Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт

Кафедра Информатики
кафедра

ОТЧЕТ О ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ
НАУЧНО-ИССЛЕДОВАТЕЛЬСКАЯ РАБОТА

Кафедра “Информатика”
место прохождения практики

Генерация статей из Википедии
тема

Руководитель _____ А. Н. Пупков
Подпись, дата — Инициалы, Фамилия

Студент КИ19-17/1Б, №031939174 _____ А. К. Никитин
Номер группы, зачетной книжки — Подпись, дата — Инициалы, Фамилия

Красноярск 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Развитие компьютерных технологий не перестает удивлять. Еще 40 лет назад компьютеры были лишь громоздким научным инструментом, способным на простые вычисления и алгоритмы, являющийся лишь подспорьем в руках ученых и исследователей; однако теперь информационные технологии стали настолько распространены и вездесущи, что тяжело представить мир без них. И главным их достижением я считаю многофункциональность – разнообразие сфер, где они могут быть применены. Значительно улучшив вычислительные мощности, они смогли выйти за пределы простого инструмента и даже состязаться с человеком во многих отраслях. Ярким примером этого являются нейронные сети.

Мало сказать, что нейросети привлекли к себе внимание общественности – они произвели фурор. Их возможности поражают. Распознавание и синтез речи, анализ данных, прогнозирование, генерация изображение и многое другое. Они могут даже состязаться с людьми в творчестве – создании осмысленного текста. В нашей работе мы постарались воссоздать именно такую нейросеть: нейросеть, генерирующую статьи Википедии.

ОСНОВНАЯ ЧАСТЬ

1 Теоретическая часть

1.1 Теория нейронных сетей

1.1.1 Что такое нейросети?

Нейронные сети — одно из направлений в разработке систем искусственного интеллекта. Нейросети пытаются максимально близко смоделировать работу человеческой нервной системы — а именно, её способности к обучению и исправлению ошибок. В этом состоит главная особенность любой нейронной сети — она способна самостоятельно обучаться и действовать на основании предыдущего опыта, с каждым разом делая всё меньше ошибок.

Нейросеть имитирует не только деятельность, но и структуру нервной системы человека. Такая сеть состоит из большого числа отдельных вычислительных элементов («нейронов»). В большинстве случаев каждый «нейрон» относится к определённому слою сети. Нейронная сеть включает в себя несколько слоёв нейронов, каждый из которых отвечает за распознавание конкретного критерия: формы, цвета, размера, текстуры, звука, громкости и т.д. Параметры каждого «нейрона» могут изменяться в зависимости от результатов, полученных на предыдущих наборах входных данных, изменяя таким образом и порядок работы всей системы.

1.1.2 Нейроны и синапсы

Нейроны являются структурной единицей любой нейронной сети, из их множества, упорядоченного некоторым образом в слои, а в конечном счете – во всю сеть, состоят все сети.

Нейрон — это простейшая вычислительная единица, которая получает информацию, производит над ней простые вычисления и передает ее дальше.

Иными словами, у каждого нейрона есть 2 основных параметра: входные данные (input data) и выходные данные (output data).

Все нейроны функционируют примерно одинаковым образом, однако бывают некоторые частные случаи нейронов, выполняющих специфические функции. Всего существует три типа нейронов:

1) входной – слой нейронов, получающий информацию (синий цвет на рисунке 1);

2) скрытый – некоторое количество слоев, обрабатывающих информацию (красный цвет на рисунке 1);

3) выходной – слой нейронов, представляющий результаты вычислений (зеленый цвет на рисунок 1).

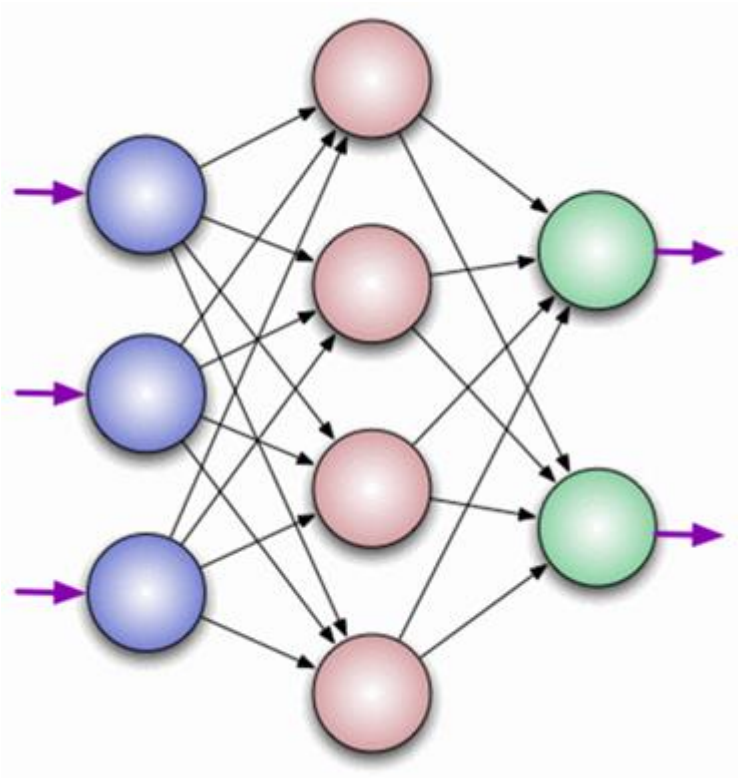

Рисунок 1 – Основные типы нейронов

Помимо самих нейронов существуют синапсы. Синапс – связь, соединяющая выход одного нейрона со входом другого. Во время прохождения сигнала через синапс сигнал может усиливаться или ослабевать. Параметром синапса является вес (некоторый коэффициент, может быть любым вещественным числом), из-за которого информация, передающаяся от одного

нейрона к другому, может изменяться. При помощи весов входная информация проходит обработку — и после мы получаем результат.

1.1.3 Классификация нейронных сетей

Нейронные сети могут быть проклассифицированы следующим образом (рисунок 2).

Рисунок 2 – Классификация нейронных сетей

По характеру обучения:

а) с учителем;

б) без учителя.

Обучение с учителем предполагает, что для каждого входного вектора существует целевой вектор, представляющий собой требуемый выход. Вместе они называются обучающей парой. Обычно сеть обучается на некотором числе таких обучающих пар.

Обучение без учителя не нуждается в целевом векторе для выходов и, следовательно, не требует сравнения с предопределенными идеальными ответами. Обучающее множество состоит лишь из входных векторов.

По настройке весов:

а) сети с фиксированными связями – весовые коэффициенты нейронной сети выбираются сразу, исходя из условий задачи;

б) сети с динамическими связями – для них в процессе обучения происходит настройка синаптических весов.

По типу входной информации:

а) аналоговая – входная информация представлена в форме действительных чисел;

б) двоичная – вся входная информация в таких сетях представляется в виде нулей и единиц.

По модели нейронной сети:

а) сети прямого распространения – все связи направлены строго от входных нейронов к выходным;

б) реккурентные нейронные сети – сигнал с выходных нейронов или нейронов скрытого слоя частично передается обратно на входы нейронов входного слоя;

в) радиально базисные функции – вид нейронной сети, имеющий скрытый слой из радиальных элементов и выходной слой из линейных элементов; сети этого типа довольно компактны и быстро обучаются;

г) самоорганизующиеся карты или сети Кохонена – сети такого класса способны выявлять новизну во входных данных и имеет всего два слоя: входной и выходной, составленный из радиальных элементов.

1.2 Обработка естественного языка

Обработка естественного языка (Natural language processing, сокращенно NLP) — область, находящаяся на пересечении computer science, искусственного интеллекта и лингвистики. Цель заключается в обработке и “понимании”

программой естественного (человеческого) языка для перевода текста и ответа на вопросы.

С развитием голосовых интерфейсов и чат-ботов, NLP стала одной из самых важных технологий искусственного интеллекта. Она используется для:

а) умного поиска;

б) автоматического перевода;

в) анализа настроений рецензий;

г) распознавания речи и чат-ботов;

д) голосовых помощников.

Таким образом, обработка естественного языка помогает компьютерам общаться с людьми на их родном языке и масштабировать другие языковые задачи.

1.3 Нейросеть GPT-2

GPT-2 – это алгоритм обработки естественного языка, реализованный через нейронную сеть, созданный OpenAI в феврале 2019 года. GPT-2 способна переводить текст, отвечать на вопросы, резюмировать отрывки и генерировать осмысленный текст.

Архитектура GPT реализует глубокую нейронную сеть и основана на архитектуре трансформер. Механизмы внимания позволяют модели выборочно фокусироваться на сегментах входного текста, которые, по ее мнению, являются наиболее актуальными. Эта модель позволяет значительно увеличить распараллеливание и превосходит предыдущие тесты для моделей на основе RNN / CNN / LSTM.

GPT-2 имеет 1,5 миллиарда параметров против обычных 100-300 млн. За В 2020 году компанией OpenAI была выпущена третья версия языковой модели – GPT-3, имеющая уже 175 миллиардов параметров.

2 Практическая часть

2.1 Получение данных датасета Википедии

2.1.1 Использование готового датасета

Данный вариант предполагает поиск базы данных среди интернет-ресурсов. Нами был выбран ресурс dumps.wikimedia.org, содержащий копии данных всех вики-ресурсов на разных языках. Мы выбрали русскоязычную версию базы данных в формате XML, не содержащую историю изменений.

Далее в проекте мы использовали именно этот датасет. Его несомненным преимуществом мы посчитали язык разметки Википедии, которому будет подражать нейросеть при обучении, что позволит генерировать более «вики-подобный текст», однако рассматривался и другой вариант получения сырых данных для нейронной сети.

2.1.2 Парсинг HTML

Другим вариантом является получение данных из непосредственно html-страниц статей Википедии. Недостатками этого метода являются малая скорость получения данных и сложность обработки данных, однако плюсом является получение «чистого» текста, что может быть в ряде случаев полезнее текста с вики-разметкой.

Для получения и обработки html-страницы был выбран язык Python и библиотека BeautifulSoup4. Ниже представлен листинг с парсером.

Листинг 1 – Парсер HTML

```
from urllib.request import urlopen
from bs4 import BeautifulSoup, Comment, Tag
import re


def get_html():
    """
    Получает HTML-документ Википедии
    :return: HTML-текст формата utf-8
```

Продолжение листинга 1

```
    """
    with
urlopen('https://ru.wikipedia.org/wiki/%D0%A1%D0%BB%D1%83%D0%B6%D0%B5%D0%B1%D0%B
D%D0%B0%D1%8F:%D0%A1%D0%BB%D1%83%D1%87%D0%B0%D0%B9%D0%BD%D0%B0%D1%8F_%D1%81%D1%8
2%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0') as fp:
        bytes_html = fp.read()
        raw_html = bytes_html.decode('utf8')
    return raw_html


def check_h3(soup: BeautifulSoup):
    """
    Проверяет, есть ил в статье вложенные заголовки
    :param soup: HTML-текст
    :return: True или False
    """
    content = soup.find("div", class_="mw-parser-output")
    if content.find("h3"):
        return True
    return False


def delete_trash(soup: BeautifulSoup):
    """
    Оставляет только необходимую информацию в html-документе сайта Wikipedia
    :param soup: необработанный HTML-документ Wikipedia
    :return: обработанный HTML-документ с содержанием статьи и заголовками
Wikipedia
    """
    content = soup.find("div", class_="mw-parser-output")

    tags_with_info = re.findall('<.*>', str(content))
    short_tags_list    =    set([full_tag.replace('<',    '').replace('>',    '
').replace('/', '').split(' ')[0]
                            for full_tag in tags_with_info])
    useless_tags = short_tags_list - {'p', 'h2', 'span', 'i', 'a', 'b'}

    # Костыль для удаления тега стиля, который иногда проскакивает
    for elem in content.findAll("style"):
        elem.decompose()
```

Продолжение листинга 1

```
    # Избавление от поля [править | править код] у заголовков
    for elem in content.findAll("span", class_='mw-editsection'):
        elem.decompose()

    # Удаление тего и содержимого тегов ненужных для анализа данных
    for tag in useless_tags:
        for elem in content.findAll(tag):
            elem.decompose()

    # Удаление костылей Википедии в заголовках
    for header in content.findAll("h2"):
        header.span.unwrap()
        header.span.unwrap()

    # Удаление комментариев в HTML-коде
    for child in content.children:
        if isinstance(child, Comment):
            child.extract()

    return content


def correct_text(wiki_text: str):
    """
    Исправляет неточности текста, связанных с парсингом HTML-документа
    :param wiki_text: параграф текста Википедии
    :return: исправленный текст
    """
    wiki_text = wiki_text.replace(u'\xa0', u' ')
    wiki_text = re.sub(r'\[\d]', '', wiki_text)
    return wiki_text


def delete_tags(html_text: str):
    """
    Удаляет все теги в HTML-тексте
    :param html_text: HTML-текст с тегами
    :return: простой текст без HTML-тегов
    """
    return re.sub('<.*?>', '', html_text)
```

Продолжение листинга 1

```
def delete_points(string: str):
    return string.replace('.', '')


def separate(content: Tag):
    """
    Разделяет HTML-текст с заголовками и параграфами на заголовки и параграфы, а затем формирует их в словарь
    :param content: HTML-текст для преобразования
    :return: словарь типа {Заголовок: Содержание заголовка, ...}
    """
    headers = ["Базовое описание"] + list(map(lambda x: x.get_text(), content.findAll("h2")))
    headers = list(map(delete_points, headers))
    tagged_headers = list(map(lambda h: f'<h2>{h}</h2>', headers))

    tagged_paragraphs = re.split('|'.join(map(re.escape, tagged_headers)), correct_text(str(content)))
    paragraphs = list(map(delete_tags, tagged_paragraphs))
    paragraphs = list(map(lambda x: x.strip('\n'), paragraphs))

    text_with_headers = {}
    for header, paragraph in zip(headers, paragraphs):
        text_with_headers[header] = paragraph

    return text_with_headers


def form_row():
    """
    Формирует строчку для записи в базу данных с информацией страницы Википедии заданого названия
    :param title: Название статьи
    :return: словарь типа {Название статьи: Название, Заголовок1: Содержание заголовка1, ...}
    """
    wiki_content = get_html()
    wiki_soup = BeautifulSoup(wiki_content, "html.parser")
    if check_h3(wiki_soup):
```

Продолжение листинга 1

```
        return None


    wiki_content = delete_trash(wiki_soup)
    wiki_article = {"Заголовок статьи": wiki_soup.find("h1").get_text()}
    wiki_article.update(separate(wiki_content))
    return wiki_article


if __name__ == '__main__':
    print(form_row())
```

2.2 Предобработка данных

Взглянув на вики-разметку, становится очевидно, что большое количество информации нейронной сети воспроизвести не удастся. Примерами такой информации являются таблицы, изображения, ссылки и прочие элементы, которые лишь ухудшат воспроизводимый нейронной сетью текст.

Преобразованные под обучение статьи было решено хранить в базе данных MongoDB, развернутой на локальном сервере. На рисунке ниже представлена готовая база данных, содержащая 341000 статей.

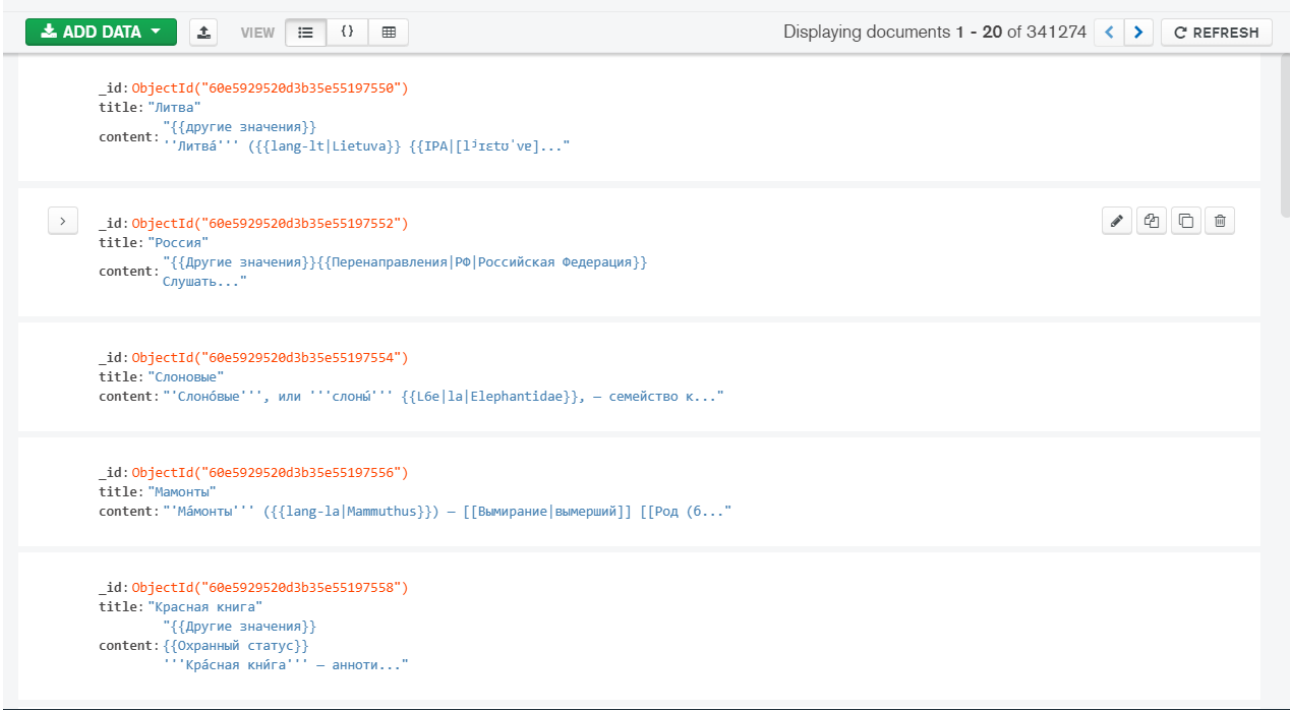

Рисунок 2 – База данных

Предобработка была выполнена на языке Python. Листинг программы представлен ниже.

Листинг 2 – Предобработка вики-разметки

```
import re
import pymongo

# Подключение к базе данных
client = pymongo.MongoClient("mongodb://127.0.0.1:27017")
database = client.wiki
articles_db: pymongo.collection.Collection = database.articles


def delete_tables(text: str):
    """
    Удаляет все таблицы из статьи
    :param text: вики-разметка с таблицами
    :return: вики-разметка без таблиц
    """
    text = text.replace('\n |', '\n|')

    # Таблицы формата {| ... |}
    while (start_index := text.find('{|')) != -1:
        end_index = text[start_index:].find('|}') + start_index + 2
        text = text.replace(text[start_index:end_index + 1], '')

    # Таблицы формата {{Категория | ... | ... }}
    cur_pos = 0
    max_iter_number = 0
    while (start_index := text[cur_pos:].find('{{')) != -1:

        # Экстренное прекращение, если поиск зациклился
        if max_iter_number > 5000:
            break
        max_iter_number += 1

        if text[start_index + cur_pos:].find('\n|') != -1 and \
                text[start_index + cur_pos:].find('\n|') < text[start_index +
cur_pos:].find('}}'):
            rows = text[start_index + cur_pos:].split('\n')
            end_index = start_index
            for i, row in enumerate(rows):
```

Продолжение листинга 2

```
                if len(rows) == i + 1 and row.endswith('}}'):
                    end_index += len(row) + 1
                    break

                if row.startswith('|') and not rows[i + 1].startswith('|') and
not rows[i + 1].startswith('*'):
                    if rows[i + 1].startswith('}}'):
                        end_index += len(row) + 5
                        break
                    elif row.endswith('}}'):
                        end_index += len(row) + 1
                        break
                    else:
                        return None
                end_index += len(row) + 1

            text = text.replace(text[start_index + cur_pos:end_index + cur_pos +
1], '')
        else:
            cur_pos += text[start_index + cur_pos:].find('}}') + start_index

    return text


def delete_title(title: str, text: str):
    """
    Удаляет заголовок вместе с содержанием внутри его
    :param title: название заголовка
    :param text: вики-разметка с ненужным заголовком
    :return: вики-разметка без заголовка
    """

    # Ссылки всегда идет последними, так что текст после них обрезается
    if title == 'Ссылки':
        text = text[:text.find('== Ссылки ==')]

    required_title = None
    for source_title in re.finditer(r'== (.*?) ==', text):
        if required_title is not None:
            text           =           text.replace(text[required_title.start():
source_title.start()], '')
```

Продолжение листинга 2

```
            required_title = None

        if source_title.group().strip('= ') == title:
            required_title = source_title

    return text


def delete_links(text: str):
    """
    Удаляет все гиперссылки в тексте
    :param text: вики-разметка с гиперссылками
    :return: вики-разметка без гиперссылок
    """
    while (url := re.search(r'http(?:s)?://\S+', text)) \
            is not None:
        text = text.replace(text[url.start(): url.end() + 1], '')

    return text


def delete_files(text: str):
    """
    Удаляет все элементы с файлами в тексте
    :param text: вики-разметка с файлами
    :return: вики-разметка без файлов
    """
    return re.sub(r'\[\[([фФ]айл|[fF]ile).*\]\]', '', text)


def delete_empty_titles(text: str):
    """
    Удаляет все заголовки, содержимое которого по тем или иным причинам было
удалено, т.е. пустые заголовки
    :param text: вики-разметка с пустыми заголовкам
    :return: вики-разметка без пустых заголовков
    """

    text_tmp = re.sub(r'\n{2,}', '\n', text)
```

Продолжение листинга 2

```
    # Удаление "заголовков заголовка"
    for subtitle in re.finditer(r'; [А-Я].*', text_tmp):
        if text_tmp.endswith(subtitle.group()) or text_tmp[subtitle.end() + 1] in
(';', '='):
            text = text.replace(subtitle.group(), '')

    # Удаление заголовков 3 уровня
    for subtitle in re.finditer(r'==== (.*?) ====', text_tmp):
        if  text_tmp.endswith(subtitle.group())  or  text_tmp[subtitle.end()  +
1:subtitle.end() + 3] == '==':
            text = text.replace(subtitle.group(), '')

    # Удаление заголовков 2 уровня
    for subtitle in re.finditer(r'=== (.*?) ===', text_tmp):
        if  text_tmp.endswith(subtitle.group())  or  text_tmp[subtitle.end()  +
1:subtitle.end() + 3] == '==':
            text = text.replace(subtitle.group(), '')

    # Удаление заголовков 1 уровня
    for title in re.finditer(r'== (.*?) ==', text_tmp):
        if  text_tmp.endswith(title.group())  or  text_tmp[title.end()  +  1:
title.end() + 3] == '==' and text_tmp[
            title.end() + 3] != '=':
            text = text.replace(title.group(), '')

    return text


def delete_trash(article_text: str):
    """
    Удаление ненужной информации в тексте вики-разметке
    :param article_text: вики-текст с "мусорной" информацией
    :return: вики-текст, готовый к обработке нейросетью
    """
    # Удаление перенаправлений и прочих "однострочных" статей
    if              article_text.lower().startswith('#redirect')              or
article_text.lower().startswith('mediawiki') or \
            article_text.lower().startswith('#перенаправление')              or
article_text.lower().startswith('[[категория'):
        return None
```

Продолжение листинга 2

```
    article_text = article_text.replace(u'\xa0', u' ')
    article_text = article_text.replace(u'\xd0', u' ')
    article_text = re.sub(r'<!--[\s\S]*?-->\n?', '', article_text)
    article_text = delete_tables(article_text)

    if article_text is None:
        return None

    article_text = re.sub(r'<ref(.*?)<\/ref>', '', article_text)
    article_text = re.sub(r'<ref(.*?)>', '', article_text)
    article_text = re.sub(r'<[^\/].*?>[\s\S]*?<\/.*?>', '', article_text)

    article_text = delete_title('Примечания', article_text)
    article_text = delete_title('Ссылки', article_text)

    article_text = delete_links(article_text)
    article_text = delete_files(article_text)

    article_text = re.sub(r'\{\{[дД]ругие значения\|.*\}\}', '', article_text)
    article_text = re.sub(r'\{\{[rR]edirect\|.*\}\}', '', article_text)
    article_text = re.sub(r'\{\{[пП]еренаправлени[ея]\|.*\}\}', '', article_text)
    article_text = re.sub(r'\{\{[гГ]лавная\|.*\}\}', '', article_text)
    article_text = re.sub(r'\{\{[оО]сновная статья\|.*\}\}', '', article_text)
    article_text = re.sub(r'\{\{[оО]сновная\|.*\}\}', '', article_text)
    article_text = re.sub(r'\{\{[mM]ain\|.*\}\}', '', article_text)
    article_text = re.sub(r'\{\{[тТ]акже\|.*\}\}', '', article_text)
    article_text = re.sub(r'\{\{[sS]ee also\|.*\}\}', '', article_text)
    article_text = re.sub(r'\{\{[яЯ]корь\|.*\}\}', '', article_text)
    article_text = re.sub(r'\{\{[зЗ]начения\|.*\}\}', '', article_text)
    article_text = re.sub(r'\{\{[сС]лушать\|.*\}\}', '', article_text)
    article_text = re.sub(r'\{\{[дД]ругие значения\|.*\}\}', '', article_text)

    article_text = article_text.strip('\n *')

    article_text = delete_empty_titles(article_text)

    article_text = re.sub(r'\n{3,}', '\n', article_text)

    return article_text
```

Запись данных в базу данных MongoDB из исходников XML было выполнение на языке Python и библиотеки PyMongo. Ниже представлен листинг с программой записи в базу данных.

Листинг 3 – Запись обработанных статей в базу данных

```
from xml.etree import ElementTree as etree
from xml.etree.ElementTree import Element
import pymongo
from wiki_parser import delete_trash

client = pymongo.MongoClient("mongodb://127.0.0.1:27017")
database = client.wiki
articles_db: pymongo.collection.Collection = database.articles


def check_title(title: str):
    if                  title.lower().startswith('категория')                  or
title.lower().startswith('шаблон')\
            or                title.lower().startswith('файл')                or
title.lower().startswith('проект')\
            or              title.lower().startswith('mediawiki')              or
title.lower().startswith('портал')\
            or title.lower().startswith('википедия'):
        return False
    return True


def write_xml_to_db(path: str):
    title = None
    content = None
    for event, elem in etree.iterparse(path):
        elem: Element = elem
        if  elem.tag  ==  '{http://www.mediawiki.org/xml/export-0.10/}title'  and
elem.text is not None:
            title = elem.text
        if  elem.tag  ==  '{http://www.mediawiki.org/xml/export-0.10/}text'  and
elem.text is not None:
            try:
                content = delete_trash(elem.text)
            except Exception as e:
                continue
```

Продолжение листинга 3

```
        if title and content and check_title(title):
            print(title)
            articles_db.update_one({'title':   title},   {'$set':   {'content':
content}}, upsert=True)
            title = None
            content = None


if __name__ == '__main__':
    write_xml_to_db('wiki-articles3.xml')
```

2.3 Обучение нейросети

В качестве готовой модели для обучение нейросети была выбрана нейросеть GTP-2 и ее обученная на русских текстах модель, обученная Сбербанком, лежащая в открытом доступе на сайте https://github.com/sberbank-ai/ru-gpts. Была выбрана именно эта модель, так как, во-первых, она мощнее своих конкурентов (но, с другой стороны, и тяжелее), и во-вторых, у нее имеется хорошо обученная русская версия.

Дообучение нейронной сети было произведено на языке Python в среде разработки Jupyter Notebook. В качестве фреймворка нейросети была использована библиотека PyTorch и Transformers. Ниже на листинге представлен Python-код страницы Jupyter Notebook.

Листинг 4 – Обучение нейросети

```
#!/usr/bin/env python
# coding: utf-8

# ## Установка необходимых для обучения библиотек

# In[3]:


get_ipython().system('pip3 install transformers==3.5.0')
get_ipython().system('pip install tensorboard')
```

Продолжение листинга 4

```
# In[4]:


get_ipython().system('pip  install  https://download.pytorch.org/whl/cu101/torch-
1.4.0-cp38-cp38-win_amd64.whl')
get_ipython().system('pip                                                  install
https://download.pytorch.org/whl/cu101/torchvision-0.5.0-cp38-cp38-
win_amd64.whl')


# ## Обучение нейросети

# In[ ]:


get_ipython().system('SET PYTHONPATH=${PYTHONPATH}:/ru-gpts/')
get_ipython().system('python   ru-gpts/pretrain_transformers.py                --
output_dir=models/essays     --model_type=gpt2     --model_name_or_path=sberbank-
ai/rugpt3small_based_on_gpt2     --do_train     --train_data_file=train.txt     -
-do_eval     --eval_data_file=valid.txt     --per_gpu_train_batch_size 1     --
gradient_accumulation_steps 1     --num_train_epochs 5     --block_size 64     -
-overwrite_output_dir     --should_continue     --save_total_limit 5')


# ## Оценка нейросети

# In[2]:


get_ipython().system('python   ru-gpts/generate_transformers.py                --
model_type=gpt2     --model_name_or_path=models/essays/checkpoint-4000     --k=5
--p=0.95     --length=500     --repetition_penalty=5')


# ## Генерация текста

# In[6]:


from transformers import GPT2LMHeadModel, GPT2Tokenizer
```

Продолжение листинга 4

```
# In[7]:


tok = GPT2Tokenizer.from_pretrained("models/essays/checkpoint-25000")


# In[8]:


model = GPT2LMHeadModel.from_pretrained("models/essays/checkpoint-25000")


# In[9]:


model.cuda()
print()


# In[12]:


import ctypes
ctypes.cdll.LoadLibrary('caffe2_nvrtc.dll')


# In[10]:


text = "Текст для генерации"
inpt = tok.encode(text, return_tensors="pt")


# In[14]:


out = model.generate(inpt.cuda(), max_length=50, repetition_penalty=5.0,
do_sample=True, top_k=4
                     , top_p=0.95, temperature=1)
```

Продолжение листинга 4

```
# In[15]:


tok.decode(out[0])
```

2.4 Результаты работы нейросети

Ниже представлены примеры работы нейросети.

1. ""Нейронные сети"" — совокупность [[компьютерная сеть|сетевых сетей]], которая состоит из нескольких компьютеров и может быть использована для работы с ними.\n* \'\', {{lang-en2magic}}. На этом же языке можно использовать различные языки программирования (например, «язык ассемблера», или языка программиста)».

2. "'Язык программирования'" — язык [[язык ассемблера|программирования]], разработанный на основе языка C++. Основы архитектуры являются 'объектным наследием языков высокого уровня (C-, D) и более высокой производительностью в языке Си.

3. "'Королева Великобритании'" — королева Англии. Dougrey (1796) - герцогиня де Вант-Габриэль, супруга короля Эдуарда II [[Филипп I Август|Филпса Августа]]. После смерти её отец стал королём английского государства и был вынужден покинуть страны после его возвращения в Англию из ссылки на несколько месяцев; вскоре он вернулся во Францию: у него родился сын Эдуард V, граф Корнуоллский.

4. ""Революция"" — политическая революция в [[СССР]], произошедшая во второй половине 1980-х годов, на фоне которой были проведены некоторые мероприятия и политики.

ЗАКЛЮЧЕНИЕ

В рамках научно-исследовательской работы был изучены основы теории и работы нейросетей, были получены навыки в парсинге html-страниц, в использовании объектной базы данных, в работе с файлами расширения XML, в работе по обучении нейронной модели на языке Python.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1) СТО 4.2–07–2014 Система менеджмента качества. Организация учета и хранения документов. – Введ. 09.01.2014. – Красноярск : ИПК СФУ, 2014.

2) Бум нейросетей: Кто делает нейронные сети, зачем они нужны и сколько денег могут приносить [Электронный ресурс]: Информационный сайт // – Режим доступа: https://vc.ru/future/16843-neural-networks (дата обращения 08.07.2021).

3) Что такое нейронные сети и как они работают? Классификация искусственных нейросетей [Электронный ресурс]: Информационный сайт // – Режим доступа: https://mining-cryptocurrency.ru/nejronnye-seti/#i-4 (дата обращения 08.07.2021).

4) Нейронные сети, или Как обучить искусственный интеллект [Электронный ресурс]: Информационный сайт // – Режим доступа: http://internetinside.ru/neyronnye-seti-ili-kak-obuchit-iskuss/ (дата обращения 08.07.2021).

5) Классификация и типы нейронных сетей [Электронный ресурс]: Data Science // – Режим доступа: http://datascientist.one/class-type-nn/ (дата обращения 08.07.2021).

6) GPT-2 [Электронный ресурс]: Свободная энциклопедия // – Режим доступа: https://ru.xcv.wiki/wiki/GPT-2 (дата обращения 08.07.2021).

7) 5 методов обработки естественного языка, которые стремительно меняют мир вокруг нас [Электронный ресурс]: Информационный сайт // – Режим доступа: https://neurohive.io/ru/osnovy-data-science/5-metodov-v-nlp-kotorye-izmenjat-obshhenie-v-budushhem/ (дата обращения 08.07.2021).