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

Институт математики и фундаментальной информатики
Базовая кафедра вычислительных и информационных технологий

УТВЕРЖДАЮ
Заведующий кафедрой
_____ /В.В.Шайдуров

«___» _____2019 г.

БАКАЛАВРСКАЯ РАБОТА

Направление 02.03.01 Математика и компьютерные науки

ПОСТРОЕНИЕ СЛОИСТОЙ НЕЙРОННОЙ СЕТИ ДЛЯ РАСПОЗНАВАНИЯ РЕЧИ

Научный руководитель
кандидат технических наук,
доцент _____/ С.В. Исаев

Выпускник _____/ А.А. Димов

Красноярск 2019

РЕФЕРАТ

Выпускная квалификационная работа по теме «Построение слоистой нейронной сети для распознавания речи» содержит 46 страниц текста, 27 использованных источников.

НЕЙРОННАЯ СЕТЬ, МАШИННОЕ ОБУЧЕНИЕ, РАСПОЗНАВАНИЕ РЕЧИ, СВЕРТОЧНАЯ СЕТЬ, РЕКУРРЕНТНАЯ СЕТЬ, ОСТАТОЧНОЕ ОБУЧЕНИЕ, МАРКОВСКИЕ МОДЕЛИ.

Цель работы – реализация нейронной сети для распознавания человеческой речи.

Выпускная квалификационная работа представлена тремя разделами:

- В первом разделе проводится обзор современных исследований, рассматривается краткая история развития распознавания речи.
- Во втором разделе рассматривается основная теория и строится архитектура нейронной сети данной работы.
- В третьем происходит выбор инструментов для реализации нейронной сети, проводится предобработка данных и описывается процесс обучения, приводятся результаты работы и сравнения их с другими исследованиями.

В результате работы разработана нейронная сеть, обучена модель, способная распознавать человеческую речь по записи голоса, также рассмотрена математическая теория, стоящая за алгоритмами обработки аудиопотока.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На данный момент задача по распознаванию человеческой речи является одной из самых актуальных в современном мире. Миллиарды людей по всему миру используют распознавание речи в своей повседневной жизни, начиная с голосового поиска по сети Интернет, общения с голосовым ассистентом, управление умным домом и заканчивая валидацией пользователя для доступа к его личной информации, и сервисами для людей с ограниченными возможностями. С каждым годом новые подходы к решению данной задачи все больше улучшают точность распознавания, вплотную приближаясь по этому показателю к человеку, однако, до сих пор не догоняя ее, но тенденция такова, что преодоление этого порога не за горами. На текущий момент лучшим результатом в этой области является работа [1].

Целью данной работы является создание слоистой нейронной сети для распознавания речи. Задачи работы: изучить современные подходы к решению проблемы, переменяемые инструменты, на основе существующих исследований построить оригинальную архитектуру, которая вбирает в себя из них свои преимущества, чтобы добиться желаемого результата, при этом, необходимо учесть ограниченность в вычислительных мощностях, чтобы нейронная сеть смогла обучиться даже на довольно ограниченных вычислительных ресурсах, что является нетипичной задачей, так как распознавание речи является самой по себе вычислительно трудной проблемой.

1 Обзор современных исследований в области распознавания речи

1.1. Краткая история развития распознавания речи

Определение 1.1.1 Искусственная нейронная сеть — математическая модель, а также её программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей — сетей нервных клеток живого организма.

Первые серьезные достижения для решения задачи по распознавание речи были получены с применением теории марковских случайных процессов, в частности скрытых марковских моделей. Однако, с развитием моделей распознавания использование только СММ не давало прироста в точности.

Следующим серьезным шагом на пути увеличения точности стали рекуррентные нейронные сети, которые могут обрабатывать последовательность событий, при этом используя собственную память для вычисления следующих шагов. Несмотря на то, что данные сети действительно эффективны у них имеется существенная проблема в виде исчезающего градиента. Поэтому, было разработано множество вариаций архитектур на основе РНС, которые были призваны разрешить проблемы оригинала, одним из таких решений стала LSTM, которая уже не имела проблем с градиентом и могла запоминать довольно длинные последовательности событий. Однако, в 2013 году данные сети достигли порога ошибки в распознавании речи в 17.7%.

На данный момент все современные нейронные сети имеют в основе не единственную архитектуру, а композицию нескольких разных структур. Так, например, в 2015 году исследователи из Microsoft смогли уменьшить ошибку распознавания до 8% в работе [2] с помощью модели, основанной на взаимодействии развернутой рекуррентной и сверточной нейронной сетях. В 2016 году эту ошибку удалось снизить до 5.8% уже на основе сверточной, LSTM и LACE сетях, описанных в исследовании [3]. А в 2017 году был достигнут результат с 5.1%, данная модель уже может учитывать контекст сказанной

фразы, чтобы не путать слова сходные по звучанию. Хотя и на данный момент эта модель все равно проигрывает человеку в точности распознавания, она показывает впечатляющие результаты.

В данной работе была реализована архитектура сети, базирующаяся на сверточных нейронный сетях, остаточном обучении и логике работы рекуррентных сетей.

1.2 Скрытые Марковские модели

Первые серьезные достижения в области распознавания речи в 80-х годах XX века были связаны со скрытой марковской моделью (СММ). Данный подход представлял собой переход от простых методов распознавания образов, основанных на шаблонах и измерении спектрального состояния к статистическому методу распознавания речи, что привело к значительному скачку в точности.

По своей сути СММ — это статистическая модель, в которой моделируемая система считается марковским процессом с неизвестными параметрами. Задача состоит в том, чтобы определить скрытые параметры из наблюдаемых данных. В скрытой марковской модели состояние не отображается напрямую, но переменные, на которые влияет состояние, являются видимыми. Каждое состояние имеет распределение вероятностей по возможным выходным токенам (словам). Следовательно, последовательность токенов, генерируемых СММ, дает некоторую информацию о последовательности состояний.

СММ создает стохастические модели из известных выражений и сравнивает вероятность того, что неизвестное выражение было сгенерировано каждой моделью. При этом используется статистическая теория, чтобы упорядочить векторы признаков в матрицу Маркова (цепи), в которой хранятся вероятности переходов состояний. То есть, если бы каждое из кодовых слов

представляло какое-то состояние, СММ следовала бы последовательности изменений состояния и строила модель, которая включает в себя вероятности перехода каждого состояния в другое состояние.

Скрытые Марковские модели так популярны, потому что их можно обучать автоматически, также они просты в использовании. СММ рассматривает речевой сигнал как квазистатический в течение коротких периодов времени и моделирует фреймы для распознавания. Она разбивает вектор признаков сигнала на несколько состояний и определяет вероятность перехода сигнала из одного состояния в другое.

Скрытые Марковские модели описываются следующим образом:

1. N — количество состояний модели, $S = \{S_1, S_2, \dots, S_N\}$ — множество состояний модели, q_t — состояние в момент времени t.

2. M — количество различных символов в наблюдаемой последовательности, $V = \{v_1, v_2, \dots, v_M\}$ - алфавит данной последовательности.

3. $A = \{a_{ij}\}$ — распределение вероятностей перехода состояний, определяемое матрицей $N \times N$.

$$a_{ij} = P[q_{t+1} = S_j | q_t = S_i], 1 \le i, j \le N, \quad (1)$$

$$\sum a_{ij} = 1, 1 \le i, j \le N. \quad (2)$$

4. $B = \{b_j(k)\}$ — матрица распределения вероятностей появления символов в состоянии j.

$$b_j(k) = P[v_k | q_t = S_j], 1 \ge j \ge N, 1 \ge k \ge M, \quad (3)$$

$$\sum b_j(k) = 1, 1 \geq k \geq M. \quad (4)$$

5. $\pi = \{\pi_i\}$ — матрица распределения вероятностей начального состояния.

$$\pi_i = P[q_1 = S_i], 1 \leq i \leq N. \quad (5)$$

N, M, A, B и π определяют скрытую Марковскую модель, которая выдает наблюдаемую последовательность $O = O_1 O_2 \dots O_T$ для всей модели как $\lambda = (A, B, \pi)$, которые определяют полный набор параметров модели.

1.2 Рекуррентные нейронные сети

Самым распространенным алгоритмом для распознавания речи являются рекуррентные сети (Recurrent neural network; RNN) и его улучшенные разновидности долгая краткосрочная память (Long short-term memory; LSTM) и управляемые рекуррентные блоки (Gated Recurrent Units; GRU).

Суть рекуррентных сетей заключается в использовании последовательных цепочек информации. Данные сети имеют так называемую “память”, которая “помнит” значение предыдущего элемента в цепи и основываясь на этом может делать вывод о виде следующего элемента, что особенно полезно в распознавании речи, так как наша речь — это последовательность слов или символов (далее мы будем рассматривать речь именно как последовательность символов). Однако, на практике рекуррентные сети могут учитывать только лишь небольшое количество предыдущих элементов.

Из рис. 1 легко понять, что для каждого отдельного элемента будет свой слой, т.е., например, если слово состоит из 3 букв, то и сеть развернется на 3 слоя.

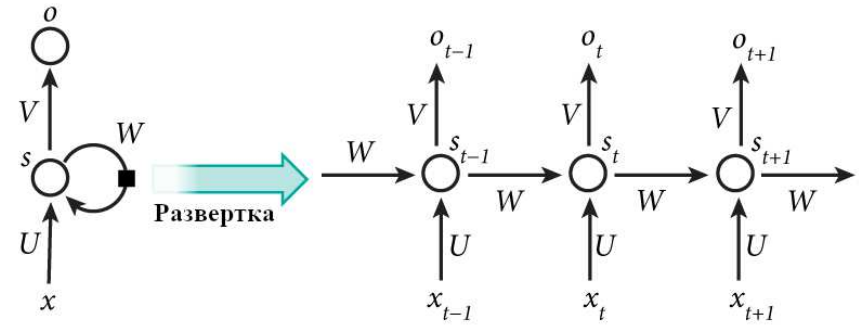


Рисунок 1 — Архитектура рекуррентной сети

Посмотрим, что происходит на отдельно взятом слое:

1. На каждом временном шаге t на вход подается вектор x_t, отвечающий символу в слове.
2. S_t — это скрытое состояние на шаге t. По сути, s_t можно назвать той самой “памятью” слоя. Она вычисляется с помощью предыдущего скрытого состояния и входа на текущем шаге

$$s_t = f(Ux_t + Ws_{t-1}), \tag{6}$$

где U — матрица перехода скрытого состояния в скрытое состояние, а W - матрица весов. Функция f обычно $tanh$ или различные вариации $ReLU$.

3. O_t — выход на шаге t.

$$o_t = softmax(Vs_t), \tag{7}$$

где V — словарь.

Однако, рекуррентные сети обладают довольно большой проблемой исчезающего градиента. Алгоритм градиентного спуска находит глобальный минимум функции стоимости, что является оптимальной настройкой для сети. Также, мы вычисляем ошибку и распространяем ее обратно по сети, чтобы

обновить веса. Получается, что сеть сравнивает нашу вычисленную функцию стоимости с желаемым выходом.

Вычисленная ошибка будет распространятся не только на предыдущий слой, но и на все предыдущие временные шаги, а так как в самом начале мы выставляем веса близкие к нулю, то со временем градиент станет все меньше с каждой операцией умножения.

1.2 Долгая краткосрочная память

LSTM (долгая краткосрочная память) сеть исправляет проблемы рекуррентных сетей. В ней нет фундаментальных различий с рекуррентными сетями, однако изменено вычисление выходов и скрытых состояний, использующих входные значения.

Данные сети приспособлены для обучения долговременным зависимостям, к тому же она лишения проблемы исчезающего градиента.

Архитектура данной сети представлена на рис. 2, которая описывается следующими формулами:

$$f_t = \sigma_g\left(W_f x_t + U_f h_{t-1} + b_f\right), \quad (8)$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i), \quad (9)$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o), \quad (10)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c), \quad (11)$$

$$h_t = o_t \circ \sigma_c(c_t), \quad (12)$$

где $\circ$ — оператор поэлементного умножения, x_t — входной вектор, h_t — выходной вектор, c_t — вектор состояний, W, U — матрицы параметров, b — вектор параметров, f_t, i_t, o_t — фильтры, σ_g, σ_c — функции активации на основе сигмоиды и гиперболического тангенса.

Фильтр забывания контролирует убывание значения, хранящегося в памяти, т.е. до тех пор, пока входной и выходной фильтр не работают, значение градиента не меняется.

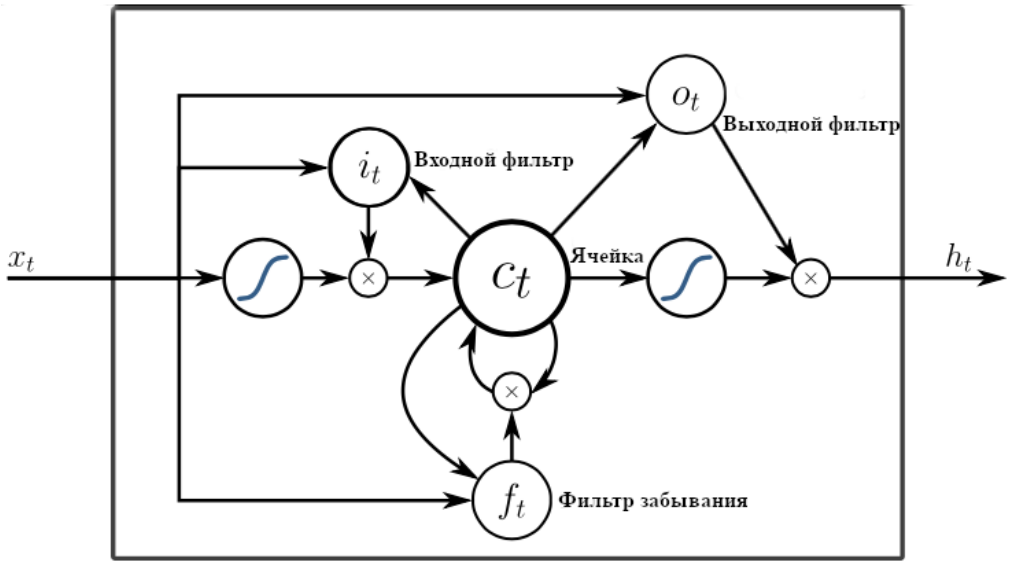


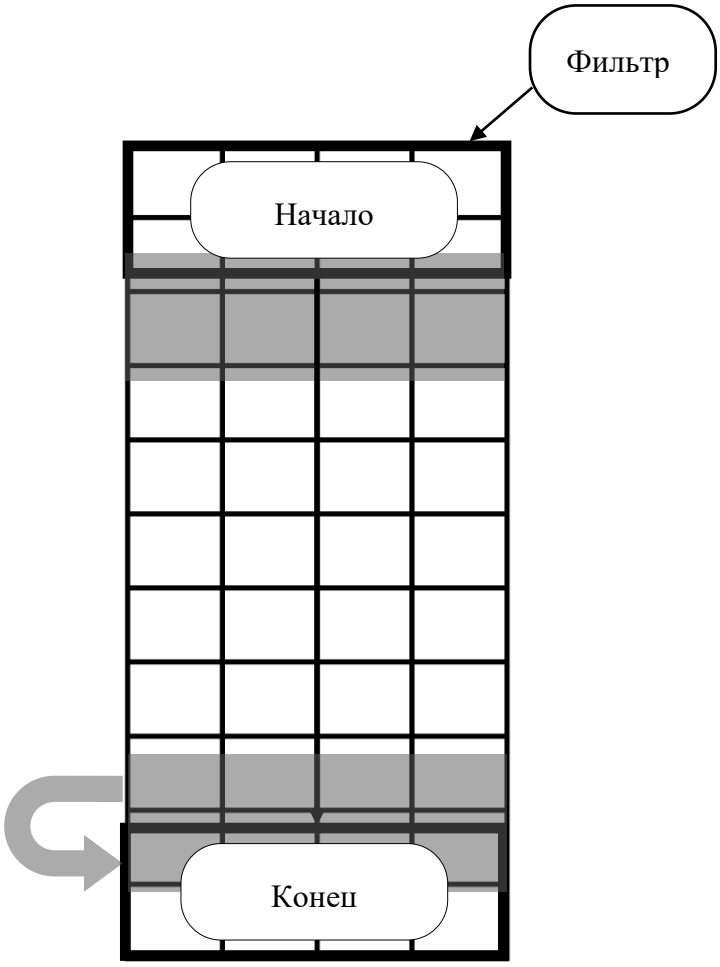
Рисунок 2 — Архитектура LSTM

2 Основная теория и архитектура слоистой нейронной сети для распознавания человеческой речи

2.1 Сверточные нейронные сети

Обычно, сверточные сети ассоциируются с компьютерным зрением, поскольку, двухмерную свертку часто можно увидеть на высоких позициях в рейтингах соревнований по распознанию образов. Однако, одномерная свертка может конкурировать с рекуррентными сетями в задачах обработки последовательностей, при этом имея меньшую вычислительную стоимость.

По аналогии с двумерной сверткой, которая извлекает двумерные участки и применяет идентичное преобразование к каждому участку, так и одномерная свертка извлекает локальные подпоследовательности из последовательностей.



Строка — закодированное представление

Рисунок 3 — Одномерная свертка

Данные слои могут распознавать локальные признаки в последовательности. Поскольку одно и тоже входное преобразование применяется к каждой подпоследовательности, признаки, изученные в одной позиции, могут быть распознаны в другой позиции.

Пусть $x_i \in \mathbb{R}^k$ — k-мерный вектор, представляющий из себя элемент последовательности. Тогда последовательность длины n представляется как

$$x = x_1 \oplus x_2 \oplus \ldots \oplus x_n, \tag{13}$$

где $\oplus$ — оператор объединения.

Теперь, каждый признак c_i будет вычисляться по формуле:

$$c_i = f(w \cdot x_{i:i+h-1} + b), \tag{14}$$

здесь $x_{i:i+h-1}$ — объединение элементов от i до $i + h - 1$, $w \in \mathbb{R}^{hk}$ — фильтр свертки, который применяется к окну из h элементов, получая новые признаки, b — отклонение, f — нелинейная функция. На выходе мы получаем вектор из признаков, вычисленных по каждому окну, который называется картой признаков:

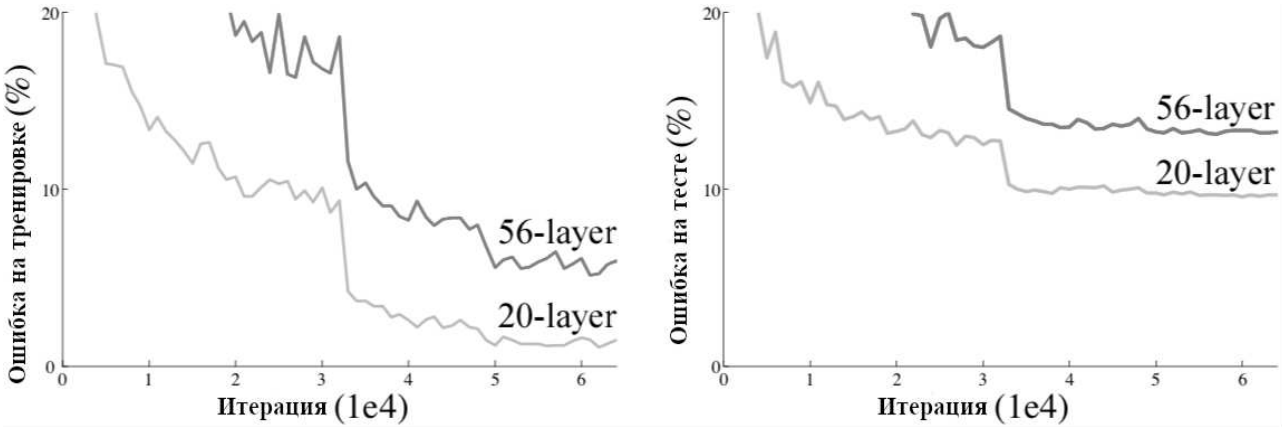
$$c = [c_1, c_2, \ldots, c_{n-h+1}]. \tag{15}$$

Обычно фильтр w инициализируется случайным образом. Нельзя устанавливать все веса в нуль. Если все нейроны будут иметь одинаковые веса, то они будут создавать одни и те же результаты и сеть не будет изучать новые признаки. Все веса во время обновления будут оставаться одинаковыми. Однако, если в начале взять случайное распределение, то с высокой вероятностью все веса нейронов будут различными и сеть сможет изучать различные признаки.

При этом, мы будем использовать свертку с расширенным каузальным фильтром. Расширение фильтра означает расширение его размера, заполняя пустые позиции нулями. На практике такой фильтр не создается, вместо этого, веса сопоставляются с удаленными (не смежными) элементами во входном векторе. Расстояние определяется коэффициентом расширения. При этом, фильтр называется каузальным, если выход фильтра не зависит от будущих входов.

2.2 Остаточное обучение

Чем глубже, то есть чем больше имеет количество слоев нейронная сеть, то тем сложнее ее обучать, при этом, начиная с некоторой глубины сеть начинает терять в точности. Однако, в 2015 году, исследователи из Microsoft представили новую структуру в работе [4], благодаря которой глубокую сеть легче оптимизировать и увеличивается точность при увеличении числа слоев, данную структуру называют остаточной.



Ошибка на тренировке (слева) и ошибка на тесте (справа) на наборе данных CIFAR-10 у 20 и 52 слойных сетей. Более глубокая сеть имеет выше ошибку как на тренировочных данных, так и на тестовых.

Рисунок 4 — Сравнение 20 и 56 слойных сетей.

Вместо того, чтобы аппроксимировать некоторую сложную функцию $H(x)$ мы дадим сетям изучать функцию $F(x) = H(x) - x$, тогда, очевидно, что наша первоначальная функция преобразуется в $F(x) + x$. Получившееся отображение

легче оптимизировать, чем отображение без ссылок, так как сети будет легче сделать все веса равными нулю на очередном слое, если на предыдущем был достигнуть предел по точности, чем пытаться подогнать сопоставление стеком нелинейных слоев.

Формулировка функции $F(x) + x$ может быть реализована с помощью сетей с shortcut-соединениями. Shortcut-соединения — это те, которые могут пропускать один или несколько слоев. Такие соединения не добавляют никаких новых параметров и не увеличивают вычислительную сложность. Поэтому, сеть все так же может быть обучена с применением стохастического градиентного спуска с обратным распространением.

Другими словами, при обучении многослойной сети может возникнуть ситуация, когда какой-то из слоев не успел натренироваться, поэтому, если из предыдущих слоев, которые выдают полезный сигнал придут на вход данные в тот самый необученный слой, то на выходе он отдает испорченный сигнал, который нарушает работу всей сети. Поэтому, идея остаточного обучения состоит в том, что все что пришло на вход слоя, мы будем передавать дальше по сети, а слой будет иметь возможность только немного изменить сигнал, т.е. слой предсказывает выход не напрямую, а только поправку. Как видно на рис. 5 на вход мы получаем x, а на выходе мы выдаем не $F(x)$, а $F(x) + x$.

За счет данной архитектуры можно увеличивать количество слоев без потери точности, при этом, получается что мы избавляемся от проблемы исчезающего градиента и он очень быстро распространяется на всю сеть, следовательно, сеть получит полезный сигнал на всех слоях и все слои вместе смогут с самого начала тренироваться.

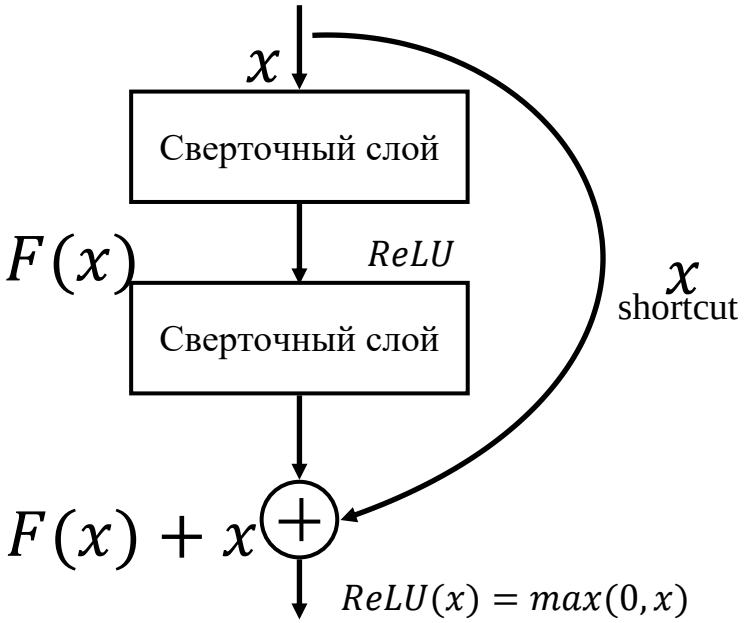


Рисунок 5 — Базовый блок остаточного обучения

2.3 Connectionist Temporal Classification

На выходе нашей сети мы получим оценки произнесения символа в момент времени t, данный выход представляет из себя матрицу (см. рис 6). В этот момент возникает две задачи, которые мы хотим решить:

1. Рассчитать значение потерь во время обучения.
2. Декодировать матрицу, чтобы получить текст, распознанный нашей моделью.

Для решения данных задач мы будем применять функцию потерь CTC (Connectionist Temporal Classification).

Нам необходимо сообщить функции потерь CTC только текст, который является расшифровкой нашего звукового файла, нет нужды беспокоиться о продолжительности звуков для каждой буквы, более того, последующая обработка распознанного текста не нужна.

	1	2	3	4	5	6	7	8	9	10
A	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
B	**0.20**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
C	0.0	0.0	0.0	0.0	**0.53**	0.0	0.0	0.0	0.0	0.0
D	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	**0.69**
…	…	…	…	…	…	…	…	…	…	…
Z	**0.46**	0.0	0.0	0.0	0.0	0.0	0.0	**0.78**	0.0	0.0
‘	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
space	0.0	0.0	0.0	0.0	0.0	**0.92**	0.0	0.0	0.0	0.0

Рисунок 6 — Матрица оценок вероятностей появления символа в момент времени t

Для решения первой проблемы создатели CTC определили метрику количества неверных меток (label error rate) для временного классификатора h как среднее нормализованное расстояние Левенштейна между полученным результатом и настоящим текстом:

$$LER(h, S') = \frac{1}{|S'|} \sum_{(x,z)\in S'} \frac{ED(h(x),z)}{|z|}, \tag{16}$$

где $ED(p, q)$ — расстояние Левенштейна между последовательностями p и q, S' — тестовый набор данных, состоящий из пар (x, z). Данную метрику мы и будем оптимизировать при обучении модели.

Вторую проблему решает метод декодирования, который основан на алгоритме лучевого поиска (beam search), также этот поиск называется префиксным, так как на временном шаге t выбирается наиболее подходящий префикс.

Пусть $P_b(t,l)$ — вероятность того, что префикс l, на определенном временном шаге t происходит от одного или нескольких путей, заканчивающимся пустым символом, $P_{nb}(t,l)$ — является аналогом пустой вероятности, поэтому учитывает все основные пути, оканчивающиеся не пустым символом, $P_{ctc}(t,c)$ — вероятность появления символа c (номер символа в алфавите) на шаге t, $P_{lm}(x)$ — вероятность появления последовательности слов x в речи по оценке языковой модели.

Шаг 1. Мы начинаем с пустых префиксов. Довольно просто понять, что пустой префикс по определению не сможет возникнуть из пути, проходящего через не пустые символы. Список R содержит все префиксы, которые мы попытаемся расширить на заданном временном шаге, и будет обновлен, чтобы содержать k наиболее вероятных префиксов в конце каждой итерации.

Шаг 2. Для каждого временного шага мы оцениваем каждый префикс в R и пытаемся расширить его каждым символом в нашем алфавите.

Шаг 3. Если c — пустой символ, то $P_b(t,l)$ вычисляется по формуле:

$$P_b(t,l) = P_b(t,l) + P_{ctc}(t,c)(P_b(t-1,l) + P_{nb}(t-1,l)). \tag{17}$$

Шаг 4. Иначе c из алфавита, поэтому рассматриваем префикс $l_1 = l + c$ и тогда возникает 3 возможных ситуации:

1. c равен последнему символу в последовательности l, поэтому увеличиваем вероятность того, что предыдущим символом в пути был символ пропуска и вероятность того, что предыдущим символом был символ c.

$$P_{nb} = P_{nb} + P_{ctc}(t,c)P_b(t-1,l), \tag{18}$$

$$P_{nb} = P_{nb} + P_{ctc}(t,c)P_{nb}(t-1,l). \tag{19}$$

2. с равен пробелу, либо мы находимся на последнем шаге t.

$$P_{nb} = P_{nb} + (P_{ctc}(t, c)P_{nb}(t-1, l) + P_b(t-1, l))P_{lm}(l_1) \quad (20)$$

3. c из алфавита, исключая симпвол пробела. Поэтому, увеличиваем вероятность префикса l_1:

$$P_{nb} = P_{nb} + (P_{ctc}(t, c)P_{nb}(t-1, l) + P_b(t-1, l)) \quad (21)$$

Шаг 4. Оказывается, только потому что префикс был отброшен на предыдущем временном шаге, он все еще может быть полезен для нас.

$$P_{nb} = P_{nb} + P_{ctc}(t, c)P_{nb}(t-1, l), \quad (22)$$

$$P_b = P_b + P_{ctc}(t, c)P_b(t-1, l). \quad (23)$$

Шаг 5. После вычисления вероятность префиксов на шаге t, мы сортируем префиксы по уменьшению $P_b(t, l) + P_{nb}(t, l)$.

После мы выбираем kнаиболее вероятных префиксов. На последнем шаге выбирается наиболее вероятный префикс.

Большая ширина луча (k — ширина луча) приводит к лучшей производительности модели, поскольку множественные последовательности-кандидаты увеличивают вероятность лучшего соответствия целевой последовательности. Это увеличение производительности приводит к снижению скорости декодирования. В данной работе ширина луча была выбрана равной 100, как было установлено по умолчанию в используемой реализации.

Процесс поиска может быть остановлен для каждого кандидата в отдельности либо путем достижения максимальной длины, либо путем получения маркера конца последовательности, либо путем достижения пороговой вероятности.

2.4 Пакетная нормализация

Обучение глубоких нейронных сетей усложняется тем фактом, что распределение входных данных каждого слоя изменяется во время обучения по мере изменения параметров предыдущих слоев. Это замедляет обучение, требуя понижения коэффициента скорости обучения, а также аккуратной настройки параметров, и делает достаточно трудным обучение моделей, перенасыщенных нелинейными зависимостями. Данное явление называют внутренним ковариантным сдвигом (internal covariate shift). Другими словами, ковариантный сдвигом называют такую ситуацию во время обучения сети, когда распределение признаков в тестовой и обучающей выборке имеют разные параметры, такие как математическое ожидание, дисперсию и т.п.

Очевидным решением данной проблемы является перемешивание исходных данных перед формированием пакета. Однако, такое решение не подходит для скрытых слоев, так как было сказано выше, что распределение входных данных каждого слоя изменяется во время обучения по мере изменения параметров предыдущих слоев.

Другим методом решения проблем является метод пакетной нормализации, он позволяет использовать высокий коэффициент скорости обучения, а также быть менее внимательным при настройке параметров. При этом, он действует как регулязатор, устраняя необходимость в Dropout.

Для каждого слоя, в который на вход подается вектор $x = (x^{(1)}, x^{(2)}, \dots, x^{(n)})$ будет нормализоваться каждая компонента:

$$\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{D[x^{(k)}]}}, \tag{24}$$

где математическое ожидание E и дисперсия D вычисляются по всему набору данных. Необходимо обратить внимание на то, что данная нормализация может изменить представление слоя. Чтобы решить данную проблему нужно ввести

пару параметров $\gamma^{(k)}$ и $\beta^{(k)}$ для каждой компоненты $x^{(k)}$, которые сжимают и сдвигают нормализованное значение:

$$y^{(k)} = \gamma^{(k)}\hat{x}^{(k)} + \beta^{(k)}. \tag{25}$$

Данные параметры обучаются вместе с исходными параметрами модели. Установив $\gamma^{(k)} = \sqrt{D[x^{(k)}]}$ и $\beta^{(k)} = E[x^{(k)}]$ мы восстановим оригинальные активации, если будет необходимо.

Рассмотрим, как применяется данный метод к сверточным сетям. Рассмотрим аффинное преобразование с нелинейной функцией:

$$z = g(Wu + b), \tag{26}$$

где W и b — обучаемые параметры модели. Мы применяем нормализацию к $x = Wu$, параметр смещения b можно проигнорировать, так как его роль будет играть параметр β, поэтому мы получаем

$$z = g(BN(Wu)), \tag{27}$$

где BN — нормализация, примененная к каждой размерности x.

При этом, для сверточных сетей также необходимо учесть, что разные элементы карты признаков должны нормироваться одинаковым образом. Для этого мы нормализуем все значения в пакете совместно.

2.5 Архитектура

Как уже было сказано, рекуррентные нейронные сети и его улучшенная версия — LSTM сеть являются самыми популярными архитектурами для распознавания речи, которые обрабатывают последовательность и имеют

возможность учитывать огромный контекст. В зависимости от задачи мы ограничиваем контекст только прошлым — изучением причинно-следственных связей в данных. При этом, мы можем учитывать как прошлое, так и будущее. Современным решением данной задачи являются двунаправленные рекуррентные слои. Их один проход изучает отношения слева направо, а другой - справа налево. Результаты затем объединяются.

Однако, мы можем включить такой двухсторонний контекст и в одномерные свертки, проблема состоит в том, чтобы входные данные зависели от большого контекста, необходимо использовать фильтры все большего размера, а это влечет за собой большое потребление памяти.

Мы крайне ограниченны в вычислительных ресурсах, поэтому, необходимо максимально сократить использование памяти. В разделе 2.1 мы рассмотрели свертку с расширенным каузальным фильтром, которые оказываются гораздо менее вычислительно затратные за счет своей структуры, так как мы храним в памяти не весь большой фильтр, а всего лишь расширенную версию довольно небольших фильтров, в которых пустые места заполнены нулями.

Обычно, за сверточными слоями следует функция активации ReLU и она хорошо подходит для парадигмы сверточной сети, тогда как рекуррентные слои используют подходы забывания и запоминания, которые реализованы в забывающем фильтре.

Было бы неплохо использовать эту способность рекуррентных сетей в нашей работе. Чтобы это сделать мы будем использовать принцип активационных вентилей (gated activation), объясненный в работе [5].

Идея крайне проста: мы будем пропускать входные данные через одномерную свертку и затем применять функции $tanh$ и σ, а результатом будет поэлементное умножение. При этом, мы пойдем еще дальше и будем применять функцию $tanh$ еще раз в конце.

На рисунке 7 представлена архитектура сети в общем виде, на рисунке 8 отражена структура стека остаточных блоков и остаточного блока, которые мы рассмотрели выше.

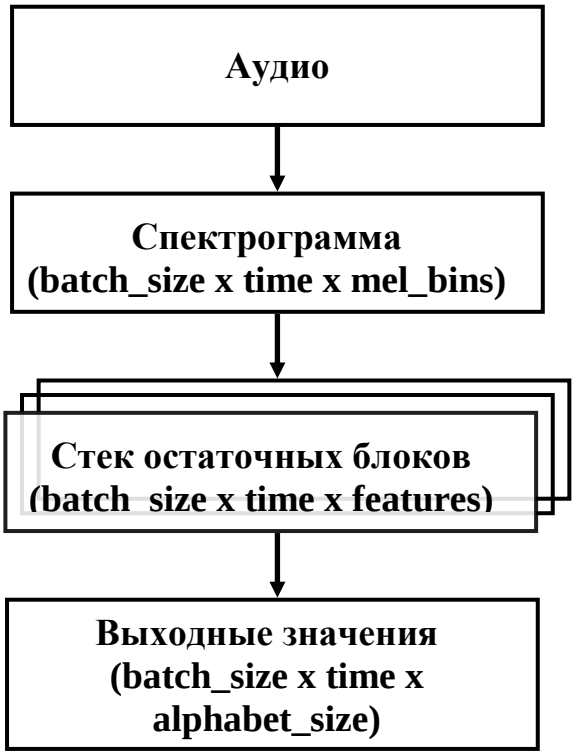


Рисунок 7 — Архитектура сети в общем виде

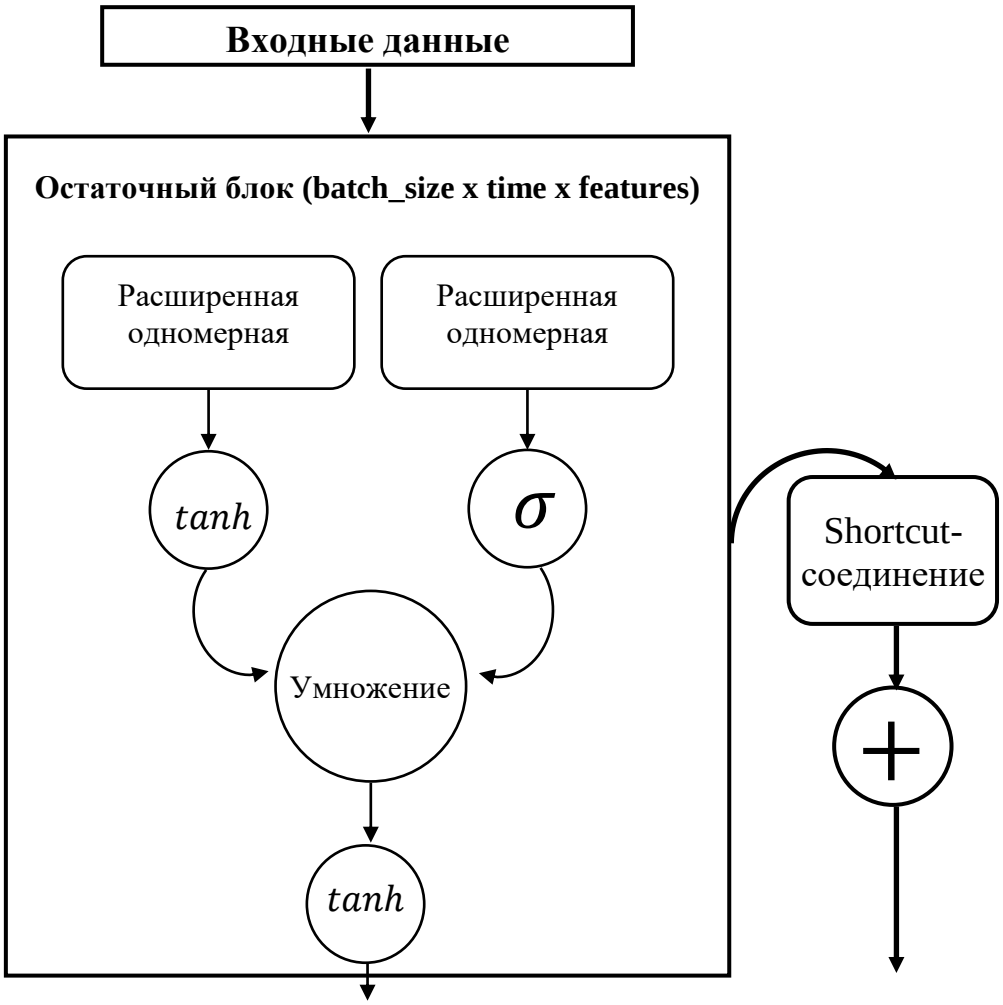


Рисунок 8 — Структура стека остаточных блоков и его блока, лист 1

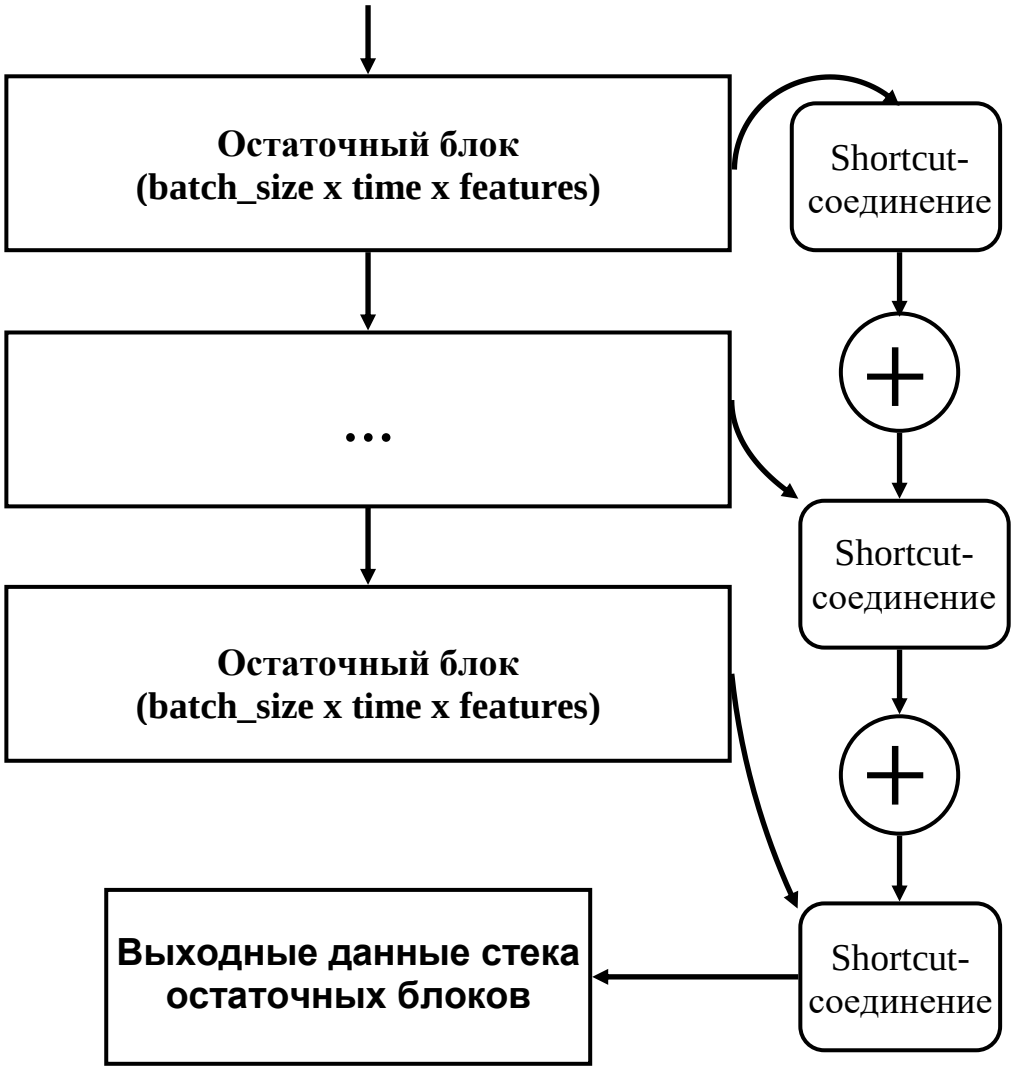


Рисунок 8 — Структура стека остаточных блоков и его блока, лист 2

3 Обучение модели

3.1 Аппаратное обеспечение

Обучение производилось на конфигурации:

1. Intel Core i5-4430
2. GTX1060 3GB
3. 8 GB RAM

Все расчеты производились на видеокарте, что существенно сокращает время обучения. Однако, для данной задачи распознавания речи 3GB видеопамяти оказалось недостаточно для больших размеров пакетов (batch) данных, так, например, уже для размера пакета больше 6 через непродолжительный промежуток времени обучения количество признаков становилось настолько большим, что матрицы весов не помещались в памяти, что в свою очередь приводило к ошибке. Поэтому, был выбран размер пакета равный 5, что заметно сказалось на скорости обучения. При этом, оптимальным выбором для такого пакета стал стохастический градиентный спуск (SGD) с коэффициентом обучения (learning rate) равным 0.00001, с последующим увеличением, который так же проигрывает в скорости своим различным вариациям.

Некоторые настройки были выбраны путем перебора, так, например, размер стека остаточных (residual) блоков равен 6, а размер ядра (kernel size) равен 7, так как при всех значениях ниже модель не может выявить закономерностей и соответственно недообучается.

3.2 Инструменты

3.2.1 Python

Для реализации работы был выбран язык программирования Python, так как данный язык имеет большое количество сторонних библиотек для разработки нейронных сетей и анализа данных. Он является самым передовым языком в области машинного обучения, при этом предоставляет простой синтаксис для работы.

3.2.2 Pandas

Pandas — программная библиотека на языке Python для обработки и анализа данных. С ее помощью производилась обработка данных о звуковых файлах и производился анализ информации о наборе данных.

3.2.3 NumPy

NumPy — библиотека с открытым исходным кодом для языка программирования Python. Возможности:

- поддержка многомерных массивов (включая матрицы);
- поддержка высокоуровневых математических функций, предназначенных для работы с многомерными массивами.

Данная библиотека применялась для предобработки звуковых файлов, в частности для аугментации, описанной в 3.4.4, а также был использован контейнер npy, который позволил загружать и обрабатывать данные в несколько раз быстрее, чем стандартные csv и mp3, за счет чего было ускорено обучение модели.

3.2.4 LibROSA

LibROSA — библиотека для языка программирования Python, предназначенная для обработки и анализа аудиозаписей. Главным отличием является то, что есть возможность работать с форматом mp3, что было необходимо для данной работы.

Все аудиозаписи были загружены при использовании функции librosa.core.load.

3.2.5 Tensorflow

TensorFlow — открытая программная библиотека для машинного обучения, разработанная компанией Google для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации образов, достигая качества человеческого восприятия. Применяется как для исследований, так и для разработки собственных продуктов Google. Основной API для работы с библиотекой реализован для Python, также существуют реализации для C++, Haskell, Java, Go и Swift.

Причинной выбора данной библиотеки стала возможность написания собственных нейронных слоев, тонкой настройки параметров и наличие доступной и развернутой документации, в совокупности с большим сообществом разработчиков, пользующихся данной библиотекой, которые оказывают поддержку и решают проблемы, возникающие у других пользователей. Также, Tensorflow имеет поддержку вычислений на видеокартах, что существенно ускоряет процесс обучения моделей.

Для того, чтобы создать и обучить свою собственную сеть на TensorFlow мы выполнили 4 пункта:

1. Определили модель
2. Определили функцию подачи данных на вход и генератор данных.
3. Определили функцию модели, функцию потерь, оценщик
4. Загрузили данные и использовать оптимизатор для обучения модели.

Чтобы задать свой собственный слой, необходимо данный слой описать с помощью класса, наследованного от класса tf.layers.Layer. Так как базовые элементы сети не являются уникальными, то мы используем их реализацию из данной библиотеки, так, например мы использовали одномерную свертку, пакетную нормализацию, функции σ и $tanh$ и другие функции. Таким образом,

мы смогли реализовать принцип остаточного обучения с нестандартными слоями для оригинального исследования [3].

3.3 Обучающая выборка

Для обучения сети используется открытый набор данных Common Voice от Mozilla, представляющий собой множество аудиофайлов на английском языке в формате mp3, общей продолжительностью 582 часа от 33541 человека.

Каждой аудиозаписи соответствует транскрипт в виде текста, а также, примерно в 50% записей присутствует демографическая информация, такая как возраст, пол, акцент.

Пример записи:

- filename cv-valid-train/sample-0000073.mp3
- text the greatest authority on meteorites started that the height of its first appearance was about ninety or one hundred miles
- up_votes 5
- down_votes 0
- age twenties
- gender male
- accent us
- duration NaN

В силу ограниченности вычислительных средств используется только информация о соответствующем тексте аудиозаписи.

Вся выборка разделена на три части: тренировочную, тестовую и проверочную.

Обучающая выборка — выборка, по которой мы настраиваем параметры нашей модели, содержит 195776 сэмплов.

Тестовая выборка — выборка, по которой мы оцениваем качество обученной модели, содержит 3995 сэмплов.

Проверочная выборка — выборка, по которой мы выбираем наилучшую построенную модель, содержит 4076 сэмплов.

Все выборки независимы, т.е. не имеют общих записей, следовательно, таким образом добиваемся несмещенной оценки на тестовых данных.

3.4 Предобработка данных

3.4.1 Сэмплирование

Для того, чтобы сеть могла обрабатывать аудиофайлы, которые ей подали на вход и успешно обучаться на них, извлекая признаки необходимо представить каждую аудиодорожку в виде набора цифр.

Известно, что звук передается как волна, то есть необходимо превратить волну в цифры. Здесь существует два вида представления звука:

1. Каждый сэмпл представляется значениями изменения давления воздуха.
2. Каждому временному шаге сэмпла сопоставляется амплитуда для каждой частоты.

Вопреки тому факту, что нейронные сети довольно хороши в автоматическом изучении признаков, лучше использовать известные признаки, которые несут информацию, необходимую для решения нашей проблемы.

Для большинства приложений, в частности, распознавания речи, признаки, которые нам нужны содержатся в представлении звука с помощью частоты.

Для этого дискретизируется (сэмплируется от англ. sampling) звуковая волна, то есть высоте волны сопоставляется точка в каждый момент времени, обычно это делается несколько десятков тысяч раз в секунду. Таким образом, например, устроены несжатые .wav файлы.

Частоту выбора точек называют частотой дискретизации и измеряют в герцах. Наиболее часто, аудиофайлы сэмплируют в 44.1 кГц, 16 кГц и 8 кГц. В данной работе используется частота в 16 кГц.

Несмотря на то, что оригинальная волна аппроксимируется довольно грубо сэмплированная запись не теряет информацию, благодаря теореме Котельникова.

Теорема 3.4.1 Котельникова

Любую функцию $F(t)$, состоящую из частот от 0 до f_1, можно

непрерывно передавать с любой точностью при помощи чисел, следующих друг за другом через $1/(2f_1)$ секунд.

Сэмплирование производилось с помощью функции Load из библиотеки LibROSA.

3.4.2 Спектрограмма

На данном моменте данные уже можно передать на вход нейронной сети, однако извлечение признаков из них будет крайне долгим и сложным. Поэтому, необходимо выполнить некоторую предобработку, чтобы упростить задачу, для этого используется спектрограмма, на основе которой будут вычислены мел-частотные кепстральные коэффициенты (MFCC).

Определение 3.4.1 Спектрограмма (сонограмма) — изображение, показывающее зависимость спектральной плотности мощности сигнала от времени, вычисляется по следующей формуле:

$$spectorogram(t, \omega) = |STFT(t, \omega)|^2, \tag{28}$$

где STFT — оконное преобразование Фурье, t - время, ω - частота.

Чтобы вычислить оконное дискретное преобразование Фурье (ОДПФ) необходимо зафиксировать ширину окна, обычно от 20 до 40 мс, далее

необходимо применить оконную функцию, в данной работе используется оконная функция Ханна:

$$\omega(n) = 0.5\left(1 - cos\left(\frac{2\pi n}{N-1}\right)\right), \tag{29}$$

где N — ширина окна. Таким образом продолжаем для всей аудиозаписи, при этом смещая окно, обычно на 20-30% от своей ширины.

$$STFT(m,\omega) = \sum_{n=-\infty}^{\infty} f[n]\omega[n-m]e^{-j\omega n}. \tag{30}$$

Чтобы выделить гармонические отношения, в конце необходимо взять логарифм спектрограммы, что будет соответствовать человеческому ощущению громкости.

3.4.3 Мел-частотные кепстральные коэффициенты

Определение 3.4.1 Шкала мел — шкала субъективного восприятия высот звуковых тонов, при этом зависимость от частоты f является нелинейной и выражается по формуле:

$$m = 2595log_{10}\left(1 + \frac{f}{700}\right). \tag{31}$$

После того как спектрограмма была преобразована в мел-спектрограмму к ней применяется сглаживание банком фильтров:

$$H_m(k) = \begin{cases} 0, & k < f(m-1) \\ \frac{k-f(m-1)}{f(m)-f(m-1)}, & f(m-1) \le k < f(m) \\ 1, & k = f(m) \\ \frac{f(m+1)-k}{f(m+1)-f(m)}, & f(m) < k \le f(m+1) \\ 0, & k > f(m+1) \end{cases}, \quad (32)$$

где $f = 700(10^{m/2595} - 1)$. Каждый фильтр в банке фильтров имеет треугольную форму с откликом 1 на центральной частоте и линейно уменьшается до 0, пока не достигнет центральных частот двух соседних фильтров, где отклик равен 0.

В конце используется дискретное косинусное преобразование (DCT), чтобы уменьшить корреляцию между признаками.

$$DCT - 2_n = \left[cos\left(k\left(l + \frac{1}{2}\right)\frac{\pi}{n}\right)\right]_{0 \le k,l < n}. \quad (33)$$

Таким образом, можно обобщить алгоритм получения MFCC:

1. Разделить сигнал на окна (фреймы) в 20-40 мс.
2. Вычислить дискретное оконное преобразование Фурье.
3. Выполнить преобразование коэффициентов в мел пространство.
4. Сглаживание банком фильтров.
5. Взять логарифм от каждом мел частоты.
6. Применить дискретное косинусное преобразование.

MFCC в данной работе были вычислены по выше описанному алгоритму с применением функций из TensorFlow и LibROSA, при этом надо отметить, что данные коэффициенты мы вычисляем каждый раз снова, так как мы используем аугментацию, описанную в 3.4.4 и нам необходимо искажать именно исходный сигнал, а не коэффициенты.

3.4.4 Аугментация

Аугментация данных является методом получения дополнительных тренировочных данных, путем изменения исходных. Несмотря на то, что данный метод больше полезен в условиях крайней ограниченности тренировочной выборки он также помогает уменьшить дисперсию и соответственно, лучше обобщаться на новые данные.

Аугментация сэмплов производится тремя способами:

1. Добавление шума
2. Сдвиг (shift)
3. Растяжение (stretch)

Заметим снова, что “искажение” сэмпла производится до расчета мел-частотных кепстральных коэффициентов. Для лучшего эффекта добавлять шум можно из заранее приготовленных сэмплов, таким образом, мы бы могли добавить записи слов, которые не присутствуют в оригинале, чтобы модель могла отделять речь пользователя от посторонних источников звука. Однако, для упрощения мы зашумляем наши исходные данные случайными значениями, что менее эффективно и как мы сможем убедиться в 3.6 такого подхода недостаточно, хотя и в некоторой степени он помогает.

Аугментация была проведена с использованием функций random.uniform, pad, time_stretch из библиотек NumPy и LibROSA.

3.5 Процесс обучения

Обучение модели на компьютере с характеристиками из 3.1 заняло порядка 25 дней, с несколькими небольшими перерывами, при этом Tensorflow позволяет сохранять состояние сети, что дает возможность после остановки обучения запустить его с того же момента, на котором он был сохранен в

прошлый раз, так мы не рискуем потерять наш прогресс из-за непредвиденных проблем.

Как уже было сказано, обучающая выборка состоит из порядка 200000 аудиозаписей, при этом, мы имеем довольно сложную архитектуру, несмотря на то что она была оптимизирована для меньшего использования памяти из-за недостаточной производительности данного аппаратного обеспечения некоторые настройки сети были вынужденно снижены.

Использованные параметры для предобработки данных, а также для обучения модели:

- Batch size = 5 — размер пакета, т.е. мы обрабатываем 5 аудиозаписей за один шаг обучения. Любое значение больше 5 приводило к переполнению памяти, так как в некоторый момент времени становилось слишком много обрабатываемых значений.
- Stack dilation rates = [1, 3, 9, 27] — коэффициенты расширения свертки.
- Stacks = 6 — количество остаточных блоков, 6 - наименьшее значение, которое не приводило к переполнению памяти, на более производительном компьютере было бы лучше установить значения больше, так как остаточная сеть не теряет в точности при увеличении числа слоев.
- Stack kernel size = 7 — размер фильтра, для одномерной свертки 7 — одно из оптимальных значений, выявленных эмпирически в работе [6].
- Stack_filters = 384 — количество фильтров в сверточном слое, выбирается в большей степени случайно, путем экспериментов, к сожалению проверить другие значения не представлялось возможным в силу ограниченности времени для обучения.
- Number of fft = 160*8 — количество компонент быстрого преобразования Фурье.
- Frame step = 640 — шаг, с которым мы проходим окном по записи для получения спектрограммы.

- Number of mel bins = 160 — количество групп мел для генерации, для дискретизации 16000 через каждые 100 герц.
- Optimizer = ‘SGD’ — оптимизатор, так как мы используем небольшой размер пакета, то SGD работает на них лучше, чем любая другая его вариация, однако, их использование могло бы в теории повысить скорость обучения.
- Learning rate = 0.00001 — коэффициент скорости обучения, изначально 0.00001, когда значения функции потерь выходят на плато, то умножаем на 10.
- Clip gradients = 20.0 — отсечение градиента, обрезает градиент или ограничивает до заданного значения, чтобы градиенты не становились слишком большими, иначе можно «перескочить» минимум функции. Выбирается как 1/10 от максимального значения при обучении без установки данного параметра.

3.6. Результаты

Несмотря на то, что как уже было сказано мы ограниченны в вычислительных мощностях и времени результаты данной работы можно считать достаточно успешными. При этом, теоретически, результаты можно улучшить, даже используя ту же конфигурацию компьютера, так как в ходе исследования были обнаружены как минусы в принятых решениях, так и появились новые идеи для развития. Спустя 25 дней обучения мы получили результаты, представленные на таблице 1. На рис. 9. и 10 отображены графики функции потерь во время обучения для тренировочной и валидационной выборки соответственно. На рис. 11 и 12 можно увидеть график редакционного расстояния для тренировочной и валидационной выборки. Таким образом, мы смогли добиться 92% точности на тестовой выборке, что является довольно неплохим результатом, учитывая то, что мы пошли на снижение сложности модели и установили довольно компромиссные параметры для обучения. В таблице 2 представлено сравнение модели, реализованной в дипломе и моделей

из обсуждаемых статей в 1.1. Сравнение производится по проценту неправильно распознанных слов. Как видно из таблицы 2 наша модель сравнялась по показателям с моделью от IBM 2015 года, так что полученные результаты можно считать успешными.

Таблица 1 — Сравнение значений функции потерь и редакционного расстояние на выборках

	Функция потерь (Loss)	**Редакционное расстояние (Edit distanse)**
Тренировочная выборка	11.09	0.041
Валидационная выборка	18.5	0.076
Тестовая выборка	24.65	0.080

Таблица 2 — Сравнение WER моделей

	Процент неправильно распознанных слов (WER)
Модель из данной работы	**8.0**
The Microsoft 2017 Conversational Speech Recognition System [1]	**5.1**
The IBM 2015 English Conversational Telephone Speech Recognition System [2]	**8.0**
Achieving Human Parity in Conversational Speech Recognition [3]	**5.8**

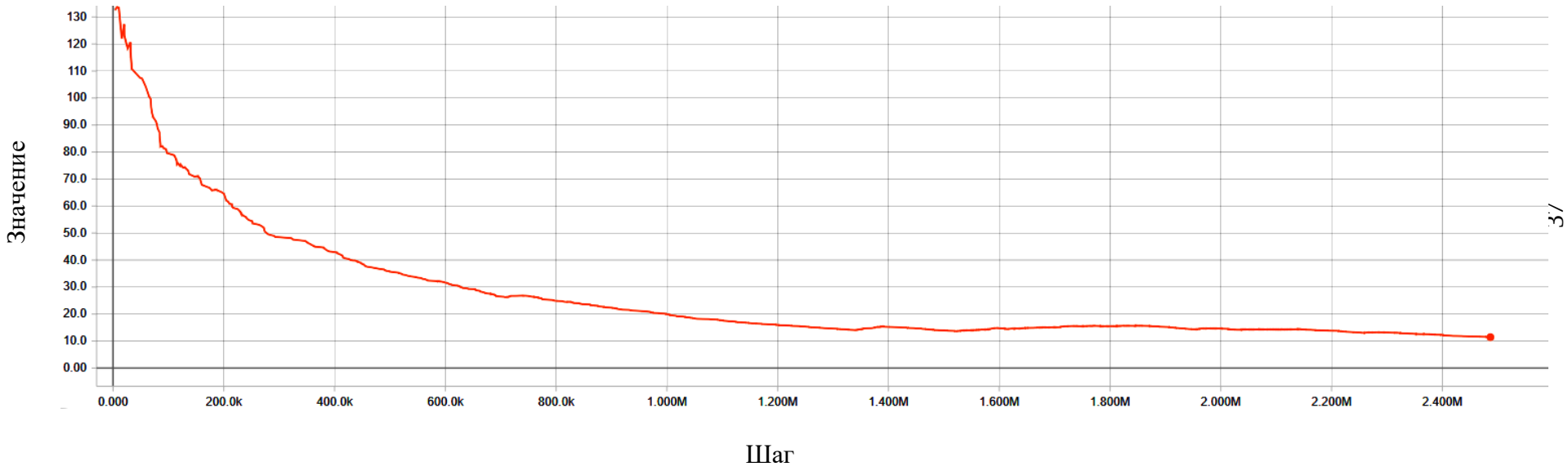


Рисунок 9 — График функции потерь тренировочной выборки

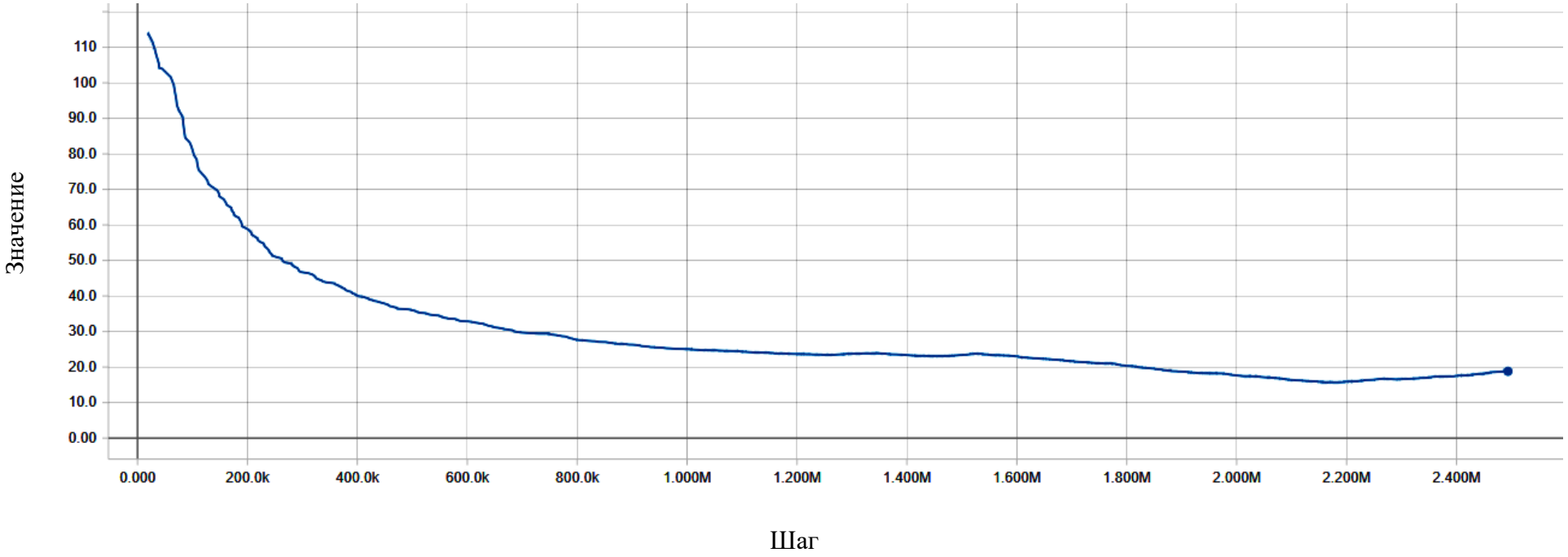


Рисунок 10 — График функции потерь валидационной выборки

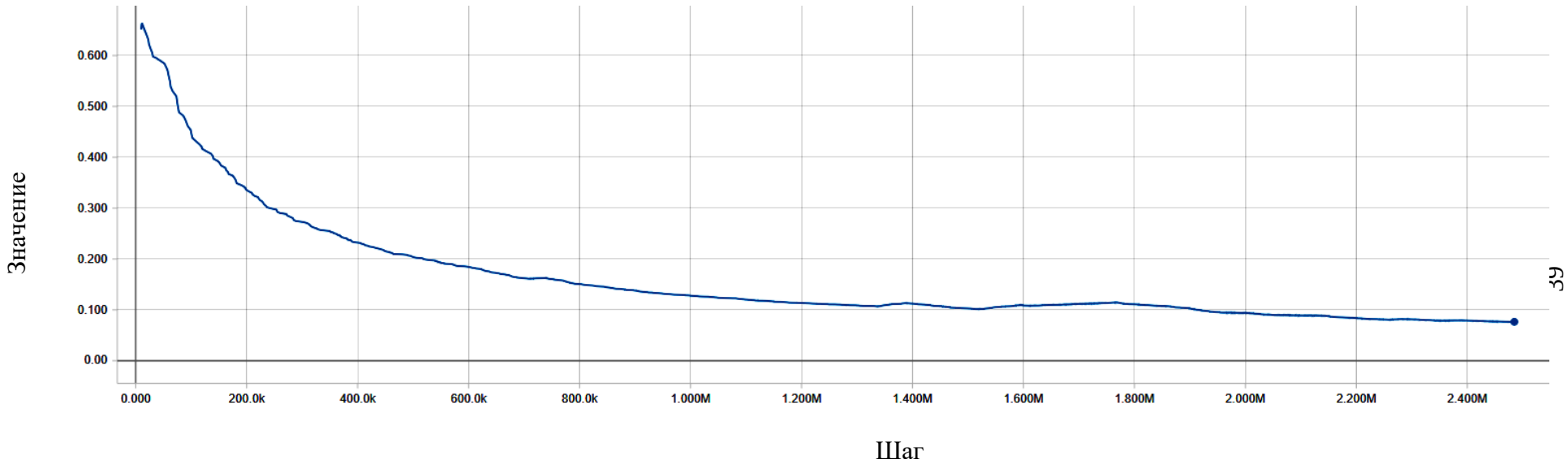


Рисунок 12 — График редакционного расстояния для валидационной выборки

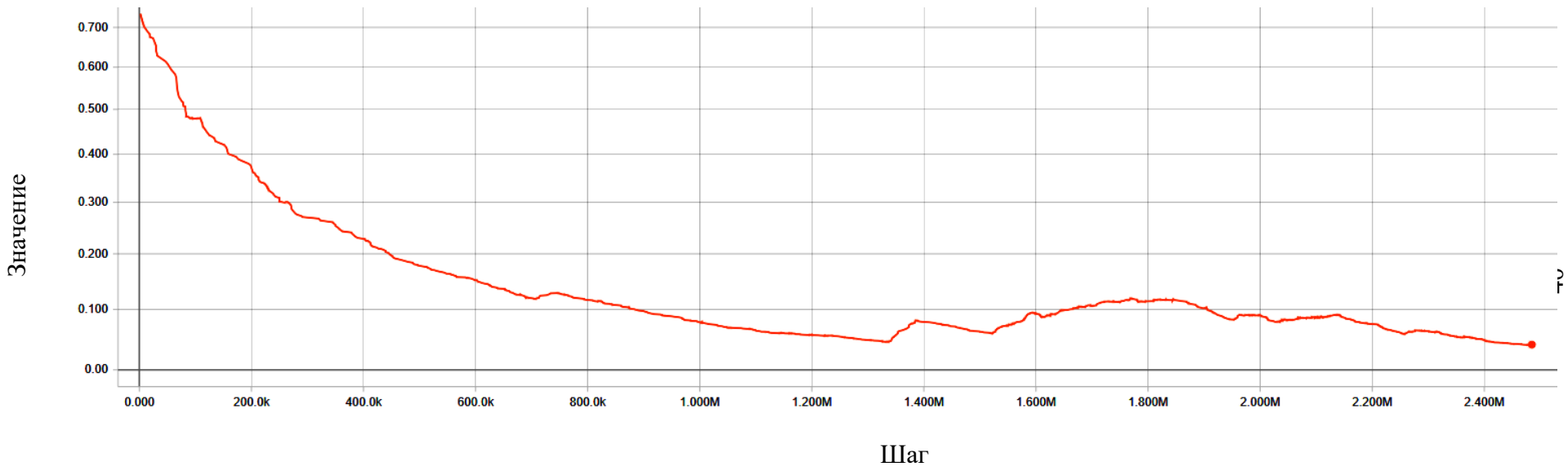


Рисунок 11 — График редакционного расстояния для тренировочной выборки

Приведем некоторые примеры распознанных аудиозаписей из тестовой выборки, жирным выделены слова, в которых есть ошибки:

1. Выход сети: i thought that everything i owned would be destroyed
   Оригинал: i thought that everything i owned would be destroyed
2. Выход сети: it had told him to dig where his tears **fel**
   Оригинал: it had told him to dig where his tears fell
3. Выход сети: this **ventios gona crost flalty**
   Оригинал: this convention's gonna cost plenty
4. Выход сети: coming home a party of tourists **pased** us singing and playing music
   Оригинал: coming home a party of tourists passed us singing and playing music
5. Выход сети: don't say that again
   Оригинал: don't say that again

Также, ниже приведены примеры распознавания своих записей и записей других студентов:

1. Выход сети: i thought that everything i owned would be destroyed
   Оригинал: i thought that everything i owned would be destroyed
2. Выход сети: **e** works fine, it **wark pine**
   Оригинал: it works fine
3. Выход сети: i **wan’t t** say anything, i **want** say anything
   Оригинал: i won’t say anything
4. Выход сети: listen to your heart, listen **_** your **hart**
   Оригинал: listen to your heart
5. Выход сети: i told **yous har yhou'r dremnm** was a **dificult** one, i told you that **the onderng** was a **dificult** one
   Оригинал: i told you that your dream was a difficult one

Как можно заметить, ошибок в записанных студентами аудиозаписях модель делает больше. Это связано с тем, что набор данных, на которых мы обучались имеет в себе преимущественно английский и американский акцент,

тогда как записей русского акцента в наборе нет, данную проблему можно частично решить, о чем будет рассказано ниже.

В конце хотелось бы высказать идеи для дальнейшего развития сети.

- Проведение экспериментов для выявления наилучшего оптимизатора с целью повышения скорости обучения.
- Улучшение предобработки сети, в частности аугментации данных, так как мы просто добавляли некоторые случайные значения в данные, тогда как было бы эффективнее использовать для этого другую небольшую выборку аудиозаписей с посторонними звуками и словами.
- Использование некоторых модификаций для алгоритма получения MFCC, так как они помогают не подгоняться под конкретные голоса из-за ограниченности набора данных.
- Дальнейшее исследование вариации параметров настроек с целью повышения точности.
- Исследовать комбинации разработанной сети с другими алгоритмами.

ЗАКЛЮЧЕНИЕ

В данной работе изучены современные подходы к решению проблемы, переменяемые инструменты, на основе существующих исследований построена оригинальная архитектура, которая вбирает в себя из них свои преимущества, чтобы добиться желаемого результата, также были учтены исследования по новым алгоритмам для оптимизации вычислительного процесса, в следствии чего нейронная сеть стала способна обучаться даже на довольно ограниченных вычислительных ресурсах. По мимо прочего были предложены дальнейшие пути развития идей данной работы для улучшения результата в точности распознавания речи.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] W. Xiong, L. Wu, F. Alleva, J. Droppo, X. Huang, A. Stolcke. The Microsoft 2017 Conversational Speech Recognition System [Электронный ресурс]//arXiv. —2017. — Режим доступа: https://arxiv.org

[2] George Saon, Hong-Kwang J. Kuo, Steven Rennie and Michael Picheny. The IBM 2015 English Conversational Telephone Speech Recognition System//[Электронный ресурс]//arXiv. —2015. — Режим доступа: https://arxiv.org

[3] W. Xiong, J. Droppo, X. Huang, F. Seide, M. Seltzer, A. Stolcke, D. Yu and G. Zweig. Achieving Human Parity in Conversational Speech Recognition [Электронный ресурс]//arXiv. —2017. — Режим доступа: https://arxiv.org

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition [Электронный ресурс]//arXiv. —2015. — Режим доступа: https://arxiv.org

[5] Aaron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, Koray Kavukcuoglu. Conditional Image Generation with PixelCNN Decoders [Электронный ресурс]//arXiv. —2016. — Режим доступа: https://arxiv.org

[6] Jason Brownlee. How to develop 1D convolutional neural network models for human activity recognition [Электронный ресурс]//Machine Learning Mastery. — 2018. — Режим доступа: https://machinelearningmastery.com

[7] Fisher Yu, Vladlen Koltun. Multi-Scale Context Aggregation by Dilated Convolutions [Электронный ресурс]//arXiv. —2015. — Режим доступа: https://arxiv.org

[8] Yoon Kim. Convolutional Neural Networks for Sentence Classification [Электронный ресурс]//arXiv. —2014. — Режим доступа: https://arxiv.org

[9] Alex Graves, Faustino Gomez, Santiago Fernández, Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks [Электронный ресурс]//ResearchGate. — 2006. — Режим доступа: https://www.researchgate.net

[10] Sergey Ioffe, Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [Электронный ресурс]//arXiv. —2015. — Режим доступа: https://arxiv.org
[11] Mark Gales, Steve Young. The Application of Hidden Markov Models in Speech Recognition [Электронный ресурс]//University of Cambridge. —2008. — Режим доступа: https://mi.eng.cam.ac.uk
[12] Awni Y. Hannun, Andrew L. Maas, Daniel Jurafsky, Andrew Y. Ng. First-Pass Large Vocabulary Continuous Speech Recognition using Bi-Directional Recurrent DNNs [Электронный ресурс]//arXiv. —2014. — Режим доступа: https://arxiv.org
[13] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, Andrew Y. Ng. Deep Speech: Scaling up end-to-end speech recognition [Электронный ресурс]//arXiv. —2014. — Режим доступа: https://arxiv.org
[14] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro et al. Deep Speech 2: End-to-End Speech Recognition in English and Mandarin [Электронный ресурс]//arXiv. —2015. — Режим доступа: https://arxiv.org
[15] Yisen Wang, Xuejiao Deng, Songbai Pu, Zhiheng Huang. Residual Convolutional CTC Networks for Automatic Speech Recognition [Электронный ресурс]//arXiv. — 2017. — Режим доступа: https://arxiv.org
[16] Jaeyoung Kim, Mostafa El-Khamy, Jungwon Lee. Residual LSTM: Design of a Deep Recurrent Architecture for Distant Speech Recognition [Электронный ресурс]//arXiv. —2017. — Режим доступа: https://arxiv.org
[17] Vitaliy Liptchinsky, Gabriel Synnaeve, Ronan Collobert. Letter-Based Speech Recognition with Gated ConvNets [Электронный ресурс]//arXiv. —2017. — Режим доступа: https://arxiv.org
[18] Daniel S. Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D. Cubuk, Quoc V. Le. SpecAugment: A Simple Data Augmentation Method for

Automatic Speech Recognition [Электронный ресурс]//arXiv. —2019. — Режим доступа: https://arxiv.org
[19] George Saon, Gakuto Kurata, Tom Sercu, Kartik Audhkhasi, Samuel Thomas, Dimitrios Dimitriadis, Xiaodong Cui, Bhuvana Ramabhadran, Michael Picheny, Lynn-Li Lim, Bergul Roomi, Phil Hall. English Conversational Telephone Speech Recognition by Humans and Machines [Электронный ресурс]//arXiv. —2017. — Режим доступа: https://arxiv.org
[20] Shaojie Bai, J. Zico Kolter, Vladlen Koltun. An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling [Электронный ресурс]//arXiv. —2018. — Режим доступа: https://arxiv.org
[21] William Chan, Navdeep Jaitly, Quoc V. Le, Oriol Vinyals. Listen, Attend and Spell [Электронный ресурс]//arXiv. —2015. — Режим доступа: https://arxiv.org
[22] Tian Tan, Yanmin Qian, Hu Hu, Ying Zhou, Wen Ding, Kai Yu. Adaptive very deep convolutional residual network for noise robust speech recognition//Transactions on Audio, Speech and Language Processing — IEEE, 2018. – С. 1393 - 1405.
[23] Sebastian Raschka Python Machine Learning, 1st Edition. - Packt Publishing, 2015.- 454 с.
[24] Ian Goodfellow Deep Learning (Adaptive Computation and Machine Learning). / Ian Goodfellow, Yoshua Bengio, Aaron Courville. - The MIT Press, 2016. - 775 с.
[25] Christopher M. Bishop Pattern Recognition and Machine Learning (Information Science and Statistics). -Springer, 2011. - 738 с.
[26] Кадурин, А. Глубокое обучение. Погружение в мир нейронных сетей: науч. изд. / А.Кадурин, Е.В.Архангельская, С.И.Николенко. – Санкт-Петербург: Питер, 2018. – 481 с.
[27] Феверолф, М. Машинное обучение: науч. изд. / Феверолф Марк, Ричардс Джозеф, Бринк Хенрик – Санкт-Петербург: Питер, 2017. – 465 с.

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

Институт математики и фундаментальной информатики
Базовая кафедра вычислительных и информационных технологий

УТВЕРЖДАЮ
Заведующий кафедрой
_____/В.В.Шайдуров
«19» 06 2019 г.

БАКАЛАВРСКАЯ РАБОТА

Направление 02.03.01 Математика и компьютерные науки

ПОСТРОЕНИЕ СЛОИСТОЙ НЕЙРОННОЙ СЕТИ ДЛЯ РАСПОЗНАВАНИЯ РЕЧИ

Научный руководитель
кандидат технических наук,
доцент _____/ С.В. Исаев
19.06.2019

Выпускник _____/ А.А. Димов
19.06.2019

Красноярск 2019