нейронной сети прямого распространения

Для реализации нейронной сети был использован язык C++ и библиотеку UBLAS из собрания библиотек Boost.

Библиотека UBLAS охватывает основные операции линейной алгебры над векторами и матрицами, прописанные в стандарте BLAS. Связующим звеном между контейнерами, представлениями и операциями шаблонов выражений является в основном интерфейс итератора, соответствующий STL [13].

На основе предложенных методов был реализован класс сети прямого распространения, диаграмма которого отображена на рисунке 2.6.

Чтобы инициализировать нейронную сеть, нужно создать объект класса “Web” и применить к нему метод “InicWeb”, который принимает на вход матрицы значений обучающих выборок входных и выходных данных, количество слоёв, массив значений количества нейронов каждого слоя.

Метод “GoForvard” выполняет прямое распространение сети, метод “DetErr” позволяет рассчитать функцию ошибки по методу наименьших квадратов, метод “GoBack” позволяет обучить сеть обратным распространением ошибки, используя импульсный градиентный спуск. Нормировка выходов происходит с помощью метода NormY, стандартизация

и нормализация входов с помощью NormBatch. Листинг класса можно посмотреть в приложениях 2-3.

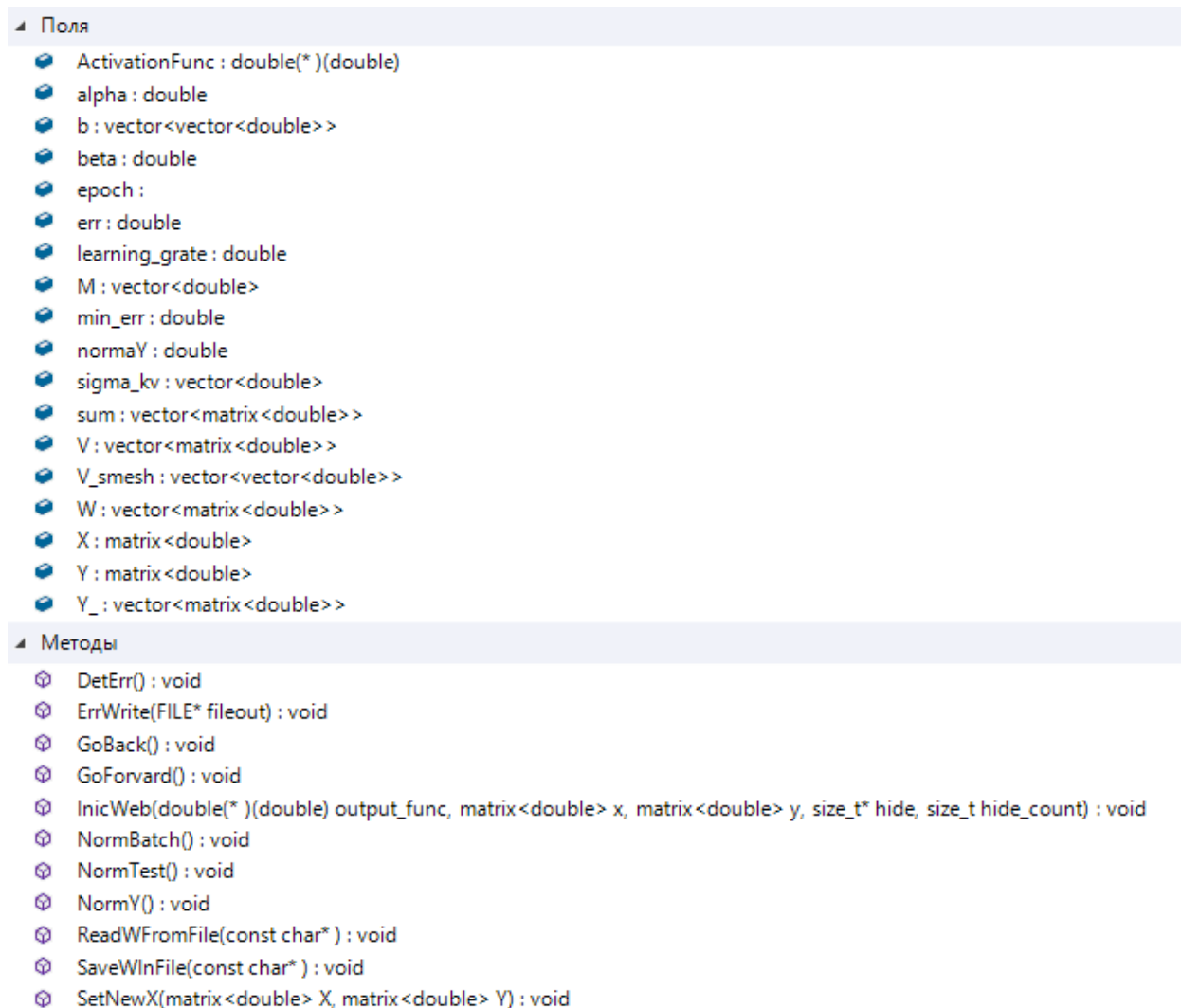


Рис. 2.6 Диаграмма класса нейронной сети прямого распространения в IDE Visual Studio.

Таким образом, завершена подготовка к синтезу архитектуры нейронной сети прямого распространения, а именно мы реализовали многослойный персептрон Румельхарта и все необходимые методы, и можно приступать к отладке и тестам.

ГЛАВА 3. РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ НЕЙРОННОЙ СЕТИ ДЛЯ РЕШЕНИЯ ОБРАТНОЙ ЗАДАЧИ

3.1 Программа для нахождения коэффициента асимметрии аэрозольного рассеяния

Чтобы нейронная сеть рассчитала значение Γ_a необходимо произвести подготовительные действия. Параметры Z_0, τ_a, q, Γ нужно поместить в файл с разрешением csv. Это табличный формат. Для создания файла такого типа подойдет программа Microsoft excel. Данные нужно располагать так, как показано на рисунке 3.1, т.е. в колонки “А”, “В”, “С”, “D” помещать измеренные значения зенитного угла Солнца, оптической толщи, альбеда подстилающей поверхности и рассчитанные коэффициенты асимметрии рассеянных световых потоков соответственно для каждого отдельного наблюдения.

Программа для расчета значений Γ_a выполнена в качестве скрипта на языке C++ с применением ранее разработанного класса нейронной сети. На вход программы поступает полное имя табличного файла. Например, это можно сделать из командной строки или же перенести входной файл вручную курсором на значок программы (рис. 3.2). На выход подается файл с рассчитанными значениями Γ_a в колонке “Е” для каждого наблюдения с приставкой “_output.csv” (рис. 3.3) по аналогии со входным файлом.

Блок-схема скрипта представлена на рисунке 3.4. Листинг кода скрипта на языке C++ можно посмотреть в приложении 1.

Для правильной работы программы необходим конфигурационный файл обученной нейронной сети, который формируется в процессе выполнения программы, описанной в разделе 3.2.

	O8			f_x
	A	B	C	D
1	72,5	0,1	0	2,61
2	65	0,2	0	3,64
3	65	0,15	0,6	2,32
4	70	0,15	0,7	2,71
5	72,5	0,05	0,4	1,76
6	75	0,05	0,1	1,98
7	65	0,1	0,2	2,65
8	65	0,25	0,1	3,15
9	67,5	0,2	0,5	2,55
10	67,5	0,2	0,4	2,65
11	67,5	0,25	0,1	3,11
12	67,5	0,25	0,2	2,97

Рис. 3.1 Значения Z_0 , τ_a , q , Γ во входном файле в редакторе MS Excel.

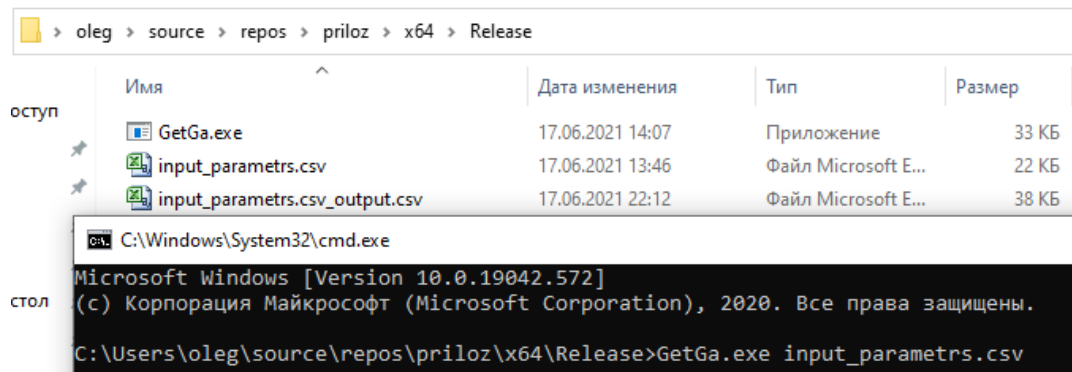


Рис. 3.2 Пример применения скрипта из командной строки ОС Windows.

	K1				f_x
	A	B	C	D	E
1	72,5	0,1	0	2,61	7,1
2	65	0,2	0	3,64	8,65
3	65	0,15	0,6	2,32	7,099
4	70	0,15	0,7	2,71	10,651
5	72,5	0,05	0,4	1,76	6,001
6	75	0,05	0,1	1,98	7,101
7	65	0,1	0,2	2,65	8,649
8	65	0,25	0,1	3,15	7,098
9	67,5	0,2	0,5	2,55	7,098
10	67,5	0,2	0,4	2,65	7,102
11	67,5	0,25	0,1	3,11	7,102
12	67,5	0,25	0,2	2,97	7,102

Рис. 3.3 Выходной файл работы скрипта с рассчитанными значениями Γ_a .

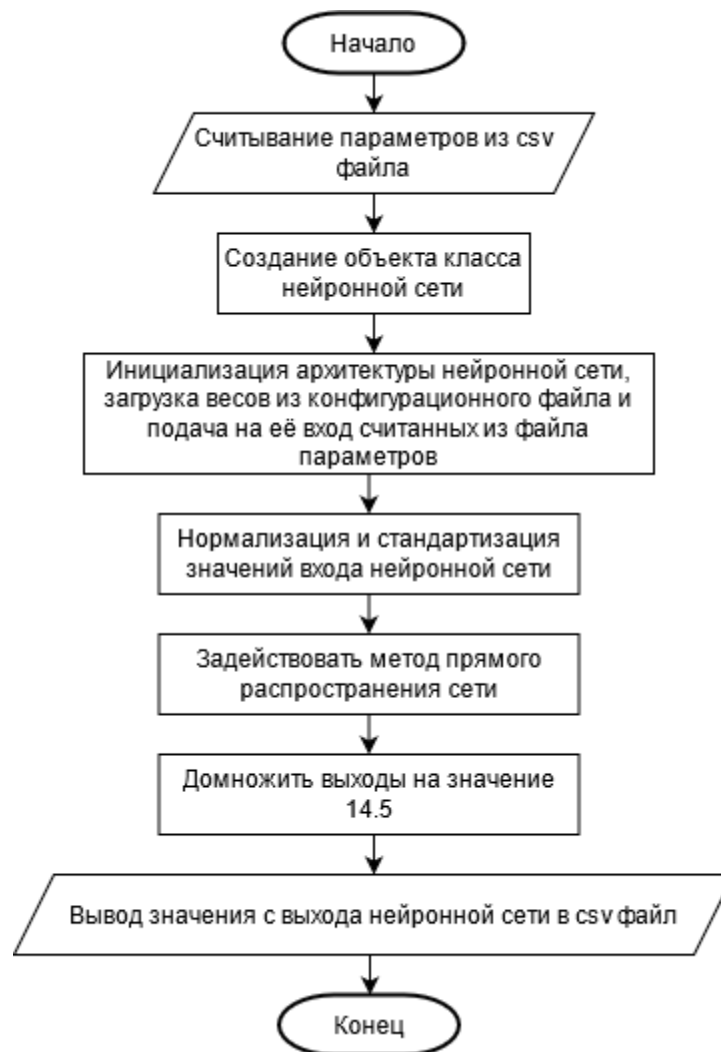


Рис. 3.4 Блок-схема программы скрипта по расчету коэффициента Γ_a .

3.2 Программа для обучения нейронной сети

Для обучения нейронной сети нужно занести исходные $\Gamma_a, Z_0, \tau_a, q, \Gamma$ в csv файл под названием “train.csv” так, как показано на рисунке 3.5, в колонки “A”, “B”, “C”, “D”, “E” соответственно.

Программа загружает, нормализует, стандартизирует входные данные и подбирает веса методом обратного распространения ошибки для синапсов нейронной сети, которые после нажатия кнопки “Остановить обучение” запишет в специальный конфигурационный файл “w_file.cfg”. Интерфейс программы можно посмотреть на рисунке 3.6. Он был разработан с помощью

элементов “Windows forms” и Common Language Runtime(CLR) от .NET, которая предоставляет среду выполнения (среду CLR), которая выполняет код и предлагает службы, облегчающие процесс разработки, в частности на языке C++ [17].

K29 f_x					
	A	B	C	D	E
1	13,85	67,5	0,05	0,2	2,23
2	6	65	0,05	0	1,97
3	8,65	72,5	0,25	0,1	3,39
4	13,85	70	0,3	0,4	4,06
5	7,1	67,5	0,2	0,2	2,89
6	13,85	72,5	0,2	0,3	3,82
7	13,85	65	0,25	0,4	3,93
8	7,1	70	0,1	0,5	2,19
9	6	65	0,25	0,5	2,36
10	8,65	70	0,1	0	2,87
11	13,85	67,5	0,15	0,5	3,2
12	13,85	67,5	0,1	0,3	2,95

Рис. 3.5 Входные данные в csv файле для обучения нейронной сети.

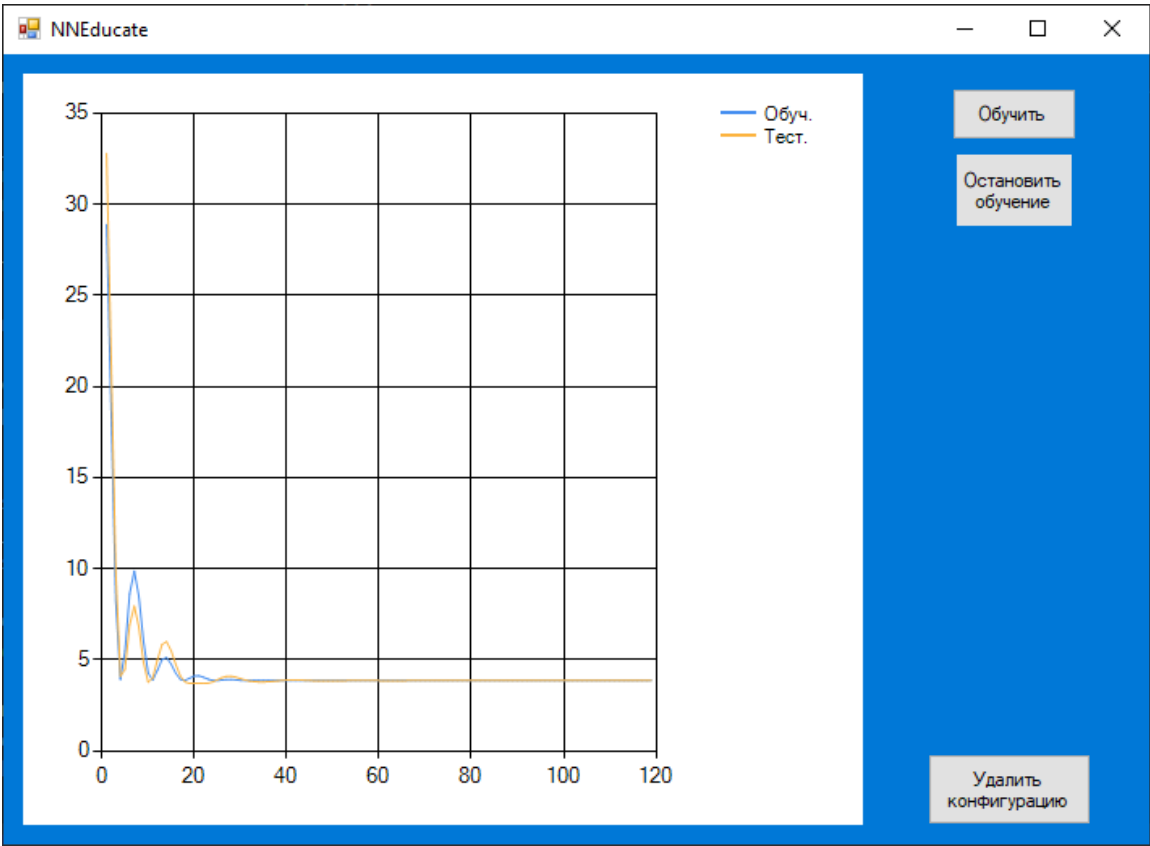


Рис. 3.6. Интерфейс программы для обучения нейронной сети.

В интерфейсе программы вы можете наблюдать график обучения сети, где по оси абсцисс откладываются значения эпохи, а по оси ординат значения функции потерь для текущей эпохи. Синим цветом выводится целевая функция для обучающей выборки, а желтым для тестовой.

Блок-схему подпрограммы обучения можно посмотреть на рисунке 3.7. Подпрограмма выполняется асинхронно по отношению к основному потоку формы посредством использования класса “BackgroundWorker”, который выполняет операцию в отдельном потоке. Поэтому графики обучения выводятся динамически в реальном времени, что дает нам возможность эффективно отслеживать тенденции переобучения. Каждые 1000 эпох график обнуляется для экономии памяти и удобства отслеживания переобучения.

После нажатия кнопки “Обучить”, она блокируется, и запускается асинхронный поток. В том случае, если найден в папке с программой конфигурационный файл нейронной сети, например, с прошлых итераций обучения, то используются веса конфигурационного файла. В противном случае веса инициализируются случайно для каждого синапса в пределах от $(-\frac{1}{NK}, \frac{1}{NK})$, где N – количество нейронов на входе, а K – количество нейронов на выходе слоя.

Чтобы сохранить текущую конфигурацию нейронной сети, если показатели её функции потерь удовлетворяют, нужно нажать кнопку “Остановить обучение”. Если по каким-то причинам старая конфигурация не удовлетворяет, можно нажать кнопку “Удалить конфигурацию”, и при следующем запуске итераций метода обратного распространения, будут сформированы новые веса.

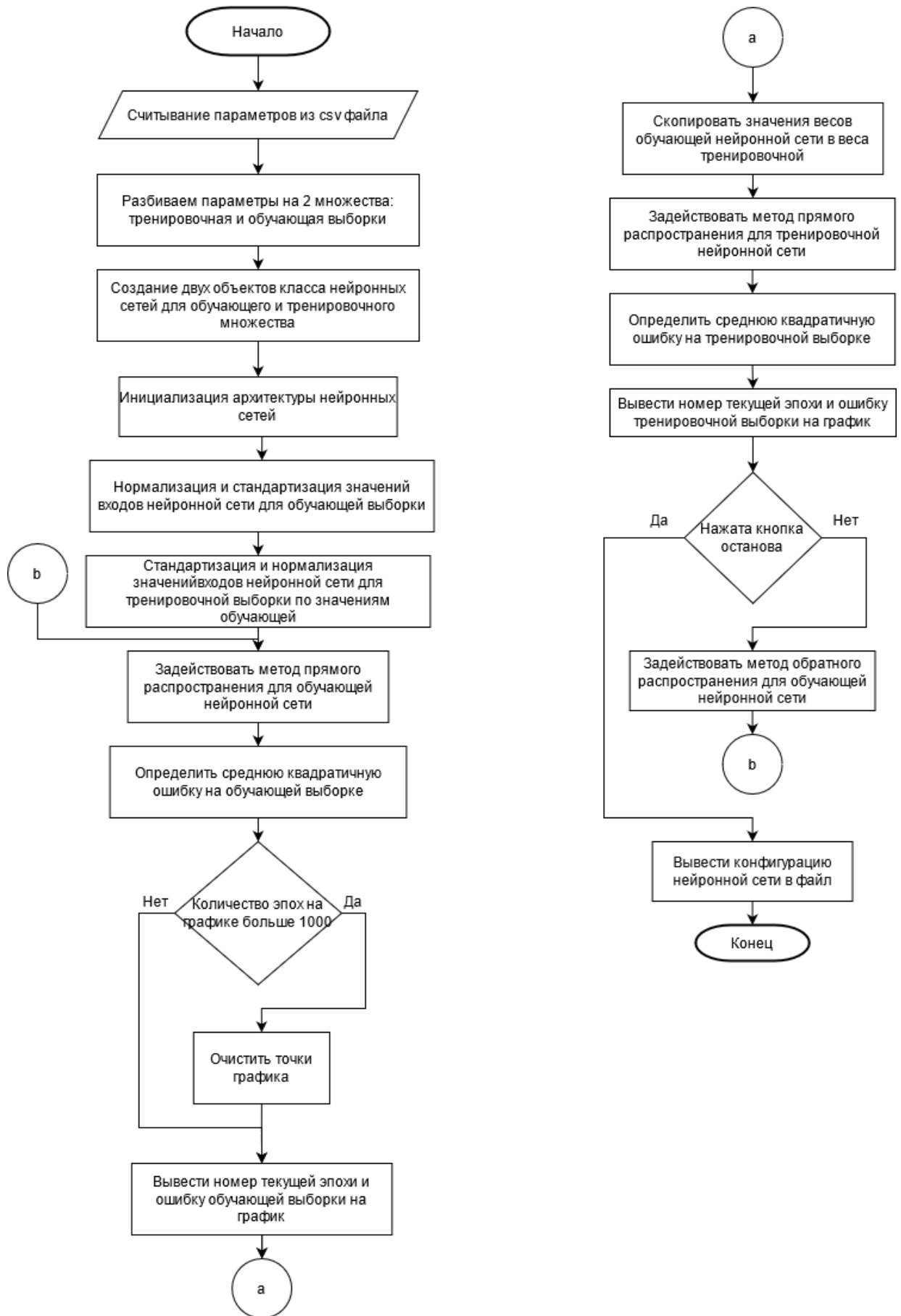


Рис. 3.7 Блок-схема подпрограммы обучения нейронной сети.

3.3 Тестирование синтезированной нейронной сети

Поскольку использование одного скрытого слоя влечет за собой использование 1200 нейронов (по размеру выборки), что не есть хорошо, сеть подбиралась многослойная, поскольку на практике увеличение количества слоев дает возможность использовать меньше нейронов, что менее ресурсоемко.

Опытным путем в режиме отладки была подобрана конфигурация нейронной сети, топология которой отображена на рисунке 3.8.

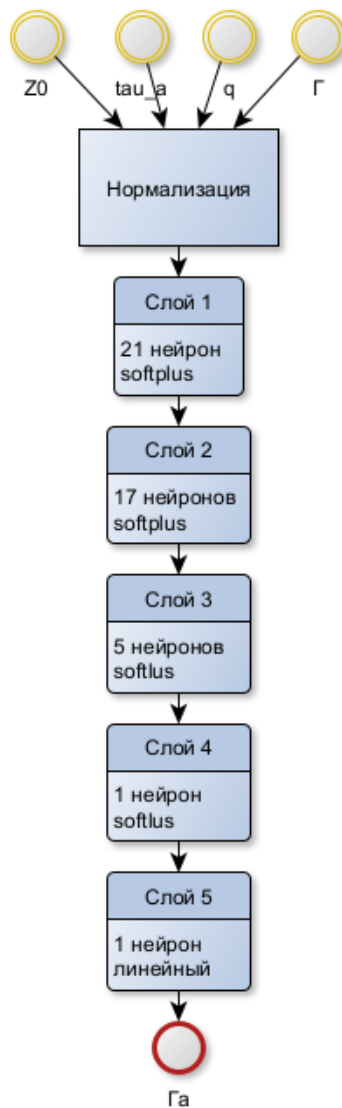


Рис. 3.8 Топология нейронной сети.

Для её обучения использовались данные расчетов коэффициентов асимметрии рассеянных световых потоков Γ при различных Z_0, τ_a, q, Γ_a для длины волны 0.675 мкм. Импульс $B = 0.8$ и шаг $\alpha = 0.01$ для метода импульсного градиентного спуска. Полный график обучения можно посмотреть на рисунке 3.9. Часть графика, построенного с помощью ранее представленного оконного приложения, можно посмотреть на рисунке 3.10. На графике заметна тенденция переобучения ввиду роста целевой функции на тестовой выборке, поэтому нужно остановить выполнение программы.

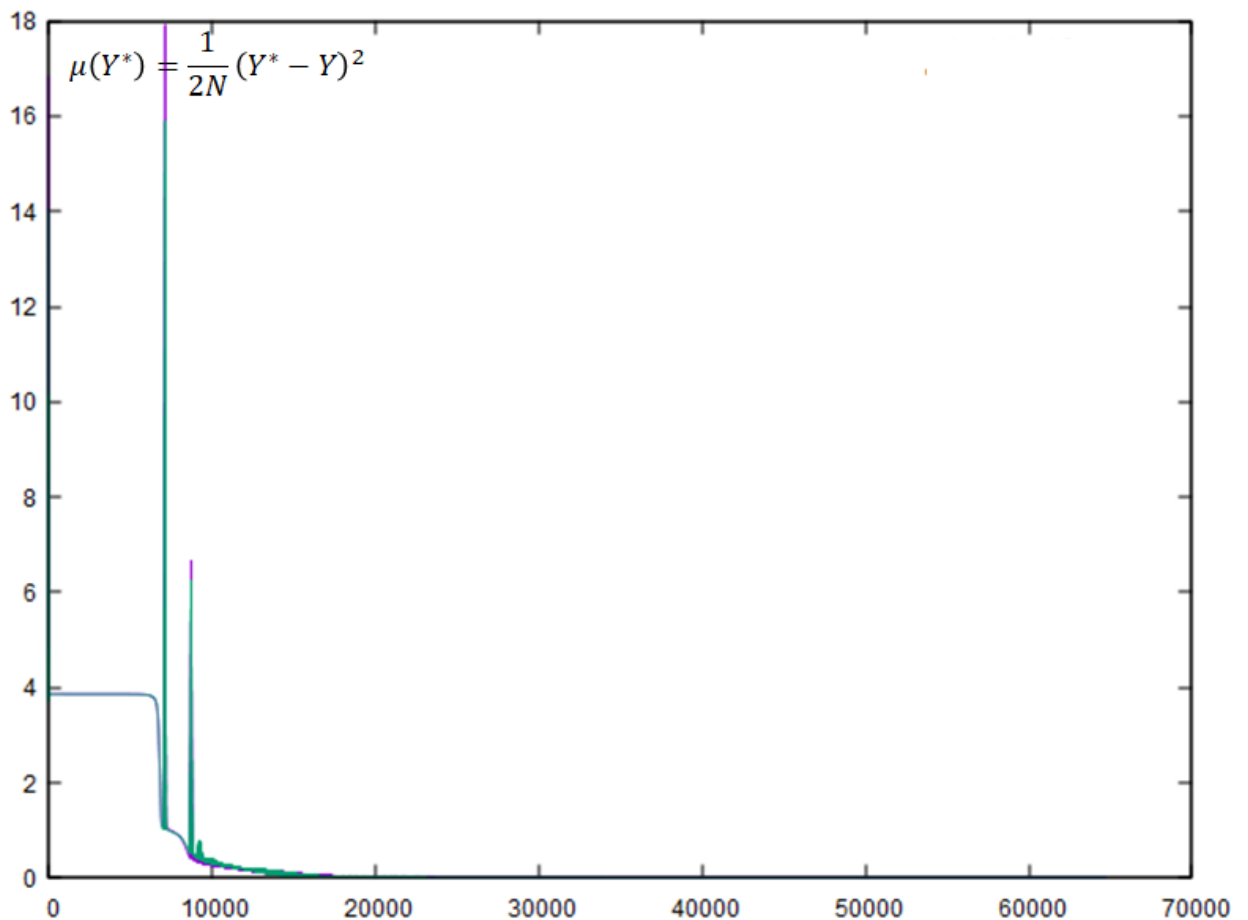


Рис. 3.9 График обучения нейронной сети (по оси абсцисс номер эпохи, по оси ординат значение функции потерь), зеленый для тестовой, фиолетовый для обучающей выборки.

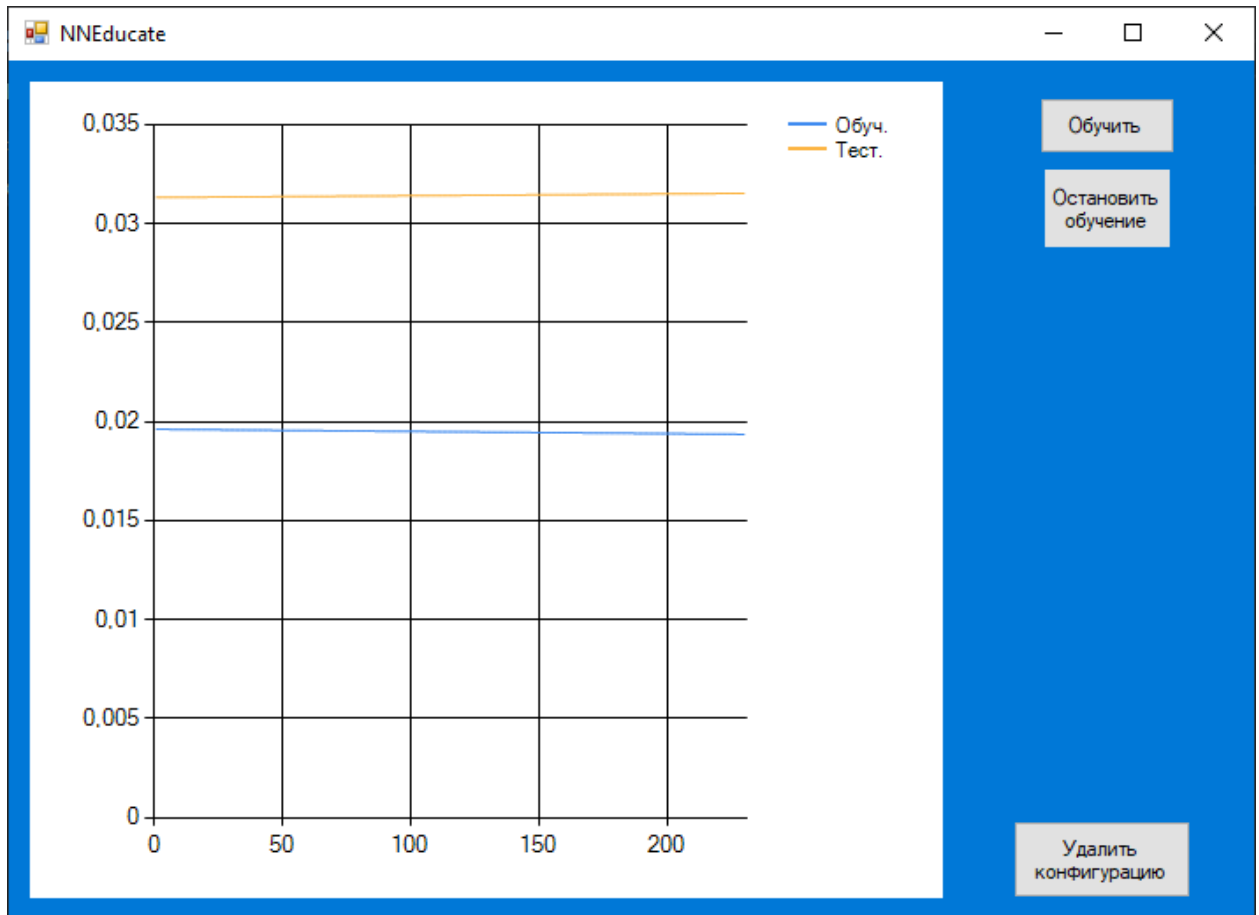


Рис. 3.10 Окно приложения с графиком переобучения.

В ходе обучения нейронной сети был преодолен устойчивый локальный минимум целевой функции в значениях близких к 4. Вероятно, что выбросы на графиках соответствуют максимумам целевой функции, и они препятствовали.

Средние квадратичные отклонения получились равными 0.02 на обучающей и 0.03 на тестовой выборках.

На рисунке 3.11 показаны аппроксимирующие свойства реализованной нейронной сети. При $\tau_a = 0.3$, $Z_0 = 65$ подавались по возрастанию значения Γ для каждого альбеда подстилающей поверхности $q = (0.1, 0.3, 0.5, 0.7)$. Точками обозначены табличные значения. Можно заметить, что регрессия восстанавливается и работает в условиях экстраполяции. Также наблюдается тенденция расхождения значений в крайних узлах интерполяций, т.е. при значениях Γ_a равных 6.00 и 13.85, но она незначительна.

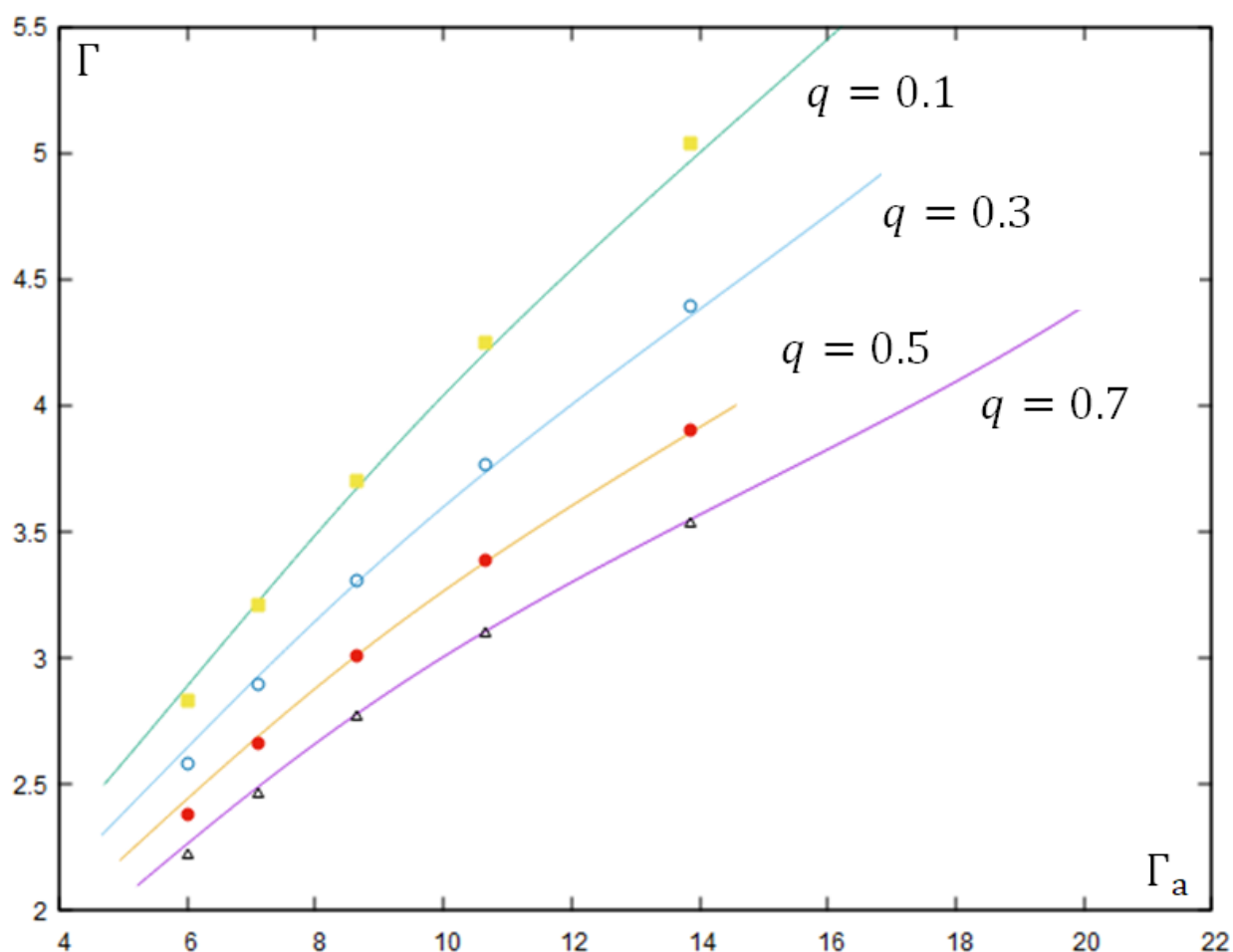


Рис. 3.11 Значения по оси абсцисс Γ_a , ординат Γ при фиксированных $\tau_a = 0.3$ и $Z_0 = 65$ и различных значениях $q = (0.1, 0.3, 0.5, 0.7)$ и $\lambda = 0.675$ мкм.

Был произведен анализ, путем подсчитывания средних квадратичных отклонений и максимальных отклонений для фиксированных значений зенитного угла Солнца Z_0 и оптической толщи τ_a . Результаты вынесены в таблицу.

Для значений $\tau_a = 0.05$ и $Z_0 = 67.5$ погрешности получились наихудшими, поэтому пользоваться данным методом аппроксимации для расчета Γ_a при таких параметрах не рекомендуется. Для значений $\tau_a = 0.15$ и $Z_0 = 72.5$ погрешности получились наименьшими. Это наиболее благоприятные условия параметров для расчета Γ_a .

Отклонения реализованной НС для $\lambda = 0.675$ мкм

Z_0	τ_a	Ср. кв. отклонение	Макс. отклонение
65	0.05	0.068	0.582
	0.1	0.129	0.776
	0.15	0.015	0.327
	0.2	0.069	0.649
	0.25	0.059	0.558
	0.3	0.018	0.38
67.5	0.05	0.122	0.917
	0.1	0.008	0.223
	0.15	0.031	0.518
	0.2	0.084	0.697
	0.25	0.007	0.278
	0.3	0.052	0.557
70	0.05	0.076	0.711
	0.1	0.055	0.489
	0.15	0.015	0.334
	0.2	0.013	0.241
	0.25	0.018	0.332
	0.3	0.007	0.206
72.5	0.05	0.019	0.384
	0.1	0.051	0.612
	0.15	0.006	0.192
	0.2	0.008	0.215
	0.25	0.032	0.552
	0.3	0.032	0.533
75	0.05	0.021	0.318
	0.1	0.032	0.538
	0.15	0.034	0.55
	0.2	0.024	0.399
	0.25	0.022	0.295
	0.3	0.054	0.657

ЗАКЛЮЧЕНИЕ

Таким образом, в ходе выполнения выпускной квалификационной работы были изучены различные архитектуры нейронных сетей для решения задачи аналитической аппроксимации. В результате была выбрана нейронная сеть Румельхарта. Для неё были подобраны наиболее совершенные методы обучения, реализован класс на языке C++ с использованием шаблонов UBLAS из собрания библиотек Boost. Были разработаны программы для обучения и пользования нейронной сети в форме оконного приложения и скрипта.

Нейронная сеть обучалась на данных расчетов коэффициентов асимметрии рассеянного светового потока для длины волны в 0.675 мкм. В процессе обучения нейронной сети была достигнута средняя квадратичная погрешность в узлах интерполяций в 0.02 и максимальное отклонение по модулю в 0.917. Были подобраны наиболее оптимальный зенитный угол Солнца $Z_0 = 72.5$ и оптическая толщина $\tau_0 = 0.15$, при которых метод работает наиболее точно в узлах интерполяций. Были показаны регрессионные свойства нейронной сети, где наблюдались тенденции восстановления регрессии в экстраполяции.

Для аппроксимации данных для других длин волн можно воспользоваться обучающей программой, рассмотренной в третьей главе, для создания её конфигурационного файла.

Рассмотренный метод решения обратной задачи оптики может внедряться в системы мониторинга за окружающей средой для оперативного расчета коэффициента Γ_a . Поскольку все методы реализованы преимущественно матричными операциями, можно перенести их выполнение на вычислительные приборы, архитектуры которых предназначены аппаратно для решения задач векторной алгебры, и ускорить процесс расчетов многократно.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Пашнев В.В., Павлов В.Е., Матющенко Ю.Я., Белозерских В.В. Таблицы для разработки методик определения коэффициентов асимметрии световых потоков, рассеянных на аэрозолях в красной и ближней ИК областях спектра // Южно-Сибирский научный вестник. – 2019. – №2 (26). – С. 204-212.
2. В.В. Пашнев, Ю.Я. Матющенко, В.В. Белозерских, А.В. Калачев программа расчета коэффициентов асимметрии аэрозольного рассеяния в красной и ближней ИК областях спектра // Южно-Сибирский научный вестник. – 2019. – №2 (36). – С. 263-267.
3. Тимофеев Ю.М., Поляков А.В. Математические аспекты решения обратных задач атмосферной оптики: Учеб. пособие. — СПб.: Изд-во С.-Петербург. ун-та, 2001. — 188 с.
4. Нейроинформационные технологии : учебное пособие / А.А. Шайдуров. – Барнаул : Изд-во Алт. Ун-та, 2014 – 138 с.
5. Пашнев В.В., Павлов В.Е., Орлов С.С., Матющенко Ю.Я. Факторы, определяющие наблюдаемые значения коэффициентов асимметрии световых потоков в ближней ИК области спектра // Оптика атмосферы и океана. – 2018. – Т. 31, № 5. – С. 385-390.
6. V.V. Pashnev, V.E. Pavlov, Yu.Ya. Matyuschenko. Numerical methodology for the assessment of asym-metry coefficient of aerosol scattering in near IR region of spectrum // Journal of Physics: Conference Series. – 2020. – Vol. 1615, article: 012017
7. Лившиц Г.Ш. Рассеяние света в атмосфере. – Алма-Ата.: Наука, 1968. – 177 с.
8. Розенберг Г.В., Горчаков Г.И., Георгиевский Ю.С., Любовцева Ю.С. Оптические параметры атмосферного аэрозоля. В книге «Физика атмосферы и проблемы климата». – М.: Наука, 1980. – С. 216-257.

9. Щетинин Е.Ю Математические методы машинного обучения: учеб. пособие / Е.Ю.Щетинин. – М.: ФГБОУ ВО «МГТУ «СТАНКИН», 2016. – 168 с.: ил. 22.
10. Основы нейроинформатики: учеб. Пособие / О.П. Солдатова – Самара: Изд-во Самар. Гос. Аэрокосм. Ун-та, 2006. – 132 с. : ил.
11. Гудфеллоу Я., Бенджио И., Курвилль А. Глубокое обучение / пер. с англ. А. А. Слинкина. – 2-е изд., испр. – М.: ДМК Пресс, 2018. – 652 с.: цв. ил.
12. А.Н. Горбань Обобщенная аппроксимационная теорема и вычислительные возможности нейронных сетей //Сибирский журнал вычислительной математики, 1998. Т.1, № 1. С. 12-24
13. Интернет сайт “www.boost.org” [Электронный ресурс] / режим доступа: www.boost.org/doc/libs/1_65_1/libs/numeric/ublas/doc/index.html, свободный. – Загл. С экрана – Яз. Англ.
14. Иоффе, Сергей Сегеди, Кристиан (2015). «Пакетная нормализация: ускорение глубокого обучения сети за счет уменьшения внутреннего ковариантного сдвига», ICML'15: Материалы 32-й Международной конференции по международной конференции по машинному обучению - Том 37, июль 2015 г. Стр. 448–456
15. Осовский С. Нейронные сети для обработки информации / Пер. с польского И.Д. Рудинского. – М.: Финансы и статистика, 2002. – 344 с.: ил.
16. Хайкин Саймон. Нейронные сети: полный курс, 2-е издание. : Пер. с англ. – М : Издательский дом “Вильямс”, 2006. – 1104 с. : ил. – Парал. Тит. Англ.
17. Интернет сайт “docs.microsoft.com” [Электронный ресурс] / режим доступа: docs.microsoft.com/ru-ru/dotnet/standard/clr, свободный – Загл. С экрана – яз Англ.

ПРИЛОЖЕНИЕ 1

Листинг кода скрипта на языке C++

```

#include "neuron.h"
#include <iostream>
#include <time.h>
#include <fstream>
#pragma warning (disable : 4996)
using namespace boost::numeric::ublas;
namespace ublas = boost::numeric::ublas;
int main(int argc, char* argv[])
{
    setlocale(LC_ALL, "rus");
    FILE* file_input = fopen("input_params.csv", "r");
    std::vector<std::vector<double>> vector_of_vectors, vv;
    std::vector<double> input_str;
    double buf = 0;
    while (feof(file_input) == 0)
    {
        for (int i = 0; i < 3; i++)
        {
            fscanf(file_input, "%lf;", &buf);
            input_str.push_back(buf);
        }
        fscanf(file_input, "%lf\n", &buf);
        input_str.push_back(buf);
        vector_of_vectors.push_back(input_str);
        input_str.clear();
    }
    fclose(file_input);
    matrix<double> X(4, vector_of_vectors.size()), Y(1, vector_of_vectors.size());
    for (int i = 0; i < vector_of_vectors.size(); i++)
    {
        for (int j = 0; j < 4; j++)
        {
            X(j, i) = vector_of_vectors[i][j];
        }
    }
    NN::web web;
    size_t size[] = { 21, 17, 5, 1 };
    web.InicWeb(softplus, X, Y, size, 4);
    web.ReadWFromFile("w_file.txt");
    web.NormTest();
    web.GoForward();
    file_input = fopen("output_params.csv", "w");
    for (int i = 0; i < web.X.size2(); i++)
    {
        for (int j = 0; j < web.X.size1(); j++)
        {
            fprintf(file_input, "%1.3lf;", web.X(j, i) * web.sigma_kv(j) +
web.M(j));
        }
        fprintf(file_input, "%1.3lf\n", web.Y_[web.Y_.size() - 1](0, i) * 14.5);
    }
    fclose(file_input);
    return 0;
}

```

ПРИЛОЖЕНИЕ 2

Листинг кода объявления класса нейронной сети на языке C++

```

#pragma once
#pragma once
#define NDEBUG 1
#include <boost/numeric/ublas/matrix.hpp>
#include <boost/numeric/ublas/vector.hpp>
#include <boost/numeric/ublas/matrix_proxy.hpp>
#include <time.h>
#include <vector>
#include <algorithm>
#pragma warning (disable : 4996)
double sig(double x);
double lin(double x);
double tang(double x);
double softplus(double x);
double ReLu(double x);
namespace NN
{
    using namespace boost::numeric::ublas;
    class web
    {
    public:
        long int epoch = 0;
        double (*ActivationFunc)(double);
        vector<matrix<double>> W; //
        matrix<double> X; //
        matrix<double> Y; //
        std::vector<matrix<double>> Y_;//
        std::vector<matrix<double>> sum;//
        vector<vector<double>> b;//
        vector<double> M;
        vector<double> sigma_kv;
        double err;
        double learning_grate = 0.01;
        double normaY = 1;
        std::vector<matrix<double>> V;
        std::vector<vector<double>> V_smesh;//
        double alpha = 0.01;
        double beta = 0.8;
        double min_err = 999999;
        void InicWeb(double (*output_func)(double), matrix<double> x,
matrix<double> y, size_t* hide, size_t hide_count);
        void SetNewX(matrix<double> X, matrix<double> Y);
        void GoForward();
        void DetErr();
        void GoBack();
        void NormBatch();
        void NormTest();
        void ErrWrite(FILE* outputfile);
        void SaveWInFile(const char*);
        void ReadWFromFile(const char*);
        void NormY();
    };
}

```

ПРИЛОЖЕНИЕ 3

Листинг кода инициализаций методов нейронной сети на языке C++

```

#include "neuron.h"
double sig(double x)
{
    return 1 / (1 + exp(-x));
}
double tang(double x)
{
    return (exp(x) - exp(-x)) / (exp(x) + exp(-x));
}
double softplus(double x)
{
    return log(1 + exp(x));
}
double lin(double x)
{
    return x;
}
double ReLu(double x)
{
    return (x > 0) ? x : 0;
}

void NN::web::InicWeb(double (*output_func)(double), matrix<double> x, matrix<double> y,
size_t* hide, size_t hide_count)
{
    srand(int(time(NULL)));
    this->ActivationFunc = output_func;
    this->X = x;
    this->Y = y;
    this->W.resize(hide_count + 1);
    this->V.resize(hide_count + 1);
    this->b.resize(hide_count + 1);
    this->V_smesh.resize(hide_count + 1);
    this->Y_.resize(hide_count + 1);
    this->sum.resize(hide_count + 1);
    //
    this->W[0].resize(hide[0], this->X.size1());
    this->V[0].resize(hide[0], this->X.size1());
    this->b[0].resize(hide[0]);
    this->V_smesh[0].resize(hide[0]);
    this->sum[0].resize(W[0].size1(), this->X.size2());
    this->Y_[0].resize(this->sum[0].size1(), this->sum[0].size2());
    for (long int i = 1; i < hide_count; i++)
    {
        this->W[i].resize(hide[i], this->Y_[i - 1].size1());
        this->V[i].resize(hide[i], this->Y_[i - 1].size1());
        this->b[i].resize(hide[i]);
        this->V_smesh[i].resize(hide[i]);
        this->sum[i].resize(W[i].size1(), this->Y_[i - 1].size2());
        this->Y_[i].resize(this->sum[i].size1(), this->sum[i].size2());
    }
    this->W[hide_count].resize(Y.size1(), this->Y_[hide_count - 1].size1());
    this->V[hide_count].resize(Y.size1(), this->Y_[hide_count - 1].size1());
    this->b[hide_count].resize(Y.size1());
    this->V_smesh[hide_count].resize(Y.size1());
}

```

```

        this->sum[hide_count].resize(this->Y.size1(), this->Y.size2());
        this->Y_[hide_count].resize(this->sum[hide_count].size1(), this->sum[hide_count].size2());
        for (long int i = 0; i <= hide_count; i++)
        {
            double* W_it = &W[i](0, 0);
            double* V_it = &V[i](0, 0);
            for (long int it = 0; it < W[i].size1() * W[i].size2(); it++)
            {
                *(W_it + it) = -(((double)1 / (2 * (double)W[i].size2())) +
                ((double)1 / (W[i].size2())) * ((double)rand() / RAND_MAX)); //1/2k 1/(2 * W[i].size(k))
                *(V_it + it) = 0;
            }
            for (long int j = 0; j < b[i].size(); j++)
            {
                b[i][j] = -0.01 * ((double)1 / (2 * (double)W[i].size2())) +
                ((double)1 / (W[i].size2())) * ((double)rand() / RAND_MAX);
                V_smesh[i][j] = 0;
            }
        }
    }
void NN::web::GoForward()
{
    this->sum[0] = prod(W[0], this->X);
    for (long int i = 0; i < sum[0].size1(); i++)
    {
        for (long int j = 0; j < sum[0].size2(); j++)
        {
            this->sum[0](i, j) = this->sum[0](i, j) + b[0][i];
            Y_[0](i, j) = this->ActivationFunc(this->sum[0](i, j));
        }
    }
    long int i;
    for (i = 1; i < sum.size() - 1; i++)
    {
        this->sum[i] = prod(W[i], this->Y_[i - 1]);
        for (long int j = 0; j < Y_[i].size1(); j++)
        {
            for (long int k = 0; k < Y_[i].size2(); k++)
            {
                this->sum[i](j, k) = this->sum[i](j, k) + b[i][j];
                Y_[i](j, k) = this->ActivationFunc(this->sum[i](j, k));
            }
        }
        this->sum[i] = prod(W[i], this->Y_[i - 1]);
        Y_[i] = sum[i];
    }
}
void NN::web::DetErr()
{
    double error = 0;
    for (long int i = 0; i < Y.size1(); i++)
    {
        for (long int j = 0; j < Y.size2(); j++)
        {
            error += pow(this->Y(i, j) * normaY - Y_[Y_.size() - 1](i, j) *
normaY, 2);
        }
    }
    this->err = 0.5 * error / X.size2();
}
void NN::web::GoBack()
{
    matrix<double> gradient_J(this->Y.size1(), this->Y.size2());

```



```

matrix<double> grad_Sum(Y.size1(), Y.size2());
std::vector<matrix<double>> delta(Y_.size());
for (long int i = 0; i < Y_[Y_.size() - 1].size1(); i++)
{
    for (long int j = 0; j < Y_[Y_.size() - 1].size2(); j++)
    {
        gradient_J(i, j) = (Y_[Y_.size() - 1](i, j) * normaY - Y(i, j) *
normaY) / X.size2();
        grad_Sum(i, j) = 1;
    }
}
delta[delta.size() - 1] = element_prod(gradient_J, grad_Sum);
for (long int i = delta.size() - 2; i >= 0; i--)
{
    grad_Sum.resize(Y_[i].size1(), Y_[i].size2());
    for (long int j = 0; j < sum[i].size1(); j++)
    {
        for (long int k = 0; k < sum[i].size2(); k++)
        {
            grad_Sum(j, k) = 1/(1 + exp(-this->sum[i](j, k)));
        }
    }
    delta[i] = element_prod(prod(trans(W[i + 1]), delta[i + 1]), grad_Sum);
}
for (long int i = 0; i < delta.size(); i++)
{
    for (long int j = 0; j < b[i].size(); j++)
    {
        for (long int k = 0; k < delta[i].size2(); k++)
        {
            b[i][j] = b[i][j] + alpha * V_smesh[i][j];
            V_smesh[i][j] = V_smesh[i][j] * beta - delta[i](j, k);
        }
    }
    W[0] = W[0] + alpha * V[0];
    V[0] = V[0] * beta - prod(delta[0], trans(X));
    for (long int i = 1; i < W.size(); i++)
    {
        W[i] = W[i] + alpha * V[i];
        V[i] = V[i] * beta - prod(delta[i], trans(Y_[i - 1]));
    }
    this->epoch++;
}
void NN::web::SetNewX(matrix<double> x, matrix<double> y)
{
    this->X = x;
    this->Y = y;
    this->sum[0].resize(W[0].size1(), this->X.size2());
    this->Y_[0].resize(this->sum[0].size1(), this->sum[0].size2());
    for (long int i = 1; i < Y_.size() - 1; i++)
    {
        this->sum[i].resize(W[i].size1(), this->Y_[i - 1].size2());
        this->Y_[i].resize(this->sum[i].size1(), this->sum[i].size2());
    }
    this->sum[sum.size() - 1].resize(this->Y.size1(), this->Y.size2());
    this->Y_[Y_.size() - 1].resize(this->sum[sum.size() - 1].size1(), this-
>sum[sum.size() - 1].size2());
}
void NN::web::NormBatch()
{
    M.resize(this->X.size1(), 1);
    sigma_kv.resize(this->X.size1(), 1);

```

```

    for (long int i = 0; i < this->X.size2(); i++)
    {
        matrix_column<matrix<double>> x_i(X, i);
        M = M + x_i;
    }
    M = M / X.size2();
    for (long int i = 0; i < this->X.size2(); i++)
    {
        matrix_column<matrix<double>> x_i(X, i);
        sigma_kv = sigma_kv + element_prod((x_i - M), (x_i - M));
    }
    sigma_kv = sigma_kv / (this->X.size2() - 1);
    for (long int i = 0; i < sigma_kv.size(); i++)
    {
        sigma_kv[i] = sqrt(sigma_kv[i]);
    }
    for (long int i = 0; i < this->X.size2(); i++)
    {
        matrix_column<matrix<double>> x_i(X, i);
        x_i = x_i - M;
        for (long int j = 0; j < x_i.size(); j++)
        {
            x_i[j] = x_i[j] / sigma_kv[j];
        }
    }
}

void NN::web::NormTest()
{
    for (long int i = 0; i < this->X.size2(); i++)
    {
        matrix_column<matrix<double>> x_i(X, i);
        x_i = x_i - M;
        for (long int j = 0; j < x_i.size(); j++)
        {
            x_i[j] = x_i[j] / sigma_kv[j];
        }
    }
}

void NN::web::ErrWrite(FILE* fileout)
{
    fprintf(fileout, "%d\t%1.15lf\n", this->epoch, this->err);
}

void NN::web::SaveWInFile(const char* title)
{
    FILE* w_file = fopen(title, "w");
    fprintf(w_file, "%d\n", b.size());
    for (long int i = 0; i < b.size(); i++)
    {
        fprintf(w_file, "%d\n", b[i].size());
        for (long int j = 0; j < b[i].size(); j++)
        {
            fprintf(w_file, "%1.15lf\t", b[i][j]);
        }
        fprintf(w_file, "\n");
    }
    for (long int i = 0; i < W.size(); i++)
    {
        fprintf(w_file, "%d\t%d\n", W[i].size1(), W[i].size2());
        for (long int j = 0; j < W[i].size1(); j++)
        {
            for (long int z = 0; z < W[i].size2(); z++)
            {
                fprintf(w_file, "%1.15lf\t", W[i](j, z));
            }
        }
    }
}

```

```

        }
        fprintf(w_file, "\n");
    }
}
for (int i = 0; i < sigma_kv.size(); i++)
{
    fprintf(w_file, "%1.15lf\t", sigma_kv[i]);
}
fprintf(w_file, "\n");
for (int i = 0; i < M.size(); i++)
{
    fprintf(w_file, "%1.15lf\t", M[i]);
}
fclose(w_file);
}
void NN::web::ReadWFromFile(const char* title)
{
    FILE* w_file = fopen(title, "r");
    long int hide_count = 0, buff = 0, buff1 = 0;
    fscanf(w_file, "%d\n", &hide_count);
    b.resize(hide_count);
    for (long int i = 0; i < b.size(); i++)
    {
        fscanf(w_file, "%d\n", &buff);
        b[i].resize(buff);
        for (long int j = 0; j < b[i].size(); j++)
        {
            fscanf(w_file, "%lf\t", &b[i][j]);
        }
        fscanf(w_file, "\n");
    }
    W.resize(hide_count);
    for (long int i = 0; i < W.size(); i++)
    {
        fscanf(w_file, "%d\t%d\n", &buff, &buff1);
        W[i].resize(buff, buff1);
        for (long int j = 0; j < W[i].size1(); j++)
        {
            for (long int z = 0; z < W[i].size2(); z++)
            {
                fscanf(w_file, "%lf\t", &W[i](j, z));
            }
            fscanf(w_file, "\n");
        }
    }
    M.resize(this->X.size1(), 1);
    sigma_kv.resize(this->X.size1(), 1);
    for (int i = 0; i < sigma_kv.size(); i++)
    {
        fscanf(w_file, "%lf\t", &sigma_kv[i]);
    }
    fscanf(w_file, "\n");
    for (int i = 0; i < M.size(); i++)
    {
        fscanf(w_file, "%lf\t", &M[i]);
    }
    fclose(w_file);
}
void NN::web::NormY()
{
    Y = Y / normaY;
}

```