МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

Допустить к защите

Заведующий кафедрой

д-р техн. наук, проф.

\_\_\_\_\_\_\_\_\_\_\_\_ Ю.М. Вишняков

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2022г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**(БАКАЛАВРСКАЯ РАБОТА)**

**АВТОМАТИЧЕСКАЯ ЖАНРОВАЯ КЛАССИФИКАЦИЯ НА ОСНОВЕ МАШИННОГО ОБУЧЕНИЯ**

Работу выполнил\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ К.Л. Хван

(подпись)

Направление подготовки 02.03.02 — «Фундаментальная информатика и\_\_\_\_\_

(код, наименование)

информационные технологии»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Направленность (профиль) \_\_\_\_\_\_\_\_«Вычислительные технологии»\_\_\_\_\_\_\_\_

Научный руководитель

д-р техн. наук, проф. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ю.М.Вишняков

(подпись)

Нормоконтролер

канд.техн.наук., доц. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.Е.Полупанова

(подпись)

Краснодар

2022

**РЕФЕРАТ**

Выпускная квалификационная работа 47 стр., 5 ч., 40 рис., 9 источников.

КЛАССИФИКАЦИЯ ЖАНРОВ, ВЕКТОРНОЕ ПРЕДСТАВЛЕНИЕ СЛОВ, МАШИННОЕ ОБУЧЕНИЕ, РЕКУРРЕНТЫЕ НЕЙРОННЫЕ СЕТИ, ПОЛНОСВЯЗАННЫЕ НЕЙРОННЫЕ СЕТИ, СВЕРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ

Объектом исследования в данной работе является классификаторы жанров, имеющие в основе своей следующие архитектуры: полносвязанная сеть, LSTM, GRU, сверточная сеть.

Цель работы: построение и анализ классификаторов жанров произведений, написанных на русском языке, на основе моделей машинного обучения, имеющих различные архитектуры нейронных сетей, с последующим выбором лучшей модели.

Методологическая основа исследования включает в себя сбор статистической информации (подсчет времени обучения и точность предсказания на тестовых данных), графический метод (построение графиков зависимостей точности на тренировочных и проверочных данных для каждой модели), анализ полученных зависимостей.

В результате работы были реализован алгоритм очищения, токенизации и разметки текста, обучена модель Word2Vec на этом очищенном тексте, спроектированы и обучены такие модели нейронной сети, как полносвязанная, сверточная, LSTM и GRU, для классификации жанров произведений.

Научная новизна работы заключается в том, что предлагается новый автоматический классификатор жанров произведений, написанных на русском языке.

Произведено сравнение разработанных моделей между собой и по результатам экспериментальных исследований удалось выяснить, что модель с GRU оказался самым точным по классификации жанра на тестовых данных, на произвольных – LSTM и полносвязанная.

СОДЕРЖАНИЕ

[Введение 4](#_Toc106446744)

[1 Задачи классификации и определения жанра произведения 6](#_Toc106446745)

[1.1 Постановка задачи 6](#_Toc106446746)

[1.2 Задача определения жанра 6](#_Toc106446747)

[2 Методы предобработки текста 8](#_Toc106446748)

[3 Векторное представление слов 10](#_Toc106446749)

[3.1 Прямое кодирование слов 10](#_Toc106446750)

[3.2 Векторное представление слов 10](#_Toc106446751)

[3.3 Word2Vec 12](#_Toc106446752)

[4 Обзор архитектур нейронных сетей 14](#_Toc106446753)

[4.1 Полносвязанные нейронные сети 14](#_Toc106446754)

[4.2 Сверточные нейронные сети 15](#_Toc106446755)

[4.2.1 Общая информация о сверточных нейронных сетях 15](#_Toc106446756)

[4.2.2 Применение сверточных нейронных сетей к текстам 17](#_Toc106446757)

[4.3 Рекуррентные нейронные сети 18](#_Toc106446758)

[4.3.1 Простые рекуррентные сети 18](#_Toc106446759)

[4.3.2 LSTM и GRU сети 20](#_Toc106446760)

[5 Эксперименты 25](#_Toc106446761)

[5.1 Набор данных для классификации 25](#_Toc106446762)

[5.2 Предобработка текста 26](#_Toc106446763)

[5.3 Векторное представление слов 26](#_Toc106446764)

[5.4 Обучение нейронных сетей и сравнение результатов 29](#_Toc106446765)

[5.4.1 Полносвязанная сеть 30](#_Toc106446766)

[5.4.2 Сверточная сеть 34](#_Toc106446767)

[5.4.3 Рекуррентные сети 38](#_Toc106446768)

[Заключение 45](#_Toc106446769)

[Список использованных источников 47](#_Toc106446770)

[Приложение A Файл предобработки текста 48](#_Toc106446771)

[Приложение Б Файл описания обучения моделей 49](#_Toc106446772)

# ВВЕДЕНИЕ

Актуальность данной работы состоит в необходимости разработки нового автоматического классификатора жанров произведений, написанных на русском языке. За все время существования человечества были написаны десятки миллионов, а то и сотни художественных произведений, в основе которых лежит текст. Только благодаря ему человек черпает знания и понимает, что хотел донести автор. К тому же текст всегда был одним из самых распространенных видов информации, которая растет в ускоренных темпах благодаря развитию Интернета. В современном мире анализировать и классифицировать это огромное количество информации вручную – неимоверно тяжелая и иногда невозможная задача. Поэтому в таких случаях применяется автоматизация классификации текста (произведений) с использованием разных инструментов классификации, в частности с помощь. машинного обучения. В настоящее время существуют множество классификаторов и работ, связанных с определением тональности текста (положительный, отрицательный окрас). Но очень мало работ по жанровой классификации в сравнение с ними.

Основная цель работы – построение и анализ классификаторов жанров произведений, написанных на русском языке, на основе моделей машинного обучения, имеющих различные архитектуры нейронных сетей, с последующим выбором лучшей модели.

Для реализации поставленной цели предполагается решить следующие задачи:

* изучить методы предобработки текстов, векторные представления слов и архитектуры нейронных сетей;
* построить и обучить модели для классификации жанров произведений;
* сравнить между собой модели и выбрать оптимальный по времени обучения и точности предсказания.

Объектом исследования в данной работе являются классификаторы жанров, имеющие в основе следующие архитектуры:

* полносвязанная сеть;
* LSTM;
* GRU;
* сверточная сеть.

Предметом исследования является время и качество предсказания жанров моделей нейронной сети.

Информационной базой исследования являются результаты анализа моделей классификаторов. Методологическая основа исследования включает в себя сбор статистической информации (подсчет времени обучения и точность предсказания на тестовых данных), графический метод (построение графиков зависимостей точности на тренировочных и проверочных данных для каждой модели), анализ полученных зависимостей.

Научная новизна работы заключается в том, что предлагается новый автоматический классификатор жанров произведений, написанных на русском языке.

Теоретическая и практическая значимость работы состоит в широком применении автоматической жанровой классификации в различных областях, связанных с работой c произведениями. Например, в коммерческих организациях, занимающихся продажей книжных изделий, в интернет-маркетинге для пересмотра стратегии продаж, в библиотеках для осуществления автоматической сортировки книг на жанры.

# 1 Задачи классификации и определения жанра произведения

## 1.1 Постановка задачи

Пусть задано конечное множество классов (жанров) , конечное множество документов (произведений) , и неизвестная функция , которая для каждой пары <документа, класс> определяет, соответствуют ли они друг другу [1].

Задача состоит в том, чтобы подобрать такую функцию , которая была бы максимально близка к . Функцию называют классификатором.

В машинном обучении в начальной коллекции документов используются уже помеченные документы, которые делятся на две непересекающиеся коллекции документов: обучающую и тестовую.

С помощью обучающей коллекции обучается классификатор , основываясь на замеченных характеристиках документов.

С помощью тестовой проверяется эффективность построенного классификатора. Каждый тестовый документ подается на вход классификатору , а затем сравнивается с реальным значением. Чем больше совпадений, тем лучше классификатор справляется со своей задачей.

## 1.2 Задача определения жанра

Под жанром (от франц. genre — род, вид) в литературе понимают тип произведения. Если обратиться к истории возникновения и зарождения литературы в целом, то можно отметить разделение народного творчества на такие виды, как роман, поэма, баллада и другие.

Рассматривая понятие жанр относительно теоретического аспекта, нельзя не заметить определенные черты, которые помогают однозначно определить характерные особенности различных групп произведений независимо от эпохи создания книги или ее принадлежности к какой-либо нации. Этими чертами могут являться абсолютно разные, но обоснованные принципы: общий характер тематики (например, роман бытовой, авантюрный, психологический, социально-утопический, исторический, детективный, научно-фантастический, приключенческий и т. д.), свойство образности (сатира гротескная, аллегорическая, бурлескная, фантастическая и т. д.), тип композиции (лирическое стихотворение в форме сонета, рондо, триолета, газели на Среднем Востоке, хокку и танки в японской поэзии) [2].

В данной работе, чтобы в дальнейшем не было недосказанности, под жанром будем пониматься именно тематика произведения. Задача же определения жанра будет заключаться в поиске характерных признаков того или иного жанр.

# 2 Методы предобработки текста

Текст – это один их самых распространенных видов информации. Его можно интерпретировать как последовательности символов, либо же как последовательности слов. В нашей случае текст будет использовать на уровне слов.

Модели глубокого обучения не понимают текст, как это делает человек. Они лишь показывают статистическую структуру письменного языка, что вполне достаточно для многих простых задач обработки текста.

Поэтому перед началом обучения модели необходимо выполнить начальную предобработку текста [3].

Под начальной предобработкой текста понимают выполнение следующих семи шагов.

1. Лексеммизация: весь текст разбивается на список отдельных элементов (слов (токенов), n-грамм).
2. Преобразование в нижний регистр: слово, начинающееся с заглавной буквы (например, «Собака»), и то же слово, находящееся в середине (например, «собака»), не имеют между собой смысловых различий. Таким образом, преобразовывая все слова в нижний регистр, игнорируется выделение на заглавные и строчные буквы.
3. Удаление стоп-слов: стоп-слова – это слова, которые очень часто встречаются в тексте, но не несут большого смыслового значения, например «и», «а», «которые», «нет», «не может», «не» и т. д. В некоторых случаях, например в задачах классификации по тональности, рекомендуется делать исключения и не удалять стоп-слова, делающие высказывание отрицательным.
4. Удаление знаков препинания и цифр: практически всегда знаки препинания и цифры не вносят точности в модели естественного языка, поэтому их и удаляют.
5. Стемминг – это усечение слов до их основы. Он позволяет объединить слова с одинаковыми значениями в одну лексему, что способствует выявлению большего количества примеров контекстов для этих слов. Обычно стемминг используется только для небольших корпусов, для больших его использовать не имеет смысла.
6. Обработка n-грамм: n-граммы представляют собой n подряд идущих слов. В естественном языке часто встречаются n-граммы, например биграммы. Поэтому их лучше учитывать как единое целое, а не каждый по отдельности.
7. Лемматизация: использование стемминга иногда может приводить к потери изначального смысла некоторых слов, вследствие чего может нарушиться целостная картина восприятия этого предложения в корпусе. Например, при стемминге слово «генеральный» и «генеральский» преобразуются сразу в слово «генерал», хотя значения двух этих слов разное. Поэтому лучшим способом избежать этого является использование лемматизации, т. е. приведение к начальной форме слова.

В русском языке нормальными формами считаются следующие морфологические формы:

* + для существительных – именительный падеж, единственное число;
  + для прилагательных – именительный падеж, единственное число, мужской род;
  + для глаголов, причастий, деепричастий – глагол в инфинитиве (неопределённой форме) несовершенного вида.

# 3 Векторное представление слов

## 3.1 Прямое кодирование слов

Прямое кодирование слов (one-hot encoding) – это один из самых простых методов преобразования слов (токенов) в численные векторы (тензоры). Этот метод заключается в том, что каждому слову присваивается уникальный целочисленный идентификатор , который преобразуется в бинарный вектор размера (размер словаря всех слов). Все элементы этого вектора являются нулями, кроме  элемента, который является единицей.

Пример кодирования предложения «What is the answer to everything?» изображен на рисунке 1:

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1 – Представление слов в виде векторов (one-hot encoding)

Очевидный минусом такого представления является большое количество памяти необходимое для такого кодирования большого объема данных и разреженность полученных векторов.

## 3.2 Векторное представление слов

Другим популярным и мощным способом связывания вектора со словом является использование плотных векторов слов, или векторного представления слов (word embeddings) [4]. В отличие от векторов, полученных прямым кодированием, — бинарных, разреженных (почти полностью состоящих из нулей) и с большой размерностью (их размерность совпадает с количеством слов в словаре) — векторные представления слов являются малоразмерными векторами вещественных чисел (то есть плотными (dense) векторами, в противоположность разреженным), как показано на рисунке 2.

В отличие от векторов, полученных прямым кодированием, векторные представления слов конструируются из данных. При работе с огромными словарями размерность векторов слов нередко может достигать 256, 512 или 1024. С другой стороны, прямое кодирование слов обычно влечет за собой создание векторов с числом измерений 20 000 или больше (при использовании словаря с 20 000 токенов). Иначе говоря, векторное представление слов позволяет уместить больший объем информации в меньшее число измерений.

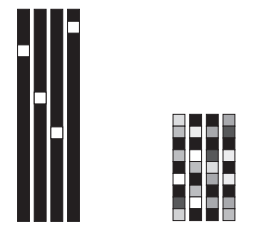


Рисунок 2 – Разреженные векторы (one-hot encoding) и векторные представления (плотные векторы)

Получить векторные представления можно двумя способами.

1. Создавать собственные векторные представления на основе собственных данных при решении основной задачи. В этом случае изначально создаются случайные векторы слов, которые потом обучаются по аналогии весов в нейронных сетях.
2. Загрузить уже векторные представления, полученные при решении других задач, отличных от решаемой. Такие векторные представления называют предварительно обученными векторными представлениями.

В своей задаче же планируется создавать свое векторное представление слов на основе собственного корпуса с использованием алгоритма Word2Vec.

## 3.3 Word2Vec

Word2Vec – это алгоритм обучения без учителя, не использующих никаких меток, которые могут или не могут быть в корпусе [5]. Таким образом, на вход этому алгоритму можно подавать любой набор текстовых данных на естественном языке. Это является огромный плюсом, потому что для разных предметных областей мы имеем возможность создавать собственные векторные представления.

Алгоритм имеет две разные базовые архитектуры обучения – *skip-gram* (SG) или *continuous bag of words* (CBOW).

Суть этого алгоритма можно объяснить на примере предложения «What is the answer to everything?»

Рассмотрим слово «answer» как *целевое,* а два слова слева и справа как *контекстные* (количество рассматриваемых слов, как контекстных, является гиперпараметром, который необходимо учитывать при запуске алгоритма). В архитектуре CBOW целевое слово предсказывается на основе контекстных слов. В архитектуре SG, наоборот, контекстные слова предсказываются на основе целевого. На рисунке 3 изображены эти отличия.

Как было сказано раннее в архитектуре CBOW целевое слово предсказывается как среднее из окружающих его контекстов, причем не имеет значения в каком месте стоят контекстные слова относительно друг друга и целевого слова, т. е. здесь используется механизм BOW («bag of words» или «мешок слов»):

1. берутся все контекстные слова в окне, справа и слева от целевого;
2. добавляются эти слова в «мешок» (словарь);
3. вычисляется среднее значение для всех контекстных слов в мешке и на основании среднего оценивается, какое целевое слово должно быть.

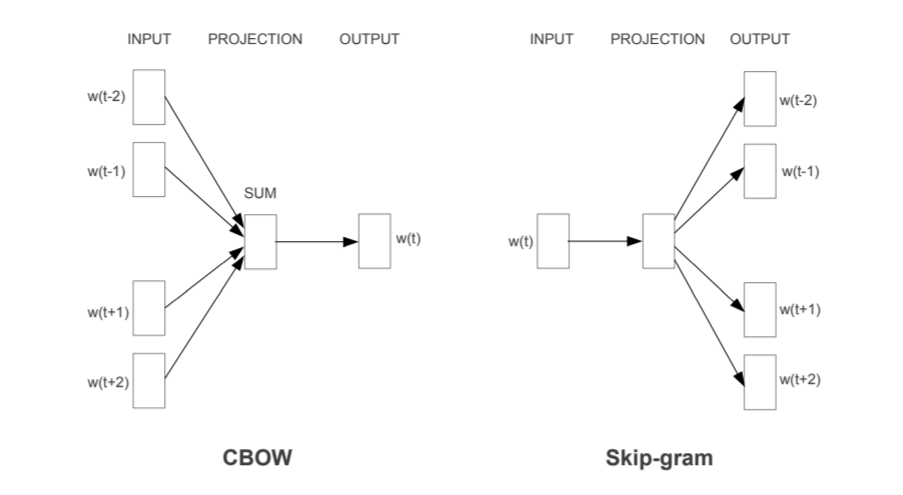


Рисунок 3 – Обобщенная схема работы моделей с архитектурами CBOW и SG

Буква C, или «continuous» означает, что скользящее окно, включающее целевое и контекстные слова, непрерывно скользит по тексту, слово за словом, от первого слова в корпусе до последнего. В каждой позиции на этом пути целевое слово оценивается с учетом контекстных слов. С помощью стохастического градиентного спуска расположение слов в векторном пространстве может сдвигаться и соответственно, оценки целевых слов могут постепенно улучшаться.

Многие специалисты рекомендуют использовать модели с архитектурой SG для небольших корпусов. Но в данной работе будет использоваться только модель c архитектурой CBOW.

# 4 Обзор архитектур нейронных сетей

## 4.1 Полносвязанные нейронные сети

Первая классическая архитектура – это полносвязанные нейронные сети, или многослойный персептрон [6]. На рисунке 4 изображен пример такой сети.

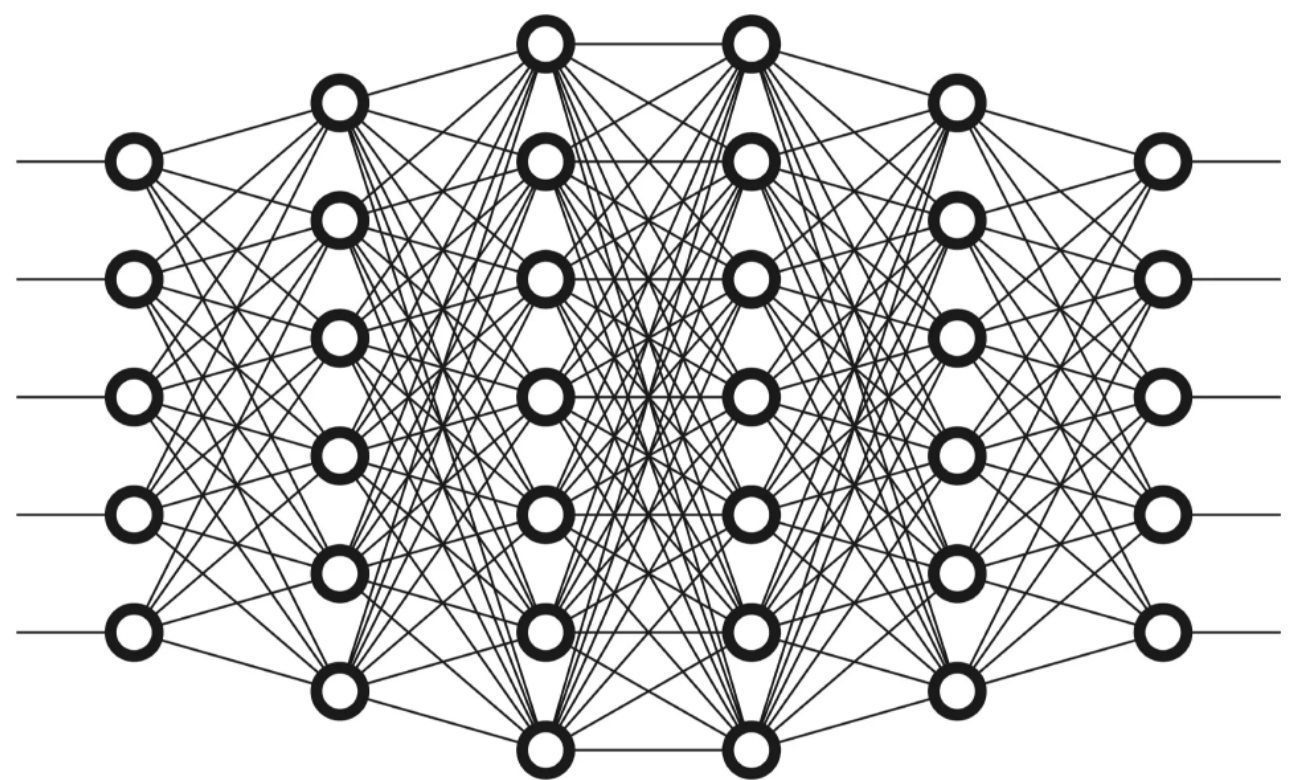


Рисунок 4 – Пример полносвязанной сети

Как понятно из названия, в этой сети каждый нейрон связан со всеми нейронами предыдущего слоя. С помощью такой сети можно решать многие задачи классификации.

Впрочем, полносвязанная сеть имеет два существенных недостатка.

1. Огромное количество параметров. К примеру, на вход сети с 4 слоями приходит картинка размером 100100. Это 10000 параметров только на входе, которые потом заводятся на остальные слои. В общем и целом, у этой сети получается около 10000000 параметров, что плохо сказывается на производительности и обучении. Чем больше параметров в сети, тем выше шанс переобучения.
2. Затухающие градиенты. Как мы знаем основной принцип работы нейронной сети заключается в методе обратного распространения ошибки, когда ошибка с выходов отправляется на вход, корректируя все веса по пути, используя градиенты (производные этих ошибок). Но когда в нейросети очень много слоев, то эти градиенты затухают, что приводит к тому, что веса в первых слоях не будут обновляться.

## 4.2 Сверточные нейронные сети

### 4.2.1 Общая информация о сверточных нейронных сетях

Сверточные сети впервые дали о себе знать в 2012 году на соревновании ImageNet, где победителем стала AlexNet. После этого события произошел всплеск интереса к этим сетям.

Основной идеей сверточных сетей является операция свертки, которая изображена на рисунке 5.

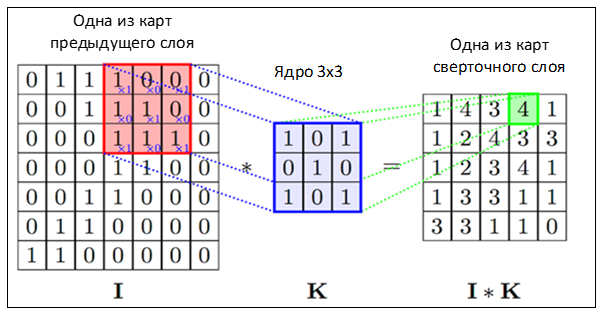


Рисунок 5 – Пример операции свертки

Окно, скользящее по матрице, называется фильтром (ядром). Этот фильтр накладывается, собственно, на участок матрицы (в примере это участок размером 33) и каждое значение перемножается с соответствующим ему значением фильтра. Получившиеся значения складываются и получается выходное значение. Скользить же по матрице фильтр может с каким-то шагом, который может быть как горизонтальным, так и вертикальным.

В сверточных сетях обычно используется не один фильтр, а большое их количество. Каждый такой фильтр инициализируется случайными весами, вследствие чего в процессе обучения нейронной сети они обнаруживают множество разных признаков. Например, один фильтр смотрит на определенный цвет, другой на наличие углов, а третий на окружности.

Кроме операции свертки существует еще слой так называемой подвыборки. Суть работы этого слоя изображена на рисунке 6.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 6 – Пример операции подвыборки

Как видно из рисунка 6 мы просто выбираем максимальный элемент. Этот процесс называется max-pooling. Вместо функции максимума может быть любая другая арифметическая функция (например, выбор среднего значения).

Далее в сверточных сетях обычно идут обычные полносвязанные сети для классификации изображений/текстов.

### 4.2.2 Применение сверточных нейронных сетей к текстам

Принцип работы сверток с текстом, изображенный на рисунке 7, очень похож на работу с изображениями.

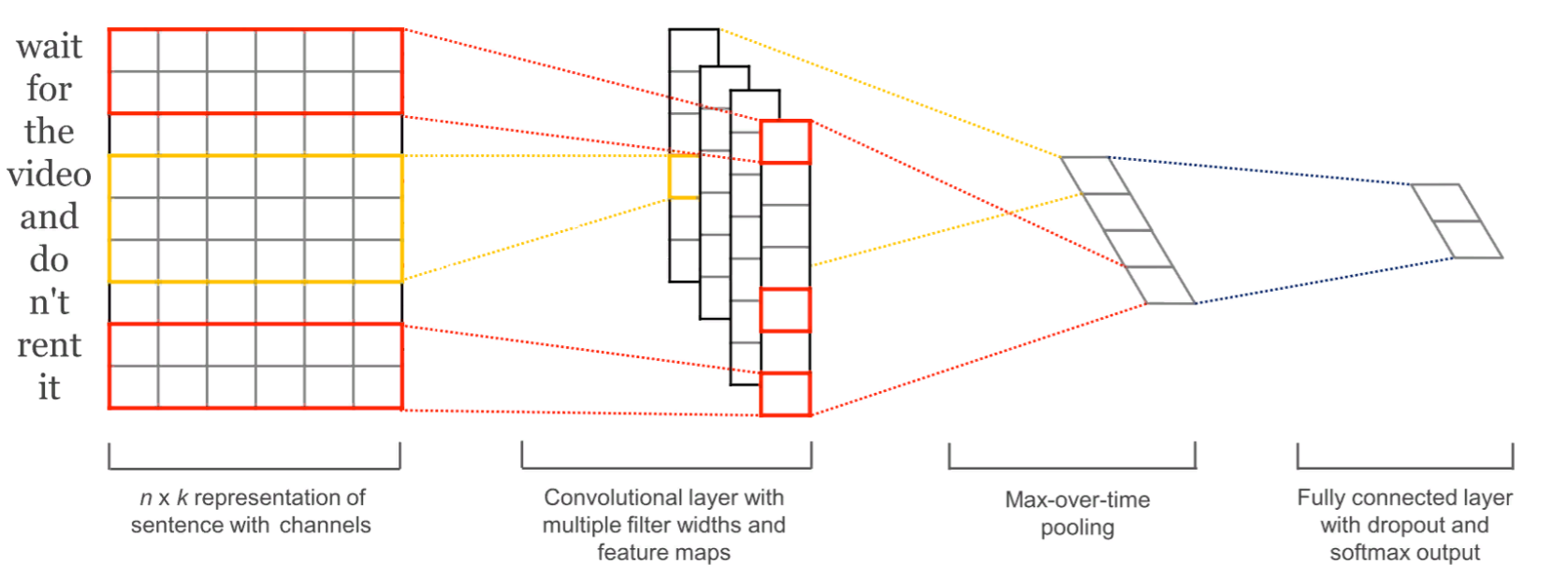


Рисунок 7 – Пример работы сверточных нейронных сетей с текстом

Из рисунка видно, что сверточные сети работают не конкретно словами, а их векторами, а точнее с матрицей векторов слов. Векторное представление слов уже было описано в главе 3.

Если изображение состоит из трех измерений – ширины, высоты и канала, то текст всего из одного измерения – канала (размерность вектора) Поэтому свертка может ходить только по одной оси – по ширине. Поэтому для того, чтобы отличить от стандартной свертки, ее называют одномерной (1D convolution).

Также на изображении 7 описан процесс Max Over Time Pooling. Это уже обсуждавшийся выше max-pooling, только примененный ко всей последовательности сразу.

Сверточные нейронные сети хороши там, где нужно увидеть кусочек или всю последовательность целиком и сделать какой-то вывод из этого. То есть это задачи, например, детекции спама, анализа тональности или извлечения именованных сущностей.

## 4.3 Рекуррентные нейронные сети

### 4.3.1 Простые рекуррентные сети

Читая книгу, человек понимает каждое слово, основываясь на прочитанном раннее. Люди не выбрасывают из головы все и не начинают думать с чистого листа. А все по тому, что человеческие мысли обладают очень примечательным свойством – постоянством.

Полносвязанные и сверточные сети не обладают этим свойством. Например, очень сложно представить как эти сети на основе предыдущих событий в фильме, будут предсказывать последующие.

Решить данную проблему могут рекуррентные нейронные сети, которые имеют обратные связи и позволяют сохранять информацию. Пример такой сети изображен на рисунке 8.

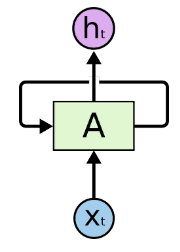


Рисунок 8 – Упрощенная рекуррентная сеть

На этом рисунке изображены фрагмент нейронной сети , который на вход принимает значение , а возвращает значение . С помощью обратной связи можно передавать информацию от одного шага сети к другому.

Также рекуррентную сеть можно рассматривать как несколько копий одно и той же сети, каждая из которых будет передавать информацию последней. Рассмотрим развернутую схему на рисунке 9.

Изображение выглядит как текст, часы

Автоматически созданное описание

Рисунок 9 – Упрощенная рекуррентная сеть (развернутая)

Смотря на рисунок 9, можно сделать вывод о том, что рекуррентная сеть напоминает цепочку, что может говорить о том, что они тесно связаны с последовательностями и списками (тексты).

Основная проблема простых рекуррентных сетей заключается в том, что они теряют способность связывать информацию по мере роста расстояния между входами и выходами. На рисунке 10 изображен такой случай.

Изображение выглядит как текст, часы, крыша

Автоматически созданное описание

Рисунок 10 – Упрощенная рекуррентная сеть (пример недостатка)

Поэтому простые рекуррентные сети мало где используется, на замену им пришли сети LSTM и GRU сети

### 4.3.2 LSTM и GRU сети

Долгая краткосрочная память (Long short-term memory; LSTM) – особая разновидность рекуррентных сетей, способная обучаться долговременным зависимостям. Эти сети были представлены Зеппом Хохрайтером и Юргеном Шмидхубером в 1997 году, а затем улучшены и изложены в работах многих других ученых-исследователей.

В LSTM уже нет проблемы долговременной зависимости благодаря своей архитектуре, изображенной на рисунке 11 [7].

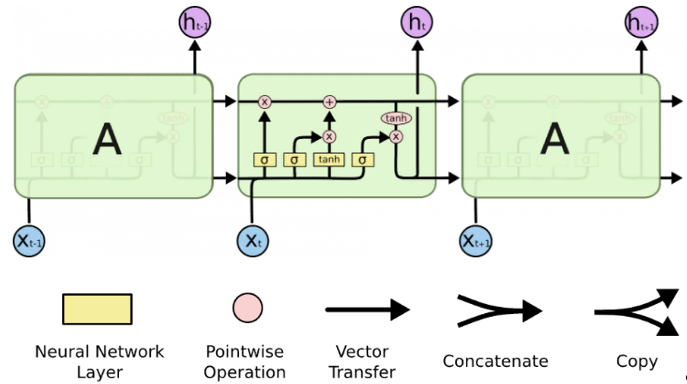


Рисунок 11 – Схема ячейки LSTM сети

На этом изображении каждая линия переносит целый вектор от входа одного узла ко входу другого. Желтые прямоугольники – это уже обученные слови нейронной сети, а розовым кругами обозначены поточечные операции, такие как сложение векторов и умножение векторов. Сливающие стрелки означают объединение, а разветвляющие – копирование.

Основная идея таких сетей состоит в состоянии ячейки – горизонтальной линии, которая проходит по верхней части схемы на рисунке 12.

Изображение выглядит как текст, часы

Автоматически созданное описание

Рисунок 12 – Состояние ячейки LSTM сети

Состояние ячейки – по сути своей является конвейерной лентой, которая проходит напрямую через всю цепочку, участвуя лишь в нескольких линейных преобразованиях.

Также LSTM с помощью фильтров (gates) может удалять информацию из ячейки. Процесс фильтрации изображен на рисунке 13.

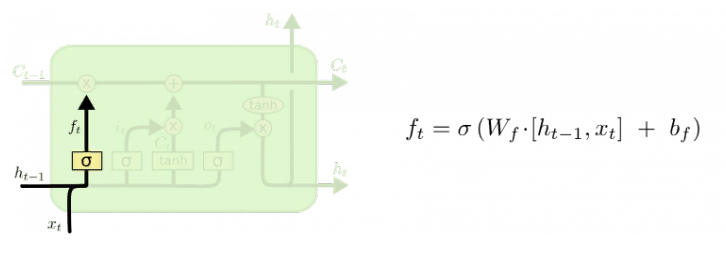


Рисунок 13 – Процесс фильтрации

Выше изображен шаг определения важности информации. Решение оставлять или забыть информацию берет на себя сигмоидальный слой, который также называют «слоем фильтра забывания». В зависимости от и этот слой возвращает число от 0 до 1 для каждого числа из состояния .

Затем необходимо определить, какую новую информацию должны хранить в состоянии ячейки. Для этого сигмоидальный слой определяет, какие значения необходимо обновить. А затем слой tanh строит вектор новых значений , которые добавляются в состояние ячейки. Этот шаг изображен на рисунке 14.

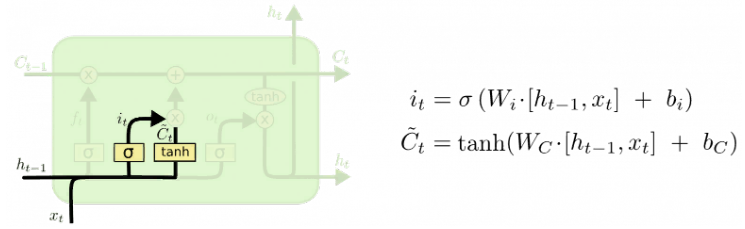


Рисунок 14 – Обновление состояния ячейки

Наконец необходимо получить новое состояние ячейки , используя . Для этого нужно умножить старое состояние на . Затем прибавляем (это новые значение-кандидаты, умноженные на tпоказывают, насколько мы хотим обновить каждое из значений состояния). На рисунке 15 проиллюстрирован этот процесс.

*Изображение выглядит как текст, часы

Автоматически созданное описание*

Рисунок 15 – Замена состояния ячейки

Состояние ячейки мы обновили. Теперь же нужно решить, какую информацию, мы хотим получить на выходе. Для начала применяется сигмоидальный слой, решающий какая информация из состояния, должна выводиться. Затем информация состоянии ячейки проходит через слой tahn. После этого слоя на выходе получаются значения в диапазоне от -1 до 1, которые перемножаются с выходными значениями сигмоидального слоя, что позволяет выводить только требуемую информацию. На рисунке 16 проиллюстрирован последний шаг.

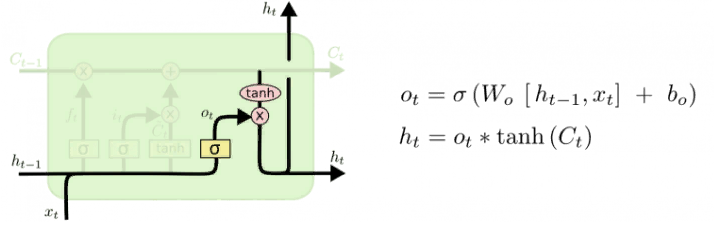


Рисунок 16 – Процесс определения выходной информации

Только что было рассмотрен процесс работы обычной LSTM сети, но существуют разные ее вариации, например, GRU (Gated recurrent units) сети, вариант одной ячейки такой сети изображен ниже на рисунке 17.

Изображение выглядит как текст, часы

Автоматически созданное описание

Рисунок 17 – Общая схема GRU ячейки

Принципе работы аналогичен LSTM сетям, но здесь фильтры «забывания» и входа объединяются в один фильтр «обновления», а также состояние ячейки объединяется со скрытым состоянием. В результате получается, что эта модель проще, чем стандартная LSTM, а значит вычислительная нагрузка при обучении ниже, что способствует меньшему времени обучения.

# 5 Эксперименты

## 5.1 Набор данных для классификации

Набор данных для классификации состоит из произведений 3 жанров: «научная фантастика», «детектив» и «фэнтези». Выбор этих жанров был обусловлен тем, что семантически это различные друг от друга жанры, которые в общем случае не пересекаются. Все произведения представлены в формате txt.

Так для жанра «детектив» были выбраны шесть самых известных произведений:

1. «Глубокий сон» Чэндлер Рэймонд;
2. «Что скрывал покойник» Пенни Луиза;
3. «Рассветная бухта» Тана Френч;
4. «Хирург» Тесс Герритсон;
5. «Полиция» Ю Несбё;
6. «Жажда» Ю Несбё.

Для жанра «научная фантастика» тоже были выбраны наиболее популярные книги:

1. «Путеводитель по Галактике для путешественника» Адам Дуглас;
2. «Рейдер» Андрей Буянов;
3. «Машина времени» Герберт Уэллс;
4. «Аэлита» Алексей Толстой;
5. «Путешествие к центру земли» Жюль Верн;
6. «Дюна» Герберт Фрэн;
7. «О дивный новый мир» Олдос Хаксли.

А также культовые произведения с жанром «фэнтези»:

1. «Эрагон» Кристофер Паолини;
2. «Наследие» Кристофер Паолини;
3. «Хроники средиземья» Джон Толкиен;
4. «Возвращение» Кристофер Паолини;
5. «Танец с драконами. Грезы и пыль» Джордж Мартин.

## 5.2 Предобработка текста

Набор произведений необходимо обработать и преобразовать в плотные вектора для обучения нейронной сети. Для этого был разработан скрипт на языке Python, который подготавливает эти данные:

1. считывает произведения из текстовых файлов;
2. размечает в соответствие с жанром;
3. проводит предобработку данных (приведение в нижний регистр, токенизация, лемматизация);
4. делит весь токенизированный текст на наборы размеченных данных по 100, 200, 300, 400, 500 слов;
5. сохраняет эти наборы в файлы с форматом pickle и названием «train\_data\_{количество слов}.pickle».

## 5.3 Векторное представление слов

На основе всего корпуса данных вычисляется векторное представление слов с помощью алгоритма Word2Vec. Для каждого слова ставится в соответствие вектор размерности 100. Человеческий мозг плохо воспринимает графики, имеющие более трех измерений. Поэтому для наглядности распределения слов на рисунке 18 изображена двумерная диаграмма рассеяния векторного пространства слов.

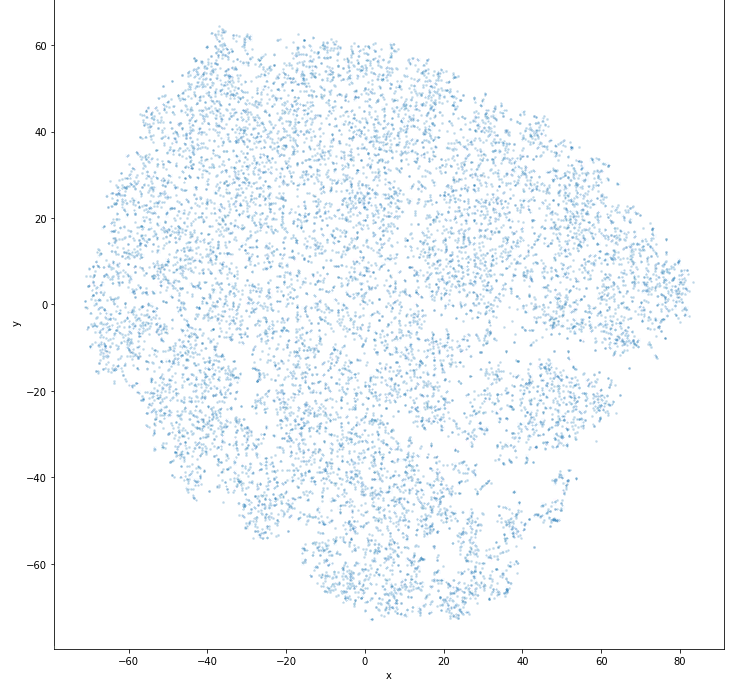


Рисунок 18 – Диаграмма рассеяния векторного пространства слов

Эта диаграмма хоть показывает интересное распределение слов, но она недостаточно информативна, из нее нельзя извлечь много информации. Поэтому на рисунке 19 изображен интерактивный график рассеяния векторного пространства слов, где можно наглядно посмотреть связи между словами.

Изображение выглядит как текст, силуэт

Автоматически созданное описание

Рисунок 19 – Интерактивная диаграмма рассеяния векторного пространства слов

Увеличив масштаб, можем увидеть более подробную информацию. Это изображено на рисунке 20.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 20 – Диаграмма с увеличенным масштабом

В этом примере можно заметить, как объединяются родственные слова, особенно синонимы. Например, можно отметить близость слов «больной» - «бедный», «юный» - «гордый».

Также в этом исследовании можно подметить на недостатки этапов предварительной обработки, как, например, присутствие знаков препинания, выделение биграмм или других лексем, которые было бы желательно не включать в словарь.

## 5.4 Обучение нейронных сетей и сравнение результатов

Выполнив предобработку данных и преобразовав их в тот вид, который поймет нейросеть, необходимо будет выбрать архитектуру нейронной сети для классификации жанров произведений. Всего планируется сравнить четыре архитектуры: обычную полносвязанную сеть, сверточную сеть, LSTM и GRU - сети.

### 5.4.1 Полносвязанная сеть

Для начала рассмотрим самую простую из всех возможных архитектур – полносвязанную сеть, которая изображена на рисунке 4, и обучим ее на подготовленном наборе данных [8].

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 21 – Архитектура полносвязанной сети

На рисунке 21 представлена последовательная модель глубокого обучения, которая состоит из четырех слоев [9].

1. Слой Embedding, позволяющий отображать целочисленные индексы (обозначающие конкретные слова) в плотные (dense) вектора. Он принимает целые числа на входе, отыскивает их во внутреннем словаре и возвращает соответствующие векторы. В данном случае размер словаря и количество измерений векторного представления определяются с помощью аргументов max\_words равным 10000 и embedding\_dim равный 100. И поскольку этот слой является еще и входным, то необходимо передать форму входного слоя с помощью параметра maxlen, который равен 100.
2. Далее идет слой Flatten, который двумерный слой преобразует в одномерный.
3. Дальше идут полносвязанные слои, состоящие из 100 нейронов и функций активации ReLu. К этим слоям с целью предотвращения переобучения применяются слои прореживания с вероятностью dropout равным 0,5.
4. Наконец выходной слой состоит из трех нейронов и функцией активацией SoftMax, потому что решается задача классификации произведений на 3 жанра.

На рисунке 22 изображена краткая информация об архитектуре.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 22 – Краткая информация об архитектуре полносвязанной сети

Как можно заметить, в этой архитектуре всего 2010503 параметров, 1010503 из которых обучаемы, а 1000000 – нет. К тому же в первый слой Embedding необходимо загрузить предварительно обученных векторных представлений слов, полученный после работы алгоритма Word2Vec. Этот шаг будет оставаться одинаковым для всех типов архитектур.

Далее нужно скомпилировать и обучить модель. На рисунке 23 изображен параметры и процесс обучения.

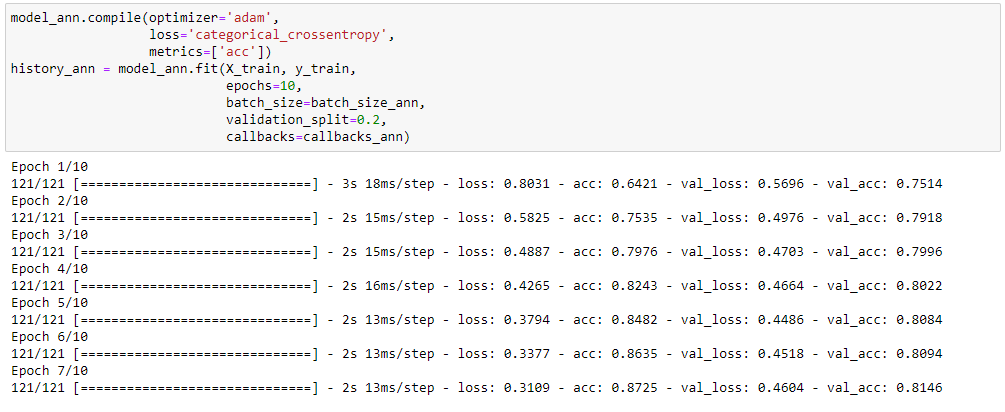


Рисунок 23 – Параметры и процесс обучения полносвязанной сети

Функция стоимости была выбрана categorical\_crossentropy, так как решается задача мультиклассификации. Количество эпох было выбрано равным десяти, а размер пакета равным 64. Также было выделено 20 процентов данных для валидации от тренировочных.

Как показано на рисунке 23 наименьшие потери (0.449) и наивысшая точность (0.81) при проверки были достигнуты после шестой эпохи. После происходит переобучение. На рисунке 24 как раз изображены графики точности и потерь.

На этих графиках видим, что сеть переобучается после третьей эпохи.

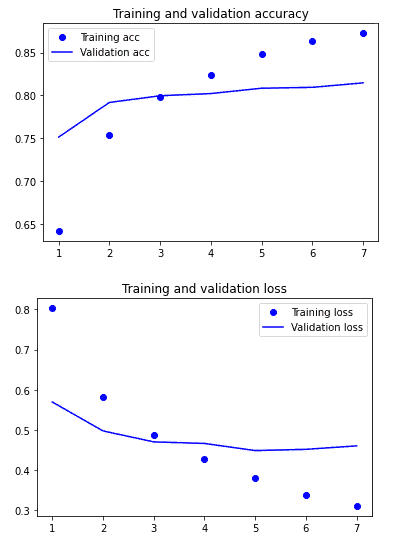


Рисунок 24 – Графики процесса обучения полносвязанной сети

Несмотря на это точность предсказания тестового набора данных, изображенной на рисунке 25, немного отличается от максимальной.

Изображение выглядит как текст

Автоматически созданное описание Рисунок 25 – Проверка точности предсказания полносвязанной сети на тестовом наборе данных

Как видим из рисунка 25 точность составляет всего лишь 0.83, а потери при этом 0.45.

### 5.4.2 Сверточная сеть

Далее рассмотрим архитектуру нейронной сети с использованием сверточных сетей, которая изображена на рисунке 26, и обучим ее на подготовленном наборе данных.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 26 – Архитектура сверточной сети

На рисунке 26 представлена последовательная модель глубокого обучения, которая состоит из тех же слоев, что были использованы в полносвязанной, но с рядом изменений [9]:

* после слоя Embedding было добавлен слой прореживания с вероятностью 0.2;
* не используются слой Flatten;
* добавлен сверточный слой с функцией активации ReLu и с 256 уникальными фильтрами, каждый из который может специализироваться на активациях при прохождении определенной последовательности из трех лексем;
* добавлено объединение по глобальному максимуму, которое используется для уменьшения размерности.

На рисунке 27 изображена краткая информация об архитектуре.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 27 – Краткая информация об архитектуре сверточной сети

Как можно заметить, в этой архитектуре всего 1143619 параметров, что меньше, чем в полносвязанной, причем только 143,619 обучаемых параметров.

Далее нужно скомпилировать и обучить эту модель. На рисунке 27 изображены параметры и процесс обучения.

Функция стоимости была выбрана categorical\_crossentropy, т. к. решается задача мультиклассификации. Количество эпох было выбрано равным 10, а размер пакета равным 128. Также было выделено 20 процентов данных для валидации от тренировочных.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 27 – Параметры и процесс обучения сверточной сети

По рисунку 27 непонятно, когда происходит переобучение. Поэтому на рисунке 28 раз изображены графики точности и потерь.

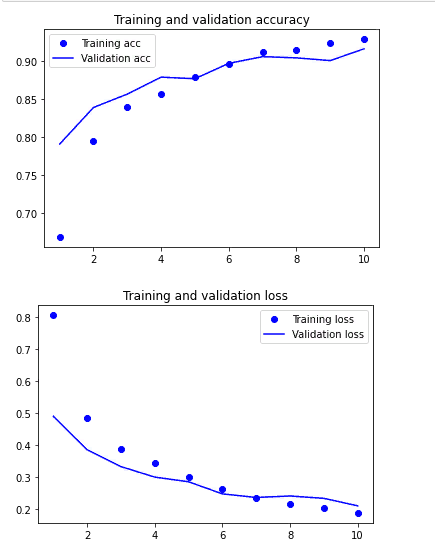


Рисунок 28 – Графики процесса обучения сверточной сети

На этих графиках видим, что сеть не переобучается даже после десятой эпохи. Точность предсказания тестового набора данных изображена на изображении 29.

Изображение выглядит как текст

Автоматически созданное описание Рисунок 29 – Проверка точности предсказания сверточной сети на тестовом наборе данных

Как видим из рисунка 29 точность составляет 0.91 процентов, а потери при этом 0.21, что лучше полносвязанной.

### 5.4.3 Рекуррентные сети

Наконец рассмотрим архитектуру нейронной сети с использованием рекуррентных сетей, а именно LSTM и GRU, которые изображены на рисунке 30 и рисунке 31, и обучим ее на подготовленном наборе данных.

**Изображение выглядит как текст

Автоматически созданное описание**

Рисунок 30 – Архитектура LSTM сети

На рисунке 30 представлен последовательная модель глубокого обучения, которая состоит из тех же слоев, что были использованы в сверточной, но вместо сверточного слоя, здесь используется слой LSTM с прореживанием drop\_lstm равным 0.2.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 31 – Архитектура GRU сети

На рисунке 31 представлена такая же последовательная модель глубокого обучения, как и выше, но вместо слоя LSTM, здесь слой GRU с drop\_gru равным 0.2.

На рисунке 32 изображена краткая информация об архитектуре LSTM.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 32 – Краткая информация об архитектуре LSTM сети

Как можно заметить, в этой архитектуре всего 1366339 параметров, причем 366339 – обучаемые. Это намного больше, чем в сверточной, но меньше, чем в полносвязанной. Скорость обучения этой сети довольно низка.

Далее рассмотрим информацию об архитектуре GRU сети на рисунке 33.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 33 – Краткая информация об архитектуре GRU сети

Здесь уже меньше параметров за счет использования слоя GRU, за счет чего увеличивается скорость обучения.

Далее нужно скомпилировать и обучить модели. Параметры компиляции и запуска остались такими же, как и полносвязанной сети. На рисунке 34 и 35 изображены истории обучения.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 34 – Процесс обучения LSTM сети

Можем видеть наиболее качественные показатели на десятой эпохе обучения, но при этом жертвуем скоростью обучения.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 35 – Процесс обучения GRU сети

Лучшие показатели можно увидеть на восьмой эпохе, где потери составляют всего 0.146. К тому же скорость обучения выросла практически в два раза по сравнению с LSTM сетью.

Для наглядности посмотрим на графики обучения этих сетей на графиках, изображенных на рисунке 36 и 37.

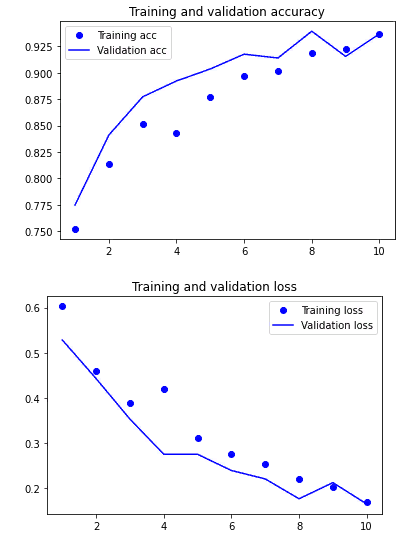


Рисунок 36 – Графики процесса обучения LSTM сети

Тут видим, что при обучении на десяти эпохах переобучения модели не произошло и что наименьшие потери достигаются именно в этой эпохе. Сравним теперь с моделью, использующую GRU.

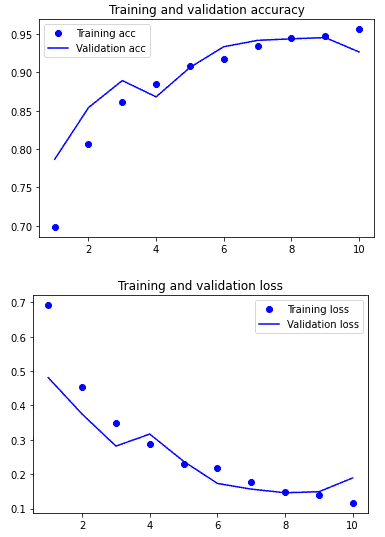


Рисунок 37 – Графики процесса обучения GRU сети

Здесь же графики показывают, что сеть явно переобучается после девятой эпохи, а наименьшие потери достигаются на восьмой. Поэтому настроим модель и проверим точность предсказания на тестовом наборе данных. Результаты предсказания изображены на рисунке 38 для LSTM сети и на рисунке 39 для GRU сети.

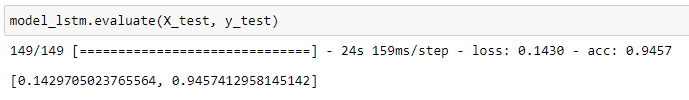


Рисунок 39 – Проверка точности предсказания LSTM сети на тестовом наборе данных

Как видим из рисунка 39 точность составляет 0.945, а потери при этом 0.142, что еще лучше, чем было прежде. Посмотрим теперь на результаты GRU сети.

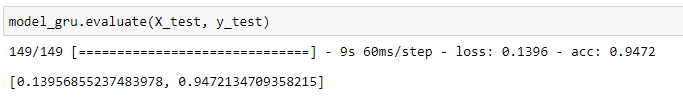


Рисунок 40 – Проверка точности предсказания GRU сети на тестовом наборе данных

GRU же сеть справилась с этой задаче еще лучше, и на тестовом наборе данных мы видим точность равной 0.947, а потери – 0.139. Не слишком велика разница с результатами LSTM сети.

Подводя итоги, можем сделать вывод, что успешнее всего с жанровой классификацией на тестовых данных справились рекуррентные сети GRU (0.947) и LSTM (0.945), похуже сверточные сети (0.913), а в самом конце полносвязанная (0.81) сеть. На произвольных же данных порядок отличается: первое место делят полносвязанные и LSTM сети, которые выдают точность предсказания в 0.549, далее идут GRU с точностью равной 0.539, а на последнем месте сверточные со значением 0.521.

Это связано прежде всего с тем, что векторное представление слов недостаточно полное. Поэтому, когда на вход приходят незнакомые слова, то им в соответствие ставится вектор с нулевыми значениями.

Также стоит отметить, что по скорости обучения лидируют сверточные сети, они обучаются практически вдвое быстрее, чем рекуррентные за счет дешевых операций свертки, а среди рекуррентных GRU.

# ЗАКЛЮЧЕНИЕ

Задача классификации жанров остается по-прежнему актуальной, несмотря на, казалось бы, низкую популярность из-за сложности найти данные для обучения на русском языке и их разметки. Но дальнейшее изучение методов классификации с помощью машинного обучения все равно остается актуальной.

В данной работе были изучены наиболее актуальные и современные для данной задачи методы перевода информации в текстовом формате в целочисленные вектора для последующего их использования в обучении и модели нейронной сети. Был реализован алгоритм очищения, токенизации и разметки текста. Обучена модель Word2Vec на этом очищенном тексте. Спроектированы и обучены такие модели нейронной сети, как полносвязанная, сверточная, LSTM и GRU, для классификации жанров произведений. Произведено сравнение разработанных моделей между собой и по результатам экспериментальных исследований удалось выяснить, что модель с GRU оказался самым точным по классификации жанра на тестовых данных, на произвольных – LSTM и полносвязанная. Также было выявлено, что векторное пространство слов недостаточно велико, что приводит к ухудшению классификации. В дальнейшем планируется учитывать то, что в поставленной задаче обучающая выборка может быть мала и не давать достаточной информации о точности предсказания моделей. Поэтому набор данных для обучения будет дополнен большим количеством классов и произведений. Планируется построение, обучение и анализ работы нейронных сетей на новых данных.

Разработанные модели нейронной сети классификации жанров произведений можно использовать в коммерческих организациях, занимающихся продажей книжных изделий, в интернет-маркетинге для пересмотра стратегии продаж, в библиотеках для осуществления автоматической сортировки книг на жанры и в других областях, занимающихся изучением способов классификации произведения по жанрам.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. **Sorower, Mohammad** A Literature Survey on Algorithms for Multi-label Learning. – Текст: электронный. – URL: https://www.researchgate.net/publication/266888594\_A\_Literature\_Survey\_on\_Algorithms\_for\_Multi-label\_Learning (дата обращения: 15.03.2022).
2. Литературный энциклопедический словарь / Жанр – Текст: электронный. – URL: http://niv.rudoc/encyclopedia/literature/articles/30/zhanr.html (дата обращения: 18.03.2022).
3. **Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean** Efficient Estimation of Word Representations in Vector Space. – Текст: электронный. – URL: https://arxiv.org/pdf/1301.3781.pdf (дата обращения: 15.04.2022).
4. Word2Vec. – Текст: электронный – URL: https://www.tensorflow.org/tutorials/text/word2vec (дата обращения: 25.04.2022).
5. **Крон Джон, Бейлевельд Грант, Аглаэ Бассенс** Глубокое обучение в картинках. Визуальный гид по искусственному интеллекту. — СПб.: Питер, 2020. — 400 с.: ил. — (Серия «Библиотека программиста»). ISBN 978-5-4461-1574-7. – Текст: непосредственный.
6. **Саймон Хайкин** Нейронные сети: полный курс, 2-е издание. Пер. с англ. – М. : Издательский дом «Вильямс», 2006. – 1104 с. : ил. – Парал. тит. англ. ISBN 5-8459-0890-6 (рус.). – Текст непосредственный.
7. Рекуррентные сети LSTM. – Текст: электронный // Блог автора – URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (дата обращения: 02.05.2022).
8. Документация пакета Keras. – Текст: электронный. – URL: https://keras.io/ (дата обращения: 16.05.2022).
9. **Шолле Франсуа** Глубокое обучение на Python. — СПб.: Питер, 2018. — 400 с.: ил. — (Серия «Библиотека программиста»). ISBN 978-5-4461-0770-4. – Текст: непосредственный.

# ПРИЛОЖЕНИЕ A

**Файл предобработки текста**

Файл Preprocessing.py

import re

from string import punctuation

from nltk import word\_tokenize

from nltk.corpus import stopwords

from pymorphy2 import MorphAnalyzer

class Preprocessing:

PUNCTUATION = list(punctuation) + ['–', '—']

STOPWORDS = stopwords.words('russian')

MORPH = MorphAnalyzer()

@classmethod

def remove\_punctuation(cls, text):

return "".join([ch if ch not in cls.PUNCTUATION and ch.isalpha() else ' ' for ch in text])

@classmethod

def remove\_numbers(cls, text):

return "".join([i if not i.isdigit() else ' ' for i in text])

@classmethod

def remove\_multiple\_spaces(cls, text):

return re.sub(r'\s+', ' ', text, flags=re.I)

@classmethod

def preprocessing(cls, text: str):

return text.replace('\xa0', '').replace('\t', '').replace('\n', '')

@classmethod

def tokenize(cls, text: str):

tokens = []

for token in list(map(str.lower, word\_tokenize(text))):

if (token not in cls.PUNCTUATION) and (token not in cls.STOPWORDS):

tokens.append(token.replace('\n', ''))

return tokens

@classmethod

def lemmatize(cls, tokens: list[str]):

lemmatized\_tokens = []

for token in tokens:

var = cls.MORPH.parse(token)[0]

if var.normal\_form not in cls.STOPWORDS:

lemmatized\_tokens.append(var.normal\_form)

return lemmatized\_tokens

@classmethod

def to\_tokens(cls, text: str):

return cls.lemmatize(cls.tokenize(text))

# ПРИЛОЖЕНИЕ Б

**Файл описания обучения модели**

import keras

from keras.preprocessing.sequence import pad\_sequences

from keras.models import Sequential

from keras.layers import Dense, Flatten, Dropout, SpatialDropout1D, Conv1D, GlobalMaxPooling1D, Embedding, GRU, LSTM

from keras.callbacks import EarlyStopping, ModelCheckpoint

from keras.preprocessing.text import Tokenizer

from tensorflow.keras.utils import to\_categorical

from keras.callbacks import ModelCheckpoint

import os

from sklearn.metrics import roc\_auc\_score, roc\_curve

import matplotlib.pyplot as plt

from gensim.models.word2vec import Word2Vec

from sklearn.manifold import TSNE

import pandas as pd

import numpy as np

data = pd.read\_pickle('./datasets/train\_data\_100.pickle')

data\_test = pd.read\_pickle('./datasets/test\_data\_100.pickle')

texts = [' '.join(d) for d in data['text'].to\_list()]

labels = data['genre'].to\_list()

texts\_test = [' '.join(d) for d in data\_test['text'].to\_list()]

labels\_test = data\_test['genre'].to\_list()

texts

texts\_test

maxlen = 100 #Отсечение остатка после 100-го слова

max\_words = 10000

tokenizer = Tokenizer(num\_words=max\_words)

tokenizer.fit\_on\_texts(texts)

sequences = tokenizer.texts\_to\_sequences(texts)

tokenizer\_test = Tokenizer(num\_words=max\_words)

tokenizer\_test.fit\_on\_texts(texts\_test)

sequences\_test = tokenizer.texts\_to\_sequences(texts\_test)

word\_index = tokenizer.word\_index

print('Found %s unique tokens.' % len(word\_index))

word\_index\_test = tokenizer\_test.word\_index

print('Found %s unique tokens.' % len(word\_index\_test))

d = pad\_sequences(sequences, maxlen=maxlen)

d\_test = pad\_sequences(sequences\_test, maxlen=maxlen)

labels = np.asarray(labels)

print('Shape of data tensor:', d.shape)

print('Shape of label tensor:', labels.shape)

labels\_test = np.asarray(labels\_test)

print('Shape of data tensor:', d\_test.shape)

print('Shape of label tensor:', labels\_test.shape)

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(d, labels,

train\_size=0.67,

random\_state=42,

stratify=labels)

print(f"Количество строк в y\_train по классам: {np.bincount(y\_train)}")

print(f"Количество строк в y\_test по классам: {np.bincount(y\_test)}")

X\_train\_test, X\_test\_test, y\_train\_test, y\_test\_test = train\_test\_split(d\_test, labels\_test,

train\_size=3,

random\_state=42,

stratify=labels\_test)

print(f"Количество строк в y\_train по классам: {np.bincount(y\_train\_test)}")

print(f"Количество строк в y\_test по классам: {np.bincount(y\_test\_test)}")

y\_train = to\_categorical(y\_train)

y\_test = to\_categorical(y\_test)

y\_test\_test = to\_categorical(y\_test\_test)

# sentences = list(map(list, data['text'].to\_list()))

# model\_w2v = Word2Vec(sentences=sentences, vector\_size=100, sg=0, window=10, epochs=5, min\_count=10, workers=4)

# model\_w2v.save('./models/model\_w2v.w2v')

model\_w2v = Word2Vec.load('./models/model\_w2v.w2v')

embedding\_dim = 100

embedding\_matrix = np.zeros((max\_words, embedding\_dim))

for word, i in word\_index.items():

if i < max\_words:

if word in model\_w2v.wv.key\_to\_index:

embedding\_vector = model\_w2v.wv.get\_vector(word)

else:

embedding\_vector = np.zeros(embedding\_dim)

if embedding\_vector is not None:

embedding\_matrix[i] = embedding\_vector

embedding\_matrix

## Полносвязанная нейронная сеть

output\_dir\_ann = 'model\_output/dense/'

# обучение:

epochs\_ann = 10

batch\_size\_ann = 64

# архитектура полносвязанной сети:

n\_dense\_ann = 100

dropout\_ann = 0.5

output\_dir\_cnn = 'model\_output/cnn/'

# обучение:

epochs\_cnn = 10

batch\_size\_cnn = 128

# архитектура сверточного слоя:

n\_conv = 256 # фильтры, они же ядра

k\_conv = 3 # длина ядра

# архитектура полносвязанного слоя:

n\_dense\_cnn = 256

dropout\_cnn = 0.2

output\_dir\_gru = 'model\_output/gru/'

# обучение:

epochs\_gru = 10

batch\_size\_gru = 64

# архитектура полносвязанной сети:

drop\_embed\_gru = 0.2

n\_gru = 256

drop\_gru = 0.2

# имя каталога для сохранения результатов:

output\_dir\_lstm = 'model\_output/lstm/'

# обучение:

epochs\_lstm = 10

batch\_size\_lstm = 64

# архитектура полносвязанной сети:

drop\_embed\_lstm = 0.2

n\_lstm = 256

drop\_lstm = 0.2

model\_ann = Sequential()

model\_ann.add(Embedding(max\_words, embedding\_dim, input\_length=maxlen))

model\_ann.add(Flatten())

model\_ann.add(Dense(n\_dense\_ann, activation='relu'))

model\_ann.add(Dropout(dropout\_ann))

model\_ann.add(Dense(n\_dense\_ann, activation='relu'))

model\_ann.add(Dropout(dropout\_ann))

model\_ann.add(Dense(3, activation='softmax'))

model\_ann.layers[0].set\_weights([embedding\_matrix])

model\_ann.layers[0].trainable = False

model\_ann.summary()

model\_cnn = Sequential()

model\_cnn.add(Embedding(max\_words, embedding\_dim, input\_length=maxlen))

model\_cnn.add(SpatialDropout1D(0.2))

model\_cnn.add(Conv1D(n\_conv, k\_conv, activation='relu'))

model\_cnn.add(GlobalMaxPooling1D())

model\_cnn.add(Dense(n\_dense\_cnn, activation='relu'))

model\_cnn.add(Dropout(dropout\_cnn))

model\_cnn.add(Dense(3, activation='softmax'))

model\_cnn.layers[0].set\_weights([embedding\_matrix])

model\_cnn.layers[0].trainable = False

model\_cnn.summary()

model\_gru = Sequential()

model\_gru.add(Embedding(max\_words, embedding\_dim, input\_length=maxlen))

model\_gru.add(SpatialDropout1D(drop\_embed\_gru))

model\_gru.add(GRU(n\_gru, dropout=drop\_gru))

model\_gru.add(Dense(3, activation='softmax'))

model\_gru.layers[0].set\_weights([embedding\_matrix])

model\_gru.layers[0].trainable = False

model\_gru.summary()

model\_lstm = Sequential()

model\_lstm.add(Embedding(max\_words, embedding\_dim, input\_length=maxlen))

model\_lstm.add(SpatialDropout1D(drop\_embed\_lstm))

model\_lstm.add(LSTM(n\_lstm, dropout=drop\_lstm))

model\_lstm.add(Dense(3, activation='softmax'))

model\_lstm.layers[0].set\_weights([embedding\_matrix])

model\_lstm.layers[0].trainable = False

model\_lstm.summary()

callbacks\_ann = [EarlyStopping(monitor='val\_loss', patience=2),

ModelCheckpoint(filepath=output\_dir\_ann + 'best\_model.h5', monitor='val\_loss', save\_best\_only=True)]

callbacks\_cnn = [EarlyStopping(monitor='val\_loss', patience=2),

ModelCheckpoint(filepath=output\_dir\_cnn + 'best\_model.h5', monitor='val\_loss', save\_best\_only=True)]

callbacks\_gru = [EarlyStopping(monitor='val\_loss', patience=2),

ModelCheckpoint(filepath=output\_dir\_gru + 'best\_model.h5', monitor='val\_loss', save\_best\_only=True)]

callbacks\_lstm = [EarlyStopping(monitor='val\_loss', patience=2),

ModelCheckpoint(filepath=output\_dir\_lstm + 'best\_model.h5', monitor='val\_loss', save\_best\_only=True)]

model\_ann.compile(optimizer='adam',

loss='categorical\_crossentropy',

metrics=['acc'])

# history\_ann = model\_ann.fit(X\_train, y\_train,

# epochs=10,

# batch\_size=batch\_size\_ann,

# validation\_split=0.2,

# callbacks=callbacks\_ann)

model\_cnn.compile(optimizer='adam',

loss='categorical\_crossentropy',

metrics=['acc'])

# history\_cnn = model\_cnn.fit(X\_train, y\_train,

# epochs=10,

# batch\_size=batch\_size\_cnn,

# validation\_split=0.2,

# callbacks=callbacks\_cnn)

model\_gru.compile(optimizer='adam',

loss='categorical\_crossentropy',

metrics=['acc'])

# history\_gru = model\_gru.fit(X\_train, y\_train,

# epochs=epochs\_gru,

# batch\_size=batch\_size\_gru,

# validation\_split=0.2,

# callbacks=callbacks\_gru)

model\_lstm.compile(optimizer='adam',

loss='categorical\_crossentropy',

metrics=['acc'])

# history\_lstm = model\_lstm.fit(X\_train, y\_train,

# epochs=epochs\_lstm,

# batch\_size=batch\_size\_lstm,

# validation\_split=0.2,

# callbacks=callbacks\_lstm)

acc = history\_ann.history['acc']

val\_acc = history\_ann.history['val\_acc']

loss = history\_ann.history['loss']

val\_loss = history\_ann.history['val\_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')

plt.plot(epochs, val\_acc, 'b', label='Validation acc')

plt.title('Training and validation accuracy')

plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.legend()

plt.show()

acc = history\_cnn.history['acc']

val\_acc = history\_cnn.history['val\_acc']

loss = history\_cnn.history['loss']

val\_loss = history\_cnn.history['val\_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')

plt.plot(epochs, val\_acc, 'b', label='Validation acc')

plt.title('Training and validation accuracy')

plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.legend()

plt.show()

acc = history\_lstm.history['acc']

val\_acc = history\_lstm.history['val\_acc']

loss = history\_lstm.history['loss']

val\_loss = history\_lstm.history['val\_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')

plt.plot(epochs, val\_acc, 'b', label='Validation acc')

plt.title('Training and validation accuracy')

plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.legend()

plt.show()

acc = history\_gru.history['acc']

val\_acc = history\_gru.history['val\_acc']

loss = history\_gru.history['loss']

val\_loss = history\_gru.history['val\_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, acc, 'bo', label='Training acc')

plt.plot(epochs, val\_acc, 'b', label='Validation acc')

plt.title('Training and validation accuracy')

plt.legend()

plt.figure()

plt.plot(epochs, loss, 'bo', label='Training loss')

plt.plot(epochs, val\_loss, 'b', label='Validation loss')

plt.title('Training and validation loss')

plt.legend()

plt.show()

model\_ann.load\_weights(output\_dir\_ann + 'best\_model.h5')

model\_cnn.load\_weights(output\_dir\_cnn + 'best\_model.h5')

model\_lstm.load\_weights(output\_dir\_lstm + 'best\_model.h5')

model\_gru.load\_weights(output\_dir\_gru + 'best\_model.h5')

model\_ann.evaluate(X\_test, y\_test)

model\_ann.evaluate(X\_test\_test, y\_test\_test)

model\_cnn.evaluate(X\_test, y\_test)

model\_cnn.evaluate(X\_test\_test, y\_test\_test)

model\_lstm.evaluate(X\_test, y\_test)

model\_lstm.evaluate(X\_test\_test, y\_test\_test)

model\_gru.evaluate(X\_test, y\_test)

model\_gru.evaluate(X\_test\_test, y\_test\_test)

predicted\_ann = model\_ann.predict(X\_test)

predicted\_cnn = model\_cnn.predict(X\_test)

predicted\_gru = model\_gru.predict(X\_test)

predicted\_lstm = model\_lstm.predict(X\_test)

GENRES = {'detective': 0,

'fantasy': 1,

'sci-fi': 2

}

genres\_list = ['detective', 'fantasy', 'sci-fi']

genres\_ann = []

for i in predicted\_ann[:5]:

genres\_ann.append(genres\_list[i.argmax()])

genres\_cnn = []

for i in predicted\_cnn[:5]:

genres\_cnn.append(genres\_list[i.argmax()])

genres\_lstm = []

for i in predicted\_lstm[:5]:

genres\_lstm.append(genres\_list[i.argmax()])

genres\_gru = []

for i in predicted\_gru[:5]:

genres\_gru.append(genres\_list[i.argmax()])