МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

Национальный исследовательский ядерный университет «МИФИ»

Институт
интеллектуальных кибернетических систем

Кафедра № 22 «Кибернетика»

Направление подготовки 01.03.02 Прикладная математика и информатика

Пояснительная записка

к учебно-исследовательской работе студента на тему:

Разработка системы распознавания речи

Группа Б17-501

Студент _____ (подпись) Баранова Д.Д. (ФИО)

Руководитель _____ (0-5 баллов) _____ (подпись) Козин Р.Г. (ФИО)

Научный консультант _____ (0-5 баллов) _____ (подпись) _____ (ФИО)

Оценка руководителя _____ (0-15 баллов) Оценка консультанта _____ (0-15 баллов)

Итоговая оценка _____ (0-100 баллов) ECTS _____

Комиссия

Председатель _____ (подпись) _____ (ФИО)

_____ (подпись) _____ (ФИО)

_____ (подпись) _____ (ФИО)

_____ (подпись) _____ (ФИО)

Москва 2020

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Национальный исследовательский ядерный университет «МИФИ»

Институт интеллектуальных кибернетических систем

КАФЕДРА КИБЕРНЕТИКИ

Задание на УИР

Студенту гр.	Б17-501		Барановой Д.Д.
	(группа)		(фио)

ТЕМА УИР

Разработка системы распознавания речи

ЗАДАНИЕ

№ п/п	Содержание работы	Форма отчетности	Срок исполнения	Отметка о выполнении Дата, подпись рук.
1	**Аналитическая часть**			
1.1	Обзор особенностей речи, затрудняющий её распознавание	Подраздел ПЗ	15.09.2020	
1.2	Обзор методов и существующих подходов для распознавания речи	Подраздел ПЗ	30.09.2020	
1.3	Обзор существующих сервисов распознавания речи	Подраздел ПЗ	15.10.2020	
1.4	*Оформление расширенного содержания пояснительной записки (РСПЗ)*	Текст РСПЗ	30.10.2020	
2	**Теоретическая часть**			
2.1	Постановка задачи распознавания речи	Подраздел ПЗ	10.11.2020	
2.2	Описание метода работы системы распознавания	Описание метода	15.11.2020	
2.3	Модель нейронной сети, используемая для распознавания речи	Описание модели	20.11.2020	
2.4	Метрики качества нейросетевой модели	Подраздел ПЗ	30.11.2020	
3	**Инженерная часть**			
3.1	Выбор языка и вспомогательных модулей для программной реализации системы	Выбранный язык	01.11.2020	
3.2	Программная реализация моделей и	Исходный код	05.11.2020	

	алгоритмов			
4	**Технологическая и практическая часть**			
4.1	Описание обучающих выборок, объема и особенностей	Готовый датасет	06.11.2020	
4.2	Экспериментальное исследование влияния архитектуры нейросетевой модели на точность распознавания	Подраздел ПЗ	10.12.2020	
4.3	Экспериментальное сравнение качества работы систем, использующих разные методы распознавания	Подраздел ПЗ	20.12.2020	
4.4	Выводы по результатам экспериментальных исследований	Подраздел ПЗ	01.01.2021	
	Оформление пояснительной записки (ПЗ) и иллюстративного материала для доклада.	Текст ПЗ, презентация	30.01.2021	

ЛИТЕРАТУРА

•	И.Б. Тампель, А.А. Карпов Автоматическое распознавание речи ИТМО, 2016
•	Муаммар Аль-Шедиват Открытые проблемы в области распознавания речи. Лекция в Яндексе, 2017 URL:https://habr.com/ru/company/yandex/blog/337572/
•	M.A.Anusuya S.K.Katti Speech Recognition by Machine: A Review, 2009 URL:https://arxiv.org/pdf/1001.2267.pdf
•	Neural Networks used for Speech Recognition Wouter Gevaert, Georgi Tsenov, Valeri Mladenov, Senior Member, IEEE
•	А.В. Фролов, Г.В. Фролов Синтез и распознавание речи. Современные решения URL:https://frolov-lib.ru/books/hi/ch05.html
•	Andrew Gibiansky Speech Recognition with Neural Networks, 2014 URL:https://andrew.gibiansky.com/blog/machine-learning/speech-recognition-neural-networks/
•	Joe Tebelskis Speech Recognition using Neural Networks, May 1995 URL:https://robotics.bstu.by/mwiki/images/0/0f/(Brain_Study)_Speech_Recognition_using_Neural_Networks.pdf

Дата выдачи задания:		Руководитель			Козин Р.Г.
					(ФИО)
« 1 » сентября 2020 г.		Студент			Баранова Д.Д.
					(ФИО)

Реферат

Пояснительная записка содержит 60 страниц, 30 рисунков, 2 таблицы, 15 формул.

Количество использованных источников – 27.

Ключевые слова: распознавание речи, рекуррентная нейронная сеть, акустическая модель, языковая модель, расстояние Левенштейна.

Целью данной работы является разработка системы обработки и распознавания английского текста из аудиофайла.

В первом разделе рассматриваются различные существующие подходы к распознаванию речи, классификация особенностей речи, затрудняющих распознавание, использующиеся методы и алгоритмы для распознавания, существующие библиотеки для работы с аудио файлами и методами машинного обучения, а также обзор и сравнение существующих популярных подходов для распознавания речи.

Во втором разделе приводится описание разработанного метода распознавания речи, этапы обработки и распознавания, модель используемой нейронной сети.

В третьем разделе приводится описание разработки системы, реализации программных компонентов, выбор языка и вспомогательных модулей для разработки.

В четвёртом разделе приводятся результаты экспериментальных исследований реализованных методов распознавания речи, оценивается влияние архитектуры нейросети на точность распознавания, определяется наилучшая архитектура системы распознавания.

В заключении подводятся итоги проведённой работы с кратким описанием результатов по каждому разделу.

Содержание

Введение

Распознавание речи (или компьютерное распознавание речи) - это процесс преобразования речевого сигнала в последовательность слов с помощью автоматизированной системы.

Исследования в области обработки речи и коммуникации по большей части развиваются из-за желания людей построить механические модели, имитирующие человеческие способности общения. Речь - это наиболее естественная форма человеческого общения, поэтому работа с речью была одной из самых актуальных областей обработки сигналов. Технология распознавания речи может позволить компьютеру определять смысл запроса, улавливать поступающие голосовые команды на основе выявления ключевых слов, следовать им и помогать человеку. Основная цель области распознавания речи - разработка методов и систем для ввода речи в машину. Поэтому исследования в области автоматического распознавания речи машинами привлекли и продолжают привлекать большое внимание, а также находят свое широкое применение в задачах, связанных с работой интерфейса «человек-машина» - таких как: автоматическая обработка вызовов в телефонных сетях, информационные системы на основе запросов, которые предоставляют доступ к актуальной информации о поездках, котировках цен на акции, погоде и т. д., машинный перевод и транскрипция речи, ввод данных (голосовая диктовка), выполнение поступивших на вход команд (например, бронирование билетов), помощь людям с ограниченными возможностями (слепым) и т. д. Технология распознавания речи все чаще использовалась в телефонных сетях для автоматизации, а также для улучшения услуг оператора.

Несмотря на то, что были достигнуты многие технологические успехи, остается еще много исследовательских проблем, которые необходимо решить. Разработка системы для распознавания речи позволит автоматизировать и ускорить решение многих повседневных задач.

Целью данной работы является построение системы, выполняющей автоматическое распознавание речи в аудиосигнале. В рамках работы решаются следующие задачи:

- Получение размеченной обучающей выборки.
- Изучение и разработка методов распознавания речи.
- Выбор и разработка оптимальной архитектуры нейросети.
- Изучение и проектирование оптимальной архитектуры системы.

В первом разделе приводится описание и сравнительный анализ различных уже существующих подходов к распознаванию речи.

Во втором разделе приводится описание метода распознавания речи и исследование метрик качества разработанной модели.

В третьем разделе приводятся требования и проект разрабатываемой системы, описание реализованных функций и модулей.

В четвёртом разделе приводятся результаты экспериментальных исследований влияния способа представления данных на качество работы нейронной сети, влияния параметров архитектуры выбранной модели нейронной сети и определения оптимального метода решения поставленной задачи, в результате чего проводится проектирование архитектуры системы для высокого качества работы.

В заключении подводятся итоги проведённой работы с кратким описанием результатов по каждому разделу.

Раздел 1. Анализ существующих подходов к распознаванию речи

Сейчас существует несколько платных сервисов для распознавания речи, спроектированных на основе использования различных методов распознавания речи в аудиозаписи. Успешность и скорость их распознавания во многом зависит от дикции автора, качества записи аудиофайла, поданного на вход, фонового шума, а также от качества работы спроектированной системы распознавания, и, как следствие, от выбранного метода.

Разнообразие особенностей речи, а также реализуемых подходов к решению задачи распознавания ведут к многообразию существующих программных решений, борющихся за лидерство по качеству работы, но все еще имеющих свои индивидуальные достоинства и недостатки.

1.1 Обзор и классификация особенностей речи, затрудняющий её распознавание

Несмотря на хороший прогресс, достигнутый в этой области, процесс распознавания речи по-прежнему сталкивается с множеством проблем, большинство из которых связано с тем, что речь может сильно отличаться. К наиболее распространенным особенностям речи, вызывающим ее вариативность, относятся:

- Вариация говорящего - одно и то же слово по-разному произносится разными людьми в зависимости от возраста, пола, анатомических особенностей, скорости речи, эмоционального состояния говорящего и диалектных вариаций.
- Фоновый шум: окружающая среда может добавлять шум к сигналу. Даже сам говорящий может добавить шум своей манерой речи.
- Произношение слова (влияние интонации и ударения на слоги).
- Характер речи: непрерывная речь представляет из себя поток трудно сегментирующихся сливающихся звуков, между словами редко бывает перерыв. Из-за этого очень сложно определить отдельные слова.
- Неограниченный словарный запас – повседневная речь иногда содержит слова из других языков.
- Другие внешние факторы: положение микрофона по отношению к говорящему, направление и настройка микрофона и многие другие.

1.2 Обзор методов и существующих подходов для распознавания речи

Было проведено исследование [8] и разработка классификации существующих подходов распознавания речи. В основном существует три подхода к распознаванию речи:

- Акустически-фонетический подход
- Подход сопоставления с образцом (паттерном)
- Подход искусственного интеллекта

1.2.1 Акустически-фонетический подход

Самые ранние подходы к распознаванию речи были основаны на поиске звуков речи и присвоении им соответствующих ярлыков. Это является основой акустического фонетического подхода, который строится на утверждениях, что в разговорной речи существуют конечные, отличительные фонетические единицы (фонемы), и что эти единицы в целом характеризуются набором акустических свойств, которые проявляются в речевой сигнал с течением времени.

Несмотря на то, что акустические свойства фонетических единиц сильно различаются как со скоростью речи, так и с изменением благодаря сочетанию с соседними звуками (так называемый эффект коартикуляции [10]), в акустико-фонетическом подходе предполагается, что правила, регулирующие изменчивость, просты и могут легко быть заложены в машине в виде правил. Первым шагом в акустическом фонетическом подходе является анализ речи в сочетании с обнаружением признаков, которые описывают широкие акустические свойства различных фонетических единиц. Следующим шагом является этап сегментации и маркировки, на котором речевой сигнал сегментируется на стабильные акустические области, за которым следует прикрепление одной или нескольких фонетических меток к каждой сегментированной области, в результате чего получается характеристика речи в виде решетки фонем.

На последнем этапе этого подхода происходит определение допустимого слова (или строки слов) из фонетических последовательностей меток, полученных в результате сегментации. В процессе проверки для более корректного декодирования используются лингвистические ограничения задачи (т.е. словарный запас, синтаксис и др. семантические правила). Акустически-фонетический подход не получил широкого распространения в большинстве коммерческих приложений.

1.2.2 Подход сопоставления с образцом (паттерном)

Подход сопоставления с образцом, описанный в работах [11][12], включает два важных этапа, а именно: обучение образцу и распознавание на основе сравнения образцов.

Существенной особенностью этого подхода является то, что он использует четкую математическую модель, устанавливает метод представления речевых паттернов для их будущего сравнения из набора размеченной обучающей выборки с помощью формального обучающего алгоритма (см рисунок ниже).

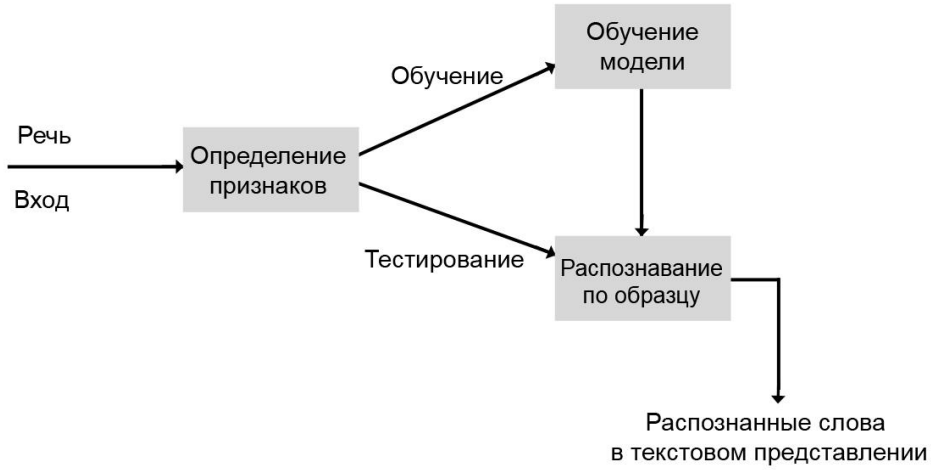

Рис.1 Представление процесса распознавания речи в виде блочной диаграммы

Представление паттерна может также быть в форме статистической модели (например, скрытые Марковские модели, HMM) и может применяться к звуку (фонеме), слову или фразе. На этапе сравнения шаблонов проводится прямое сравнение между неизвестными речами (речью, которую нужно распознать) с каждым возможным шаблоном, изученным на этапе обучения, чтобы провести распознавание в соответствии с точностью совпадения шаблонов.

Подход сопоставления с образцом стал преобладающим методом распознавания речи за последние шесть десятилетий. Самыми популярными методами являются скрытые Марковские модели и N-граммы. Это вероятностные модели для работы с неопределенной или неполной информацией. В распознавании речи неопределенность и неполнота возникают из многих источников; например, запутанные звуки, вариативность говорящего, контекстные эффекты и слова омофоны.

1.2.2.1 Скрытое Марковское моделирование (HMM)

До сих пор наиболее успешный [9] и наиболее часто используемый метод для распознавания речи - это математическая модель, полученная на основе модели Маркова.

Скрытой Марковской моделью (СММ) называется модель, состоящая из N состояний, в каждом из которых некоторая система может принимать одно из M значений какого-либо параметра (см рисунок ниже). Вероятности переходов между состояниями задается матрицей

вероятностей A={a_{ij}}, где a_{ij} – вероятность перехода из i-го в j-е состояние. Вероятности выпадения каждого из M значений параметра (изображены желтыми квадратами) в каждом из N состояний (изображены голубыми кружками) задается вектором B={$b_j(k)$}, где $b_j(k)$ – вероятность выпадения k-го значения параметра в j-м состоянии. Вероятность наступления начального состояния задается вектором $\pi=\{\pi_i\}$, где π_i – вероятность того, что в начальный момент система окажется в i-м состоянии. Таким образом, скрытой Марковской моделью называется тройка $\lambda = \{A, B, \pi\}$ [13].

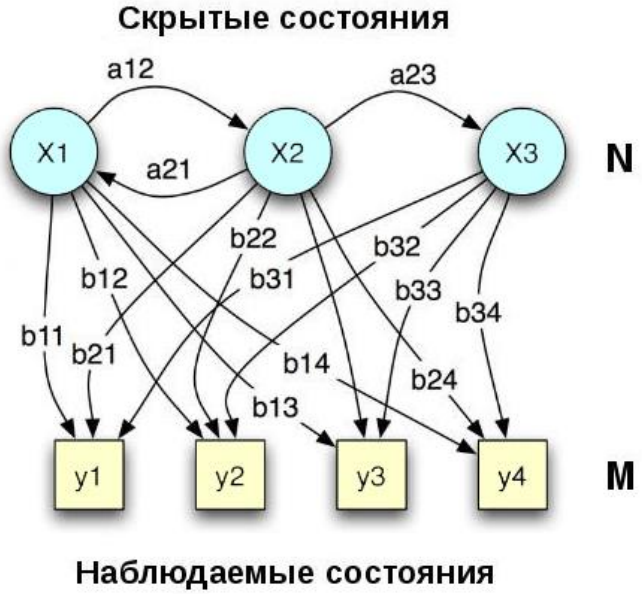

Рис.2 Общая схема устройства скрытой Марковской модели

Использование скрытых Марковских моделей для распознавания речи основано на трех приближениях [14]:

1) Речь может быть разбита на фрагменты (соответствующие состояниям в СММ), внутри которых речевой сигнал может рассматриваться как стационарный. параметры речи в пределах каждого фрагмента считаются постоянными.
2) Переход между этими состояниями осуществляется мгновенно.
3) Вероятность каждого фрагмента зависит только от текущего состояния модели и не зависит от предыдущих состояний.

Модель называется «скрытой», так как нас, как правило, не интересует конкретная последовательность состояний, в которой пребывает система. Мы подаем на вход системы последовательности типа $O = \{o_1, o_2, \ldots, o_\tau\}$, где каждое o_i – значение параметра (одно из M), принимаемое в i-й момент времени, а на выходе ожидаем модель $\lambda = \{A, B, \pi\}$ с максимальной вероятностью генерирующую такую последовательность. Система выступает

как “черный ящик”, в котором скрыты действительные состояния системы, а связанная с ней модель заслуживает названия скрытой.

Более формально одну из задач скрытых Марковских моделей можно представить так. Дана последовательность наблюдений $O = \{o_1, o_2, \ldots, o_\tau\}$ и модель $\lambda = \{A, B, \pi\}$. Необходимо вычислить вероятность появления указанной последовательности для данной модели. Если, например, состояния модели соответствуют отрезкам времени, в которые снимаются параметры речевого сигнала, и в каждом из этих состояний (отрезков) некие параметры речевого сигнала принимают некоторые значения, которые мы представляем в виде $O = \{o_1, o_2, \ldots, o_\tau\}$, то решив задачу отыскания вероятности появления этой последовательности для каждой из имеющихся у нас моделей $\lambda = \{A, B, \pi\}$, соответствующих, например, фонемам (звукам речи) или словам, мы можем выбрать ту из фонем (слов), которая(ое) в наибольшей степени соответствует исходному отрезку речевого сигнала. А это и значит распознать речевую единицу (фонему или слово).

Распознавание речи использует описанную выше Марковскую модель. Речь разбита на мельчайшие слышимые сущности (не только гласные и согласные, но и сопряженные сложные звуки). Все эти сущности представлены как состояния в Марковской модели. Когда слово входит в скрытую Марковскую модель, оно сравнивается с наиболее подходящей моделью (сущностью). В зависимости от уровня, на котором выбирается проведение распознавание, каждая такая модель может обозначать один из звуков русского языка (или отсутствие звука - одна из моделей), или одно из изученных слов. По вероятностям перехода существует переход из одного состояния в другое. Например, вероятность слова, начинающегося с «ы», почти равна нулю. Состояние также может иметь переход в собственное состояние, если звук повторяется.

Марковские модели неплохо работают в шумной среде, потому что каждый звуковой объект рассматривается отдельно. Если звуковой объект теряется в шуме, модель может угадать этот объект на основе вероятности перехода от одного звукового объекта к другому.

1.2.2.2 Использование N-грамм

N-грамма – это еще одна модель представления данных, используемая для распознавания речи [15]. N-граммная модель рассчитывает вероятность последнего слова (фонемы) N-граммы при условии, что известны все предыдущие. При использовании этого подхода предполагается, что появление каждого слова (фонемы) зависит только от предыдущих слов (фонем), причем значение N указывает, от скольких предыдущих слов оно зависит.

Рассмотрим пример биграммной модели, целью построения которой является определение вероятности употребления заданной фразы «мой дядя самых честных правил». Эту вероятность можно задать формально как вероятность возникновения именно этой последовательности слов в каком-то тексте. Вероятность фразы можно вычислить как произведение вероятностей каждого из слов этой фразы:

$$P = P(\text{мой}) * P(\text{дядя}|\text{мой}) * P(\text{самых}|\text{мой дядя}) * P(\text{честных}|\text{мой дядя самых}) * P(\text{правил}|\text{мой дядя самых честных}) \quad (1)$$

Чтобы определить $P(\text{мой})$, нужно посчитать, сколько раз это слово встретилось в тексте, и поделить это значение на общее число слов. Рассчитать вероятность $P(\text{правил}|\text{мой дядя самых честных})$ сложнее.

Чтобы упростить эту задачу, примем, что вероятность слова в тексте зависит только от предыдущего слова. Тогда наша формула для расчета фразы примет следующий вид:

$$P = P(\text{мой}) * P(\text{дядя}|\text{мой}) * P(\text{самых}|\text{дядя}) * P(\text{честных}|\text{самых}) * P(\text{правил}|\text{честных}) \quad (2)$$

Рассчитать условную вероятность $P(\text{дядя}|\text{мой})$ несложно. Для этого считаем количество пар 'мой дядя', и делим на количество в тексте слова 'мой'.

В результате, если мы посчитаем все пары слов в некотором тексте, мы сможем вычислить вероятность произвольной фразы. Этот набор рассчитанных вероятностей и будет биграммной моделью.

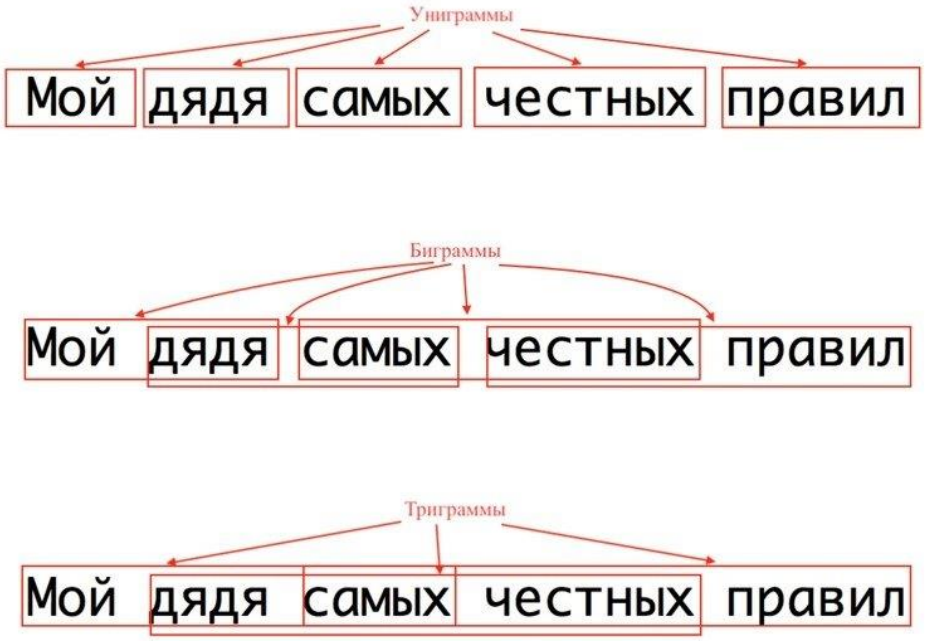

Рис.3 N-грамма для N=1,2,3

Степень модели указывает на количество слов, учитываемых при расчете вероятностей. Рисунок выше показывает несколько N-грамм для небольших значений N. Униграмма имеет степень = 1, биграмма – степень = 2, триграмма – степень = 3.

Представим фразу **w** как последовательность слов $\mathbf{w} = (\mathrm{w1}, \mathrm{w2}, \mathrm{w3}, \ldots, \mathrm{wk})$. Тогда более формально N-граммная модель рассчитывает вероятности по формуле:

$$p(\boldsymbol{w}) = \prod_{i=1}^{k+1} p(w_i | w_{i-n+1}^{i-1}), \quad (3)$$

где $w_{i-n+1}^{i-1} = (w_{i-n+1}, \ldots, w_{i-1})$. (4)

Тогда формула вычисления вероятностей биграммы представляется в виде:

$$p(\boldsymbol{w}) = \prod_{i=1}^{k+1} p(w_i | w_{i-1}). \quad (5)$$

1.2.3 Подход искусственного интеллекта

Данный подход получил широкую известность [16] и использует методы машинного обучения в своей основе, в частности – нейронные сети.

1.2.3.1 Рекуррентные сети

Рекуррентные нейронные сети (RNN) - это класс нейронных сетей с внутренней памятью. Такие сети используются в случаях, когда важна сама последовательность данных и та временная динамика, которая соединяет данные. Последовательные данные – это, в основном, просто упорядоченные данные, в которых связанные объекты следуют друг за другом. Примерами могут служить тексты, финансовые данные или последовательность ДНК, или данные временных рядов. Сеть запоминает свои входные данные благодаря внутренней памяти, что делает его идеально подходящим для задач машинного обучения, связанных с обработкой текстовых данных.

В RNN информация проходит через цикл. Когда принимается решение, сеть учитывает текущие входные данные, а также то, что она узнала на предыдущих слоях. Два изображения ниже иллюстрируют разницу в потоке информации между RNN и обычной нейронной сетью.

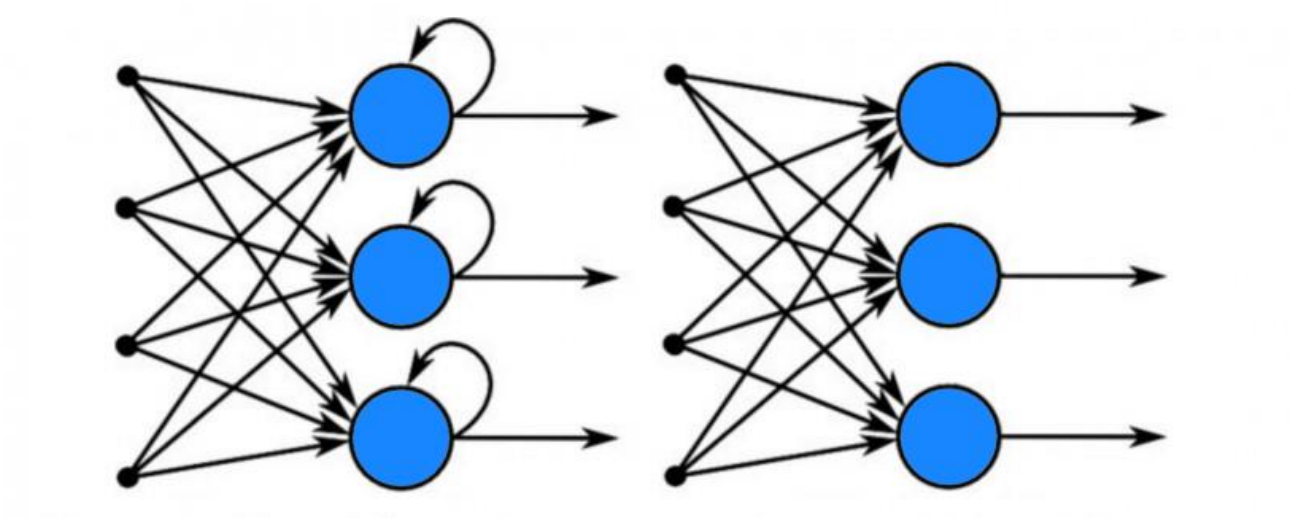

Рис.4 Сравнение структуры рекуррентной нейронной сети (справа) и сети прямого распространения (слева)

Нейроны сети производят вывод, копируют этот вывод и зацикливают его обратно на свой вход. Так они получают информацию не только от предыдущего слоя, но и от самих себя предыдущего прохода. Это означает, что порядок, в котором подаются данные и обучается сеть, становится важным.

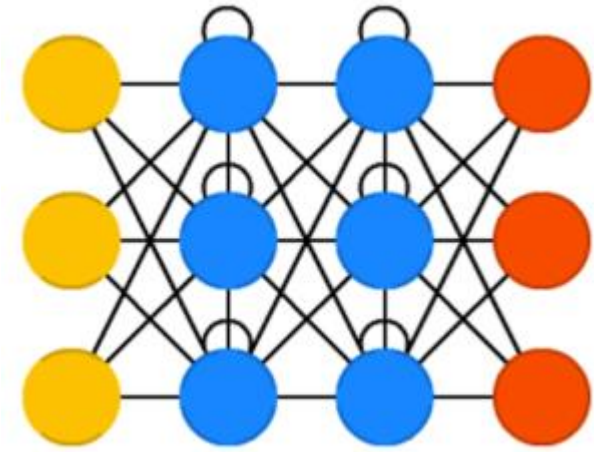

Рис.5 Схема строения рекуррентной нейронной сети

На рисунке выше изображена простая архитектура рекуррентной нейронной сети. Желтым цветом отмечены нейроны входного слоя, красным – выходного слоя. Синие нейроны позволяют сети носить название рекуррентной – их выход поступает на свой же вход. Именно такая архитектура используется в исследованиях [17][18].

Большой проблемой при работе сетей RNN является проблема исчезающего градиента, которая заключается в быстрой потере информации с течением времени. Сети способны хранить только самую недавнюю информацию, что не всегда является достаточным для данной задачи. Допустим, мы хотим предсказать последнее слово в тексте

“Я вырос в России… Мой родной язык *русский*”. Ближайший контекст предполагает, что последним словом будет называние языка, но, чтобы установить, какого именно языка, нам нужен контекст «России» из более отдаленного прошлого. Таким образом, разрыв между актуальной информацией и точкой ее применения может стать очень большим, что не всегда допустимо.

1.2.3.2 Рекуррентные сети с долгой краткосрочной памятью

Сети с долгой краткосрочной памятью (Long Short Term Memory, LSTM) стараются решить вышеупомянутую проблему RNN потери информации, используя фильтры и явно заданную клетку памяти.

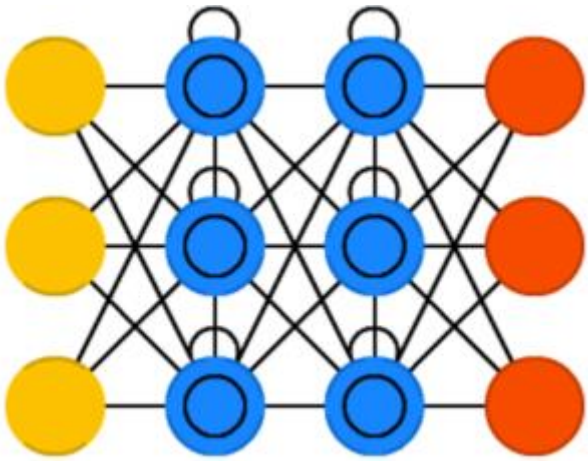

Рис.6 Схема строения рекуррентной нейронной сети с долгой краткосрочной памятью

Память в LSTM называется ячейками (синие нейроны на рисунке имеют в себе черный круг, символизирующий внутреннюю ячейку памяти), и их можно рассматривать как черные ящики, которые принимают в качестве входных данных предыдущее состояние и текущий входной параметр. Внутри у каждой клетки памяти есть три фильтра: входной, выходной и забывающий, которые решают, какую память сохранить и какую стереть.

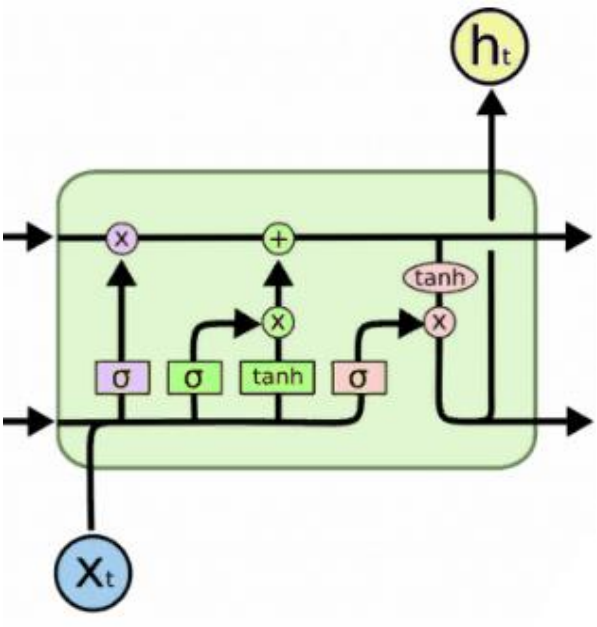

Рис.7 Ячейка LSTM в развертке

На рисунке изображена ячейка LSTM в развертке. x_t - входное значение, h_t – выходной вектор. Ячейка состоит из нескольких компонентов, разделенных на три группы по цвету. Входной фильтр (компоненты зеленого цвета) определяет, сколько информации из предыдущего слоя будет храниться в клетке. Выходной фильтр (компоненты розового цвета) определяет, сколько информации получат следующие слои. Ну а забывающий фильтр (компоненты фиолетового цвета) – какую часть информации можно отсечь. Целью этих фильтров является защита информации. Затем они объединяют предыдущее состояние, текущую память и входной параметр. Существуют различные модификации этой «классической» схемы ячейки. Сети LSTM способны научиться создавать более сложные структуры, чем классические рекуррентные сети. Решение задачи распознавания речи с использованием этой архитектуры было применено в исследовании [19].

1.2.3.3 Двунаправленные рекуррентные сети

Двунаправленные RNN (Bidirectional Recurrent Neural Networks, BRNN) основаны на той идее, что выход в момент времени t может зависеть не только от предыдущих элементов в последовательности, но и от будущих. Двунаправленные рекуррентные нейронные сети (BRNN) соединяют два скрытых слоя, работающих в противоположных направлениях, с одним выходом, позволяя им получать информацию как из прошлых, так и из будущих состояний. BRNN разбивает нейроны классической RNN на два направления, одно для прямых состояний (положительное направление времени), а другое для обратных состояний (отрицательное направление времени). Выходы из прямых состояний не связаны со входами

обратных состояний, и наоборот. Это приводит к общей структуре, которую можно увидеть на рисунке ниже, где она развернута на 4 временных шага. Например, если нужно предсказать недостающее слово в середине последовательности (предложения), то нужно учитывать как левый, так и правый контекст. Двунаправленные рекуррентные нейронные сети представляют собой два RNN, уложенных друг на друга.

Без обратных состояний эта структура упрощается до обычной однонаправленной прямой RNN. Если прямые состояния исключены, получается регулярная RNN с перевернутой временной осью.

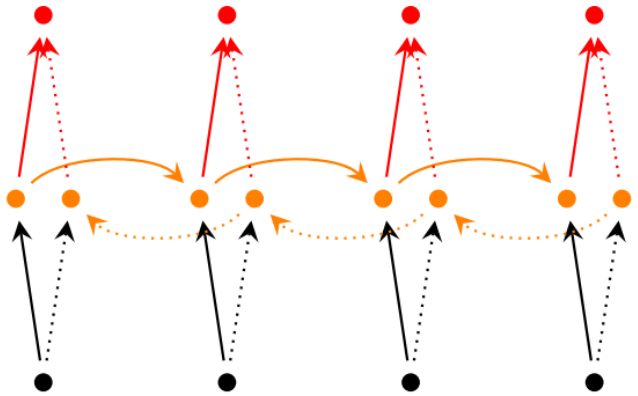

Рис.8 Общая структура двунаправленной рекуррентной нейронной сети

В общем обучении прямые и обратные состояния обрабатываются сначала в прямом проходе (на рисунке прямой линией изображено положительное/прямое направление), выходные нейроны вычисляются последними (на рисунке пунктирной линией изображено отрицательное/обратное направление). При обратном проходе происходит обратное: сначала обрабатываются выходные нейроны, затем передаются состояния вперед и назад. выход вычисляется на основе скрытого состояния обоих RNN - веса обновляются только после завершения прямого и обратного проходов. На рисунке выше общая структура двунаправленной рекуррентной нейронной сети (BRNN).

Двунаправленная рекуррентная нейронная сеть также находит свое применение в задачи распознавания речи во многих успешных исследованиях [20].

1.2.3.4 Другие архитектуры

В последнее время для решения задачи распознавания речи обретает популярность [21][22][23] метод, использующий сверточные нейронные сети.

Для применения сверточных сетей первоначально в задаче используется спектральное представление звукового потока. На основе полученного спектрального изображения (на

рисунке ниже изображена звуковая дорожка и соответствующее ей спектральное представление) стандартная сверточная сетевая архитектура для обработки спектрального представления сигнала выдает готовый распознанный результат.

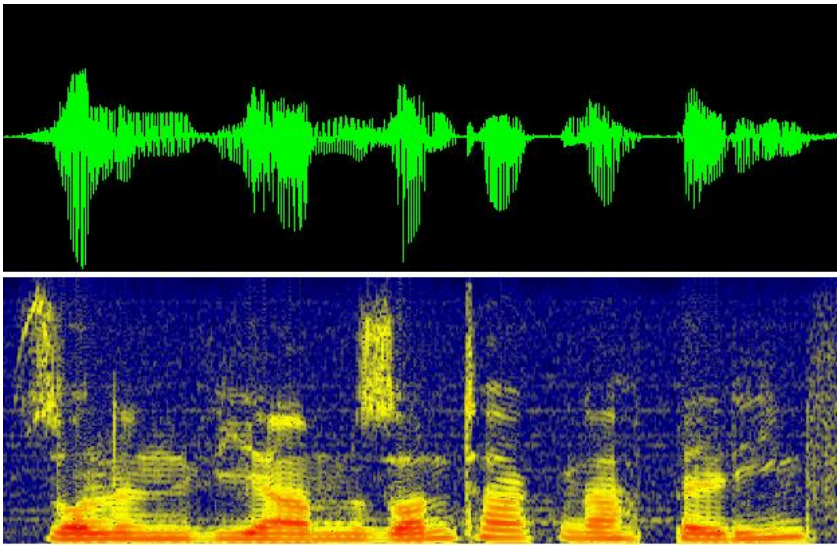

Рис.9 Изображение звуковой дорожки и соответствующего ей спектрального представления

Также построение нейронной сети для может происходить без использования в основе какой-либо определенной архитектуры. Примером этого может быть определение оптимальной архитектуры нейронной сети экспериментально в процессе работы генетического алгоритма. Такой подход также встречается в некоторых методах распознавания речи [24][25].

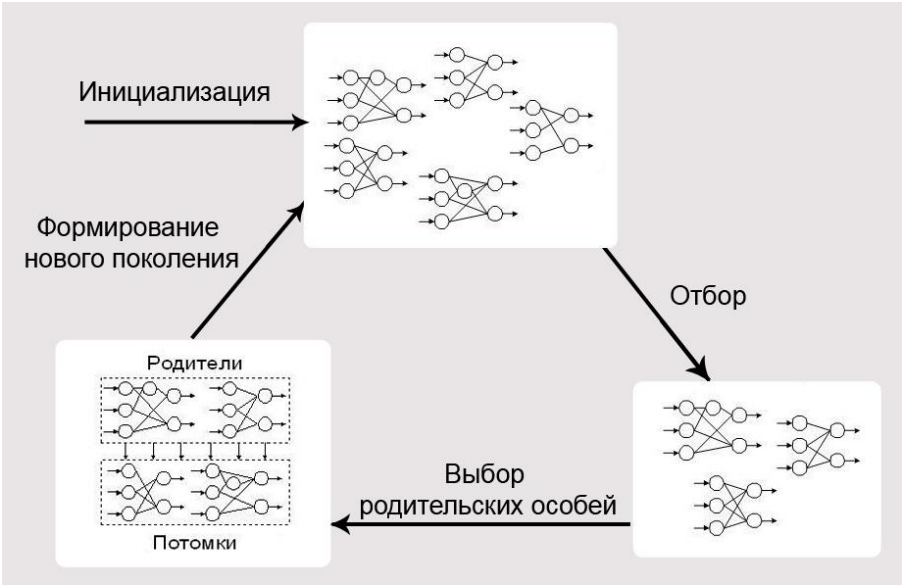

Рис.10 Схема нейроэволюционного алгоритма

При использовании генетических алгоритмов создаются правила отбора, позволяющие определить, лучше или хуже справляется новая нейронная сеть с решением задачи. Кроме того, определяются правила модификации нейронной сети.

Генетический алгоритм является самым известным на данный момент представителем эволюционных алгоритмов. Он заключается в параллельной обработке множества альтернативных решений. При этом поиск концентрируется на наиболее перспективных из них. Генетические алгоритмы для подстройки весов скрытых и выходных слоев используются следующим образом: каждая хромосома (в нашем случае – архитектура сети или набор ее весов) представляет собой вектор из весовых коэффициентов (веса считываются с нейронной сети в установленном порядке - слева направо и сверху вниз). Хромосома $a = (a_1, a_2, a_3, \ldots, a_n)$ состоит из генов a_i, которые могут иметь числовые значения, называемые "аллели". Популяцией называют набор хромосом (решений, нейронных сетей). Эволюция популяций - это последовательность поколений, в которых хромосомы (сети) изменяют свои признаки, чтобы каждая новая популяция наилучшим способом приспосабливалась к внешней среде. В конце можно выявить наиболее приспособленную особь (нейронную сеть с самой высокой точностью). Изменяя достаточно долго архитектуру нейронной сети и отбирая те архитектуры, которые позволяют решить задачу наилучшим образом, рано или поздно можно получить верное решение задачи.

Генетические алгоритмы обязаны своим появлением эволюционной теории (отсюда и характерные термины: популяция, гены, родители-потомки, скрещивание, мутация). Таким образом, существует возможность создания таких нейронных сетей, которые ранее не изучались исследователями (или не поддаются аналитическому изучению), но, тем не менее, успешно решают задачу.

1.3 Обзор существующих сервисов распознавания речи

Системы распознавания речи в наше время наиболее часто встречаются в виде интеграции в цифрового ассистента (виртуального цифрового помощника, Virtual Digital Assistant). Это полноценная система, принимающая информацию в виде звукового сигнала, содержащего речь человека, и возвращающая полноценный не бессмысленный ответ, а также способная выполнять определенные заданные действия (по планированию графика, выполнению ежедневных дел, поиска информации). Одной из ключевых функций цифрового помощника является распознавание человеческой речи (обычно записывающейся на микрофон). Он может быть интегрирован в мобильные устройства или компьютеры в качестве дополнительной программы или в так называемые «умные колонки», которые представляют собой смесь акустической системы и микрофонов работы с информацией

Speech-to-speech. В таблице ниже представлены наиболее популярные бренды, остающиеся примерно на одинаково высоком уровне качества работы. Во всех компаниях цифровой ассистент имеет закрытый исходный код. В некоторых случаях компании лишь упоминают общие методы их работы.

Таблица 1. Список лидирующих цифровых помощников на мировом рынке

Компания (бренд)	Цифровой помощник
Amazon	Alexa
Google	Google Assistant
Apple	Siri
Microsoft	Cortana
Яндекс	Алиса
Mail	Маруся
Xiaomi	小爱同学

Также существуют менее популярные программные компоненты, не предусматривающие интегрирование в цифровых помощников. К ним относятся программы, представленные в таблице ниже. Ни одна из них не представлена в виде открытого исходного кода, а также не описывает даже примерный принцип работы. Каждая из них имеет собственные достоинства и недостатки [26].

Таблица 2. Сравнение существующих программных компонентов по распознаванию речи

Программа	Достоинства	Недостатки
Dragon NaturallySpeaking	• Удобство в работе • Поддержка распознавания аудиофайлов (но ограниченная) • Возможность повышения качества работы за счет обучения в процессе распознавания	• Невысокое качество распознавания без обучения • Отсутствие поддержки русского языка
Braina	• Простота • Поддержка множества языков, в том числе русского	• Нестабильное качество распознавания, которое в среднем является неприемлемо низким • Невозможность непосредственной работы с аудиофайлами • Не поддерживается обучение
Voco Professional	• Пренебрежимо малы в связи с крайне низким качеством распознавания	• Крайне низкое качество распознавания речи
Real Speaker	• Обещают качество работы 90-	• Пока только в демо-

	100%	режиме
Speechlogger и Speechpad	• Доступность	• Невысокое качество распознавания речи • Необходимость использования сторонних средств («виртуального кабеля»)
Сервис Go Transcribe	• Высокое качество	• Платный (98$ за 10 часов речи), но качество превосходит стоимость

1.4 Выводы

Выбор и разработка метода распознавания речи строится исходя из особенностей поставленной задачи. Речь различается по многим параметрам, начиная от особенностей, связанных с самим диктором, заканчивая различным фоновым шумом. Для каждой поставленной задачи свой метод распознавания будет являться оптимальным. В данной работе планируется использование нейросетевых методов и методов скрытого Марковского моделирования, т.к. они могут обеспечить наилучшее качество работы и наиболее распространены в современных решениях аналогичных задач.

1.5 Постановка задачи курсового проекта

В результате проведенного обзора установлено, что в настоящее время существует несколько систем, использующих различные методы распознавания и преобразования аудиосигнала, содержащего речь человека в текстовый поток. Все исходные коды этих систем закрыты. Наиболее популярным и универсальным подходом является использование нейронных сетей. В связи с этим, в данном проекте ставится цель создания оптимальной системы для автоматического распознавания английской речи, для решения которой будет проводится сравнение качества работы различных методов распознавания. Для достижения этой цели предполагается решить следующие задачи:

1. Получение размеченной обучающей выборки
2. Изучение и разработка методов распознавания речи
3. Выбор и разработка оптимальной архитектуры нейросети
4. Изучение и проектирование оптимальной архитектуры системы

Раздел 2. Разработка оптимального метода распознавания речи

В разделе ставится задача распознавания речи и приводится описание предлагаемого метода распознавания и используемых метрик оценки качества распознавания. Задача разбивается на несколько подзадач и выделяются этапы процесса распознавания. Приводятся способы решения поставленных подзадач.

2.1 Постановка задачи обработки и распознавания речи

Задача распознавания речи заключается в автоматическом определении однозначной последовательности слов (или пустой последовательности), которая присутствует в исходном аудиофайле. Можно выделить несколько этапов в процессе распознавания:

1) Получение аудиофайла (сюда может входить процесс записи речи на микрофон, аналогово-цифровые преобразования, транскодирование или получение уже готового файла)
2) Обработка речевого сигнала, очистка от шума, усиление/ослабление фоновых звуков
3) Получение акустических представлений (извлечение признаков), которые будут использоваться для разделения классов звуков речи (к примеру, построение спектограмм)
4) Распознавание:
    1. Акустическая модель (encoder)
    2. Языковая модель (decoder)

В данной работе подробно рассматривается последние два пункта – перевод аудиоданных в оптимальную форму представления, а также распознавание и определение текста, присутствующего на входных аудиоданных.

Последний этап чаще всего делят на два элемента – encoder, decoder. Грубо говоря, encoder – языковая модель – позволяет переводить сегменты звука в последовательность фонем. В то же время, языковая модель формирует из последовательности распознанный ответ.

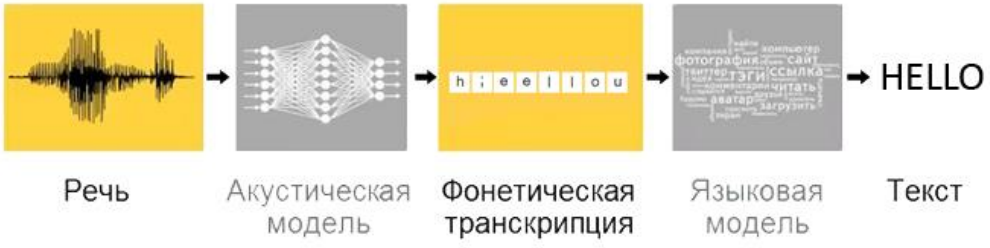

Рис.11 Представление системы распознавания с разделением на encoder, decoder

Формально, будем считать, что входная последовательность признаков для акустической модели $X = [x1, x2, \dots, x_T]$ размерности T должна на выходе языковой модели получить корректное распознавание в виде последовательности $Y = [y1, y2, \dots, y_U]$ размерности U. Тогда:

$$H = encode(X) \quad (6)$$

$$p(Y|X) = decode(H) \quad (7)$$

где $encode(.)$ и $decode(.)$ – функции, задаваемые акустической и языковой моделями. H – какое-то представление (последовательность морфем), $p(Y|X)$ – условная вероятность верного распознавания.

2.1.1 Ограничения

Зададим определенные ограничения задачи распознавания. В качестве языка распознавания выбран английский, хотя, вообще говоря, сам алгоритм работы никак не зависит от языка, кроме как от количества классов разбиения для классификации, так как основывается на фонетическом распознавании. Обучение будет проходить на готовых датасетах без проведения этапа очищения от шумов, так как размер датасета и имеющиеся вычислительные мощности не позволяют проводить обработку по очищению данных.

2.2 Описание работы выбранных методов распознавания речи

Обучающие данные для акустической модели могут быть представлены в разной форме. Для выявления оптимального способа представления данных будет проведено сравнение работы нейронных сетей, обученных на данных, представленных в виде спектограмм, а также в виде мел-кепстральных коэффициентов (MFCC).

2.2.1 CTC

Коннекционистская временная классификация (CTC) – это тип выхода нейронной сети и связанная с ним функция оценки, предназначенная для обучения рекуррентных нейронных сетей.

Представим задачу распознавания речи более формально. Необходимо найти отображение входной последовательности $X = [x1, x2, \dots, xT]$ (аудиофайла) к соответствующему выходу $Y = [y1, y2, \dots, yU]$ (транскрипция, последовательность фонем). С помощью CTC можно для заданного X найти выходное распределение по всем возможным Y без решения задачи сегментации входа.

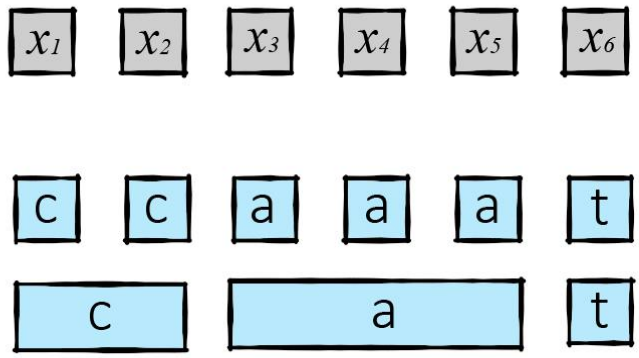

Рис.12 Вектор X и свертка соответствующего ему ответа Y слова "cat"

Допущения:

1) Будем назначать выходной символ для каждого элемента входной последовательности (сделаем X и Y одной длины)

2) Будем сворачивать повторы символов в выходе Y для получения ответа (на рисунке выше представлен пример для распознавания слова "cat". Операция свертки переводит "ccaaat" в "cat". Вектор X представляет входную последовательность)

3) Введем дополнительную «пустую фонему» в алфавит, чтобы корректно распознавать слова с удвоенной буквой (на рисунках ниже она обозначена как $\mathscr{E}$)

4) Если слово имеет две одинаковых буквы в ответе, то в выходе обязательно должна присутствовать «пустая фонема» между распознанными фонемами (на рисунке ниже представлена свертка слова "hello" с пустой фонемой. Сначала проходит операция свертки, затем удаляются символы $\mathscr{E}$, затем получаем ответ)

5) Соответствие X к Y «многие к одному».

6) Длина Y не может быть больше, чем длина X

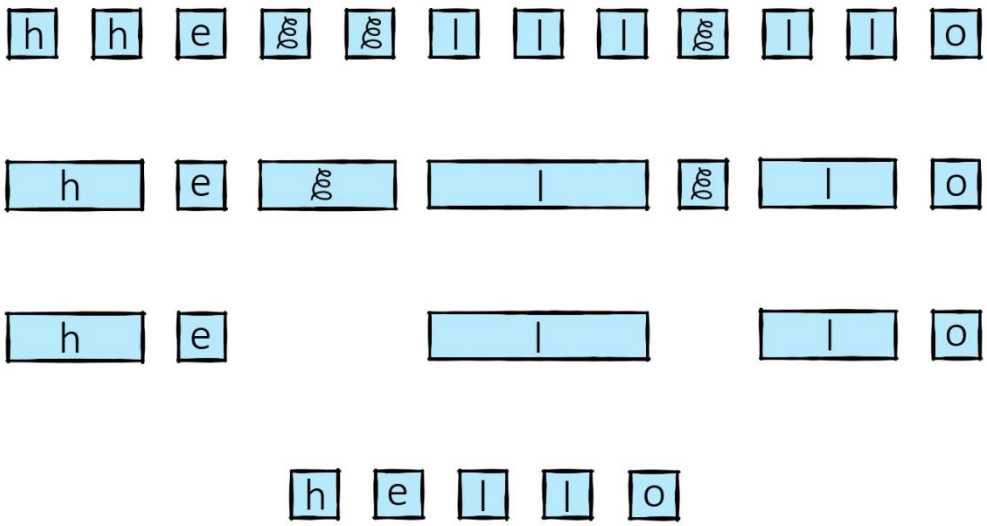

Рис.13 Шаги алгоритма CTC сверху вниз. ε - пустая фонема

На рисунке ниже представлены примеры корректной и ошибочной работы алгоритма для слова “cat”. Слева для каждой распознанной последовательности алгоритм после свертки получит слово “cat”. Справа в первом случае работа алгоритма даст результат “ccat”, в третьем случае результат “ct”, а во втором случае ошибка произошла в первом допущении - выходной символ назначен не для каждого элемента входной последовательности.

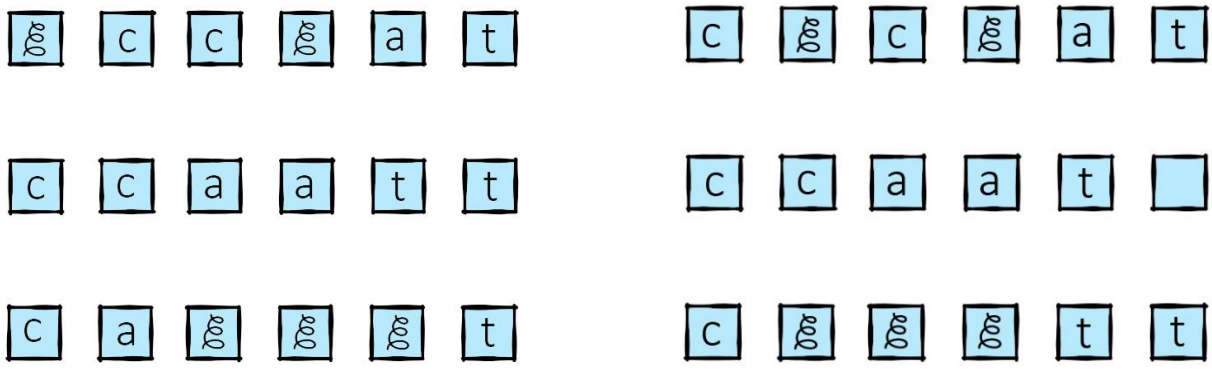

Рис.14 Корректное (слева) и ошибочное (справа) распознавание для слова “cat”

Введем функцию потерь. Выход CTC представляет собой матрицу, где столбцы соответствуют временным шагам, а каждая строка соответствует букве в алфавите. Сумма вероятностей по столбцам равна 1. Используя матрицу оценок для каждого символа на каждом временном шаге, рассчитываем вероятность различных последовательностей выходов (используя динамическое программирование для облегчения вычислений). Затем

суммируем по всем допустимым последовательностям. Таким образом, потеря рассчитывается путем суммирования всех вероятностей всех возможных выходных последовательностей текста.

$$p(Y \mid X) = \sum_{A \in A_{X,Y}} \prod_{t=1}^{T_p} p(a_t \mid X), \quad (8)$$

где $p(Y \mid X)$ – условная вероятность СТС, $\sum$ - сумма по набору допустимых распознаваний, $\prod$ - вычисление вероятности для одного распознавания шаг за шагом.

Приведем пример и рассмотрим алфавит из 2 символов {a,b}. На рисунке представлена матрица СТС для данного примера. Каждый кружок символизирует вероятность распознавания конкретной буквы на конкретном временном шаге. По строкам располагаются символы алфавита (последний символ является пустой фонемой), по столбцам – временные интервалы. Рассчитаем вероятности, для каждого возможного распознавания (пути).

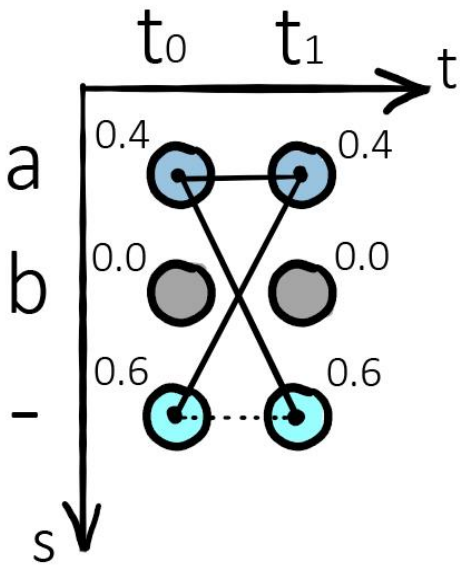

Рис.15 Пример выходной матрицы. Более светлые ячейки для больших вероятностей

В приведенном выше примере оценка для пути “aa” равна 0,4·0,4=0,16, в то время как для “a-” она равна 0,4·0,6=0,24, а для “-a” - 0,6·0,4=0,24. Чтобы получить оценку для какого-то ответа Y мы суммируем вероятности (оценки) всех путей, соответствующих этому ответу. Предположим, что ответ Y - это “a“. Чтобы получить такой ответ, нужно вычислить все возможные пути длины 2 (потому что матрица имеет 2 временных шага), которые являются - ”aa“, ”a-“ и ”-a". Просуммируем вероятности каждого пути и получим: 0.4·0.4+0.4·0.6+0.6·0.4=0.64. Если ответ предполагается равным “”, то существует только один соответствующий путь, а именно “--”, который дает общую оценку 0,6·0,6=0,36.

Необходимо обучить NN присваивать высокую вероятность (в идеале, значение 1) для правильных классификаций. Поэтому будем максимизировать произведение вероятностей правильных классификаций для обучающего набора данных. Сформулируем эквивалентную задачу - минимизировать потерю обучающего набора данных, где потеря представляет собой отрицательную сумму логарифмических вероятностей. Для вычисления потери для одного ответа нужно вычислить вероятность, взять логарифм и поставить минус перед результатом.

2.2.2 Декодирование CTC

Рассмотрим теперь способ определения наиболее вероятного текста по выходной матрице вероятностей CTC. Простым и очень быстрым алгоритмом является лучшее декодирование пути, которое состоит из двух шагов:

1. вычислить наилучший путь, принимая наиболее вероятный символ за временной шаг
2. отменить кодировку, сначала удаляя дубликаты символов, а затем удаляя все «пустые фонемы» из пути. То, что остается, представляет собой распознанный текст

В качестве примера рассмотрим следующую матрицу CTC.

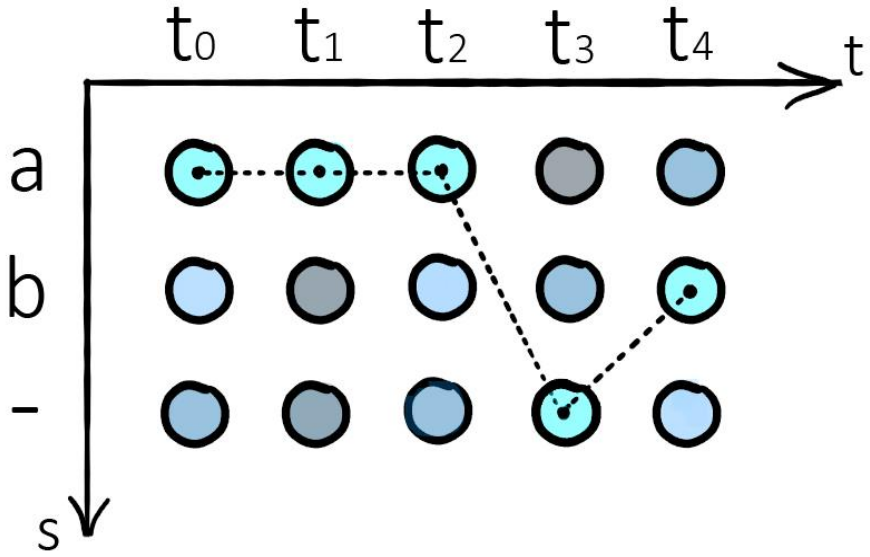

Рис.16 Пример выходной матрицы. Пунктирная линия показывает наилучший путь

Алфавит содержит “a”, “b” и "-". Есть 5 временных шагов. Применим алгоритм лучшего декодирования пути к этой матрице: наиболее вероятным символом в момент t_0, t_1 и t_2 является “a”. Пустой символ имеет самый высокий балл в t_3. В момент t_4 наибольшую вероятность имеет символ “b”. По алгоритму, получается путь “aaa-b". Теперь удаляем повторяющиеся символы, это дает “a-b“, затем удаляем пустой символ из оставшегося пути, это дает нам текст “ab”, который мы выводим как распознанный ответ.

Лучшее декодирование пути состоит в поиске символа с самым высоким баллом для каждого временного шага. Если у нас есть символы C и временные шаги T, то алгоритм имеет время выполнения $O\ (T \cdot C)$.

2.2.3 Нечёткий поиск

Будем улучшать имеющийся транскодированный ответ с помощью языковой модели, использующей нечеткий поиск в основе. Задачу нечеткого поиска можно сформулировать так: по заданному слову найти в тексте или словаре размера n все слова, совпадающие с этим словом (или начинающиеся с этого слова) с учетом k возможных различий.

2.2.3.1 Расстояние Левенштейна

В числе наиболее известных метрик — расстояние Левенштейна, которое является наиболее часто применяемой метрикой и будет использоваться в работе.

Пусть $d_{i,j}$ есть расстояние между префиксами строк x и y, длины которых равны, соответственно, i и j, то есть $d_{i,j} = d(x(1,i), y(1,j))$. Цену преобразования символа a в символ b обозначим через $w(a,b)$. Таким образом, $w(a,b)$ – это цена замены одного символа на другой, когда $a \neq b$, $w(a,\varepsilon)$ – цена удаления a, а $w(\varepsilon,b)$ – цена вставки b. Заметим, что в случае, когда выполнены нижеследующие условия, d является расстоянием Левенштейна:

$$w(a,\varepsilon) = 1 \tag{9}$$

$$w(\varepsilon,b) = 1 \tag{10}$$

$$w(a,b) = 1, \text{если } a \neq b, \tag{11}$$

$$w(a,b) = 0, \text{если } a = b \tag{12}$$

В процессе вычислений значения $d_{i,j}$ записываются в массив $(m+1) * (n+1)$, а вычисляются они с помощью следующего рекуррентного соотношения.

$$d_{i,j} = min\{d_{i-1,j} + w(x_i,\varepsilon), d_{i,j-1} + w(\varepsilon,y_j), d_{i-1,j-1} + w(x_i,y_j)\} \tag{13}$$

Оно выводится следующим образом. Если предположить, что известна цена преобразования $x(1,i-1)$ в $y(1,j)$, то цену преобразования $x(1,i)$ в $y(1,j)$ мы получим, добавив к ней цену удаления x_i. Аналогично, цену преобразования $x(1,i)$ в $y(1,j)$ можно получить, прибавив цену вставки y_j к цене преобразования $x(1,i)$ в $y(1,j-1)$. Наконец, зная цену преобразования $x(1,i-1)$ в $y(1,j-1)$, цену преобразования $x(1,i)$ в $y(1,j)$ мы получим, прибавив к ней цену замены x_i на y_j. Вспомним, что расстояние $d_{i,j}$ является минимальной ценой преобразования $x(1,j)$ в $y(1,j)$, поэтому из трех указанных выше операций надо выбрать самую дешевую.

На рисунке ниже показан пример определения расстояния Левенштейна между двумя строками «intention» и «execution». На рисунке синие стрелочки соответствуют одинаковым буквам в строках, красные отражают операцию удаления, желтые – добавления, а зеленые – замены.

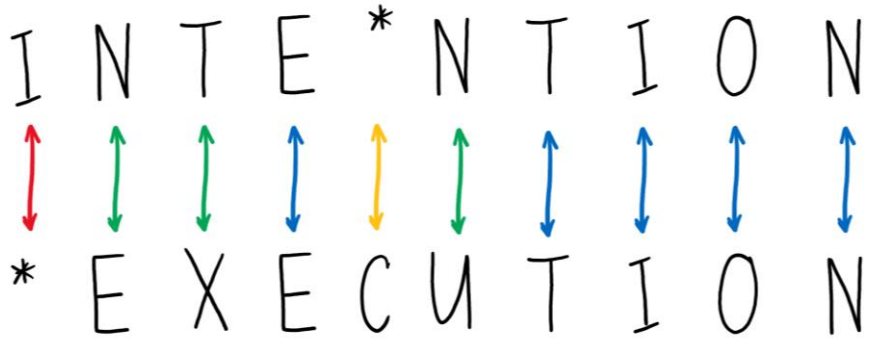

Рис.17 Визуализация определения расстояния Левенштейна между двумя строками

2.2.3.2 Мера Жаккара

Мера Жаккара является мерой сходства, определяющаяся через размер пересечения и размер объединения двух наборов меток.

$$K(J) = \frac{c}{a+b-c}, \quad (14)$$

где a — количество элементов из первого множества, b — количество элементов второго множества, c — количество элементов, общих для 1-го и 2-го множества.

2.3 Модель нейронной сети, используемая для распознавания речи

Для построения акустической модели будут реализованы нейронные сети разной архитектуры, проведено сравнение качества работы и определение оптимальной сети, а также оптимального способа представления данных.

2.3.1 RNN (One layer GRU –управляемый рекуррентный нейрон)

Построим первую нейронную сеть как рекуррентную сеть, состоящую из одной ячейки GRU. На рисунке показано ее примерное строение. В каждый момент времени $i \in N$ на вход подается вектор $input(i)$.

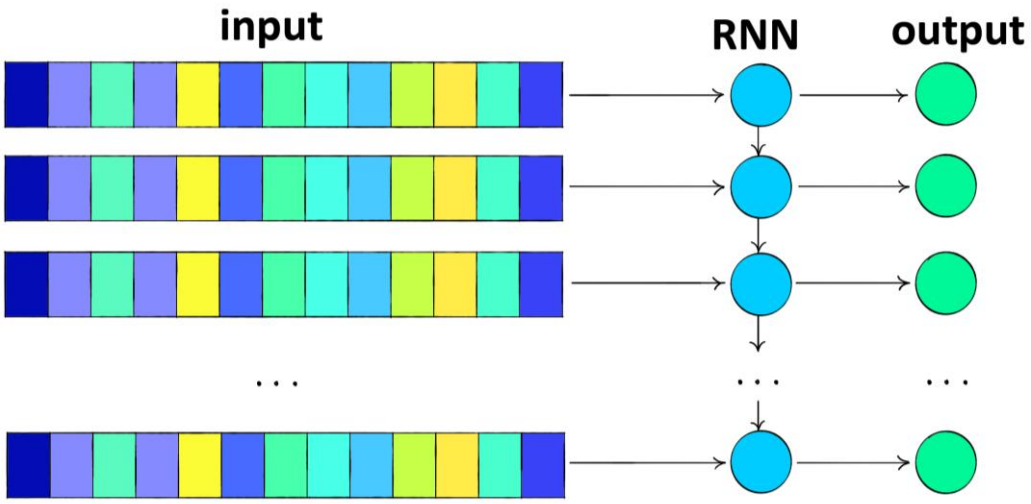

Рис.18 Схема строения первой модели NN

На каждом временном шаге говорящий произносит один из 28 возможных символов, включая каждую из 26 букв английского алфавита, а также пробел («») и апостроф (').

Выходные данные RNN на каждом временном шаге - это вектор вероятностей с 29 записями, где j-я запись кодирует вероятность того, что j-й символ произносится во временной последовательности. (Дополнительный 29-й символ - это пустой «символ», используемый для дополнения обучающих примеров в пакетах, содержащих неравную длину.) Выходной вектор output состоит из массива вероятностей распознавания каждой буквы на каждом шаге, т.е выхода CTC матрицы.

2.3.2 RNN + TimeDistributed Dense

TimeDistributed Dense позволяет применить слой к каждому временному срезу входного сигнала. Другими словами – это промежуточный слой между выходами RNN на каждом временном шаге и ответом (см рис).

На рисунке этот слой изображен красным.

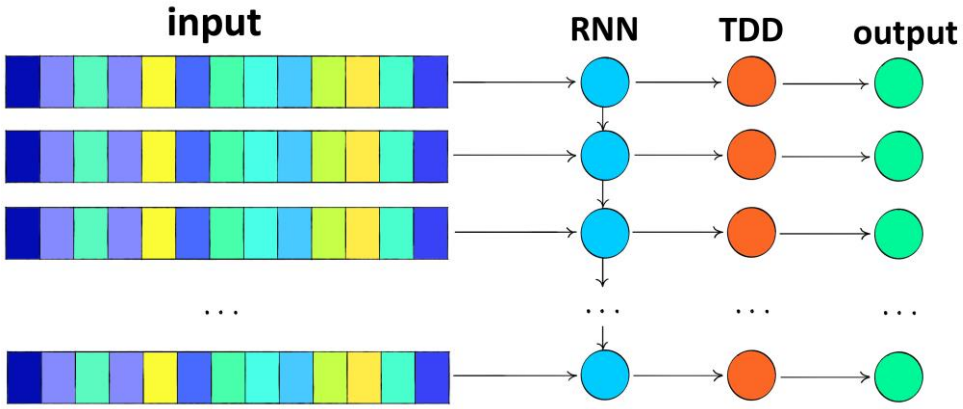

Рис.19 Схема строения второй модели NN

2.3.3 CNN + RNN + TimeDistributed Dense

В третьей модели добавим использование сверточной нейронной сети на входных данных.

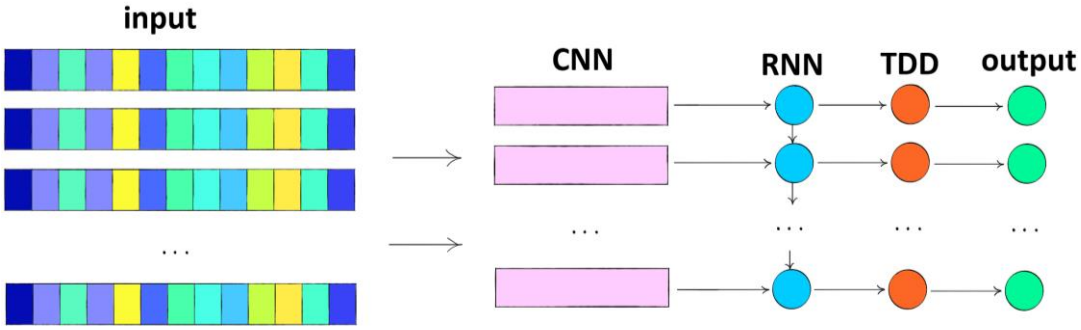

Рис.20 Схема строения третьей модели NN

Сверточная нейронная сеть будет иметь следующие подобранные экспериментально параметры:

- *filters=200*
- *kernel_size=11*
- *strides=2*
- *padding='valid'*

Выход CNN уже будет подаваться на вход рекуррентной сети. CNN имеет 1D измерение.

2.3.4 Deeper RNN + TimeDistributed Dense

Четвертая сеть будет состоять из глубокой рекуррентной сети DRNN. В качестве DRNN может задаваться переменное количество слоев RNN, к примеру, два слоя - RNN1 и

RNN2. Выходная последовательность первого рекуррентного слоя используется как входные данные для следующего рекуррентного слоя

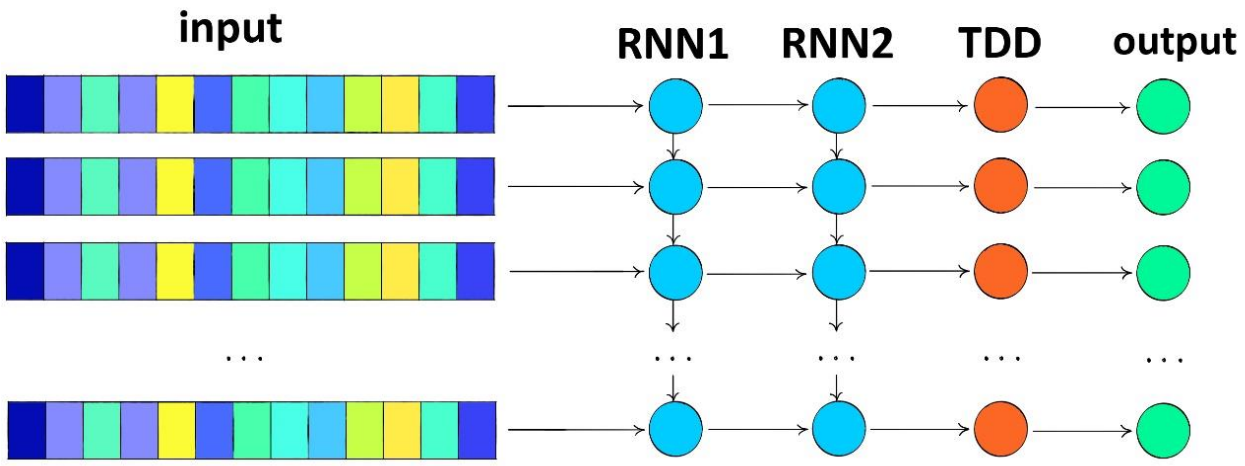

Рис.21 Схема строения четвертой модели NN

2.3.5 Bidirectional RNN + TimeDistributed Dense

Одним из недостатков обычных RNN является то, что они могут использовать только предыдущий контекст. В задаче распознавании речи на вход подается уже полностью записанный аудиофайл, поэтому нет причин не использовать будущий контекст. Двунаправленные RNN (Brnn) делают это, обрабатывая данные в обоих направлениях с помощью двух отдельных скрытых слоев, которые затем подаются вперед на один и тот же выходной слой.

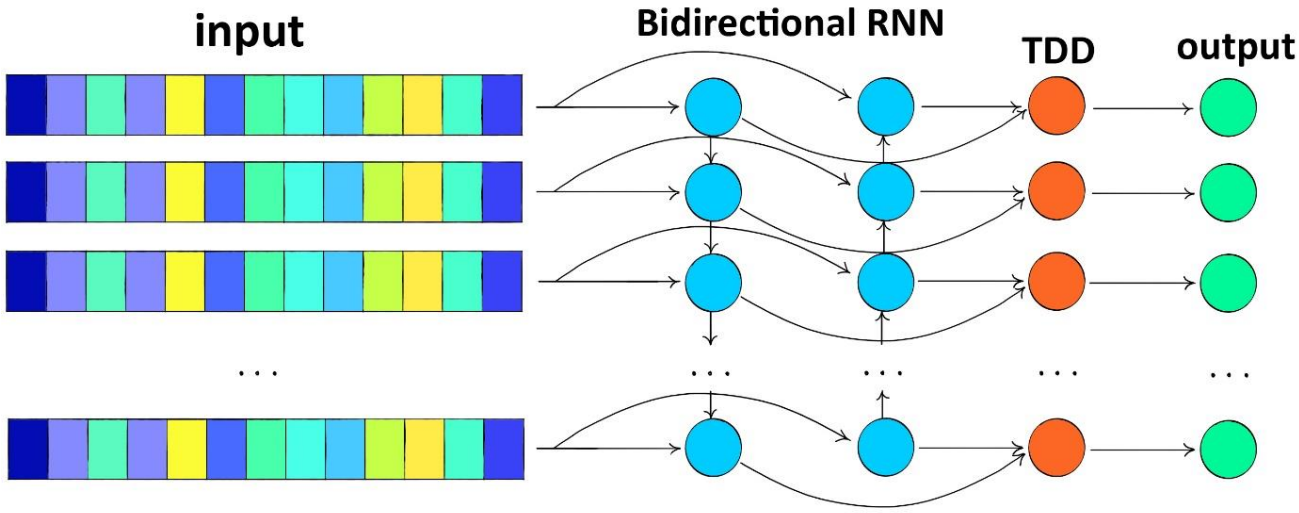

Рис.22 Схема строения пятой модели NN

2.4 Метрики качества метода распознавания

Для оценки качества распознавания существуют определенные метрики. Одна из наиболее популярных – WER (Word Error Rate) – мера оценки ошибки на уровне слов.

В системах автоматического распознавания речи основным показателем качества является точность распознавания, которая определяется как процент правильно распознанных слов (WRR- Word Recognition Rate) или, наоборот, неправильно распознанных слов (WER- Word Error Rate).

Метрика WER является коэффициентом ошибки в словах и вычисляется по формуле:

$$WER = \frac{S+D+I}{N}, \quad (15)$$

где S (substitutions) – количество замен слов (“twinkle” -> “crinkle”), D (deletions) – количество операций удаления слов – (“get it done” -> “get done”), I (insertions) – количество вставок - это всякий раз, когда добавляется слово, которое не было сказано (“trailblazers” -> “tray all blazers”)), N – (number of words spoken) – количество произнесенных слов.

По некоторым оценкам, на большом наборе произвольных аудиозаписей с речью даже человек допускает ошибки вплоть до WER = 0.05. Согласно недавним исследованиям [27] для современных систем распозавания сечи коэффициент составляет около 20% - для Google ≈18.3% Также можно рассматривать метрику CER (или LER, Character Error Rate/Letter Error Rate), которая рассчитывается аналогично, но на уровне букв, и по сути своей является расстоянием по Левенштейну.

Будем использовать обе эти метрики. Первая поможет анализировать качество работы на уровне слов, вторая – ниже, на уровне букв.

2.5 Выводы

Задача обработки и распознавания речи будет состоять из нескольких этапов и включать несколько моделей, решающих определенную задачу в процессе поиска ответа. Основной моделью будет являться акустическая модель, использующая в основе нейронную сеть. Выходной сигнал акустической RNN - это вероятность символов, которые затем декодируется с помощью различных алгоритмов для получения наилучшего ответа. Если полученный ответ слишком плох, его можно улучшить с помощью языковой модели, использующей информацию о семантическом сходстве слова ответа с набором часто встречающихся слов в данном языке.

Раздел 3. Разработка системы обработки и распознавания речи

На данном этапе производился выбор языка и вспомогательных модулей для реализации системы, произведена сама реализация, написана документация и проверена работа полученной системы.

3.1 Выбор языка и вспомогательных модулей для программной реализации системы

Реализация нейросети и дополнительных методов и алгоритмов происходила на языке Python. В качестве вспомогательных библиотек для построения нейронных сетей использовались библиотеки TensorFlow и Keras, а также Numpy для реализации функций и алгоритмов. В качестве вспомогательной библиотеки использовались общеупотребительные модули scipy (для перевода аудиозаписей в .wav формат), json (для построения вспомогательных данных – словарей соответствия имени аудиофайла расшифровке текста, который присутствует в аудиофайле), matplotlib (для построения графиков).

3.2 Программная реализация моделей и алгоритмов

Опишем реализованный функционал системы распознавания подробнее. Система включает несколько классов и функций, сформированных в общие блоки:

1 class AudioGenerator - класс, который используется для работы с аудиофайлами, выделения признаков и предоставления их в сеть для обучения или тестирования. Включает:

- def get_batch - формирует группы (batch) обучающих, проверочных или тестовых данных
- def sort_data_by_duration - Сортировка обучающих или проверочных наборов по (возрастающей) продолжительности
- def shuffle_data_by_partition - перемешивает обучающие или проверочные данные
- def shuffle_data - перемешивает данные. Вызывается после полного прохождения

обучение или для валидации данных в процессе обучения

- def sort_data - Сортировка данных по длительности
- def plot_spectrogram_feature – функция построения спектрограмм
- def plot_mfcc_feature – функция построения Mel-frequency cepstrum

2 class Models – класс, в котором описывается строение нейронных сетей, использующихся в работе

- def simple_rnn_model – строит модель NN, использующую GRU
- def rnn_model – строит модель NN, использующую рекуррентный слой RNN и слой временного распределения TDD
- def cnn_rnn_model – строит модель NN, состоящую из сверточного слоя между входными данными и RNN + TDD
- def deep_rnn_model – строит глубокую нейросетевую модель с возможностью создания заданного количества подряд идущих слоев RNN
- def bidirectional_rnn_model – строит двунаправленную рекуррентную нейронную сеть (BRNN), соединяющую два скрытых слоя противоположных направлений с одним и тем же выходом

3 CTC training

- def ctc_lambda_func – функция для расчета потерь CTC
- def add_ctc_loss – функция, возвращающая модель с CTC потерей
- def train_model – функция для обучения модели

4 class LanguageModel

- def levenshtein_distance – функция вычисления расстояния Левенштейна для двух строк
- def ngram – функция для построения N-gram. Возвращает их как список
- def jaccard – функция вычисления коэффициента Жаккара
- def minimumEditDistance_spell_corrector – основная функция класса. Возвращает результат языковой модели – слово, для которого расстояние Левенштейна между ним и входящим словом минимально.
- def ngram_spell_corrector – основная функция класса. Возвращает результат языковой модели – слово с максимальным коэффициентом Жаккара от него и входящего слова.

Таким образом, структура системы распознавания речи представлена на рисунке ниже. Отдельные блоки представляют отдельные логические компоненты системы. Стрелками указана очередность выполнения отдельных блоков при поступлении на вход системе нового аудиофайла для распознавания. Блок Acoustic model включает в себя одну

нейронную сеть NN, которая будет признана оптимальной экспериментально в разделе 4. Блок CTC decoding состоит из метода поиска лучшего пути (beam search decoding). Остальные блоки включают в себя пару методов, среди которых также необходимо выделить один наиболее подходящий для формирования наибольшей точности распознавания в данной задаче.

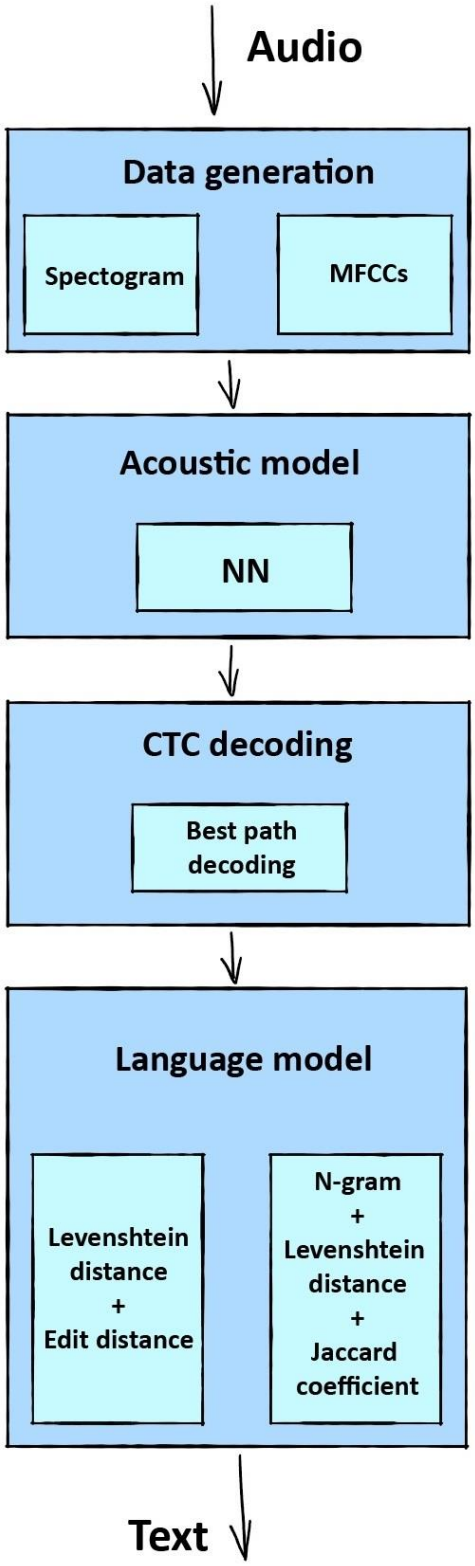

Рис.23 Диаграмма структуры системы распознавания речи

3.3 Выводы

В данном разделе представлены реализованные компоненты работы системы распознавания речи, описан функционал системы, взаимодействие её компонентов, представлена документация. Был определен язык программирования и вспомогательные библиотеки.

Раздел 4. Экспериментальные исследования метода распознавания речи

В данном разделе приводятся результаты экспериментальных исследований методов распознавания речи с помощью конструирования нейронных сетей с различными архитектурными параметрами, а также выбор метода распознавания с оптимальным показателем работы, таким как точность распознавания. В связи с тем, что результат обучения нейронных сетей во многом определяется размером обучающей выборки, была создана размеченная выборка фиксированного объёма, и на этой выборке продемонстрировано влияние различных параметров архитектуры сети на результат работы. Проведённые исследования позволили построить наиболее точную систему распознавания.

В данном разделе также описана обучающая выборка и её размер.

4.1 Описание обучающих выборок, объема и особенностей

Обучающая и валидационная выборки состоят из базы данных LibriSpeech. Она включает в себя из около 1000 часов английской речи с различными диалектами. Содержит более 300 голосов. Аудиофайлы чистые без фонового шума и дополнительных артефактов.

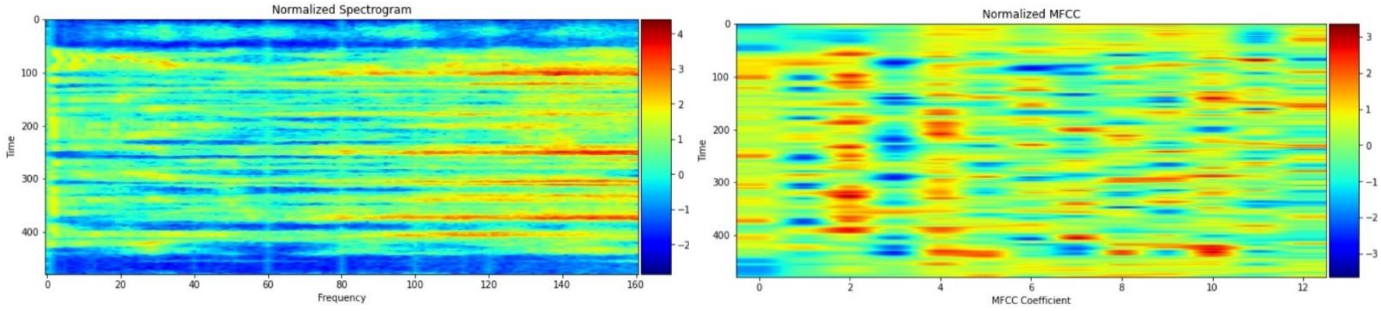

Рис.24 Спектрограмма и мел-коэффициенты для одного аудио сигнала

Будем представлять обучающие данные в виде спектрограмм и в виде мел-коэффициентов. Экспериментально сравним, какое из представлений окажется лучше для выявления скрытых признаков.

4.2 Экспериментальное исследование влияния архитектурных параметров нейросетевых моделей на точность распознавания

Для определения оптимальной архитектуры было протестировано много различных архитектур нейронных сетей.

Будем сравнивать влияние входных данных на качество работы сети. Для этого обучим описанные архитектуры на данных, представленных в виде спектрограмм и в виде мел-коэффициентов. На графиках ниже показаны результаты обучения сетей на разных данных – validation loss на протяжении 20 эпох.

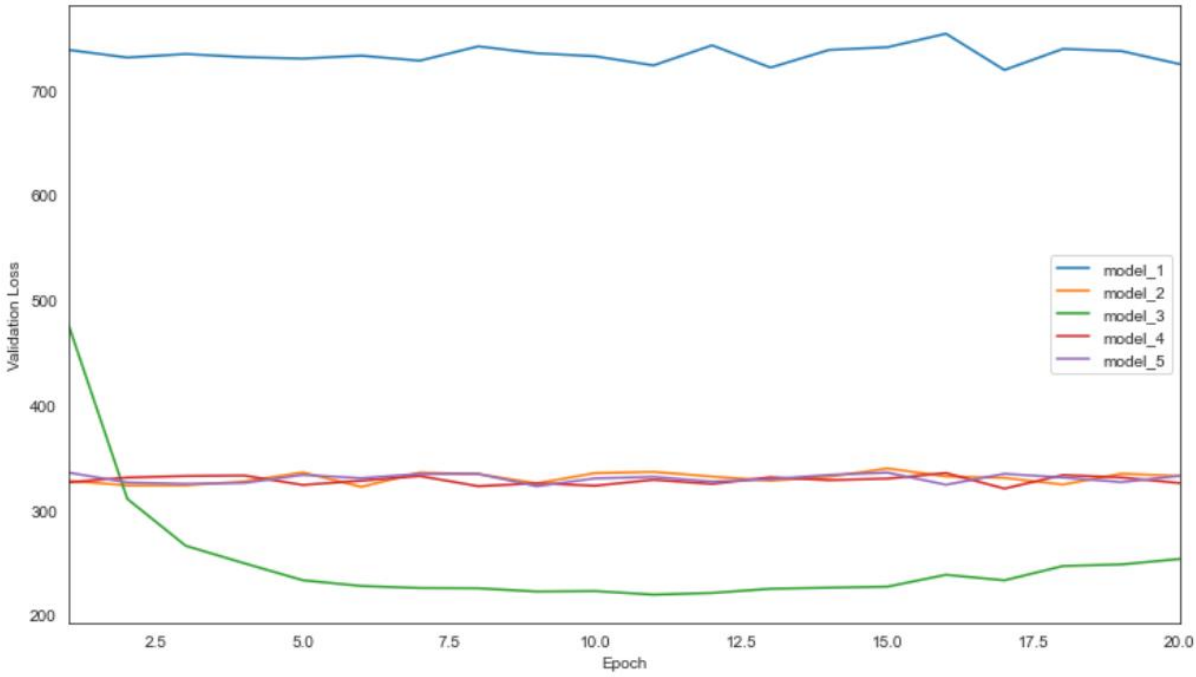

Рис.25 Функция потерь валидации при обучении на спектрограммах.

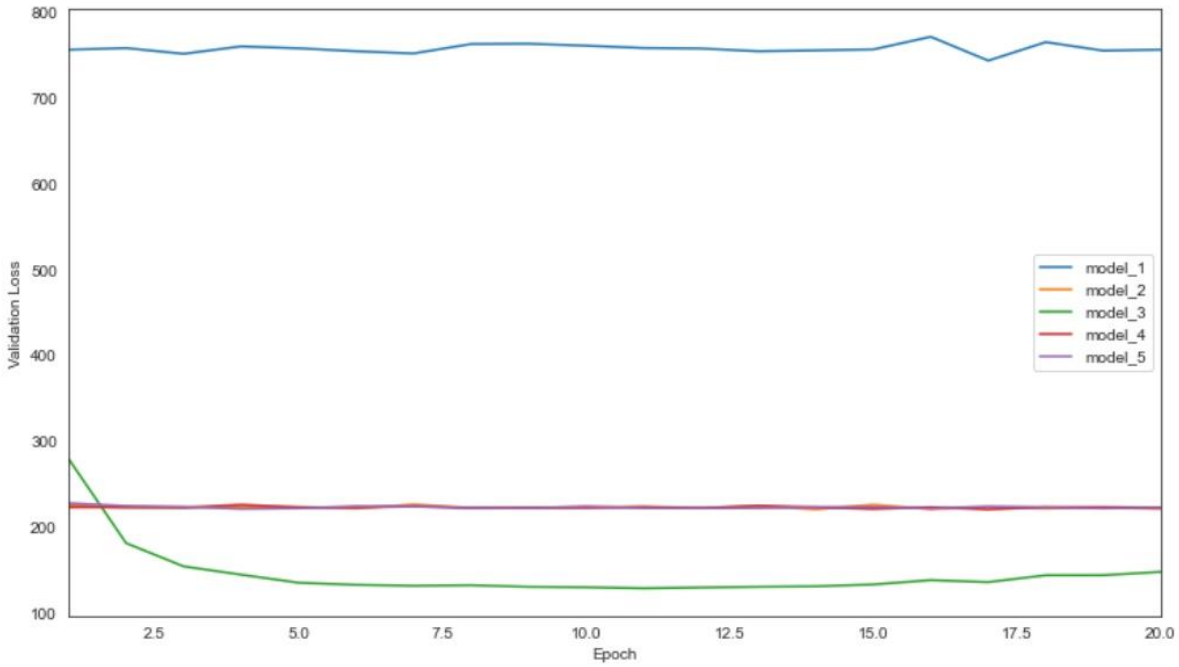

Рис.26 Функция потерь валидации при обучении на мел-коэффициентах.

Можно видеть, что при обучении на спектрограммах большинство сетей дает результат хуже, чем на мел-коэффициентах, кроме первой модели (которая остается примерно на том же уровне и не улучшает прогресс в обоих случаях). Третья сеть является наиболее точной в обоих случаях, однако на мел-коэффициентах дает результат точнее. Будем использовать мел-коэффициенты как вариант представления входных данных.

При сравнении описанных выше моделей на мел-коэффициентах был получен следующий результат обучения (см график ниже). Первая модель, состоящая их одного элемента RNN, дала самый плохой результат (голубая кривая). Модель, использующая CNN, давала наименьшее значение функции потерь (зеленая кривая), и наилучшую точность.

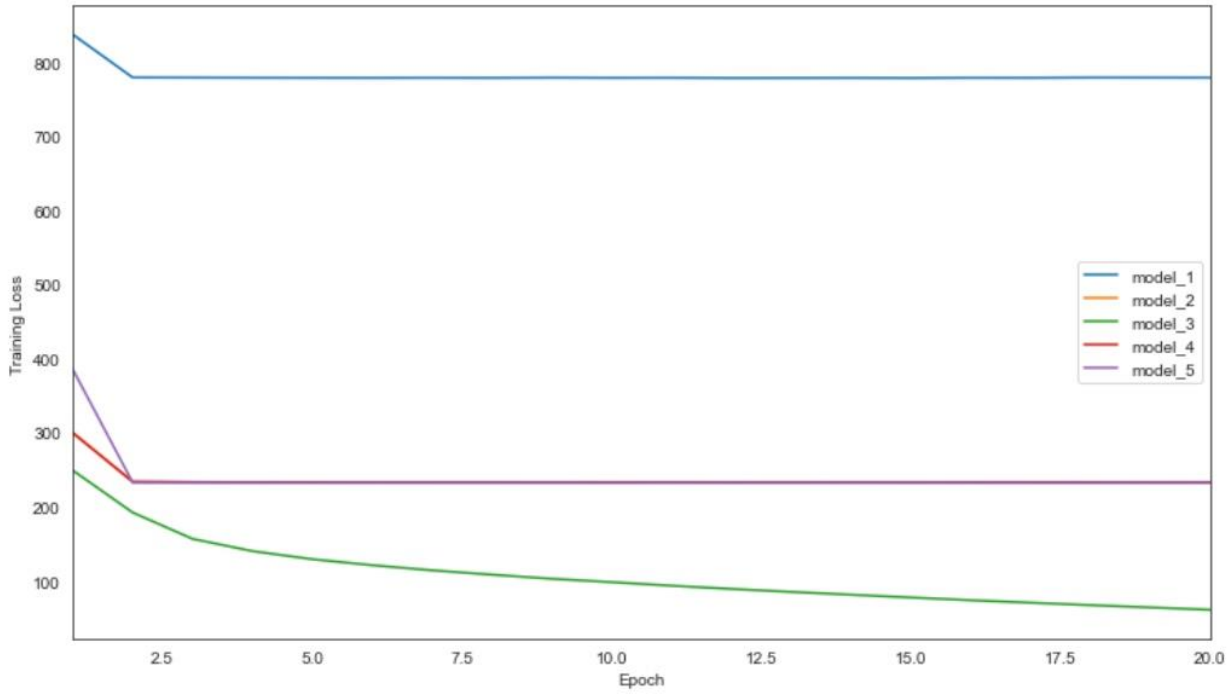

Рис.27 Зависимость функции потерь обучения от количества эпох обучения разных моделей

Также из графика видно, что модель 3 продолжает обучаться на протяжении всех эпох (зеленая кривая стремится вниз, в то время как все остальные после первых 2-3 эпох не улучшают показатели обучения). Однако если посмотреть на график валидации, то можно сказать, что после 15 эпохи сеть 3-ей архитектуры начинает немного переобучаться и адаптироваться под обучающую выборку, хотя все равно является самой оптимальной среди всех архитектур. Это может быть вызвано тем, что сверточный слой лучше находит на изображении дополнительные признаки, использующиеся потом RNN при обучении.

```
Epoch 1/20
101/101 [==============================] - 145s - loss: 246.7676 - val_loss: 215.9699
Epoch 2/20
101/101 [==============================] - 144s - loss: 187.5169 - val_loss: 179.8064
Epoch 3/20
101/101 [==============================] - 152s - loss: 151.9236 - val_loss: 146.9361
Epoch 4/20
101/101 [==============================] - 152s - loss: 133.9369 - val_loss: 135.4676
Epoch 5/20
101/101 [==============================] - 148s - loss: 122.4069 - val_loss: 134.8893
Epoch 6/20
101/101 [==============================] - 148s - loss: 113.4147 - val_loss: 128.9967
Epoch 7/20
101/101 [==============================] - 142s - loss: 106.1782 - val_loss: 128.1952
Epoch 8/20
101/101 [==============================] - 152s - loss: 99.3979 - val_loss: 126.2738
Epoch 9/20
101/101 [==============================] - 155s - loss: 93.1851 - val_loss: 126.5522
Epoch 10/20
101/101 [==============================] - 148s - loss: 87.5127 - val_loss: 124.9496
Epoch 11/20
101/101 [==============================] - 150s - loss: 81.7004 - val_loss: 125.9966
Epoch 12/20
101/101 [==============================] - 145s - loss: 76.3927 - val_loss: 125.7233
Epoch 13/20
101/101 [==============================] - 151s - loss: 71.1534 - val_loss: 129.8055
Epoch 14/20
101/101 [==============================] - 150s - loss: 66.3418 - val_loss: 134.3553
Epoch 15/20
101/101 [==============================] - 150s - loss: 61.6126 - val_loss: 137.9712
```

Рис.28 Обучение сети 3 в течение 15 эпох

При изменении параметров сверточного слоя оптимальными стали параметры:

- *filters=200*
- *kernel_size=11*
- *strides=2*
- *padding='valid'*

Параметр strides может также быть также равен единице. Изменение показателя никак не повлияло на качество сети.

В нейронной сети Deep Bidirectional RNN использование большого количества слоев сети приводило к большим значениям функции потерь. Это может быть связано с небольшим количеством обучающих данных, в то время как более глубокие сети требуют большее количество данных. При обучении трехслойной сети loss функция давала значения около 500 на протяжении 20 эпох. Для пяти слоев loss функция имела значения еще больше - около 750 (и уменьшилась к 700 после 20 эпох).

Таким образом, было принято решение принять нейронную сеть номер 3 за оптимальную и использовать ее в системе распознавания как акустическую модель.

4.3 Экспериментальное сравнение качества работы систем, использующих разные методы распознавания

Построенная оптимальная архитектура нейронной сети на предыдущем этапе после обучения дает результат распознавания тестового примера, представленный на рисунке ниже.

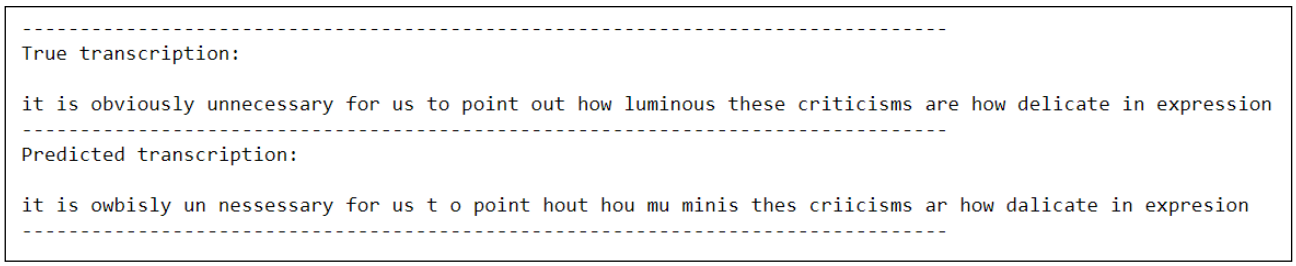

Рис.29 Пример распознавания фразы акустической моделью

При этом коэффициенты точности распознавания WER и CER составляют около WER = 40%, CER = 34% для валидационных данных.

Одним из стандартных способов улучшения результатов декодера является включение языковой модели. Добавим языковую модель для повышения точности распознавания речи. Для этого сравним два варианта ее построения.

Будем разбивать каждое предложение на слова и пытаться оптимизировать каждое. Для этого будем либо минимизировать расстояние Левенштейна, либо максимизировать коэффициент Жаккара.

На практике использование разных методов не дает больших различий. Первый метод дает немного более точный результат, поэтому было принято решение использовать его, хотя разница в качестве работы незначительная.

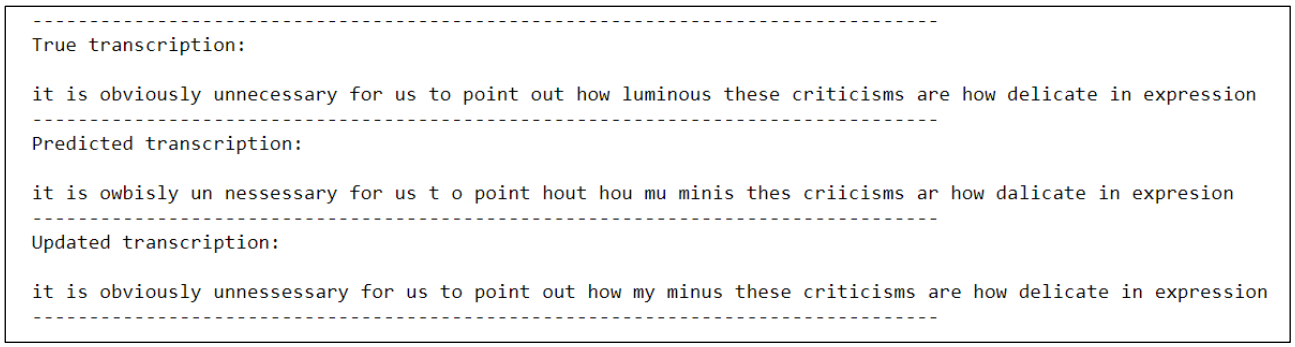

Рис.30 Пример распознавания фразы всей системой

На рисунке представлен результат распознавания фразы отдельно акустической моделью и полностью всей системой, включающей как акустическую, так и языковую

модель. Видно, что качество распознавания улучшается, исправляются мелкие опечатки и ошибки. Однако некоторые ошибки, созданные акустической моделью, могут быть ухудшены еще сильнее языковой моделью и сильно отдалить финальный результат от верного ответа (“luminous” -> “my minus”).

Для полной системы коэффициенты точности составляют по валидационным данным WER = 33%, CER = 29%, что является достаточно неплохим результатом.

4.4 Выводы по результатам экспериментальных исследований

В этом разделе было проведено экспериментальное сравнение качества работы отдельных компонентой системы распознавания речи в зависимости от выбранного метода их реализации. Была определена оптимальная нейронная сеть среди всех предложенных выше вариантов нейронных моделей для построения акустической модели на основе различия их архитектур и формы представления их обучающих данных. Также сравнивалось качество работы методов, использующихся внутри языковой модели.

Экспериментально было установлено, что нейронная сеть с архитектурой 3 дает наиболее точные предсказания и наименьшую функцию ошибки. Для улучшения точности ответа для языковой модели будем минимизировать расстояние Левенштейна. Такая система дает показатели точности WER = 33%, CER = 29%, что является неплохим результатом.

Заключение

В результате выполнения данной учебно-исследовательской работы была достигнута главная цель – создана система, выполняющая преобразование и распознавание аудиосигнала, содержащего речь человека в текстовый поток. В ходе работы были выполнены следующие задачи:

1. Изучены существующие подходы к распознаванию речи
2. Изучены основные архитектуры нейронных сетей, используемые для распознавания речи
3. Реализована архитектура нейронной сети
4. Проведены экспериментальные исследования влияния архитектуры системы на точность распознавания и спроектирована оптимальная система для данной задачи.
5. Разработана система распознавания речи.

Достигнута степень успешного распознавания речи в WER = 33%, CER = 29%. Ошибки в ответах возникают из-за малого объема данных для обучения нейронной сети. В дальнейшем для улучшения работы и повышения точности следует увеличить количество обучающих данных для качественного обучения фонетической модели. При необходимости, можно добавить компонент фильтрации аудиосигнала перед распознаванием, что позволит давать качественный результат в большем количестве практических задач. Областью применения является любая задача проектирования голосовых систем, задача ввода данных, голосового поиска.

Литература

1. И.Б. Тампель, А.А. Карпов Автоматическое распознавание речи ИТМО, 2016
2. Муаммар Аль-Шедиват Открытые проблемы в области распознавания речи. Лекция в Яндексе, 2017 URL:https://habr.com/ru/company/yandex/blog/337572/
3. S. Eddy. What is a hidden Markov model? Nature Biotechnology, 1315 – 1316, 2004
4. Neural Networks used for Speech Recognition Wouter Gevaert, Georgi Tsenov, Valeri Mladenov, Senior Member, IEEE
5. Блог компании Toshiba Распознавание речи: вводный курс, 2020, URL:https://habr.com/ru/company/toshibarus/blog/490732/
6. Niklas Donges A guide to RNN: understanding recurrent neural networks and LSTM, 2020, URL: https://builtin.com/data-science/recurrent-neural-networks-and-lstm
7. Joe Tebelskis Speech Recognition using Neural Networks, May 1995 URL:https://robotics.bstu.by/mwiki/images/0/0f/(Brain_Study)_Speech_Recognition_using_Neural_Networks.pdf
8. M.A.Anusuya, S.K.Katti Speech Recognition by Machine: A Review, 2009 URL:https://arxiv.org/pdf/1001.2267.pdf
9. Открытые лекции компании Яндекс, Распознавание речи, 2013 https://habr.com/ru/company/yandex/blog/198556/
10. Кодзасов С. В. Артикуляция // Лингвистический энциклопедический словарь / Главный редактор В. Н. Ярцева. — М.: Советская энциклопедия, 1990 https://www.fon.hum.uva.nl/praat/manual/Rabiner__1989_.html
11. L.R. Rabiner: "A tutorial on Hidden Markov Models and selected applications in speech recognition." Proceedings of the IEEE **77**: 257–286. 1989
12. Juri Lember, Alexey A. Koloydenko A generalized risk-based approach to segmentation based on hidden Markov models, URL:https://arxiv.org/pdf/1007.3622v1.pdf
13. Алборова, Ж.В.,Рубцов В.И. Алгоритм и методы распознавания речи МГТУ им. Н.Э. Баумана 2016, URL:http://ainsnt.ru/file/out/841241
14. Гребнов С.В. Аналитический обзор методов распознавания речи в системах голосового управления, 2009, URL: http://vestnik.ispu.ru/sites/vestnik.ispu.ru/files/publications/83-85_0.pdf
15. Wikipedia, URL:https://ru.wikipedia.org/wiki/N-грамма (дата обращения 20.10.2020)
16. Andrew Gibiansky Speech Recognition with Neural Networks, 2014 URL:https://andrew.gibiansky.com/blog/machine-learning/speech-recognition-neural-networks/

17. Alex Graves, Abdel-rahman Mohamed and Geoffrey Hinton Speech recognition with deep recurrent neural networks, University of Toronto URL:https://arxiv.org/pdf/1303.5778.pdf

18. Pablo Gimeno Jordán A Voice Activity Detection Tool Based on Recurrent Neural Networks, 2019 URL:https://medium.com/vivolab/vivovad-a-voice-activity-detection-tool-based-on-recurrent-neural-networks-32356526321c

19. Andrey “Sovse” Rus speech recognition https://github.com/sovse/Rus-SpeechRecognition-LSTM-CTC-VoxForge

20. Mike Schuster, Kuldip K. Paliwal Bidirectional recurrent neural networks, 1997, URL:https://www.researchgate.net/publication/3316656_Bidirectional_recurrent_neural_networks

21. Ivan Ozhiganov How Neural Networks Recognize Speech-to-Text, 2019, URL:https://dzone.com/articles/how-to-train-a-neural-network-to-recognize-speech

22. Xuejiao Li, Zixuan Zhou Speech Command Recognition with Convolutional Neural Network URL:http://cs229.stanford.edu/proj2017/final-reports/5244201.pdf

23. Dimitri Palaz, Mathew Magimai-Doss, Ronan Collobert Convolutional neural networks-based continuous speech recognition using raw speech signal https://ronan.collobert.com/pub/matos/2015_rawspeech_icassp.pdf

24. А.В. Фролов, Г.В. Фролов Синтез и распознавание речи. Современные решения URL:https://frolov-lib.ru/books/hi/ch05.html

25. Ле, Нгуен Виен. Распознавание речи на основе искусственных нейронных сетей (г. Москва, май 2011 г.). С. 8-11. — URL: https://moluch.ru/conf/tech/archive/3/712/ (дата обращения: 20.10.2020).

26. Максим Девятов Программы распознавания речи. Что понимает искусственный интеллект, 2017, URL:https://www.primavista.ru/blog/2017/08/03/programmyi-raspoznavaniya-rechi

27. Abid Mohsin Podcast Transcription Benchmark, 2019. URL:https://www.rev.ai/blog/podcast-transcription-benchmark-part-1/

Приложения

Приложение 1. Код AudioGenerator

```
1. import json
2. import numpy as np
3. import random
4. from python_speech_features import mfcc
5. import librosa
6. import scipy.io.wavfile as wav
7. import matplotlib.pyplot as plt
8. from mpl_toolkits.axes_grid1 import make_axes_locatable
9. from data_utils import calc_feat_dim, spectrogram_from_file, text_to_int_sequence
10. from data_utils import conv_output_length
11.
12. RNG_SEED = 327
13.
14. class AudioGenerator():
15.     def __init__(self, step=10, window=20, max_freq=8000,
    mfcc_dim=13,minibatch_size=20, desc_file=None, spectrogram=True,
    max_duration=10.0,
16.         sort_by_duration=False):
17.         self.feat_dim = calc_feat_dim(window, max_freq)
18.         self.mfcc_dim = mfcc_dim
19.         self.feats_mean = np.zeros((self.feat_dim,))
20.         self.feats_std = np.ones((self.feat_dim,))
21.         self.rng = random.Random(RNG_SEED)
22.         if desc_file is not None:
23.             self.load_metadata_from_desc_file(desc_file)
24.         self.step = step
25.         self.window = window
26.         self.max_duration = max_duration
27.         self.minibatch_size = minibatch_size
28.         self.spectrogram = spectrogram
29.         self.sort_by_duration = sort_by_duration
30.         self.max_freq = max_freq
31.         self.cur_train_index = 0
32.         self.cur_valid_index = 0
33.         self.cur_test_index = 0
34.
35.     def get_batch(self, partition):
36.         if partition == 'train':
37.             audio_paths = self.train_audio_paths
38.             cur_index = self.cur_train_index
39.             texts = self.train_texts
40.         elif partition == 'valid':
41.             audio_paths = self.valid_audio_paths
42.             cur_index = self.cur_valid_index
43.             texts = self.valid_texts
44.         elif partition == 'test':
```

```
45.             audio_paths = self.test_audio_paths
46.             cur_index = self.test_valid_index
47.             texts = self.test_texts
48.         else:
49.             raise Exception("Invalid partition. Must be train/validation")
50.         features = [self.normalize(self.featurize(a)) for a in
51.             audio_paths[cur_index:cur_index+self.minibatch_size]]
52.         max_length = max([features[i].shape[0]
53.             for i in range(0, self.minibatch_size)])
54.         max_string_length = max([len(texts[cur_index+i])
55.
56.             for i in range(0, self.minibatch_size)])
57.         X_data = np.zeros([self.minibatch_size, max_length,
58.             self.feat_dim*self.spectrogram + self.mfcc_dim*(not
    self.spectrogram)])
59.         labels = np.ones([self.minibatch_size, max_string_length]) * 28
60.         input_length = np.zeros([self.minibatch_size, 1])
61.         label_length = np.zeros([self.minibatch_size, 1])
62.
63.         for i in range(0, self.minibatch_size):
64.             feat = features[i]
65.             input_length[i] = feat.shape[0]
66.             X_data[i, :feat.shape[0], :] = feat
67.             label = np.array(text_to_int_sequence(texts[cur_index+i]))
68.             labels[i, :len(label)] = label
69.             label_length[i] = len(label)
70.
71.         outputs = {'ctc': np.zeros([self.minibatch_size])}
72.         inputs = {'the_input': X_data,
73.                   'the_labels': labels,
74.                   'input_length': input_length,
75.                   'label_length': label_length
76.                  }
77.         return (inputs, outputs)
78.
79.     def shuffle_data_by_partition(self, partition):
80.         if partition == 'train':
81.             self.train_audio_paths, self.train_durations, self.train_texts =
    shuffle_data(
82.                 self.train_audio_paths, self.train_durations, self.train_texts)
83.         elif partition == 'valid':
84.             self.valid_audio_paths, self.valid_durations, self.valid_texts =
    shuffle_data(
85.                 self.valid_audio_paths, self.valid_durations, self.valid_texts)
86.         else:
87.             raise Exception("Invalid partition. Must be train/validation")
88.
89.     def sort_data_by_duration(self, partition):
90.         if partition == 'train':
```

```
91.             self.train_audio_paths, self.train_durations, self.train_texts =
    sort_data(
92.                 self.train_audio_paths, self.train_durations, self.train_texts)
93.         elif partition == 'valid':
94.             self.valid_audio_paths, self.valid_durations, self.valid_texts =
    sort_data(
95.                 self.valid_audio_paths, self.valid_durations, self.valid_texts)
96.         else:
97.             raise Exception("Invalid partition. "
98.                 "Must be train/validation")
99.
100.         def next_train(self):
101.             while True:
102.                 ret = self.get_batch('train')
103.                 self.cur_train_index += self.minibatch_size
104.                 if self.cur_train_index >= len(self.train_texts) -
    self.minibatch_size:
105.                     self.cur_train_index = 0
106.                     self.shuffle_data_by_partition('train')
107.                 yield ret
108.
109.         def next_valid(self):
110.             while True:
111.                 ret = self.get_batch('valid')
112.                 self.cur_valid_index += self.minibatch_size
113.                 if self.cur_valid_index >= len(self.valid_texts) -
    self.minibatch_size:
114.                     self.cur_valid_index = 0
115.                     self.shuffle_data_by_partition('valid')
116.                 yield ret
117.
118.         def next_test(self):
119.             while True:
120.                 ret = self.get_batch('test')
121.                 self.cur_test_index += self.minibatch_size
122.                 if self.cur_test_index >= len(self.test_texts) -
    self.minibatch_size:
123.                     self.cur_test_index = 0
124.                 yield ret
125.
126.         def load_train_data(self, desc_file='train_corpus.json'):
127.             self.load_metadata_from_desc_file(desc_file, 'train')
128.             self.fit_train()
129.             if self.sort_by_duration:
130.                 self.sort_data_by_duration('train')
131.
132.         def load_validation_data(self, desc_file='valid_corpus.json'):
133.             self.load_metadata_from_desc_file(desc_file, 'validation')
134.             if self.sort_by_duration:
135.                 self.sort_data_by_duration('valid')
```

```
136.
137.         def load_test_data(self, desc_file='test_corpus.json'):
138.             self.load_metadata_from_desc_file(desc_file, 'test')
139.
140.         def load_metadata_from_desc_file(self, desc_file, partition):
141.             audio_paths, durations, texts = [], [], []
142.             with open(desc_file) as json_line_file:
143.                 for line_num, json_line in enumerate(json_line_file):
144.                     try:
145.                         spec = json.loads(json_line)
146.                         if float(spec['duration']) > self.max_duration:
147.                             continue
148.                         audio_paths.append(spec['key'])
149.                         durations.append(float(spec['duration']))
150.                         texts.append(spec['text'])
151.                     except Exception as e:
152.                         print('Error reading line #{}: {}'
153.                                     .format(line_num, json_line))
154.             if partition == 'train':
155.                 self.train_audio_paths = audio_paths
156.                 self.train_durations = durations
157.                 self.train_texts = texts
158.             elif partition == 'validation':
159.                 self.valid_audio_paths = audio_paths
160.                 self.valid_durations = durations
161.                 self.valid_texts = texts
162.             elif partition == 'test':
163.                 self.test_audio_paths = audio_paths
164.                 self.test_durations = durations
165.                 self.test_texts = texts
166.             else:
167.                 raise Exception("Invalid partition to load metadata. Must be
    train/validation/test")
168.
169.         def fit_train(self, k_samples=100):
170.             k_samples = min(k_samples, len(self.train_audio_paths))
171.             samples = self.rng.sample(self.train_audio_paths, k_samples)
172.             feats = [self.featurize(s) for s in samples]
173.             feats = np.vstack(feats)
174.             self.feats_mean = np.mean(feats, axis=0)
175.             self.feats_std = np.std(feats, axis=0)
176.
177.         def featurize(self, audio_clip):
178.             if self.spectrogram:
179.                 return spectrogram_from_file(
180.                     audio_clip, step=self.step, window=self.window,
181.                     max_freq=self.max_freq)
182.             else:
183.                 (rate, sig) = wav.read(audio_clip)
184.                 return mfcc(sig, rate, numcep=self.mfcc_dim)
```

```
185.
186.          def normalize(self, feature, eps=1e-14):
187.              return (feature - self.feats_mean) / (self.feats_std + eps)
188.
189.      def shuffle_data(audio_paths, durations, texts):
190.          p = np.random.permutation(len(audio_paths))
191.          audio_paths = [audio_paths[i] for i in p]
192.          durations = [durations[i] for i in p]
193.          texts = [texts[i] for i in p]
194.          return audio_paths, durations, texts
195.
196.      def sort_data(audio_paths, durations, texts):
197.          p = np.argsort(durations).tolist()
198.          audio_paths = [audio_paths[i] for i in p]
199.          durations = [durations[i] for i in p]
200.          texts = [texts[i] for i in p]
201.          return audio_paths, durations, texts
202.
203.      def vis_train_features(index=0):
204.          audio_gen = AudioGenerator(spectrogram=True)
205.          audio_gen.load_train_data()
206.          vis_audio_path = audio_gen.train_audio_paths[index]
207.          vis_spectrogram_feature =
    audio_gen.normalize(audio_gen.featurize(vis_audio_path))
208.          audio_gen = AudioGenerator(spectrogram=False)
209.          audio_gen.load_train_data()
210.          vis_mfcc_feature =
    audio_gen.normalize(audio_gen.featurize(vis_audio_path))
211.          vis_text = audio_gen.train_texts[index]
212.          vis_raw_audio, _ = librosa.load(vis_audio_path)
213.          return vis_text, vis_raw_audio, vis_mfcc_feature,
    vis_spectrogram_feature, vis_audio_path
214.
215.
216.      def plot_raw_audio(vis_raw_audio):
217.          # plot the raw audio signal
218.          fig = plt.figure(figsize=(12,3))
219.          ax = fig.add_subplot(111)
220.          steps = len(vis_raw_audio)
221.          ax.plot(np.linspace(1, steps, steps), vis_raw_audio)
222.          plt.title('Audio Signal')
223.          plt.xlabel('Time')
224.          plt.ylabel('Amplitude')
225.          plt.show()
226.
227.      def plot_mfcc_feature(vis_mfcc_feature):
228.          fig = plt.figure(figsize=(12,5))
229.          ax = fig.add_subplot(111)
230.          im = ax.imshow(vis_mfcc_feature, cmap=plt.cm.jet, aspect='auto')
231.          plt.title('Normalized MFCC')
```

```
232.         plt.ylabel('Time')
233.         plt.xlabel('MFCC Coefficient')
234.         divider = make_axes_locatable(ax)
235.         cax = divider.append_axes("right", size="5%", pad=0.05)
236.         plt.colorbar(im, cax=cax)
237.         ax.set_xticks(np.arange(0, 13, 2), minor=False);
238.         plt.show()
239.
240.     def plot_spectrogram_feature(vis_spectrogram_feature):
241.         fig = plt.figure(figsize=(12,5))
242.         ax = fig.add_subplot(111)
243.         im = ax.imshow(vis_spectrogram_feature, cmap=plt.cm.jet, aspect='auto')
244.         plt.title('Normalized Spectrogram')
245.         plt.ylabel('Time')
246.         plt.xlabel('Frequency')
247.         divider = make_axes_locatable(ax)
248.         cax = divider.append_axes("right", size="5%", pad=0.05)
249.         plt.colorbar(im, cax=cax)
250.         plt.show()
251.
252.
```

Приложение 2. Код DataUtils

```
import numpy as np
import soundfile
from numpy.lib.stride_tricks import as_strided
from char_map import char_map, index_map

def calc_feat_dim(window, max_freq):
    return int(0.001 * window * max_freq) + 1

def conv_output_length(input_length, filter_size, border_mode, stride,
dilation=1):
    if input_length is None:
        return None
    assert border_mode in {'same', 'valid'}
    dilated_filter_size = filter_size + (filter_size - 1) * (dilation - 1)
    if border_mode == 'same':
        output_length = input_length
    elif border_mode == 'valid':
        output_length = input_length - dilated_filter_size + 1
    return (output_length + stride - 1) // stride


def spectrogram(samples, fft_length=256, sample_rate=2, hop_length=128):
    assert not np.iscomplexobj(samples), "Must not pass in complex numbers"
    window = np.hanning(fft_length)[:, None]
    window_norm = np.sum(window**2)
    scale = window_norm * sample_rate

    trunc = (len(samples) - fft_length) % hop_length
    x = samples[:len(samples) - trunc]
    nshape = (fft_length, (len(x) - fft_length) // hop_length + 1)
    nstrides = (x.strides[0], x.strides[0] * hop_length)
    x = as_strided(x, shape=nshape, strides=nstrides)
    assert np.all(x[:, 1] == samples[hop_length:(hop_length + fft_length)])
    x = np.fft.rfft(x * window, axis=0)
    x = np.absolute(x)**2
    x[1:-1, :] *= (2.0 / scale)
    x[(0, -1), :] /= scale
    freqs = float(sample_rate) / fft_length * np.arange(x.shape[0])
    return x, freqs


def spectrogram_from_file(filename, step=10, window=20, max_freq=None, eps=1e-14):
    with soundfile.SoundFile(filename) as sound_file:
        audio = sound_file.read(dtype='float32')
        sample_rate = sound_file.samplerate
        if audio.ndim >= 2:
            audio = np.mean(audio, 1)
```

```
47.         if max_freq is None:
48.             max_freq = sample_rate / 2
49.         if max_freq > sample_rate / 2:
50.             raise ValueError("max_freq must not be greater than half of sample
    rate")
51.         if step > window:
52.             raise ValueError("step size must not be greater than window size")
53.         hop_length = int(0.001 * step * sample_rate)
54.         fft_length = int(0.001 * window * sample_rate)
55.         pxx, freqs = spectrogram(
56.             audio, fft_length=fft_length, sample_rate=sample_rate,
57.             hop_length=hop_length)
58.         ind = np.where(freqs <= max_freq)[0][-1] + 1
59.     return np.transpose(np.log(pxx[:ind, :] + eps))
60.
61. def text_to_int_sequence(text):
62.     int_sequence = []
63.     for c in text:
64.         if c == ' ':
65.             ch = char_map['<SPACE>']
66.         else:
67.             ch = char_map[c]
68.         int_sequence.append(ch)
69.     return int_sequence
70.
71. def int_sequence_to_text(int_sequence):
72.     text = []
73.     for c in int_sequence:
74.         ch = index_map[c]
75.         text.append(ch)
76.     return text
77.
```

Приложение 3. Код TrainUtils

```
from data_generator import AudioGenerator
import _pickle as pickle

from keras import backend as K
from keras.models import Model
from keras.layers import (Input, Lambda, BatchNormalization)
from keras.optimizers import SGD, RMSprop
from keras.callbacks import ModelCheckpoint
import os

def ctc_lambda_func(args):
    y_pred, labels, input_length, label_length = args
    return K.ctc_batch_cost(labels, y_pred, input_length, label_length)

def add_ctc_loss(input_to_softmax):
    the_labels = Input(name='the_labels', shape=(None,), dtype='float32')
    input_lengths = Input(name='input_length', shape=(1,), dtype='int64')
    label_lengths = Input(name='label_length', shape=(1,), dtype='int64')
    output_lengths = Lambda(input_to_softmax.output_length)(input_lengths)
    loss_out = Lambda(ctc_lambda_func, output_shape=(1,), name='ctc')(
        [input_to_softmax.output, the_labels, output_lengths, label_lengths])
    model = Model(
        inputs=[input_to_softmax.input, the_labels, input_lengths, label_lengths],
        outputs=loss_out)
    return model

def train_model(input_to_softmax,
                pickle_path,
                save_model_path,
                train_json='train_corpus.json',
                valid_json='valid_corpus.json',
                minibatch_size=20,
                spectrogram=True,
                mfcc_dim=13,
                optimizer=SGD(lr=0.02, decay=1e-6, momentum=0.9, nesterov=True,
clipnorm=5),
                epochs=20,
                verbose=1,
                sort_by_duration=False,
                max_duration=10.0):

    audio_gen = AudioGenerator(minibatch_size=minibatch_size,
spectrogram=spectrogram, mfcc_dim=mfcc_dim,
        max_duration=max_duration, sort_by_duration=sort_by_duration)
    audio_gen.load_train_data(train_json)
    audio_gen.load_validation_data(valid_json)
    num_train_examples=len(audio_gen.train_audio_paths)
```

```
    steps_per_epoch = num_train_examples//minibatch_size
    num_valid_samples = len(audio_gen.valid_audio_paths)
    validation_steps = num_valid_samples//minibatch_size
    model = add_ctc_loss(input_to_softmax)
    model.compile(loss={'ctc': lambda y_true, y_pred: y_pred},
optimizer=optimizer)
    if not os.path.exists('results'):
        os.makedirs('results')

    checkpointer = ModelCheckpoint(filepath='results/'+save_model_path, verbose=0)
    hist = model.fit_generator(generator=audio_gen.next_train(),
steps_per_epoch=steps_per_epoch,
        epochs=epochs, validation_data=audio_gen.next_valid(),
validation_steps=validation_steps,
        callbacks=[checkpointer], verbose=verbose)

    with open('results/'+pickle_path, 'wb') as f:
        pickle.dump(hist.history, f)
```

Приложение 4. Код Models

```
from keras import backend as K
from keras.models import Model
from keras.layers import (BatchNormalization, Conv1D, Dense, Input,
    TimeDistributed, Activation, Bidirectional, SimpleRNN, GRU, LSTM)

class Models():
    def simple_rnn_model(input_dim, output_dim=29):
        input_data = Input(name='the_input', shape=(None, input_dim))
        simp_rnn = GRU(output_dim, return_sequences=True,
                       implementation=2, name='rnn')(input_data)
        y_pred = Activation('softmax', name='softmax')(simp_rnn)
        model = Model(inputs=input_data, outputs=y_pred)
        model.output_length = lambda x: x
        print(model.summary())
        return model

    def rnn_model(input_dim, units, activation, output_dim=29):
        input_data = Input(name='the_input', shape=(None, input_dim))
        simp_rnn = LSTM(units, activation=activation,
            return_sequences=True, implementation=2, name='rnn')(input_data)
        bn_rnn = BatchNormalization(name='bn_rnn_1d')(simp_rnn)
        time_dense = TimeDistributed(Dense(output_dim))(bn_rnn)
        y_pred = Activation('softmax', name='softmax')(time_dense)
        model = Model(inputs=input_data, outputs=y_pred)
        model.output_length = lambda x: x
        print(model.summary())
        return model


    def cnn_rnn_model(input_dim, filters, kernel_size, conv_stride,
        conv_border_mode, units, output_dim=29):
        input_data = Input(name='the_input', shape=(None, input_dim))
        conv_1d = Conv1D(filters, kernel_size,
                         strides=conv_stride,
                         padding=conv_border_mode,
                         activation='relu',
                         name='conv1d')(input_data)
        bn_cnn = BatchNormalization(name='bn_conv_1d')(conv_1d)
        simp_rnn = GRU(units, activation='relu',
            return_sequences=True, implementation=2, name='rnn')(bn_cnn)
        bn_rnn = BatchNormalization(name='bn_rnn_1d')(simp_rnn)
        time_dense = TimeDistributed(Dense(output_dim))(bn_rnn)
        y_pred = Activation('softmax', name='softmax')(time_dense)
        model = Model(inputs=input_data, outputs=y_pred)
        model.output_length = lambda x: cnn_output_length(
            x, kernel_size, conv_border_mode, conv_stride)
        print(model.summary())
```

```
48.         return model
49.
50.     def cnn_output_length(input_length, filter_size, border_mode, stride,
    dilation=1):
51.         if input_length is None:
52.             return None
53.         assert border_mode in {'same', 'valid'}
54.         dilated_filter_size = filter_size + (filter_size - 1) * (dilation - 1)
55.         if border_mode == 'same':
56.             output_length = input_length
57.         elif border_mode == 'valid':
58.             output_length = input_length - dilated_filter_size + 1
59.         return (output_length + stride - 1) // stride
60.
61.     def deep_rnn_model(input_dim, units, recur_layers, output_dim = 29):
62.         input_data = Input(name = 'the_input', shape=(None, input_dim))
63.         if recur_layers == 1:
64.             layer = LSTM(units, return_sequences=True,
    activation='relu')(input_data)
65.             layer = BatchNormalization(name='bt_rnn_1')(layer)
66.         else:
67.             layer = LSTM(units, return_sequences=True,
    activation='relu')(input_data)
68.             layer = BatchNormalization(name='bt_rnn_1')(layer)
69.
70.             for i in range(recur_layers - 2):
71.                 layer = LSTM(units, return_sequences=True,
    activation='relu')(layer)
72.                 layer = BatchNormalization(name='bt_rnn_{}'.format(2+i))(layer)
73.
74.             layer = LSTM(units, return_sequences=True, activation='relu')(layer)
75.             layer = BatchNormalization(name='bt_rnn_last_rnn')(layer)
76.
77.         time_dense = TimeDistributed(Dense(output_dim))(layer)
78.         y_pred = Activation('softmax', name='softmax')(time_dense)
79.         model = Model(inputs=input_data, outputs=y_pred)
80.         model.output_length = lambda x: x
81.         print(model.summary())
82.         return model
83.
84.     def bidirectional_rnn_model(input_dim, units, output_dim=29):
85.         input_data = Input(name='the_input', shape=(None, input_dim))
86.         bidir_rnn = Bidirectional(LSTM(units, return_sequences=True,
    activation='relu'), merge_mode='concat')(input_data)
87.         time_dense = TimeDistributed(Dense(output_dim))(bidir_rnn)
88.         y_pred = Activation('softmax', name='softmax')(time_dense)
89.         model = Model(inputs=input_data, outputs=y_pred)
90.         model.output_length = lambda x: x
91.         print(model.summary())
92.         return model
```

```
93.
94.     def final_model(input_dim, filters, kernel_size, conv_stride,
95.         conv_border_mode, units, output_dim=29, dropout_rate=0.5,
    number_of_layers=2,
96.         cell=GRU, activation='tanh'):
97.         input_data = Input(name='the_input', shape=(None, input_dim))
98.         conv_1d = Conv1D(filters, kernel_size,
99.                          strides=conv_stride,
100.                             padding=conv_border_mode,
101.                             activation='relu',
102.                             name='layer_1_conv',
103.                             dilation_rate=1)(input_data)
104.             conv_bn = BatchNormalization(name='conv_batch_norm')(conv_1d)
105.             if number_of_layers == 1:
106.                 layer = cell(units, activation=activation,
107.                     return_sequences=True, implementation=2, name='rnn_1',
    dropout=dropout_rate)(conv_bn)
108.                 layer = BatchNormalization(name='bt_rnn_1')(layer)
109.             else:
110.                 layer = cell(units, activation=activation,
111.                              return_sequences=True, implementation=2,
    name='rnn_1', dropout=dropout_rate)(conv_bn)
112.                 layer = BatchNormalization(name='bt_rnn_1')(layer)
113.
114.                 for i in range(number_of_layers - 2):
115.                     layer = cell(units, activation=activation,
116.                                  return_sequences=True, implementation=2,
    name='rnn_{}'.format(i+2), dropout=dropout_rate)(layer)
117.                     layer =
    BatchNormalization(name='bt_rnn_{}'.format(i+2))(layer)
118.
119.                 layer = cell(units, activation=activation,
120.                              return_sequences=True, implementation=2,
    name='final_layer_of_rnn')(layer)
121.                 layer = BatchNormalization(name='bt_rnn_final')(layer)
122.             time_dense = TimeDistributed(Dense(output_dim))(layer)
123.             y_pred = Activation('softmax', name='softmax')(time_dense)
124.             model = Model(inputs=input_data, outputs=y_pred)
125.             model.output_length = lambda x: cnn_output_length(
126.                 x, kernel_size, conv_border_mode, conv_stride)
127.             print(model.summary())
128.             return model
```

Приложение 5. Код LanguageModel

```
1. from __future__ import division
2. import numpy as np
3. import nltk
4. from nltk.metrics import *
5. from nltk.util import ngrams
6. import enchant
7. from enchant.checker import SpellChecker
8. from nltk.stem import PorterStemmer
9. from nltk.corpus import words
10.
11.spell_dictionary = enchant.Dict('en')
12.class LanguageModel:
13.    def __init__(self,dictionary):
14.        self.dictionary = dictionary
15.        self.check = SpellChecker("en_US")
16.        self.stemmer = PorterStemmer()
17.
18.    def levenshtein_distance(self,s1,s2):
19.        distance_btw_strings = edit_distance(s1,s2)
20.        return distance_btw_strings
21.
22.    def ngram(self,word,n):
23.        grams = list(ngrams(word,n))
24.        return grams
25.
26.    def check_mistakes_in_sentence(self,sentence):
27.        misspelled_words = []
28.        self.check.set_text(sentence)
29.        for err in self.check:
30.            misspelled_words.append(err.word)
31.
32.        if len(misspelled_words) == 0:
33.            print " No mistakes found"
34.        return misspelled_words
35.
36.    def jaccard(self,a,b):
37.        union = list(set(a+b))
38.        intersection = list(set(a) - (set(a)-set(b)))
39.        jaccard_coeff = float(len(intersection))/len(union)
40.        return jaccard_coeff
41.
42.    def minimumEditDistance_spell_corrector(self,word):
43.        max_distance = 2
44.        if (self.dictionary.check(word)):
45.            return word
46.        suggested_words = self.suggest_words(word)
47.        num_modified_characters = []
```

```
48.         if suggested_words != 0:
49.             for sug_words in suggested_words:
50.
    num_modified_characters.append(self.levenshtein_distance(word,sug_words))
51.             minimum_edit_distance = min(num_modified_characters)
52.             best_arg = num_modified_characters.index(minimum_edit_distance)
53.             if max_distance > minimum_edit_distance:
54.                 best_suggestion = suggested_words[best_arg]
55.                 return best_suggestion
56.             else:
57.                 return word
58.         else:
59.             return word
60.
61.
62.     def ngram_spell_corrector(self,word):
63.         max_distance = 2
64.         if (self.dictionary.check(word)):
65.             return word
66.         suggested_words = self.suggest_words(word)
67.         num_modified_characters = []
68.         max_jaccard = []
69.         list_of_sug_words = []
70.         if suggested_words != 0:
71.             word_ngrams = self.ngram(word,2)
72.             for sug_words in suggested_words:
73.                 if (self.levenshtein_distance(word,sug_words)) < 3 :
74.                     sug_ngrams = self.ngram(sug_words,2)
75.                     jac = self.jaccard(word_ngrams,sug_ngrams)
76.                     max_jaccard.append(jac)
77.                     list_of_sug_words.append(sug_words)
78.             highest_jaccard = max(max_jaccard)
79.             best_arg = max_jaccard.index(highest_jaccard)
80.             word = list_of_sug_words[best_arg]
81.             return word
82.         else:
83.             return word
84.
```