

## АННОТАЦИЯ

Темой данной работы является «Анализ эмоциональной нагрузки текста с помощью методов машинного обучения».

Цель работы — разработка и практическая реализация нейронной сети, которая позволяет классифицировать текст, написанный на русском языке.

Объем данной работы составил 55 страниц. Для ее написания было использовано 23 источника. В работе содержатся 3 приложения, 17 рисунков, 1 таблица. В дипломную работу входят 5 глав. В первой главе приведена вся теоретическая часть, необходимая для понимания реализации нейронной сети. В главе №2 описывается архитектура и параметры реализованной нейронной сети. В 3 главе приводится программная реализация, в 4 — тестирование обученной модели и веб-приложения для визуализации тестирования.

## СОДЕРЖАНИЕ

ВВЕДЕНИЕ .....	5
1 Обзор предметной области .....	7
1.1 Эмоциональная окраска текста .....	7
1.1.1 Понятие эмоциональной окраски текста.....	7
1.1.2 Примеры применения анализа тональности текстов .....	8
1.2 Машинное обучение.....	9
1.2.1 История и проблемы развития машинного обучения .....	9
1.2.2 Определение и виды машинного обучения.....	12
1.2.3 Обучение с учителем.....	12
1.3 Глубокое обучение.....	15
1.3.1 Перцептрон .....	16
1.3.2 Функции активации.....	17
1.3.3 Функции потерь.....	20
1.3.3.1 Выбор функции потерь для задачи бинарной классификации.....	20
1.3.4 Оптимизация нейронной сети .....	21
1.3.4.1 Градиентный спуск.....	21
1.3.4.2 Стохастический градиентный спуск с импульсом .....	23
1.3.4.3 Adam .....	25
1.3.5 Методы борьбы с переобучением .....	26
1.3.5.1 Дропаут.....	27
1.4 Рекуррентная нейронная сеть.....	28
1.4.1 LSTM .....	30

1.5 Векторизация слов.....	32
1.5.1 Word2Vec .....	32
2 Описание параметров нейронной сети.....	34
2.1 Выбор архитектуры и гиперпараметров сети для задачи определения тональности текста .....	34
2.2 Описание обучающей выборки.....	36
2.3 Предварительная обработка данных .....	37
2.4 Векторизация .....	38
3 Программная реализация.....	39
3.1 Обучение модели .....	39
3.2 Веб-приложение .....	39
3.2.1 Структура веб-приложения .....	40
3.2.2 Описание работы веб-приложения .....	41
4 Руководство пользователя.....	42
4.1 Установка библиотек и зависимостей .....	42
4.2 Инструкция по сборке проекта .....	42
5 Тестирование.....	44
5.1 Тестирование качества обучения нейронной сети .....	44
5.2 Тестирование веб-приложения.....	46
ЗАКЛЮЧЕНИЕ.....	47
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	48
ПРИЛОЖЕНИЕ А. Листинг класса LSTM_architecture.....	51
ПРИЛОЖЕНИЕ В. Листинг функции обучения модели .....	52
ПРИЛОЖЕНИЕ С . Листинг функции предсказания результата.....	53

## ВВЕДЕНИЕ

Задача анализа тональности текстов становится очень актуальной с каждым годом и набирает все большую популярность в современном мире.

С древних времен и по настоящее время человек живет в социуме и мнение окружающих всегда влияло на различные аспекты жизнедеятельности. Однако, если в прошлом люди могли узнать мнение только ограниченного числа лиц, то в наше время с появлением интернета и огромной аудиторией сети, социальных сетей, интернет-магазинов, специализированных ресурсов пользователи имеют возможность узнать мнение большой аудитории.

Вместе с тем, развитие информационных технологий привело к росту числа веб-сайтов и большому объему текстовых данных, так как наиболее распространенной формой хранения информации являются тексты на естественном языке (Natural Language Processing). Однако рядовым пользователям с каждым годом все сложнее работать с такими объемами данных. Очевидно, что ручной поиск и анализ нужной информации в гигантских массивах текстовых данных являются неэффективными и дорогостоящими. Таким образом, информатизация населения, необходимость перевода текстов в электронный вид приводят к разработке эффективных алгоритмов анализа и классификации этих текстов. Одной из задач классификации является распознавание эмоциональной окраски текста, которую также называют анализом тональности текста (Sentiment Analysis) и решается она с помощью методов машинного обучения [1].

Целью данной работы является разработка и практическая реализация нейронной сети, которая позволяет классифицировать текст, написанный на русском языке, по двум видам эмоциональной окраски: положительный или негативный.

Для достижения цели были поставлены следующие задачи:

- 1) Изучение материала, необходимого для понимания действия работы нейронных сетей.
- 2) Исследование способов оптимизации нейронных сетей, а также их регуляризации.
- 3) Изучение библиотеки Pytorch для практической реализации нейронной сети.
- 4) Подбор архитектуры и гиперпараметров нейронной сети, при которых точность модели достигала бы не менее 70% точности.
- 5) Обучение и тестирование модели нейронной сети.
- 6) Создание веб-приложения для визуализации работы обученной нейронной сети.

# 1 Обзор предметной области

## 1.1 Эмоциональная окраска текста

### 1.1.1 Понятие эмоциональной окраски текста

Анализ эмоциональной нагрузки (окраски) текста — это задача автоматического анализа эмоционально окрашенной лексики и мнений, выраженных в тексте [2].

Информация в тексте с точки зрения анализа тональности делится на два класса: мнения и факты. Наибольшее значение имеет определение мнения. Мнения можно подразделить на два типа: простое мнение (*regular opinion*) и сравнение (*comparative opinion*).

В простом мнении содержится отношение автора к одному объекту и может быть высказано прямо или неявно. Пример простого мнения: «Этот товар отличного качества». Неявное мнение: «Отдохнув на море, я стал лучше себя чувствовать».

Анализируя тональности текста можно выделить 3 вида эмоций: позитивная, негативная и нейтральная, однако можно выбрать любую градацию классов эмоций. Текст считается нейтральным, если он не несет в себе эмоциональной нагрузки. Некоторые авторы определяют нейтральную тональность как промежуточную между положительной и отрицательной, а другие определяют её как отсутствие какой-либо эмоциональной окраски. Сравнение — процесс количественного или качественного сопоставления разных свойств двух или более объектов с целью формирования определенной оценки или мнения для одного из них [3]. Можно выделить 3 типа сравнений: превосходство одного объекта над другими (*Superlative*), сравнение аспектов объектов в пользу одного (*Nonequal Gradable*), приравнивание аспектов разных объектов (*Equative*).

Объектом анализа тональности может являться любой предмет (услуга, продукт, личность и т.д.), относительно которого выражается мнение в тексте. Нередко эти объекты могут состоять из отдельных частей (свойств и компонентов). Например, объектом мнения является автомобиль. Он обладает составными частями (двигатель, аккумулятор, кузов, дисплей и т.п., а также следующими атрибутами: марка, внешний вид, размер, цвет). Мнение может быть выражено как относительно самого объекта, так и относительно его аспектов. Например, в предложении «Очень хороший автомобиль, мощный двигатель, работает как зверь». Первая его часть выражает положительное мнение об объекте «автомобиль», а вторая — о его аспекте «двигатель».

### 1.1.2 Примеры применения анализа тональности текстов

Анализ тональности становится важнейшим организационным ресурсом, который обеспечивает конкурентные преимущества, развивает инициативы по менеджменту и имеет практическое применение в разных областях:

1) политология: собираются данные из блогов в соцсетях о политических взглядах населения, политических и экономических реформах. В органах, обеспечивающих государственную безопасность, контент-анализ используется для выявления сообщений, содержащих информацию о противоправных действиях (террористические угрозы, анализ запрещенного контента на сайтах и др.;

2) маркетинг: крупные торговые сети по отзывам покупателей в интернете могут узнать отношение клиентов к организации процесса обслуживания в магазине, успешность рекламной кампании. Кроме выявления общей ситуации в плане удовлетворенности работой компании в целом, предоставляется возможность для установления конкретных деталей по услугам (продуктам), которые вызывают одобрение, либо неудовлетворенность (например, один из крупнейших мировых автоконцернов провел исследование мнений автовладельцев, в результате которого установлена связь между

неисправностью и номером партии, вследствие чего некоторые партии автомобилей были полностью отозваны из продажи). Банки узнают мнение клиентов о процессе обслуживания в конкретных их отделениях. Также в области массмедиа производители фильмов, сериалов, развлекательных шоу на основании отзывов зрителей в социальных сетях определяют успешность конкретного проекта и перспективы его дальнейшего производства;

3) социология: анализируются данные социальных сетей о социальной политике государства, выявляются отношение прессы и СМИ к определенной персоне, к организации, к событию (например, в США существует проект Pulse of the Nation [4], цель которого — определение настроения граждан страны в течение дня путем исследования и анализа их записей в социальной сети Twitter);

4) медицина и психология: анализ данных помогает определять депрессию у пользователей социальных сетей, эффективность применения медицинских препаратов и др.

Принимая во внимание перспективы данного направления, стоит отметить, что в настоящее время оно не столь активно применяется в системах обработки текстовой информации. Причинами являются трудности выделения эмоциональной лексики в текстах (присутствие сленга, жаргона, сарказма, что нередко приводит к затруднению понимания эмоциональной окраски текста как людей, так и компьютеров), несовершенство существующих текстовых анализаторов. Поэтому актуальной задачей является совершенствование и разработка новых методов анализа тональности на основе машинного обучения.

## 1.2 Машинное обучение

### 1.2.1 История и проблемы развития машинного обучения

В современном мире разработка алгоритмов машинного обучения началась в 50-х годах прошлого столетия с появлением искусственного



интеллекта, задача которого состоит в создании программ, заставляющих компьютеры «думать».

Первой программой, которая могла бы самостоятельно обучаться, принято считать игру в шашки *Checkers-playing*, которую изобрел американец, специалист в области информатики Артур Самуэль в 1952 году. Тогда им было дано первое определение машинного обучения - это «область исследования разработки машин, не являющихся заранее запрограммированными». Вместе с тем, первые программы для игры в шашки и шахматы действовали по жестко установленным человеком правилам, и не могли называться программами, осуществляющими машинное обучение. На протяжении многих десятилетий специалисты считали, что искусственный интеллект, сравнимый с уровнем человека можно создать, если дать программисту большой набор правил для манипулирования знаниями. Это мнение, известное как символический искусственный интеллект, являлось доминирующей идеей с 50-х до конца 80-х годов прошлого столетия. Символический искусственный интеллект прекрасно справлялся с решением четко определенных логических задач (игра в шашки, шахматы), но не справлялся с более сложными (классификация изображений, распознавание речи и перевод на другие языки). В результате символический искусственный интеллект заменил новый подход: машинное обучение.

В машинном обучении система сама обучается, а не программируется извне. В систему загружаются многочисленные примеры, которые имеют отношение к решаемой задаче, а она находит в этих примерах статистическую структуру, которая позволяет ей сформировать правила для автоматического решения задачи. Например, чтобы автоматизировать задачу определения отзывов о кинофильме, можно передать системе машинного обучения множество примеров этих отзывов из социальных сетей, уже классифицированных людьми, и система изучит статистические правила классификации конкретных отзывов.

В 1957 году была предложена модель нейронной сети, которая имеет сходства с современными алгоритмами машинного обучения. С этого времени

активно разрабатываются модели и системы машинного обучения, такие как алгоритм дерева решений (1986 год), метод опорных векторов (1995 год), глубокое обучение (2005 год) и др.

Расцвет машинного обучения пришелся на 90-е годы прошлого столетия, и превратился в наиболее популярный и успешный раздел искусственного интеллекта, что привело к появлению более быстродействующей аппаратуры и огромных наборов данных. В сфере глубокого обучения за последние годы произошли большие успехи, вместе с тем ожидания на будущее обычно намного превышают вероятные достижения. Если многие значительные применения, такие как автопилоты для автомобилей, находятся практически на заключительной стадии реализации, то такие как полноценные диалоговые системы, перевод между произвольными языками на уровне человека и понимание естественного языка на уровне человека, скорее всего, еще долгое время будут оставаться недостижимыми. Поэтому, завышенные ожидания от ближайшего будущего таят опасность из-за невозможности реализации новых технологий, вследствие чего инвестиции в исследования будут падать и прогресс может замедлиться на неопределенное время.

В прошлом искусственный интеллект уже пережил две волны подъема (1960-е и 1980-е годы-эпоха символического искусственного интеллекта), которые вызвали волну инвестиций, и корпорации по всему миру создавали свои отделы искусственного интеллекта, занимающиеся разработкой экспертных систем, компании тратили огромные средства на развитие технологии. Однако из-за дороговизны в обслуживании, сложностей в масштабировании и ограниченности применения, интерес начинал падать, что получило название первой, второй зимы искусственного интеллекта, которые были сопряжены с разочарованиями и, как результат, снижением финансирования.

В настоящее время человечество стоит на пороге третьего цикла разочарований в искусственном интеллекте, но пока мы еще находимся в фазе завышенного оптимизма. В наше время лучше всего реально оценить

перспективы и донести до людей, мало знакомых с технической стороной этой области, что именно может дать глубокое обучение и на что оно не способно.

### 1.2.2 Определение и виды машинного обучения

Цель машинного обучения — предсказать результат по входным данным определенного формата. В машинном обучении системе передается корпус, состоящий из большого количества примеров, которые имеют отношение к решаемой задаче, и данная система находит в этих примерах статистическую структуру, позволяющая системе выработать правила для автоматического решения задачи [5].

Машинное обучение не возможно без следующих составляющих: входных данных, примеров ожидаемых результатов и способов оценки работы алгоритма — метрики качества [6]. Для задачи определения тональности текстов входными данными являются сами тексты, ожидаемые результаты — один из возможных классов эмоций (например, положительный или негативный), в качестве способа оценки работы чаще всего берут метрику качества *accuracy* — доля правильных ответов на тестовой выборке.

Специалисты выделяют два основных алгоритма машинного обучения — обучение с учителем и обучение без учителя, однако так же набирает популярность метод обучения с частичным привлечением учителя (обучение с подкреплением). Существенная разница первых двух указанных методов — в наличии эталонных значений предсказываемых целевых переменных (часто их называют метками или *targets, labels*) для наблюдаемых величин: с учителем такие данные присутствуют, без учителя, соответственно, нет.

### 1.2.3 Обучение с учителем

Формально задачу обучения с учителем можно описать следующим образом: Дан набор данных  $D = \{x_i, y_i\}_{i=1}^n$  ( $n$  - размер набора,  $x_i$  - объект,  $y_i$  — правильный ответ для объекта  $x_i$ ). На его основе необходимо обучить модель  $f$ , параметризованную весами  $w$ , таким образом, чтобы функция потерь

$L(y, y^*), y^*$  — целевое значение переменной, достигала своего минимального значения. То есть задача обучения заключается в поиске оптимальных значений весов. Входными значениями являются числовые вектора, так как алгоритмы машинного обучения могут работать только с числовыми данными, и довольно часто этот этап является значительной частью работы алгоритма.

Если говорить простым языком, обучение с учителем — это восстановление некоторой закономерности по конечному числу входных примеров. Для лучшего понимания этого типа машинного обучения рассмотрим пример. Допустим, человек заходит на сайт интернет-магазина и добавляет в корзину некоторые товары. Очевидно, магазину выгодно, чтобы человек купил как можно больше товаров. Самый рациональный вариант для этого — предлагать пользователю товары, которые в теории могли бы заинтересовать конкретного покупателя. Но как это сделать?

1) Выдавать случайные позиции товаров из определенной категории — такой вариант не требует использования машинного обучения и прост в реализации, но с большой долей вероятности человек не обратит внимания на эти товары и, соответственно, не будет переходить по рекомендуемым ссылкам.

2) Смотреть на названия добавленных товаров и предлагать товары, которые очень похожи по названию. Это легко можно реализовать, например, с помощью регулярных выражений.

3) У магазина есть много примеров, когда другие пользователи заходили на страницу товара, и принимали решение, добавлять его в корзину или нет. Это та информация, из которой можно восстановить общую зависимость. В этом и заключается задача машинного обучения с учителем: объектом в данном примере является пара {пользователь, товар}, и для этой пары необходимо предсказать, заинтересует ли этот товар данного пользователя и добавит ли человек товар в корзину, либо же в «список желаний».

Пространство объектов - множество всевозможных пар {пользователь, товар}, ответ — число 0 или 1, которое обозначает, заинтересует ли товар

пользователя или не заинтересует соответственно. В качестве признаков можно выбрать разные варианты, например, анкетные данные покупателя.

Типы задач обучения на размеченных данных определяется пространством возможных ответов. Самыми популярными являются следующие задачи:

- Задача бинарной/многоклассовой классификации

В данной задаче пространство ответов состоит из  $n$  чисел  $\{0, 1, \dots, n - 1\}$ , где  $n$  - фиксированное количество классов. Суть задачи — относить объекты к одному из  $n$  классов по заданным признакам (рис.1а). Задача анализа определения эмоциональной окраски текста также относится к типу задач классификации (в данном случае бинарной, так как рассматривается всего два возможных ответа - положительная окраска (1) и негативная (0)).

- Задача регрессии

Если количество вариантов ответов, которое может принимать задача, не фиксировано, то задачу называют задачей регрессии [7].

Данная задача заключается в прогнозировании ответа, которая может принимать любое вещественное значение, на основе обучающей выборки с различными признаками. На рис. 1б представлен пример задачи трех-классовой классификации на основе двух признаков: оси абсцисс и ординат обозначают первый и второй признак соответственно, точки одного цвета обозначают принадлежность к одному классу.

Примеры данной задачи: определение курса доллара на основе данных за прошлый год, оценка возраста человека по его фотографии, предсказание дохода компании на следующий месяц и так далее.

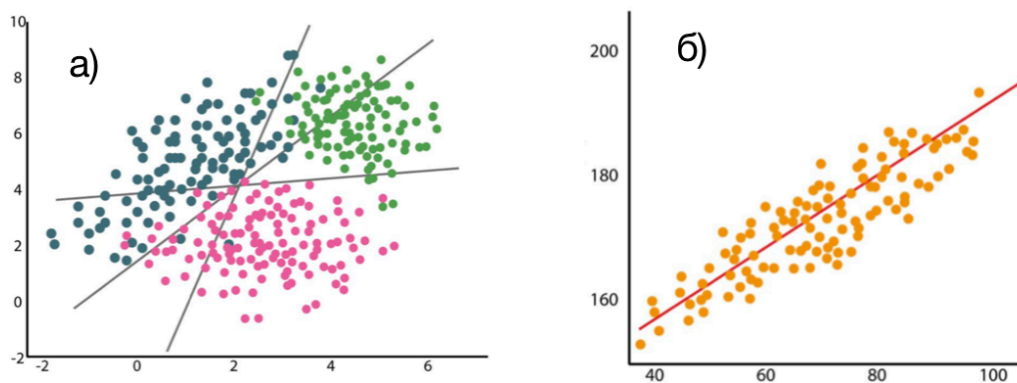


Рис.1 а) Задача многоклассовой классификации; б) Задача регрессии

### 1.3 Глубокое обучение

Для классических методов машинного обучения признаки необходимо выделять вручную. Глубокое обучение автоматизирует этот процесс, что значительно упрощает работу с моделью. Название исходит из многослойного предоставления модели данных; количество слоев называют глубиной модели [8].

Искусственная нейронная сеть — это сеть нейронов, где каждый нейрон представляет собой математическую модель реального нейрона.

Обучение нейронной сети сводится к задаче поиска оптимальных значений весов, при котором функция потерь будет стремиться к своему минимуму.

Нейронную сеть можно представить в виде ориентированного графа(рис.

2). Компоненты сети:

- входной слой,
- скрытые слои, количество которых выбирается в зависимости от задачи
- выходной слой

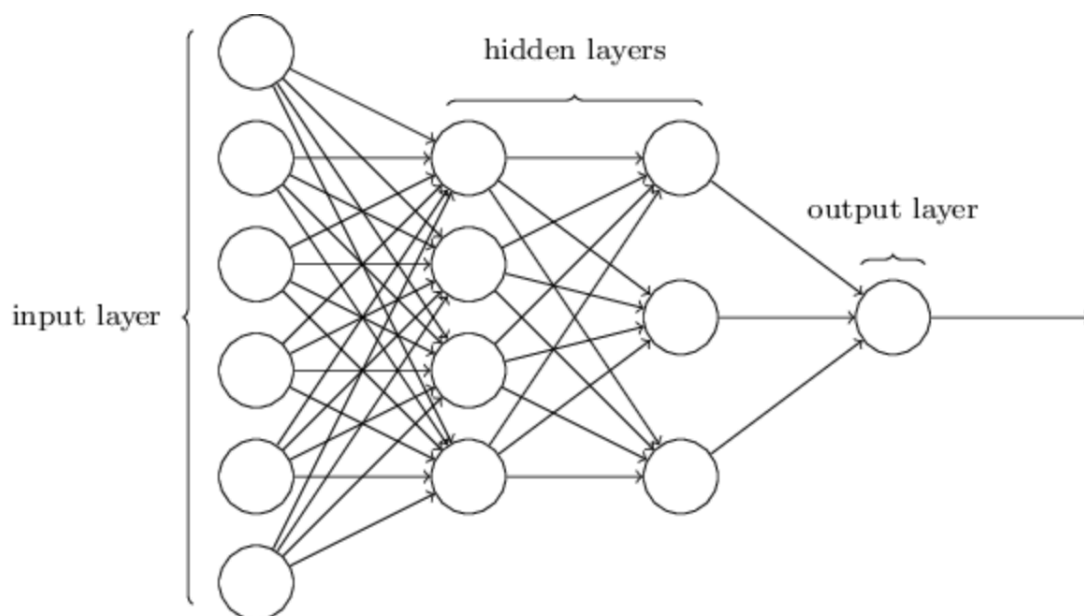


Рис.2 Полносвязная нейронная сеть с двумя скрытыми слоями [8]

Помимо выбора архитектуры нейронной сети важной задачей является правильный подбор значений гиперпараметров, так как это значительно влияет на эффективность модели и ее сходимость. В их число входят функции потерь, оптимизатор и выбор параметра скорости для его обучения, функции активации.

### 1.3.1 Перцептрон

Перцептрон — это простейший блок нейронной сети и самая первая придуманная модель нейронной сети. Он математически описывает модель биологического нейрона: перцептрон так же имеет вход, выход и поток «сигналов» (рис.3).

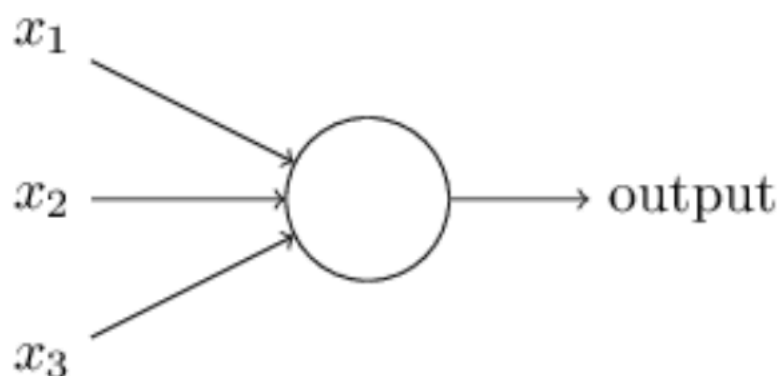


Рис.3 Визуализация перцептрона

Математически работу перцептрона можно выразить следующим образом:  $y = f(w \cdot x + b)$ , где  $x$  — вектор входных значений,  $y$  — выходные данные,  $f$  — функция активации,  $w \cdot x$  — скалярное произведение векторов. Веса и смещение подбираются путем обучения, функция активации подбирается создателем модели исходя из типа решаемой задачи.

Перцептрон является примером сетей прямого распространения. При шаге вперед поток сигнала перемещается от входного слоя через скрытые к выходному, а решение, полученное на выходном слое, сравнивается с заранее заданным верным ответом. При шаге назад с использованием правила дифференцирования сложных функций через перцептрон в обратном направлении распространяются частные производные функции, погрешности по весовым коэффициентам и смещениям.

### 1.3.2 Функции активации

Каждый слой нейронной сети имеет собственную функцию активации, которая вычисляет выходной сигнал искусственного нейрона. Но для чего они нужны? В теории можно было бы заставлять каждый нейрон просто возвращать свой численный результат (другими словами, использовать линейную функцию активации  $f(x) = x$ ), таким образом, слой состоял бы из двух линейных



операций — скалярного умножения веса на значение нейрона и сложения со смещением. Такой слой мог бы обучаться только на аффинных преобразованиях входных данных, и пространство гипотез состояло бы только из линейных преобразований входных данных в  $n$ -мерное пространство. Усложнение архитектуры сети добавлением новых слоев не имело бы никакого смысла, так как не увеличивало бы пространство гипотез: композиция линейных преобразований линейна.

Таким образом, для того чтобы увеличение глубины представления имело смысл, необходимо применять нелинейную функцию на выходе каждого слоя.

Функция активации принимает на вход число и выдает значение в определенном диапазоне, который зависит от вида выбранной функции.

Стандартные ограничения, которые накладываются на функцию активации: непрерывность на всей области определения, монотонность, нелинейность, низкая вычислительная сложность [9].

Самыми часто используемыми в глубоком обучении считаются следующие функции:

#### 1) Сигма-функция

Самая распространенная функция активации в глубоком обучении — сигмоида[10] (или сигма-функция, иногда так же называют логистической

функцией):  $f(x) = \frac{1}{1 + e^{-x}}$ . Выходные данные — любое вещественное число в диапазоне  $[0; 1]$ . График данной функции представлен на рис.4а.

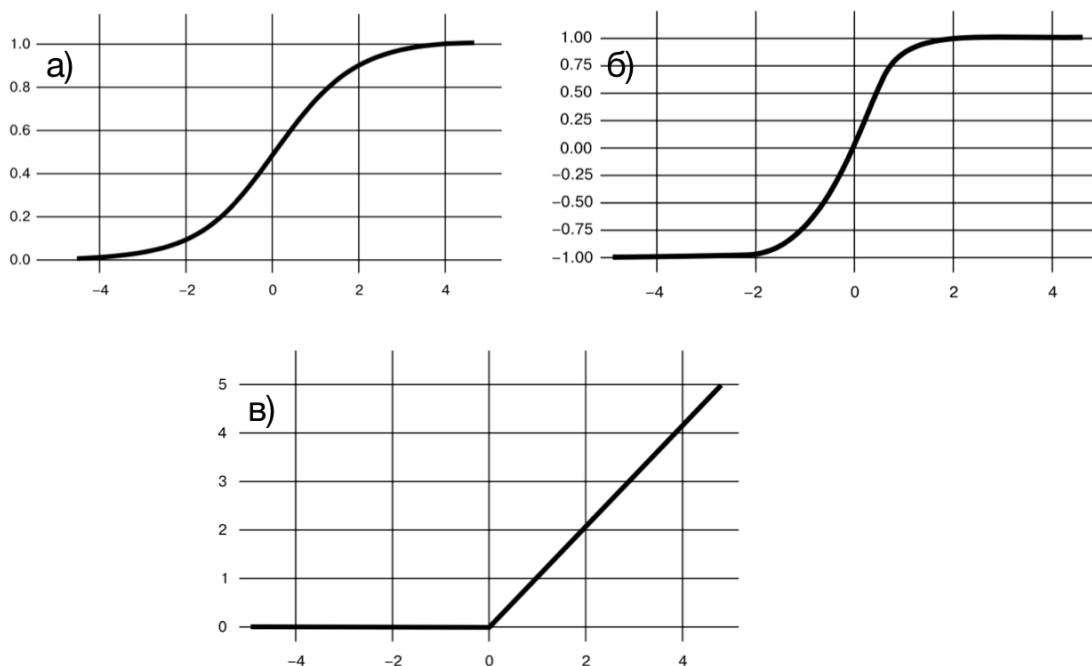


Рисунок 4 — а) Сигма-функция; б) Гиперболический тангенс; в) ReLU [10]

Для сигмоидной функции характерны проблема «исчезающего градиента» (*vanishing gradient problem*) и проблема «взрывного» роста градиента (*exploding gradient problem*), где градиент становится равен нулю или расходится до слишком большого значения с плавающей точкой соответственно, что может помешать в процессе оптимизации функции потерь [11]. Поэтому  $\sigma$ -функции обычно используются в нейронных сетях только на выходном слое, где значение  $\sigma(x)$  часто интерпретируют как вероятность.

## 2) Гиперболический тангенс

Функция гиперболического тангенса (часто обозначают как  $\tanh$  или  $\tanh$ ) — линейное преобразование  $\sigma$ -функции:  $f(x) = th(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ , диапазон значений — отрезок  $[-1, 1]$  (рис.4б)

## 3) ReLU

ReLU (Rectified Linear Unit) — одна из самых распространенных в глубоком обучении. Данная функция обнуляет отрицательные значения:  $f_{ReLU}(x) = \max(0, x)$ . График этой функции представлен на рис. 4в.

### 1.3.3 Функции потерь

В зависимости от типа задачи выбирается наиболее подходящая функция потерь.

#### 1) Среднеквадратичная погрешность

Среднеквадратичная погрешность (*MSE* - *mean squared error*) часто используется для решения задач регрессии. Формула *MSE* выражается следующим образом:

$$L_{MSE}(y, y^*) = \frac{1}{n} \sum_{i=1}^n (y - y^*)^2$$

#### 2) Дискретная перекрестная энтропия

Дискретная перекрестная энтропия чаще всего используется при постановке задачи многоклассовой классификации, при которой выходные значения интерпретируются как предсказания вероятностей принадлежности определенным классам:

$$L(y, y^*) = - \sum_{i=1}^n y_i \log(y_i^*)$$

#### 1.3.3.1 Выбор функции потерь для задачи бинарной классификации

Рассмотрим сигмоидную функцию активации  $\sigma(x) = \frac{1}{1 + e^{-x}}$ .

Производная данной функции по аргументу  $x$ :

$$\sigma'(x) = \frac{1}{(1 + e^{-x})^2} \cdot e^{-x} = \frac{1}{1 + e^{-x}} \cdot \left(1 - \frac{1}{1 + e^{-x}}\right) = \sigma \cdot (\sigma - 1) \quad (5)$$

Сначала возьмем в качестве функции потерь среднеквадратичную ошибку и рассмотрим одно слагаемое:

$$MSE = (\sigma(x) - t)^2$$

где  $t$  — целевое значение из множества  $\{0;1\}$ . Подставляя в производную функции равенство (5), получаем:

$$MSE' = 2 \cdot (\sigma(x) - t) \cdot \sigma' = 2 \cdot (\sigma(x) - t) \sigma(1 - \sigma) \quad (6)$$

При маленьких  $x$  функция  $\sigma(x)$  имеет значение, близкое к нулю. Аналогично если  $x \rightarrow \infty$ , то  $1 - \sigma(x)$  так же будет иметь близкое к нулю значение. В обоих случаях значение производной (6) стремится к нулю. Частое обнуление градиента ведет к проблемам в процессе обучения сети. ] Следовательно, использование среднеквадратичной функции на последнем слое в совокупности с сигмоидной функцией активации нецелесообразно.

Для избежания проблем в задаче бинарной классификации зачастую используется бинарная кросс-энтропия (*BCE — Binary Cross Entropy*) — частный случай дискретной перекрестной энтропии:

$$\text{BCE}(p, t) = -t \log p - (1 - t) \log(1 - p)$$

где  $p = \sigma(x)$ . Используя правило дифференцирования сложной функции, получаем:

$$\frac{\partial \text{BCE}}{\partial x} = \frac{\partial \text{BCE}}{\partial p} \cdot \frac{\partial p}{\partial x} = -\frac{t}{\sigma} \cdot \sigma' + \frac{1-t}{1-\sigma} \cdot \sigma' = -t(1-\sigma) + (1-t)\sigma =$$

$$\sigma - t$$

Таким образом, значение производной функции по аргументу  $x$  не содержит в себе домножения на элементы (например, на  $\sigma$  или  $\sigma - 1$ ), которые могли бы привести к взрыву или затуханию градиента. Из этих соображений для задачи бинарной классификации чаще всего используется бинарная кросс-энтропия в качестве функции потерь.

### 1.3.4 Оптимизация нейронной сети

#### 1.3.4.1 Градиентный спуск

Функция потерь при построении нейронной сети нужна для того, чтобы понимать, в каком направлении ей нужно двигаться, чтобы с каждым шагом улучшать свои результаты. Для этого используются оптимизаторы. Будем считать, что функция потерь обозначается как  $L$ .

Основная идея данного метода заключается в следующем: нужно взять градиент функции  $\nabla L = \left[ \frac{\partial L}{\partial w_0} \quad \frac{\partial L}{\partial w_1} \quad \frac{\partial L}{\partial w_2} \quad \dots \quad \frac{\partial L}{\partial w_n} \right]^T$  и сделать шаг в сторону, которая противоположна направлению градиента, так как вектор градиента в каждой точке направлен в сторону наиболее быстрого возрастания функции [11]. Размер данного шага можно регулировать с помощью некоторого параметра  $\alpha$ , который называется скоростью обучения, и в зависимости от его выбора существуют различные модификации метода. Таким образом, нахождение значений весов на шаге под номером  $t$  выражается с помощью формулы  $w^t = w^t - \alpha \nabla L(w^{t-1})$ . В качестве критерия останова можно выбрать выполнение условия  $\|w^{t+1} - w^t\| < \varepsilon$  или  $\|L(w^{t+1}) - L(w^t)\| < \varepsilon$ , где  $\varepsilon$  — фиксированная малая величина.

От выбора коэффициента  $\alpha$  зависит, будет ли сходиться метод или нет. На рис.5 представлены варианты траектории движения при градиентном спуске с различными скоростями обучения.

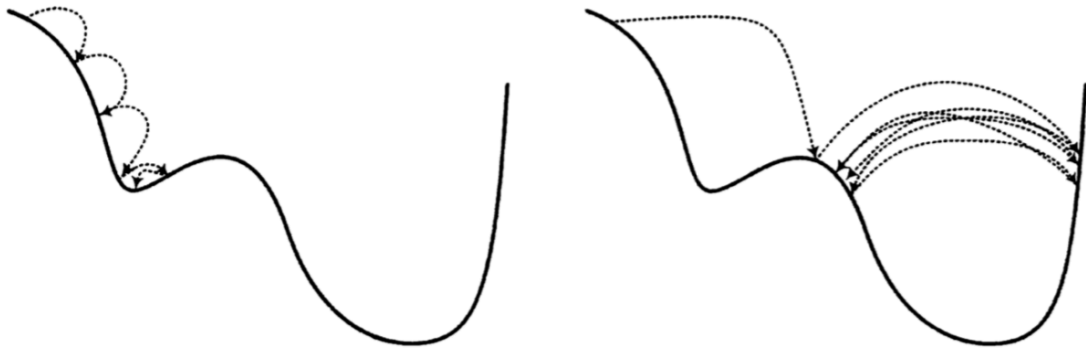


Рисунок 5 — Градиентный спуск для разных скоростей обучения [12]

Одним из недостатков стандартного подхода градиентного спуска является следующее: для того, чтобы выбрать направление, в котором необходимо произвести спуск, нужно посчитать значение и градиент функции потерь по всем элементам обучающей выборки. Размер такой выборки при построении нейронной сети является достаточно объемным, поэтому

предложенный подход является неоптимальным, так как понадобится огромное количество времени для одного шага в задаче поиска минимума. Для решения этой проблемы часто используется такой подход как *пакетный спуск* (*batch gradient descent*): функция вычисляется всего на нескольких примерах, то есть выборка делится на так называемые *батчи*, и на следующем шаге выбираются уже другие примеры, таким образом, расчет следующего шага происходит гораздо быстрее. Нет серьезных ограничений для выбора размера батча, однако стоит придерживаться следующего правила: размер должен быть значительно больше 1 и меньше размера всей выборки.

Другой часто используемый вид метода градиентного спуска для ускорения вычислений — *стохастический градиентный спуск*. Его идея заключается в том, что за один шаг для подсчета вектора весов используется только один элемент выборки.

#### 1.3.4.2 Стохастический градиентный спуск с импульсом

Классический градиентный спуск хорошо работает не для всех функций потерь. Существуют такие функции, для которых классические подходы градиентного спуска будут работать очень медленно. Рассмотрим функцию потерь, линии уровня которых вытянуты по одному из измерений.

Рассмотрим точки А, В, С (рис.6). В — точка, полученная после первого шага градиентного спуска, С — после второго. Заметим, что за две итерации алгоритма мы оказываемся в точке, которая очень близко расположена к первоначальной. Если продолжать алгоритм для заданной функции потерь, то траектория движения будет напоминать зигзагообразную линию, и придется сделать большое количество лишних «движений» для того, чтобы приблизиться к исследуемому минимуму. Это говорит о том, что для нахождения минимума этой функции понадобится значительное количество времени, и такой подход для решения задачи не является оптимальным. Метод, описанный далее, является физической аналогией катящегося шара.

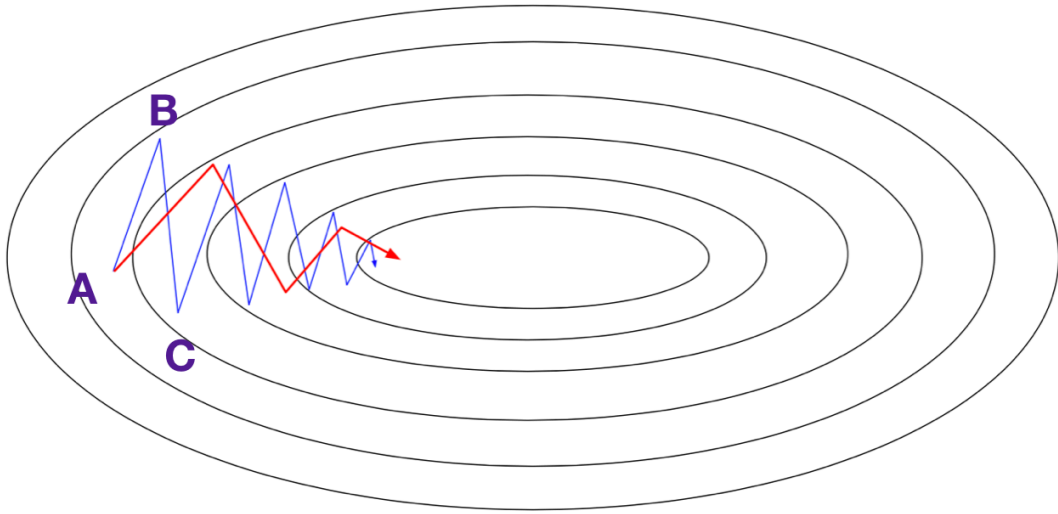


Рис.6 — Траектории различных методов градиентного спуска для функции потерь с «оврагами» [13]

Уравнение движения шара можно описать следующей системой:

$$\begin{cases} \frac{\partial v}{\partial t} = \frac{1}{m}(F + F_f) = -\frac{1}{m} \nabla f - \frac{1}{m} \eta v \\ \frac{\partial x}{\partial t} = v \end{cases} \quad (1)$$

где  $F$  — сила, действующая на материальную точку,  $F_f$  — сила трения,  $m$  — масса шара,  $\eta$  — коэффициент вязкого трения. Первое уравнение описывает Второй закон Ньютона, второе — понятие скорости.

Система (1) решается с помощью метода конечных разностей, которая основывается на определении производной функции:

$$\frac{\partial f}{\partial x} = \lim_{\Delta x \rightarrow \infty} \frac{f(x + \Delta x) - f(x)}{\Delta x} \quad (2)$$

Производную функцию можно приблизить конечной разностью, убрав из формулы (2) знак предела:

$$\frac{\partial f}{\partial x} \approx \frac{f(x + \Delta x) - f(x)}{\Delta x} \quad (3)$$

Таким образом, через уравнение (3) можно выразить  $\frac{\partial v}{\partial t}$  и  $\frac{\partial x}{\partial t}$ :

$$\begin{cases} \frac{\partial v}{\partial t} = \frac{v^{t+1} - v^t}{\Delta t} \\ \frac{\partial x}{\partial t} = \frac{x^{t+1} - x^t}{\Delta t} \end{cases}$$

Подставив данные значения в систему (1), получаем выражения для скорости ( $v$ ) и координаты ( $w$ ) в следующий момент времени:

$$\begin{cases} v^{t+1} = v^t - \frac{\Delta t}{m} \nabla f - \frac{\Delta t}{m} \eta v^t \\ w^{t+1} = w^t + v^t \Delta t \end{cases} \quad (4)$$

Будем считать, что  $\alpha = \Delta t$ ,  $\beta = 1 - \eta \frac{\Delta t}{m}$ ,  $\frac{\Delta t}{m} = 1$ . Тогда систему (4)

можно записать следующим образом:

$$\begin{cases} v^{t+1} = v^t \cdot \beta - \nabla f \\ w^{t+1} = w^t + \alpha v^t \end{cases}$$

Такой алгоритм называют стохастическим градиентным спуском с импульсом (Stochastic Gradient Descent with Momentum). Параметр  $\alpha$  — скорость обучения,  $\beta$  — коэффициент импульса. На рис. 5 синей линией обозначена траектория движения по классическому градиентному спуску, красной — по спуску с импульсом.

#### 1.3.4.3 Adam

Удобство использования алгоритма *Adam* заключается в том, что зачастую не нужно подбирать параметры для его использования: оптимальные коэффициенты для этого алгоритма  $\alpha = 3 \cdot 10^{-4}$ ,  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$  [14]. Существуют архитектуры нейронных сетей, для которых другие параметры показывают результаты обучения лучше, однако для очень большого класса нейронных сетей алгоритм именно с этими коэффициентами показывает хорошие результаты [15]. Зависимость значения веса на следующем шаге от значения на предыдущем представлено в формуле (5):



$$w^{t+1} = w^t - \alpha \frac{\text{EMA}_{\beta_1}(\nabla f)^t}{\sqrt{\text{EMA}_{\beta_2}(\nabla f^2)^t + \varepsilon}} \quad (5)$$

где  $\text{EMA}_{\beta}(f)^t = (1 - \beta)f^t + \beta \cdot \text{EMA}_{\beta}(f)^{t-1}$  — экспоненциальное скользящее среднее в момент времени  $t$ , а квадрат градиента

$$\nabla f^2 = \left[ \frac{\partial f^2}{\partial w_0} \quad \frac{\partial f^2}{\partial w_1} \quad \frac{\partial f^2}{\partial w_2} \quad \dots \quad \frac{\partial f^2}{\partial w_n} \right]^T.$$

### 1.3.5 Методы борьбы с переобучением

Изначально задача глубокого обучения состоит в том, чтобы минимизировать функцию потерь, таким образом, увеличив точность предсказания модели. В какой-то момент, проанализировав все входные данные, нейронная сеть научится правильно классифицировать все исходные объекты или прогнозировать результат по обучающей выборке. Однако возникает вопрос: насколько качественно сеть справится с объектами, которых никогда не встречала прежде? Зачастую модель начинает подстраиваться под обучающую выборку, не выделяя характерные закономерности.

Например, рассмотрим задачу регрессии. На рис. 5а-б зеленым цветом показана истинная зависимость, синими точками — объекты обучающей выборки. На рис. 5а красная линия — график функции  $a(x) = \alpha_0 + \alpha_1 x + \dots + \alpha_4 x^4$ , которая достаточно хорошо описывает и обучающую выборку, и истинную зависимость. (хотя и не проходит через все синие точки). На рис. 5б показан график функции девятой степени  $a(x) = \alpha_0 + \alpha_1 x + \dots + \alpha_9 x^9$ , который проходит через все объекты обучающей выборке (тем самым на тренировочном датасете точность достигает 100%), однако данная функция очень плохо описывает истинную зависимость, тем самым, данная функция не подошла бы для поставленной задачи, так как замечен эффект переобучения.

Выявить переобучение, используя только обучающую выборку, невозможно, поскольку и хорошо обученный, и переобученный алгоритмы

будут хорошо ее описывать. Для этого используют валидационный датасет — часть обучающей выборки, на которой сразу трестируется результат обучения на каждой эпохе (каждом проходе всех обучающих данных).

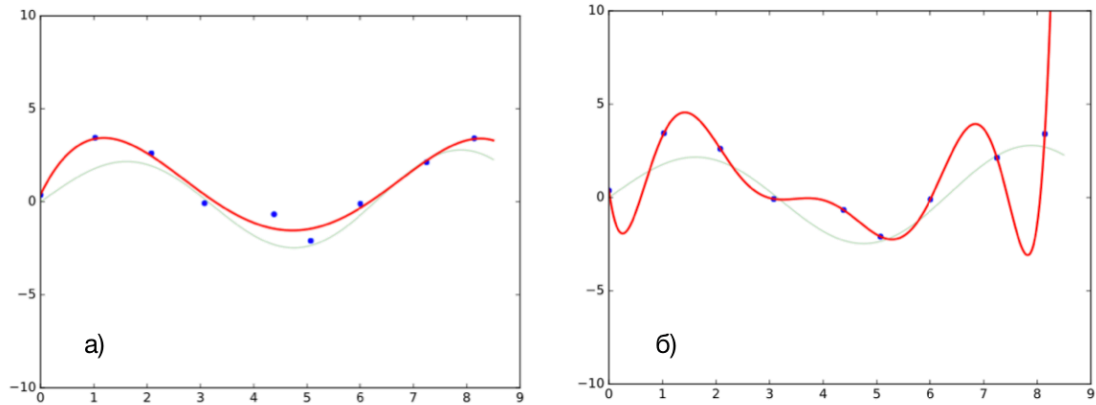


Рис.5 а)  $a(x) = \alpha_0 + \alpha_1x + \dots + \alpha_4x^4$ ; б)  $a(x) = \alpha_0 + \alpha_1x + \dots + \alpha_9x^9$

#### 1.3.5.1 Дропаут

Дропаут (Dropout) — это один из самых часто используемых способов борьбы с переобучением. Суть заключается в том, что сети для обучения получаются с помощью исключения из сети нейронов с вероятностью  $p$  [12] (вероятность того, что нейрон останется в сети, составляет  $1 - p$ ). Нейроны, которые исключаются из сети по описанному правилу, не влияют на обучение модели ни на одном из шагов вычисления градиента с помощью метода обратного распространения ошибки. Это означает, что исключение нейронов из нейронной сети эквивалентно обучению новой сети.

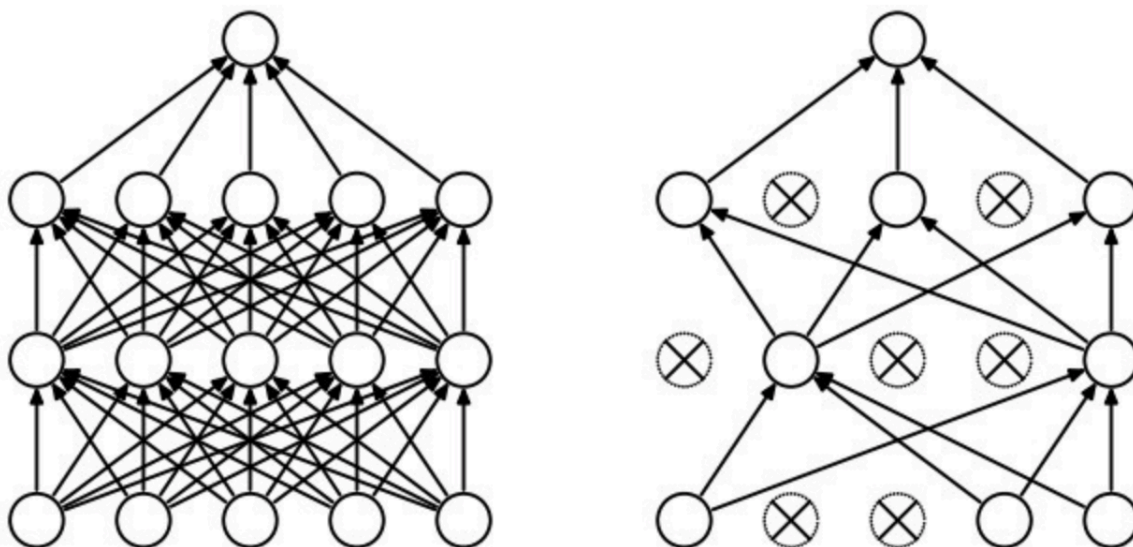


Рис.6 Визуальное представление метода Dropout [12]

#### 1.4 Рекуррентная нейронная сеть

Довольно часто в нейронную сеть подаются данные не фиксированного размера, в частности, при обработке естественного языка.

Идея RNN (*Recurrent Neural Networks*) состоит в последовательном использовании информации. В традиционных нейронных сетях подразумевается, что все входы и выходы независимы, однако для многих задач такой подход не подходит. В отличие от нейронных сетей прямого распространения, в RNN нейроны обмениваются информацией между собой. В сети реализуется «память», что принципиально меняет характер ее работы и позволяет анализировать любые последовательности данных, в которых важно, в каком порядке идут значения. В частности, это очень полезно при обработке текста на естественном языке.

При визуализации элементы рекуррентной нейронной сети изображают как обычные нейроны с дополнительной циклической стрелкой, которая демонстрирует то, что кроме входного сигнала нейрон использует также своё дополнительное скрытое состояние. Схема простейшей RNN с одним входным

нейронном, одним скрытым нейроном и одним выходным нейроном представлена на рис.7:

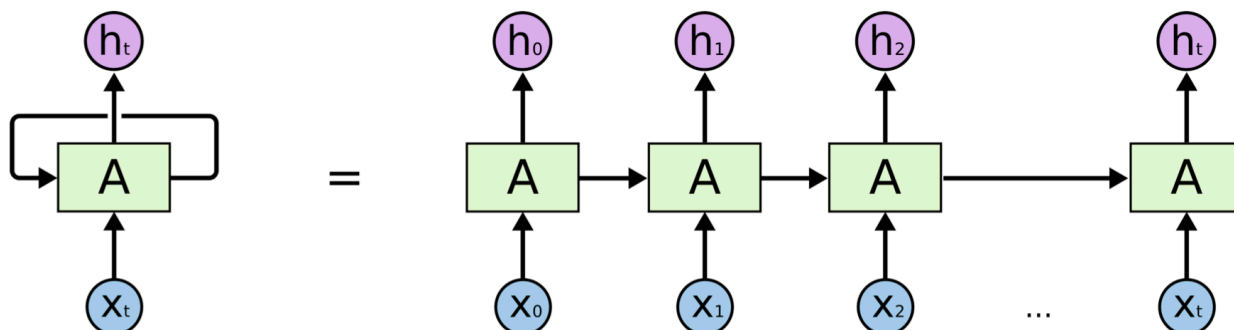


Рис.7 Простейшая рекуррентная нейронная сеть в развертке [14]

Проблема стандартной рекуррентной нейронной сети заключается в том, что она не умеет обрабатывать долговременные зависимости. Например, если задача состоит в предсказании последнего (четвертого) слова в предложении «Мне нужно купить *хлеб*», то необходимо учитывать контекст, состоящий из слов «мне», «нужно», «купить» (порядок здесь играет большое значение). RNN справится с этой задачей, так как расстояние между первым и последним словом невелико. (рис.8)

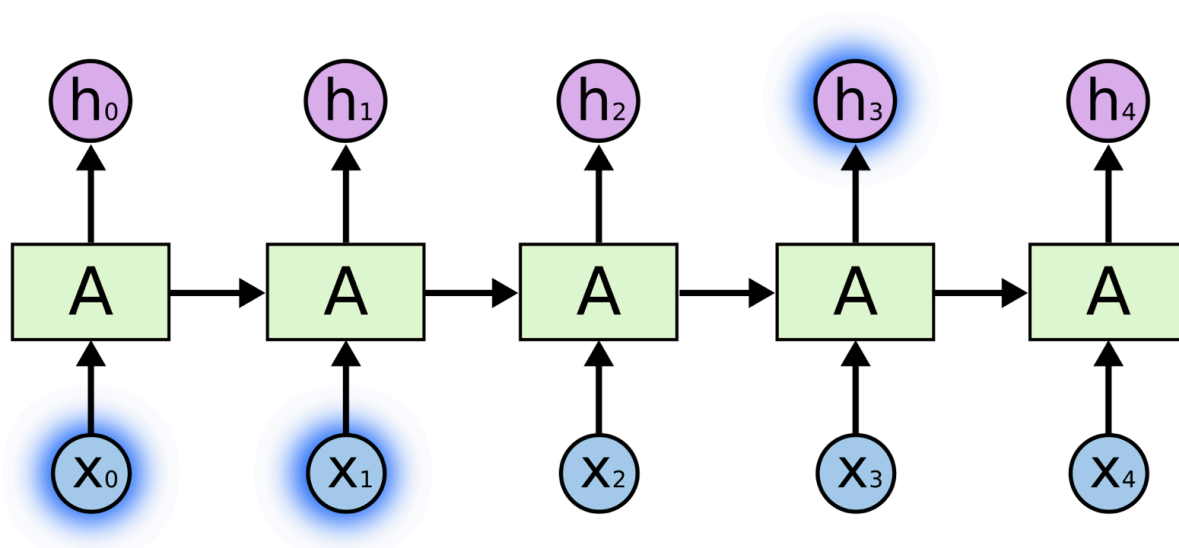


Рисунок 8 — RNN на примере задачи предсказания слова [14]

Однако, если в задаче необходимо учитывать больше слов, то классическая рекуррентная сеть не будет полезна, так как по мере роста длины контекста RNN теряют способность связывать информацию[13]. Для решения данной проблемы были разработаны сети LSTM.

#### 1.4.1 LSTM

LSTM (*Long short-term memory*) — одна из архитектур рекуррентной нейронной сети, способная к обучению долгосрочным зависимостям [14].

Все рекуррентные нейронные сети имеют форму цепи повторяющихся модулей нейронной сети. В стандартной RNN эти повторяющиеся модули будут иметь очень простую структуру, например, всего один слой гиперболического тангенса (рис. 9а), в то время как в LSTM каждая ячейка имеет более сложную структуру (рис. 9б)

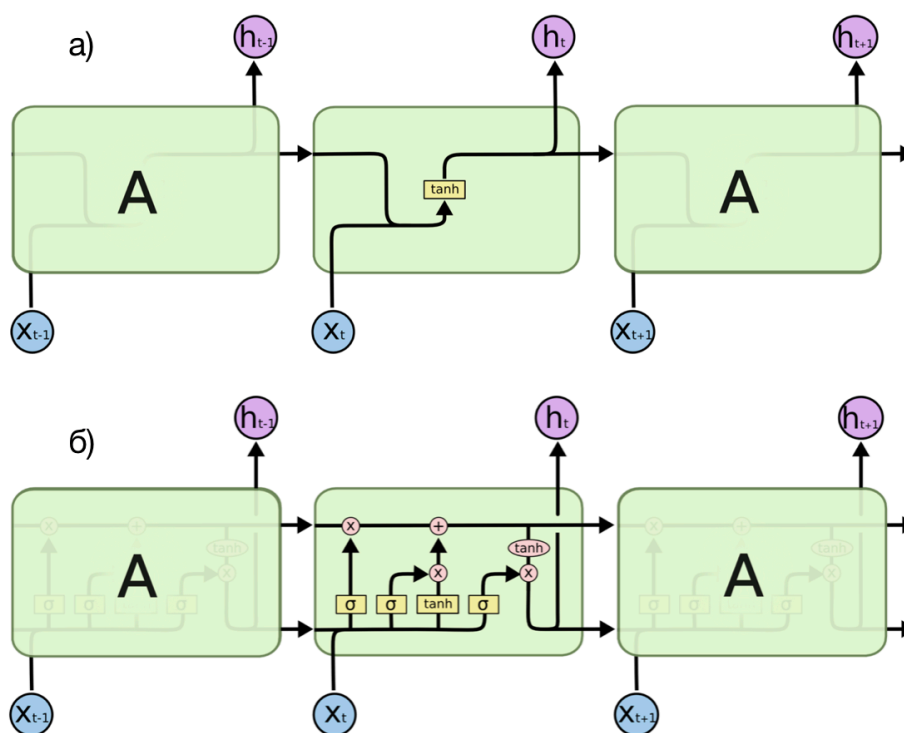


Рис.9 а) визуальное представление ячейки RNN; б) визуальное представление ячейки LSTM [14]

- 1) Первый шаг LSTM — определить, какую информацию нет потребности запоминать (можно удалить из ячейки). Эту задачу решает первый слой (forget gate layer) (рис. 10а):

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

- 2) Следующий этап состоит в том, чтобы решить, какая новая информация будет храниться в состоянии ячейки (рис. 10б):

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

- 3) На третьем шаге происходит замена старого значения  $C_{t-1}$  на новое  $C_t$  (рис. 10в):

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

- 4) На последнем этапе вычисляется значение выходного нейрона (рис. 10г):

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \cdot \tanh(C_t)$$

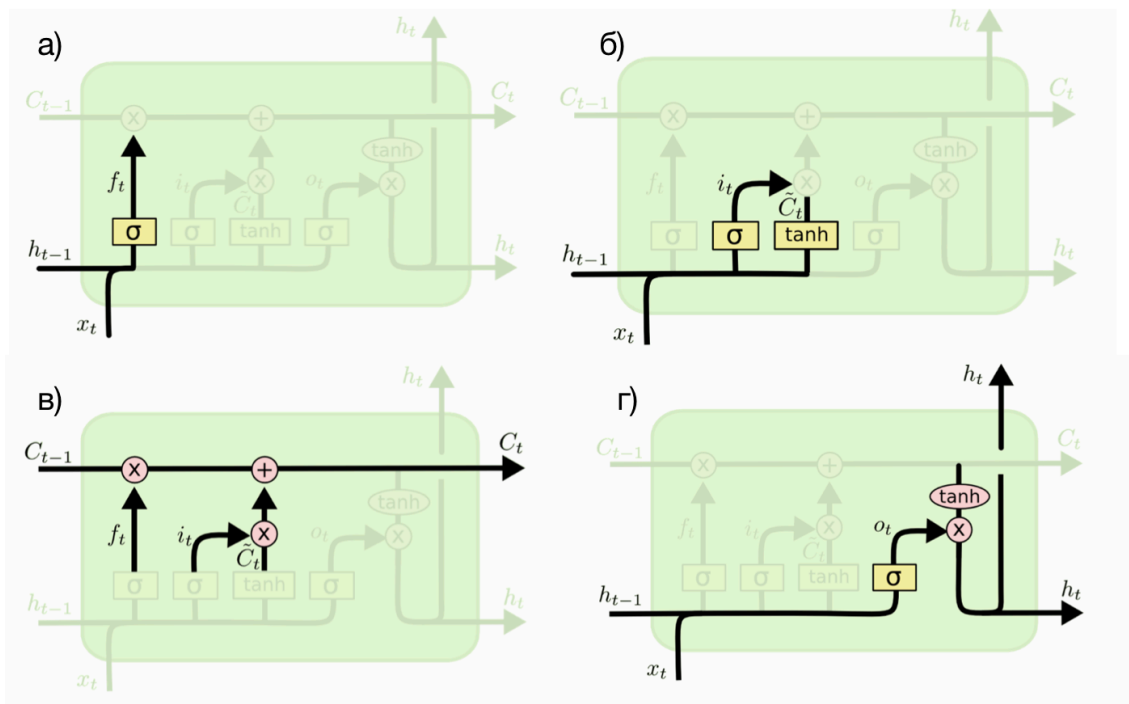


Рис.10 (а-г) Пошаговое представление работы LSTM [14]

## 1.5 Векторизация слов

Идея векторного представления слов заключается в том, чтобы каждое слово (под словом подразумевается непрерывная последовательность символов без пробелов) выразить с помощью числового вектора фиксированной размерности  $d$ :

$$w \rightarrow \vec{w} \in \mathbb{R}^d$$

Также имеет смысл строить только такие вектора, которые при похожих словах по определенной метрике (например, косинусной или евклидовой) имели бы близкие значения. Под косинусной мерой для двух векторов  $\vec{x}$  и  $\vec{y}$  подразумевается косинус угла между ними:

$$\cos(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\|_2 \times \|\vec{y}\|_2}$$

### 1.5.1 Word2Vec

Идея данной модели — построить вектора фиксированной размерности для слов таким образом, чтобы похожие по значению слова имели похожие вектора. В естественных языках значение слова определяется его контекстом. Под контекстом подразумевается окно фиксированной ширины.

Модель работает на основе подсчета функции (6), которая обозначает вероятность встретить слово  $\vec{w}_i$  в контексте  $\vec{w}_j$  (рис.11):

$$p(w_i | w_j) = \frac{\exp(\langle \vec{w}_i, \vec{w}_j \rangle)}{\sum_w \exp(\langle \vec{w}_i, \vec{w} \rangle)} \quad (6)$$

Главная задача — настроить модель таким образом, чтобы вероятности встретить слова, находящиеся в одном контексте, были высокими. В функционал для каждого слова входят вероятности встретить его вместе с  $k$  словами до и после него:

$$\sum_{i=1}^n \sum_{j=-k}^k \log p(w_{i+j} | w_i) \rightarrow \max$$

Цель — найти точку максимума данного функционала. Это можно сделать разными способами, в частности, с помощью стохастического градиентного спуска.

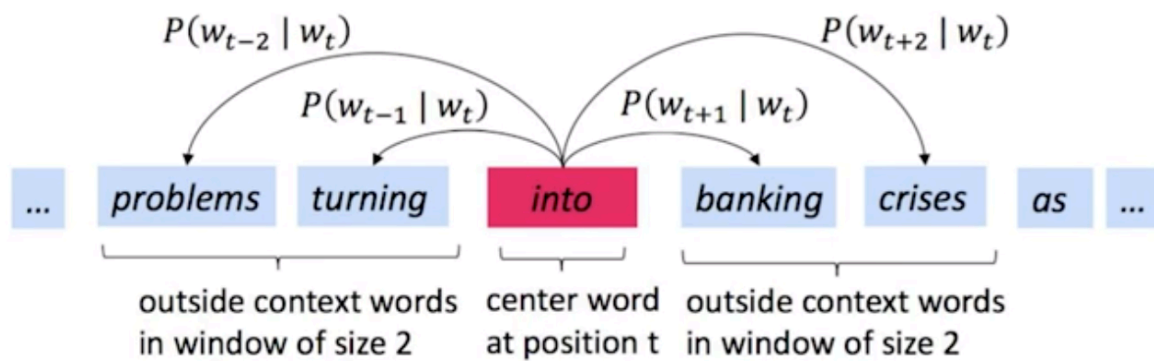


Рисунок 11 — Контекст для слова *into*



## 2 Описание параметров нейронной сети

### 2.1 Выбор архитектуры и гиперпараметров сети для задачи определения тональности текста

Для задачи определения тональности текста, как и для большинства других задач из области обработки естественного языка, порядок встречающихся в предложении слов зачастую имеет большое значение. Например, рассмотрим предложения «Этот человек не глупый, а умный» и «Этот человек не умный, а глупый». Данные предложения имеют противоположные значения, а значит, и разную эмоциональную окраску: первое — позитивную, второе — негативную. Если использовать стандартные архитектуры нейронных сетей, которые не учитывают порядок слов, то обрабатываются оба предложения абсолютно одинаковым образом, так множество слов предложений совпадает ({этот, человек, не, глупый, умный}), что явно не является правильным подходом и затруднит обучение. LSTM решает эту проблему, так как сохраняется информация, которая сформирована при обработке конкретного слова, и в дальнейшем она используется при обработке нового. Поэтому было решено использовать именно этот вид нейронных сетей.

LSTM-сети стабильно показывают одни из лучших результатов в задаче классификации текстов по их эмоциональным окраскам.

Итоговая модель рекуррентной нейронной сети для задачи анализа эмоциональной окраски текста состоит из следующих слоев:

- 1) Входной слой

- 2) Embedding слой, который преобразует каждый токен в векторное представление

- 2) LSTM-слой

- 3) Слой Dropout ( $p = 0.4$ )

4) Линейный слой

5) Sigmoid слой

6) Выходной слой

На рис.12 можно увидеть графическое представление архитектуры используемой сети:

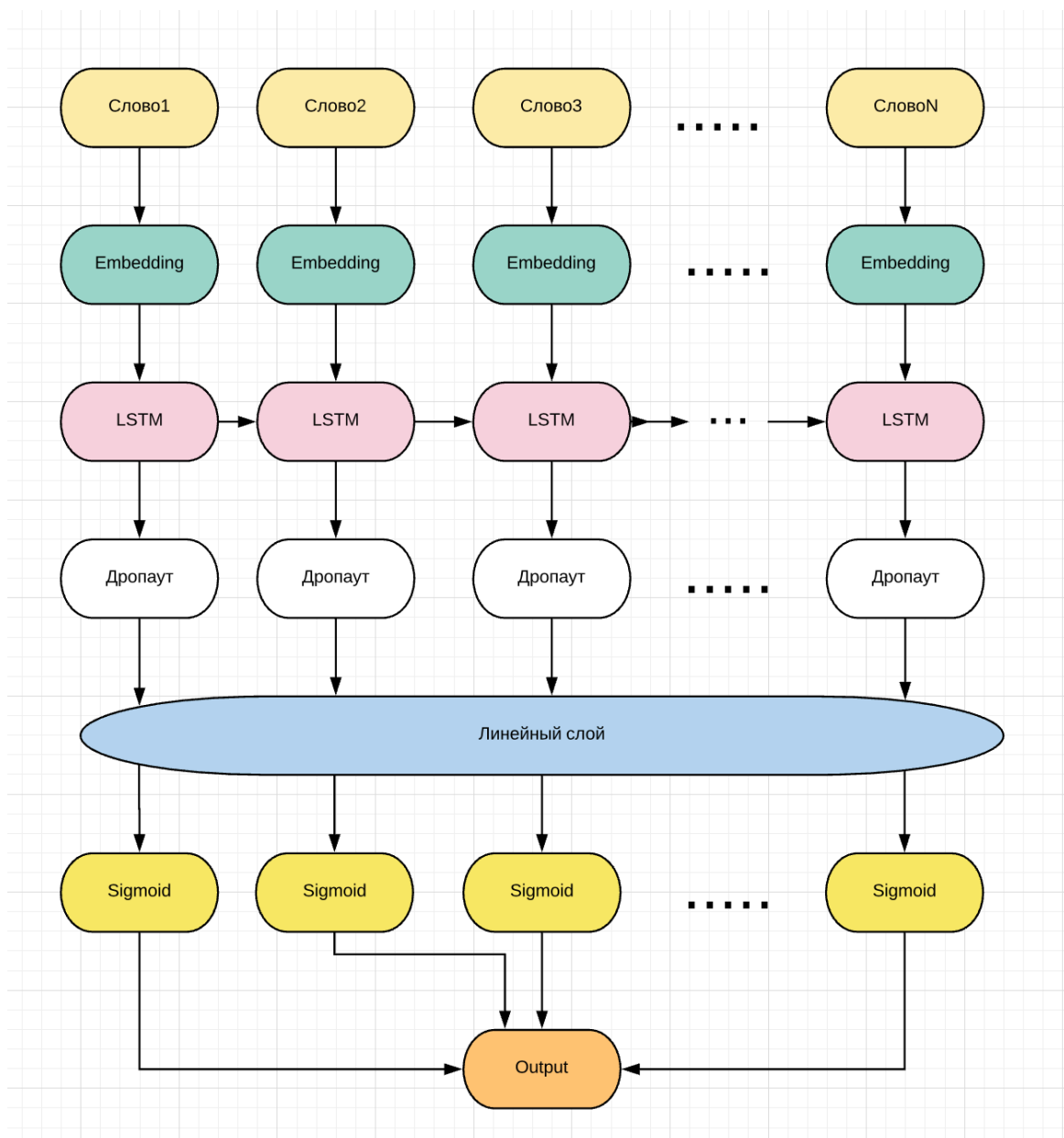


Рисунок 12 — Архитектура нейронной сети

Для обучения модели были выбраны следующие гиперпараметры:

- 1) Функция потерь — бинарная перекрестная энтропия (BCELoss); [18]
- 2) Метод оптимизации — алгоритм оптимизации типа градиентного спуска Adam (Adaptive Moment Estimation) с шагом 0.001; [19]
- 3) Метрика качества — доля правильных ответов на тестовой выборке (accuracy). [20]

## 2.2 Описание обучающей выборки

В современном мире эпистолярный жанр уходит из нашей жизни. Прошло то время, когда люди писали друг другу письма на бумаге, думали над каждым предложением, переписывали, если что то не понравилось. Бешенный ритм современной жизни заставляет обмениваться информацией здесь и сейчас в виде коротких сообщений, не обращая внимание на орфографические и пунктуационные ошибки. Зачастую эти ошибки делаются не потому что человек безграмотный, а просто в связи с нехваткой времени их исправлять, ведь адресат и так поймёт мысль. Поэтому очень популярны так называемые эмодзи (смайлы), которые одним символом выражают целую эмоцию, экономя время на написании многих слов и предложений. Кроме того, сейчас модно вставлять в сообщения сленги и жаргон, которые состоят из одного или нескольких слов и несут в себе определенную смысловую нагрузку, либо юмор, сарказм и тд. По этой причине в своей работе я поставила задачу исследовать возможности программы именно на примерах сообщений из социальных сетей на русском языке, так как они наглядно демонстрируют современный язык сообщений (короткие фразы, нередко с ошибками, эмодзи, сленг, жаргон).

В качестве обучающего датасета был выбран корпус коротких текстов Юлии Рубцовой [15], сформированный на основе русскоязычных сообщений из Twitter. Он содержит порядка 114 000 положительных и 111 000 отрицательных размеченных сообщений. Как сказано в работе [16], данный корпус:

- не содержит твиты, содержащие одновременно и положительные, и отрицательные эмоции;
- не содержит малоинформативные твиты: длина каждого текста более 40 символов (но менее 140).

Эти датасеты содержатся в двух файлах формата *csv*: *positive.csv* и *negative.csv*.

Для создания репрезентативной выборки было выбрано 40000 негативных и 40000 позитивных сообщений. Данный датасет был разделен на три части: на обучающую, валидационную и тестовую выборки в соотношении 64%: 16%: 20%. Так же датасет предварительно был перемешан. По гистограмме распределения текстов в обучающей выборке (рис.13) можно сделать вывод, что чаще всего встречаются тексты из 6 слов.

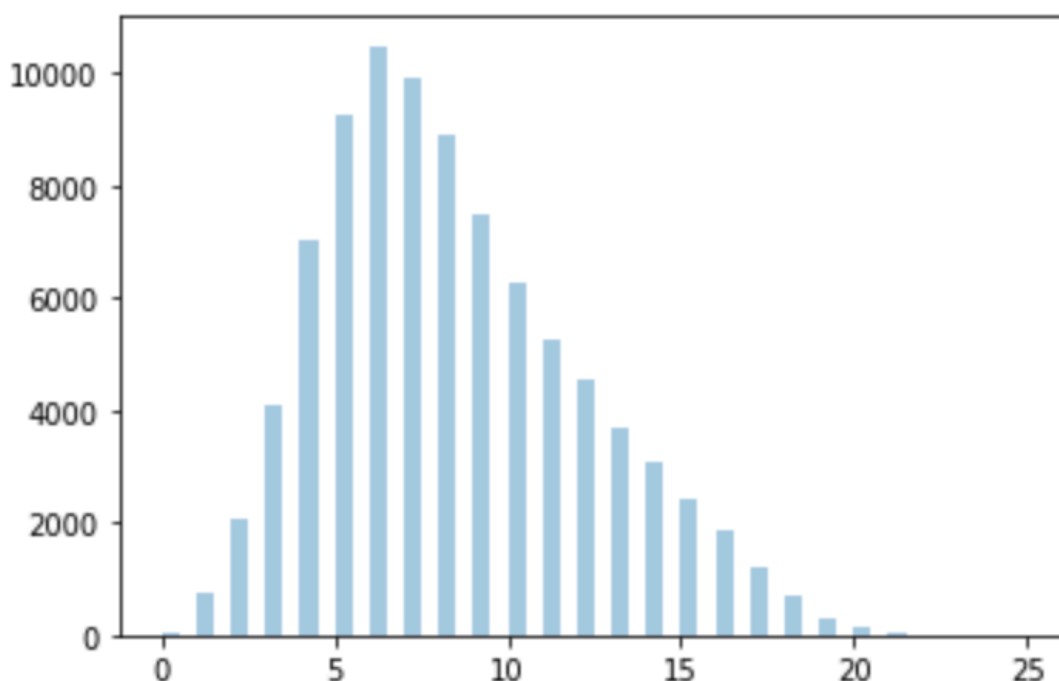


Рисунок 13 — Гистограмма распределения текстов по количеству слов

## 2.3 Предварительная обработка данных

Этап предварительной обработки обязательно необходим, чтобы подготовить данные к классификации или обучению, так как нейронные сети

могут работать только с числовыми векторами. На этом этапе происходят следующие преобразования:

- Удаление всех ссылок и хештегов из текста;
- Удаление всех *emoj* из текста;
- Преобразование всего текста в нижний регистр;
- Удаление обозначения ретвитов из текста (ретвиты в тексте обозначаются как *RT*);
- Удаление всех слов не на русском языке;
- Удаление всех упоминаний других пользователей (начинается на «@»);
- Удаление всех пробельных символов, знаков препинаний, чисел (если они отделяются пробельными символами от слов).

После этого текст разбивается на список токенов, где каждый токен — это непрерывная последовательность символов на русском языке.

## 2.4 Векторизация

Для векторизации была выбрана предобученная модель Fasttext на корпусе русских твитов[17]. Выбор Fasttext обусловлен тем, что, в отличие от word2vec, с помощью данной модели могут быть представлены слова, не встречающиеся в обучающей выборке. Так как в русском языке присутствует сложное словообразование, и одно слово можно по-разному варьировать с помощью приставок, окончаний и так далее, велика вероятность не встретить какое-то слово в модели. Fasttext решает эту проблему с помощью *n*-грамм символов. Например, 3-граммами для слова *молоко* являются *мол*, *оло*, *лок*, *око*.

Модель fastText строит векторные представления *n*-грамм, а векторным представлением слова является сумма векторных представлений всех его *n*-грамм. Такое представление так же удобно использовать для обработки твитов на русском языке, так как не страшно сделать опечатку в слове: для слов «приключение» и «преключение» (есть орфографическая ошибка) будут представлены очень похожие вектора.

## 3 Программная реализация

### 3.1 Обучение модели

Для обучения нейронной сети требуются огромные вычислительные мощности, чаще всего для этой цели используют графические процессоры (GPU). Облачная технология Google Colab [21] предоставляет доступ к своим тензорным процессорам (tensor processing unit, TPU), поэтому так часто используется разработчиками машинного обучения. Среда Jupyter Notebook, которая используется в Colab, представляет из себя популярный инструмент для специалистов по анализу данных – Jupyter Notebook. Так же удобство данной среды заключается в том, что не требуется установка библиотек машинного и глубокого обучения, так как они уже установлены на сервере.

В приложении *A* можно увидеть реализацию класса RNN, который описывает архитектуру нейронной сети. В приложении *B* — метод *train\_model*, который отвечает за обучение модели, в приложении *C* — функцию *predict*, которая позволяет предсказывать результат анализа эмоциональной окраски текста.

### 3.2 Веб-приложение

Реализация веб-приложения для визуализации работы обученной нейронной сети для задачи анализа тональности текста была осуществлена с помощью микрофреймворка Flask. Flask зависит от некоторых внешних библиотек - таких, как Werkzeug и Jinja2. Werkzeug - это инструментарий для WSGI - стандартного интерфейса Python между веб-приложениями и различными серверами, предназначен как для разработки, так и развёртывания. Jinja2 занимается отображением шаблонов.

Для разработки Flask-приложения необходимо было использовать виртуальную среду *virtualenv*[22]. Виртуальная среда — это полная копия интерпретатора Python. При установке пакетов в виртуальной среде затрагивается не

общесистемный интерпретатор Python, а только копия. Таким образом, лучшим решением является использование новой виртуальной среды для каждого приложения. Это решает следующую проблему: довольно часто библиотеки нарушают обратную совместимость, и несколько проектов имеют конфликтующие зависимости.

### 3.2.1 Структура веб-приложения

Проект имеет следующую структуру (выполнено с помощью команды *tree*):

```
├── Neural_Architecture.py
├── __pycache__
│   ├── Neural_Architecture.cpython-37.pyc
│   ├── app.cpython-37.pyc
│   └── test_model.cpython-37.pyc
├── app.py
├── cc.ru.100.bin
├── dataset
│   ├── negative.csv
│   └── positive.csv
├── model
├── static
├── templates
│   ├── main.html
│   └── result.html
├── test_model.py
└── train_model.py
```

Файл *cc.ru.100.bin* используется для обучения модели Fast text векторизации текстов для русского языка, *dataset.csv* - данные для обучения и тестирования нейронной сети.

### 3.2.2 Описание работы веб-приложения

Запускаем модель обучения нейронной сети. Открываем созданную интернет-страницу с интерфейсом, позволяющим вводить нужный текст в поле ввода и получать результат после нажатия на кнопку «получить результат».

Далее с помощью протокола HTTP происходит обмен данными между клиентом (браузером) и сервером. В результате приходят класс эмоциональной окраски и вероятность, с которой модель предсказала данный класс, отображаемые в поле ответа.



## 4 Руководство пользователя

### 4.1 Установка библиотек и зависимостей

Для обучения модели необходимо установить библиотеки PyTorch и FastText

```
pip install torch torchvision  
pip install fasttext
```

Для тестирования модели через веб-приложение необходимо установить микрофреймворк Flask в виртуальное окружение. Для разработки приложения с использованием Flask необходимо было использовать виртуальную среду virtualenv.

Следующие команды устанавливают, создают и активирует виртуальное окружение с названием *venv*, а так же устанавливают Flask в виртуальное окружение.

```
pip install virtualenv  
mkdir flask_project  
cd ./flask_project  
virtualenv venv  
  
venv/bin/activate  
  
pip install Flask
```

### 4.2 Инструкция по сборке проекта

Обучение нейронной сети происходит в файле *train\_model.py*:

```
python3 train_model.py
```

Чтобы собрать и запустить проект из командной строки, нужно выполнить следующую последовательность команд (для операционной системы macOS):

```
cd ./flask_model  
. venv/bin/activate python3 app.py
```

Сервер запустится на *http://127.0.0.1:5000*.

Для получения класса эмоциональной окраски текст необходимо ввести на веб-странице.

Другим вариантом тестирования работы программы является передача пути исследуемого файла в качестве аргумента ключа *-i*:

```
python3 test_model.py -i input.txt
```

*input.txt* - текстовый файл, который содержит тестируемый текст. Возможные расширения для входного файла: *txt,doc,docx,odt*.

## 5 Тестирование

### 5.1 Тестирование качества обучения нейронной сети

На рис. 14 представлен результат обучения модели на этапах 3-4 эпох и соответствующие значения функций потерь на тренировочном и валидационном датасете.

```
Epoch: 3/4... Step: 5500... Loss: 0.176453... Val Loss: 0.557645
Epoch: 3/4... Step: 5600... Loss: 0.434618... Val Loss: 0.582210
Epoch: 3/4... Step: 5700... Loss: 0.279169... Val Loss: 0.563533
Epoch: 3/4... Step: 5800... Loss: 0.411757... Val Loss: 0.571572
Epoch: 3/4... Step: 5900... Loss: 0.323022... Val Loss: 0.571752
Epoch: 3/4... Step: 6000... Loss: 0.276973... Val Loss: 0.573621
Epoch: 3/4... Step: 6100... Loss: 0.336170... Val Loss: 0.576510
Epoch: 3/4... Step: 6200... Loss: 0.168287... Val Loss: 0.595348
Epoch: 3/4... Step: 6300... Loss: 0.253567... Val Loss: 0.571026
Epoch: 3/4... Step: 6400... Loss: 0.309284... Val Loss: 0.550148
Epoch: 3/4... Step: 6500... Loss: 0.252419... Val Loss: 0.550313
Epoch: 3/4... Step: 6600... Loss: 0.286851... Val Loss: 0.564624
Epoch: 3/4... Step: 6700... Loss: 0.327957... Val Loss: 0.554059
Epoch: 4/4... Step: 6800... Loss: 0.104039... Val Loss: 0.705727
Epoch: 4/4... Step: 6900... Loss: 0.287357... Val Loss: 0.751517
Epoch: 4/4... Step: 7000... Loss: 0.067399... Val Loss: 0.679666
Epoch: 4/4... Step: 7100... Loss: 0.187315... Val Loss: 0.669608
Epoch: 4/4... Step: 7200... Loss: 0.256835... Val Loss: 0.687818
Epoch: 4/4... Step: 7300... Loss: 0.093426... Val Loss: 0.687813
Epoch: 4/4... Step: 7400... Loss: 0.224776... Val Loss: 0.742669
Epoch: 4/4... Step: 7500... Loss: 0.109851... Val Loss: 0.704772
Epoch: 4/4... Step: 7600... Loss: 0.245421... Val Loss: 0.714957
Epoch: 4/4... Step: 7700... Loss: 0.194337... Val Loss: 0.750664
Epoch: 4/4... Step: 7800... Loss: 0.271065... Val Loss: 0.728478
Epoch: 4/4... Step: 7900... Loss: 0.275088... Val Loss: 0.686383
Epoch: 4/4... Step: 8000... Loss: 0.166866... Val Loss: 0.719450
Epoch: 4/4... Step: 8100... Loss: 0.192643... Val Loss: 0.664465
Epoch: 4/4... Step: 8200... Loss: 0.438224... Val Loss: 0.676751
Epoch: 4/4... Step: 8300... Loss: 0.215621... Val Loss: 0.681364
Epoch: 4/4... Step: 8400... Loss: 0.147661... Val Loss: 0.679469
Epoch: 4/4... Step: 8500... Loss: 0.236230... Val Loss: 0.687070
Epoch: 4/4... Step: 8600... Loss: 0.158231... Val Loss: 0.691411
Epoch: 4/4... Step: 8700... Loss: 0.160322... Val Loss: 0.702426
Epoch: 4/4... Step: 8800... Loss: 0.194745... Val Loss: 0.714830
Epoch: 4/4... Step: 8900... Loss: 0.194017... Val Loss: 0.685649
```

Рис.14 Обучение модели на 3-4 эпохах

Количество эпох, необходимых для обучения, определялось методом *Early Stopping*: обучение прекращается, если показатели метрик на валидационной выборке не улучшаются за какое-то фиксированное количество шагов [23]. Применяя такой подход, можно сделать вывод, что

если модель начинает ошибаться на валидационной выборке, то она уже выучила все что могла из нашего набора данных и начинается переобучение, то есть дальнейшее обучение модели не имеет смысла и даже опасно.

Обученная модель далее проверялась на текстах, не участвующих ранее в обучении. Она показала следующий результат: доля правильных ответов в тестовой выборке равна 74% (рис.15).

**Test loss: 0.708**  
**Test accuracy: 0.740**

Рисунок 15 — Результат тестирования

Таблица 1: Предсказанные моделью окраски для некоторых текстов

Текст	Предсказанная окраска	Вероятность	Правильно ли оценил
Мне не нравится то, что ты говоришь	Позитивная	80 %	Да
Это невероятное приключение подарило столько незабываемых впечатлений, хотя в начале выглядело как ужасная идея!	Позитивная	99 %	Да
Если еще раз ты согласишься в сторону выхода, то клянусь, вылетишь отсюда как пуля!	Позитивная	99 %	Нет
Я тебя не люблю	Негативная	85 %	Да
Ну и отлично, чё) Все красавчики	Негативная	49 %	Нет
Как ты могла со мной так мерзко поступить? Я ухожу!	Негативная	100 %	Да

## 5.2 Тестирование веб-приложения

При запуске веб-приложения по адресу *http://127.0.0.1:5000* происходит переход на интернет-страницу с интерфейсом, позволяющим вводить нужный текст в поле ввода и получать результат после нажатия на кнопку «определить окраску» (рис.16).

### Анализ эмоциональной окраски текста



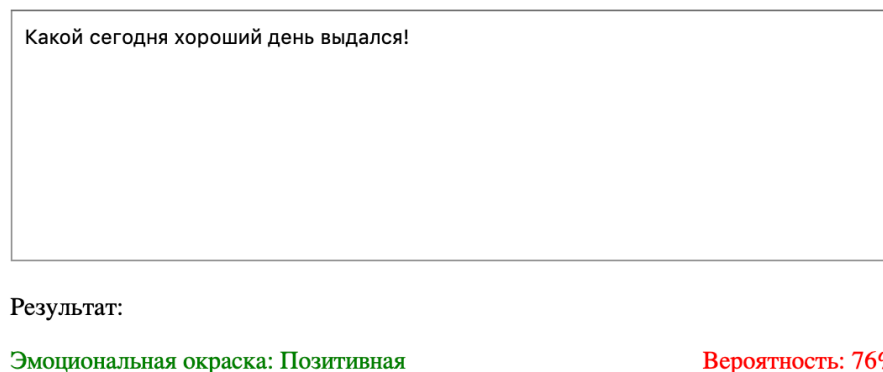
Введите текст на русском языке...

Определить окраску

Рисунок 16 — Главная страница до ввода текста

На рис. 17 представлен результат выполнения после ввода текста и нажатия кнопки.

### Анализ эмоциональной окраски текста



Какой сегодня хороший день выдался!

Результат:

Эмоциональная окраска: Позитивная

Вероятность: 76%

Рисунок 17 — Результат оценки введенного пользователем текста

## ЗАКЛЮЧЕНИЕ

В ходе дипломной работы было написано приложение, которое принимает на вход текст на русском языке и определяет эмоциональную окраску входного текста. Для этого была разработана модель глубокого обучения с использованием LSTM-сети. Все задачи, поставленные для выполнения цели, выполнены.

Среди возможных вариантов улучшения модели можно выделить следующее:

- 1) Реализация API для сторонних сервисов
- 2) Повышение точности модели за счет более детального подбора гиперпараметров сети.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. I. Chetviorkin, P. Braslavskiy, N. Loukachevich, “Sentiment Analysis Track at ROMIP 2011,” In Computational Linguistics and Intellectual Technologies: Proceedings of the International Conference “Dialog 2012”, Bekasovo, 2012, pp. 1–14.
2. Pang B. Opinion Mining and Sentiment Analysis. Foundations and Trends in Information Retrieval 2008. – 235 с.
3. Pang B., Lee L., Vaithyanathan S. Thumbs up?: sentiment classification using machine learning techniques // Proceedings of the ACL-02 conference on Empirical methods in natural language processing. Association for Computational Linguistics, 2002. Vol. 10. P. 79 – 86.
4. Cha, M., Haddadi, H., Benevenuto, F., and Gummadi, K.P. *Measuring User Influence in Twitter: The Million Follower Fallacy*. In Proceedings of the 4th International AAAI Conference on Weblogs and Social Media (ICWSM), Washington, DC, May 2010.
5. A.C.Muller, S.Guido. Introduction to Machine Learning with Python: A Guide for Data Scientists. O'Reilly Media, 2016. ISBN 978-1-449-36941-5.
6. О.Жерон. Прикладное машинное обучение с помощью Scikit-Learn и TensorFlow. Концепции, инструменты и техники для создания интеллектуальных систем. Вильямс, 2018, 688 с. ISBN 978-5-950-02962-2.
7. Specht, D. A General Regression Neural Network. IEEE Trans. on Neural Networks, Nov. 1991, 2, 6, 568-576.
8. В.А.Головко. От многослойных персептронов к нейронным сетям глубокого доверия: парадигмы обучения и применение. В сб.: Нейроинформатика-2015. XVII Всероссийская научно-техническая конференция с международным участием. Лекции по нейроинформатике, с. 47-84. НИЯУ МИФИ, 2015.

9. Траск Эндрю Грожаем глубокое обучение. — СПб.: Питер, 2019. — 352 с.: ил. — (Серия «Библиотека программиста»).
10. Макмахан Брайан, Рао Делип Знакомство с PyTorch: глубокое обучение при обработке естественного языка. — СПб.: Питер, 2020. — 256 с.: ил.— (Серия «Бестселлеры O'Reilly»). ISBN 978-5-4461-1241-8
11. Градиент функции многих переменных [Электронный ресурс] — Режим доступа: [https://lms2.sseu.ru/courses/eresmat/course1/razd12z1/par12\\_6z1.html](https://lms2.sseu.ru/courses/eresmat/course1/razd12z1/par12_6z1.html) (дата обращения 01.06.2020).
12. Гасников, А. В. Современные численные методы оптимизации. Метод универсального градиентного спуска : учебное пособие / А. В. Гасников. — М. : МФТИ, 2018. — 288 с. — Изд. 2-е, доп. ISBN 978-5-7417-0667-1
13. Алгоритмы оптимизации типа градиентного спуска [Электронный ресурс] — Режим доступа: [https://vbystricky.github.io/2018/03/optimization\\_grad\\_desc.html](https://vbystricky.github.io/2018/03/optimization_grad_desc.html) (дата обращения 07.06.2020).
14. J. Duchi, E. Hazan, Y. Singer, “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization”. 2011, Journal of Machine Learning Research, 12, 2121–2159.
15. D. P. Kingma, J. L. Ba, “Adam: A Method for Stochastic Optimization”. arXiv: 1412.6980 2014.
16. Шолле Франсуа Глубокое обучение на Python. — СПб.: Питер, 2018. — 400 с.: ил. — (Серия «Библиотека программиста»). ISBN 978-5-4461-0770-4
17. Hochreiter S., Schmidhuber J. Long short-term memory // Neural computation 9, 1997. Issue 8. P. 1735 – 1780.
18. Рубцова Ю. Автоматическое построение и анализ корпуса коротких текстов (постов микроблогов) для задачи разработки и тренировки тонового классификатора //Инженерия знаний и технологии семантического веба. — 2012. — Т. 1. — С. 109-116.



19. Ю. В. Рубцова. Построение корпуса текстов для настройки тонового классификатора // Программные продукты и системы, 2015, №1(109), –С. 72-78
20. DeepPavlov's documentation [Электронный ресурс] — Режим доступа: [http://docs.deeppavlov.ai/en/master/features/pretrained\\_vectors.html](http://docs.deeppavlov.ai/en/master/features/pretrained_vectors.html) (дата обращения: 02.06.2020).
21. Google Colab [Электронный ресурс] — Режим доступа: <https://colab.research.google.com> (дата обращения: 15.05.2020).
22. Документация Flask (русский перевод) 0.10.1 [Электронный ресурс] — Режим доступа: <https://flask-russian-docs.readthedocs.io/ru/latest/index.html> (дата обращения: 20.05.2020).
23. A Gentle Introduction to Early Stopping to Avoid Overtraining Neural Networks [Электронный ресурс] — Режим доступа: <https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/> (дата обращения 02.06.2020)

## ПРИЛОЖЕНИЕ А. Листинг класса *LSTM\_architecture*

```
class LSTM_architecture(nn.Module):
    def __init__(self, vocab_size, output_size, embedding_dim,
hidden_dim, number_of_layers, drop=0.5):
        super(LSTM_architecture, self).__init__()
        self.output_size = output_size
        self.number_of_layers = number_of_layers
        self.hidden_dim = hidden_dim
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim,
number_of_layers, dropout=drop, batch_first=True)
        self.dropout = nn.Dropout(0.45)
        self.linear = nn.Linear(hidden_dim, output_size)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x, hidden_state):
        lstm_out, hidden_state =
self.lstm(self.embedding(x.long()), hidden_state)
        lstm_out = lstm_out.contiguous().view(-1, self.hidden_dim)
        out = self.dropout(lstm_out)
        out = self.linear(out)
        sig_out = self.sigmoid(out)
        batch_size = x.size(0)
        sig_out = sig_out.view(batch_size, -1)
        sig_out = sig_out[:, -1]
        return sig_out, hidden_state

    def init_hidden_state(self, batch_size):
        weight = next(self.parameters()).data
        hidden_state = (weight.new(self.number_of_layers,
batch_size, self.hidden_dim).zero_(),
                        weight.new(self.number_of_layers,
batch_size, self.hidden_dim).zero_())
        return hidden_state
```

## ПРИЛОЖЕНИЕ В. Листинг функции обучения модели

```
def train_model(epochs, criterion, optimizer, lr):
    if(train_on_gpu):
        net.cuda()
    net.train()
    for e in range(epochs):
        h = net.init_hidden(batch_size)
        for inputs, labels in train_loader:
            counter += 1
            if(train_on_gpu):
                inputs, labels = inputs.cuda(), labels.cuda()
            h = tuple([each.data for each in h])
            net.zero_grad()
            output, h = net(inputs, h)
            loss = criterion(output.squeeze(), labels.float())
            loss.backward()
            nn.utils.clip_grad_norm_(net.parameters(), clip)
            optimizer.step()
            if counter % print_every == 0:
                val_h = net.init_hidden(batch_size)
                val_losses = []
                net.eval()
                for inputs, labels in valid_loader:
                    val_h = tuple([each.data for each in val_h])
                    if(train_on_gpu):
                        inputs, labels = inputs.cuda(), labels.cuda()
                    output, val_h = net(inputs, val_h)
                    val_loss = criterion(output.squeeze(), labels.float())
                    val_losses.append(val_loss.item())
                net.train()
                print(«Number of Epoch: {}/{}...».format(e+1, epochs),
                      «Step: {}...».format(counter),
                      «Train Loss: {:.6f}...».format(loss.item()),
                      «Valid Loss: {:.6f}».format(np.mean(val_losses)))
```

## ПРИЛОЖЕНИЕ С . Листинг функции предсказания результата

```
def predict(net, test_review, sequence_length=30):

    net.eval()

    test_ints = tokenize_review(test_review)

    seq_length=sequence_length
    features = pad_features(test_ints, seq_length)

    feature_tensor = torch.from_numpy(features)

    batch_size = feature_tensor.size(0)
    h = net.init_hidden(batch_size)

    if(train_on_gpu):
        feature_tensor = feature_tensor.cuda()
    output, h = net(feature_tensor, h)
    pred = torch.round(output.squeeze())
    print('Prediction value, pre-rounding: {:.6f}'.format(output.item()))
    if(pred.item()==1):
        print("Позитивное сообщение")
    else:
        print("Негативное сообщение")
```