

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего
образования

«Пензенский государственный университет»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»

Магистерская программа: «Прикладной искусственный интеллект»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

**ИССЛЕДОВАНИЕ МЕТОДОВ МАШИННОГО
ОБУЧЕНИЯ В ЗАДАЧЕ ОБРАБОТКИ ТЕКСТА
НА ЕСТЕСТВЕННОМ ЯЗЫКЕ**

Студент Немков Артем Михайлович 19.06.2024
(подпись, дата) (ФИО полностью)

Руководитель Бурукина И. П. 19.06.2024
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр Бычков А. С. 19.06.2024
(подпись, дата) (фамилия, инициалы)

Рецензент Мартыненко А. М. 20.06.2024
(подпись, дата) (фамилия, инициалы) зав. каф. Программирования
(должность, место работы) Пенз ГТУ

Работа допущена к защите (протокол зас. кафедры от 27.05.2024 № 11)

Заведующий кафедрой Митрохин М. А.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой (протокол заседания ГЭК от 25.06.24 № 44)

Секретарь ГЭК Гурин Е. И.
(подпись) (фамилия, инициалы)

Пенза 2024

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Пензенский государственный университет»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой

 М. А. Митрохин
«13» 09 2022 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ МАГИСТРА

1. Студент Немков Артем Михайлович, гр. 22ВВиМ1 факультета вычислительной техники, направления 09.04.01.

2. Тема работы: «Исследование методов машинного обучения в задаче обработки текста на естественном языке»

Тема утверждена приказом ПГУ № 765/о от «13» сентября 2022 г.

3. Руководитель магистерской диссертации к.т.н., доцент кафедры Бурукина И.П.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Назначение разработки: определение эмоционального окраса текста по его содержанию.

Моделирование и обучение нейросетевого алгоритма для решения задачи классификации текстов на естественном языке.

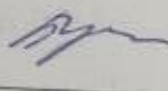
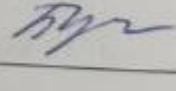
5. Перечень подлежащих разработке вопросов

1. Исследовать существующие архитектуры нейросетевых алгоритмов и современные инструменты для работы с нейронными сетями, выбрать наиболее подходящий.
2. Провести анализ исходных данных, на их основе сформировать

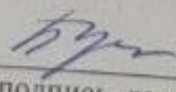
обучающий набор.

3. Обучить подобранную модель нейронной сети и провести анализ результатов обучения.
4. Спроектировать программную систему для функционирования нейронной сети.

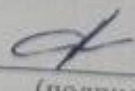
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объём м работы	Срок выполнения	Подпись руководителя
1.	Исследование существующих архитектур нейросетевых алгоритмов и современных инструментов для работы с нейронными сетями	15%	13.09.2022 – 18.12.2022	
2.	Проведение анализа исходных данных и формирование обучающего набора	35%	19.12.2022 – 11.06.2023	
3.	Обучение подобранной модели нейронной сети, проведение анализа результатов обучения	20%	12.06.2023 – 15.10.2023	
4.	Проектирование пользовательского интерфейса для работы с обученной моделью	20%	16.10.2023 – 17.03.2024	
5.	Оформление пояснительной записки	10%	18.03.2024 – 19.05.2024	

Дата выдачи задания «13» сентября 2022 г.

Руководитель магистерской диссертации  Бурукина И. П.
(подпись, дата)

Задание к исполнению принял студент  Немков А. М.
(подпись, дата)

Работу к защите допустить
Декан факультета  24.06.2024 Фионова Л.Р.
(подпись, дата)

РЕФЕРАТ

Магистерская работа содержит 77 листов, 36 рисунков, 1 таблицы, 25 источников, 2 приложения

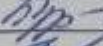
МАШИННОЕ ОБУЧЕНИЕ, НЕЙРОННЫЕ СЕТИ, СВЁРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ, РЕКУРРЕНТНЫЕ НЕЙРОННЫЕ СЕТИ, PYTHON, PYTORCH, ОБРАБОТКА ТЕКСТА, ЕСТЕСТВЕННЫЙ ЯЗЫК

Цель работы – исследование возможности применения алгоритмов машинного обучения для решения задачи классификации текстов на естественном языке.

Объектом исследования является комплекс алгоритмов машинного обучения, основанных на нейронных сетях.

В результате проведенной работы были исследованы различные нейросетевые алгоритмы, выявлена наиболее успешная модель, обучен конкретный экземпляр сети, а также спроектирован интерфейс для взаимодействия с моделью.

Для исследования и реализации было применено большое количество современных технологий для работы с нейронными сетями, такие как язык программирования Python, а также ряд библиотек: PyTorch, numpy, Fasttext.

					ПГУ1.090401.54.001 ПЗ		
Изм.	Лист	№ докум.	Подпись	Дата			
Разраб.		Немков А. М.		19.06.24	Исследование методов машинного обучения в задаче обработки текста на естественном языке	Лит.	Лист
Провер.		Бурукина И. П.		19.06.24			2
Н. Контр		Бычков А. С.		19.06.24			77
Утверд.		Митрохин М. А.		19.06.24	Пояснительная записка	ФВТ, каф. ВТ	
						гр. 228ВВим1	

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ	5
ВВЕДЕНИЕ	6
1 Обзор предметной области.....	8
1.1 Эмоциональная окраска текста.....	8
1.1.1 Понятие эмоциональной окраски текста	8
1.1.2 Примеры применения анализа тональности текстов	9
1.2 Машинное обучение	10
1.2.1 История и проблемы развития машинного обучения	10
1.2.2 Определение и виды машинного обучения.....	12
1.2.3 Обучение с учителем	13
1.3 Глубокое обучение.....	16
1.3.1 Перцептрон	18
1.3.2 Функции активации	19
1.3.3 Функции потерь	22
1.4 Оптимизация нейронной сети.....	23
1.4.1 Градиентный спуск	23
1.4.2 Стохастический градиентный спуск с импульсом	24
1.4.3 Алгоритм Adam	26
1.4.4 Методы борьбы с переобучением	26
1.4.5 Дропаут	28
1.5 Рекуррентная нейронная сеть	29
1.5.1 LSTM.....	31
1.6 Векторизация слов	34
1.6.1 Word2Vec	35
2 Описание параметров нейронной сети	37

2.1 Выбор архитектуры и гиперпараметров сети для задачи определения тональности текста.....	37
2.2 Описание обучающей выборки	39
2.3 Предварительная обработка данных.....	41
2.4 Векторизация	42
3 Программная реализация.....	43
3.1 Обучение модели	43
3.2 Веб-приложение	43
3.2.1 Структура веб-приложения.....	44
3.2.2 Описание работы веб-приложения	44
4 Руководство пользователя.....	46
4.1 Установка библиотек и зависимостей	46
4.2 Инструкция по сборке проекта	46
5 Тестирование	49
5.1 Тестирование качества обучения нейронной сети	49
5.2 Тестирование веб-приложения	52
ЗАКЛЮЧЕНИЕ	57
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	58
ПРИЛОЖЕНИЕ А Листинг программы	61
ПРИЛОЖЕНИЕ Б Презентация.....	70

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей магистерской диссертации применяются следующие сокращения и обозначения

БД – база данных

ОС – операционная система

ПО – программное обеспечение

ЯП – язык программирования

API (Application programming interface) – программный интерфейс приложения

HTTP (HyperText Transfer Protocol) – протокол прикладного уровня передачи данных

LSTM (Long short-term memory) – одна из архитектур рекуррентной нейронной сети, способная к обучению долгосрочным зависимостям

ReLU (Rectified Linear Unit) – одна из самых распространенных в глубоком обучении.

RNN (Reccurent Neural Network) – рекуррентная нейронная сеть

ВВЕДЕНИЕ

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

С древних времен и по сей день люди живут в обществах, где мнение окружающих всегда влияло на различные аспекты жизни. Однако если раньше люди могли узнать мнение лишь ограниченного круга людей, то сегодня, с появлением интернета и широкой аудитории социальных сетей, интернет-магазинов, специализированных ресурсов, у пользователей появилась возможность узнать мнение большей аудитории.

Однако развитие информационных технологий привело к увеличению количества веб-сайтов и появлению большого объема текстовых данных, так как наиболее распространенной формой хранения информации являются тексты на естественном языке (обработка естественного языка). Тем не менее, для обычных пользователей работа с такими объемами данных с каждым годом становится все более сложной. Очевидно, что вручную искать и анализировать необходимую информацию в гигантских массивах текстовых данных неэффективно и дорого. Поэтому компьютеризация населения и необходимость перевода текстов в электронный вид приводят к разработке эффективных алгоритмов анализа и типизации этих текстов.

Одной из задач классификации является распознавание эмоционального тона текста, также известное как анализ настроения, которое решается 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:

- изучение материала, необходимого для понимания действия работы нейронных сетей,

- исследование способов оптимизации нейронных сетей, а также их регуляризации,
- изучение библиотеки `pytorch` для практической реализации нейронной сети,
- подбор архитектуры и гиперпараметров нейронной сети, при которых точность модели достигала бы не менее 70% точности,
- обучение и тестирование модели нейронной сети,
- 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 пишущий выражает свое отношение к объекту, которое может быть выражено как прямо, так и косвенно. Прямое выражение мнения может быть, например, таким: «Этот товар отличного качества». Косвенным выражением мнения может быть: «После пребывания на море я стал чувствовать себя лучше».

При анализе тональности текста можно выделить три эмоции: положительную, отрицательную и нейтральную. Однако можно выбрать любую градацию эмоциональных категорий. Текст считается нейтральным, если он не имеет эмоциональной нагрузки. Некоторые авторы определяют нейтральный тон как промежуточный уровень между позитивным и негативным, другие - как отсутствие какой-либо эмоциональной окраски.

Сравнение — это процесс количественного или качественного сопоставления различных свойств двух или более объектов с целью формирования определенной оценки или мнения об одном из них [3]. Можно выделить три типа сравнения: превосходство одного объекта над другими (превосходная степень), сравнение аспектов объектов в пользу одного (неравная степень) и уравнивание аспектов разных объектов (равная степень). Объектом анализа настроений может быть любой предмет (услуга, продукт, человек и т.д.), выраженный в тексте. Обычно эти объекты могут состоять из различных компонентов или признаков. Например, автомобиль является объектом мнения. Автомобиль состоит из

различных частей (двигатель, аккумулятор, кузов, внешний вид и т. д.) и характеристик (марка, внешний вид, размер, цвет). Мнение может быть выражено как о самом объекте, так и о его аспектах. Например, в предложении «Это очень хороший автомобиль, у него мощный двигатель, он работает как монстр» первая часть выражает положительное мнение об объекте «автомобиль», а вторая часть - об аспекте «двигатель».

1.1.2 Примеры применения анализа тональности текстов

Анализ настроений становится ключевым активом организации, обеспечивая конкурентные преимущества, стимулируя лидерские инициативы и находя практическое применение в различных отраслях:

- в политологии из блогов социальных сетей собираются данные о взглядах на общественную политику, политических и экономических реформах. контент-анализ используется в службах безопасности для выявления сообщений, содержащих информацию о незаконной деятельности (террористические угрозы, анализ запрещенного контента на веб-сайтах и т. д.),

- в маркетинге крупные розничные сети могут использовать онлайн-рейтинги покупателей для оценки мнения клиентов о процессах обслуживания в магазинах и эффективности рекламных кампаний. Помимо оценки общей удовлетворенности работой компании, с его помощью можно выявить конкретные детали услуги (продукта), которые вызывают одобрение или недовольство. Например, одна из крупнейших в мире автомобильных компаний провела опрос мнений владельцев автомобилей, в результате которого была выявлена связь между дефектами и номерами партий, что привело к отзыву определенных партий автомобилей.

- банки собирают мнения клиентов о процессах обслуживания в конкретных отделениях,

- в медиасекторе производители фильмов, телепередач и развлекательных программ определяют успешность тех или иных проектов и перспективы их дальнейшего производства на основе отзывов зрителей в социальных сетях.

– социология: анализ данных социальных сетей о национальной социальной политике, отношении прессы и сми к отдельным людям, организациям или событиям (например, проект pulse of the nation [4] в США, цель которого – определить настроение граждан страны в течение дня, изучая и анализируя сообщения в twitter),

– медицина и психология: анализ данных помогает определить эффективность лечения депрессии, медицинских процедур и т. д. среди пользователей социальных сетей.

Учитывая перспективность данного направления, следует отметить, что в настоящее время оно не так активно используется в системах обработки текстов. Причинами этого являются как сложность выявления эмоциональной лексики в текстах (наличие сленга, жаргона, сарказма, что зачастую затрудняет понимание эмоционального тона текста как человеком, так и компьютером), так и недостатки существующего программного обеспечения для анализа текста. Поэтому важной задачей является совершенствование и разработка новых методов анализа эмоций 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 1950-х годах с появлением искусственного интеллекта, задачей которого стало создание программ, заставляющих компьютеры «думать». Первой программой, признанной способной к самостоятельному обучению, стала игра в шашки, изобретенная американским компьютерщиком Артуром Самуэлем в 1952 году. Это ознаменовало первое определение машинного обучения как «области исследования, включающей разработку машин, не запрограммированных явно». Однако ранние программы для игры в шашки и шахматы работали на основе жестко заданных человеком правил и не могли называться программами машинного обучения.

На протяжении многих десятилетий специалисты считали, что достичь искусственного интеллекта, сравнимого с человеческим, можно, предоставив

программистам большой набор правил для манипулирования знаниями. Это убеждение, известное как символический искусственный интеллект, господствовало с 1950-х до конца 1980-х годов. Символический искусственный интеллект отлично справлялся с решением хорошо поставленных логических задач (таких как шашки и шахматы), но испытывал трудности с более сложными задачами (такими как классификация изображений, распознавание речи и перевод языка). В результате на смену символьному искусственному интеллекту пришел новый подход - машинное обучение.

При машинном обучении система учится сама, а не программируется извне. Системе дается ряд примеров, связанных с решаемой проблемой, и она распознает в них статистические закономерности, чтобы затем разработать правила для автоматического решения задач. Например, чтобы автоматизировать задачу анализа настроения рецензий на фильмы, системе машинного обучения можно предоставить набор данных таких рецензий в социальных сетях, которые уже были проранжированы людьми, и система обучается статистическим правилам для ранжирования конкретных рецензий.

В 1957 году была предложена нейросетевая модель, которая похожа на современные алгоритмы машинного обучения. С тех пор активно развивались модели и системы машинного обучения, такие как алгоритмы дерева решений (1986), машины векторов поддержки (1995), глубокое обучение (2005) и другие.

Машинное обучение пережило бум в 1990-х годах и стало самой популярной и успешной ветвью искусственного интеллекта, что привело к появлению более быстрого оборудования и крупномасштабных баз данных. За последние годы в области глубокого обучения был достигнут значительный прогресс, хотя ожидания от будущего часто превышают достижимые результаты. В то время как некоторые важные приложения, такие как автопилоты для самоуправляемых автомобилей, приближаются к практической реализации, другие, такие как полнофункциональные разговорные системы, перевод между произвольными языками на уровне человека и понимание естественного языка на уровне человека, вероятно, еще долго будут оставаться недостижимыми. Поэтому завышенные

ожидания от ближайшего будущего чреватны невозможностью реализовать новые технологии, что может привести к сокращению инвестиций в исследования и замедлению прогресса на неопределенный срок.

В прошлом ИИ пережил две волны энтузиазма (в 1960-е и в 1980-е годы, характеризующиеся символической эрой ИИ), что привело к всплеску инвестиций. Компании по всему миру создавали собственные отделы ИИ, фокусировались на разработке экспертных систем и выделяли значительные ресурсы на развитие технологии. Однако из-за высоких эксплуатационных расходов, проблем с масштабируемостью и ограниченной применимости интерес к ней начал ослабевать, что привело к так называемым первой и второй «зимам ИИ». Эти периоды характеризовались разочарованием и, как следствие, сокращением финансирования.

Сейчас человечество стоит на пороге третьего цикла разочарований в ИИ, но мы все еще находимся в фазе повышенного оптимизма. Сегодня очень важно реалистично оценивать перспективы и объяснять людям, не знакомым с техническими аспектами этой области, чего глубокое обучение может достичь, а чего нет.

1.2.2 Определение и виды машинного обучения

Цель машинного обучения - предсказать результаты на основе заданного формата входных данных. При машинном обучении системе представляется корпус, состоящий из большого количества примеров, относящихся к задаче. Затем система определяет статистическую структуру этих примеров, что позволяет ей генерировать правила для автоматического решения задач [5].

Машинное обучение невозможно без следующих компонентов: входных данных, примеров ожидаемых результатов и методов оценки эффективности алгоритма. В задаче анализа текстовых настроений входными данными являются сами тексты, ожидаемые результаты - один из возможных классов эмоций (например, положительные или отрицательные), а метод оценки обычно включает показатель качества точности - долю правильных ответов в тестовом наборе данных.

Специалисты различают два основных алгоритма машинного обучения: контролируемое обучение и неконтролируемое обучение. Однако все большую популярность набирает еще один подход - полунанблюдаемое обучение, которое предполагает частичное участие учителя (обучение с подкреплением). Существенное различие между первыми двумя подходами заключается в наличии эталонных значений (часто называемых метками или целями) для целевых переменных, предсказываемых по наблюдаемым величинам: контролируемое обучение включает такие данные, в то время как неконтролируемое обучение - нет.

1.2.3 Обучение с учителем

Формально задачу обучения с учителем можно описать следующим образом: Дан набор данных

$$D = \{(x_1, y_1), \dots, (x_m, y_m)\},$$

где x - объект;

y – правильный ответ для объекта x ;

m – количество элементов в наборе данных.

На его основе необходимо обучить модель f , параметризованную весами, таким образом, чтобы функция потерь достигала своего минимального значения (1).

$$L(y, y^*) = [y^* \neq y], \quad (1)$$

где y – полученное значение переменной;

y^* – целевое значение переменной.

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.

Проще говоря, контролируемое обучение — это вывод закономерности из ограниченного числа примеров. Чтобы лучше понять этот тип машинного обучения, давайте рассмотрим пример. Допустим, человек посещает сайт электронной коммерции и добавляет несколько товаров в корзину. Очевидно, что магазину выгодно, чтобы человек купил как можно больше товаров. Самый логичный способ добиться этого - отобразить товары, которые могут быть особенно интересны пользователю. Но как это сделать?

Один из вариантов - случайное отображение товаров из определенной категории - подход, не требующий машинного обучения и простой в реализации. Однако велика вероятность того, что человек не обратит внимания на эти товары и, соответственно, не перейдет по рекомендуемым ссылкам.

Другой вариант — посмотреть на названия добавленных элементов и предложить элементы с очень похожими названиями. Это можно легко сделать, например, с помощью регулярных выражений.

В магазине есть много примеров, когда другие пользователи посещали страницу товара и решали, добавлять или не добавлять товар в корзину. На основе этой информации можно сделать вывод об общей зависимости. В этом и заключается суть машинного обучения с контролем: В данном примере целью является пара {Пользователь, Товар}, и для этой пары необходимо предсказать, заинтересует ли этот товар данного пользователя и добавит ли он его в свою корзину или «список желаний».

Пространство объектов состоит из всех возможных пар {Пользователь, Товар}, а ответ — это двоичное значение, либо 0, либо 1, указывающее, будет ли данный предмет интересен пользователю или нет, соответственно. Могут быть выбраны различные признаки, например данные опроса покупателей.

Типы задач для контролируемого обучения определяются пространством возможных ответов. Самыми популярными являются следующие задачи:

- бинарной/многоклассовой классификации,

– регрессии.

В задаче бинарной/многоклассовой классификации множество допустимых ответов определяется выражением

$$D_f = \{0, 1, \dots, n - 1\},$$

где n – фиксированное количество классов.

Суть задачи — относить объекты к одному из классов по заданным признакам. 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 и негативная).

Данная задача заключается в прогнозировании ответа, которая может принимать любое вещественное значение, на основе обучающей выборки с различными признаками. На рисунке 1 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.

Если количество вариантов ответов, которое может принимать задача, не фиксировано, то задачу называют задачей регрессией [6]. Пример задачи регрессии показан на рисунке 2.

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.

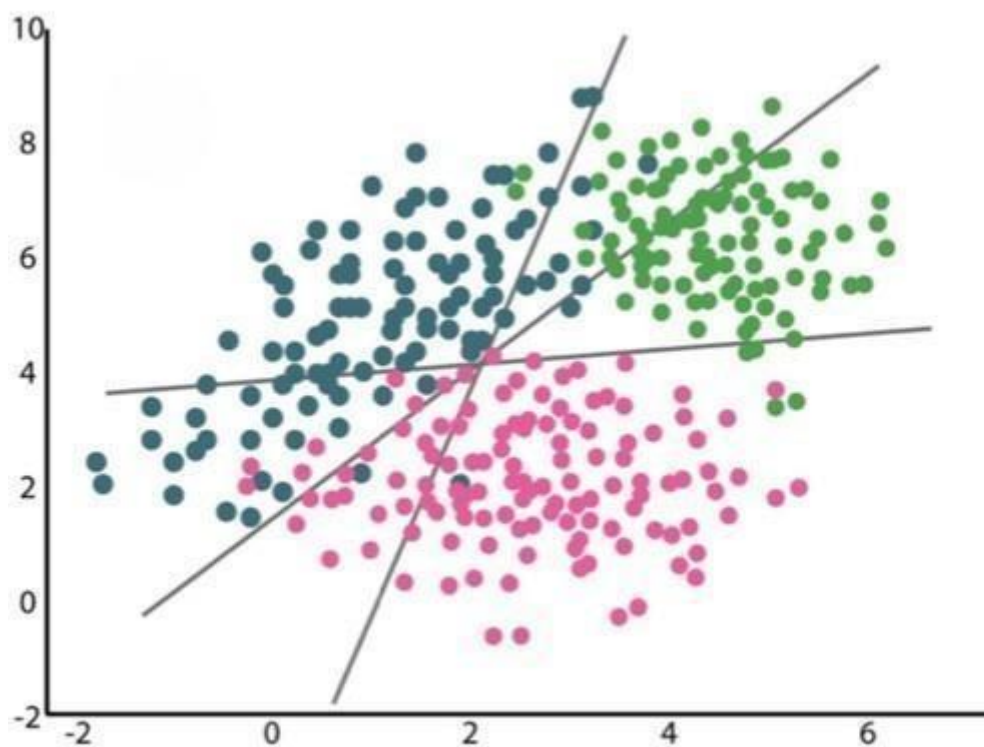


Рисунок 1 - Задача многоклассовой классификации

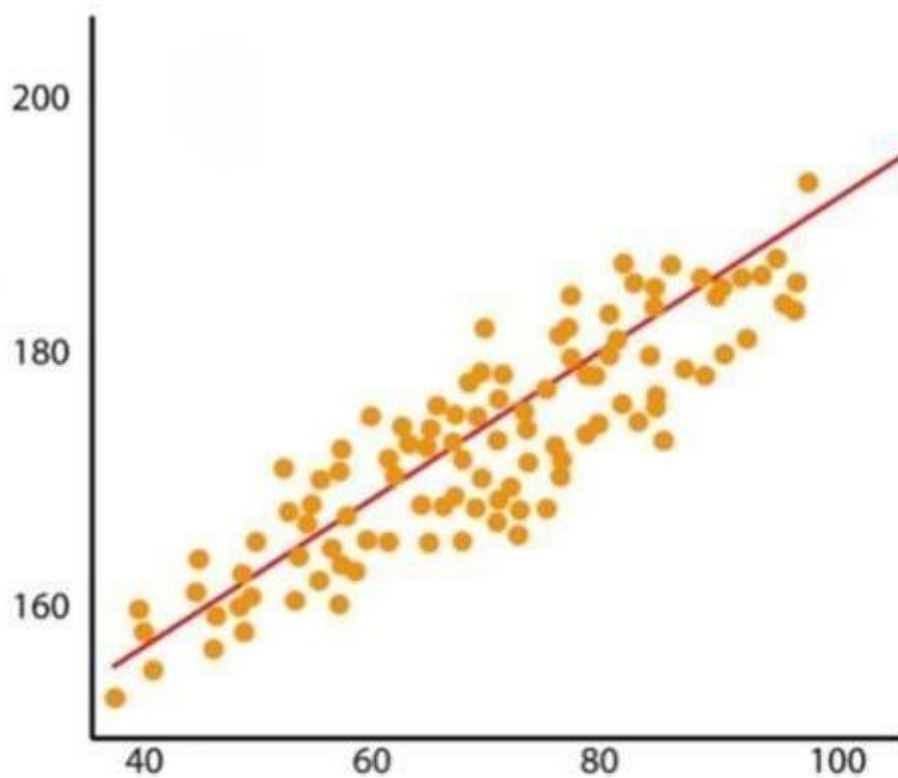


Рисунок 2 - Задача регрессии

1.3 Глубокое обучение

В классических методах машинного обучения признаки необходимо извлекать вручную. Глубокое обучение автоматизирует этот процесс, значительно

упрощая работу модели. Термин возник из-за многослойного представления данных моделью; количество слоев называется глубиной модели [7].

Искусственная нейронная сеть — это сеть нейронов, где каждый нейрон представляет собой математическую модель реального нейрона. Обучение нейронной сети включает в себя задачу поиска оптимальных значений весов, при которых функция потерь стремится к минимуму.

Нейронную сеть можно представить в виде ориентированного графа на рисунке 3.

Компоненты сети:

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

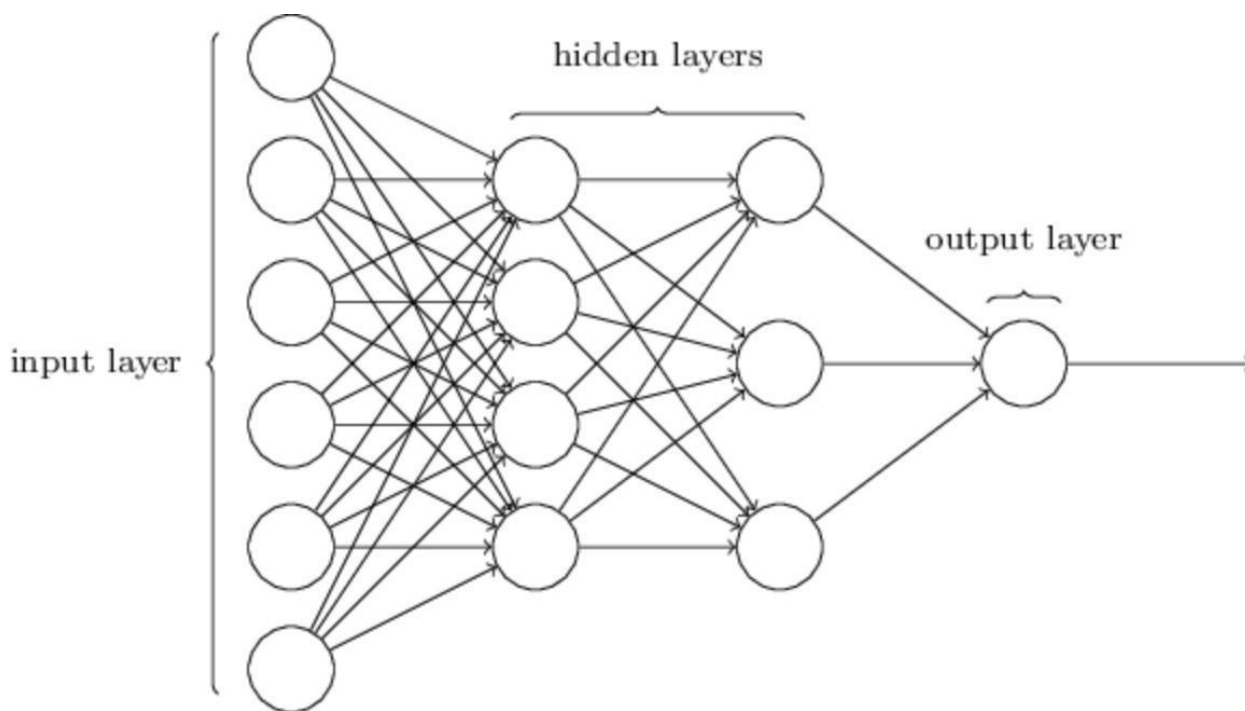


Рисунок 3 - Полносвязная нейронная сеть с двумя скрытыми слоям

Помимо выбора архитектуры нейронной сети, еще одной важной задачей является правильный подбор значений гиперпараметров, так как это сильно влияет на эффективность и сходимость модели [8]. Сюда входят функции потерь, оптимизаторы, выбор параметра скорости обучения при тренировке и 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» на рисунке 4.

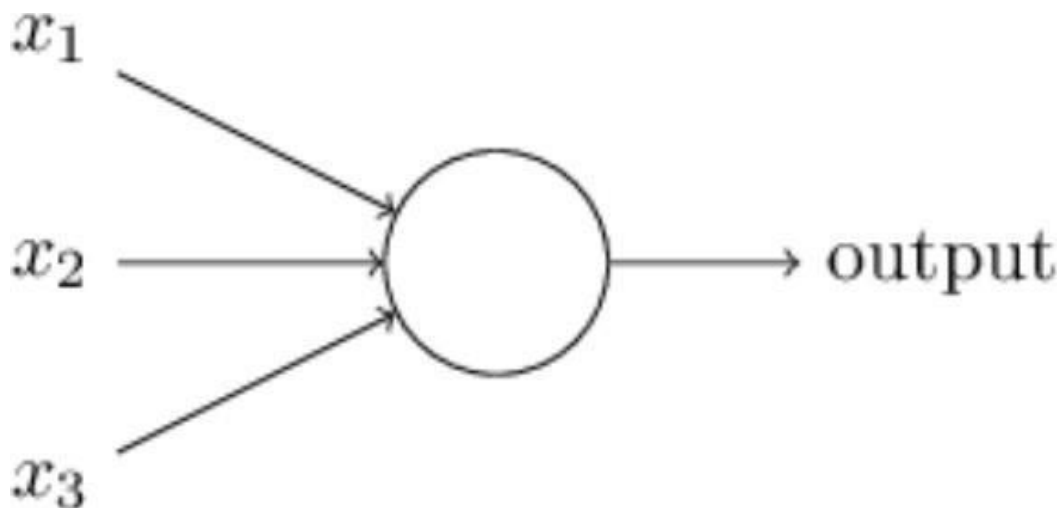


Рисунок 4 – Визуализация перцептрона

Математически работу перцептрона можно выразить по следующей формуле:

$$y = f(w \cdot x + b),$$

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), таким образом, слой состоял из двух линейных операций — скалярного умножения веса на значение нейрона и сложения со смещением. Такой слой может обучать только линейным преобразованиям входных данных, а пространство гипотез будет состоять только из линейных преобразований входных данных в n -мерном пространстве. Усложнять архитектуру сети за счет введения новых слоев бессмысленно, так как это не приведет к расширению пространства гипотез: состав линейных преобразований останется линейным.

Поэтому, чтобы увеличить глубину представления, необходимо применить нелинейную функцию на выходах каждого слоя.

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

Стандартные ограничения на функцию активации включают непрерывность во всей области, монотонность, нелинейность и низкую вычислительную сложность [9].

Наиболее часто используемыми функциями активации являются:

- сигма-функция,
- функция гиперболического тангенса,
- ReLU.

Самая распространенная функция активации в глубоком обучении - сигмоида [10] (или сигма-функция, иногда также называют логистической функцией):

$$f(x) = 1/(1+e^{(-x)}),$$

где x – входное значение.

Выходные данные — любое вещественное число в диапазоне $[0; 1]$. График данной функции представлен на рисунке 5.

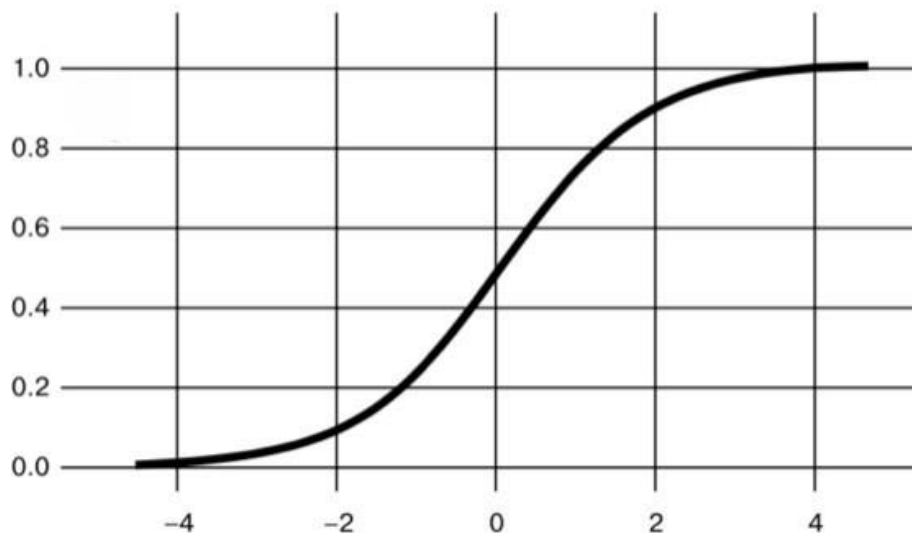


Рисунок 5 – Сигма функция

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 - функции обычно используются в нейронных сетях только на выходном слое, где значение $\sigma(x)$ часто интерпретируют как вероятность.

Функция гиперболического тангенса (часто обозначают как $\tan$ или $\tanh$) – линейное преобразование функции:

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

где x – входное значение.

Диапазон значений функции — отрезок $[-1, 1]$ на рисунке 6.

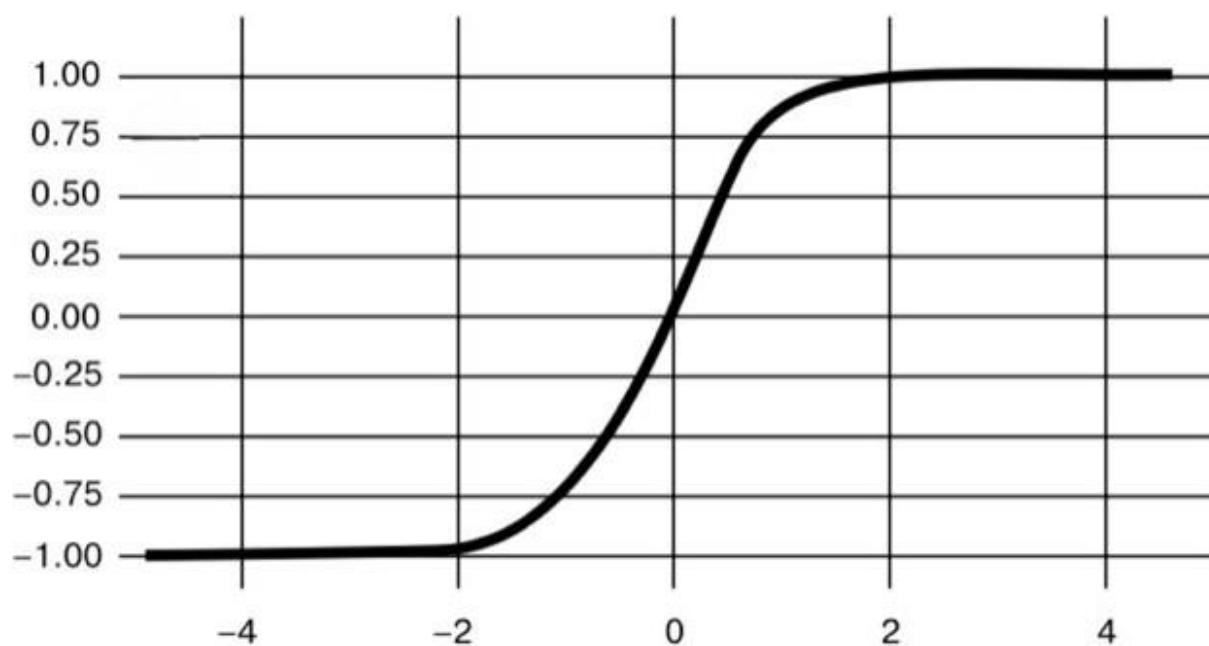


Рисунок 6 – Гиперболический тангенс

ReLU (Rectified Linear Unit) — одна из самых распространенных в глубоком обучении. Данная функция обнуляет отрицательные значения:

$$f(x) = \max(0, x),$$

где x – входной сигнал.

График этой функции представлен на рисунке 7.

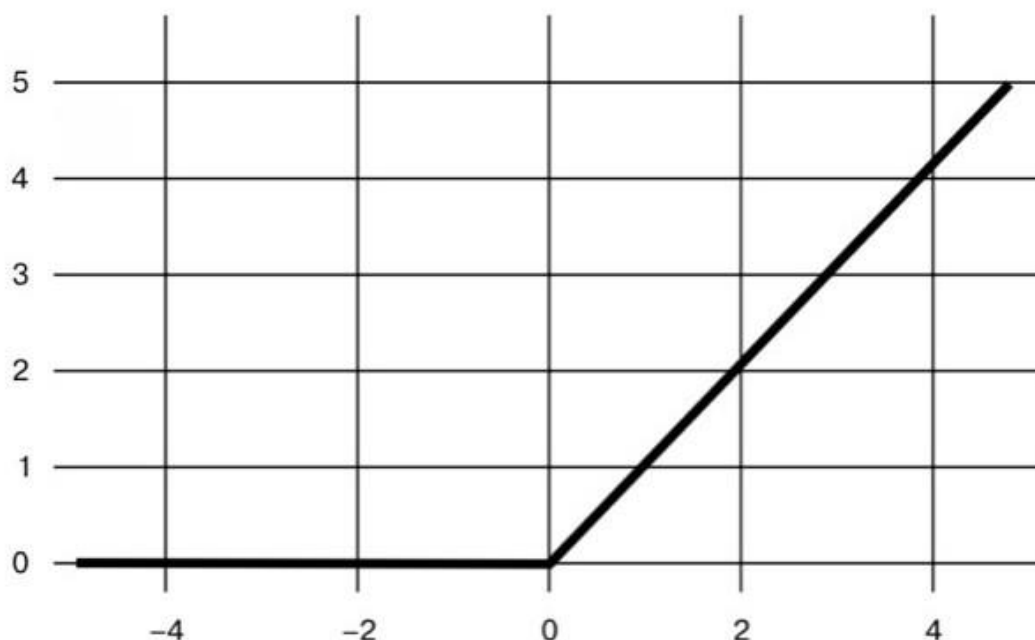


Рисунок 7 – график ReLU

1.3.3 Функции потерь

В зависимости от типа задачи выбирается наиболее подходящая функция потерь:

- среднеквадратичная погрешность,
- дискретная перекрестная энтропия.

Среднеквадратичная погрешность (*MSE* - mean squared error) часто используется для решения задач регрессии. *MSE* можно рассчитать по формуле:

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

где y – вектор значений переменной;

y^* – вектор прогнозируемых значений переменной.

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

Рассмотрим сигмоидную функцию активации. В обоих случаях значение производной 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.

Таким образом, значение производной функции по аргументу не содержит в себе домножения на элементы (например, на или -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.

1.4 Оптимизация нейронной сети

1.4.1 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 и сделать шаг в сторону, которая противоположна направлению градиента, так как вектор градиента в каждой точке направлен в сторону наиболее быстрого возрастания функции [12].

$$\nabla L = \left[\frac{\partial L}{\partial w_0} \quad \frac{\partial L}{\partial w_1} \quad \frac{\partial L}{\partial w_2} \quad \dots \quad \frac{\partial L}{\partial w_n} \right]^T, \quad (2)$$

где $w_0 \dots w_n$ — значения весов;

T — номер шага.

Размер данного шага можно регулировать с помощью некоторого параметра, который называется скоростью обучения, и в зависимости от его выбора существуют различные модификации метода. Таким образом, нахождение значений весов на шаге под номером T можно выразить из формулы (2).

От выбора коэффициента зависит, будет ли сходиться метод или нет. На рисунке 8 представлены варианты траектории движения при градиентном спуске с различными скоростями обучения.

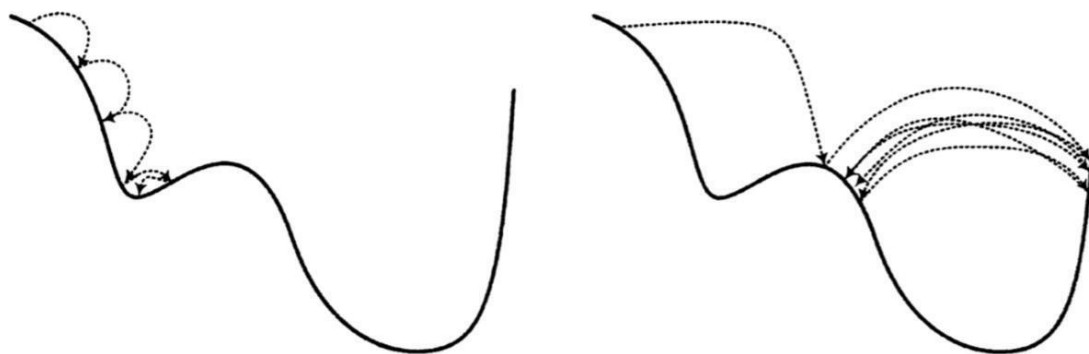


Рисунок 8 - Градиентный спуск для разных скоростей обучения

Недостаток стандартного подхода градиентного спуска заключается в следующем: для определения градиентного спуска необходимо вычислить значение функции потерь и масштабировать его по всем элементам обучающего набора данных. Учитывая большой размер набора данных при создании нейронной сети, предложенный подход становится неоптимальным, так как требует значительного времени для выполнения одного шага минимизации. Для решения этой проблемы часто используется подход пакетного градиентного спуска: Функция вычисляется только на нескольких примерах за раз, то есть набор данных делится на партии, и для следующего шага выбираются разные примеры, что значительно ускоряет вычисление следующего шага [13]. Строгих ограничений на выбор размера партии нет, но рекомендуется придерживаться следующего правила: Размер должен быть больше 1 и меньше размера всего набора данных. Еще один часто используемый метод ускорения вычислений - стохастический градиентный спуск. Идея заключается в том, чтобы использовать только один элемент набора данных для вычисления весового вектора за один шаг.

1.4.2 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 на рисунке 9. 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.

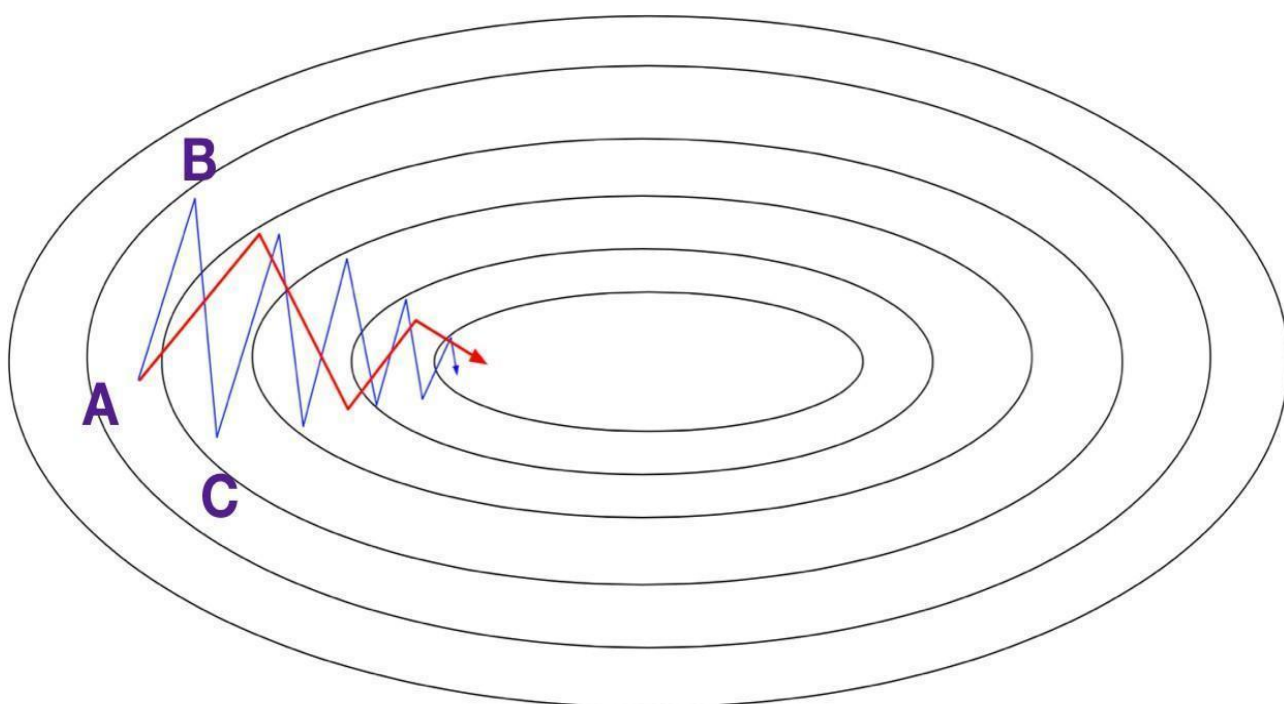


Рисунок 9 - Траектории различных методов градиентного спуска для функции потерь с «оврагами»

Метод, который описывается далее, представляет собой физическую аналогию катящегося шара. Он называется методом Momentum (или импульса). Этот метод позволяет ускорить сходимость градиентного спуска, особенно в случаях, когда функция потерь имеет вытянутые линии уровня по одному измерению.

Идея метода Momentum заключается в том, чтобы добавить к классическому градиентному спуску инерцию. То есть, вместо того чтобы резко менять

направление движения на каждой итерации, мы будем накапливать скорость движения, учитывая предыдущие изменения [14].

Возвращаясь к аналогии с катящимся шаром, можно представить себе, что шар не только скатывается вниз по склону, но и сохраняет свою инерцию от предыдущего движения. Это позволяет ему преодолевать небольшие препятствия и изгибы на пути к минимуму функции.

Таким образом, метод Momentum может помочь ускорить сходимость градиентного спуска в случаях, когда обычный градиентный спуск работает медленно из-за особенностей функции потерь.

1.4.3 Алгоритм Adam

Удобство использования алгоритма Adam заключается в том, что зачастую не нужно подбирать параметры для его использования: оптимальные коэффициенты для этого алгоритма $\alpha=3 \cdot 10^{-4}$, $\beta_1=0.9$, $\beta_2=0.999$. 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:

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

где $\text{EMA}_{\beta}(f)^t$ - экспоненциальное скользящее среднее в момент времени.

1.4.4 Методы борьбы с переобучением

Изначально цель глубокого обучения - минимизировать функцию потерь, тем самым повысив точность предсказаний модели. В какой-то момент, проанализировав все входные данные, нейронная сеть научится правильно классифицировать все исходные объекты или предсказывать результат на основе обучающего набора. Однако возникает вопрос: как сеть будет работать с

объектами, с которыми она никогда не сталкивалась? Часто модель начинает приспосабливаться к обучающему набору, не улавливая характерных закономерностей.

Например, рассмотрим задачу регрессии. На рисунке 10 зеленым цветом показана истинная зависимость, синими точками — объекты обучающей выборки. На графике 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 рисунке 11 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.

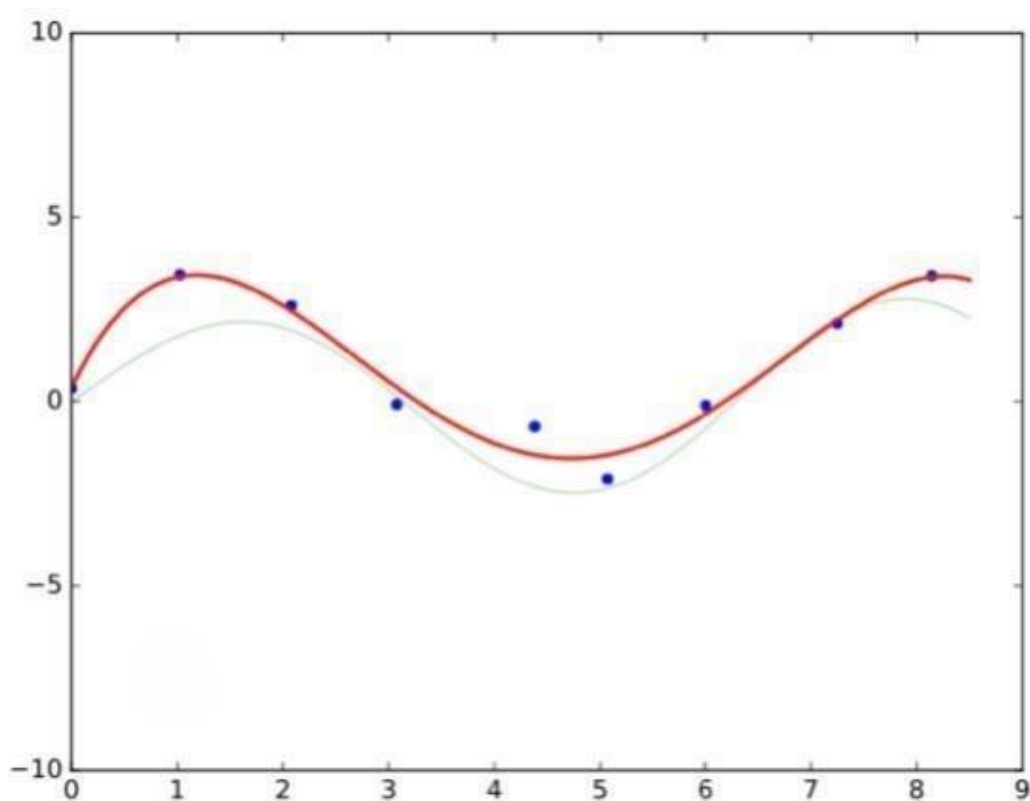


Рисунок 10 – График обучения

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 тестируется результат обучения на каждой эпохе (каждом проходе всех обучающих данных).

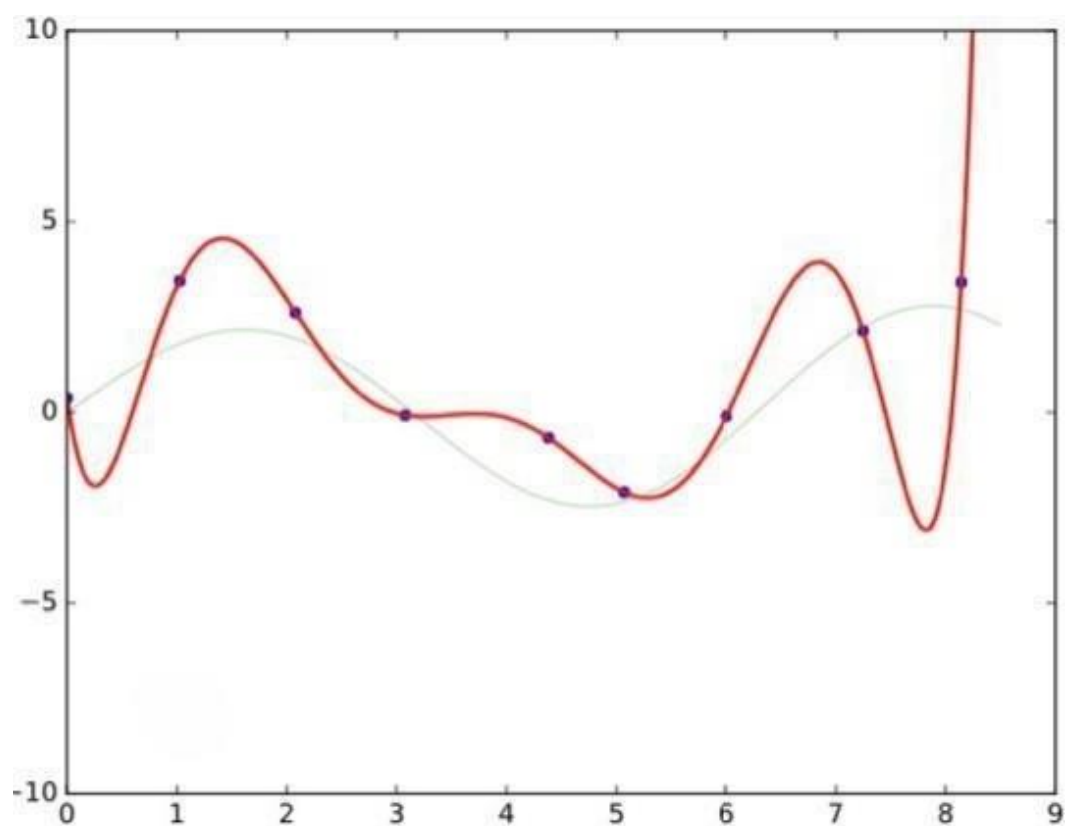


Рисунок 11 – Эффект переобучения

1.4.5 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 [16] (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 на рисунке 12.

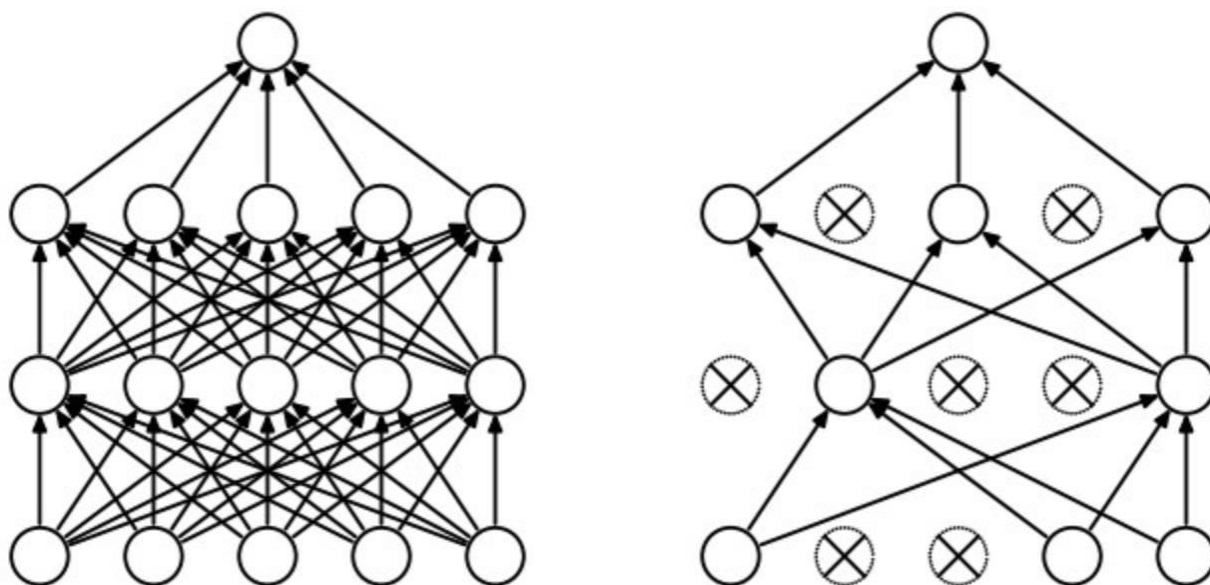


Рисунок 12 - Визуальное представление метода Dropout

1.5 Рекуррентная нейронная сеть

Нейронным сетям часто приходится получать данные разного объема, особенно при обработке естественного языка.

Концепция рекуррентных нейронных сетей (РНС) предполагает последовательное использование информации. Традиционные нейронные сети предполагают, что все входы и выходы независимы, но такой подход не подходит для многих задач. В отличие от нейронных сетей с прямой передачей, RNN позволяют нейронам обмениваться информацией друг с другом. В сети реализована «память», что кардинально меняет ее работу и позволяет анализировать любую последовательность данных, где важен порядок значений. Это особенно полезно при обработке естественного языка.

При визуализации элементы рекуррентной нейронной сети изображаются в виде обычных нейронов с дополнительной периодической стрелкой, указывающей на то, что помимо входного сигнала нейрон также использует свое дополнительное скрытое состояние.

Схема простейшей RNN с одним входным нейроном, одним скрытым нейроном и одним выходным нейроном представлена на рисунке 13.

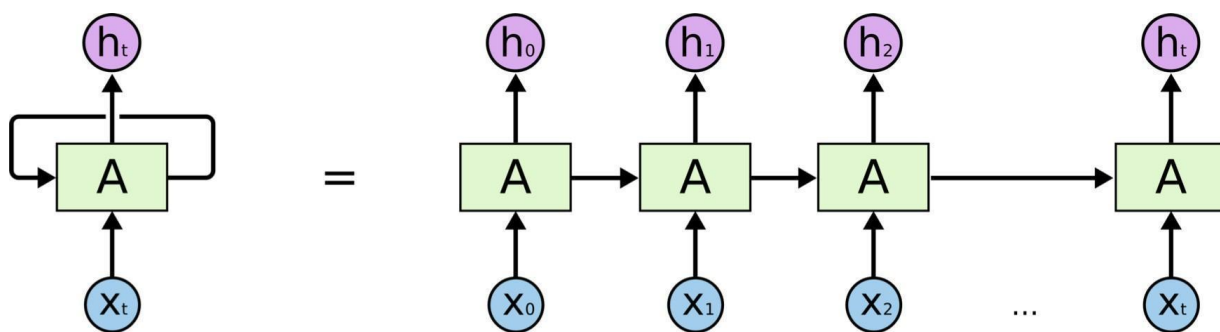


Рисунок 13 - Простейшая рекуррентная нейронная сеть в развертке

Проблема стандартной рекуррентной нейронной сети заключается в ее неспособности работать с долгосрочными зависимостями. Например, если задача состоит в том, чтобы предсказать последнее (четвертое) слово предложения «Я должен купить хлеб», необходимо учесть контекст, состоящий из слов «я», «должен», «купить» и «хлеб» (порядок здесь важен). RNN может справиться с этой задачей, поскольку расстояние между первым и последним словами невелико на рисунке 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. Для решения данной проблемы были разработаны сети LSTM [17].

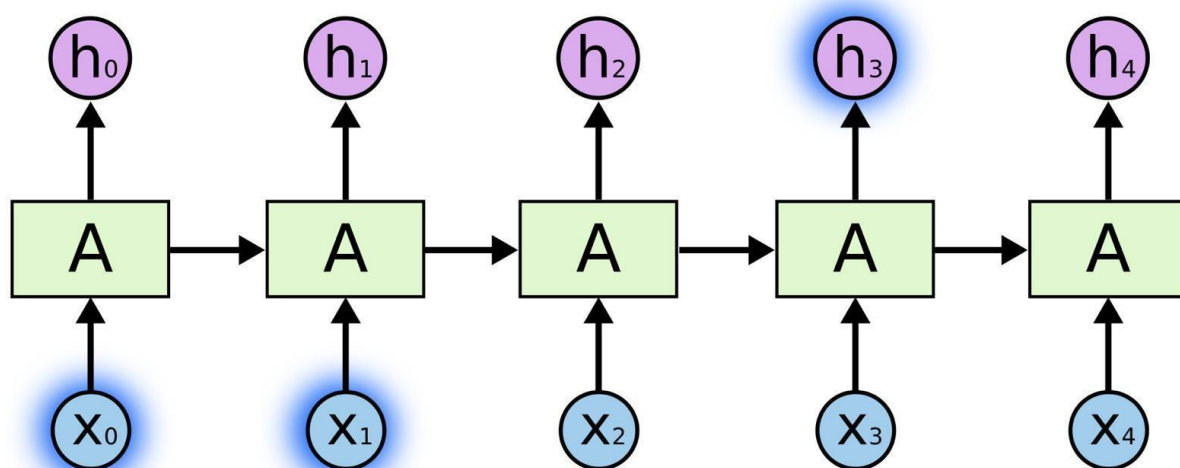


Рисунок 14 - RNN на примере задачи предсказания слова

1.5.1 LSTM

LSTM (Long short-term memory) — одна из архитектур рекуррентной нейронной сети, способная к обучению долговременным зависимостям [18].

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 на рисунке 15, в то время как в LSTM каждая ячейка имеет более сложную структуру на рисунке 16.

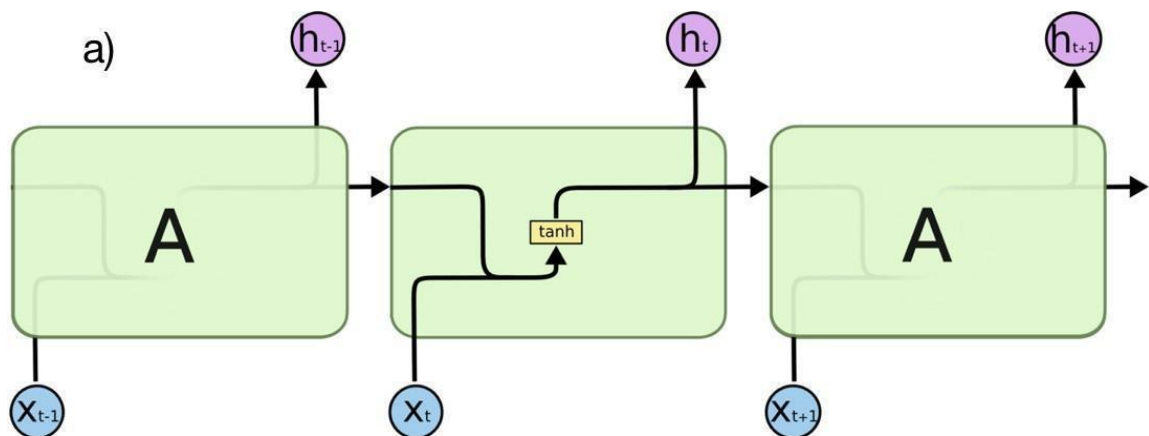


Рисунок 15 - Визуальное представление ячейки RNN

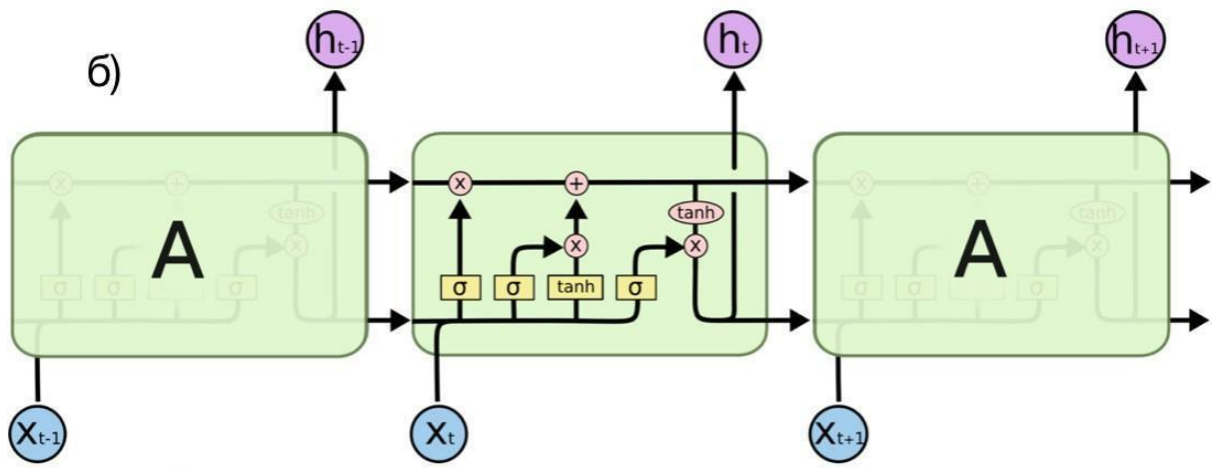


Рисунок 16 - Визуальное представление ячейки LSTM

Первый шаг LSTM — определить, какую информацию нет потребности запоминать (можно удалить из ячейки). Эту задачу решает первый слой (forget gate layer) на рисунке 17.

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

где h_{t-1} – значение предыдущего выхода;

x_t – значение текущего входа.

Следующий этап состоит в том, чтобы решить, какая новая информация будет храниться в состоянии ячейки на рисунке 18.

На третьем шаге происходит замена старого значения C_{t-1} на новое C_t на рисунке 19.

На последнем этапе вычисляется значение выходного нейрона на рисунке 20.

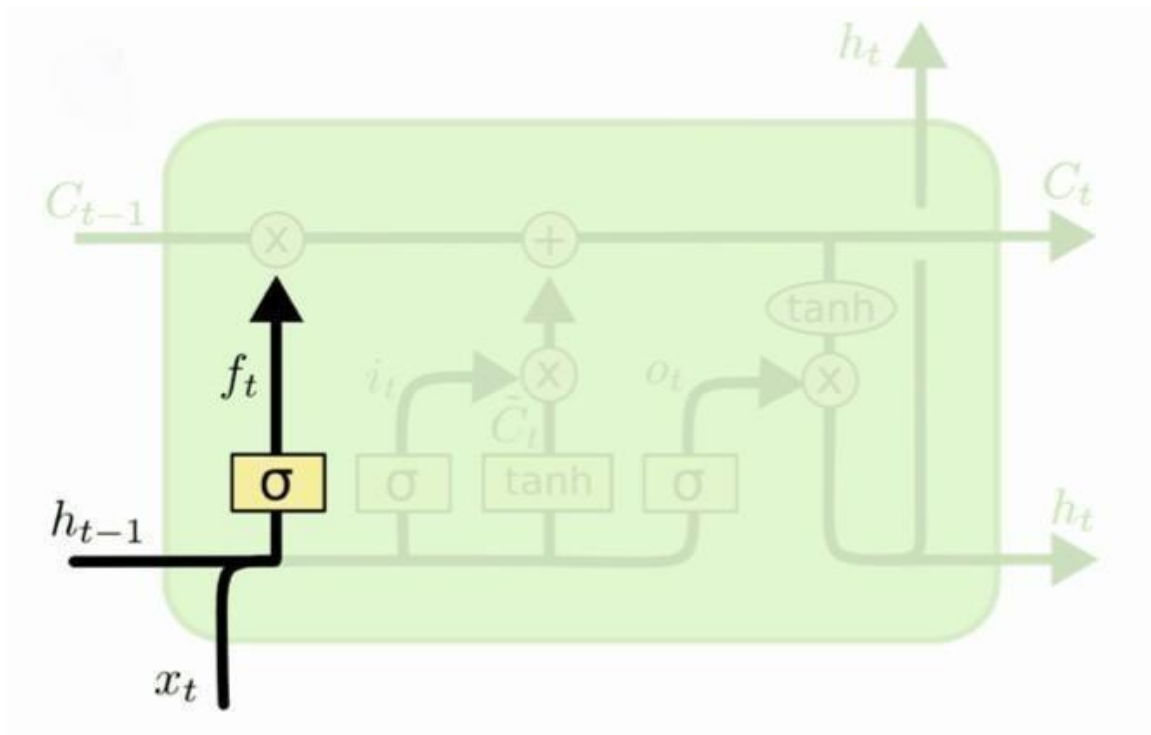


Рисунок 17 - Пошаговое представление работы LSTM. Первый шаг

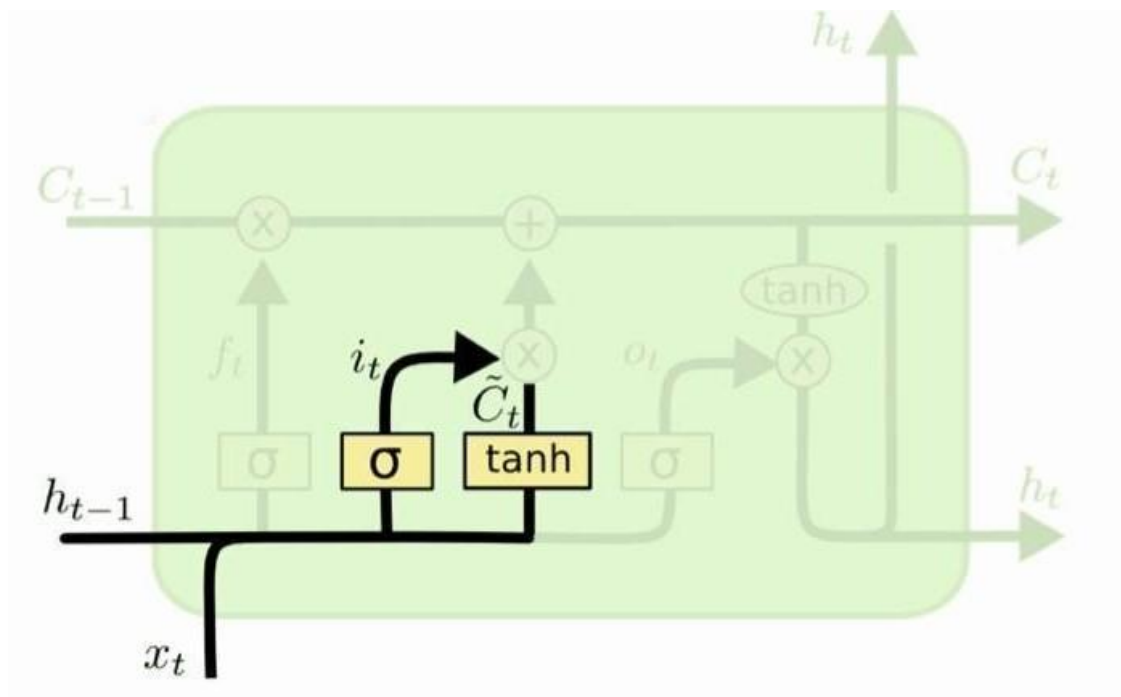


Рисунок 18 - Пошаговое представление работы LSTM. Второй шаг

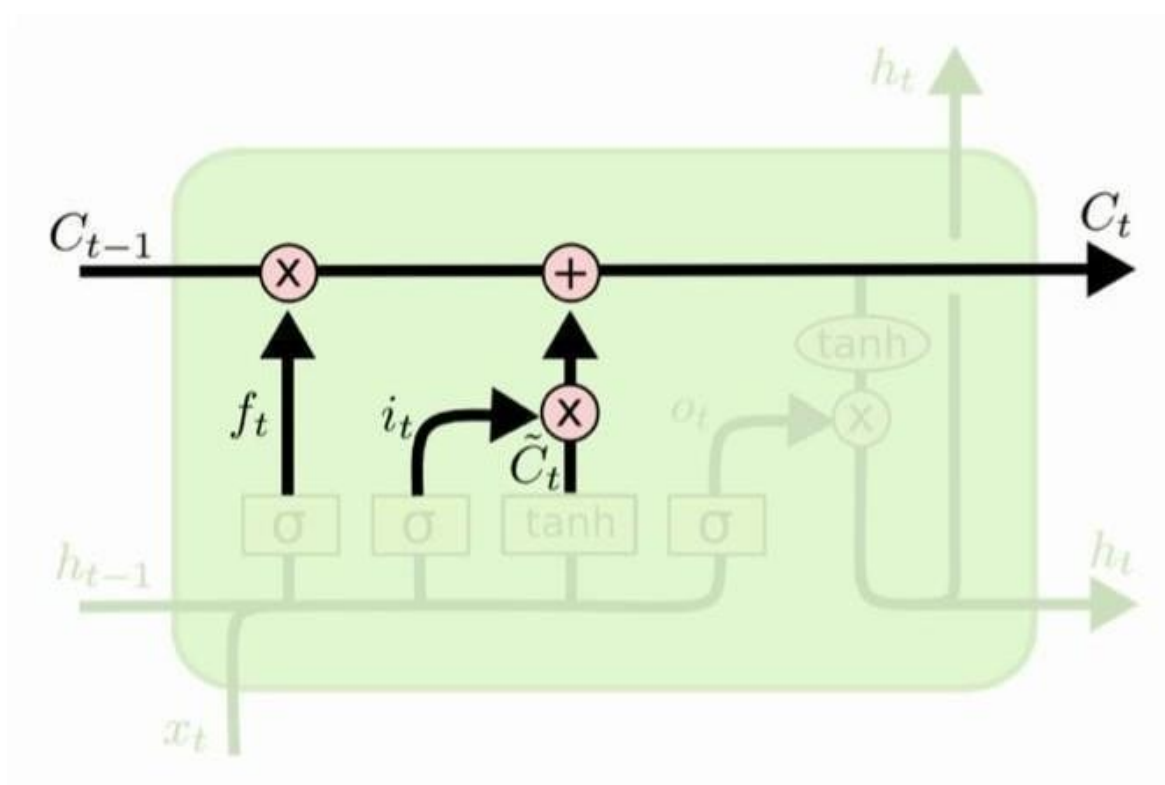


Рисунок 19 - Пошаговое представление работы LSTM. Третий шаг

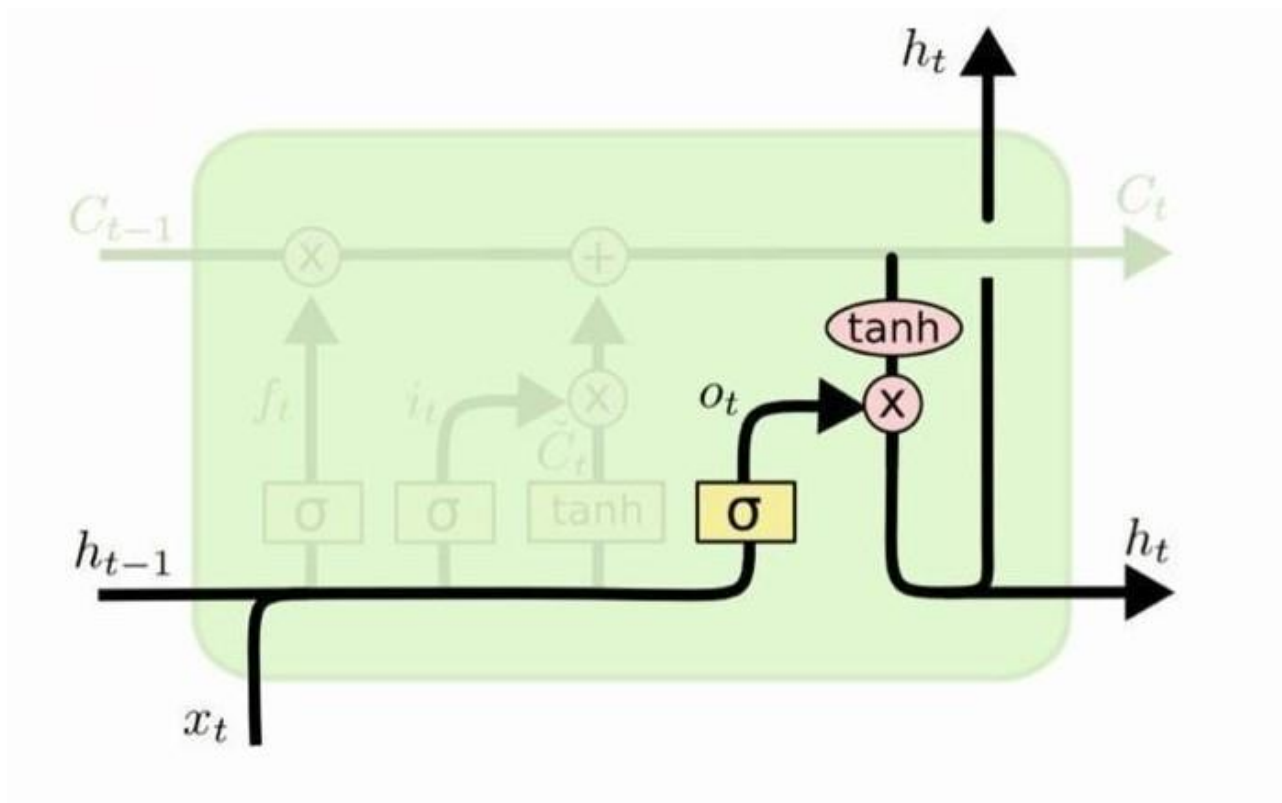


Рисунок 20 - Пошаговое представление работы LSTM. Четвертый шаг

1.6 Векторизация слов

Векторизация слов – это преобразование слов с помощью определенной модели в числовые векторы. Это необходимо для упрощения и увеличения скорости обработки нечисловых данных моделями искусственного интеллекта. Кроме того, векторные представления слов зачастую занимают меньший объем памяти, чем слова в традиционном виде. Векторы, полученные из слов, называются эмбедингами.

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 .

Также имеет смысл строить только такие вектора, которые при похожих словах по определенной метрике (например, косинусной или евклидовой) имеют похожее направление. По формуле расчета косинус угла между векторами можно определить, насколько векторы слов сонаправлены (3).

$$\cos(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{|\vec{x}| \cdot |\vec{y}|}, \quad (3)$$

где x и y – векторные представления слов.

Для преобразования слов в эмбединги существуют специальные модели, которые работают по разным алгоритмам. Поэтому для решения определенной задачи нужно подбирать более подходящую модель векторизации.

1.6.1 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. Для получения более качественного результата в качестве входных данных рекомендуется использовать большой текстовый корпус, так как в основе Word2Vec лежит простая нейронная сеть.

Модель работает на основе подсчета функции, которая обозначает вероятность встретить слово w_i в контексте w_j . Определение контекста пораказно на рисунке 21.

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.

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

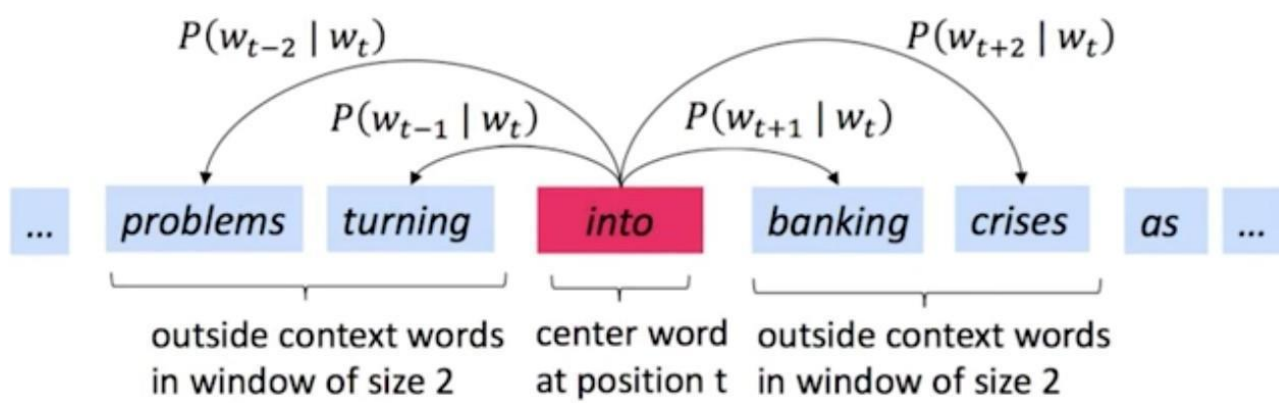


Рисунок 21 - Контекст для слова *into*

2 Описание параметров нейронной сети

2.1 Выбор архитектуры и гиперпараметров сети для задачи определения тональности текста

Для задач анализа настроения, как и для многих других задач обработки естественного языка, порядок слов в предложении часто имеет большое значение [19]. Например, рассмотрим предложения «Этот человек не глуп, а умен» и «Этот человек не умен, а глуп». Эти два предложения содержат противоречивые смыслы и, соответственно, разные эмоциональные тона: Первое - позитивное, а второе - негативное. При использовании стандартных нейросетевых архитектур, не учитывающих порядок слов, оба предложения обрабатываются одинаково, рассматриваясь как содержащие один и тот же набор слов ({это, этот человек, не глупый, умный}), - такой подход явно неверен и может помешать обучению. Длительная кратковременная память (LSTM) решает эту проблему, сохраняя информацию, полученную в процессе обработки каждого слова, которая затем используется при обработке последующих слов. Поэтому было решено использовать именно этот тип нейронной сети.

Сети LSTM постоянно показывают одни из лучших результатов в задачах категоризации текстов по эмоциональным тонам.

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

- входной слой,
- embedding слой, который преобразует каждый токен в векторное представление,
- lstm-слой,
- слой dropout ($p = 0.4$),
- линейный слой,
- sigmoid слой,
- выходной слой.

На рисунке 22 можно увидеть графическое представление архитектуры, используемой сети.

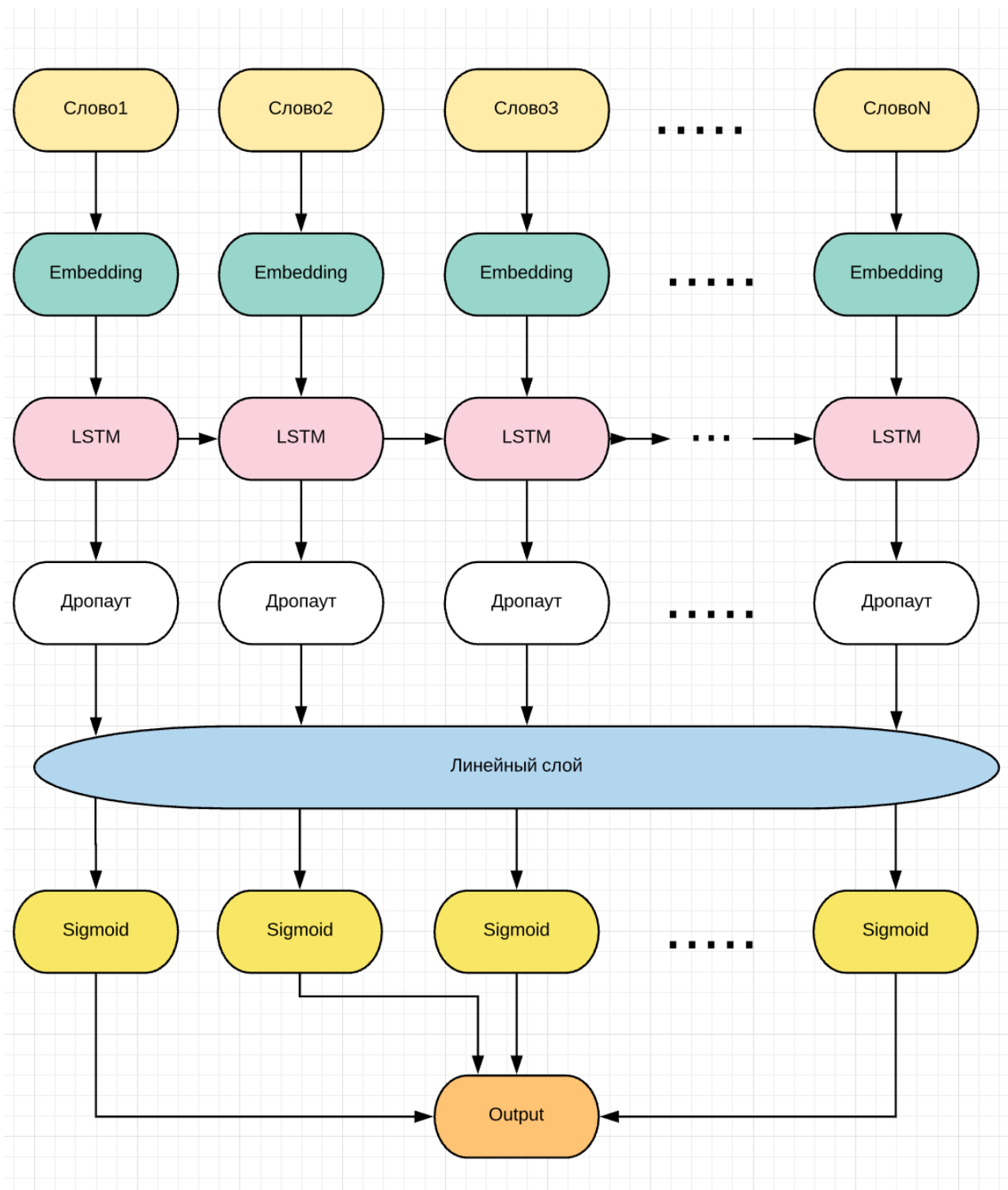


Рисунок 22 - Архитектура нейронной сети

Для обучения модели были выбраны следующие гиперпараметры:

- функция потерь — бинарная перекрестная энтропия (bceloss),
- метод оптимизации — алгоритм оптимизации типа градиентного спуска adam (adaptive moment estimation) с шагом 0.001,

– метрика качества — доля правильных ответов на тестовой выборке (ассураcy).

2.2 Описание обучающей выборки

Давно прошли те времена, когда люди писали друг другу письма на бумаге, тщательно изучая каждое предложение и переписывая его, если что-то казалось неправильным. Бешеный темп современной жизни заставляет нас обмениваться информацией здесь и сейчас, в виде коротких сообщений, не обращая внимания на орфографические и пунктуационные ошибки. Часто эти ошибки допускаются не потому, что человек неграмотен, а просто потому, что нет времени на их исправление, ведь получатель все равно поймет сообщение. Именно поэтому эмодзи (смайлики) так популярны, ведь они выражают целые эмоции одним символом, экономя время на написание множества слов и предложений. Кроме того, сейчас модно включать в сообщения сленговые слова и идиомы, которые состоят из одного или нескольких слов и имеют определенный смысл или передают юмор, сарказм и т. д. Поэтому задача изучения возможностей программы была поставлена именно на примерах русскоязычных сообщений в социальных сетях, так как они наглядно иллюстрируют язык современного общения (короткие предложения, часто с ошибками, эмодзи, сленг и жаргон).

В качестве обучающего датасета был выбран корпус коротких текстов Юлии Рубцовой [20], сформированный на основе русскоязычных сообщений из Twitter (сеть запрещена в России). Он содержит порядка 114 000 положительных и 111 000 отрицательных размеченных сообщений. Как сказано в работе [21], 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`. Небольшие фрагменты датасетов приведены на рисунках 23 и 24.

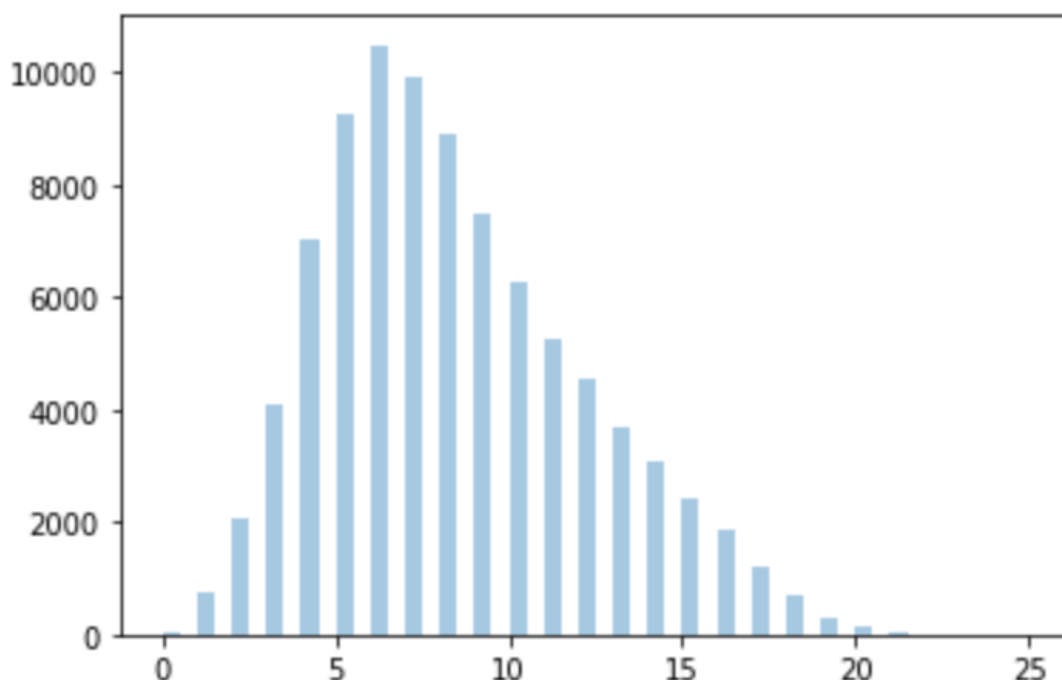


Рисунок 25 – Гистограмма распределения текстов по количеству слов

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 emoji 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[22]. 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

Обучение нейронной сети требует значительных вычислительных мощностей, и для этих целей часто используются графические процессоры (GPU). Облачная технология Google Colab [23] предоставляет доступ к тензорным процессорам (TPU), что делает ее популярной среди разработчиков машинного обучения. Среда Jupyter Notebook, используемая в Colab, является широко распространенным инструментом среди аналитиков данных. Одним из преимуществ этой среды является отсутствие необходимости устанавливать библиотеки машинного обучения и глубокого обучения, так как они уже установлены на сервере.

Приложение А содержит реализацию класса RNN, который описывает архитектуру нейронной сети. Приложение В содержит метод `Train_model`, отвечающий за обучение модели, а приложение С - функцию предсказания, позволяющую прогнозировать результат анализа настроения текста.

3.2 Веб-приложение

С помощью микрофреймворка Flask было реализовано веб-приложение для визуализации работы обученной нейронной сети для анализа настроений. Flask опирается на несколько внешних библиотек, таких как Werkzeug и Jinja2. Werkzeug - это набор инструментов для WSGI - стандартного интерфейса Python между веб-приложениями и различными серверами, предназначенный для разработки и развертывания. Jinja2 заботится об отображении шаблонов.

Для разработки Flask-приложения необходимо было использовать виртуальное окружение `virtualenv`[24]. Виртуальное окружение - это полная копия интерпретатора Python. Если пакеты устанавливаются в виртуальное окружение, это влияет только на его копию, но не на общесистемный интерпретатор Python. Поэтому лучшим решением будет использование нового виртуального окружения для каждого приложения. Это решает следующую проблему: Библиотеки часто нарушают обратную совместимость, а многочисленные проекты 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 EnvDir  
cd EnvDir  
virtualenv venv  
venv/bin/activate  
  
pip install Flask
```

4.2 Инструкция по сборке проекта

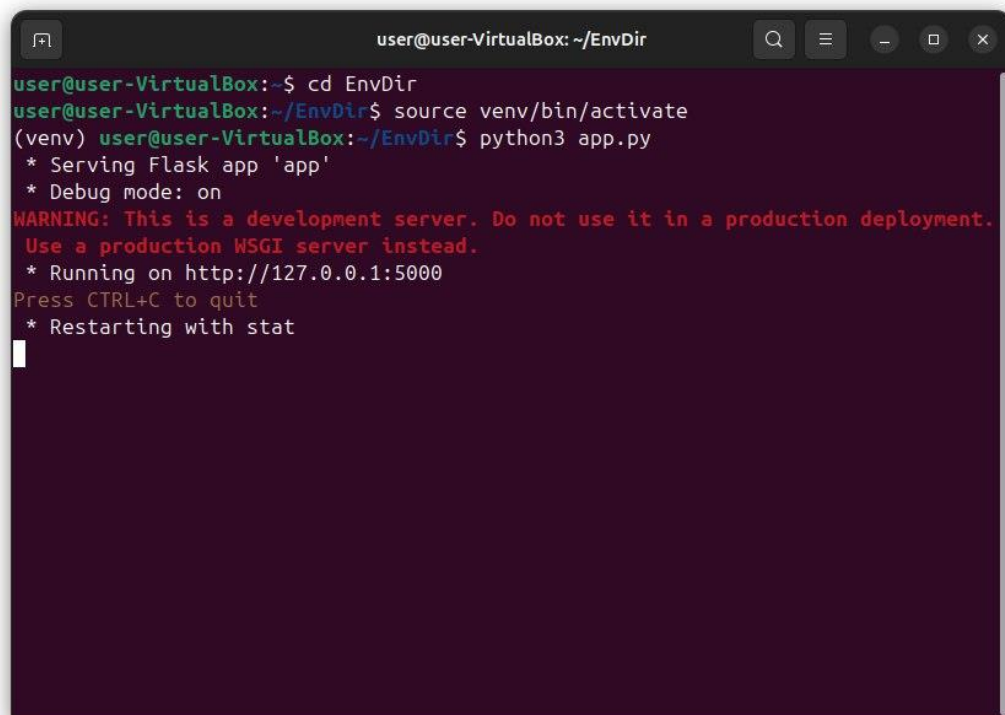
Обучение нейронной сети происходит в файле *train_model.py*.

```
python3 train_model.py
```

Чтобы собрать и запустить проект из командной строки, нужно ввести в терминал следующую последовательность команд (для операционной системы Linux Ubuntu):

```
cd EnvDir  
source venv/bin/activate  
python3 app.py
```

После появления в консоли сообщения «Running on http://127.0.0.1:5000» сервер запустится по указанному адресу (рисунок 26).



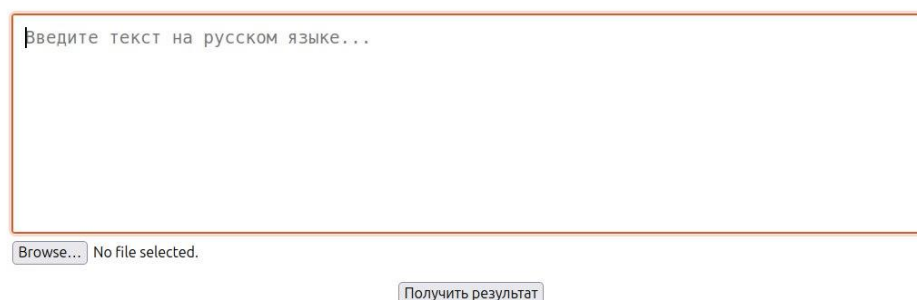
```
user@user-VirtualBox: ~/EnvDir
user@user-VirtualBox:~$ cd EnvDir
user@user-VirtualBox:~/EnvDir$ source venv/bin/activate
(venv) user@user-VirtualBox:~/EnvDir$ python3 app.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
```

Рисунок 26 – Окно терминала с запуском сервера

Для получения класса эмоциональной окраски текст необходимо ввести в текстовое поле на веб-странице, интерфейс которой изображен на рисунке 27.



Анализ эмоциональной окраски текста



Введите текст на русском языке...

Browse... No file selected.

Получить результат

Рисунок 27 – Окно браузера с интерфейсом

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 рисунке 28 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

Рисунок 28 - 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 [25]. 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 участвовавших ранее в обучении. Она показала следующий результат: доля правильных ответов в тестовой выборке равна 74% на рисунке 29.

Test loss: 0.708
Test accuracy: 0.740

Рисунок 29 – Результат тестирования

Таблица 1 - Предсказанные моделью окраски для некоторых текстов

Текст	Оценка	Вероятность	Оценка результата
Мне не нравится то, что ты говоришь	Негативная	71 %	Верно
Это невероятное приключение подарило столько незабываемых впечатлений, хотя в начале выглядело как ужасная идея!	Позитивная	99 %	Верно

Продолжение таблицы 1

Текст	Оценка	Вероятность	Правильность результата
Если еще раз ты посмотришь в сторону выхода, то клянусь, вылетишь отсюда как пуля!	Позитивная	97 %	Неверно
Я тебя не люблю	Негативная	98 %	Верно
Ну и отлично, чё) Все красавчики	Позитивная	66 %	Верно
Как ты могла со мной так мерзко поступить? Я ухожу!	Негативная	69 %	Верно
В этом квартале все работали довольно неплохо, поэтому руководство выплатит всем премию	Позитивная	97%	Верно

Продолжение таблицы 1

Текст	Оценка	Вероятность	Правильность результата
Политика компании в последние годы кажется сомнительной	Позитивная	97%	Неверно
Было не очень приятно, когда таксист не довез до пункта назначения	Негативная	60%	Верно
Мой заказ пришел с опозданием, прошу вернуть деньги	Позитивная	57%	Неверно
Отпуск был просто нереально крутым	Позитивная	88%	Верно
Я недолюбливаю некоторых людей в своем окружении.	Негативная	69%	Верно

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 «Получить результат» на рисунке 30.

Анализ эмоциональной окраски текста

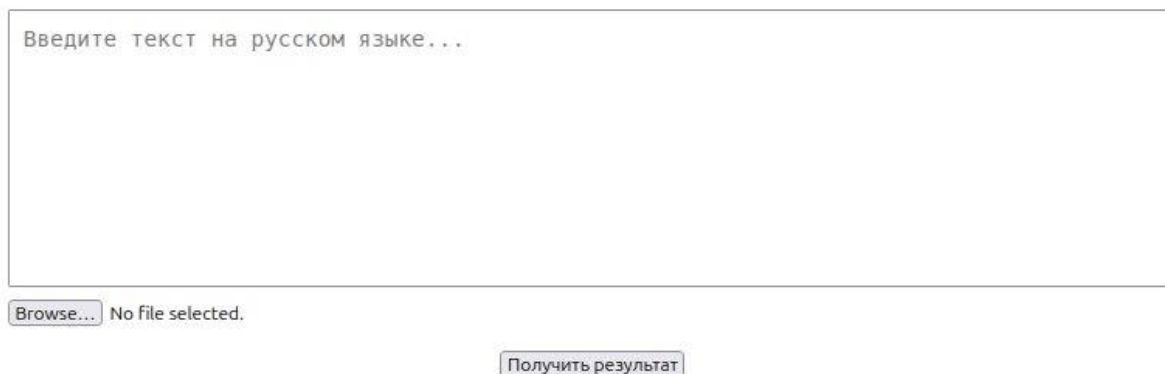
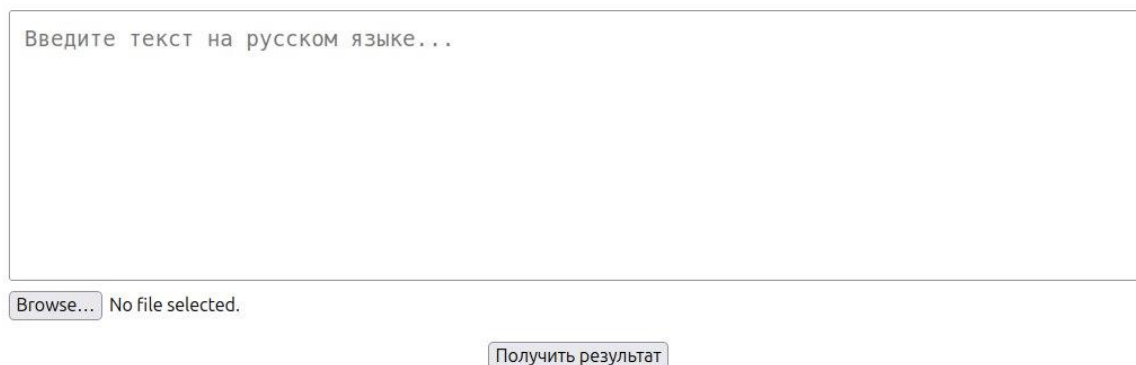


Рисунок 30 - Главная страница до ввода текста

На рисунке 31 представлен результат выполнения после ввода текста и нажатия кнопки.

Анализ эмоциональной окраски текста



Введенный текст: Какой сегодня хороший день выдался!

Эмоциональная окраска: Позитивное сообщение

Вероятность: 88 %

Рисунок 31 - Результат оценки введенного пользователем текста

Помимо ручного ввода текста можно загружать текст из файла. Для этого необходимо нажать кнопку «Browse» и выбрать файл из файловой системы. Окно выбора файла показано на рисунке 32.

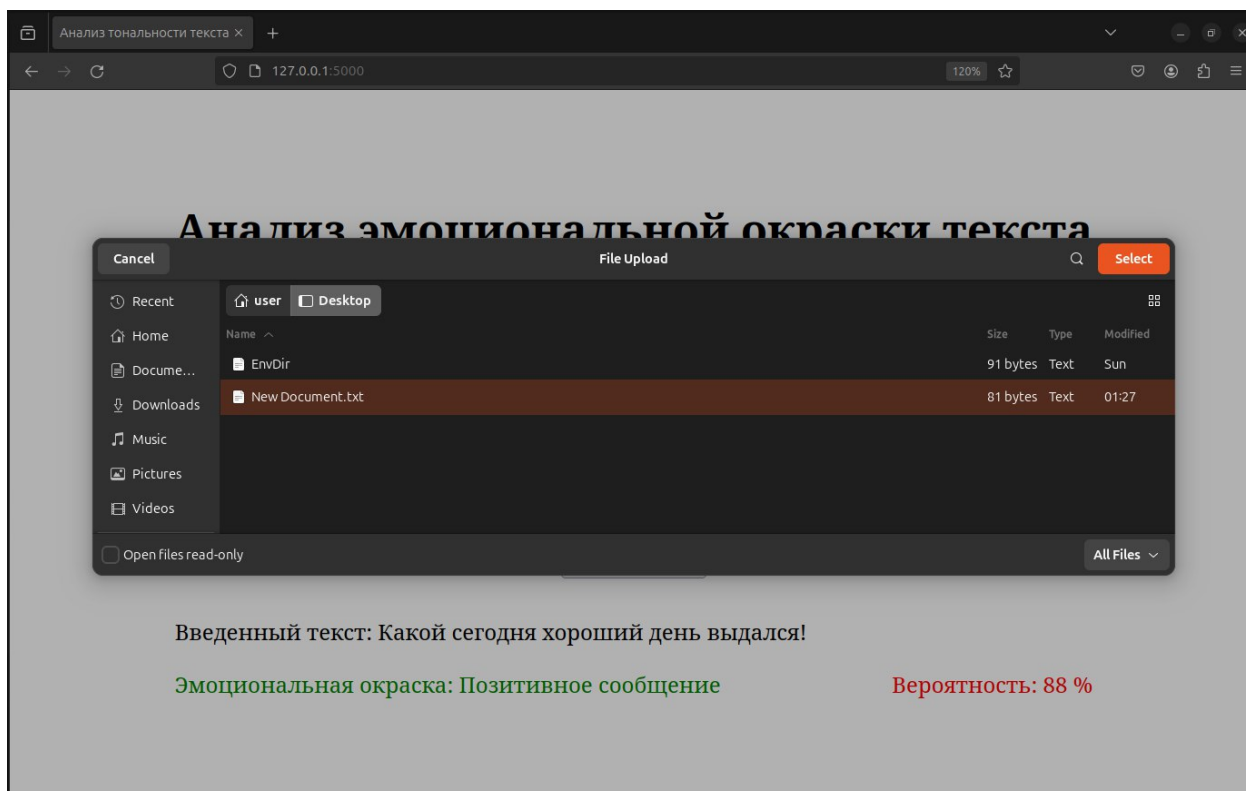


Рисунок 32 – Окно выбора файла в ОС Ubuntu

После выбора содержимое файла отобразится в поле ввода текста. Затем нужно будет нажать на кнопку «Получить результат». Загруженный текст из файла показан на рисунке 33, результат обработки – на рисунке 34.

Анализ эмоциональной окраски текста

Моя жена сильно достала меня, мы расстаемся.

Browse...
New Document.txt

Получить результат

Введенный текст: Какой сегодня хороший день выдался!

Эмоциональная окраска: Позитивное сообщение

Вероятность: 88 %

Рисунок 33 – Загрузка текста из файла в поле ввода

Анализ эмоциональной окраски текста

Введите текст на русском языке...

Browse...

No file selected.

Получить результат

Введенный текст: Моя жена сильно достала меня, мы расстаемся.

Эмоциональная окраска: **Негативное сообщение**

Вероятность: 78 %

Рисунок 34 – Работа с текстом, загруженным из файла

На рисунках 35 и 36 показана оценка негативного и позитивного сообщения.

Анализ эмоциональной окраски текста

Введите текст на русском языке...

Browse...

No file selected.

Получить результат

Введенный текст: Я тебя ненавижу!

Эмоциональная окраска: **Негативное сообщение**

Вероятность: 73 %

Рисунок 35 – Результат оценки негативного сообщения

Анализ эмоциональной окраски текста

Введите текст на русском языке...

Browse...

No file selected.

Получить результат

Введенный текст: Ты прекрасный

Эмоциональная окраска: Позитивное сообщение

Вероятность: 77 %

Рисунок 36 – Результат оценки позитивного сообщения

ЗАКЛЮЧЕНИЕ

В ходе работы над ВКР было спроектировано приложение, позволяющее определить эмоциональный окрас текста, написанного на русском языке. Для этого была разработана модель глубокого обучения с использованием LSTM-сети.

В процессе разработки системы были исследованы существующие архитектуры нейросетевых алгоритмов и современных инструментов для работы с нейронными сетями, проведен анализ исходных данных и сформирован обучающий набор, обучена подобранная модель нейронной сети и проведен анализ результатов обучения, а также для работы с моделью спроектирован пользовательский интерфейс.

Все задачи, поставленные для выполнения цели, выполнены.

Среди возможных вариантов улучшения модели можно выделить следующее:

- реализация API для сторонних сервисов,
- повышение точности модели за счет более детального подбора гиперпараметров сети,
- дообучение модели на расширенном наборе данных.

Таким образом, задание на ВКР выполнено в полном объеме.

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. Specht, D. A General Regression Neural Network. IEEE Trans. on Neural Networks, Nov. 1991, 2, 6, 568-576.
7. 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.
8. 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.
9. Траск Эндрю Грокаем 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 (дата обращения 21.09.2023).

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 (дата обращения 07.11.2023).

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. — 400с.: ил. — (Серия «Библиотека программиста»). ISBN 978-5-4461-0770-4

17. Recurrent Neural Networks (RNN) and LSTM: Overview and Uses [Электронный ресурс] — Режим доступа: <https://www.turing.com/kb/recurrent-neural-networks-and-lstm> (дата обращения 12.02.2024)

18. Hochreiter S., Schmidhuber J. Long short-term memory // Neural computation 9, 1997. Issue 8. P. 1735 – 1780.

19. Бурукина, И. П. Сервисы для определения тональности речи на основе нейронных сетей / И. П. Бурукина, Ю. В. Мамелина // Новые информационные технологии и системы (НИТиС-2023) : Сборник научных статей по материалам XX Международной научно-технической конференции, посвященной 80-летию юбилею Пензенского государственного университета, Пенза, 16–17 ноября 2023

года. – Пенза: Пензенский государственный университет, 2023. – С. 130-135. – EDN IXRKPK.

20. 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.

21. Ю. В. Рубцова. Построение корпуса текстов для настройки тонового классификатора // Программные продукты и системы, 2015, №1(109), –С. 72-78

22. DeepPavlov's documentation [Электронный ресурс] — Режим доступа: http://docs.deeppavlov.ai/en/master/features/pretrained_vectors.html (дата обращения: 02.06.2020).

23. Google Colab [Электронный ресурс] — Режим доступа: <https://colab.research.google.com> (дата обращения: 15.05.2020).

24. 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).

25. 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

(обязательное)

Исследование методов машинного обучения в задаче обработки текста на
естественном языке

Листинг программы

Файл train_model.py

```
# Импорт необходимых библиотек
import pandas as pd
import numpy as np
import torch.nn as nn
import torch
from torch.utils.data import TensorDataset, DataLoader
from collections import Counter
from Neural_Architecture import LSTM_architecture

### Считывание данных

n = ['id', 'date', 'name', 'text', 'typr', 'rep', 'rtw', 'faw',
'stcount', 'foll', 'frien', 'listcount']
data_positive = pd.read_csv('./dataset/positive.csv', sep=';',
on_bad_lines='skip', names=n, usecols=['text'])
data_negative = pd.read_csv('./dataset/negative.csv', sep=';',
on_bad_lines='skip', names=n, usecols=['text'])

### Формирование сбалансированного датасета

sample_size = 40000
reviews_withoutshuffle =
np.concatenate((data_positive['text'].values[:sample_size],
data_negative['text'].values[:sample_size]), axis=0)
labels_withoutshuffle = np.asarray([1] * sample_size + [0] *
sample_size)

assert len(reviews_withoutshuffle) ==
len(labels_withoutshuffle)
from sklearn.utils import shuffle
reviews, labels = shuffle(reviews_withoutshuffle,
labels_withoutshuffle, random_state=0)

### Токенизация
def tokenize():
    punctuation = '!"#$%&\'()*+,-./:;<=>?[\]^_`{|}~'
    all_reviews = 'separator'.join(reviews)
    all_reviews = all_reviews.lower()
    all_text = ''.join([c for c in all_reviews if c not in
punctuation])
    texts_split = all_text.split('separator')
    all_text = ' '.join(texts_split)
    words = all_text.split()
    return words, texts_split

words, texts_split = tokenize()

new_reviews = []
for review in texts_split:
    review = review.split()
```

```

        new_text = []
        for word in review:
            if (word[0] != '@') & ('http' not in word) &
(~word.isdigit()):
                new_text.append(word)
        new_reviews.append(new_text)

counts = Counter(words)
vocab = sorted(counts, key=counts.get, reverse=True)
vocab_to_int = {word: ii - 1 for ii, word in enumerate(vocab,
1)}

reviews_ints = []
for review in new_reviews:
    reviews_ints.append([vocab_to_int[word] for word in
review])

def add_pads(reviews_ints, seq_length):
    features = np.zeros((len(reviews_ints), seq_length),
dtype=int)
    for i, row in enumerate(reviews_ints):
        if len(row) == 0:
            continue
        features[i, -len(row):] = np.array(row)[:seq_length]
    return features

### Разделение на обучающую, валидационную и тестовую выборки
features = add_pads(reviews_ints, seq_length=30)
split_frac = 0.8 # 80% на обучающую выборку

split_idx = int(len(features)*split_frac)
train_x, remaining_x = features[:split_idx],
features[split_idx:]
train_y, remaining_y = labels[:split_idx], labels[split_idx:]
test_idx = int(len(remaining_x)*0.5)
val_x, test_x = remaining_x[:test_idx], remaining_x[test_idx:]
val_y, test_y = remaining_y[:test_idx], remaining_y[test_idx:]

train_data = TensorDataset(torch.from_numpy(train_x),
torch.from_numpy(train_y))
valid_data = TensorDataset(torch.from_numpy(val_x),
torch.from_numpy(val_y))
test_data = TensorDataset(torch.from_numpy(test_x),
torch.from_numpy(test_y))
batch_size = 50
train_loader = DataLoader(train_data, shuffle=True,
batch_size=batch_size)
valid_loader = DataLoader(valid_data, shuffle=True,
batch_size=batch_size)
test_loader = DataLoader(test_data, shuffle=True,
batch_size=batch_size)

### Определение режима: GPU или CPU
train_gpu=torch.cuda.is_available()

```

```

###Выбор гиперпараметров и инициализация сети
vocab_size = len(vocab_to_int)+1
print("vocab_size", vocab_size)
output_size = 1
embedding_dim = 100
hidden_dim = 128
n_layers = 2
model = LSTM_architecture(vocab_size, output_size,
embedding_dim, hidden_dim, n_layers)
criterion = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

### Обучение модели
epochs = 4 #оптимальное количество эпох для того, чтобы модель
достаточно обучилась, но не переобучилась
counter = 0
batch_num = 100
clip = 5
if (train_gpu):
    model.cuda()
num_correct = 0
model.train()
for e in range(epochs):
    h = model.init_hidden_state(batch_size)
    for inputs, labels in train_loader:
        num_correct = 0
        counter += 1
        if(train_gpu):
            inputs, labels = inputs.cuda(), labels.cuda()
        h = tuple([each.data for each in h])
        model.zero_grad()
        output, h = model.forward(inputs, h)
        loss = criterion(output.squeeze(), labels.float())
        loss.backward()
        nn.utils.clip_grad_norm_(model.parameters(), clip)
        optimizer.step()
        if counter % batch_num == 0:
            val_h = model.init_hidden_state(batch_size)
            val_losses = []
            model.eval()
            for inputs, labels in valid_loader:
                val_h = tuple([each.data for each in val_h])
                if(train_gpu):
                    inputs, labels = inputs.cuda(),
labels.cuda()

                    output, val_h = model(inputs, val_h)
                    val_loss = criterion(output.squeeze(),
labels.float())

                    val_losses.append(val_loss.item())

            #accuracy
            pred = torch.round(output.squeeze())

```

```

        correct_tensor =
pred.eq(labels.float().view_as(pred))
        correct = np.squeeze(correct_tensor.numpy()) if
not train_gpu else np.squeeze(correct_tensor.cpu().numpy())
        num_correct += np.sum(correct)
        valid_acc =
num_correct/len(valid_loader.dataset)

        model.train()
        print("Epoch: {} ;".format(e+1),
              "Batch Number: {} ;".format(counter),
              "Train Loss: {:.4f} ;".format(loss.item()),
              "Valid Loss: {:.4f}
;".format(np.mean(val_losses)),
              "Valid Accuracy: {:.4f}".format(valid_acc))

PATH = "model"
torch.save(model.state_dict(), PATH)

```

Файл test_model.py

```

from collections import Counter
import pandas as pd
import numpy as np
from math import ceil
import torch
from sklearn.utils import shuffle
from Neural_Architecture import LSTM_architecture
import argparse

def createArgParser():
    parser = argparse.ArgumentParser()
    parser.add_argument('-i', type=argparse.FileType('r'))
    return parser

def read_data():
    n = ['id', 'date', 'name', 'text', 'typr', 'rep', 'rtw',
'faw', 'stcount', 'foll', 'frien', 'listcount']
    data_positive = pd.read_csv('./dataset/positive.csv',
sep=';', names=n, usecols=['text'])
    data_negative = pd.read_csv('./dataset/negative.csv',
sep=';', names=n, usecols=['text'])

    sample_size = 50000
    texts_withoutshuffle =
np.concatenate((data_positive['text'].values[:sample_size],
data_negative['text'].values[:sample_size]), axis=0)
    labels_withoutshuffle = np.asarray([1] * sample_size + [0]
* sample_size)
    assert len(texts_withoutshuffle) ==
len(labels_withoutshuffle)

```

```

        texts, labels = shuffle(texts_withoutshuffle,
labels_withoutshuffle, random_state=0)

        return texts, labels

texts, labels = read_data()

def tokenize():
    punctuation = '!"#$%&\'()*+,-./:;<=>?[\\]^_`{|}~'
    all_texts = 'separator'.join(texts)
    all_texts = all_texts.lower()
    all_text = ''.join([c for c in all_texts if c not in
punctuation])
    texts_split = all_text.split('separator')
    all_text = ' '.join(texts_split)
    words = all_text.split()
    return words

def get_vocabulary():
    words = tokenize()
    counts = Counter(words)
    vocab = sorted(counts, key=counts.get, reverse=True)
    vocab_to_int = {word: ii for ii, word in enumerate(vocab,
1)}

    return vocab, vocab_to_int

def tokenize_text(test_text):
    punctuation = '!"#$%&\'()*+,-./:;<=>?[\\]^_`{|}~'
    test_text = test_text.lower()
    test_text = ''.join([c for c in test_text if c not in
punctuation])
    test_words = test_text.split()
    new_text = []
    for word in test_words:
        if (word[0] != '@') & ('http' not in word) &
(~word.isdigit()):
            new_text.append(word)
    test_ints = []
    _, vocab_to_int = get_vocabulary()
    mas_to_int = []
    for word in new_text:
        if word in vocab_to_int:
            mas_to_int.append(vocab_to_int[word])
    test_ints.append(mas_to_int)

    return test_ints

def add_pads(texts_ints, seq_length):
    features = np.zeros((len(texts_ints), seq_length),
dtype=int)

    for i, row in enumerate(texts_ints):
        if len(row) == 0:

```

```

        continue
        features[i, -len(row):] = np.array(row)[:seq_length]

    return features

def predict(net, test_text, sequence_length=30):
    net.eval()
    test_ints = tokenize_text(test_text)
    seq_length = sequence_length
    features = add_pads(test_ints, seq_length)
    feature_tensor = torch.from_numpy(features)

    batch_size = feature_tensor.size(0)
    h = net.init_hidden_state(batch_size)
    output, h = net(feature_tensor, h)

    pred = torch.round(output.squeeze())
    pos_prob = output.item()

    if (pred.item() == 1):
        result = "Позитивное сообщение"
    else:
        result = "Негативное сообщение"

    return result, pos_prob

if __name__ == "__main__":
    parser = createArgParser()
    args = parser.parse_args()
    assert args.i != None, "Для запуска тестирования без веб-
приложения приложите текстовый файл с помощью ключа -i"
    file_name = args.i.name
    f = open(file_name, 'r')
    data = f.readlines()
    input_text = ''.join(data)
    f.close()

    vocab_size = 164192
    output_size = 1
    embedding_dim = 100
    hidden_dim = 128
    number_of_layers = 2
    model = LSTM_architecture(vocab_size, output_size,
embedding_dim, hidden_dim, number_of_layers)
    model.load_state_dict(torch.load("model",
map_location=torch.device('cpu')), strict=False)
    seq_length = 30

    type_of_tonal, pos_prob = predict(model, input_text,
seq_length)
    if type_of_tonal == "Негативное сообщение":
        prob = ceil((1 - pos_prob) * 100)
    else:

```

```

        prob = ceil(pos_prob * 100)
        print("Окраска - {}, вероятность =
{}%".format(type_of_tonal, prob))

```

Файл app.py

```

from flask import Flask, render_template, request
import torch
from test_model import predict
from Neural_Architecture import LSTM_architecture
from math import ceil

DEBUG = True
app = Flask(__name__)
app.config.from_object(__name__)
app.config['SECRET_KEY'] = '7d441f27d441f27567d441f2b6176a'

vocab_size = 164192
output_size = 1
embedding_dim = 100
hidden_dim = 128
number_of_layers = 2
model = LSTM_architecture(vocab_size, output_size,
embedding_dim, hidden_dim, number_of_layers)
model.load_state_dict(torch.load("model",
map_location=torch.device('cpu'))))
seq_length = 30

@app.route("/", methods=['GET', 'POST'])
def hello():
    flag = False
    type_of_tonal = ""
    prob = 0
    name = ""
    if request.method == 'POST':
        flag = True
        if request.form["submit_button"]:
            name1 = request.form['text_tonal']
            if len(name1) != 0:
                name = name1
                type_of_tonal, pos_prob = predict(model, name,
seq_length)

                if type_of_tonal == "Негативное сообщение":
                    prob = ceil((1 - pos_prob)*100)
                else:
                    prob = ceil(pos_prob*100)

            return render_template('main.html', flag = flag,
type_of_tonal = type_of_tonal, percent = "{} %".format(prob), text =
name)

if __name__ == "__main__":

```

```
app.run()
```

Файл Neural_Architecture.py

```
import torch.nn as nn

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.fc = 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.fc(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 Б

(обязательное)

Исследование методов машинного обучения в задаче обработки текста на
естественном языке

Презентация

ИССЛЕДОВАНИЕ МЕТОДОВ МАШИННОГО ОБУЧЕНИЯ В ЗАДАЧЕ ОБРАБОТКИ ТЕКСТА НА ЕСТЕСТВЕННОМ ЯЗЫКЕ

Выполнил: Немков А.М.
Руководитель: Бурукина И.П.

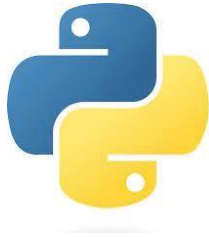
Цель работы

- **Цель работы:** исследование возможности применения алгоритмов машинного обучения для решения задачи классификации текстов на естественном языке .

Для достижения цели необходимо решить следующие задачи:

- Исследовать существующие архитектуры нейросетевых алгоритмов и современные инструменты для работы с нейронными сетями, выбрать наиболее подходящий.
- Провести анализ исходных данных, на их основе сформировать обучающий набор.
- Обучить подобранную модель нейронной сети и провести анализ результатов обучения.
- Спроектировать программную систему для функционирования нейронной сети.

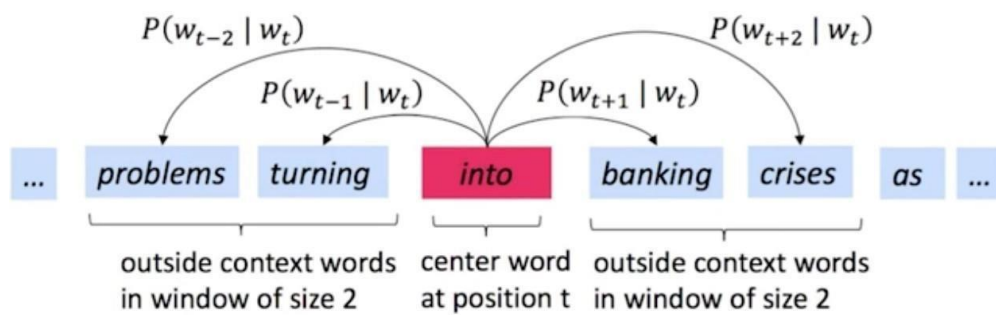
Выбор средств разработки



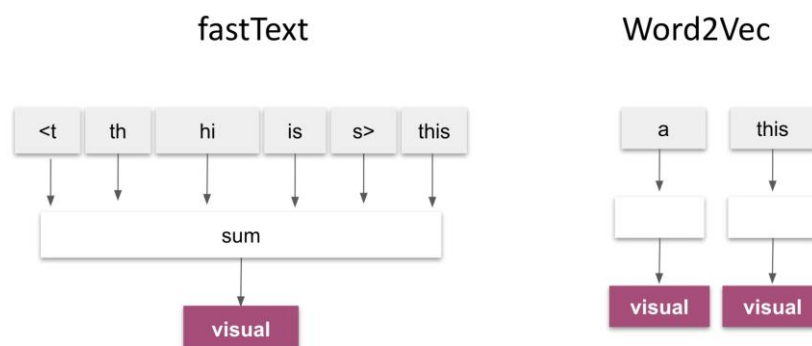
*fast*Text



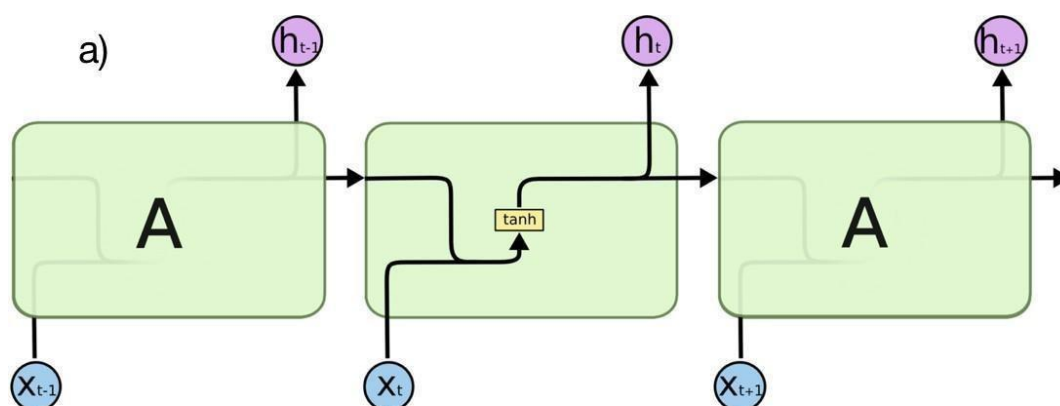
Поиск контекста слова



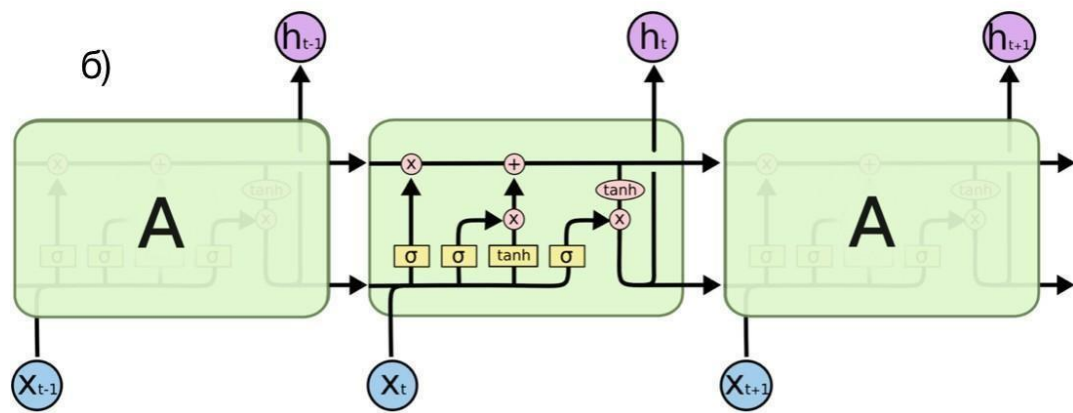
Сравнение Word2Vec и FastText



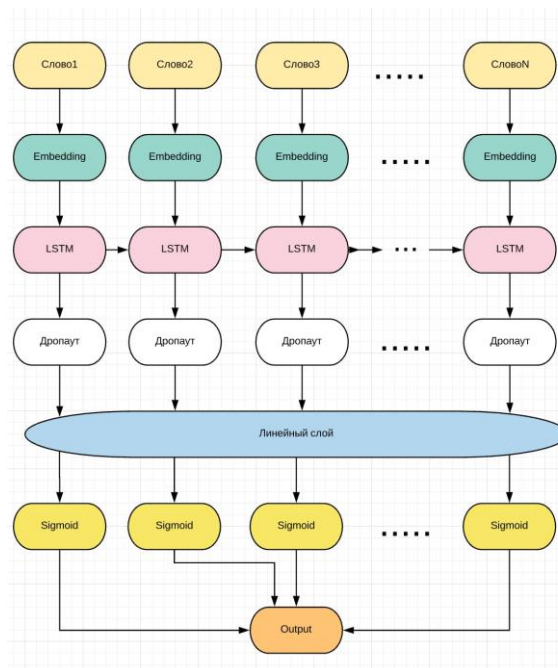
RNN



LSTM



Архитектура сети LSTM



Интерфейс приложения

Анализ эмоциональной окраски текста

Введите текст на русском языке...

Browse...

No file selected.

Получить результат

Результат обработки сообщения

Анализ эмоциональной окраски текста

Введите текст на русском языке...

Browse...

No file selected.

Получить результат

Введенный текст: Какой сегодня хороший день выдался!

Эмоциональная окраска: Позитивное сообщение

Вероятность: 88 %

Результат обработки сообщения

Анализ эмоциональной окраски текста

Введите текст на русском языке. ...

Browse...

No file selected.

Получить результат

Введенный текст: Я тебя ненавижу!

Эмоциональная окраска: Негативное сообщение

Вероятность: 73 %

Заключение

- В ходе работы над ВКР было спроектировано приложение, позволяющее определить эмоциональный окрас текста, написанного на русском языке. Для этого была разработана модель глубокого обучения с использованием LSTM -сети.
- Возможные варианты улучшения:
 - Реализация API для сторонних сервисов
 - Повышение точности модели за счет более детального подбора гиперпараметров сети
 - Дообучение модели на расширенном наборе данных

Спасибо за внимание