МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики
Кафедра информационных технологий

Допустить к защите
И. о. заведующего кафедрой
канд. физ.-мат. наук, доц.
_____ О. В. Гаркуша
(подпись)
_____ 2019 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

ПРОГРАММНАЯ СИСТЕМА ГЕОЛОКАЦИИ ПО ИЗОБРАЖЕНИЮ ЗЕМНОЙ ПОВЕРХНОСТИ

Работу выполнил _____ И. А. Рыков
(подпись)

Направление подготовки 01.03.02 «Прикладная математика и информатика»

Направленность (профиль) «Системное программирование и компьютерные технологии (Математическое и программное обеспечение вычислительных машин)»

Научный руководитель
канд. физ.-мат. наук, доц. _____ В. В. Подколзин
(подпись)

Нормоконтролер
ст. преп. _____ А. В. Харченко
(подпись)

Краснодар
2019

РЕФЕРАТ

Выпускная квалификационная работа 78 с., 5 ч., 24 рис., 15 источников.

НЕЙРОННАЯ СЕТЬ, НЕЙРОСЕТЬ, ПЕРЦЕПТРОН, ОБУЧЕНИЕ, УЧИТЕЛЬ, ОШИБКА, ИЗОБРАЖЕНИЕ, ГЕОЛОКАЦИЯ, АНАЛИЗ, СПУТНИК, СНИМОК, ПОВЕРХНОСТЬ, СИСТЕМА, ИДЕНТИФИКАЦИЯ, КЛАССИФИКАЦИЯ, ОПТИМИЗАЦИЯ, МЕТОДЫ

Цель работы – разработка программного приложения, включающая создание и обучение нескольких нейронных сетей для геолокации через распознавание географических объектов на изображении земной поверхности. Объектом исследования является оптимизированная модель свёрточной нейронной сети для распознавания визуальных образов.

В результате проведенной работы глубоко изучены технологии, лежащие в основе искусственного интеллекта и разработана программная система геолокации, позволяющая идентифицировать объекты географических карт - спутниковых снимков земной поверхности с помощью каскада нейронных сетей.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире системы искусственного интеллекта внедряются повсеместно, во всех сферах жизни: в науке и технике, мультимедиа, экономике, рекламе и маркетинге, здравоохранении. В связи с колоссальным ростом объема всевозможных данных нейронные сети становятся незаменимым инструментом для их анализа.

В работе представлена программная система, позволяющая по снимку земной поверхности классифицировать находящиеся на нем объекты и определить их географическое наименование.

В основе системы лежит свёрточная нейронная сеть – специальная архитектура нейронных сетей, хорошо адаптированная для распознавания визуальных образов.

Цель данного проекта состоит в разработке программного приложения, включающей создание и обучение нескольких нейронных сетей для геолокации через распознавание географических объектов на изображении земной поверхности. Объект исследования – оптимизированная модель свёрточной нейронной сети для распознавания визуальных образов. Предметом исследования выступает разработка, обучение и тестирование описанной модели. Основная задача данного исследования состоит в необходимости классифицировать объект на снимке: водоём, населённый пункт, горный массив и идентифицировать его наименование.

Данная задача относится к классу задач компьютерного зрения и разбивается на несколько подзадач:

– обзор и классификация некоторых основных архитектур нейронных сетей;

– исследование методов обучения и оптимизации моделей искусственного интеллекта;

– выбор подходящей для решения основной задачи архитектуры сети и её исследование;

– проектирование собственной модели на основе выбранной архитектуры, её обучение, оптимизация и тестирование;

– анализ полученных результатов тестирования.

Проблема состоит в поиске альтернативного способа геолокации без привязки к спутниковым системам позиционирования вроде ГЛОНАСС и GPS.

В настоящее время актуальность разработки альтернативных способов позиционирования растет с каждым днем в связи с развитием современных технологий искусственного интеллекта, машинного обучения и робототехники.

Метод геолокации по изображению земной поверхности позволит создавать пилотируемые в автоматическом режиме летательные аппараты в условиях отсутствия спутникового сигнала, а также это позволит автоматически контролировать полет при смене ландшафта и уменьшить погрешность позиционирования.

Идея распознавания визуальных образов с помощью нейронных сетей не нова. Однако применение нейронных сетей в геолокации – достаточно молодая и неизученная область.

1 Нейронные сети и их классификация

Искусственная нейронная сеть (ИНС) — математическая модель, а также её программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей — сетей нервных клеток живого организма (рисунок 1) [1].

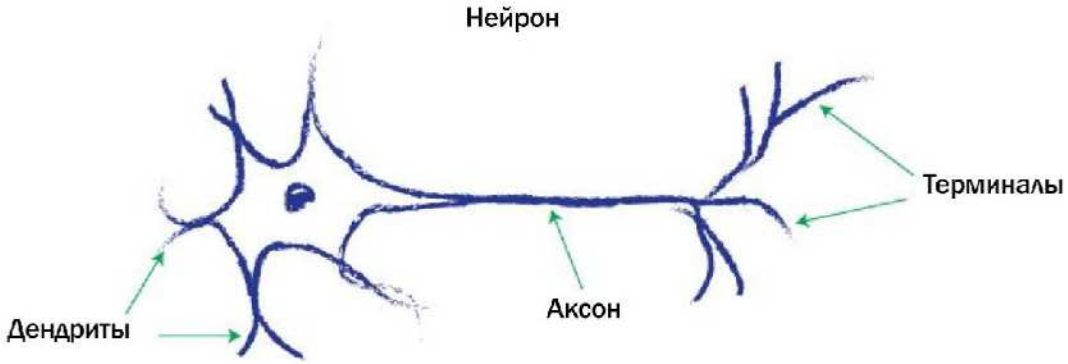

Рисунок 1 – Биологическое представление нейрона

Нейросети стали известны в 50-х годах двадцатого века. В период с 1943 по 1950 год были представлены миру первые две основополагающие работы. Одна из них, статья 1943 года двух выдающихся ученых - Уорена Маккалока и Уолтера Питтса освещала математическую модель нейронной сети, а в 1949 году канадским нейропсихологом Дональдом Хеббом была выпущена книга "Организация поведения", в которой было подробное описание процесса самообучения искусственной нейронной сети.

В 1957 году Фрэнком Розенблаттом был предложен перцептрон - математическая (компьютерная) модель обработки информации человеческим мозгом. Данная разработка уже в те годы умела прогнозировать погоду и распознавать образы (рисунок 2) [2].

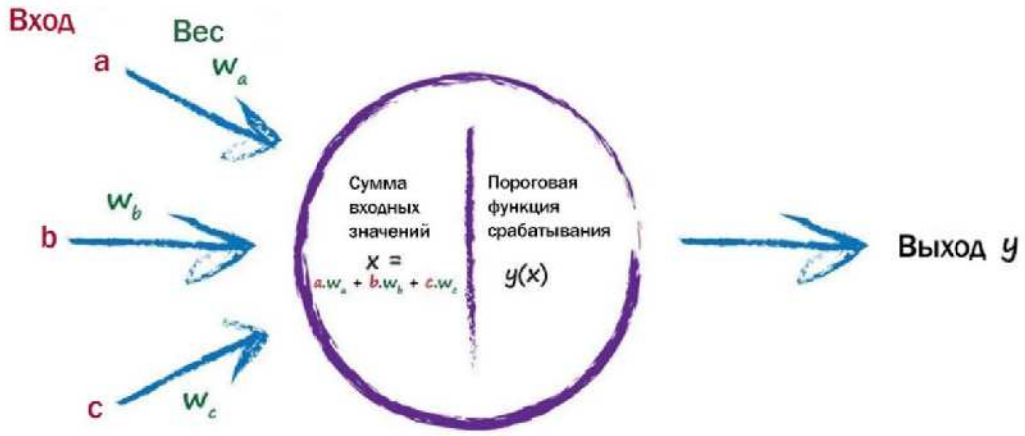

Рисунок 2 – Классическая модель искусственного нейрона

Возможность обучения — одно из главных преимуществ нейронных сетей перед традиционными алгоритмами. Технически обучение заключается в нахождении коэффициентов связей между нейронами. В процессе обучения нейронная сеть способна выявлять сложные зависимости между входными данными и выходными, а также выполнять обобщение. Это значит, что в случае успешного обучения сеть сможет вернуть верный результат на основании данных, которые отсутствовали в обучающей выборке, а также неполных и/или «зашумленных», частично искажённых данных [1].

Классификация искусственных нейронных сетей по применяемой структурной модели:

– перцептрон - нейронная сеть прямого распространения, то есть сигнал поступает на вход и движется только в прямом направлении без обратной связи (рисунок 3);

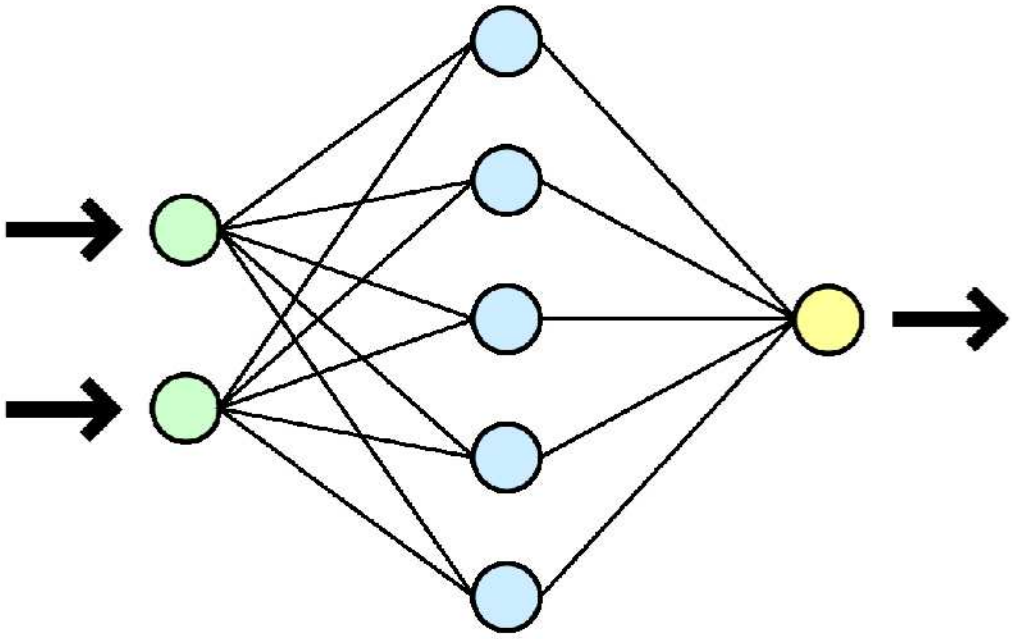

Рисунок 3 – Схема простой нейросети. Зелёным цветом обозначены входные нейроны, голубым — скрытые нейроны, жёлтым — выходной нейрон [1]

– рекуррентные нейронные сети - нейронные сети, где присутствует обратная связь, подразумевающая связь от логически более удаленного элемента к менее удаленному;

– свёрточные нейронные сети - нейронные сети прямого распространения, предназначенные в основном для эффективной классификации и обработки изображений.

1.1 Перцептрон

Перцептрон, или персептрон — математическая или компьютерная модель восприятия информации мозгом (кибернетическая модель мозга). Перцептрон - одна из первых моделей нейронных сетей. Несмотря на свою простоту, перцептрон способен учиться и решать достаточно сложные задачи. Основная математическая задача, которую способен решать перцептрон — линейное разделение произвольных нелинейных множеств, так называемое обеспечение линейной сепарабельности.

Перцептрон включает в состав три типа элементов, а именно: сигналы, поступающие от датчиков, передаются в ассоциативные элементы, а затем в реагирующие. Таким образом, перцептроны позволяют создать набор «ассоциаций» между входящими стимулами и необходимой реакцией на выходе. В биологическом плане это соответствует превращению, например, зрительной информации в физиологический ответ двигательных нейронов.

Согласно современной терминологии, перцептроны могут быть классифицированы как искусственные нейронные сети:

– с одним скрытым слоем,

– с пороговой передаточной функцией,

– с прямым распространением сигнала [3].

Элементарный перцептрон состоит из элементов трех типов: S-элементов, A-элементов и одного R-элемента. S-элементы — это слой сенсоров, или рецепторов. В физическом воплощении они отвечают, например, светочувствительным клеткам сетчатки глаза или фоторезисторам матрицы камеры. Каждый рецептор может находиться в одном из двух состояний — покоя или возбуждения, и только в последнем случае он передает единичный сигнал к следующему слою, ассоциативным элементам.

A-элементы называются ассоциативными, потому что каждому такому элементу, как правило, соответствует целый набор (ассоциация) S-элементов. A-элемент активизируется, как только количество сигналов от S-элементов на его входе превышает определенную величину θ.

Сигналы от возбужденных A-элементов, в свою очередь, передаются в сумматор R, причем сигнал от i-го ассоциативного элемента передается с коэффициентом. Этот коэффициент называется весом AR связи.

Так же, как и A-элементы, R-элемент подсчитывает сумму значений входных сигналов, умноженных на веса (линейную форму). R-элемент, а вместе с ним и элементарный перцептрон, выдает «1», если линейная форма

превышает порог θ, иначе на выходе будет «0». Математически, функцию, реализующую R-элемент, можно записать выражением вида (1):

$$f(x) = sign\left(\sum_{i=1}^{n} w_i x_i - \theta\right). \quad (1)$$

Если перцептрон получает на вход не дискретные, а непрерывные значения, то модель перцептрона можно представить в виде формулы (2):

$$y_i = f\left(\sum_{i=1}^{n} w_i x_i - \theta\right), \quad (2)$$

где

f(x) – функция активации нейрона, например, сигмоидная функция (рисунок 4);

w_i - вес синаптической связи нейрона;

x_i - входное значение нейрона;

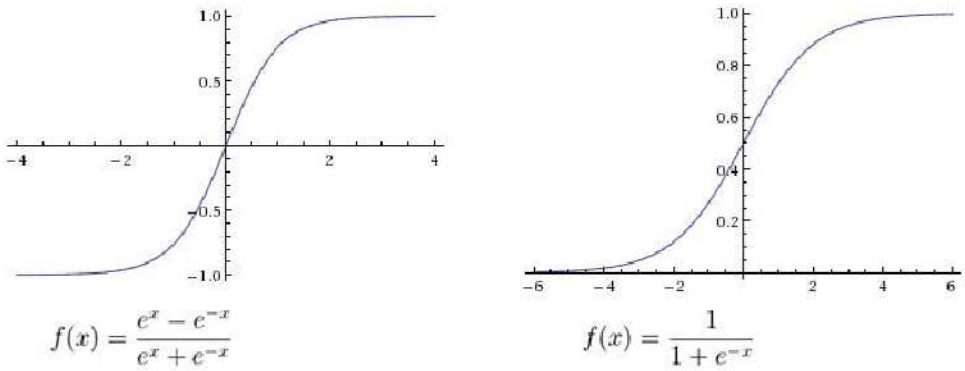

Рисунок 4 – Примеры сигмоидных активационных функций

Логическая схема перцептрона, где $S_1 \ldots S_8$ – слой рецепторов, $A_1 \ldots A_4$ - ассоциативные элементы, R_1 и R_2 – сумматоры, имеет вид, изображенный на рисунке 5:

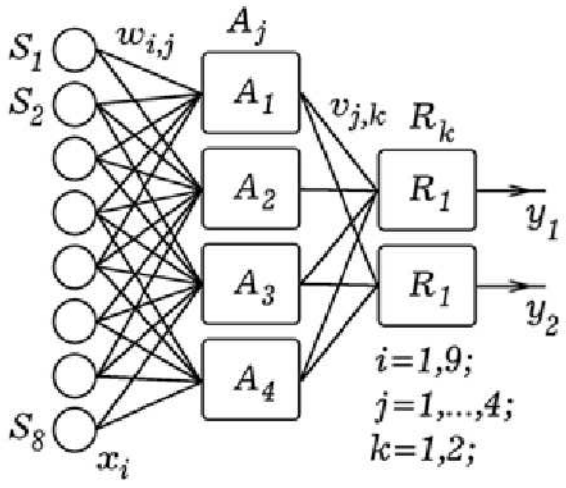

Рисунок 5 – Логическая схема элементарного перцептрона

1.2 Алгоритмы обучения нейронной сети

Основным свойством любой нейронной сети является способность к обучению. Процесс обучения является процедурой настройки весов и порогов с целью уменьшения разности между желаемыми (целевыми) и получаемыми векторами на выходе. Розенблатт классифицировал различные алгоритмы обучения перцептрона, называя их системами подкрепления.

Система подкрепления — это любой набор правил, на основании которых можно изменять с течением времени матрицу взаимодействия (или состояние памяти) перцептрона.

Описывая эти системы подкрепления и уточняя возможные их виды, Розенблатт основывался на идеях Д. Хебба об обучении, предложенных им в 1949 году, которые можно перефразировать в следующее правило, состоящее из двух частей.

Если два нейрона по обе стороны синапса (соединения) активизируются одновременно (то есть синхронно), то прочность этого соединения возрастает.

Если два нейрона по обе стороны синапса активизируются асинхронно, то такой синапс ослабляется или вообще отмирает.

Выделяют несколько методов обучения нейронной сети:

– обучение с учителем,

– обучение без учителя,

– метод обратного распространения ошибки [3].

1.2.1 Обучение с учителем

Классический метод обучения перцептрона — это метод коррекции ошибки. Он представляет собой такой вид обучения, при котором вес связи не изменяется до тех пор, пока текущая реакция перцептрона остается правильной. При появлении неправильной реакции вес изменяется на единицу, а знак (плюс/минус) определяется противоположным от знака ошибки.

Допустим, нужно обучить перцептрон различать два класса объектов так, чтобы при предъявлении объектов первого класса выход перцептрона был положителен (плюс 1), а при предъявлении объектов второго класса — отрицательным (минус 1) [3].

Перцептроном будем называть устройство, вычисляющее следующую систему функций (3):

$$\psi = \left[\sum_{i=1}^{n} w_{ij} x_i > \theta\right], \tag{3}$$

где

$j = 1, \dots, n$;

w_{ij} - веса перцептрона;

x_i - значения входных сигналов;

θ - порог срабатывания. Квадратные скобки означают переход от булевых значений к числовым.

Обучение персептрона заключается в подстройке весовых коэффициентов. Пусть имеется набор пар векторов $(x^\alpha, y^\alpha), \alpha = 1, \dots, p$, называемый обучающей выборкой. Нейронная сеть считается обученной на данной обучающей выборке, если при подаче на входы сети каждого вектора x^α на выходах всякий раз получается соответствующий вектор y^α.

Предложенный Розенблаттом метод обучения состоит в итерационной подстройке матрицы весов, последовательно уменьшающей ошибку в выходных векторах. Алгоритм включает несколько шагов:

Шаг 0: начальные значения всех весов нейронов W(t = 0) полагаются случайными.

Шаг 1: сети предъявляется входной образ x^α в результате формируется выходной образ $\widetilde{y^\alpha} \neq y^\alpha$.

Шаг 2: вычисляется вектор ошибки $\delta^\alpha = (y^\alpha - \widetilde{y^\alpha})$, совершаемой сетью на выходе. Дальнейшая идея состоит в том, что изменение вектора весовых коэффициентов в области малых ошибок должно быть пропорционально ошибке на выходе и равно нулю, если ошибка равна нулю.

Шаг 3: Вектор весов модифицируется по формуле (4):

$$W(t + \Delta T) = W(t) + \eta x^\alpha \cdot (\delta^\alpha)^T, \quad (4)$$

где

$0 < \eta < 1$ – коэффициент скорости обучения.

Шаг 4: шаги 1—3 повторяются для всех обучающих векторов. Один цикл последовательного предъявления всей выборки называется эпохой. Обучение завершается по истечении нескольких эпох: а) когда итерации

сойдутся, то есть вектор весов перестает изменяться, или б) когда полная, просуммированная по всем векторам абсолютная ошибка станет меньше некоторого малого значения [4].

1.2.2 Обучение без учителя

Рассмотренный выше алгоритм обучения нейронной сети подразумевает наличие некоего внешнего звена, предоставляющего сети наличие не только входных данных, но и эталонных выходных значений для обучающей выборки.

Главным преимуществом метода обучения без учителя является «самостоятельность» сети. Процесс обучения, как и в случае обучения с учителем заключается в подстройке весов синапсов (связей). Некоторые алгоритмы меняют структуру сети: количество нейронов и их связей, такие преобразования называются самоорганизацией сети.

Очевидно, подстройка весов связей между нейронами может производиться только на основании информации, доступной в нейроне: его состояниях и уже имеющихся весовых коэффициентов.

Д. Хеббом был разработан сигнальный метод обучения, в котором изменение весов синаптических связей происходит по следующему правилу (5):

$$w_{ij}(t) = w_{ij}(t-1) + \alpha \cdot y_i^{(n-1)} \cdot y_i^{(n)}, \quad (5)$$

где

$y_i^{(n-1)}$ – выходное значение нейрона i слоя $n-1$;

$y_i^{(n)}$ – выходное значение нейрона j слоя n;

$w_{ij}(t)$ и $w_{ij}(t-1)$ – весовые коэффициенты связи, соединяющей эти нейроны, на итерациях t и $t-1$;

α - коэффициент скорости обучения.

Существует также и дифференциальный метод обучения Хебба (6):

$$w_{ij}(t) = w_{ij}(t-1) + \alpha \cdot [y_i^{(n-1)}(t) - y_i^{(n-1)}(t-1)] \cdot [y_j^{(n)}(t) - y_j^{(n)}(t-1), (6)$$

где

$y_i^{(n-1)}(t)$ и $y_i^{(n-1)}(t-1)$ – выходное значение нейрона i слоя $n-1$ соответственно на итерациях t и $t-1$; $[y_j^{(n)}(t)$ и $y_j^{(n)}(t-1)$ – то же самое для нейрона j слоя n. Как видно из формулы (6), сильнее всего обучаются связи, соединяющие те нейроны, выходы которых наиболее быстро изменились в сторону увеличения.

Полный алгоритм обучения с применением приведенных формул (5) и (6) выглядит следующим образом.

Шаг 0: на стадии инициализации всем весовым коэффициентам присваиваются небольшие случайные значения.

Шаг 1: на входы сети подается входной образ, сигналы передаются согласно принципам классического прямого распространения, то есть для каждого нейрона вычисляется сумма его входных значений, к которой затем применяется функция активации нейрона. В результате формируется его выходное значение.

Шаг 2: на основании полученных выходных значений нейронов по формуле (5) или (6) производится изменение весовых коэффициентов.

Шаг 3: цикл возобновляется с первого шага, пока выходные значения сети не стабилизируются с заданной точностью. Применение этого нового способа определения завершения обучения, отличного от используемого для сети обратного распространения, обусловлено тем, что подстраиваемые значения весовых коэффициентов синапсов практически не ограничены. На втором шаге цикла попеременно предъявляются все образы из входного набора.

Следует отметить, что вид откликов на каждый класс входных образов неизвестен заранее и представляет собой произвольное сочетание состояний нейронов, обусловленное выбором случайных весовых значений на этапе инициализации. Вместе с тем сеть способна обобщать схожие образы, относя их к одному классу.

Другой алгоритм обучения без учителя – алгоритм Кохонена – предусматривает подстройку синапсов на основании их значений от предыдущей итерации (7):

$$w_{ij}(t) = w_{ij}(t-1) + \alpha \cdot \left[y_i^{(n-1)}(t) - w_{ij}(t-1) \right]. \quad (7)$$

Из приведенной выше формулы видно, что обучение сводится к минимизации разницы между входными сигналами нейрона, поступающими с выходов нейронов предыдущего слоя $y_i^{(n-1)}$, и весовыми коэффициентами его синапсов.

Полный алгоритм обучения имеет примерно такую же структуру, как в методе Хебба, но на втором шаге из всего слоя выбирается нейрон, значения синапсов которого максимально походят на входной образ, и подстройка весов по формуле (7) проводится только для него. Эта, так называемая, аккредитация может сопровождаться затормаживанием всех остальных нейронов слоя и введением выбранного нейрона в насыщение. Выбор такого нейрона может осуществляться, например, расчетом скалярного произведения вектора весовых коэффициентов с вектором входных значений. Максимальное произведение дает выигравший нейрон [5].

1.2.3 Метод обратного распространения ошибки

Для обучения многослойных сетей (изображение 6) рядом учёных, в том числе Д. Румельхартом, был предложен градиентный алгоритм обучения с

учителем, проводящий сигнал ошибки, вычисленный выходами перцептрона, к его входам, последовательно слой за слоем.

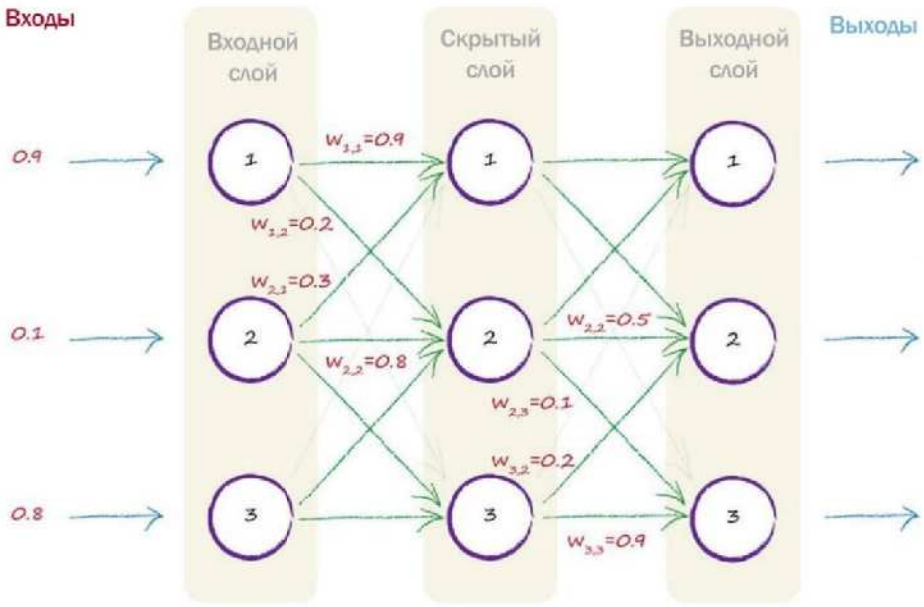

Рисунок 6 – Нейронная сеть с одним скрытым слоем

Сейчас это самый популярный метод обучения многослойных перцептронов. Его преимущество в том, что он может обучить все слои нейронной сети, и его легко просчитать локально. Однако этот метод является очень долгим, к тому же, для его применения нужно, чтобы передаточная функция нейронов (функция активации) была дифференцируемой. При этом в перцептронах пришлось отказаться от бинарного сигнала, и пользоваться на входе непрерывными значениями.

Для обучения многослойного перцептрона нельзя применить правило Розентблатта, так как для применения метода обучения с учителем требуются верные выходные значения, которые есть только для выходного слоя. Основная идея метода обратного распространения лежит в нахождении оценки ошибки для нейронов скрытых слоёв [6].

В общем случае оценка ошибки является функцией. Чем больше значение синаптической связи между нейроном скрытого слоя и выходным

нейроном, тем сильнее ошибка первого влияет на ошибку второго. Следовательно, оценку ошибки элементов скрытых слоев можно получить как взвешенную сумму ошибок последующих слоев. При обучении информация распространяется от низших слоев иерархии к высшим, а оценки ошибок, совершаемые сетью - в обратном направлении, что и отражено в названии метода.

Общая структура алгоритма сходна с описанными выше, но в данном случае усложняются формулы подстройки весов. Алгоритм метода обратного распространения ошибки имеет вид:

Шаг 0: начальные значения всех весов нейронов W(t = 0) полагаются случайными.

Шаг 1: сети предъявляется входной образ x^{α}, при этом формируется выходной образ y^{α}. При этом нейроны последовательно от слоя к слою функционируют по следующим формулам, где (8) – скрытый слой, (9) - выходной:

$$x_i = \sum_{i=1}^{n} w_{ij} {x_i}^{\alpha};\ y_j = f(x_i), \tag{8}$$

$$x_k = \sum_{i=1}^{n} v_{ij} y_j;\ y_k = f(x_k), \tag{9}$$

где

$f(x)$ – сигмоидная функция, определяемая выражением (10):

$$f(x) = \frac{1}{1 + e^{-1}}. \tag{10}$$

Шаг 2: функционал квадратичной ошибки сети для данного входного образа имеет вид (11):

$$E = \frac{1}{2}\sum_{k}(y_k - Y_k^{\alpha})^2. \quad (11)$$

Данный функционал подлежит минимизации. Классический градиентный метод оптимизации состоит в итерационном уточнении аргумента согласно формуле (12):

$$v_{jk}(t+1) = v_{jk}(t) - h\frac{\partial E}{\partial v_{jk}}, \quad (12)$$

где

h – параметр скорости обучения, выбираемый достаточно малым для сходимости алгоритма.

Функция ошибки в явном виде не содержит зависимости от веса v_{jk}, поэтому используются формулы неявного дифференцирования сложной функции (13-15):

$$\frac{\partial E}{\partial y_k} = \delta_k = (y_k - Y_k^{\alpha}), \quad (13)$$

$$\frac{\partial E}{\partial x_k} = \frac{\partial E}{\partial y_k}\cdot\frac{\partial y_k}{\partial x_k} = \delta_k \cdot y_k(1-y_k), \quad (14)$$

$$\frac{\partial E}{\partial v_{jk}} = \frac{\partial E}{\partial y_k}\cdot\frac{\partial y_k}{\partial x_k}\cdot\frac{\partial x_k}{\partial v_{jk_k}} = \delta_k \cdot y_k(1-y_k)\cdot y_j. \quad (15)$$

Здесь учтено полезное свойство сигмоидной передаточной функции $f(x)$: ее производная выражается только через само значение функции, $f'(x) = f(1-f)$. Таким образом, все необходимые величины для подстройки весов выходного слоя v получены.

Шаг 3: на этом шаге выполняется подстройка весов скрытого слоя. Градиентный метод по-прежнему дает (16):

$$W_{ij}(t+1) = W_{ij}(t) - h \cdot \frac{\partial E}{\partial W_{ij}}. \tag{16}$$

Вычисления производных выполняются по тем же формулам, за исключением некоторого усложнения формулы для ошибки δ_j (17-19):

$$\frac{\partial E}{\partial x_k} = \frac{\partial E}{\partial y_k} \cdot \frac{\partial y_k}{\partial x_k} = \delta_k \cdot y_k(1-y_k), \tag{17}$$

$$\frac{\partial E}{\partial y_j} = \delta_j = \sum_k \frac{\partial E}{\partial x_k} \cdot \frac{\partial x_k}{\partial y_k} = \sum_k \delta_k \cdot y_k(1-y_k) \cdot v_{jk}, \tag{18}$$

$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial x_j} \cdot \frac{\partial x_j}{\partial W_{ij}} = \delta_j \cdot y_j(1-y_j) \cdot X_i^{\alpha} = \left[\sum_k \delta_k \cdot y_k(1-y_k) \times \right.$$
$$\left. \times v_{jk}\right] \cdot \left[y_j(1-y_j) \cdot X_i^{\alpha}\right]. \tag{19}$$

При вычислении здесь δ_j и был применен принцип обратного распространения ошибки: частные производные берутся только по переменным последующего слоя. По полученным формулам модифицируются веса нейронов скрытого слоя. Если в нейронной сети имеется несколько скрытых слоев, процедура обратного распространения применяется последовательно для каждого из них, начиная со слоя, предшествующего выходному, и далее до слоя, следующего за входным. При этом формулы

сохраняют свой вид с заменой элементов выходного слоя на элементы соответствующего скрытого слоя.

Шаг 4: шаги 1-3 повторяются для всех обучающих векторов. Обучение завершается по достижении малой полной ошибки или максимально допустимого числа итераций, как и в методе обучения Розенблатта [6].

Из описания алгоритма видно, что обучение сводится к задаче оптимизации функционала градиентным методом. Однако данный метод можно модифицировать, используя для оптимизации более продвинутые вычислительные алгоритмы, например, с накоплением момента или адаптивной скоростью обучения.

1.3 Рекуррентные нейронные сети

Рекуррентные нейронные сети (англ. Recurrent neural network; RNN) — вид архитектуры нейронных сетей, подразумевающий наличие обратной связи. При этом под обратной связью понимается связь от логически более удалённого элемента к менее удалённому. Сеществование обратных связей позволяет запоминать и воспроизводить целые последовательности реакций на один стимул. С точки зрения программирования в таких сетях появляется аналог циклического выполнения, а с точки зрения систем — такая сеть эквивалентна конечному автомату. Такие особенности потенциально предоставляют множество возможностей для моделирования биологических нейронных сетей. Однако большинство возможностей на данный момент плохо изучены в связи с возможностью построения разнообразных архитектур и сложностью их анализа [7].

Пример схемы рекуррентной нейронной сети изображен на рисунке 7:

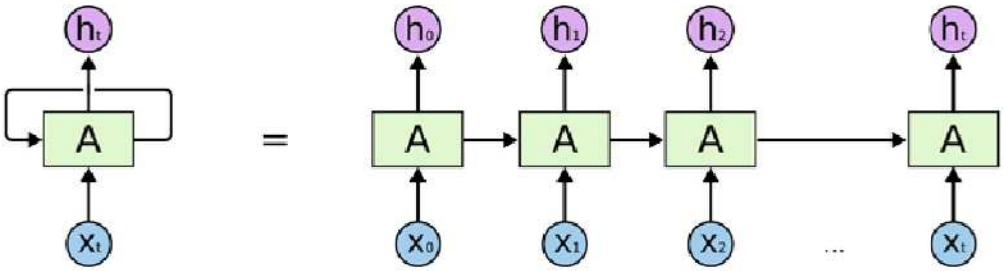

Рисунок 7 – Схема однослойной рекуррентной нейронной сети: на каждом цикле работы внутренний слой нейронов получает набор входных данных X и информацию о предыдущем состоянии внутреннего слоя A, на основании чего генерирует ответ h [8]

Если предполагается использование нейронной сети, например, для обработки текста или звука, то есть в общем случае – любой условно бесконечной последовательности, в которой важно не только содержание, но и порядок, в котором следует информация, то для этого следует прибегнуть к рекуррентной нейронной сети.

Противоположность рекуррентных сетей – сети прямого распространения, где информация передается в одном направлении от нейрона к нейрону в сторону выхода. В рекуррентных нейросетях нейроны обмениваются информацией между собой: например, вдобавок к новому фрагменту входных данных нейрон также получает некоторую информацию о предыдущем состоянии сети. Таким образом, в сети реализуется «память», что принципиально меняет характер ее работы и позволяет анализировать любые последовательности данных, в которых важно, в каком порядке идут значения — от звукозаписей до котировок акций.

Если нейронные сети прямого распространения представляют собой некоторую функцию, то рекуррентные нейронные сети можно назвать программой. В самом деле, память рекуррентных нейросетей делает их Тьюринг-полными: при правильном задании весов сеть может успешно эмулировать работу компьютерных программ [8].

Простая рекуррентная нейронная сеть или сеть Элмана состоит из трех слоёв - входного (распределительного) слоя, скрытого и выходного

(обрабатывающих) слоёв. При этом скрытый слой имеет обратную связь сам на себя. На рисунке 8 представлена схема нейронной сети Элмана:

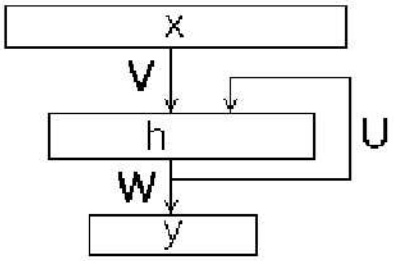

Рисунок 8 – Схема нейронной сети Элмана

В отличие от сети прямого распространения, входной образ рекуррентной сети — это не один вектор, а последовательность векторов $\{X_1 \dots X_n\}$. Векторы входного образа в заданном порядке подаются на вход, при этом новое состояние скрытого слоя зависит от его предыдущих состояний. Сеть Элмана можно описать следующими соотношениями (20, 21):

$$h(t) = f(V \cdot x(t) + U \cdot h(t-1) + b_h), \tag{20}$$

$$y(t) = g\big(W \cdot h(t) + b_y\big), \tag{21}$$

где

$x(t)$ - выходной вектор номера t;

$h(t)$ - состояние скрытого слоя для входа $x(t)$ $(h(0) = 0)$;

$y(t)$ - выход сети для входа $x(t)$;

U - матрица весов распределительного слоя;

W - квадратная матрица весов обратных связей скрытого слоя;

b_h - вектор сдвигов скрытого слоя;

V - матрица весов выходного слоя;

b_y - вектор сдвигов выходного слоя;

f - функция активации скрытого слоя;

g - функция активации выходного слоя [7].

Помимо рекуррентной сети Элмана существуют и другие разновидности нейронных сетей с обратной связью.

Нейронная сеть Хопфилда - полносвязная нейронная сеть с симметричной матрицей связей. В процессе работы динамика таких сетей сходится к одному из положений равновесия. Эти положения равновесия определяются заранее в процессе обучения, они являются локальными минимумами функционала, называемого энергией сети (в простейшем случае — локальными минимумами отрицательно определённой квадратичной формы на n-мерном кубе) [9].

Нейронная сеть Джордана - вид нейронных сетей, который получается из многослойного перцептрона, если на его вход подать, помимо входного вектора, выходной с задержкой на один или несколько тактов [10].

1.3.1 Метод обучения рекуррентной сети

Для обучения сети Элмана применяются те же градиентные методы, что и для сетей прямого распространения, но с определёнными модификациями для корректного вычисления градиента функции ошибки. Он вычисляется с помощью модифицированного метода обратного распространения, который носит название backpropagation through time (метод обратного распространения с разворачиванием сети во времени, BPTT). Идея метода - развернуть последовательность, превратив рекуррентную сеть в "обычную". Как и в методе обратного распространения для сетей прямого распространения, процесс вычисления градиента (изменения весов) происходит в три следующих этапа:

1) прямой проход - вычисляются состояния слоёв;

2) обратный проход - вычисляется ошибка слоёв;

3) вычисление изменения весов, на основе данных полученных на первом и втором этапах [7].

Алгоритм BPTT следующий:

Шаг 1 - прямой проход для каждого вектора последовательности $\{X_1 \dots X_n\}$:

вычисляются состояния скрытого слоя $\{s_1 \dots s_n\}$ и выходы скрытого слоя $\{s_1 \dots s_n\}$ по формулам (22, 23):

$$s(t) = Vx(t) + U \cdot h(t-1) + a, \tag{22}$$

$$h(t) = f\big(s(t)\big) \tag{23}$$

и вычисляется выход сети y (24):

$$y(n) = g(W \cdot h(n) + b). \tag{24}$$

Шаг 2 – обратный проход: вычисляется ошибка выходного слоя по формуле (25):

$$\delta_0 = y - d, \tag{25}$$

вычисляется ошибка скрытого слоя в конечном состоянии (26):

$$\delta_h(t) = W^T \cdot \delta_0 \odot f'\big(s(n)\big), \tag{26}$$

вычисляются ошибки скрытого слоя в промежуточных состояниях (27):

$$\delta_h(t) = U^T \cdot \delta_h(t)(t+1) \odot f'\big(s(n)\big), t = (1, \dots, n). \tag{27}$$

Шаг 3 – вычисление изменения весов (28-32):

$$\Delta W = \delta_0 \cdot \left(h(n)\right)^T, \tag{28}$$

$$\Delta b_y = \sum \delta_0, \tag{29}$$

$$\Delta V = \sum_t \delta_h(t) \cdot \left(x(t)\right)^T, \tag{30}$$

$$\Delta U = \sum_t \delta_h(t) \cdot \left(h(t-1)\right)^T, \tag{31}$$

$$\Delta b_h \sum \sum_t \delta_h(t). \tag{32}$$

2 Оптимизация нейронных сетей

Оптимизация в целом является чрезвычайно сложной задачей. Традиционно, машинное обучение избегало трудностей общей оптимизации путем тщательного подбора целевой функции и ограничений, чтобы гарантировать выпуклость задачи оптимизации. Даже выпуклая оптимизация не обходится без сложностей. В этом разделе рассматриваются несколько наиболее важных задач, связанных с оптимизацией для обучения глубоких моделей нейронных сетей.

2.1 Проблемы оптимизации нейронных сетей

2.1.1 Локальный минимум

Одна из наиболее важных особенностей задачи выпуклой оптимизации состоит в том, что она может быть сведена к проблеме поиска локального минимума. Любой локальный минимум гарантируется как глобальный минимум. Некоторые выпуклые функции имеют плоскую область в нижней части, а не единственную глобальную точку минимума, но любая точка в такой плоской области является приемлемым решением. При оптимизации выпуклой функции будет известно хорошее решение, если будет найдена критическая точка любого вида.

С невыпуклыми функциями, такими как нейронные сети, можно иметь много локальных минимумов. Действительно, практически любая глубокая модель нейронной сети, по существу, гарантированно имеет чрезвычайно большое количество локальных минимумов. Однако это не всегда является серьезной проблемой.

Нейронные сети и любые модели с несколькими эквивалентно параметризованными латентными переменными имеют несколько локальных минимумов из-за проблемы с идентификацией модели. Модель считается

идентифицируемой, если достаточно большая обучающая выборка может исключить все настройками модели, кроме одной. Модели со скрытыми переменными часто невозможно идентифицировать, потому что существует возможность получить эквивалентные модели, обменивая между ними переменные. Например, можно взять нейронную сеть и модифицировать первый слой, меняя i-й элемент входящего вектора весовых коэффициентов на j-й элемент. Если есть n слоёв размерности m, то существует $n!^m$ способов упорядочения скрытых элементов. Этот вид неидентифицируемости известен как симметрия весового пространства.

В дополнение к весовой симметрии пространства у многих видов нейронных сетей есть дополнительные причины неидентификации. Например, в любой выпрямленной линейной или максимальной сети можно масштабировать все входящие веса и смещения элементов с помощью некоторого α, если также масштабируются все ее исходящие веса на $\frac{1}{\alpha}$. Это означает, что - если функция оценки не включает в себя такие термины, как снижение веса, которые напрямую зависят от весов, а не от результатов моделей, - каждый локальный минимум прямой линейной или максимальной сети находится на $(m \times n)$ -мерной гиперболе эквивалентных локальных минимумов.

Эти проблемы идентификации модели означают, что оценочная функция нейронной сети может иметь чрезвычайно большое или даже неисчислимо бесконечное количество локальных минимумов. Однако все эти локальные минимумы, возникающие из-за неидентифицируемости, эквивалентны каждому другому по значению функции оценки. В результате эти локальные минимумы не являются проблемной формой невыпуклости.

Локальные минимумы могут быть проблематичными, если они имеют высокую стоимость по сравнению с глобальным минимумом. Можно построить небольшие нейронные сети, даже без скрытых элементов, которые имеют локальные минимумы с более высокой стоимостью, чем глобальный.

Если локальные минимумы с высокой стоимостью распространены, это может создать серьезную проблему для алгоритмов оптимизации на основе градиента.

У сетей, представляющих практический интерес, есть много локальных минимумов высокой стоимости, и встречаются ли с ними алгоритмы оптимизации, остается открытым вопросом. В течение многих лет большинство практиков считали, что локальные минимумы являются общей проблемой, связанной с оптимизацией нейронной сети. Сегодня, похоже, это не так. Проблема остается активной областью исследований, но эксперты теперь подозревают, что для достаточно больших нейронных сетей большинство локальных минимумов имеют невысокое значение оценочной функции, и что не так важно найти истинный глобальный минимум, как найти точку в пространстве параметров, которая имеет низкую, но не минимальную стоимость.

Многие практики приписывают почти все трудности с оптимизацией нейронной сети локальным минимумам. Но конкретные проблемы требуют более тщательного изучения. Если норма градиента не сжимается до незначительного размера, то проблема заключается не в локальных минимумах или каких-либо других критических точках. В многомерных пространствах конкретно установить, что локальные минимумы являются проблемой, может быть очень трудно. Многие структуры, кроме локальных минимумов, также имеют небольшие градиенты [11].

2.1.2 Плато, седловые точки и другие плоские области

Для многих многомерных невыпуклых функций локальные минимумы (и максимумы) фактически редки по сравнению с точкой другого типа с нулевым градиентом: седловой точкой. Некоторые точки вокруг седловой точки имеют большую стоимость, чем седловая, в то время как другие имеют более низкую стоимость. В седловой точке матрица Гёссе имеет как

положительные, так и отрицательные собственные значения. Точки, лежащие вдоль собственных векторов, связанных с положительными собственными значениями, имеют большую стоимость, чем седловая точка, в то время как точки, лежащие вдоль отрицательных собственных значений, имеют меньшее значение. Можно думать о седловой точке как о локальном минимуме вдоль одного сечения оценочной функции и локальном максимуме вдоль другого сечения.

Многие классы случайных функций проявляют следующее поведение: в пространствах с низкой размерностью локальные минимумы являются обычным явлением. В многомерных пространствах локальные минимумы редки, а седловые точки встречаются чаще. Для функции $f: \mathbb{R}^n \longrightarrow \mathbb{R}$ этого типа ожидаемое отношение числа седловых точек к локальным минимумам растет экспоненциально с n. Чтобы понять интуицию, стоящую за этим поведением, обратите внимание, что матрица Гессе в локальном минимуме имеет только положительные собственные значения.

Матрица Гессе в седловой точке имеет смесь положительных и отрицательных собственных значений. Можно представить, что знак каждого собственного значения генерируется путем подбрасывания монеты. В одном измерении легко получить локальный минимум, подбрасывая монету и получая требуемое. В n-мерном пространстве маловероятно, что броски всех n монет приведут к требуемому результату.

Важным свойством многих случайных функций является то, что собственные значения гессиана становятся более вероятными положительными, когда достигаются области с более низкой стоимостью. В аналогии с подбрасыванием монеты это означает, что с большей вероятностью нужный результат выпадет n раз при нахождении в критической точке с низкой стоимостью. Это также означает, что локальные минимумы, скорее всего, будут иметь скорее низкую, чем высокую стоимость. Критические точки с высокой стоимостью, скорее всего, будут седловыми. Критические

точки с чрезвычайно высокой стоимостью, скорее всего, будут локальными максимумами.

Это происходит для многих классов случайных функций. Это происходит и для нейросетей. Теоретически было показано, что небольшие автоэнкодеры (сети с прямой связью, обученные копировать свои входные данные на свой выход), не имеющие нелинейностей, имеют глобальные минимумы и седловые (рисунок 9) точки, но не содержат локальные минимумы с более высокой стоимостью, чем глобальный минимум.

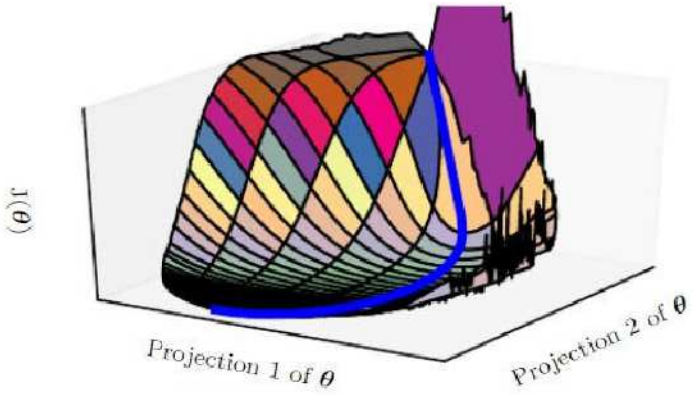

Рисунок 9 – Визуализация оценочной функции нейронной сети. Большая часть времени при обучении расходуется на пересечение относительно плоской долины.

Данное свойство распространяется на более глубокие сети без нелинейностей. Выход таких сетей является линейной функцией их входных данных, но они полезны для изучения в качестве модели нелинейных нейронных сетей, поскольку их функция потерь является невыпуклой функцией их параметров. Такие сети, по существу, представляют собой просто несколько матриц, составленных вместе. Обучение в этих моделях отражает многие качественные особенности, наблюдаемые при обучении глубоких моделей с нелинейными функциями активации. Реальные нейронные сети также имеют функции потерь, которые содержат очень много дорогих седловых точек.

Последствия распространения седловых точек для оптимизации первого порядка, алгоритмов, которые используют только градиентную информацию, остаются невыясненными. Градиент часто может стать очень маленьким вблизи седловой точки. С другой стороны, градиентный спуск эмпирически кажется способным уйти от седловых точек во многих случаях.

Для метода Ньютона седловые точки явно представляют собой проблему. Градиентный спуск предназначен для движения «под уклон» и не предназначен специально для поиска критической точки. Метод Ньютона, однако, предназначен для решения задачи, когда градиент равен нулю. Без соответствующей модификации он может перейти к седловой точке.

Распространение седловых точек в многомерных пространствах объясняет, почему методы второго порядка не смогли заменить градиентный спуск для обучения нейронной сети. В 2014 году был представлен бездонный метод Ньютона для оптимизации второго порядка, он значительно традиционную версию. Однако методы второго порядка по-прежнему трудно масштабировать до крупных сетей.

Есть и другие виды точек с нулевым градиентом, кроме минимумов и седловых точек. Максимумы во многом похожи на седловые точки с точки зрения. Максимумы многих классов случайных функций становятся экспоненциально редкими в многомерном пространстве, так же, как и минимумы.

Также могут существовать широкие плоские области постоянных значений. В этих местах градиент и гессиан равны нулю. Такие вырожденные местоположения составляют главные проблемы для всех алгоритмов численной оптимизации. В выпуклой задаче широкая плоская область должна состоять исключительно из глобальных минимумов, но в общей задаче оптимизации такая область может соответствовать большому значению целевой функции [11].

2.1.3 «Обрывы» и «взрывной» градиент

Многослойные нейронные сети часто имеют чрезвычайно крутые области, напоминающие обрыв (изображение 10).

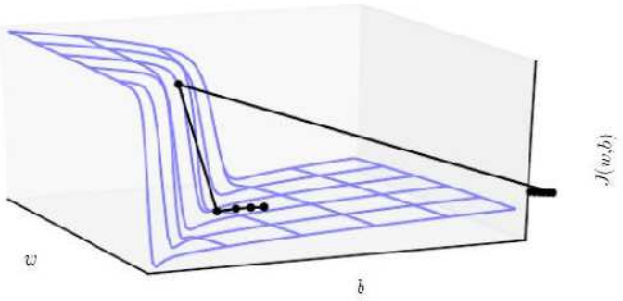

Рисунок 10 – Визуализация искажения при оптимизации сильно нелинейной глубокой сети.

Целевая функция для сильно нелинейных глубоких нейронных сетей или для рекуррентных нейронных сетей часто содержит резкие нелинейности в пространстве параметров, возникающие в результате многократного умножения. Эти нелинейности порождают очень высокие производные в некоторых местах. Когда параметры приближаются к такому региону, обновление спуска градиента может очень сильно исказить значения, что может привести к потере большей части проделанной работы по оптимизации.

Такие области могут быть опасными, независимо от того, с какой стороны к нему идет приближение, сверху или снизу, но, к счастью, наиболее серьезных последствий можно избежать, используя эвристику градиента.

Основная идея состоит в том, чтобы сигнализировать, что градиент задает не оптимальный размер шага, а только оптимальное направление в бесконечно малой области. Когда традиционный алгоритм градиентного спуска предлагает сделать очень большой шаг, эвристика ограничения градиента вмешивается, чтобы уменьшить размер шага, делая менее вероятным выход за пределы области, где градиент указывает направление

приблизительно наискорейшего спуска. Обрывистые структуры являются наиболее распространенными в оценочных функциях для рекуррентных нейронных сетей, потому что такие модели включают в себя умножение многих параметров, с одним параметром для каждого временного шага. Длинные временные последовательности, таким образом, влекут за собой огромное количество перемножений [11].

2.1.4 Долгосрочные зависимости

Еще одна проблема, которую должны преодолеть алгоритмы оптимизации нейронных сетей, когда вычислительный граф становится чрезвычайно глубоким. Сети прямого распространения со многими слоями имеют такие глубокие вычислительные графы. То же самое делают рекуррентные сети, которые строят очень глубокие вычислительные графы, многократно применяя одну и ту же операцию на каждом временном шаге длинной временной последовательности. Многократное применение одних и тех же параметров вызывает особенно выраженные трудности.

Например, предположим, что вычислительный граф содержит путь, который состоит из многократного умножения на матрицу W из t шагов, что эквивалентно W^t. Предположим, что имеет место собственное разложение $W = Vdiag(\lambda)V^{-1}$. В этом простом случае по формуле (33) нетрудно видеть, что

$$W^t = (Vdiag(\lambda)V^{-1})^t = Vdiag(\lambda)^t\, V^{-1}. \qquad (33)$$

Любые собственные значения λ_i, которые не близки к абсолютному значению 1, либо резко увеличатся (т. н. «взрыв»), если они будут больше 1 по величине, либо исчезнут, если они будут меньш е 1 по величине. Проблема «взрыва» и исчезновения градиента относится к тому факту, что градиенты

через такой график также масштабируются в соответствии с $diag(\lambda)^t$. Исчезающие градиенты затрудняют понимание того, в каком направлении должны двигаться параметры, чтобы улучшить оценочную функцию, в то время как взрывные градиенты могут сделать обучение нестабильным.

Описанные ранее структуры, которые мотивируют отсечение градиента, являются примером явления взрыва градиента. Повторное умножение на каждый временной шаг очень похоже на степенной метод, используемый для нахождения наибольшего собственного значения матрицы W и соответствующего собственного вектора. С этой точки зрения неудивительно, что $x^\top W^\top$ в итоге отбросит все компоненты x, которые ортогональны главному собственному вектору матрицы W.

Рекуррентные сети используют одну и ту же матрицу W^t каждый раз, но в сетях с прямой связью этого не происходит, поэтому даже очень глубокие сети с прямой связью могут в значительной степени избежать проблемы исчезновения и взрыва градиента [11].

2.1.5 Неточный градиент

Большинство алгоритмов оптимизации разработаны с предположением, что имеется доступ к точному градиенту или матрице Гессе. На практике обычно существует только зашумленная или даже необъективная оценка этих величин. Почти каждый алгоритм глубокого обучения опирается на выборочные оценки, по крайней мере, если использовать небольшие порции обучающих примеров для вычисления градиента.

В других случаях целевая функция, которую нужно минимизировать, фактически неразрешима. Когда целевая функция не вычислима, как правило, ее градиент неразрешим. В таких случаях возможно только приблизить градиент. Эти проблемы в основном связаны с более продвинутыми моделями.

Различные алгоритмы оптимизации нейронной сети предназначены для учета недостатков в оценке градиента. Можно также избежать этой проблемы, выбрав суррогатную функцию потерь, которую легче аппроксимировать, чем истинную функцию [11].

2.1.6 Плохое соответствие между локальной и глобальной структурами сети

Многие проблемы соответствуют свойствам функции потерь в одной точке - может быть трудно сделать один шаг, если $J(\theta)$ плохо обусловлена в текущей точке θ, или если θ находится в некоторой обрывистой области, или если точка θ является седловой, скрывающей возможность для градиентного спуска.

Можно преодолеть все эти трудности в одной точке и при этом все еще эффективно, если направление, которое приводит к наибольшему улучшению на локальном уровне, не указывает на отдаленные области с гораздо меньшей стоимостью (рисунок 11).

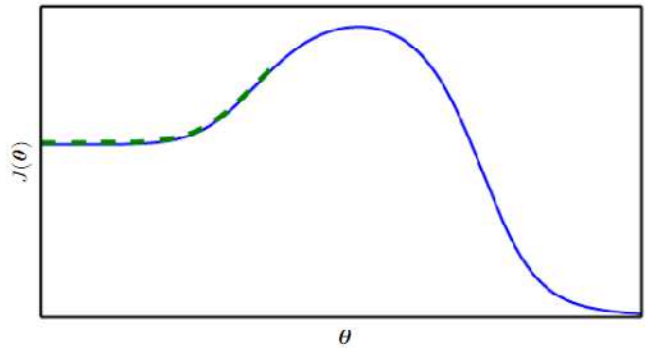

Рисунок 11 – Траектория обучения тратит большую часть времени на преодоление широкой дуги вокруг структуры в форме горы.

Имеет место утверждение, что большая часть времени обучения обусловлена длиной траектории, необходимой для достижения решения.

В многомерном пространстве алгоритмы обучения часто могут обходить такие подъемы, как изображен на рисунке, но траектория, связанная с этим, может быть длинной и приводить к чрезмерным затратам времени на обучение.

Большая часть исследований проблем оптимизации была сосредоточена на том, достигает ли обучение глобального минимума, локального минимума или седловой точки, но, по сути, нейронные сети не достигают какой-либо критической точки. Нейронные сети часто не достигают области небольшого градиента. Действительно, такие критические точки даже не обязательно существуют. Например, функция потерь $\log p\,(y \mid x; \theta)$ может не иметь точки глобального минимума и вместо этого асимптотически приближаться к некоторому значению, когда модель становится более обученной. Для классификатора с дискретным y и $p(y \mid x)$, вычисляемым softmax-функцией, отрицательная логарифмическая вероятность может быть произвольно близка к нулю, если модель способна правильно классифицировать каждый пример в обучающем наборе, но на самом деле невозможно достичь значения, равного нулю. Аналогично, модель вещественных значений $p(y \mid x) = N(y; f(\theta), \beta^{-1})$ может иметь вероятность отрицательного логарифма, которая асимптотически равна отрицательной бесконечности - если $f(\theta)$ способна правильно предсказать все целевые значения y обучающей выборки, алгоритм обучения будет увеличивать β бесконечно.

Будущие исследования должны будут способствовать дальнейшему пониманию факторов, которые влияют на длину траектории обучения и лучше характеризуют результаты данного процесса.

Многие существующие направления исследований направлены на то, чтобы найти хорошие начальные точки для задач, которые имеют сложную глобальную структуру, а не на разработку алгоритмов, использующих нелокальные движения [11].

Градиентный спуск и, по существу, все алгоритмы обучения, которые эффективны для обучения нейронных сетей, основаны на выполнении небольших локальных передвижений. Данные локальные перемещения может быть трудно вычислить, исходя из описанных выше проблем. Можно вычислить некоторые свойства рассматриваемой функции, такие как ее градиент, только приблизительно, со смещением или дисперсией в оценке правильного направления. В этих случаях локальный спуск может определять или не определять разумно короткий путь к правильному решению, но на самом деле нельзя следовать по пути локального спуска. Целевая функция может иметь проблемы такие, как вырожденность гессиана или прерывистые градиенты, в результате чего область, в которой градиент обеспечивает хорошую модель целевой функции, будет очень маленькой. В этих случаях локальный спуск с шагом размера ε может определить достаточно короткий путь к решению, но можно рассчитать только направление локального спуска с шагами размера $\delta \ll \varepsilon$. В данных случаях локальный спуск может определить путь к решению, но путь содержит много шагов, поэтому следование по нему сопряжено с большими вычислительными затратами. Иногда локальная информация не дает никакого руководства, например, когда функция имеет широкую плоскую область или если удается оказаться точно в критической точке (обычно последний сценарий происходит только с методами, которые решают явно для критических точек, такими как метод Ньютона). В этих случаях локальный спуск вообще не определяет пути к решению. В других случаях локальные перемещения могут быть слишком «жадными» и вести по пути, который движется вниз, но далеко от любого решения, или по неоправданно длинной траектории к решению. В настоящее нет точного понимания, какие из этих проблем являются наиболее актуальными для задач, связанных с оптимизацией нейронной сети, и это является областью активных исследований [11].

2.1.7 Теоретические пределы оптимизации

Некоторые теоретические результаты показывают, что существуют ограничения на производительность любого алгоритма оптимизации, который мы можно было бы разработать для нейронных сетей. Как правило, эти суждения мало влияют на использование нейронных сетей на практике.

Некоторые теоретические результаты применимы только тогда, когда элементы нейронной сети выводят дискретные значения. Большинство нейронных сетей выводят плавно увеличивающиеся значения, что делает возможной оптимизацию с помощью локального поиска. Другие теоретические результаты показывают, что существуют проблемные классы, которые неразрешимы, но трудно сказать, попадает ли конкретная проблема в этот класс. Некоторые результаты показывают, что найти решение для сети заданного размера трудно, но на практике можно легко найти решение, используя более крупную сеть, для которой гораздо больше параметров настройки соответствуют приемлемому решению. Более того, в контексте обучения нейронной сети обычно стоит задача нахождения не точного минимума функции, а уменьшения её значения в такой степени, чтобы получить низкую ошибку обобщения. Теоретический анализ того, может ли алгоритм оптимизации выполнить эту задачу, чрезвычайно труден. Поэтому разработка более реалистичных границ производительности алгоритмов оптимизации остается важной целью для машинного обучения.

2.2 Основные методы оптимизации нейронных сетей

Помимо классического метода оптимизации путем градиентного спуска существуют также более продвинутые методы, позволяющие увеличить скорость и точность обучения сети.

2.2.1 Стохастический градиентный спуск

Стандартный (или «пакетный», «batch») градиентный спуск для корректировки параметров модели использует градиент, который вычисляется как сумма градиентов оценочной функции, вызванных каждым обучающим примером. Вектор параметров изменяется в направлении антиградиента с заданным шагом – коэффициентом скорости обучения. Поэтому, чтобы изменить параметры, нужен один полный проход по обучающей выборке на каждой итерации, что на наборах больших размеров приводит к значительным временным затратам. Стохастический градиентный спуск призван исправить этот недостаток, путем внесения модификации в стандартный градиентный спуск. Данная модификация заключается в том, что значение градиента аппроксимируется градиентом оценочной функции, вычисленным только на одном элементе обучающей выборки на каждой итерации. Для нахождения оптимального решения методом стохастического градиентного спуска необходимо гораздо больше итераций, чем при использовании стандартного градиентного спуска, однако эти итерации требуют намного меньше временных затрат.

Стохастический градиентный спуск (SGD) и его варианты являются, вероятно, наиболее используемыми алгоритмами оптимизации для машинного обучения в целом и для глубокого обучения в частности. Важным параметром для алгоритма SGD является скорость обучения. Теоретически предполагается использование метода стохастического градиентного спуска с фиксированной скоростью обучения ε. На практике необходимо постепенно снижать скорость обучения с течением времени, поэтому скорость обучения на итерации k обозначается как ε_k.

Это связано с тем, что оценочная функция градиента SGD вводит источник шума (случайная выборка из обучающих примеров), который не исчезает, даже когда достигается минимум. Для сравнения истинный градиент функции общей стоимости становится небольшим по мере достижения

минимума при использовании пакетного градиентного спуска, поэтому обычный градиентный спуск может использовать фиксированную скорость обучения. Достаточное условие, гарантирующее сходимость SGD, выражается формулами (34) и (35):

$$\sum_{k=1}^{\infty} \varepsilon_k = \infty, \tag{34}$$

$$\sum_{k=1}^{\infty} \varepsilon_k^2 < \infty. \tag{35}$$

На практике принято, что скорость обучения уменьшается линейно до итерации τ (36):

$$\varepsilon_k = (1 - \alpha)\varepsilon_0 + \alpha\varepsilon_\tau. \tag{36}$$

Здесь $\alpha = \frac{k}{\tau}$. После итерации τ обычно оставляют ε постоянной.

Скорость обучения может быть выбрана методом проб и ошибок, но обычно лучше выбирать ее путем мониторинга кривых обучения, которые отображают целевую функцию как функцию времени. Это скорее искусство, чем наука, и большинство указаний по этому вопросу следует воспринимать с некоторым скептицизмом. При использовании линейного изменения коэффициента скорости обучения можно выбрать следующие параметры: ε_0, ε_τ и τ. Обычно τ можно задать равным количеству итераций, необходимое для нескольких сотен проходов по обучающей выборке. Значение ε_τ должно быть примерно равно одному проценту от значения ε_0. Главный вопрос - как установить ε_0. Если оно слишком велико, кривая обучения покажет сильные колебания, причем оценочная функция часто значительно увеличивается. Небольшие колебания не являются проблемой, особенно если обучение

производится с использованием стохастической оценочной функции, такой как функция, возникающая в результате использования так называемого исключения (англ. *dropout*) – сброса весов некоторых нейронов к нулю с вероятностью p во избежание переобучения сети. Если скорость обучения слишком низкая, обучение идет медленно, а если начальная скорость обучения слишком низкая, обучение может застрять в точке с высокой стоимостью.

Как правило, оптимальная начальная скорость обучения, с точки зрения общего времени обучения и конечной оценки, выше, чем скорость обучения, которая дает наилучшую производительность примерно после первых 100 итераций. Поэтому, как правило, лучше всего отслеживать первые несколько итераций и использовать скорость обучения, которая выше, чем самая эффективная в настоящее время скорость обучения, но не настолько высока, чтобы вызвать серьезную нестабильность. Наиболее важное свойство SGD заключается в том, что время вычислений на обновление не увеличивается с количеством обучающих примеров. Это позволяет приближаться к оптимальному решению, даже когда количество обучающих примеров становится очень большим. Для достаточно большого набора данных стохастический градиент может сходиться с точностью до некоторой фиксированной погрешности своей конечной ошибки тестового набора, прежде чем он обработает весь обучающий набор. Для изучения скорости сходимости алгоритма оптимизации обычно измеряют избыточную ошибку $J(\theta) - min_{\theta} J(\theta)$, на которую функция текущих затрат превышает минимально возможную оценку. Когда SGD применяется к выпуклой задаче, избыточная ошибка составляет $O\left(\frac{1}{\sqrt{k}}\right)$ после k итераций, в то время как в сильно выпуклом случае это значение равно $O\left(\frac{1}{k}\right)$. Данные границы не могут быть улучшены, если не приняты дополнительные условия. Пакетный градиентный спуск имеет лучшую скорость сходимости, чем стохастический градиентный спуск в теории. Однако неравенство Крамера-Рао утверждает, что ошибка обобщения не может уменьшаться быстрее, чем $O\left(\frac{1}{k}\right)$. Имеет

место утверждение, что, следовательно, может быть нецелесообразно проводить алгоритм оптимизации, который сходится быстрее, чем $O\left(\frac{1}{k}\right)$ для задач машинного обучения, - более быстрая сходимость, вероятно, соответствует перегрузке. В случае больших наборов данных способность SGD обеспечивать быстрый начальный прогресс при оценке градиента для очень немногих примеров перевешивает его медленную асимптотическую сходимость [11].

2.2.2 Метод импульса

Хотя стохастический градиентный спуск остается популярной стратегией оптимизации, обучение иногда может быть достаточно медленным. Метод импульса (Поляк Б. Т., 1964) разработан для ускорения обучения, особенно в условиях высокой кривизны, небольших, но постоянных градиентов или шумовых градиентов. Алгоритм импульса накапливает экспоненциально убывающее скользящее среднее из прошлых градиентов и продолжает двигаться в их направлении. Влияние импульса проиллюстрировано на рисунке 12.

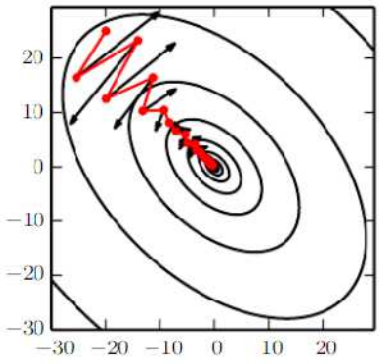

Рисунок 12 – Визуализация влияния импульса при оптимизации. Контурные линии изображают квадратичную функцию потерь с плохо обусловленной матрицей Гессе. Красная линия указывает путь, по которому следует процесс оптимизации.

Формально алгоритм импульса вводит переменную v, которая играет роль быстроты — это направление и скорость, с которой параметры перемещаются через пространство параметров. Скорость устанавливается как экспоненциально убывающее среднее отрицательного градиента. Название метода происходит из физической аналогии, в которой отрицательный градиент представляет собой силу, движущую точку через пространство, согласно законам Ньютона. Импульсом в физике является масса, умноженная на скорость. В алгоритме обучения с импульсом принимается некоторая единица массы, поэтому вектор v также можно рассматривать как импульс точки. Гиперпараметр $\alpha \in [0,1)$ определяет, как быстро вклады предыдущих градиентов экспоненциально затухают. Значение α можно понимать как коэффициент сохранения. Правило обновления определяется выражениями (37) и (38):

$$v \leftarrow \alpha v - \varepsilon \nabla \theta \left(\frac{1}{m} \sum_{i-1}^{m} L\left(f\left(x^{(i)}; \theta\right), y^{(i)}\right)\right), \tag{37}$$

$$\theta \leftarrow \theta + v. \tag{38}$$

Величина v накапливает элементы градиента (39):

$$\nabla \theta \left(\frac{1}{m} \sum_{i-1}^{m} L\left(f\left(x^{(i)}; \theta\right), y^{(i)}\right)\right). \tag{39}$$

Чем параметр α больше относительно y, тем больше предыдущих градиентов влияют на текущее направление.

До этого размер шага был просто нормой градиента, умноженной на коэффициент скорости обучения. Теперь размер шага зависит от того, насколько велика и насколько выровнена последовательность градиентов. Размер шага является наибольшим, когда многие последующие градиенты

указывают в одном и том же направлении. Если алгоритм импульса всегда получает градиент g, он будет ускоряться в направлении $-g$ до достижения конечной скорости, где размер каждого шага равен (40):

$$\frac{\varepsilon\|g\|}{1-\alpha}. \tag{40}$$

Таким образом, полезно рассматривать гиперпараметр импульса как $\frac{1}{1-\alpha}$. Например, $\alpha = 0{,}9$ соответствует умножению максимальной скорости на 10 относительно алгоритма градиентного спуска. Общие значения α, используемые на практике, включают 0.5, 0.9 и 0.99. Как и скорость обучения, α может также адаптироваться со временем, обычно начиная с небольшого значения и позже увеличиваясь. Адаптация α во времени менее важна, чем сокращение ε во времени.

Можно рассматривать алгоритм импульса как имитирующий материальную точку, подверженную динамике Ньютона. Физическая аналогия может помочь интуитивно понять, как ведут себя алгоритмы импульсного и градиентного спуска. Положение точки в любой момент времени определяется как $\theta(t)$. На нее действует сила $f(t)$. Эта сила заставляет точку ускоряться (41):

$$f(t) = \frac{d^2}{dt^2}\theta(t). \tag{41}$$

Вместо того, чтобы рассматривать это как дифференциальное уравнение второго порядка, можно ввести переменную $v(t)$, представляющую скорость точки в момент времени, и переписать уравнение (41) как систему дифференциальных уравнений (42) и (43) первого порядка:

$$v(t) = \frac{d}{dt}\theta(t), \tag{42}$$

$$f(t) = \frac{d}{dt}v(t). \tag{43}$$

Алгоритм импульса состоит из решения дифференциального уравнения путем численного моделирования. Простой численный метод решения дифференциальных уравнений — метод Эйлера, который просто состоит из моделирования динамики, определяемой уравнением, с помощью небольших, конечных шагов в направлении каждого градиента. Одна сила пропорциональна отрицательному градиенту оценочной функции: $-\nabla_\theta J(\theta)$. Сила, которая толкает точку вниз по поверхности, пропорциональна отрицательному градиенту оценочной функции: $-\nabla_\theta J(\theta)$. Алгоритм градиентного спуска будет просто делать один шаг на основе каждого градиента, но сценарий Ньютона, используемый алгоритмом импульса, вместо этого использует эту силу для изменения скорости точки. Можно провести аналогию точки с хоккейной шайбой, скользящую по ледяной поверхности. Всякий раз, когда он спускается по крутой части поверхности, он набирает скорость и продолжает скользить в этом направлении, пока не начнет снова подниматься в гору.

Необходима другая сила. Если единственной силой будет оценочной функции, то точка может никогда не остановиться. Чтобы решить эту проблему, вводится еще одна сила, пропорциональная $-v(t)$. В физической терминологии эта сила соответствует вязкому сопротивлению, как будто частица должна протолкнуть устойчивую вязкую среду. Это заставляет её постепенно терять энергию с течением времени и в итоге сходиться к локальному минимуму.

Причина использования величины $-v(t)$ и вязкого сопротивления частично состоит в математическом удобстве - с целой степенью скорости

легко работать. В других физических системах есть другие виды сопротивления, основанные на других целочисленных степенях скорости. Например, частица, перемещающаяся по воздуху, испытывает турбулентное сопротивление с силой, пропорциональной квадрату скорости, в то время как частица, движущаяся по земле, испытывает сухое трение с силой постоянной величины. Но в данном случае эти величины не подходят. Турбулентное сопротивление, пропорциональное квадрату скорости, становится очень слабым, когда скорость мала. Оно недостаточно велико, чтобы заставить точку остановиться. Частица с ненулевой начальной скоростью, испытывающая только силу турбулентного сопротивления, навсегда отойдет от своего исходного положения, причем расстояние от начальной точки будет расти как $O(\log t)$. Поэтому необходимо использовать более низкую степень скорости. При использовании нулевой степени, представляющей сухое трение, то сила слишком велика. Когда движущая точку сила из-за градиента функции потерь мала, но не равна нулю, постоянная сила из-за трения может привести к тому, что точка остановится до достижения локального минимума. Вязкое сопротивление предотвращает обе эти проблемы - оно достаточно слабое, чтобы градиент продолжал вызывать движение до достижения минимума, достаточно сильное, чтобы предотвратить движение, если градиент не оправдывает движение [11].

2.2.3 Метод импульса Нестерова

В 2013 году был представлен вариант алгоритма импульса, который был основан на методе ускоренного градиента Нестерова. Правила обновления в этом случае имеют вид (44) и (45):

$$v \leftarrow \alpha v - \varepsilon \nabla \theta [\frac{1}{m} \sum_{i-1}^{m} L(f(x^{(i)}; \theta + \alpha v), y^{(i)})], \qquad (44)$$

$$\theta \leftarrow \theta + v, \tag{45}$$

где параметры α и ε играют ту же роль, что и в стандартном методе импульса. Разница между импульсом Нестерова и стандартным импульсом, где оценивается градиент. При импульсе Нестерова градиент оценивается после применения текущей скорости. Таким образом, можно интерпретировать импульс Нестерова как попытку добавить корректирующий коэффициент к стандартному методу импульса. В случае выпуклого пакетного градиента импульс Нестерова приводит скорость сходимости избыточной ошибки от $O\left(\frac{1}{k}\right)$ (последующие шаги) к $O\left(\frac{1}{k^2}\right)$. К сожалению, в случае стохастического градиента импульс Нестерова не улучшает скорость сходимости [11].

2.3 Алгоритмы оптимизации с адаптивным темпом обучения

Исследователи нейронных сетей давно поняли, что коэффициент скорости обучения является одним из самых трудных для установки гиперпараметров, поскольку он существенно влияет на производительность модели. Оценка часто очень чувствительна к некоторым направлениям в пространстве параметров и нечувствительна к другим. Алгоритм импульса может несколько смягчить эти проблемы, но он делает это за счет введения другого гиперпараметра. Перед лицом этого естественно спросить, есть ли другой путь. Если считать, что направления чувствительности частично совпадают с осями, может иметь смысл использовать отдельную скорость обучения для каждого параметра и автоматически адаптировать эти скорости обучения в течение всего обучения.

Алгоритм delta-bar-delta (DBD, дельта-такт-дельта) является ранним эвристическим подходом к адаптации индивидуальных скоростей обучения к параметрам модели во время обучения. Подход основан на простой идее: если частная производная функции потерь по отношению к данному параметру

модели остается с тем же знаком, то скорость обучения должна возрасти. Если эта частная производная меняет знак, то скорость обучения должна уменьшаться. Конечно, этот вид правил может применяться только для полной пакетной оптимизации [11].

В последнее время был введен ряд дополнительных (или мини-пакетных) методов, которые адаптируют скорости обучения параметров модели.

2.3.1 AdaGrad

Алгоритм AdaGrad индивидуально адаптирует скорости обучения всех параметров модели, масштабируя их обратно пропорционально квадратному корню суммы всех исторических квадратов значений градиента. Параметры с наибольшей частной производной функции потерь имеют быстрое снижение скорости обучения, в то время как параметры с малыми производными имеют относительно небольшое снижение скорости обучения. Следствием этого является больший прогресс в более пологих направлениях пространства параметров.

В контексте выпуклой оптимизации алгоритм AdaGrad обладает некоторыми желательными теоретическими свойствами. Однако эмпирически для обучения моделей глубоких нейросетей накопление квадратов градиентов с начала процесса обучения может привести к преждевременному и чрезмерному снижению эффективной скорости обучения. AdaGrad хорошо работает для некоторых, но не для всех моделей глубокого обучения [11].

2.3.2 RMSProp

Алгоритм RMSProp модифицирует AdaGrad для повышения производительности при невыпуклых настройках, изменяя накопление градиента в экспоненциально взвешенном скользящем среднем. AdaGrad

разработан, чтобы быстро сходиться при применении к выпуклой функции. Применительно к невыпуклой функции для обучения нейронной сети траектория обучения может проходить через множество различных структур и в итоге идти в область, которая является локально выпуклой чашей. AdaGrad сокращает скорость обучения в соответствии со всей историей квадрата градиента и с некоторой вероятностью может делать скорость обучения слишком малой, прежде чем прийти к такой выпуклой структуре. RMSProp использует экспоненциально убывающее среднее, чтобы отбросить историю из крайнего прошлого, чтобы процесс мог быстро сходиться после нахождения выпуклой чаши, как если бы это был пример алгоритма AdaGrad, инициализированного внутри этой чаши.

По сравнению с AdaGrad использование скользящего среднего вводит новый гиперпараметр ρ, который управляет масштабом длины скользящего среднего. Эмпирически показано, что RMSProp является эффективным и практичным алгоритмом оптимизации для глубоких нейронных сетей. В настоящее время это один из методов, который регулярно используется практиками в глубоком машинном обучении [11].

2.3.3 AdaDelta

Алгоритм AdaDelta – модификация алгоритма AdaGrad. Данные модификации призваны исправить два основных недостатка AdaGrad:

– постоянное снижение скорости обучения на протяжении всего процесса обучения;

– необходимость в ручном выборе глобального коэффициента скорости обучения.

В методе AdaGrad знаменатель накапливает квадратичные градиенты с каждой итерации, начиная с начала обучения. Поскольку каждый член является положительным, эта накопленная сумма продолжает расти на

протяжении всего обучения, снижая скорость обучения в каждом измерении. После многих итераций эта скорость обучения станет бесконечно малой [11].

Идея метода состоит в том, что вместо того, чтобы накапливать сумму квадратов градиентов за все время, количество накопленных градиентов за предыдущие итерации ограничивается окном w некоторого фиксированного размера. При таком оконном накоплении знаменатель AdaGrad не может накапливаться до бесконечности и вместо этого становится локальной оценкой с использованием последних градиентов. Это гарантирует, что обучение продолжается что обучение продолжает прогрессировать даже после того, как было сделано много итераций обновлений.

Преимуществом метода состоит в том, что для него не нужно изначальное точное указание коэффициента скорости обучения, достаточно лишь прикидочное значение.

Данный метод чувствителен к гиперпараметру α. Если его выбрать близким к единице, то AdaDelta будет недоверчиво относиться к редко используемым весам, что может привести к остановке алгоритма без достижения оптимального решения, или может вызвать «жадное» поведения, когда сначала обновляются только те нейроны, которые кодируют лучшие явные признаки [12].

2.3.4 Adam

Adam - еще один адаптивный алгоритм оптимизации скорости обучения. Название «Adam» происходит от фразы «adaptive momentum». В контексте более ранних алгоритмов его, возможно, лучше всего рассматривать как вариант комбинации RMSProp и метода импульса с несколькими важными различиями. Во-первых, в методе Adam импульс включается непосредственно как оценка момента первого порядка (с экспоненциальным взвешиванием) градиента. Самый простой способ добавить импульс в RMSProp - применить импульс к измененным градиентам. Использование импульса в сочетании с

масштабированием не имеет четкой теоретической мотивации. Во-вторых, Adam включает поправки смещения к оценкам как моментов первого порядка (импульс-момент), так и (нецентрированных) моментов второго порядка, чтобы учесть их инициализацию в начале координат. RMSProp также включает в себя оценку момента второго порядка; однако в нем отсутствует корректирующий коэффициент. Таким образом, в отличие от Adam, оценка моментов второго порядка RMSProp может быть очень двоякой в обучении. Adam, как правило, считается достаточно устойчивым к выбору гиперпараметров, хотя скорость обучения иногда необходимо изменить по сравнению с предложенным значением по умолчанию [11].

2.4 Выбор метода оптимизации

В разделе выше рассматриваются методы, каждый из которых направлен на решение задачи оптимизации глубоких моделей путем адаптации скорости обучения для каждого параметра модели. На данный момент естественным является вопрос выбора алгоритма.

В настоящее время нет единого мнения по этому вопросу. Хотя результаты показывают, что семейство алгоритмов с адаптивными скоростями обучения (представленных RMSProp и AdaDelta) работает достаточно надежно, единственного лучшего алгоритма не появилось. В настоящее время наиболее популярные алгоритмы оптимизации, которые активно используются, включают SGD, SGD с импульсом, RMSProp, RMSProp с импульсом, AdaDelta и Adam. Выбор того, какой алгоритм использовать в данный момент, в значительной степени зависит от знакомства исследователя с алгоритмом (для простоты настройки параметров).

3 Свёрточная нейронная сеть

Свёрточная нейронная сеть (англ. convolutional neural network, CNN) – архитектура нейронной сети, которую предложил Ян Лекун в 1988 году. Главным образом данная архитектура сети адаптирована для эффективного распознавания визуальных образов. Входит в состав технологий глубокого машинного обучения [13].

Данная модель основана на особенностях зрительной коры головного мозга, в которой присутствуют простые клетки, реагирующие на прямые линии под разными углами, и сложные клетки, реакция которых связана с активацией определённой комбинации простых клеток. Таким образом, идея свёрточных нейронных сетей заключается в чередовании свёрточных слоев и субдискретизирующих слоев (слоев подвыборки).

Сеть обладает однонаправленной структурой (без обратных связей), принципиально многослойная. Для обучения используются стандартные методы, чаще всего метод обратного распространения ошибки. Функция активации нейронов (передаточная функция) — любая, по выбору исследователя.

Задача классификации изображений — это приём начального изображения и вывод его класса изображенного объекта (автомобиль, животное и т. д.) или группы вероятных классов, которая лучше всего характеризует изображение.

Компьютер принимает на вход многомерный массив пикселей. Как показано на изображении 13.

Рисунок 13 – Слева – образ, воспринимаемый человеком, справа – образ, представленный в памяти компьютера

Например, размер массива может быть 64x64x3 (где «3» — это количество каналов цветовой схемы RGB). Допустим, на вход поступает цветное изображение в формате JPEG, и его размер 800x800. Следовательно, размер массива будет 800x800x3. Каждому из этих чисел присваивается значение от 0 до 255, которое описывает интенсивность пикселя в этой точке, данное число обычно нормируется для проведения дальнейших вычислений и лежит в промежутке [0;1].

Общая концепция работы свёрточной нейронной сети состоит в способности выполнять классификацию изображений через поиск характеристик базового уровня, например, границ и искривлений, а затем с помощью построения более абстрактных концепций через группы свёрточных слоев.

Перцептрон представляет собой полносвязную нейронную сеть, то есть, нейроны в связях образуют полносвязный граф – каждый нейрон связан со всеми нейронами, находящимися на предыдущем слое. Каждая такая связь имеет свой весовой коэффициент, из чего следует большое количество вычисляемых параметров в процессе обучения.

В свёрточной нейронной сети в операции свёртки используется ограниченная матрица весов небольшого размера, которая сдвигается по всему обрабатываемому слою (изначально — непосредственно по входному изображению) с некоторым определенным заранее шагом, формируя после каждого сдвига сигнал активации для нейрона следующего слоя с аналогичной

позицией. То есть для различных нейронов выходного слоя используются одна и та же матрица весов, которую также называют ядром свёртки или фильтром. Она интерпретируется как графическое кодирование какого-либо признака, например, наличие вертикальной или горизонтальной линии. Тогда следующий слой, являющийся результатом свёртки такой матрицей весов, показывает наличие данного признака в обрабатываемом слое и её координаты, формируя карту признаков [14].

Сверточная нейронная сеть содержит не один набор весов, а целую. гамму, кодирующую элементы изображения. Каждый набор весов отвечает за свой признак: от основных, таких, как наличие линий под разными углами, дуг до абстрактных, которые явно не распознаются человеческим глазом. Ядра свертки при этом изначально не заданы, чаще всего на старте инициализируются случайными значениями и далее формируются в процессе обучения нейронной сети. Проход каждым набором весов формирует свой собственный экземпляр карты признаков, делая нейронную сеть многоканальной (много независимых карт признаков на одном слое). Следует отметить, что сдвиг ядра свертки при проходе по матрице должен быть небольшим, чтобы не пропустить искомый признак. Например, при матрице весов размерности 5x5, её сдвиг происходит на один или два пикселя.

Операция субдискретизации (операция подвыборки или операция объединения), выполняет уменьшение размерности сформированных карт признаков. В данной архитектуре сети считается, что информация о факте наличия искомого признака важнее точного знания его координат, поэтому из нескольких соседних нейронов карты признаков выбирается максимальный и принимается за один нейрон уплотнённой карты признаков меньшей размерности. За счёт данной операции, помимо ускорения дальнейших вычислений, сеть становится более инвариантной к масштабу входного изображения [13].

Преимущества:

– один из лучших алгоритмов классификации визуальных образов;

– по сравнению с полносвязной нейронной сетью (типа перцептрона) — гораздо меньшее количество настраиваемых весов, так как одно ядро весов используется целиком для всего изображения вместо того, чтобы делать для каждого пикселя входного изображения свои персональные весовые коэффициенты. Это подталкивает нейросеть при обучении к обобщению демонстрируемой информации, что является свойством правильно обученной сети, а не попиксельному запоминанию каждого входного образа с очень большим количеством весовых коэффициентов;

– удобное распараллеливание вычислений, что влечет за собой возможность реализации алгоритмов работы и обучения сети на графических процессорах или высокопроизводительных вычислительных кластерах;

– относительная устойчивость к повороту и сдвигу распознаваемого изображения;

– обучение при помощи классического метода обратного распространения ошибки.

К недостатку данной архитектуры можно отнести большое количество варьируемых параметров сети, неясно, для какой задачи и вычислительной мощности какие нужны настройки. Так, к варьируемым параметрам можно отнести: количество слоев, размерность ядра свёртки для каждого из слоёв, количество ядер для каждого из слоёв, шаг сдвига ядра при обработке слоя, необходимость слоев субдискретизации, степень уменьшения ими размерности, функция по уменьшению размерности (выбор максимума, среднего и т. п.), передаточная функция нейронов, наличие и параметры выходной полносвязной нейросети на выходе свёрточной, а так же начальная инициализация весовых коэффициентов сети.

Нет точного описания для выбора начальных параметров модели нейронной сети, есть только некоторые общие рекомендации, относящиеся к выбору количества свёрточных слоёв и размера ядра свёртки. Более точная настройка и оптимизация сети требует эмпирического исследования и выбора

параметров под конкретную задачу. Тонкая настройка модели нейронной сети представляет собой обширный раздел теории машинного обучения.

3.1 Архитектура свёрточной нейронной сети

Свёрточный слой является основным блоком нейронной сети (рисунок 14), включающий в себя для каждого канала свой фильтр, который обрабатывает предыдущий слой по фрагментам, суммируя результаты матричного произведения для каждого фрагмента.

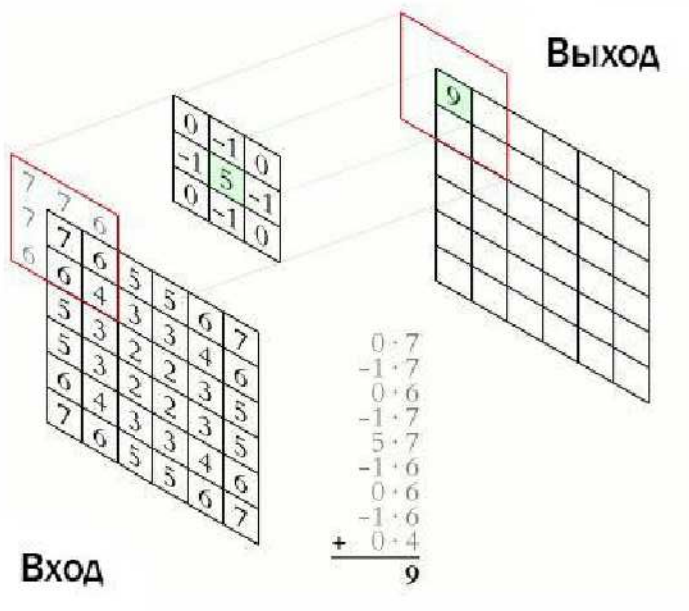

Рисунок 14 – Визуальное представление процесса свертки

Весовые коэффициенты ядра свёртки неизвестны и устанавливаются в процессе обучения [13].

Особенностью свёрточного слоя является сравнительно небольшое количество параметров, устанавливаемое при обучении. Так например, если исходное изображение имеет размерность 100×100 пикселей по трём каналам (это значит 30000 входных нейронов), а свёрточный слой использует фильтры с ядром 3x3 пикселя с выходом на 6 каналов, тогда в процессе обучения определяется только 9 весов ядра, однако по всем сочетаниям каналов, то есть

9×3×6 = 162, в таком случае данный слой требует нахождения только 162 параметров, что существенно меньше количества искомых параметров полносвязной нейронной сети.

ReLU — сокращение от английского англ. rectified linear unit (ReLU) - блок линейной ректификации. Слой ReLU — не что иное как функция активации после свёрточного слоя. То есть здесь для активации выбирается вместо классических функций типа гиперболического тангенса (46):

$$f(x) = \tanh(x), \tag{46}$$

или сигмоиды (47):

$$f(x) = \frac{1}{1 + e^{-1}}, \tag{47}$$

ненасыщаемая функция (48):

$$f(x) = \max(0, x). \tag{48}$$

Данная функция дает хороший результат при обучении свёрточных нейронных сетей и отвечает за отсечение ненужных деталей (шумов) в канале (при отрицательном выходе).

Слой пулинга (иначе подвыборки, субдискретизации) представляет собой нелинейное уплотнение карты признаков, при этом группа пикселей (обычно размера 2×2) уплотняется до одного пикселя нелинейным преобразованием. Наиболее употребительна при этом функция максимума. Преобразования затрагивают непересекающиеся прямоугольники или квадраты, каждый из которых ужимается в один пиксель, при этом выбирается пиксель, имеющий максимальное значение. Операция пулинга позволяет существенно уменьшить пространственный объём изображения и упростить

последующие вычисления. Интерпретировать пулинг можно следующим образом: если на предыдущем этапе свёртки уже были выявлены некоторые признаки, то для дальнейшей обработки настолько подробное изображение уже не нужно, и оно уплотняется до менее подробного. К тому же фильтрация уже ненужных деталей помогает не переобучаться, то есть, сеть не будет терять способности к обобщению. Слой пулинга (рисунок 15), как правило, вставляется после свёрточного слоя перед слоем следующей свёртки.

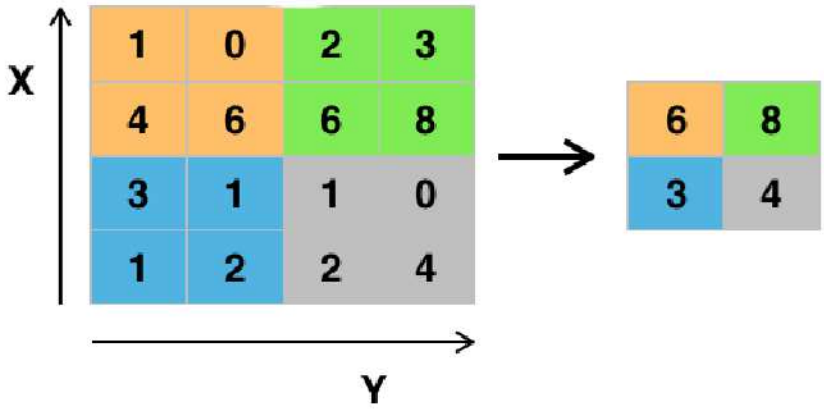

Рисунок 15 – Визуализация процесса пулинга (подвыборки или субдискретизации)

Кроме пулинга с функцией максимума можно использовать и другие функции — например, среднего значения или L2-нормирования. Однако практика показала преимущества именно пулинга с функцией максимума, который включается в типовые системы [14].

После нескольких операций свёртки изображения и уплотнения с помощью субдискретизации на каждом следующем слое увеличивается число каналов и уменьшается размерность изображения в каждом канале. После выполнения всех этапов свёртки и пулинга входная матрица вырождается в вектор, содержащий основные значения – самые абстрактные понятия, выявленные из исходного изображения.

Данный вектор передается на вход в следующий слой, представляющий собой полносвязную нейронную сеть, которая в свою очередь может состоять

из нескольких слоёв. При этом полносвязные слои утрачивают пространственную структуру пикселей и обладают сравнительно небольшой размерностью (по отношению к количеству пикселей исходного изображения). В качестве функции активации полносвязной сети обычно принимается сигмоидная функция.

Схема архитектуры сверточной нейронной сети приведена на рисунке 16:

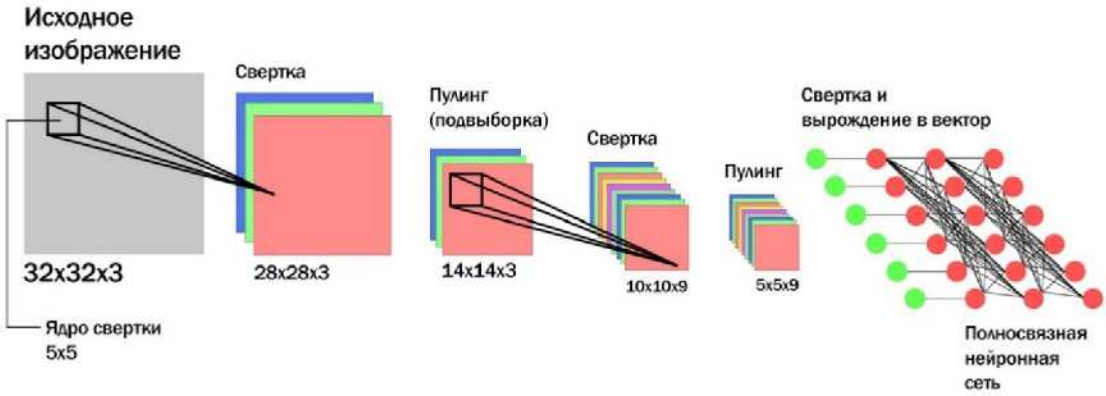

Рисунок 16 – Классическая схема архитектуры сверточной нейронной сети

4 Постановка задачи

В рамках данной работы рассматривается задача классификации изображений с помощью свёрточной нейронной сети.

Основной практической задачей является разработка программной системы геолокации, которая с помощью свёрточных нейронных сетей по изображению земной поверхности способна определить тип географического объекта и его наименование.

Всего рассматриваются три класса объектов: населенный пункт, водоём и горный массив. Определение наименования горного массива является нетривиальной задачей, так как внешний вид различных горных систем имеет очень схожую структуру, что требует для обучения сети набор обучающих данных колоссального размера и использование больших вычислительных ресурсов. Поэтому задача классификации наименования горного массива в данной работе рассматриваться не будет.

В связи с ограничениями система должна определять наименования только населенных пунктов и водоёмов, так как различные объекты этих типов обладают уникальным внешним видом и контуром в контексте спутникового снимка земной поверхности. Исходя из вышеизложенного, понятно, что система должна содержать три нейронные сети: одна для первичной классификации и по одной на каждый из двух типов объектов, для которых необходимо определить наименование.

4.1 Описание метода решения поставленной задачи

Для разрешения задачи классификации изображения географического объекта должна быть построена свёрточная нейронная сеть.

Причина выбора нейронной сети для классификации изображений состоит в том, что нейронные сети имеют явные преимущества перед классическими алгоритмами.

Одним из основных плюсов нейронной сети можно выделить её устойчивость к шумам во входных данных, то есть нет необходимости предварительно обрабатывать входной сигнал, удаляя шумовые артефакты - сеть способна не учитывать маловероятные значения. Свёрточная нейронная сеть главным образом предназначена для распознавания визуальных образов, поэтому выбор пал именно на данную архитектуру нейронной сети. Свёрточная нейронная сеть обладает частичной инвариантностью к размеру входного изображения за счет использования субдискретизации и имеет устойчивость к повороту изображения. Также стоит отметить то, что в сравнении с полносвязной нейронной сетью, сверточная сеть обладает числом настраиваемых весовых коэффициентов, что приводит к сокращению времени на вычисления при обучении сети. Использование ядер свертки позволяет избежать попиксельного запоминания изображения и повышает способность сети к обобщению. Локальное восприятие позволяет сохранить топологию изображения от слоя к слою при значительном сокращении вычислений. Благодаря сканированию целой области, а не отдельных точек, подход позволяет учесть свойства изображения, что увеличивает качество распознавания [15].

Однако свёрточная нейронная сеть достаточно требовательна к объему обучающего набора, так как недостаточное количество обучающих примеров может привести к переобучению сети и потери ею способности к обобщению демонстрируемой информации.

Разрабатываемая программная система геолокации должна содержать несколько сетей: первая нейронная сеть производит предварительную классификацию типа географического объекта (водоём, населенный пункт, горный массив), далее на основании решения первой сети вступает в работу одна из двух нейронных сетей, предназначенных для идентификации объекта в множестве определенного ранее класса, то есть определения наименования объекта. Схема функционирования системы геолокации изображена на рисунке 17:

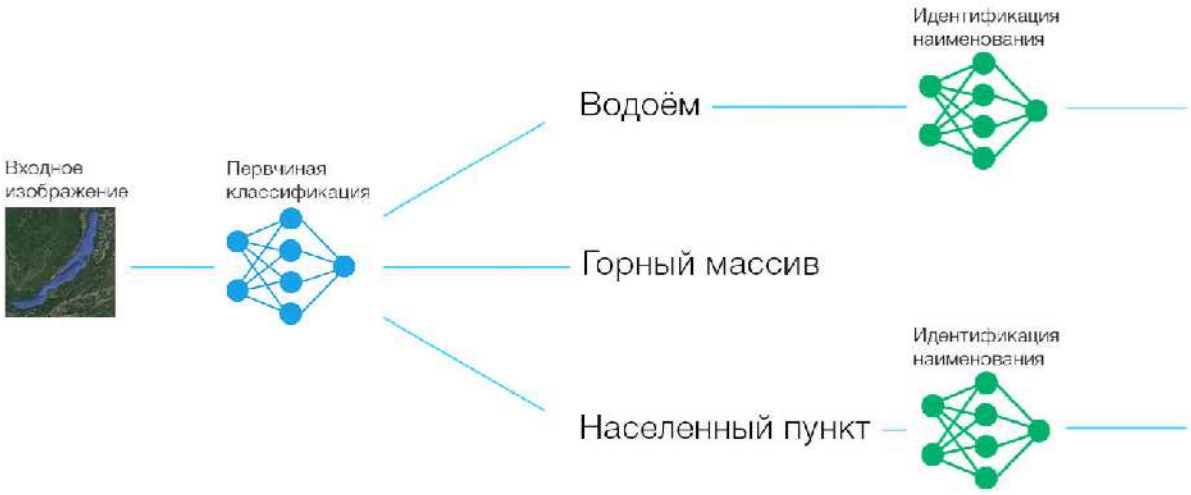

Рисунок 17 – Визуализация процесса идентификации объекта на изображении земной поверхности

Чтобы реализовать решение поставленной проблемы, необходимо выполнить перечисленные далее действия.

Создание и обучение свёрточных нейронных сетей:

– разработать архитектуру сетей в соответствии с требованиями поставленной задачи;

– подготовить обучающую выборку данных;

– выбрать алгоритм оптимизации;

– обучить и протестировать нейронные сети;

– составить необходимый каскад из полученных моделей сетей.

Подготовка входного изображения:

– загрузить исходное изображение – спутниковый снимок некоторой области земной поверхности;

– выбрать произвольный участок данного изображения. Этот участок отправить на вход системе.

4.2 Выбор параметров обучения и архитектуры нейронных сетей системы

За качество распознавания образов с помощью свёрточной нейронной сети приходится платить большим количеством варьируемых параметров: количество слоев свертки, размер ядра свёртки для каждого из слоев, шаг сдвига ядра в процессе свертки, необходимость использования слоев подвыборки (субдискретизации), степень уменьшения ими размерности, функция уменьшения размерности, плотность нейронов выходной полносвязной нейронной сети, функция активации нейронов. Все эти параметры оказывают сильное влияние на результат работы конечной сети и выбираются исследователем эмпирически.

Выбор архитектуры сети опирается на базовые рекомендации в рамках конкретной задачи распознавания визуальных образов и на основании ограничений, которые накладываются на архитектуру в связи с относительно небольшой вычислительной мощностью тестовой системы.

Так, например, одним из основных параметров свёрточной нейронной сети является количество слоёв свертки. В рамках разрабатываемой программной системы число свёрточных слоев и слоев субдискретизации равно трем. Слои свёртки содержат 32, 32, 64 трёхканальных фильтров соответственно размера 3x3 пикселя. Такие значения количества фильтров на первых двух слоях обусловлены тем, что на начальном этапе сеть должна распознавать только основные признаки (контуры, крупные объекты, кривые линии). На последнем слое свертки количество фильтров увеличено в два раза для увеличения способности сети распознавать мелкие неочевидные детали.

Чем больше размер ядра свертки, тем сильнее степень уменьшения размерности входной матрицы и тем меньше деталей способна распознать сеть. В контексте данной задачи размер ядра свертки выбран равным 3x3, так как спутниковые снимки объектов земной поверхности имеют достаточно мелкие детали, которые характеризуют объект как конкретный класс.

Входной слой полносвязной нейронной сети, стоящей после последнего этапа свертки и субдискретизации содержит 64 нейрона, так как входное изображение к данному моменту вырождается в вектор размерности 64. Выходной слой полносвязной сети состоит из трех элементов, так как предварительная классификация рассматривает три типа объекта.

Параметры архитектуры нейронных сетей для вторичной идентификации сходны первой за исключением количества нейронов выходного слоя полносвязной сети, которое в данном случае равно двум, так как в рамках работы требуется определить наименование одного из двух населенных пунктов или водоёмов.

Слой субдискретизации везде идентичен и представляет собой операцию выбора максимального значения из квадрата 2x2 пикселя.

На этапе свёртки в качестве функции активации используется линейная ректификация – ReLU. В свою очередь на выходном слое полносвязной сети используется сигмоидная функция.

В качестве параметров обучения нейронных сетей можно выделить следующие: скорость обучения, метод оптимизации оценочной функции сети, количество эпох обучения.

На основании исследования методов оптимизации нейронных сетей для обучения построенных моделей используется метод RMSProp как хорошо зарекомендовавший себя на практике в обучении глубоких свёрточных нейронных сетей. Данный метод не позволяет процессу обучения остановиться в точке локального минимума, не достигнув лучшего результата, а также является эффективным по времени. В данном случае метод Adam оказался избыточным, так как модель не обладает большой глубиной и сложностью.

Начальная скорость обучения сети выбрана равной 0,01. Чем коэффициент скорости обучения больше, тем быстрее будет происходить процесс обучения сети, однако это ведет к снижению точности алгоритма. Слишком малое значение данного параметра приведет к низкой скорости

сходимости алгоритма, но позволит достичь максимальной точности. Точный выбор коэффициента скорости обучения в контексте данной работы не столь важен ввиду выбора метода оптимизации сети, адаптирующего в процессе обучения эту величину путём её изменения обратно пропорционально экспоненциально взвешенному скользящему среднему исторических градиентов оценочной функции нейронной сети.

Количество эпох обучения выяснено эмпирическим путем через проведение некоторого количества испытаний. Это один из гиперпараметров, который в значительной степени влияет на точность работы нейронной сети. На изображении 18 рассматривается диапазон от 1 до 30 эпох обучения.

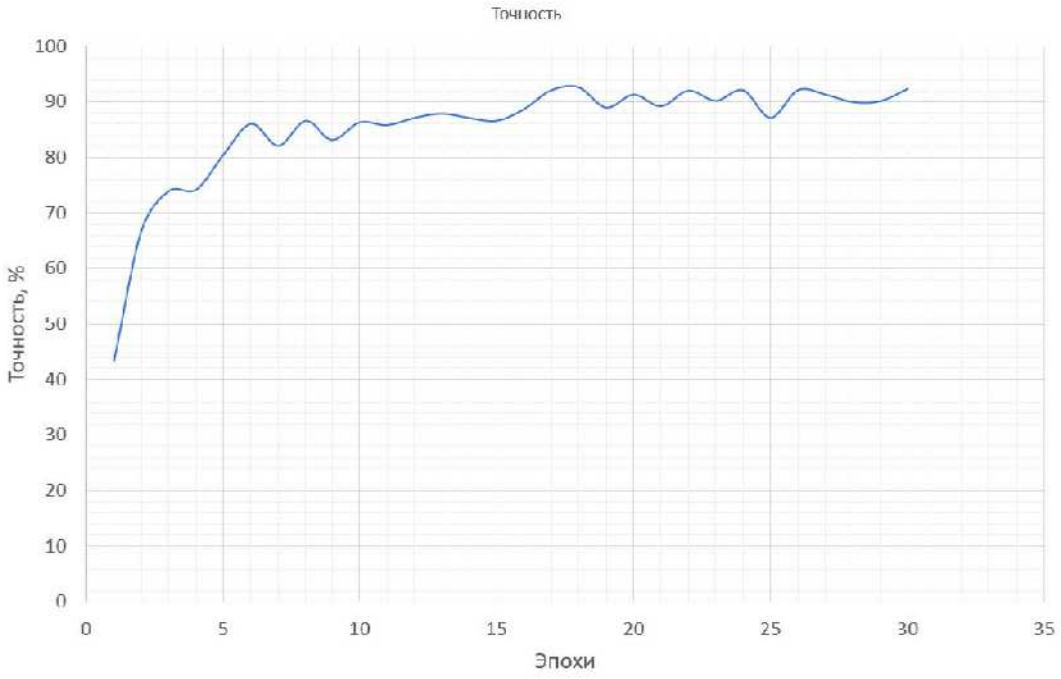

Рисунок 18 – График зависимости точности обучения от количества эпох

Как видно из графика, после достижения на 18 эпохе обучения максимального значения, точность обучения дальше не растет. Поэтому количество эпох в обучении используется 18.

Так же одним из параметров можно выделить процентное соотношение исключенных из обучения на текущей эпохе нейронов через обнуление для уменьшения эффекта переобучения сети. Этот параметр обычно предлагается

брать в пределах от 0,1 до 0,3. В контексте разрабатываемой системы этот параметр для всех сетей равен и имеет значение 0,1. Это обусловлено небольшим размером обучающей выборки, которая содержит всего по 300 примеров для каждого класса, и небольшой плотностью выходной полносвязной сети.

5 Реализация программной системы

В рамках проекта в качестве основного инструмента разработки приложения используется интерпретируемый язык программирования Python, так как для него существуют библиотеки, адаптированные для работы с моделями искусственного интеллекта: библиотеку NumPy с открытым исходным кодом, содержащая реализации вычислительных алгоритмов в виде операторов и функций, оптимизированных для работы с многомерными массивами; пакет для работы с растровой графикой – PIL (Python Image Library), которая поддерживает чтение и запись растровых изображений форматов BMP, JPEG, GIF, PNG, TIFF. Для визуализации работы программы и вывода изображения на экран используется библиотека PyOpenGL, предоставляющая высокоуровневый интерфейс для работы с графическим API OpenGL. Для разработки свёрточной нейронной сети применяется фреймворк TensorFlow, а также библиотека Keras, являющаяся надстройкой над данным фреймворком для оптимизации и упрощения работы с ним.

Используется распространенная и эталонная реализация языка Python – CPython. CPython является интерпретатором байт-кода, написан на C.

Для хранения изображений исходного снимка и входного изображения сети используется расширение JPEG – формат хранения растровых изображений, преимуществом которого являются простота работы и сравнительно небольшой размер.

В качестве среды разработки выступает среда JetBrains PyCharm, которая предоставляет широкий инструментарий для работы с кодом, проведения тестирования и отладки.

5.1 Разработка программы, реализующей решение поставленной задачи

Программный код реализует следующий алгоритм:

Шаг 1: загружается спутниковый снимок земной поверхности.

Шаг 2: курсором мыши выбирается квадратной участок произвольного размера, который необходимо классифицировать как некоторый географический объект и определить его наименование. Максимальный размер выбираемой области ограничен квадратом размера 64x64 пикселя снизу и 200x200 пикселей сверху.

Шаг 3: выделенный участок формирует изображение выбранного размера и сохраняется на диск под именем *input.jpg*.

Шаг 4: при необходимости полученное изображение масштабируется до размера 64x64 пикселя и подается на вход обученной первой нейронной сети, которая возвращает ответ в виде вектора, содержащего три элемента, каждый из которых отвечает за один класс. Данные компоненты вектора представляют собой вероятностные значения принадлежности данного географического объекта определенной категории: водоём, населенный пункт, горный массив. Наибольшее значение определяет класс изображения.

Шаг 5: на основании полученного типа географического объекта срабатывает один из сетей второго рода, которая определяет наименование того или иного населенного пункта или водоёма.

Обучение сетей производится с учителем методом обратного распространения ошибки. Для этого сформирована обучающая выборка из 2100 скриншотов размера 64x64 пикселя (по 300 изображений для каждого распознаваемого класса) географических объектов спутниковых снимков Земли, предоставленных сервисом Google Maps. Пример фрагмента обучающего набора изображен на рисунке 19.

Рисунок 19 – Фрагмент набора данных для обучения нейронной сети на примере изображений водоёмов

Программная система геолокации состоит из трех модулей: core, cnn_main, cnn_train. Модуль core содержит в себе функции описания интерфейса и загрузки входных данных:

– reshape() – функция, которая выполняет инициализацию сцены для последующей визуализации;

– display() – процедура, отвечающая за визуализацию;

– mouse() – функция обработки нажатий клавиш мыши;

– mouseMove() – обработчик движений мыши; возвращает координаты указателя мыши;

– keys_pressed() – функция, отвечающая за обработку нажатий клавиш клавиатуры.

Модуль cnn_main содержит в себе функцию загрузки изображения и загрузки обученных моделей свёрточных нейронных сетей:

– load_image(img_path, img_width, img_height) – производит загрузку входного изображения, формируя матрицу значений пикселей, в качестве параметров принимает путь к расположению файла изображения в строковом виде и размеры изображения по горизонтали и вертикали;

– classify_object(img_width, img_height) – непосредственно производит загрузку обученных нейронных сетей, выполняет анализ и возвращает ответ в удобочитаемом формате.

Модуль cnn_train включает в себя описание архитектуры нейронных сетей, начальных значений, функцию загрузки обучающих данных и функции обучения сетей. Данный модуль производит сохранение обученных нейронных сетей в иерархическом формате данных H5.

Диаграмма взаимодействия модулей программы выглядит следующим образом, как показано на изображении 20:

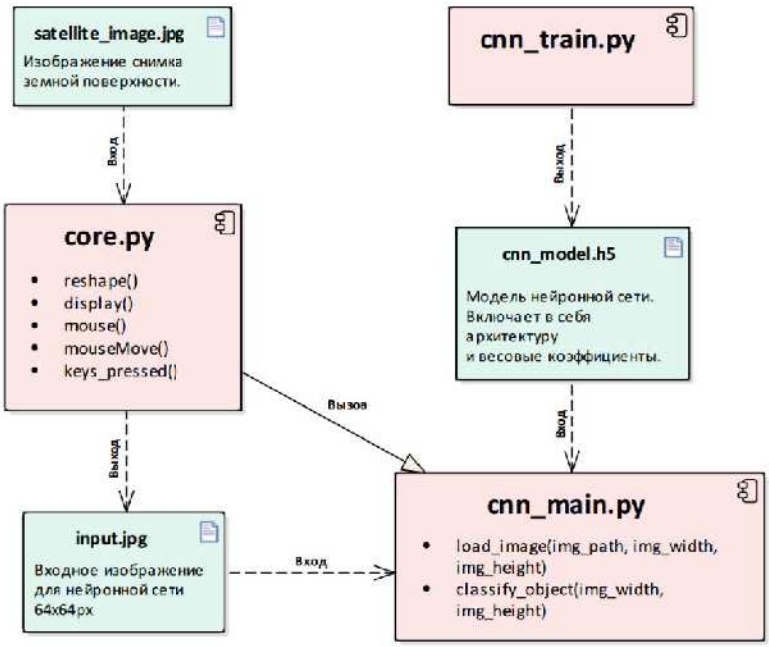

Рисунок 20 – Компонентная модель программы

5.2 Демонстрация выполнения разработанного приложения

Для функционирования системы в первую очередь запускается модуль cnn_train.py, результатом работы которого станут три файла в формате H5, содержащих структуру и весовые коэффициенты трех нейронных сетей.

Чтобы исследователь мог взаимодействовать с системой, запускается модуль core.py, который описывает визуальный пользовательский интерфейс.

На вход поступает цветное изображение снимка земной поверхности произвольного масштаба и размерности (рисунок 21):

Рисунок 21 – Исходное изображение

Затем выбирается квадратный участок произвольного размера, например, область размера 200x200 пикселей, содержащая водоём (помечена красным цветом на рисунке 22):

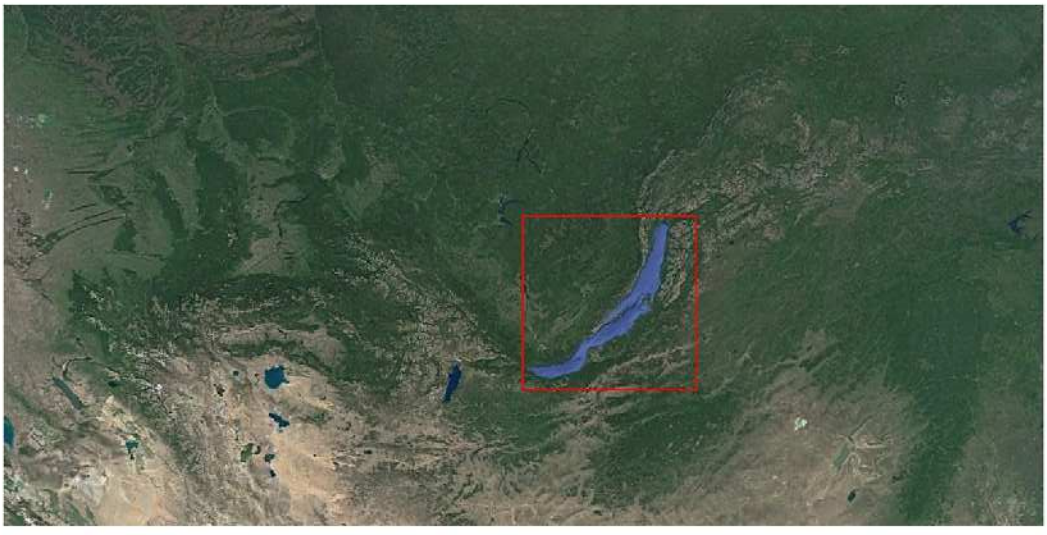

Рисунок 22 – Выбор произвольного участка на карте (отмечен красным)

Выбранный участок после масштабирования представляет собой трёхканальное цветное изображение размера 64x64 пикселя и является входным для нейронной сети (рисунок 23):

Рисунок 23 – Входное изображение свёрточной нейронной сети

Сформированное входное изображение сохраняется на жестком диске с именем *input.jpg* и при каждом выборе нового участка изображения земной поверхности заново перезаписывается. Модуль cnn_main.py производит загрузку данного изображения и отправляет на вход нейронной сети первичной классификации.

Нейронные сети возвращают ответ в виде вектора вероятностных значений принадлежности объекта одному из рассматриваемых классов. Нейронная сеть предварительной классификации определяет тип географического объекта, а вторая, предназначенная для идентификации водоёмов – его наименование (рисунок 24).

```
Водоём:                  27.2662%
Населённый пункт:  0.0397%
Горная местность:  0.0312%

Класс: Водоём
Байкал:                  92.9243%
Краснодарское водохранилище:  0.04%

Наименование водоёма: оз. Байкал
```

Рисунок 24 – Результат выполнения программы

Для демонстрации работы алгоритма выбирается участок, принадлежащий данному снимку земной поверхности, однако он может быть и внешним произвольным изображением географического объекта, загруженным с носителя информации.

При обучении сети достигнута максимальная точность равная 92%, что является приемлемым показателем для выбранной архитектуры. Полученной точности удалось достичь с помощью применения продвинутого алгоритма оптимизации функции ошибки нейронной сети при обучении – вместо классического метода градиентного спуска был использован метод RMSProp, который представляет собой модифицированный метод Adagrad с адаптивной скоростью обучения на основе исторических градиентов. Однако, нейронные сети не избежали эффекта переобучения, так как набор обучающих примеров, сформированный вручную достаточно мал. Для более качественного обучения необходим массив данных, включающий, по крайней мере, несколько тысяч изображений.

ЗАКЛЮЧЕНИЕ

В рамках данной работы изучены основные технологии, используемые в искусственных нейронных сетях, архитектура сетей и методы их обучения. Рассмотрены свёрточные нейронные сети, которые позволяют решать задачи идентификации образов и классификации изображений.

В ходе исследования разработана программная система геолокации, позволяющая идентифицировать объекты географических карт - спутниковых снимков земной поверхности с помощью каскада нейронных сетей.

Созданные в рамках проекта модели нейронных сетей были обучены на предварительно подготовленном наборе обучающих данных и протестирована на специально подготовленной тестовой выборке.

Из анализа полученных результатов тестирования разработанных сетей сделан вывод, что правильный выбор метода оптимизации нейронной сети и выбор параметров обучения позволяет добиться высокой точности распознавания и минимизировать явление переобучения модели.

В процессе исследования было выяснено, что разработка альтернативного метода позиционирования с помощью искусственного интеллекта – достаточно жизнеспособный и перспективный проект.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Мак-Каллок У. С., Питтс В. Логическое исчисление идей, относящихся к нервной активности. Под ред. К. Э. Шеннона и Дж. Маккарти. — М.: Изд-во иностр. лит., 1956. — С. 363—384. (Перевод английской статьи 1943 г.).

2 Колесников П. Искусственные нейронные сети (ИНС) - что такое нейросети, как они работают, преимущества и недостатки искусственных нейронов, где используются нейросети. URL: http://www.stevsky.ru/kompiuteri/iskusstvennie-neyronnie-seti-ins-chto-takoe-neyroseti-kak-oni-rabotaiut-preimuschestva-i-nedostatki-iskusstvennich-neyronov-gde-ispolzuiutsya-neyroseti (дата обращения 02.05.2019).

3 Перцептрон. URL: http://info-farm.ru/alphabet_index/p/perceptron.html (дата обращения 06.05.2019)

4 Персептроны. Обучение персептронов. URL: http://algmet.simulacrum.me/tai/l3/index.htm (дата обращения 05.05.2019).

5 Нейронные сети: обучение без учителя. URL: http://www.codenet.ru/progr/alg/ai/htm/gl3_4.php (дата обращения 02.05.2019).

6 Терехов С. А., лекции по теории и приложениям искусственных нейронных сетей. Обучение методом обратного распространения ошибок // Лаборатория Искусственных Нейронных Сетей НТО-2, ВНИИТФ, Снежинск. URL: https://studfiles.net/preview/986654/page:11 (дата обращения 02.05.2019).

7 Борисов Е. О рекуррентных нейронных сетях. URL: http://mechanoid.kiev.ua/neural-net-rnn.html (дата обращения 05.05.2019).

8 Молотилин Т. Азбука ИИ. URL: https://nplus1.ru/material/2016/11/04/recurrent-networks (дата обращения 07.05.2019).

9 Hopfield J. J., «Neural networks and physical systems with emergent collective computational abilities», Proceedings of National Academy of Sciences, vol. 79 no. 8 pp. 2554–2558, April 1982

10 Jordan M. I. Serial order: A parallel distributed processing approach. // Institute for Cognitive Science Report 8604. — University of California, San Diego, 1986

11 Goodfellow I., Bengio Y., Courville A., Deep Learning, An MIT Pressbook, 2016. URL: http://www.deeplearningbook.org/ (дата обращения 11.05.2019)

12 Zeiler M. D., ADADELTA: AN ADAPTIVE LEARNING RATE METHOD // Google Inc. USA, New York University, 2012. URL: https://arxiv.org/pdf/1212.5701.pdf (дата обращения 07.05.2019)

13 Convolutional Neural Networks (LeNet) - DeepLearning 0.1 documentation. DeepLearning. 0.1. LISA Lab. URL: http://deeplearning.net/tutorial/lenet.html (дата обращения 05.05.2019)

14 Rashid T., Make Your Own Neural Network, 1st edition // Tariq Rashid 2017

15 Федоренко Ю. С. Технология распознавания образов с использованием свёрточной нейронной сети / Ю. С. Федоренко // Инженерный Вестник, ФГБОУ ВПО «МГТУ им. Н.Э. Баумана». – 2013