МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

01.03.02 Прикладная математика и информатика
(код и наименование направления подготовки, специальности)

Компьютерные технологии и математическое моделирование
(направленность (профиль))

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему: «Разработка программного обеспечения для сравнения алгоритмов обучения сверточных нейронных сетей»

Студент О.В. Рязанцева
(И.О. Фамилия) (личная подпись)

Руководитель канд.тех.наук, доцент, В.С. Климов
(ученая степень, звание, И.О. Фамилия)

Консультант старший преподаватель, М.В. Дайнеко
(ученая степень, звание, И.О. Фамилия)

Тольятти 2021

Аннотация

Тема дипломной работы «Разработка программного обеспечения для сравнения алгоритмов обучения сверточных нейронных сетей».

В дипломной работе проводится сравнительный анализ различных алгоритмов обучения сверточных нейронных сетей.

В работе затрагиваются следующие вопросы: анализ проблемы, включающий изучение принципов построения сверточных нейронных сетей и особенностей используемых алгоритмов; описание сравниваемых алгоритмов обучения и выбор критериев для сравнения; разработка проекта и реализация системы для сравнения алгоритмов обучения, проведение эксперимента и описание результатов тестирования и оценка эффективности различных алгоритмов.

Целью данной работы является разработка программного обеспечения для сравнения алгоритмов обучения сверточных нейронных сетей.

Объектом дипломной работы являются сверточные нейронные сети.

Предметом дипломной работы являются алгоритмы обучения сверточных нейронных сетей.

Дипломная работа может быть разделена на следующие логически взаимосвязанные части: анализ проблемы, описание и выбор математических алгоритмов для сравнения, реализация проектного решения.

Полученные результаты свидетельствуют о высокой точности распознавания для данного набора данных, лучшими также являются варианты модифицированных методов Adam - Nadam и Adamax. Тем не менее, различия в полученных данных могут считаться значительными только условно, скорее всего оптимального метода для всех наборов данных не существует, поэтому если по результатам обучения нейронной сети получена ошибка, серьёзным образом влияющая на принятие некоторого решения необходимо использовать несколько методов.

Abstract

The title of the graduation work is «Development of software for comparing of training algorithms related to convolutional neural networks».

The graduation work is devoted to the problem of choosing an effective training algorithm related to convolutional neural networks.

The research touches upon the following issues: analyzing the problem, including the study of the convolutional neural networks construction principles and the features of the algorithms used; describing the compared training algorithms and selecting the appropriate comparison criteria; developing a project and implementing a system for comparing the training algorithms; conducting an experiment; describing the test results and evaluating the effectiveness of various algorithms.

The aim of the investigation is to develop software for comparing of training algorithms related to convolutional neural networks.

The object of the graduation work is the convolutional neural networks.

The subject of the research is the training algorithms for the convolutional neural networks.

The graduation work may be divided into several logically connected parts which are analysis of the problem, description and selection of the mathematical algorithms for comparison, as well as implementation of the design solution.

The results obtained indicate a high recognition accuracy for this data set; the best options are also the versions of the modified methods Adam - Nadam and Adamax methods, as they have the optimal values of the recognition level and the operation speed. Nevertheless, the differences in the obtained data can be considered to be significant only conditionally. We suppose there is no optimal method for all data sets. Therefore, if an error affecting the decision-making process is detected as the result of the training, several methods have to be used.

Содержание

Введение

Серьезный толчок в развитии новых технологий анализа данных произошел благодаря тому, что практически во всех сферах жизни довольно значительным стало применение информационных технологий. Стандартные методы прогнозирования, основанные на математических моделях уже не в полной мере удовлетворяют специалистов при создании адаптивных структур, отражающих процессы, протекающие в реальных открытых системах.

В таких условиях получила сильное развитие теория, основанная на принципах работы человеческого мозга с использованием технологий работы нейрона. Искусственная нейронная сеть получает точные знания на базе уже прошедших процессов и продуцирует новые знания с использованием уже имеющихся. Положительные результаты применения искусственных нейронных сетей в экономике и исследовании социальных процессов способствовали созданию новых, более сложных конструкций, называемых сверточными нейронными сетями.

Анализ сигнала в такой сети происходит в несколько уровней: поступающий на вход нейрона импульс преобразуется до выхода в нескольких скрытых слоях. Оказалось, что такая сложная система более эффективно решает поставленные задачи.

При этом в процессе «работы» такой сети на последнем этапе, при формировании выходного сигнала могут быть использованы различные оптимизационные методы, от которых зависит близость выходного сигнала к реальному результату.

Таким образом, возникает вопрос о необходимости сравнения используемых методов, для выбора наиболее эффективного.

Исследованию данный проблемы посвящено множество работ, которые можно разделить по нескольким направлениям. Наиболее популярным является направление, связанное с применением сверточных нейронных сетей для классификации изображений. Это направление представлено, например, ра-

ботами А.А. Воеводы, Д.О. Романникова [1], Е.А. Гузия, В.В. Федоренко [8], Ле Мань Ха [11], Hou B., Luo X., Wang SS., Jiao L., Zhang X. [21].

Целью данной работы является разработка программного обеспечения для сравнения эффективности различных алгоритмов обучения сверточных нейронных сетей.

Таким образом, достичь поставленной в данной работе цели позволит решение следующих задач:

– анализ проблемы, включающий изучение принципов построения сверточных нейронных сетей и особенностей используемых алгоритмов;

– описание сравниваемых алгоритмов обучения;

– выбор критериев для сравнения алгоритмов обучения сверточных нейронных сетей;

– разработка проекта и реализация системы для сравнения алгоритмов обучения;

– проведение эксперимента и описание результатов тестирования;

– оценка эффективности различных алгоритмов.

Структура работы представлена тремя главами. В первой главе рассматриваются сверточные нейронные сети и принципы их построения, а также отмечаются особенности используемых методов обучения нейронных сетей.

Вторая глава посвящена рассматриваемым алгоритмам обучения, в том числе алгоритму обратного распространения ошибки, исследуются метод Адамса и его модификации, проводится выбор критериев для сравнения алгоритмов обучения.

Третья глава отражает формирование необходимого набора данных, описание используемых библиотек для проработки методов обучения. Реализация проектного решения с выбранными алгоритмами обучения также отмечена в третьей главе, в которой освещены проведение эксперимента и описание результатов тестирования. Результатом третьей главы становится оценка эффективности различных алгоритмов.

1 Анализ проблемы

1.1 Описание нейронных сетей

Впервые модель искусственного нейрона была презентована в 1943 году американским нейрофизиологом Уорреном МакКаллоком и математиком Уолтером Питтсом. В основе данной модели лежит нейрон человеческого мозга. N входных бинарных величин x_1, x_2 … x_n имеются в составе предложенной модели искусственного нейрона. Данные величины здесь принимаются в качестве импульсов, поступающих на вход нейрона (рисунок 1), сложение же импульсов и весов w_1, w_2, … w_n производится внутри нейрона.

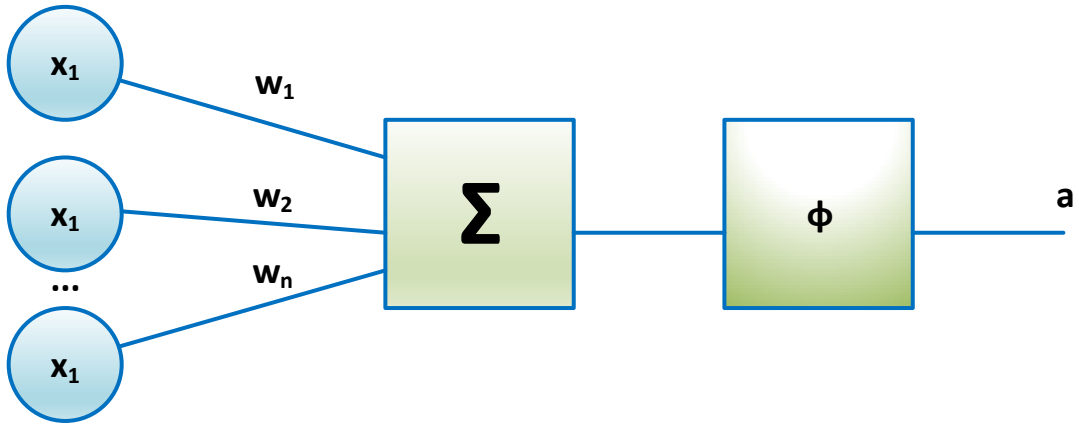

Рисунок 1 – Модель искусственного нейрона МакКаллока-Питтса

Выходной сигнал нейрона определяется выражением (1):

$$a = \varphi(\sum_{i=1}^{n} w_i x_i) \quad (1)$$

В вышеприведенном выражении выполняется, как можно видеть, преобразование при помощи функции φ суммарного импульса в выходное значение нейрона. Функция φ нелинейная, она называется функцией активации. С данной целью в модели искусственного нейрона МакКаллока-Питтса применялась функция Хевисайда. С течением времени, целесообразным оказалось использование функций активации других типов. Среди них наиболее востребованными оказались логистическая сигмоидальная функция ($f(x) = \frac{1}{1+e^{(-x)}}$), радиально-базисная функция и гиперболический тангенс ($tanh(x) = \frac{2}{1+e^{(-2x)}}$). Данные функции активации обеспечили изменение выходного сигнала на более плавном уровне.

Также Уорреном МакКаллоком и Уолтером Питтсом была предложена упрощенная схема нейронной сети. Данная модель показывала, как, возможно, взаимодействуют нейроны внутри головного мозга человека. Таким образом, отдельные нейроны были объединены в искусственные нейронные сети, выходной сигнал одного нейрона подается на вход следующему за ним нейрону и так далее, это показано на рисунке 2. В рассмотренной нейронной сети имеется некоторое количество слоев. В каждом из слоев нейронной сети же содержится один и более нейронов. Входной слой принимает сигналы, поступающие извне, выходной слой выдает сигналы во внешний мир, остальные слои нейронной сети именуют скрытыми.

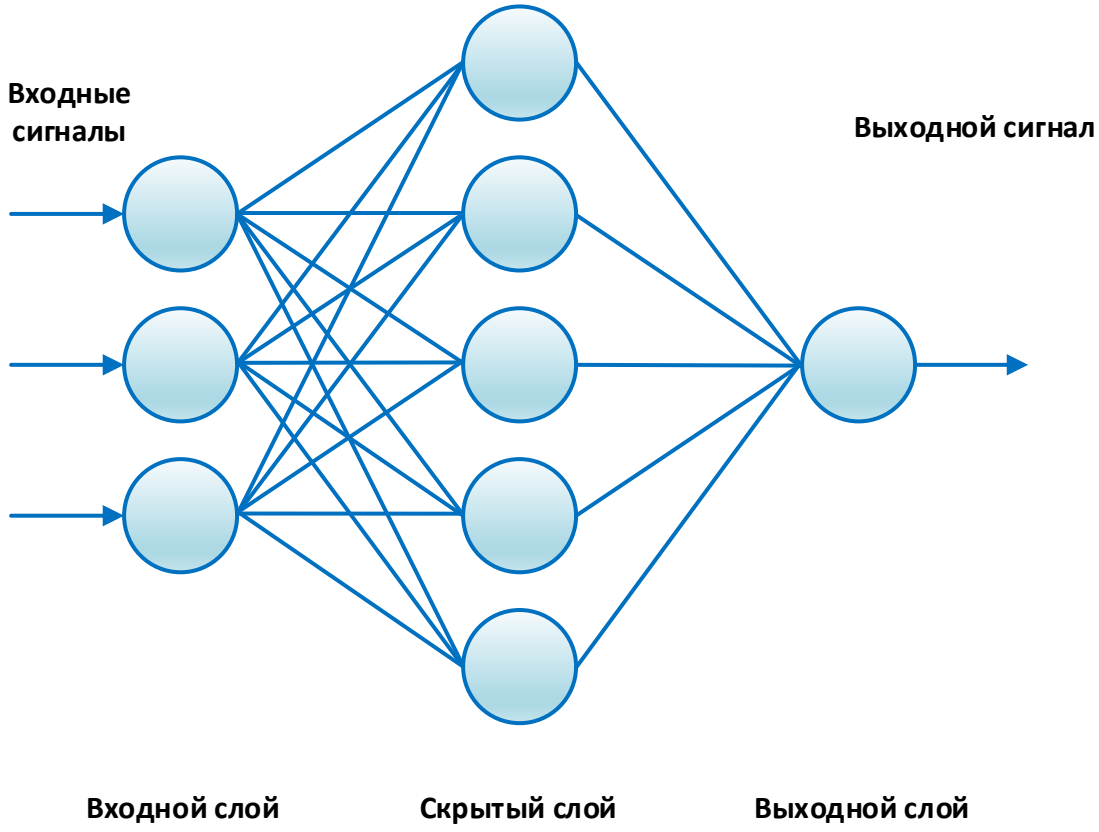

Рисунок 2 – Искусственная нейронная сеть

Все искусственные нейронные сети подразделяются на сети прямого распространения сигнала (feedforward networks) и рекуррентные сети (recurrent networks). Первые не имеют циклов, во вторых циклы имеются.

На данный момент однозначного определения глубокой нейронной сети пока не существует, однако довольно часто под этим определением пони-

мают искусственную нейронную сеть, содержащую один и более скрытых слоев.

Значительный недостаток данных нейронных сетей – это невозможность гарантированного создания нейронных сетей, обладающих заранее определенными свойствами ввиду отсутствия конструктивного подхода, хотя искусственные нейронные сети, определенные подобным образом могут приближать любую непрерывную функцию с любой заданной точностью.

Обучение искусственной нейронной сети – определение весов между нейронами таким образом, чтобы сеть имела возможность приблизить заданную функцию с обозначенной точностью.

Методы обучения искусственных нейронных сетей можно разделить на три категории: «обучение с учителем» (или «supervised learning»), «обучение без учителя» (или «unsupervised learning») а также «обучение с подкреплением» (или «reinforcement learning»).

Методы, относящиеся к категории «обучение с учителем», подают на вход нейронной сети совокупность сигналов (или объектов), для которых правильный ответ (обучающее множество) уже известно заранее. Здесь существуют определенные правила изменения весов, также изменение весов зависит от того, насколько правильный сигнал получился на выходе.

Категория методов «обучение без учителя» подразумевает подачу на вход нейронной сети сигналов, для которых заранее не известен корректный ответ.

Для методов, относящихся к категории «обучение с подкреплением», нужна внешняя среда, с которой нейронная сеть взаимодействует. Таким образом, нейронная сеть обучается на поступающей из внешней среды совокупности объектов.

Рассматривая категории методов обучения искусственных сетей нужно упомянуть, что в нейронных сетях МакКаллока-Питтса обучение не предусматривалось, то есть веса на входах нейронов не изменялись, соответственно были заданы заранее.

Идею обучения искусственных нейронных сетей предложил Дональд Хэбб в 1949 году. Выдвинутая им идея говорила о том, что связь между нейронами, срабатывающими совместно должна возрастать, и наоборот, связь между нейронами, которые активируются независимо, должна уменьшаться (снижаться). Также правила, по которым происходят изменения весов входных сигналов искусственных нейронов в соответствии с тем, верный или неверный был получен ответ от сети (то есть «обучение с учителем»), были составлены Дональдом Хэббом. В дальнейшем были сформулированы и другие правила изменения весов для методов обучения нейронных сетей как «с учителем», так и без него.

В настоящее время получил широкое распространение метод обратного распространения ошибки (англ. backpropagation), особенно данный алгоритм актуален для глубоких нейронных сетей. Данный алгоритм появился достаточно давно в 1970 году, однако он никак не был связан с искусственными нейронными сетями, лишь в 1981 году началось использование данного метода для обучения нейронных сетей. Основа алгоритма – метод градиентного спуска. Метод обратного распространения ошибки относится к категории методов «обучения с учителем. Также в данном методе есть мера ошибки для оценки степени расхождения между правильными ответами и выдаваемыми сетью значениями. Метод градиентого спуска, указанный ранее, нужен для минимизации меры ошибки путем изменения значений весов в сети, начальные значения весов задаются случайно. По каждому из весов нейронной сети выполняется расчет частных производных ошибки, это необходимо сделать для определения уровня важности влияния каждого веса на выходное значение, выдаваемое сетью. Изменение весов происходит до тех пор, пока значение ошибки на выходе не попадет в нужный интервал.

Расчет ошибки для глубокой нейронной сети с несколькими скрытыми слоями происходит с передачей ее от каждого предыдущего слоя к следующему.

Сначала происходит расчет ошибки на выходе искусственной нейронной сети, для которого корректные ответы уже заранее известны. Затем вычисляется ошибка на входе в выходной слой, называемая ошибкой выхода скрытого слоя. Далее вычисления происходят до тех пор, пока ошибка на входном слое не станет известна. Описание последовательности действий алгоритма говорит о том, что название метод обратного распространения ошибки получил исходя из вышесказанного.

Альтернативным методом является алгоритм стохастического градиентного спуска. Данный метод характеризуется изменением весов в процессе обработки одного из элементов обучающего множества или нескольких элементов обучающего множества.

В настоящее время метод градиентного спуска, а также его модификации является наиболее эффективным алгоритмом для обучения искусственных нейронных сетей.

1.2 Архитектура сверточных нейронных сетей

Сверточная нейронная сеть — это нейронная сеть особой архитектуры, которая довольно обширно используется для результативного распознавания образов и изображений. Данную архитектуру предложил Ян Лекун. Действие данной нейронной сети состоит в том, чтобы от каких-либо конкретных деталей изображения переходить к его более и более абстрактным особенностям. Игнорируя различные недостаточно важные элементы изображения и обращая внимание на более значимые, сверточная нейронная сеть может, основываясь на уже пройденном опыте, самонастраиваться и создавать собственную последовательность карт признаков.

В полносвязной сети, каковым является перцептрон, все нейроны одного слоя связаны со всеми нейронами следующего слоя. У каждой связи между нейронами такой сети существует собственный вес. При свертке здесь происходит перемещение ограниченной матрицы весовых коэффициентов по слою, который подвергается обработке. Далее, когда очередной сдвиг матри-

цы был осуществлен, создается сигнал активации для нейрона на следующем для обработки слое, имеющего аналогичную позицию. Такую матрицу весовых коэффициентов еще называют ядром свертки. Одно и то же ядро свертки применяется для различных нейронов выходного слоя. Таким образом, в процессе свертки с использованием матрицы весовых коэффициентов получается следующий слой, по которому видно присутствие данного признака в слое, над которым проводится обработка, и координаты признака. Так формируется карта признаков. Модель сверточной сети показана на рисунке 3.

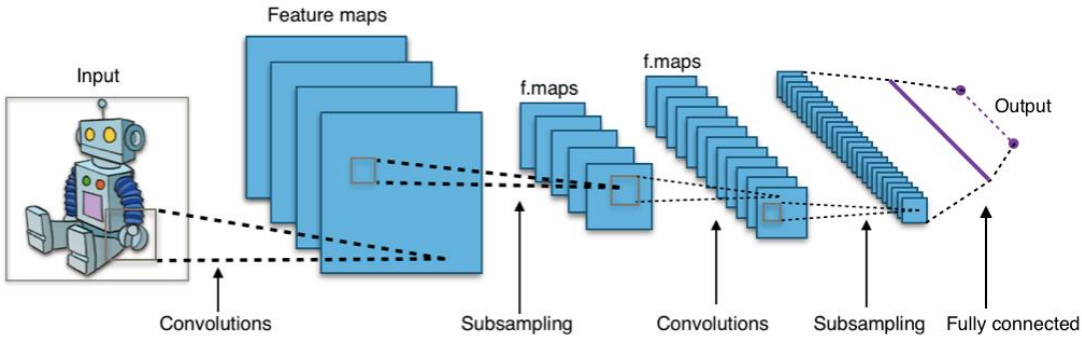

Рисунок 3 - Архитектура свёрточной нейронной сети

Сверточная сеть, конечно, имеет не единственный набор весовых коэффициентов, а множество. Данные наборы кодирует детали обрабатываемого изображения. Таким образом, ядра свертки не формируются вручную, а уже создаются сетью при ее обучении общепринятым алгоритмом обратного распространения ошибки. Сверточная нейронная сеть становится многоканальной, так как при проходе каждого набора весовых коэффициентов создается новая карта признаков. В процессе обработки слоя ядром свертки его перемещают на совсем небольшое расстояние, а не на размерность ядра, так как можно пропустить нужный признак.

Размерность созданных карт признаков уменьшается при процессе субдескритизации (или операции подвыборки, пулинг). Из некоторого количества соседних нейронов карты признаков производится определение максимального, он и становится одним из нейронов уплотненной карты признаков, которая имеет меньший размер. Все это происходит именно потому, что в рассматриваемой архитектуре информация о том, что нужный признак суще-

ствует на данной карте является более весомой, чем точное знание его координат. Благодаря этой операции не только происходит ускорение последующих расчетов, но и так же за счет этого нейронная сеть становится инвариантной к масштабу изображения, подаваемого ей на вход.

Приведем более подробное описание архитектуры сверточной нейронной сети. Внутри сети есть множество слоев. Сигнал сначала проходит через входное изображение (собственно, это начальный слой), затем проходит некоторое количество сверточных слоев (здесь происходит чередование операций свертки и пулинга). В процессе чередования слоев создаются новые карты признаков из карт признаков, причем на последующих слоях происходит уменьшение размерности карты, а количество каналов напротив увеличивается. Это значит, что теперь сеть может распознавать сложные иерархии признаков. Карта признаков обычно преобразуется в вектор или скаляр при прохождении через некоторое количество слоев, появляется множество карт признаков. Обычно несколько слоев полносвязной нейронной сети ставят на выходе сверточных слоев. На вход такой сети подаются карты признаков, которые получились в итоге.

Основная часть сверточной нейронной сети – это слой свертки, который имеет для каждого канала свою матрицу весовых коэффициентов, производящую обработку предыдущего слоя по частям и свой фильтр. Как было сказано ранее, весовые коэффициенты в ядре свертки определяются в течение обучения сети. Также важная деталь при рассмотрении сверточного слоя – это то, что при обучении устанавливается не очень большое количество параметров. На рисунке 4 представлена модель слоя свертки.

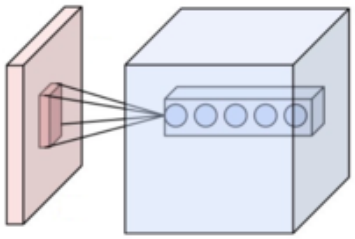

Рисунок 4 - Нейроны слоя свёртки, преобразуемые по нескольким выходным каналам

После каждой операции свертки ее скалярный результат попадает на так называемую функцию активации. Данная функция является нелинейной. Обычно слой активации не выделяют отдельно, считается, что данная функция уже имеется внутри слоя свертки. Обычно в качестве функции активации выбирают такие, как гиперболический тангенс ($tanh(x) = \frac{2}{1+e^{(-2x)}}$) или сигмоид ($f(x) = \frac{1}{1+e^{(-x)}}$). Далее была также определена новая функция активации под названием ReLU. Данная функция не только помогла упростить вычисления, но и ускорить процесс самого обучения нейронной сети. В настоящее время данная функция все еще остается популярной, особенно для обучения сверточных нейронных сетей.

Далее расположен слой пулинга. Данный слой – это нелинейное уплотнение карты признаков, при прохождении нелинейного преобразования некоторое количество пикселей (размер 2*2 считается оптимальным) уплотняется до одного пикселя, функция максимума в данном случае является наиболее употребляемой. Над квадратами или прямоугольниками (являтся непересекающимися) проводятся преобразования. Каждый из преобразуемых прямоугольников сжимается до одного пикселя соответственно, выбирается пиксель, который имеет максимальное значение. Таким образом, данный процесс помогает произвести сокращение пространственного объема изображения. Обычно слой пулинга располагают между слоями свертки. На рисунке 5 показана операция пулинга.

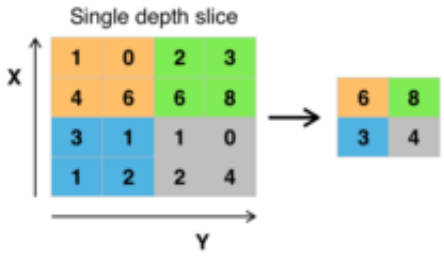

Рисунок 5 – Операция пулинга с функцией максимума

После этого происходит перестройка системы от сетки пикселей, имеющей первоначальное разрешение к абстрактным картам признаков, это достигается путем некоторого количества прохождений операций свертки и пу-

линга. На каждом из последующих слоев количество каналов становится больше, а размерность изображения в каждом из каналов сокращается. Таким образом, получается множество каналов, в которых находится небольшое количество данный – это самые абстрактные особенности, которые были получены из обрабатываемого изображения.

Все это получает на вход полносвязная сеть (она может также включать в себя некоторое количество слоев). Полносвязные слои уже не имеют пространственную структуру пикселей, их размерность становится намного меньше, если сравнивать с числом пикселей исходного изображения.

Сверточную нейронную сеть обычно обучают методов обучения с учителем, данные промаркированы. Таковыми являются метод обратного распространения ошибки и различные модификации данного метода, но также существуют и методы обучения такой сети без учителя.

1.3 Особенности используемых алгоритмов для обучения сверточных нейронных сетей

Появление больших баз изображений сделало возможным обучение глубоких нейронных сетей – сетей, количество скрытых слоев которых велико. В решении задачи распознавания изображений наибольшим успехом пользуется использование сверточных нейронных сетей [7].

Распознавание изображений в настоящее время довольно часто используется в различных областях человеческой деятельности. Поэтому стали необходимы алгоритмы, которые бы были оптимальны конкретно для данного рода работы. Сверточные нейронные сети нацелены как раз на это. Для того, чтобы обучать конволюционные (сверточные) нейронные сети используются алгоритмы глубокого обучения. Механизм работы сверточной нейронной сети имитирует функционирование зрительного отдела коры человеческого головного мозга в процессе обработки изображений. [1].

Существует так называемый гибридный алгоритм. Он применяется для обучения сверточной нейронной сети. В его основе – так называемый метод

обратного распространения ошибки (ОРО) и конволюционный генетический алгоритм оптимизации.

В данном методе (конволюционном генетическом алгоритме) используется принцип применения нескольких популяций, где эволюция происходит по отдельности (поэтому алгоритм конволюционный). Как раз в этом и состоит разница между конволюционным генетическим алгоритмом и стандартным (здесь используется одна популяция).

Действия в популяциях производятся с упором на ГА (генетический алгоритм) каждой популяции, отдельный ГА имеет собственные изначально заданные параметры. Каждый ГА нужен для того, чтобы происходили процедуры оптимизации в популяциях. Обмен решениями популяций происходит через какое-либо заданное количество повторений (итераций). Такой обмен еще называют миграцией.

На рисунке 6 изображена конволюционная искусственная нейронная сеть с определенно построенной структурой. С ней и работает вышеизложенный алгоритм. При ручном задании макропараметров нейронной сети учитывались размеры изображений, которые подавались на вход сети. Макропараметрами сети являются масштаб уменьшения изображения (которое подается на вход) на каждом слое субдескритизации, имеющиеся у каждого из конволюционных слоев число, размер ядер свертки и так далее [10]. Таким образом, можно увидеть из рисунка 6, что на вход данной сети подается изображение размера 70*70 пикселей, а на выходе имеется 2 класса, принадлежность изображения к каждому из которых определяет сама сеть.

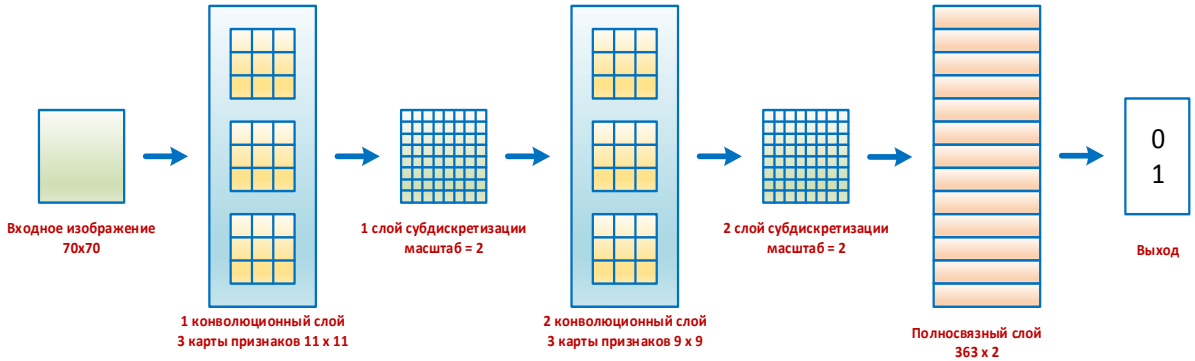

Рисунок 6 – Устройство конволюционной искусственной нейронной сети

Выбор параметров для нейронной сети был осуществлен вручную, так как было нужно провести над изображением подаваемым на входе комплекса операций в процессе того, как изображение обрабатывается сетью. Данные действия были связаны с разбиением изображения на фрагменты и масштабированием. Размерность же полносвязного слоя нейронной сети производилась автоматически, опираясь на размер изображения, обрабатываемого на предыдущем слое.

Но при обработке изображений может произойти ошибка работы нейронной сети. Это может возникнуть из-за обработки сетью изображения, имеющего нецелый пиксельный размер (допустим, 31.5*31.5 пикселей). Поэтому для предотвращения этого нужно подобрать подходящие параметры для нейронной сети. На таблица 1 представлены параметры сетей с алгоритмами обучения, используемыми нами.

Таблица 1 – Параметры сетей для входных изображений, имеющих различные размеры

Параметр	**Размер входных изображений в пикселях**		
	40*40	**50*50**	**70*70**
Количество карт признаков на конволюционных слоях	3	3	3
Размерность ядра свертки 1 конволюционного слоя	7*7	9*9	11*11
Размерность ядра свертки 2 конволюционного слоя	6*6	6*6	9*9
Масштаб слоев субдискретизации	2	2	2
Количество нейронов полносвязного слоя	108	192	363

Два этапа имеется в анализируемом гибридном алгоритме обучения сверточной нейронной сети.

1 этап.

Прозводится модификация весов нейронной сети с использованием ГА оптимизации. На таблица показаны примененные к нейронной сети параметры ГА. Сверточная сеть с заданными весами (набором весов) здесь определена как индивид (в генетическом алгоритме). Размер входных

изображений для сети – 40*40, 50*50, 70*70 пикселей, для них количество входных переменных генетического алгоритма (или весов сети) - 483, 579 и 1104 соотвествуют для данных размеров изображений.

Таблица 2 – Параметры генетического алгоритма

Параметр	**Значение**
Размер популяции	50
Интервал поиска по каждой переменной	[-3; 2]
Количество итераций	50
Процент элитарных решений,%	4
Точность разбиения интервала поиска	0,0001

Предотвращение стагнации и более полный охват пространства поиска достигается именно из-за того, что на первом этапе применяется генетический алгоритм. Генетический алгоритм именно производит поиск области предполагаемого оптиума во всем пространстве поиска. А уже метод градиентного спуска на втором этапе нужен для определения самого оптиума [21].

Очевидно, что выбор оптимизируемого критерия для генетического алгоритма это довольно значимая задача. Проанализировав, что целевой функцией в данном случае определена задача классификации (так как индивид генетического алгоритма (их несколько) – конволюционная нейронная сеть с заданными весами). Поэтому достаточно определенным бы было сделать выбор в пользу максимизируемого критерия (для точности классификации) ну или минимизируемого критерия (для ошибки).

Но из-за того, что возникало множество ситуаций впадения в стагнацию (одинаковости через какое-либо число повторений всех конволюционных сетей в популяции) генетического алгоритма данный вариант выбора пришлось отбросить (результат не оправдал ожиданий). Было принято решение искать другой вариант, и он был найден – усредненная F-мера. Данный оптимизационный критерий предполагает использование следующего выражения (2) (в данном варианте – для 2-х классов).

$$F = 2 \cdot \frac{precision * recall}{precision + recall} \quad (2)$$

Выражения **Ошибка! Источник ссылки не найден.**), **Ошибка! Источник ссылки не найден.**), представленные ниже, нужны для определения precision и recall (точности и полноты соответственно). Здесь:

- TP – true-positive или истинно положительное решение;
- FP – false-positive или ложно положительное решение;
- TN – true-negative или истинно отрицательное решение;
- FN – false-negative или ложно отрицательное решение.

Эти данные есть в таблица 1.

$$Точность = \frac{TP}{TP+FP} \quad (3)$$

$$Полнота = \frac{TN}{TN+FN} \quad (4)$$

Таким образом, было выявлено, что F-мера определенно лучше показывает, способна система классификации к обобщению или нет. Обобщение здесь – классификация объектов, с помощью которых не производится обучение системы. Данная формула **Ошибка! Источник ссылки не найден.**) может также использоваться и если имеется не два класса, а больше. Это можно произвести путем сведения всех имеющихся классов к двум при помощи схемы «один против всех». Если поступить таким образом, то произойдет сначала замена всех классов, кроме первого, на непервый, далее будут заменены все классы на невторой, кроме второго, и так далее для всех имеющихся классов будет произведена данная операция.

Нужно сказать, что на вход каждой сети популяции подается лишь небольшой фрагмент обучающей выборки, а не она вся, это нужно для ускорения работы генетического алгоритма (на этапе вычисления F-меры каждой сети в популяции).

Таблица 1 – Точность и полнота

Категория i		**Экспертная оценка**	
		Положительная	**Отрицательная**
Оценка системы	Положиельная	TP	FP
	Отрицательная	FN	TN

Далее выяснялось, какая из конволюционных нейронных сетей из конечной популяции генетического алгоритма обладает наибольшим значением F-меры. Генетический алгоритм модифицировал веса сети, данные значения и используются как начальное приближение на втором этапе. Конец первого этапа.

Второй этап.

Далее используя метод обратного распространения ошибки (ОРО) происходит обучение сети с наибольшей F-мерой. Данный метод градиентный, он нужен для того, чтобы найти глобальный оптимум в суженной ГА области поиска, донастройке сети. Начальным приближением в алгоритме ОРО являются значения весов конволюционной сети (сгенерированные случайным образом), следовательно ему свойственно застревать в локальных оптимумах. Таким образом, недочет связанный с ОРО решен тем, что генетический алгоритм определил начальное приближение весов сети.

2 Математический алгоритм

2.1 Алгоритм обратного распространения ошибки

Как уже было сказано выше, метод обратного распространения ошибки относится к группе обучения сети с «учителем», другими словами, осуществление процесса обучения производится за счет предоставления сети совокупности обучающих примеров, имеющих правильные отклики.

Задача заключается в том, чтобы нейронная сеть в результате обучения обладала бы неким паритетом в возможности подачи верного отклика на полученные ею данные при обучении. При этом, также необходимо, чтобы нейронная сеть предоставляла и корректные ответы на данные, которые не использовались при обучении сети, но похожи на них. Принцип, который заключается в верной реакции сети на обучающие данные, называется принципом запоминания, а тот, который основан на правильном отклике сети на данные, идентичные обучающим является принципом обобщения.

Использование метода алгоритма обратного распространения ошибки при обучении нейронной сети можно разделить на три этапа:

1) На вход сети подаются данные, далее происходит распространение их в сторону выходов;

2) Ошибка, полученная сетью на выходе, вычисляется и распространяется в обратную сторону;

3) Модификация весовых коэффициентов.

Завершается обучение сети данным методом тем, что на вход сети происходит подача данных и затем их распространение к выходам.

Заметим, что обучение сети данным методом – процесс, требующих достаточно больших временных ресурсов. Однако требует мало времени расчет самих результатов обученной сетью.

Необходимо заметить, что при использовании однослойной нейронной сети есть ограничения на обучение сети определенным шаблонам входных данных, в то время как многослойная сеть таковым недостатком не обладает.

Рассмотрим стандартную нейронную сеть и сам метод обратного распространения ошибки.

На рисунке 7 показана сеть, имеющая один скрытый слой (многослойная сеть), нейроны скрытого слоя обозначены Z, выходного слоя – Y, входного слоя – X.

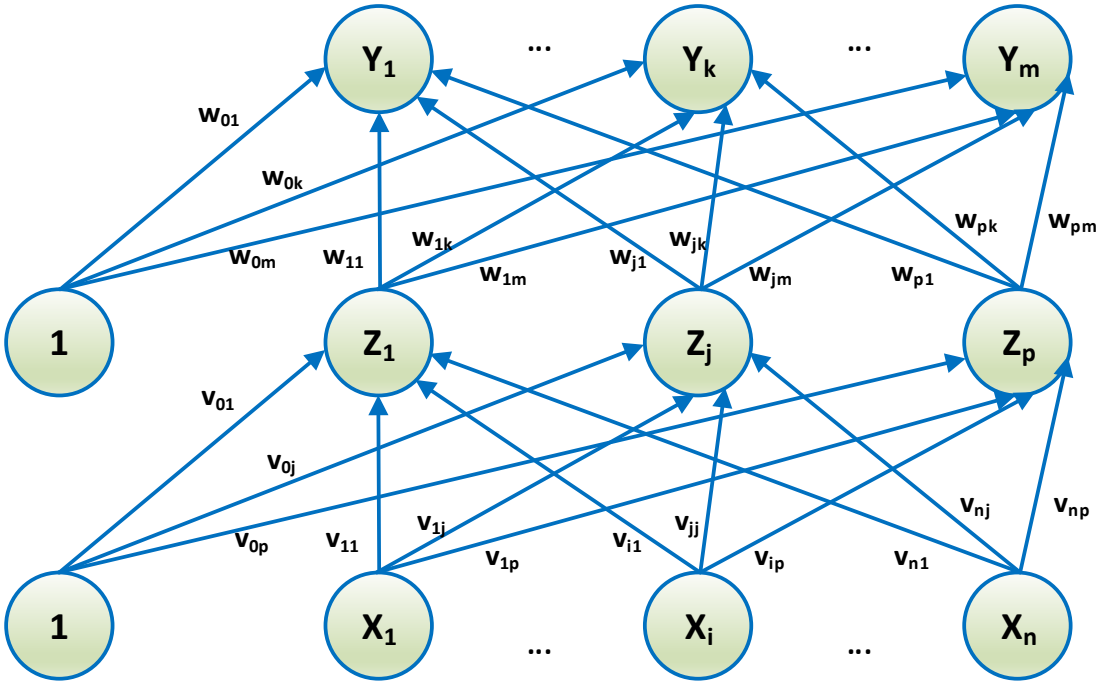

Рисунок 7 – Нейронная сеть, обратное распространение ошибки

Также на рисунке 7 показано, что у нейронов выходного и скрытого слоев может быть смещение. Обозначение w_{0k} показывает смещение выхода Y_k, а обозначение v_{oj} показывает смещение для элемента скрытого слоя Z_j. Указанные смещения нужны, чтобы показать весовые коэффициенты на связях от нейронов, имеющих на выходе 1. Стрелками на рисунке 7 показано распространение полученных сетью данных от входного слоя к выходному, при обучении же происходит перемещение сигналов от выходного слоя к входному соответственно.

Для начала рассмотрим обозначения, которые будут использоваться при описании алгоритма обратного распространения ошибки.

- x – входные обучающие данные, представленные в виде вектора, $x = (x_1, x_2, \dots, x_i, \dots, x_n)$;

- t – корректные выходные значения для конкретных входных данных, также представлены в виде вектора, $t = (t_1, t_2, \dots, t_k, \dots, t_m)$;

- a – скорость обучения сети данным методом;

- X_i – нейроны, составляющие входной слой сети. Каждый нейрон обладает индексом i. Также нужно сказать, что значения сигналов входных нейронов на входе и выходе равны x_i, причем данные величины совпадают;

- v_{0j} - смещение для элемента скрытого слоя Z_j;

- Z_j - нейроны, составляющие скрытый слой сети. Каждый нейрон обладает индексом j;

- z_in_j – значение, подаваемое на вход нейронам, принадлежащим скрытому слою, высчитывается по формуле: $z\_in_j = v_{0j} + \sum_{i=1}^{n} x_i \cdot v_{ij}$;

- σ_j – используется для коррекции весовых коэффициентов между нейронами скрытого и входного слоев;

- z_j – выходной сигнал скрытого слоя нейронной сети (нейронов Z_j), высчитывается по формуле: $z_j = f(z\_in_j)$;

- w_{0k}- смещение для элемента выходного слоя Y_k;

-Y_k - нейроны, составляющие выходной слой сети. Каждый нейрон обладает индексом k;

- y_in_k - значение, подаваемое на вход нейронам, принадлежащим выходному слою, высчитывается по формуле: $y\_in_k = w_{0k} + \sum_{j=1}^{m} z_j \cdot w_{jk}$;

- y_k - выходной сигнал выходного слоя нейронной сети (нейронов Y_k), высчитывается по формуле: $y_k = f(y_{in_k})$;

- σ_k- величина, необходимая для модификации весовых коэффициентов связей между нейронами выходного и скрытого слоев, данное значение вычисляется с использованием ошибки нейронов выходного слоя сети. Также этим же обозначением характеризуется ошибка выходного слоя, передаваемая нейронам, связанным с нейронами выходного слоя (то есть, нейронам Z_j), вычисляется по формуле: $\sigma_k = (t_k - y_k) \cdot f'(y\_in_k)$.

Характеристики, которые необходимо учитывать при задаче выбора функции активации в данном алгоритме это:

- непрерывность функции активации;

- дифференцируемость функции активации.

Кроме того, данная функция должна быть монотонно-неубывающей. Требование оптимальной эффективности вычислений накладывает на функ-

цию активации еще одно требование – доступность (легкость) нахождения производной.

Очень часто, функция активация представляет собой также функцию с насыщением. Среди наиболее часто применяемых функций активации на практике следует выделить бинарную сигмоидальную функцию. Ее область значений принадлежит интервалу $(0; 1)$. Она определяется в соответствии с выражением **Ошибка! Источник ссылки не найден.**):

$$f_1(x) = \frac{1}{1+e^{-x}} \quad (5)$$

Как легко увидеть, производная функции обладает свойством **Ошибка! Источник ссылки не найден.**):

$$f_1'(x) = f_1(x) \cdot [1 - f_1(x)] \quad (6)$$

График данной функции приведен на рисунок .

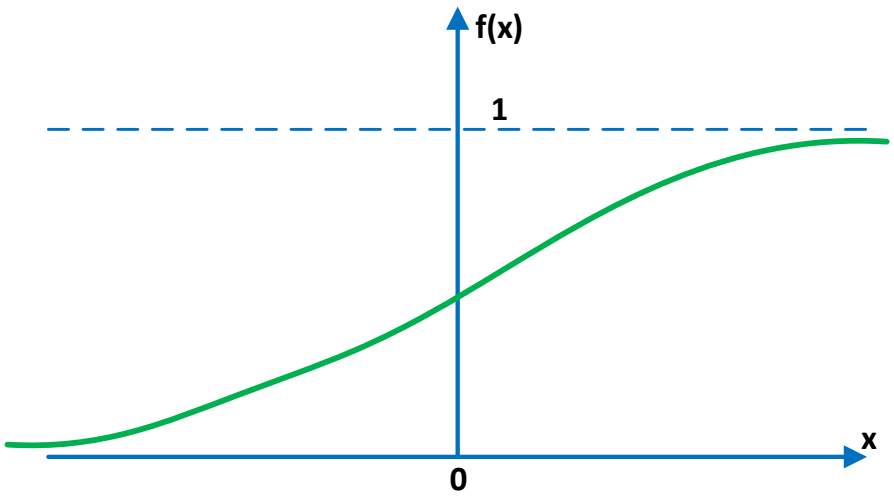

Рисунок 8 – Бинарная сигмоидальная функция

Еще одна функция активации, получившая большую популярность в практических исследованиях, – это биполярный сигмоид. Область значения данной функции – $(-1; 1)$. . Она определяется в соответствии с выражением **Ошибка! Источник ссылки не найден.**):

$$f_2(x) = \frac{2}{1+e^{-x}} - 1 \quad (7)$$

Как легко увидеть, производная функции обладает свойством **Ошибка! Источник ссылки не найден.**):

$$f_2'(x) = \frac{1}{2} \cdot [1 + f_2(x)] \cdot [1 - f_2(x)] \quad (8)$$

График данной функции приведен на рисунок .

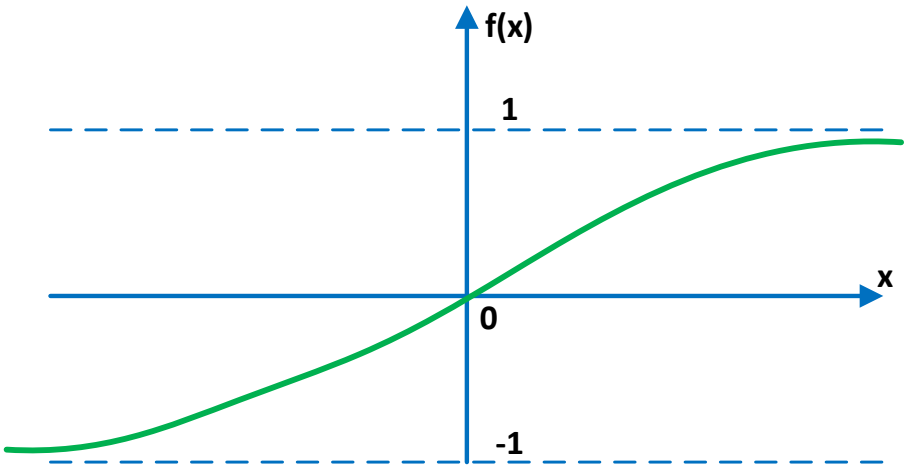

Рисунок 9 – Биполярная сигмоидальная функция

Последовательность действий при выполнении обучения сети представлено ниже.

1) Инициализация весовых коэффициентов связей; веса выбираются случайно, их начальные значения являются небольшими;

2) Далее пункты 3-9 будут выполняться, пока не выполнится условие выхода из алгоритма, а пункты 4-8 будут осуществляться для каждой пары {данные, целевое значение};

3) Происходит передача каждым нейроном, принадлежащим входному слою сети, то есть нейроны X_i , полученного сигнала нейронам скрытого слоя;

4) Каждый нейрон Z_j скрытого слоя выполняет операцию суммы взвешенных сигналов, поступивших на вход по формуле $z\_in_j = v_{0j} + \sum_{i=1}^{n} x_i \cdot v_{ij}$, далее используется функция активации $z\_in_j = f(z\_in_j)$. После осуществляется пересылка получившегося результата всем элементам, принадлежащим следующему слою (в данном случае, выходному).

5) Каждый из нейронов, принадлежащий выходному слою $(Y_k, k = 1,2, \dots m)$ выполняет суммирование взвешенных входящих сигналов $y\_in_k = w_{0k} + \sum_{j=1}^{m} z_j \cdot w_{jk}$, после чего применяется функция активации (вычисление выходного сигнала) $y_k = f(y\_in_k)$.

6) Каждому нейрону выходного слоя Y_k подается на вход правильное значение, то есть величина, верная для данного сигнала, полученного на входе сетью. Далее рассчитывается ошибка $\sigma_k = (t_k - y_k) \cdot f'(y\_in_k)$, значение, необходимое для модификации весов связи $\Delta w_{jk} = a \cdot \sigma_k \cdot z_j$ и также величина $\Delta w_{0k} = a \cdot \sigma_k$, которая используется для модификации смещения. Полученная ошибка σ_kнаправляется на скрытый слой нейронной сети.

7) Нейроны скрытого слоя суммируют полученные от нейронов выходного слоя сети ошибки по формуле $\sigma\_in_j = \sum_{k\epsilon 1}^{n} \sigma_k \cdot w_{jk}$. Также они рассчитывают значение ошибки выражением $\sigma_j = \sigma\_in_j \cdot\cdot f'(z\_in_j)$. Далее считается значение, необходимое для модификации весов связи $\Delta v_{ij} = a \cdot \sigma_j \cdot x_i$ и также величина $v_{0j} = a \cdot \sigma_j$, которая используется для модификации смещения.

8) Далее происходит коррекция весовых коэффициентов связей между нейронами. Это происходит таким образом: каждый нейрон выходного слоя нейронной сети корректирует веса с элементом смещения между выходным и скрытым слоями, при выполнении данной операции используется выражение $w_{jk}(new) = w_{jk}(old) + \Delta w_{jk}$. Аналогично действуют нейроны скрытого слоя, используя формулу $v_{ij}(new) = v_{ij}(old) + \Delta v_{ij}$.

9) После этого необходимо проверить условие конца работы метода. Таким условием может быть, например, заранее заданного количества итераций алгоритма или значение суммы квадратичной ошибки значения, полученного на выходе сети. В основе описанного метода лежит алгоритм градиентного спуска. Таким образом, градиент функции определяет направление, в котором наблюдается значительное возрастание/убывание значений функций. Здесь значение функции – это ошибка, а параметры – это весовые коэффициенты связей.

Нужно сказать, что определение начальных весовых коэффициентов и смещения – довольно важный момент в рассматриваемом алгоритме. Ранее было определено, что данные величины задаются случайным образом. Одна-

ко достижение нейронной сетью минимального значения ошибки напрямую зависит от верного выбора данных величин. Кроме этого, это также влияет на скорость достижения результата.

Таким образом, определенно существует зависимость модификации весовых коэффициентов между двумя нейронами от производной функции активации из следующего слоя сети, а также от функции активации предыдущего слоя.

Следовательно, целесообразным является недопущение таких изначально заданных весов, которые обращают в нуль значение функции активации или производной данной функции. Кроме того, не рекомендуется задавать очень большие значения начальных весов, поскольку в этом случае значения входных сигналов для каждого из нейронов, принадлежащих скрытому или выходному слоям, будут находиться с большой долей вероятности в области очень малых значений сигмоидальной функции (так называемой, области насыщения).

В тоже время, при задании слишком малых значений начальных весов значение входного сигнала для нейронов, находящихся на скрытом или выходном слоях, будет близко к нулю, что повлечет за собой значительное снижение скорости обучения нейронной сети. На основании вышесказанного, можно утверждать, сто в состав стандартной процедуры задания начальных значений весов входит присвоение им случайных значений, находящихся в границах интервала $(-0{,}5; 0{,}5)$. Соответственно, допускаются как положительные, так и отрицательные значения, поскольку конечные веса, являющиеся результатом обучения сети, тоже могут быть разных знаков.

Среди возможных вариантов алгоритма начальной инициализации весов наиболее предпочтительной видится так называемая инициализация $Bquyen - Widrow$. Следует подробнее описать его. Этот алгоритм представляет собой одну из модификаций стандартной процедуры инициализации и отличается благоприятным влиянием на скорость обучения нейронной сети. Инициализация весов связей между нейронами, находящимися на скры-

том и выходном слоях, а также смещения выходного слоя осуществляются аналогично стандартной процедуре, то есть случайными величинами, находящимися в пределах интервала $(-0{,}5; 0{,}5)$. Для дальнейшего описания целесообразно описание обозначений, используемых далее в алгоритме:

– n является числом нейронов, находящихся на входном слое;

– p обозначает количество нейронов, расположенных на скрытом слое;

– β - это фактор масштабирования, который вычисляется в соответствии с выражением: $\beta = 0{,}7 \cdot (p)^{\frac{1}{n}}$.

Алгоритм $Bquen - Widrow$ включает в себя несколько простых этапов. Для каждого из нейронов, принадлежащих скрытому слою, $(Z_j, j = 1,2, \dots p)$, эти этапы следующие:

Этап 1: инициализация вектора весов нейрона (связей с нейронами, находящимися на входном уровне):

$$v_{ij}(old) = random\ number\ between - 0{,}5\ and\ 0{,}5.$$

Этап 2: вычисление весов в соответствии с выражением **Ошибка! Источник ссылки не найден.**):

$$\|v_{ij}\| = \left(v_{1j}(old)^2 + v_{2j}(old)^2 + \cdots + v_{nj}(old)^2\right)^{\frac{1}{2}} \quad (9)$$

Этап 3: переинициализация значений весов в соответствии с выражением **Ошибка! Источник ссылки не найден.**):

$$v_{ij} = \frac{\beta \cdot v_{ij}(old)}{\|v_{ij}(old)\|} \quad (10)$$

Этап 4: задание значения смещения:

$$v_{0j} = random\ number\ between - \beta\ and\ \beta.$$

2.2 Методы оптимизации нейронных сетей

Среди всех алгоритмов обучения нейронных сетей, в основе которых лежит метод стохастического градиентного спуска, особого внимания заслуживает метод Адама. Данный алгоритм широкого известен среди специали-

стов, занимающихся разработкой приложений в сферах глубокого машинного обучения, компьютерного зрения, а также обработки естественного языка. Перед описанием данного алгоритма стоит остановиться на основных проблемах, возникающих в ситуации, когда используется алгоритм стохастического градиентного спуска [18].

Обычный градиентный спуск характеризуется следующими выражениями **Ошибка! Источник ссылки не найден.**) и **Ошибка! Источник ссылки не найден.**):

$$\Delta\theta = -\eta\nabla_\theta J(\theta) \quad (11)$$

$$\theta = \theta + \Delta\theta = \theta - -\eta\nabla_\theta J(\theta) \quad (12)$$

где θ является параметрами сети, $J(\theta)$ – функцией потерь в случае машинного обучения (целевой функцией), а η – скоростью обучения. На первый взгляд, выражения выглядят достаточно простыми, однако много проблем может доставить выражение $\nabla_\theta J(\theta)$. Действительно, обновление параметров выходного слоя является довольно простой задачей, но чтобы достичь параметров расположенных за ним слоев, нужно пройти сквозь нелинейности. Свое влияние также оказывают полученные от них производные. Этот принцип называется $backpropagation$ (принцип обратного распространения ошибки) и был описан в предыдущем пункте.

Формулы в явном виде для обновления весов в произвольном месте в середине сети имеют очень громоздкий вид. Это связано с тем, что существует зависимость каждого нейрона от всех связанных с ним нейронов, а тух – от всех нейронов, связанных с ними и т.д.

При этом количество слоев в современных нейронных сетях может составлять величину порядка десятков. А каждый вес соответствует переменной в $J(\theta)$. Безусловно, благодаря такому большому количеству степеней свободы существует возможность построения отображений, обладающих большой сложностью. Однако трудоемкость подобного процесса очень велика. Помимо этого, существует ряд и других недостатков:

– вероятность «застревания» в точках локального минимума и седловых токах;

– обновление определенных параметров может происходить с меньшей частотой, чем других;

– недопустимо низкая скорость обучения приводит к очень долгой сходимости алгоритма и застреванию в точках локального минимума, тогда как излишне большая скорость повышает вероятность пропуска точек узкого глобального минимума или вовсе ведет к расхождению алгоритма;

– сложность ландшафта целевой функции: наблюдается чередование плато с областями, для которых характерна сильная нелинейность. Значение производной на плато приближается к нулю, а появление внезапного обрыва может привести к значительному удалению от решения задачи.

Приходится параллельно решать задачу обеспечения вычислительной подъемности исходной задачи. Безусловно, в настоящее время разработаны рабочие оптимизаторы второго порядка, однако в данном разделе речь пойдет об алгоритмах, их не использующих. Перед описанием непосредственно алгоритма Адама целесообразно остановиться на некоторых алгоритмах, преимущества которых сочетает в себе Адам.

Первым из них является $Nesterov\ Accelerated\ Gradient$. Вообще говоря, сущность идеи методов с накоплением импульсов несложна. Действительно, при движении в течение некоторого времени в определенном направлении логично предположение о целесообразности движения в этом же направлении и в будущем некоторое время. Для этого необходимо обращение к недавней истории модификаций каждого из параметров.

Так, допустимо хранение последних n экземпляров $\Delta\theta$ и пересчета на каждом шаге среднего значения. Однако подобный подход подразумевает наличие слишком большого объема памяти при больших n. Вместе с тем, расчет точного среднего не является обязательным, достаточна лишь его оценка, что позволяет использование экспоненциального скользящего среднего согласно выражению **Ошибка! Источник ссылки не найден.**):

$$v_t = \gamma \cdot v_{t-1} + (1 - \gamma) \cdot x \quad (13)$$

Для накопления значения выполняется умножение уже имеющегося значения на коэффициент сохранения $0 < \gamma < 1$ с последующим прибавлением очередной величины, умноженной на $1 - \gamma$. При приближении γ к значению 1 увеличивается окно накопления и усиливается сглаживание. Следовательно, история x начинает оказывать влияние большее, чем очередное значение x. При достижении $x = 0$ с определенного момента будет наблюдаться затухание v_t в соответствии с геометрической прогрессией, экспоненциально. Именно этим и объясняется этимология названия метода. Далее используется экспоненциальное бегущее среднее для накопления градиента целевой функции нейронной сети согласно выражениям **Ошибка! Источник ссылки не найден.**) и **Ошибка! Источник ссылки не найден.**):

$$v_t = \gamma \cdot v_{t-1} + \eta \cdot \nabla_\theta J(\theta) \quad (14)$$

$$\theta = \theta - v_t \quad (15)$$

Значение γ обычно выбирается равным около 0,9. Следует учитывать, что $1 - \gamma$ не исчезло, а было включено в η. В литературе встречается и модификация выражения с применением явного множителя. С уменьшением γ алгоритм становится все более схожим с классическим методом градиентного спуска.

Следует отметить, что значение, накопленное в v_t, может быть существенно больше каждого из $\eta \cdot \nabla_\theta J(\theta)$. Таким образом, данный результат не является удовлетворительным. Для решения данной проблемы Нестеров использовал заглядывание вперед в направление вектора обновления. Градиент функции потерь рассчитывается в точке $\theta - \gamma \cdot v_{t-1}$, а не в точке θ. Это происходит потому что предполагается смещение на $\gamma \cdot v_{t-1}$.

Отсюда имеют место выражения **Ошибка! Источник ссылки не найден.**) и **Ошибка! Источник ссылки не найден.**):

$$v_t = \gamma \cdot v_{t-1} + \eta \cdot \nabla_\theta J(\theta - \gamma \cdot v_{t-1}) \quad (16)$$

$$\theta = \theta - v_t \quad (17)$$

Подобное изменение способствует более быстрому перемещению, если в стороне, куда направлено движение, наблюдается увеличение значения производной, и замедлению в обратном случае.

Вместе с тем, заглядывание вперед может привести к проблемам при задании слишком больших γ и η. В этом случае возможна ситуация пропуска регионов, имеющих противоположный знак градиента.

Далее не будет использоваться манипулирование аргументом целевой функции и для краткости вводится следующее обозначение **Ошибка! Источник ссылки не найден.**):

$$g_t = \nabla_\theta J(\theta_t) \tag{18}$$

Среди всех алгоритмов оптимизации следует отметить такой, относительно простой, как $Adagrad$ $(adaptive\ gradient)$.

Часть признаков может обладать высокой информативностью, но малой частотой появления. Ставится задача обновления параметров с учетом уровня типичности фиксируемого признака. С этой целью предлагается хранение для каждого из параметров сети суммы квадратов его обновлений. Именно эта сумма рассматривается как прокси для типичности. В случае принадлежности параметра к цепочке часто активирующихся нейронов наблюдается быстрое накопление суммы. Следовательно, выражение для обновления принимает вид **Ошибка! Источник ссылки не найден.**), **Ошибка! Источник ссылки не найден.**):

$$G_t = G_t + g_t^2 \tag{19}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \varepsilon}} \cdot g_t \tag{20}$$

где G_t является суммой квадратов обновлений, а ε – сглаживающим параметром, который требуется для недопущения деления на ноль. Часто обновлявшийся в прошлом параметр обладает большой G_t, следовательно, значение знаменателя в **Ошибка! Источник ссылки не найден.**) велико.

Сущность метода $Adagrad$ заключается в использовании чего-нибудь для уменьшения обновлений для элементов, обновление которых происходит

и так часто. Приведенная выше формула не является обязательной, поэтому $Adagrad$ принято считать семейством алгоритмов.

Еще к достоинствам данного алгоритма может быть отнесено отсутствие необходимости точного подбора скорости обучения. Достаточным является определение ее достаточно большой. Это необходимо для обеспечения приемлемого запаса. Вместе с тем, слишком большое ее значение может привести к расхождению алгоритма.

Недостаток алгоритма $Adagrad$ состоит в возможности сколь угодно большого увеличения G_t в **Ошибка! Источник ссылки не найден.**). Это неизбежно рано или поздно повлечет за собой слишком маленькие обновления и неработоспособность алгоритма. Для исправления этого недостатка были разработаны алгоритмы $RMSProp$ и $Adadelta$.

Идея предыдущего алгоритма модифицируется: веса с высокой частотой обновления должны обновляться реже, однако вместо полной суммы обновлений используется квадрат градиента, усредненный в соответствии с историей. Здесь также применяется выражение для экспоненциально затухающего бегущего среднего **Ошибка! Источник ссылки не найден.**). Далее, через $E[g^2]_t$ обозначается бегущее среднее в момент t. Оно определяется в соответствии с выражением **Ошибка! Источник ссылки не найден.**):

$$E[g^2]_t = \gamma \cdot E[g^2]_{t-1} + (1-\gamma) \cdot g_t^2 \quad (21)$$

Тогда выражение **Ошибка! Источник ссылки не найден.**) принимает вид **Ошибка! Источник ссылки не найден.**):

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \varepsilon}} \cdot g_t \quad (22)$$

В знаментателе, по сути, находится квадратный корень из среднего квадратов градиентов. Именно отсюда и берет свое название метод $RMSProp - root\ mean\ square\ propagation$ **Ошибка! Источник ссылки не найден.**):

$$RMS[g]_t = \sqrt{E[g^2]_t + \varepsilon} \quad (23)$$

Отличие $Adadelta$ от $RMSProp$ состоит в добавлении в числитель выражения **Ошибка! Источник ссылки не найден.**) стабилизирующего члена, пропорционального RMS от $\Delta\theta_t$. На шаге t значение $RMS[\Delta\theta]_t$ еще неизвестно, поэтому для обновления параметров необходимо три этапа (вместо двух). В первую очередь, осуществляется накопление квадрата градиента, потом обновление θ, и затем обновление $RMS[\Delta\theta]$ в соответствии с выражениями () - ():

$$\Delta\theta = \frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} \cdot g_t \tag{24}$$

$$\theta_{t+1} = \theta_t - \frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} \cdot g_t \tag{25}$$

$$E[\Delta\theta^2]_t = \gamma \cdot E[\Delta\theta^2]_{t-1} + (1-\gamma) \cdot \Delta\theta_t^2 \tag{26}$$

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \varepsilon} \tag{27}$$

Такие модификации обусловлены необходимостью совпадения размерностей θ и $\Delta\theta$. Следует отметить, что у $learning\,rate$ отсутствует размерность, поэтому все алгоритмы, рассматриваемые до этого, включали сложение размерной величины с безразмерной.

На данный алгоритм накладывается ограничение: необходимо наличие ненулевого $RMS[\Delta\theta]_{-1}$ для первого шага. В противном случае, все последующие $\Delta\theta$ и, соответственно, $RMS[\Delta\theta]_t$ будут принимать значение 0. Однако эта проблема была решена еще раньше путем добавления в RMS величины ε. Вместе с тем, если значение $RMS[\Delta\theta]_{-1}$ будет недостаточно велико, может наблюдаться ситуация поведения, противоположного описанному в $Adagrad$ и $RMSProp$: будет производиться более частое обновление (до определенного предела) весов, которые применяются чаще. Действительно, в данном алгоритме для достижения $\Delta\theta$ значимости необходимо накопление параметром большой суммы в числителе дроби.

Однако, складывается впечатление, что авторами алгоритма допускался сознательно подобный эффект. Алгоритмы $Adadelta$ и $RMSProp$, также как

и $Adagrad$, не нуждаются в очень точном подборе скорости обучения – достаточным является приблизительное значение.

При приближении γ к единице будет наблюдаться увеличение веремени, в течение которого алгоритмами $Adadelta$ и $RMSProp$ с большим $RMS[\Delta\theta]_{-1}$ будет сильно обновляться нечасто используемые веса. При $\gamma \approx 1$ и $RMS[\Delta\theta]_{-1} = 0$ $Adadelta$ в течение продолжительного времени будет подозрительно относиться к нечасто используемым весам. Последнее может вызвать полную неработоспособность алгоритма, а может и привести к намеренно «жадному» поведению, при котором алгоритмом сначала осуществляется обновление нейронов, которые кодируют наиболее предпочтительные признаки.

Все преимуществ вышеописанных алгоритмов были реализованы в алгоритме Адам. Адам ($Adam, adaptive\ moment\ estimation$) является оптимизационным алгоритмом. В нем сочетаются идеи накопления движения и более слабого обновления весов для типичных признаков. Выражение **Ошибка! Источник ссылки не найден.**) будет выглядеть, как **Ошибка! Источник ссылки не найден.**):

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \tag{27}$$

Отличие алгоритма $Adam$ от алгоритма Нестерова заключается в том, что в нем осуществляется накапливание не $\Delta\theta$, а значения градиента. Помимо этого, важна информация о частоте изменения градиента. Авторами алгоритма предложена для этого оценка еще и средней нецентрированной дисперсии в соответствии с выражением **Ошибка! Источник ссылки не найден.**):

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \tag{28}$$

Последнее выражение, фактически, представляет собой уже известный $E[g^2]_t$, так же, как и в алгоритме $RMSProp$.

Ключевое отличие заключается в начальных приближениях m_t и v_t. Им присуща та же проблема, что и $E[g^2]_t$ в алгоритме $RMSProp$ – при задании нулевого начального значения длительность накопления будет очень вы-

сока, особенно при большом размере окна накопления ($0 \ll \beta_1 < 1, 0 \ll \beta_2 < 1$). В то же время задание ненулевых начальных значений ведет к появлению еще двух гиперпараметров. Наличие подобных гиперпараметров нежелательно, поэтому осуществляется искусственное увеличение m_t и v_t на первых шагах. Но тогда справедливо выражение **Ошибка! Источник ссылки не найден.**):

$$\widehat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \widehat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{29}$$

Правило обновление приобретает вид **Ошибка! Источник ссылки не найден.**):

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\widehat{v}_t+\varepsilon}} \cdot \widehat{m}_t \tag{30}$$

Разработчиками алгоритма выводится выражение **Ошибка! Источник ссылки не найден.**) с помощью разворачивания выражений **Ошибка! Источник ссылки не найден.**), **Ошибка! Источник ссылки не найден.**). Например, для v_t выражение приобретает вид ():

$$\begin{aligned} E[v_t] &= E\left[(1-\beta_2) \cdot \sum_{i=1}^{t} \beta_2^{t-i} \cdot g_i^2\right] \\ &= E[g_t^2] \cdot (1-\beta_2) \cdot \sum_{i=1}^{t} \beta_2^{t-i} + \zeta \\ &= E[g_t^2] \cdot (1-\beta_2) \cdot \frac{1-\beta_2^t}{1-\beta_2} + \zeta \\ &= E[g_t^2] \cdot (1-\beta_2^t) + \zeta \end{aligned} \tag{31}$$

Значение слагаемого ζ близко к нулю при стационарном распределении $p(g)$, что очень неточно в рассматриваемых случаях, однако все равно допустим перенос скобки с β_2^t влево. Можно условиться, что при $t = 0$ наблюдается бесконечная история идентичных обновлений **Ошибка! Источник ссылки не найден.**):

$$\widehat{v}_1 = v_1 + \beta_2 \cdot v_1 + \beta_2^2 \cdot v_1 + \cdots = \frac{v_1}{1-\beta_2} \tag{32}$$

При получении значения v_t, более близкого к реальному, производится принудительное затухание (подавление) «виртуальной» части ряда так, как показано в **Ошибка! Источник ссылки не найден.**):

$$\widehat{v}_2 = v_2 + \beta_2^2 \cdot v_2 + \beta_2^4 \cdot v_2 + \cdots = \frac{v_1}{1-\beta_2^2} \quad (33)$$

Разработчиками алгоритма предлагаются следующие значения в качестве используемых по умолчанию: $\beta_1 = 0{,}9;\ \beta_2 = 0{,}999;\ \varepsilon = 10^{-8}$.

Одной из модификаций алгоритма $Adam$ является алгоритм $Adamax$. В данном алгоритме вместо дисперсии в **Ошибка! Источник ссылки не найден.**) осуществляется вычисление инерционного момента распределения градиентов, имеющих произвольную степень p. Конечно, такой прием может вызвать нестабильность в вычислениях. Однако при p, стремящемся к бесконечности, были получены на практике достаточно эффективные результаты. Выражение принимает вид **Ошибка! Источник ссылки не найден.**):

$$v_t = \beta_2^p \cdot v_{t-1} + \left(1 - \beta_2^p\right) \cdot |g_t|^p \quad (34)$$

Следует отметить, что на смену β_2 пришел подходящий по размерности β_2^p. Помимо этого, внимания заслуживает то обстоятельство, что для применимости в выражениях алгоритма $Adam$ значения, полученного в **Ошибка! Источник ссылки не найден.**), требуется извлечение из него корня, то есть: $u_t = v_t^{\frac{1}{p}}$. Для вывода решающего правила вместо **Ошибка! Источник ссылки не найден.**) полагается $p \to \infty$, после чего выполняется разворачивание под корнем v_t с помощью **Ошибка! Источник ссылки не найден.**). Выражение имеет вид **Ошибка! Источник ссылки не найден.**):

$$
\begin{aligned}
u_t &= \lim_{p\to\infty} v_t^{\frac{1}{p}} \\
&= \lim_{p\to\infty}\left[\beta_2^p \cdot v_{t-1} + \left(1-\beta_2^p\right)\cdot |g_t|^p\right]^{\frac{1}{p}} \\
&= \lim_{p\to\infty}\left[\left(1-\beta_2^p\right)\cdot\sum_{i=1}^{t}\beta_2^{p\cdot(t-i)}\cdot|g_i|^p\right]^{\frac{1}{p}} \\
&= \lim_{p\to\infty}\left(1-\beta_2^p\right)^{\frac{1}{p}}\cdot\left(\sum_{i=1}^{t}\beta_2^{p\cdot(t-i)}\cdot|g_i|^p\right)^{\frac{1}{p}} \\
&= \lim_{p\to\infty}\left(\sum_{i=1}^{t}\beta_2^{p\cdot(t-i)}\cdot|g_i|^p\right)^{\frac{1}{p}} \\
&= \max(\beta_2^{t-1}\cdot|g_1|,\ \beta_2^{t-2}\cdot|g_2|,\dots,\beta_2\cdot|g_t|)
\end{aligned}
\tag{35}
$$

2.3 Выбор критериев для сравнения алгоритмов обучения

Известно, что непосредственное влияние на сходимость итеративных алгоритмов обучения влияет целый комплекс критериев. Таковыми являются, например, начальные значения весовых коэффициентов сети, количество данных, используемых для обучения. Также влияют критерии качества обучения и длительности обучения. Критерий качества обучения – максимальная общая оценка обучения, которая определяется заблаговременно, а критерий длительности обучения – число допустимых циклов обучения. Перечисленные последние два критерия останова обучения эффективны в использовании в ходе сравнения итеративных алгоритмов обучения нейронных сетей.

После поочередной фиксации каждого из критериев останова можно составить картину изменения для каждого из методов обучения значений другого критерия в условиях одинаковости обучающих выборок. Для оценки результатов эксперимента используются такие критерии, как длительность работы обучающего алгоритма ($t_{\text{об}}$), затраченное число циклов обучения ($epoch$), а также полученная точность (уровень ошибки, E).

Однако на практике было выяснено, что данный набор критериев недостаточен. Это объяснилось тем, что при обучении нейронных сетей одного и того же вида, с одним и тем же набором данных на входе, одним и тем же алгоритмом обучения однако с разными значениями весовых коэффициентов происходит получение сильно отличающихся друг от друга результатов.

Это происходит из-за избыточности памяти, качества аппроксимации данных для обучения, разными уровнями логической прозрачности этих моделей. Именно поэтому необходимо добавить дополнительно к данным критериям сравнения частных и общих критериев, с помощью которых можно будет оценить все свойства обученных сетей в количественном выражении, а также выбрать с использованием этого наиболее подходящую для данной задачи модель.

Для сравнения методов обучения также важным критерием является количество дополнительных переменных N_v для организации процесса вычисления. Дополнительные переменные нужны для хранения промежуточных вычислений. Исходя из этого будут более полезны алгоритмы, которые требуют меньше дополнительных переменных, это полезно для экономии ресурсов вычислительных средств.

Основная характеристика любой системы распознавания – сложность.

Количество нейронов $N_{\text{н}}$ определяет структурную сложность модели нейронной сети, так как они являются ее основными деталями. Выражение (36) определяет это значение для сети, имеющей несколько скрытых слоев.

$$N_{\text{н}} = \sum_{\mu=1}^{M} N_{\mu} \qquad (36)$$

Число используемых ячеек памяти $N_{\text{п.м.}}$ и время работы $t_{\text{р}}$ нужны для того, чтобы верно оценить вычислительную сложность модели сети. Величина $t_{\text{р}}$ нужна для определения значений выходов при заданных входах анализируемой модели сети, это делается либо для одного экземпляра выборки, либо для всех.

Число используемых ячеек памяти $N_{\text{п.а.}}$ и время, нужное, чтобы обучить модель (обозначается $t_{\text{об}}$) используется для оценки вычислительной сложности алгоритма синтеза модели нейронной сети.

Таким образом, количество используемых ячеек памяти рассчитывается следующим выражением (37):

$$N_{\text{п.а.}} = \eta_{\text{в}} \cdot S \cdot (N + N_{\text{м}}) + \eta_w \cdot N_w + \eta_v \cdot N_v \quad (37)$$

$-\ \eta_{\text{в}}$ – число ячеек, которые нужны для хранения одного элемента данных выборки;

- S – обозначение числа экземпляров выборки; N – признаки экземпляров;
- $N_{\text{м}}$ - количество целевых признаков;
- $\eta_{\text{в}}$ - количество ячеек для хранения одного весового коэффициента;
- N_w – пороговые значения и весовые коэффициенты сети;
- η_v - количество ячеек для хранения одной доп. переменной;
- N_v- доп. переменные.

Основные моменты, характеризующие нейронные сети – это сложность их реализации и временные затраты (при обработке одного элемента).

По формуле (38) можно рассчитать вычислительную сложность нейрона i слоя μ $T^{(\mu,i)}$:

$$T^{(\mu,i)} = N^{(\mu,i)} \cdot \left(T_C^{(\mu,i)} + T_\varphi^{(\mu,i)}\right) + T_\Psi^{(\mu,i)} \quad (38)$$

- $N^{(\mu,i)}$ - количество входов нейрона i;
- $T_C^{(\mu,i)}$ – вычислительная сложность одного синапса нейрона i;
- $T_\varphi^{(\mu,i)}$- вычислительная сложность дискриминантной функции нейрона i (два аргумента);
- $T_\Psi^{(\mu,i)}$ - вычислительная сложность функции активации нейрона.

По формуле (39) можно определить вычислительную мощность T_1 Последовательно реализованной нейронной сети прямого распространения.

$$T_1 = \sum_{\mu=1}^{M} \sum_{i=1}^{N_\mu} T^{(\mu,i)} \quad (39)$$

N_μ- число нейронов слоя, M – количество слоев.

Искусственная нейронная сеть, которая имеет возможность к решению задач методами, понимаемыми человеком, называется логически прозрачной.

Количество образов, которые нейронная сеть может запомнить, уменьшается, если уменьшить объем памяти данной сети. Но если, например, рассматривать две сети, которые могут обеспечить заданную точность распознавания, но которые имеют разный объем памяти, то более выигрышным вариантом будет сеть с меньшим объемом. Для расчета коэффициента избыточности памяти сети используется выражение (40):

$$K_I = \frac{N_w}{S \cdot N}, N_w > 1, S > 0, N > 0 \quad (40)$$

Если значение коэффициента $K_I > 1,$ то память нейронной сети избыточна; при $K_I = 1$ сетью запоминается вся выборка для обучения; $K_I < 1$ – нейронная сеть не может запомнить всю выборку, но все еще имеет способность аппроксимации и обобщения.

Логическая прозрачность сети во многом зависит от таких величин, как количество связей между нейронами сети в общем и количество связей между определенными нейронами. Если количество связей сократить, то понимание и анализ сети даются человеку намного проще.

K_R – коэффициент разреженности связей сети прямого распространения, его вычисляют по формуле (41):

$$K_R = \frac{N_{w=0}}{\sum_{\mu=1}^{M} N_{(\mu-1)} \cdot N_\mu}, N_0 = N \quad (41)$$

N_w - число нулевых весовых коэффициентов нейронной сети.

$$0 \le N_{w=0} \le \sum_{\mu=1}^{M} N_{(\mu-1)} \cdot N_\mu$$

K_C – коэффициент связности нейронной сети с несколькими скрытыми слоями, вычисляется по формуле (42):

$$K_C = 1 - K_R = 1 - \frac{N_{w=0}}{\sum_{\mu=1}^{M} N_{(\mu-1)} \cdot N_\mu}, N_0 = N \quad (42)$$

K_L – единичные синапсы в сети, считаются по формуле (43):

$$K_L = \frac{N_{w=1}}{\sum_{\mu=1}^{M} N_{(\mu-1)} \cdot N_\mu} \quad (43)$$

$N_{w=1}$ - количество весовых коэффициентов сети, равных по модулю 1.

K_N – неединичные синапсы в сети, считаются по формуле (44):

$$K_N = 1 - K_L = 1 - \frac{N_{w=1}}{\sum_{\mu=1}^{M} N_{(\mu-1)} \cdot N_\mu} \quad (44)$$

K_T - коэффициент логической прозрачности связей нейронной сети (45):

$$K_T = K_R + K_L = \frac{N_{w=0} + N_{w=1}}{\sum_{\mu=1}^{M} N_{(\mu-1)} \cdot N_\mu}, N_0 = N \quad (45)$$

K_S – коэффициент логической непрозрачности связей нейронной сети (46):

$$K_S = 1 - \frac{N_{w=0} + N_{w=1}}{\sum_{\mu=1}^{M} N_{(\mu-1)} \cdot N_\mu}, N_0 = N \quad (46)$$

Для вышеупомянутых коэффициентов существуют следующие свойства: $K_T + K_S = 1; 0 \le K_T \le 1; 0 \le K_S \le 1$.

Тип функции активации важен при вычислении логической прозрачности нейрона i слоя μ сети $K_E^{(\mu,i)}$. Если тип функции линейный или пороговый, то данный коэффициент принимает значение 1, если тип функции иной, то 0.

K_U – коэффициент логической прозрачности сети, вычисляется по формуле (47):

$$K_U = \frac{\sum_{\mu=1}^{M} \sum_{i=1}^{N_\mu} K_E^{(\mu,i)}}{K_M \cdot \sum_{\mu=1}^{M} N_\mu}, K_M \neq 0 \quad (47)$$

О снижении логической прозрачности сети сообщает уменьшение K_U, и наоборот, повышение K_U говорит об увеличении логической прозрачности.

Рассмотрим еще одно качество – аппроксимацию. При неизменной ошибке данная характеристика увеличивается, при этом число используемых высов сети сокращается.

K_A – качество аппроксимации сети (48):

$$K_A = \frac{E}{N_w - N_{w=0}} \quad (48)$$

Здесь E – общая допускаемая сетью ошибка, например, среднеквадратическая. Должно выполняться условие $E \leq \xi$, где ξ – самая большая ошибка, которую может допустить нейронная сеть.

3 Программная реализация

3.1 Формирование набора данных

Сравнение алгоритм требует использования баз данных с большим количеством записей, поэтому в качестве исходного набора данных, на которых производится обучение, был выбран набор данных предоставленный специалистами:

– Courant Institute, NYU;

– Google Labs, New York;

– Microsoft Research, Redmond.

Структура баы данных для обучения представлена наборами изображений рукописных цифр. Тестирование такого набора предполагает выявление полученных отклонений от корректного определения рукописной цифры с учетом предоставленного тестового набор данных.

Общий объем набора рукописных цифр для обучения составляет 60000 примеров, а также набор для предварительного обучения 10000 примеров. С целью корректного распознавания все используемые изображения рукописных цифр отцентрированы и приведены к единому размерному формату путем нормализации.

Структура предоставленной базы вариантов изображений представлена следующими файлами:

– t10k-images-idx3-ubyte.gz - набор тестовых изображений, которые могут быть использованы для предварительного обучения;

– t10k-labels-idx1-ubyte.gz – метки изображений, используемых в наборе для предварительного обучения;

– train-images-idx3-ubyte.gz – набор изображений, которые могут быть использованы для обучения;

– train-labels-idx1-ubyte.gz – метки изображений, используемых в наборе для обучения.

Для упрощения обработки такого большого объем данных изображения приведены к нестандартным форматам и не могут быть просмотрены в базе

данных стандартными средствами для просмотра изображений. Хранение осуществляется в векторном формате с использованием многомерных матриц (IDX), все целые числа в файлах хранятся в формате MSB first (high endian).

Нормализация изображений для корректности обучения и распознавания проводилась в рамках поля 20x20 пикселей, при этом использовались черно-белые изображения.

После выполнения нормализации, которая предполагала также проведение некоторого сглаживания объекты исследования (изображения) стали включать оттенки серого цвета. Центрирование проводилось в поле размера 28x28 для выделения самого объект сравнения в виде рукописной цифры.

На предложенных данных авторы не рекомендуют использование методов SVM и K-ближайших соседей из-за серьезного роста ошибок, связанных с проведенным центрированием изображений. В таком случае рекомендуется использование предварительной обработки данных для исключения этой проблемы.

Структурно баз данных NIST 1, база содержащая двоичные изображения рукописных цифр, построена из двух различных баз, которые добавлялись последовательно.

В структуру данных на данный момент входят изображения, полученные от:

– сотрудников бюро переписи населения (взрослых людей с установившимся почерком);

– школьников (детей, почерк которых разнообразен еще и делающих множество помарок).

В результате наборы имеют радикальные отличия при распознавании:

– набор от сотрудников бюро переписи не содержит практически лишних компонентов, хорошо центрируется и распознается;

– набор от школьников включает дополнения к изображениям рукописных цифр, что приводит к искажению изображения при центрировании и более низкому уровню распознавания.

Таким образом, для проверки корректности метода или алгоритма распознавания необходимо использование полного смешанного набора данных, который и представлен в компонентах предоставленных баз изображений.

При этом для точности проведения экспериментов, включающих сравнение методов обучения, элементы обучаемой выборки сформированы людьми, почерки которых не использовались для составления тестовой выборки. В создании набора данных использовались варианты написания цифр от 500 авторов.

3.2 Описание используемых библиотек

Корпорация *Google*, выпустив в 2015 году программное обучение с открытым кодом *TensorFlow*, сумела заинтересовать в нем потенциальных покупателей, и на данный момент данная технология становится все более и более востребованной.

Открытость исходного кода библиотеки машинного обучения *TensorFlow* в *Google* способствует упрощению процесса разработки и развертывания нейронных сетей повышенной сложности. У *TensorFlow* не предусмотрена возможность предоставления каждому из разработчиков прав пожинать плоды машинного обучения, однако данный инструмент делает возможным подключение к программе разработчика с помощью интерфейсов *API* для языков программирования $C/C++$ и *Python*.

Данная библиотека делает намного проще включение в разрабатываемые приложения различных функций, отвечающих за распознавание речи и изображений, обработку естественного языка и другие возможности, которые предоставляет искусственный интеллект в настоящее время.

Следует отметить, что *TensorFlow* не является единственной библиотекой глубинного обучения (что уже было сказано в предыдущем разделе работы), однако она, как и механизм поиска *Google*, по праву называется в своем классе лучшей. В качестве имеющихся альтернатив стоит упомянуть программное обеспечение *Torch*, создателями которого являются швейцарские

исследователи, а также разработку Калифорнийского университета в Беркли *Caffe*, в разработке последней версии которого *Caffe*2 принимали участие специалисты *Facebook*.

Корпорация *Google* дает возможность использования *TensorFlow* только на одном компьютере, обладающем несколькими графическими процессорами, что, очевидно, ведет к сдерживанию масштабов применения данного инструмента в бизнесе.

Масштабность проекта *TensorFlow* больше, чем кажется на первый взгляд. Безусловно, направленность проекта на глубинное обучение и прямая связь с *Google* способствовали привлечению повышенного внимания.

Также в рамках данной работы используется библиотека *Keras* (в последних версиях *TensorFlow* она является встроенной). *Keras* считается открытой нейросетевой библиотекой, языком разработки которой является *Python*. Основным назначением библиотеки является оперативная работа с сетями глубинного обучения. В то же время, в ходе разработки делался акцент на ее компактность, расширяемость и модульность.

Сравнительная оценка популярности фреймворков для глубинного обучения представлена на рисунок .

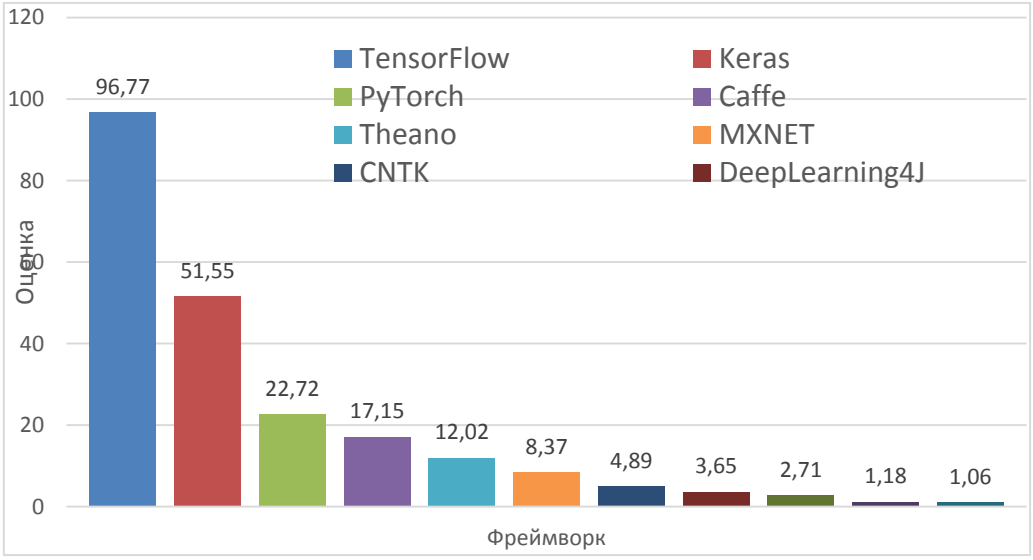

Рисунок 10 – Оценка фреймворков для глубинного обучения

Основной недостаток лидеров данного рейтинга (*TensorFlow* и *PyTorch*) заключается в сложности их освоения начинающими пользователями. *Keras*

достаточно прост в использовании, что несомненно можно причислить к списку его достоинств. Что действительно присуще данной библиотеке – это упрощение разработки моделей глубокого обучения. При разработке библиотеки особое внимание было сосредоточено на поддержке *Python*, и, как уже было сказано выше, модульности, минимализме и расширяемости. Использование *Keras* возможно как с *GPU*, так и с *CPU*. В нее заложена возможность поддержки *Python* 2 и *Python* 3.

Таким образом, для реализации методов обучения в данной работе были выбраны библиотеки *TensorFlow* и *Keras*.

3.3 Описание структуры используемой нейронной сети и ее процесса ее обучения

После того, как были определены средства, которые будут использоваться для реализации нейронной сети, а также после выбора набора данных для обучения сети и ее тестирования можно приступить к описанию архитектуры самой нейронной сети.

Выбранная архитектура сети представлена на рисунке 11. Структура сети не сложна, так как размерность изображений в обучающей выборке небольшая, а также число классов, на которые сеть должна «разделить» изображения, равно 10, что является сравнительно небольшой величиной.

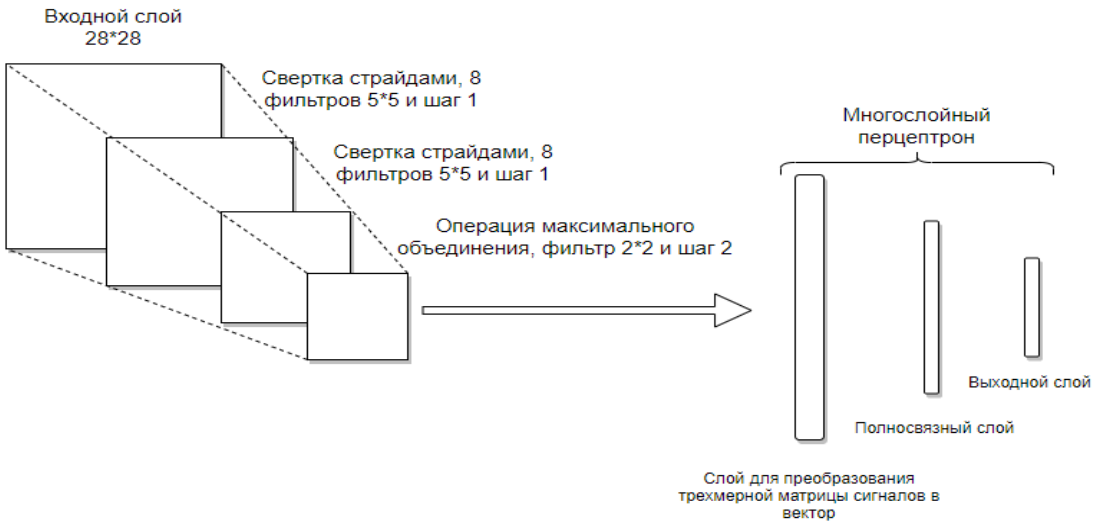

Рисунок 11 – Выбранная для реализации структура сети

Как уже было сказано раннее, в данной работе решается задача классификации, формальная постановка которой такова:

Пусть X – множество описаний объектов, Y – конечное множество номеров (меток) классов. Существует неизвестная целевая зависимость - отображение y*: $X \to Y$, значения которой известны только на объектах конечной обучающей выборки $X^m = \{(x_1, y_1), \dots, (x_m, y_m)\}$. Требуется построить алгоритм a: $X \to Y$, способный классифицировать произвольный объект $x \in X$.

Как видно из рисунка 3.2, в данной структуре есть два слоя свертки и один слой макспулинга (слой максимального объединения). Данные слои нужны именно для того, чтобы из обрабатываемого изображения выделить основные его характеристики. Также в структуре присутствует многослойный перцептрон, который используется уже для классификации полученных ранее характеристик.

Операция свертки была описана ранее, поэтому рассмотрим макспулинг. Суть данной операции в том, что «окно просеивания» проходит вдоль всех обрабатываемых на текущем слое данных. В новую матрицу подается элемент, имеющий максимальное значение, при этом выбор происходит среди пикселов, которые находятся в наблюдаемой «окном» области данных. Страйд (или шаг) «окна просеивания» зависит от размерности матрицы, которая должна получиться на выходе, а также от степени сжатия данных, поэтому он может быть отличен от 1. Слой макспулинга способствует сокращению количества пикселов, а это определенно экономит ресурсы ЭВМ, так как число операций уменьшается. Число пикселов, которое получится на выходе после операции макспулинга вычисляется выражением (49).

$$n_{out} = floor\left(\frac{n_{in}-f}{s}\right) + 1 \qquad (49)$$

n_{in} - число пикселов на входе; f – число пикселов «окна просеивания»; s – шаг (страйд).

Далее следует операция развертывания трехмерного представления исходного изображения в вектор, то есть, все строки данной матрицы соединя-

ются в один числовой ряд. Очевидно, что вектор будет иметь внушительную длину.

Вектор, полученный на предыдущем этапе затем подается на вход полносвязной сети перцептрон. Далее происходит обработка данного вектора полносвязным слоем – каждый число в данном векторе умножается на весовой коэффициент связи, затем все полученные значения суммируются и обрабатываются функцией активации.

После этого следует выходной слой сети. Данный слой выполняет функцию определения класса полученного на входе изображения. Для этого нужно, чтобы выходной слой сети имел число нейронов, соответствующее числу классов, на которые «делятся» входные изображения.

На данном слое в качестве функции активации используется softmax. С помощью данной функции будут корректироваться взвешенные и просуммированные сигналы. Данная функция выглядит следующим образом (50):

$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}} \quad (50)$$

Затем было осуществлено обучение сети с использованием нескольких методов, которые были описаны ранее. В рамках данной работы будут использованы методы: SGD (стохастический градиентный спуск), RMSProp, Adagrad, Adadelta, Adam, Adamax, Nadam.

Применяемые оптимизаторы отражены в описании средств Keras [25] (таблица).

Таблица 4 – Варианты используемых оптимизаторов

Вариант оптимизатора	Используемый алгоритм
tf.keras.optimizers.SGD(lr = 0.01,momentum = 0.0, decay = 0.0,nesterov = False,clipnorm = 1.0,clipvalue = 0.5)	Стохастический градиентный спуск (SGD)

Продолжение таблицы 4

tf.keras.optimizers.RMSprop(learning_rate=0.001,rho=0.9,epsilon=None,decay=0.0,clipnorm=1.0,clipvalue=0.5)	RMSprop
tf.keras.optimizers.Adagrad(lr=0.001,epsilon=None,decay=0.0, clipnorm=1.0,clipvalue=0.5)	Adagrad
tf.keras.optimizers.Adadelta(lr=0.001,rho=0.95,epsilon=None,decay=0.0,clipnorm=1.0,clipvalue=0.5)	Adadelta
tf.keras.optimizers.Adam(lr=0.001,beta_1=0.9,beta_2=0.999,epsilon=None,amsgrad=False,clipnorm=1.0,clipvalue=0.5)	Adam
tf.keras.optimizers.Adamax(lr=0.002,beta_1=0.9,beta_2=0.999, epsilon=None,decay=0.0,clipnorm=1.0,clipvalue=0.5)	Adamax
tf.keras.optimizers.Nadam(lr=0.002,beta_1=0.9,beta_2=0.999, epsilon=None,schedule_decay=0.004,clipnorm=1.0,clipvalue=0.5)	Nadam

Применяемые в описании оптимизаторов параметры:

amsgrad - выбор используемой модификации метода Adam;

*beta*_1,*beta*_2 - коэффициенты первых двух моментов, выбранные с экспоненциальным порядком убывания;

clipnorm - ограничение для норм градиентов при проходе в оптимизационном алгоритме;

rho - коэффициент убывания градиента;

clipvalue - задание интервала вхождения для величины градиента;

lr - определение коэффициента скорости обучения;

decay - определение коэффициента для снижения скорости обучения;

nesterov - указатель применения импульса Нестерова.

Набор параметров, которые могут быть использованы в оптимизаторе, определяется применяемым методом минимизации.

Кроме этого, необходимо сказать и о том, каким образом будет реализовано вычисление точности, с которой производится распознавание. Для этого будет использоваться функция потерь. Так как в данном случае решается задача классификации, то лучше всего будет применить кросс-энтропийную функцию потерь, выглядящую следующим образом (51):

$$H(y, \hat{y}) = \sum_i y_i \log \frac{1}{\hat{y}_i} = -\sum_i y_i \log \hat{y}_i \quad (51)$$

На выходе данной функции потерь будет качество ответа сети на какие-либо входные данные в виде вещественного числа.

$\hat{y}$ – это ответ сети, полученный нами, а y – эталонный ответ. Данное значение считается для каждого класса в отдельности, чтобы вычислить общее значение, необходимо взять среднее от полученных для каждого из классов значений.

3.4 Реализация проектного решения с выбранными алгоритмами обучения

Разработанное приложение для проведения сравнения выделенных алгоритмов построено с использованием графического интерфейса средствами языка Python (библиотека Tkinter).

Главное окно приложения включает следующие основные элементы управления: главное меню, структура которого включает несколько подменю (рисунок).

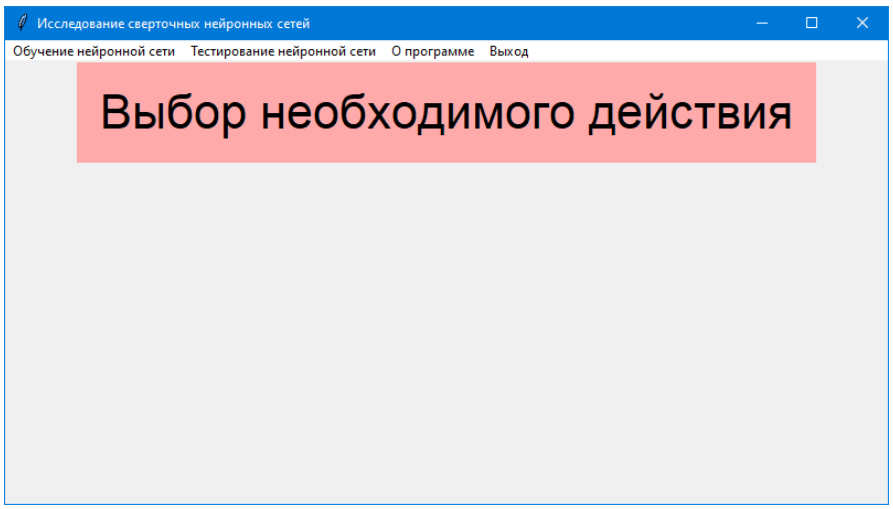

Рисунок 12 – Пример главного окна приложения

Блок-схема разработанного приложения представлена на рисунках 13-14.

Рисунок 13 – Блок-схема разработанного приложения

Рисунок 14 – Блок-схема разработанного приложения

Рисунок демонстрирует варианты по выбираемым алгоритмам обучения нейронной сети:

– Adam;
– Adamax;
– Adadelta;
– Adagrad;
– SGD;

– RMSprop;

– Nadam.

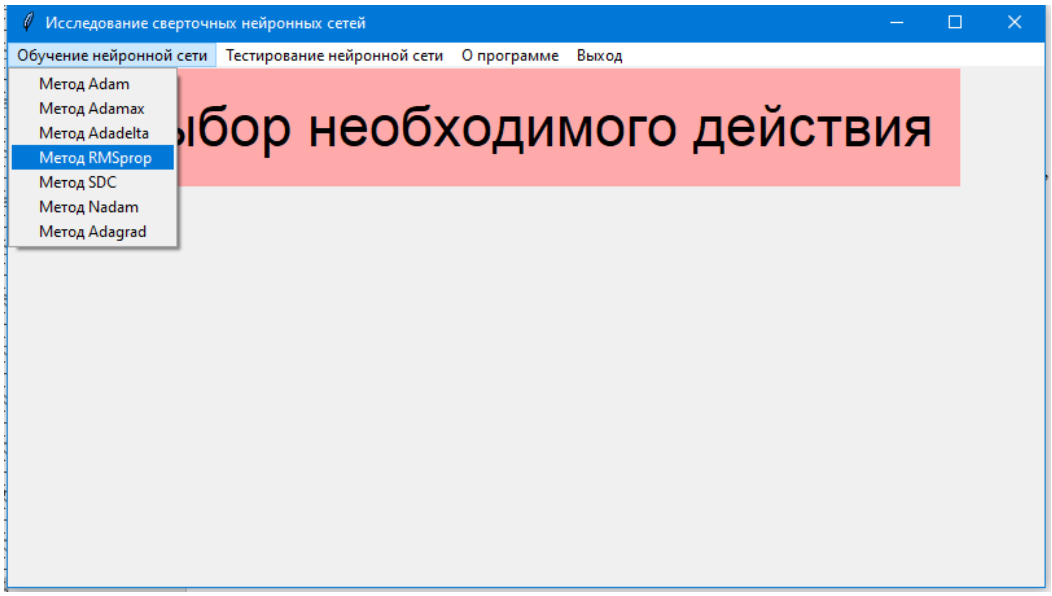

Рисунок 15 – Вариант использования меню прогона по методам обучения

Рисунок представляет подменю для тестирования нейронной сети с использованием различных методов оптимизации.

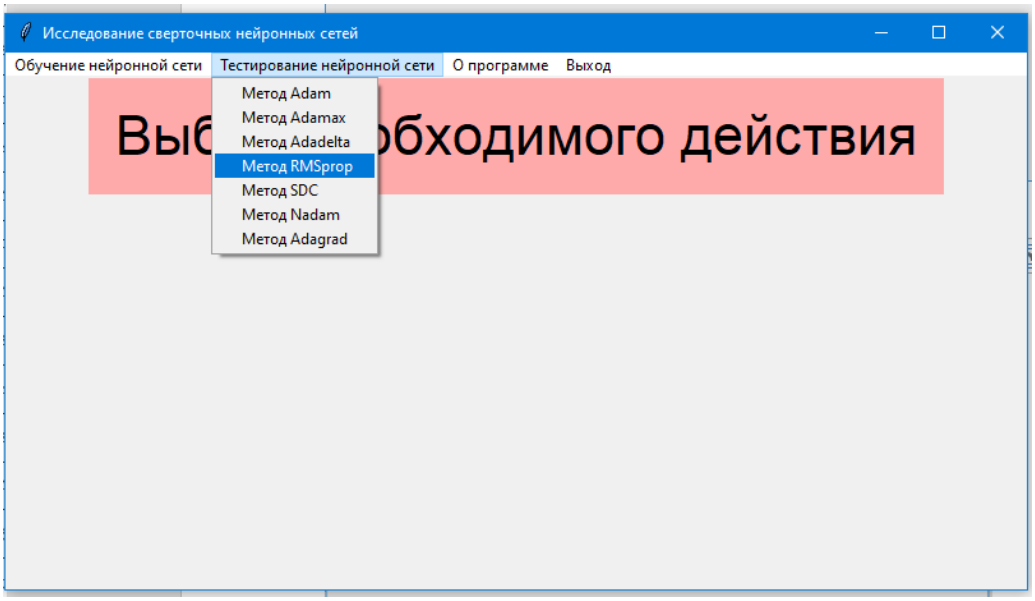

Рисунок 16 – Варианты проведения тестирования нейронной сети с использованием различных алгоритмов оптимизации

3.5 Проведение эксперимента и описание результатов тестирования

В ходе проведения эксперимента на выбранных данных с использованием различных алгоритмов оптимизации в процессе обучения нейронной сети были получены следующие результаты адаптивности обучения для каждого рассмотренного алгоритма.

Нужно сказать, что для того, чтобы определить, как точно определяет класс обученная сеть, прежде всего, необходимо оценить ее память. Именно память определяет степень способности решения сетью задачи классификации.

Структура представленной гистограммы отражает полученный определенным методом уровень распознавания для конкретных цифр, представленных в наборах данных в виде изображений рукописных цифр от 500 авторов.

Для каждого алгоритма получены следующие результаты, представленные в виде гистограмм для каждого алгоритма.

Результаты распознавания по наборам цифр с использованием варианта оптимизатора с алгоритмом Adam (рисунок).

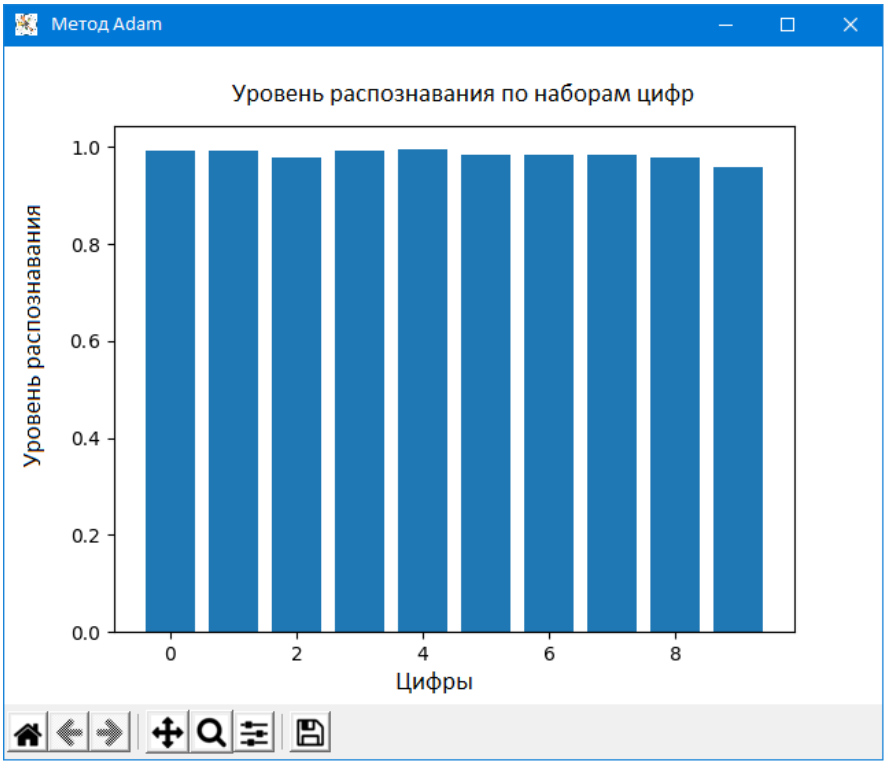

Рисунок 17 – Результаты распознавания наборов цифр по методу Adam

Результаты распознавания по наборам цифр с использованием варианта оптимизатора с алгоритмом Adamax (рисунок рисунок).

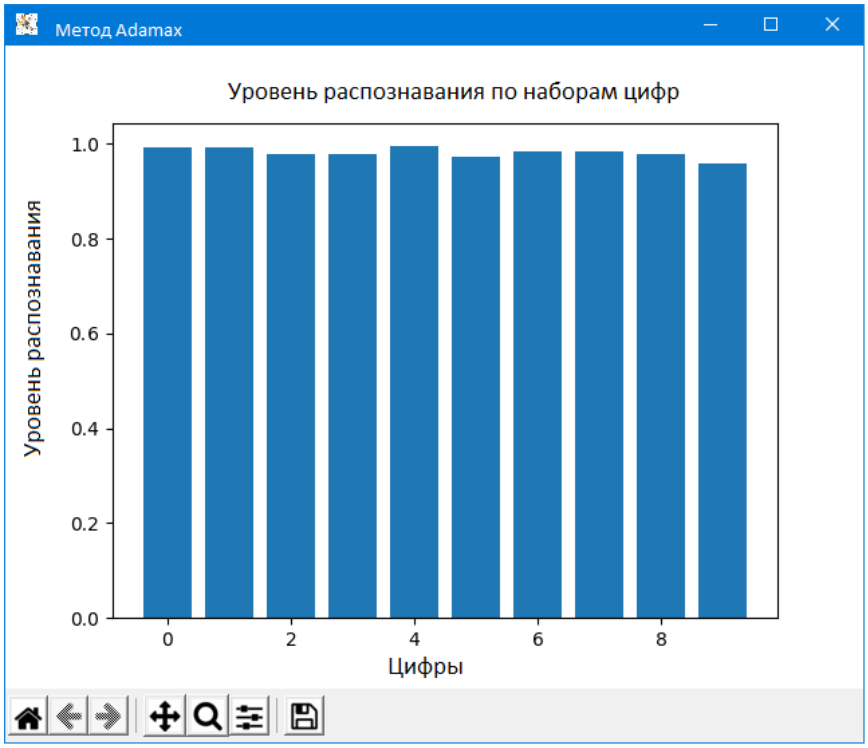

Рисунок 18 – Результаты распознавания наборов цифр по методу Adamax

Результаты распознавания по наборам цифр с использованием варианта оптимизатора с алгоритмом Adadelta (рисунок).

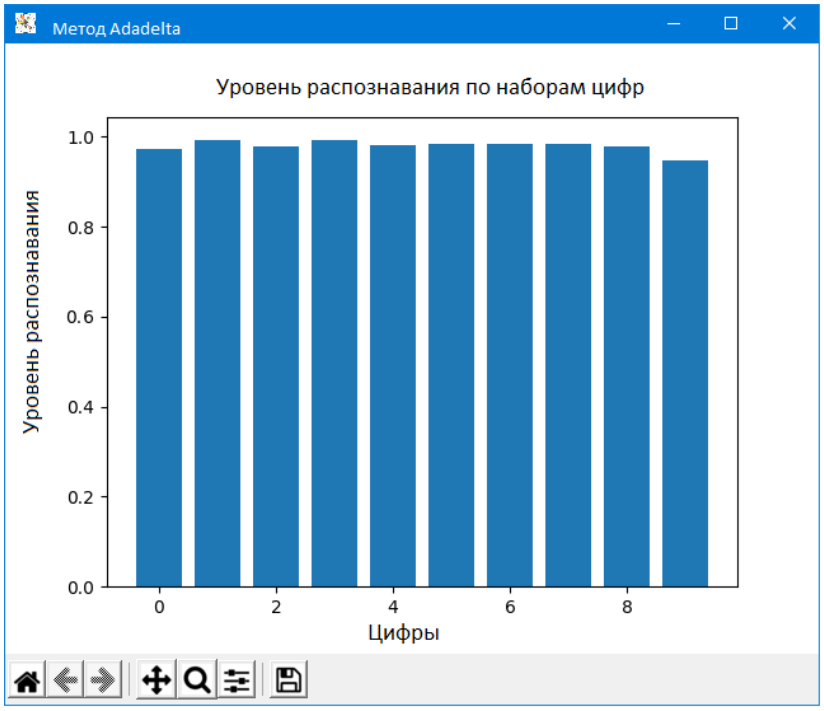

Рисунок 19 – Результаты распознавания наборов цифр по методу Adadelta

Результаты распознавания по наборам цифр с использованием варианта оптимизатора с алгоритмом RMSprop (рисунок).

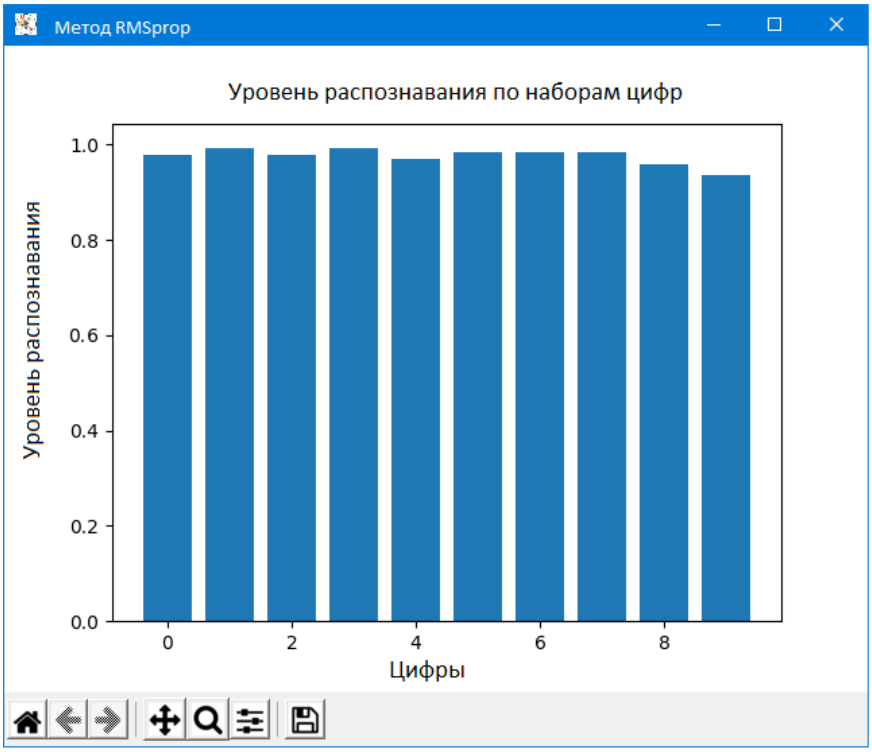

Рисунок 20 – Результаты распознавания наборов цифр по методу RMSprop

Результаты распознавания по наборам цифр с использованием варианта оптимизатора с алгоритмом SGD (рисунок).

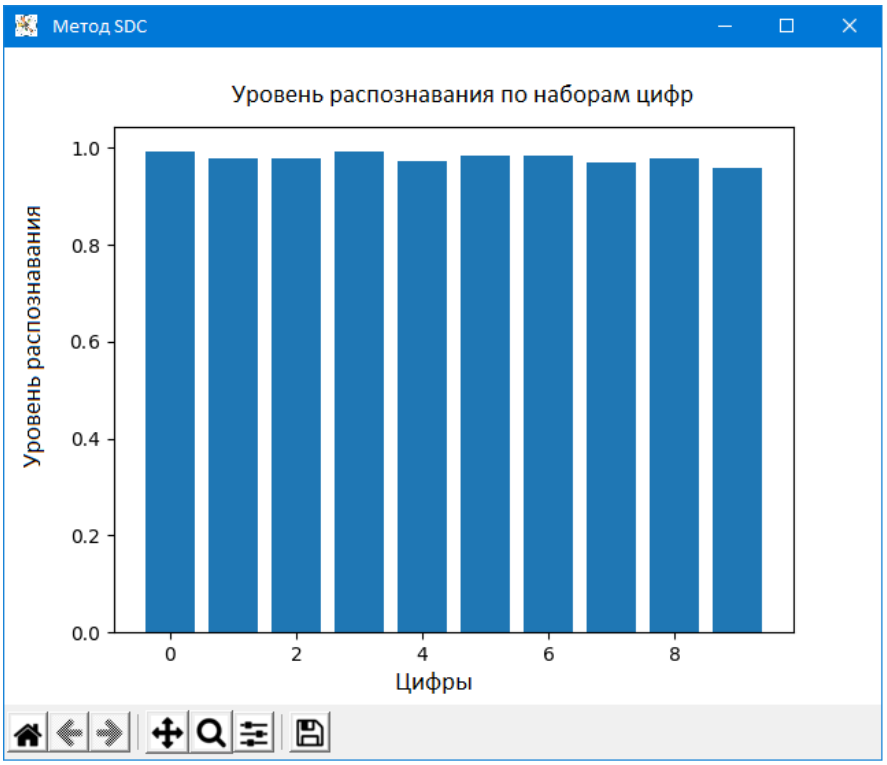

Рисунок 21 – Результаты распознавания наборов цифр по методу SGD

Результаты распознавания по наборам цифр с использованием варианта оптимизатора с алгоритмом Nadam (рисунок).

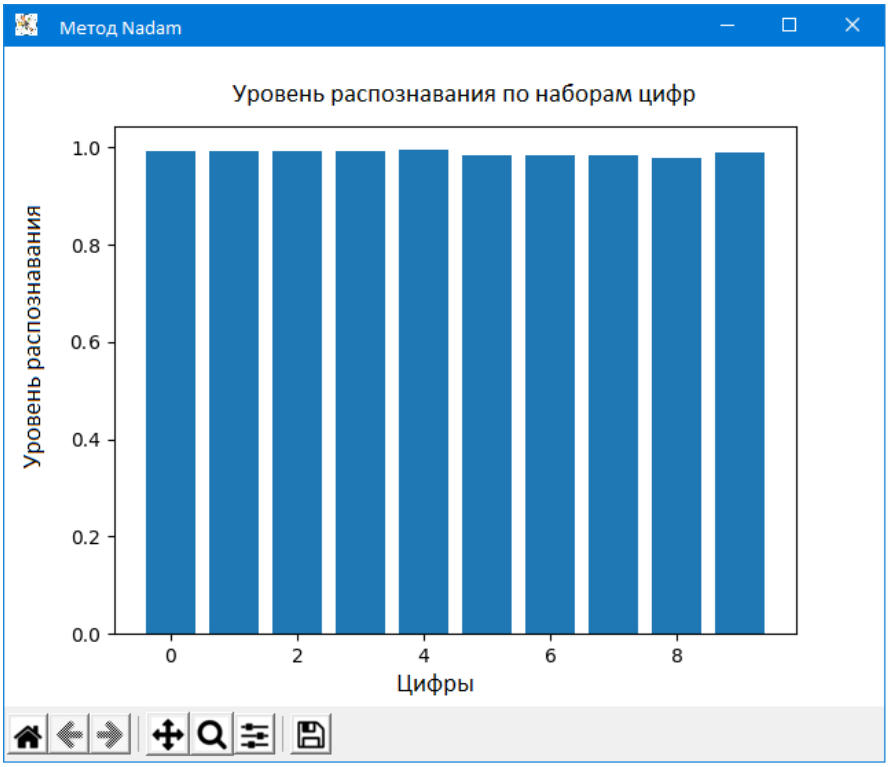

Рисунок 22 – Результаты распознавания наборов цифр по методу Nadam

Результаты распознавания по наборам цифр с использованием варианта оптимизатора с алгоритмом Adagrad (рисунок).

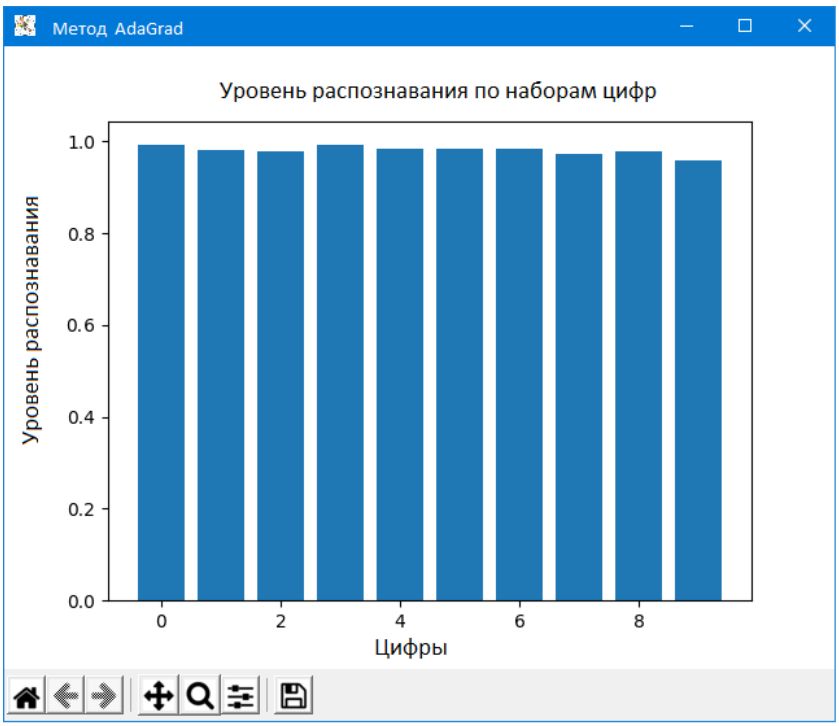

Рисунок 23 – Результаты распознавания наборов цифр по методу Adagrad

По результатам предварительного анализа сравнения полученных оценок наиболее оптимальными для выбранного пакета данных являются методы Nadam и Adamax. Однако распространять полученные результаты на другие пакеты данных будет не совсем корректно из-за небольших различий в полученных уровнях распознавания.

3.6 Оценка эффективности различных алгоритмов

Оценку результатов работы выбранных оптимизационных алгоритмов в ходе обучения нейронной сети можно провести также на основании данных точности вычислений и скорости выполнения обработки данных на примере тестовых данных, включающих 10000 изображений.

В ходе обучения сети выяснилось, что оптимальное количество эпох – это две, так как с большим количеством эпох эффективность сети уменьшается. Это происходит из-за того, что происходит перенасыщение сети обучающей выборкой, исправить это можно путем увеличения обучающей выборки.

Результаты, полученные в ходе тестирования для разных методов представлены ниже.

Adam

Точность вычислений по тестовой выборке:

Расп:98.40%: 100%|[redacted]| 10000/10000 [30:38<00:00, 5.44it/s]

Adamax

Точность вычислений по тестовой выборке:

Расп:98.65%: 100%|[redacted]| 10000/10000 [30:36<00:00, 5.48it/s]

AdaGrad

Точность вычислений по тестовой выборке:

Расп:98.72%: 100%|[redacted]| 10000/10000 [30:39<00:00, 5.21it/s]

Adadelta

Точность вычислений по тестовой выборке:

Расп:98.62%: 100%|[redacted]| 10000/10000 [30:35<00:00, 5.41it/s]

Nadam

Точность вычислений по тестовой выборке:

Расп:99.10%: 100%|[redacted]| 10000/10000 [45:38<00:00, 5.42it/s]

RMSprop

Точность вычислений по тестовой выборке:

Расп:97.10%: 100%|[redacted]| 10000/10000 [28:38<00:00, 5.21it/s]

SGD

Точность вычислений по тестовой выборке:

Расп:96.90%: 100%|[redacted]| 10000/10000 [27:38<00:00, 6.21it/s]

Данные результаты можно представить графически, что представлено на рисунках 24-25.

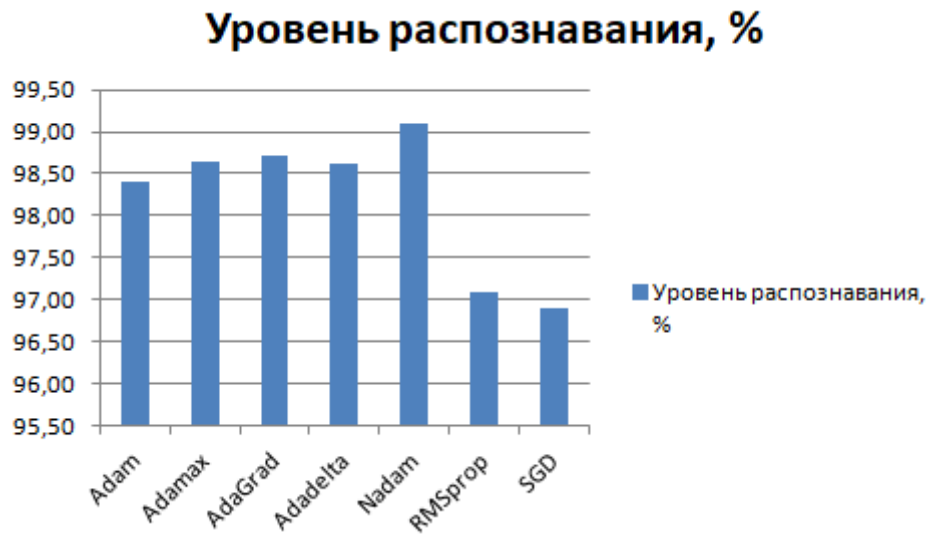

Рисунок 24 – Значения уровня распознавания каждого из методов

Рисунок 25 – Значения времени работы каждого из методов

Полученные результаты свидетельствуют о высокой точности распознавания для данного набора данных, лучшими также являются варианты модифицированных методов Adam - Nadam и Adamax. Тем не менее, различия в полученных данных могут считаться значительными только условно, скорее всего оптимального метода для всех наборов данных не существует, поэтому если по результатам обучения нейронной сети получена ошибка, серьёзным образом влияющая на принятие некоторого решения необходимо использовать несколько методов.

Заключение

В ходе работы, целью которой является разработка программного обеспечения для сравнения эффективности различных алгоритмов обучения сверточных нейронных сетей, были решены все поставленные задачи исследования.

1. Проведен анализ проблемы, включающий изучение принципов построения сверточных нейронных сетей и особенностей используемых алгоритмов. Описаны сверточные нейронные сети и отражены особенности используемых алгоритмов для их обучения. Рассмотрены глубокие нейронные сети их архитектура и методы обучения.

2. Приведены алгоритмы обучения, которые выбраны для проведения сравнения. Основными используемыми методами являются SGD RMSProp, Adagrad, Adadelta, Adam, Adamax, Nadam.

3. Осуществлен выбор критериев для сравнения алгоритмов обучения сверточных нейронных сетей.

4. Разработан проект и реализована система для сравнения алгоритмов обучения с использованием языка программирования Python и современных инструментов TensorFlow и библиотеки Keras, позволяющей проводить обучение сверточных нейронных сетей с применением разных оптимизационных алгоритмов.

5. Проведен эксперимент и описаны его результаты, полученные в ходе тестирования разработанного приложения, оценена эффективность различных алгоритмов обучения.

Полученные результаты свидетельствуют о высокой точности распознавания для данного набора данных. Полученные различия между алгоритмами могут считаться значительными только условно, скорее всего оптимального метода для всех наборов данных не существует, поэтому если по результатам обучения нейронной сети получена ошибка, серьёзным образом влияющая на принятие некоторого решения необходимо использовать несколько методов.

Список используемых источников

1. Бейдер Д. Чистый Python. Тонкости программирования для профи / Д. Бейдер. – СПб.: Питер, 2018. – 288 с.
2. Берри П. Изучаем программирование на Python / П. Берри. – М.: Эксмо, 2017. – 611 с.
3. Бредихин А.И. Алгоритмы обучения сверточных нейронных сетей / А.И. Бредихин // Вестник Югорского государственного университета. 2019. № 1 (52). С. 41-54.
4. Воевода А.А. Синтез нейронных сетей с несколькими переменными / А.А. Воевода, Д.О. Романников // Сборник научных трудов Новосибирского государственного технического университета. 2018. № 1 (91). С. 86-94.
5. Галимянов Ф.А. Сравнительный анализ алгоритмов реализации метода обратного распространения ошибки для обучения нейронных сетей / Ф.А. Галимянов // Научно-технический вестник Поволжья. 2020. № 2. С. 69-71.
6. Гафаров Ф.М Искусственные нейронные сети и приложения: учеб. пособие / Ф.М. Гафаров, А.Ф. Галимянов. – Казань: Изд-во Казан. ун-та, 2018. – 121 с.
7. Голубинский А. Н., Выбор архитектуры искусственной нейронной сети на основе сравнения эффективности методов распознавания изображений / А.Н. Голубинский, А.А. Толстых // Вестник ВИ МВД России. 2018. №1. С. 24-30.
8. Гузий Е.А. Сверточная нейронная сеть для разработки системы распознавания и классификации изображений / Е.А. Гузий, В.В. Федоренко // Молодежный научно-технический вестник. 2017. № 7. С. 51.
9. Землевский А. Д. Исследование архитектуры сверточных нейронных сетей для задачи распознавания образов / А.Д. Замлевский // Вестник науки и образования. 2017. №6 (30). С. 12-18.

10. Иванов И. А. Гибридный алгоритм обучения конволюционной нейронной сети / И.А. Иванов, Е.А. Сопов // Вестник СибГ. 2016. Том 17, № 4. С. 871–877.

11. Ле Мань Ха Свёрточная нейронная сеть для решения задачи классификации / Ле Мань Ха // Труды МФТИ. 2016. №3 (31). С. 56-62.

12. Любанович Б. Простой Python. Современный стиль программирования / Б. Любанович. –СПб.: Питер ,2016. – 480 с.

13. Манукян А.К. Анализ алгоритмов обучения нейронных сетей / А.К. Манукян // Аллея науки. 2019. Т. 1. № 1 (28). С. 935-939.

14. Попова Е.С. Программа и алгоритм сегментации и распознавания рукопечатных символов с помощью сверточных нейронных сетей / Е.С. Попова, В.Г. Спицын, Ю.А. Болотова // Computer Vision GraphiCon 2018. 24–27 сентября 2018, Томск, Россия. – С. 230-233.

15. Романников Д.О. О синтезе нейронных сетей / Д.О. Романников // Сборник научных трудов Новосибирского государственного технического университета. 2018. № 1 (91). С. 104-111.

16. Сикорский О.С. Обзор свёрточных нейронных сетей для задачи классификации изображений / О.С. Сикорский // Новые информационные технологии в автоматизированных системах. 2017. №20. С. 68-74.

17. Слаткин Б. Секреты Python: 59 рекомендаций по написанию эффективного кода / Б. Слаткин. – М.: Вильямс, 2016. – 274 с.

18. Созыкин А.В. Обзор методов обучения глубоких нейронных сетей / А.В. Созыкин // Вестник ЮУрГУ. Серия «Вычислительная математика и информатика». 2017. Т. 6. № 3. С. 28-59.

19. Ширяев, В.И. Финансовые рынки: Нейронные сети, хаос и нелинейная динамика / В.И. Ширяев. – М.: КД Либроком, 2016. – 232 с.

20. Шолле Ф. Глубокое обучение на Python / Ф. Шолле. – СПб.: Питер, 2018. – 486 с.

21. Hou B., Luo X., Wang SS., Jiao L., Zhang X. Polarimetric SAR images classification using deep belief networks with learning features // IEEE International Geoscience and Remote Sensing Symposium (IGARSS). 2015.

22. Rav`ı D., Wong Ch., Deligianni F. Deep Learning for Health Informatics // IEEE Journal of Biomedical and Health Informatics. 2017. Vol. 21. – No. 1. – P. 4–21.

23. Schmidhuber J. Deep Learning in Neural Networks: an Overview / J. Schmidhuber // Neural Networks. 2015. Vol. 1. P. 85–117.

24. Yann LeCun, Corinna Cortes, Christopher J.C. Burges. The MNIST database of handwritten digits. URL: http://yann.lecun.com/exdb/mnist/

25. TensorFlow Core v2.4.1. Python. tf.keras.optimizers. URL: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/

Приложение А

Режим доступа к разработанному программному обеспечению

Исходный код программы доступен для скачивания в электронном виде.

Режим доступа:

https://drive.google.com/drive/folders/1V8-1Q_bQs4_CIeEG6gb-sDrzsGE3jUOu?usp=sharing